ETSI TS 102 472 V1.3.1 (2009-06)

Technical Specification

**Digital Video Broadcasting (DVB);
IP Datacast over DVB-H:
Content Delivery Protocols**

European Broadcasting Union   Union Européenne de Radio-Télévision

EBU·UER

DVB
Digital Video
Broadcasting

ETSI

Reference

RTS/JTC-DVB-251

Keywords

broadcasting, digital, DVB-H, IP, multimedia, TV

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00   Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Important notice

Individual copies of the present document can be downloaded from:
http://www.etsi.org

The present document may be made available in more than one electronic version or in print. In any case of existing or
perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF).
In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive
within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
http://portal.etsi.org/tb/status/status.asp

If you find errors in the present document, please send your comment to one of the following services:
http://portal.etsi.org/chaircor/ETSI_support.asp

Copyright Notification

ETSI

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://webapp.etsi.org/IPR/home.asp).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Introduction

IP Datacast over DVB-H is an end-to-end broadcast system for delivery of any types of digital content and services using IP-based mechanisms optimized for devices with limitations on computational resources and battery. An inherent part of the IPDC system is that it comprises a unidirectional DVB broadcast path that may be combined with a bi-directional mobile/cellular interactivity path. IPDC is thus a platform that can be used for enabling the convergence of services from broadcast/media and telecommunications domains (e.g. mobile/cellular).

Harmonization of the IP Datacast over DVB-H content delivery protocols with 3GPP MBMS [1] has been one of the natural goals of the work.

# 1	Scope

The present document defines a set of Content Delivery Protocols for streaming and file delivery services to be used with IP Datacast over DVB-H [i.1]. Delivery protocols will be IP-based and will be implemented both in content servers and IP Datacast terminals.

The present document includes information applicable to broadcasters, network operators, service providers and manufacturers.

# 2	References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific.

- For a specific reference, subsequent revisions do not apply.

- Non-specific reference may be made only to a complete document or a part thereof and only in the following cases:

  - if it is accepted that it will be possible to use all future changes of the referenced document for the purposes of the referring document;

  - for informative references.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE:	While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

## 2.1	Normative references

The following referenced documents are indispensable for the application of the present document. For dated references, only the edition cited applies. For non-specific references, the latest edition of the referenced document (including any amendments) applies.

[1]	ETSI TS 126 346: "Universal Mobile Telecommunications System (UMTS); Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs (3GPP TS 26.346 version 8.1.0 Release 8)".

[2]	IETF RFC 3926: "FLUTE - File Delivery over Unidirectional Transport".

[3]	IETF RFC 3550: "RTP: A Transport Protocol for Real-Time Applications".

[4]	IETF RFC 4566: "SDP: Session Description Protocol".

[5]	IETF RFC 3890: "A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP)".

[6]	IETF RFC 3556: "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth".

[7]	ETSI EN 301 192: "Digital Video Broadcasting (DVB); DVB specification for data broadcasting".

[8]	IETF RFC 3450: "Asynchronous Layered Coding (ALC) Protocol Instantiation".

[9]	IETF RFC 3451: "Layered Coding Transport (LCT) Building Block".

[10]	IETF RFC 5052: "Forward Error Correction (FEC) Building Block".

[11]	IETF RFC 1952: "GZIP file format specification version 4.3".

[12] IETF RFC 4646: "Tags for the identification of languages".

[13] IETF RFC 3695: "Compact Forward Error Correction (FEC) Schemes".

[14] IETF RFC 2616: "Hypertext Transfer Protocol - HTTP/1.1".

[15] ETSI TS 126 245 (V6.1.0): "Universal Mobile Telecommunications System (UMTS); Transparent end-to-end Packet switched Streaming Service (PS); Timed text format (3GPP TS 26.245 version 6.1.0 Release 6)".

[16] IETF RFC 4396: "RTP Payload Format for 3GPP Timed Text".

[17] The Unicode Consortium: "The Unicode Standard", Version 3.0 Reading, MA, Addison Wesley Developers Press, 2000, ISBN 0-201-61633-5.

[18] IETF RFC 3629: "UTF-8, a transformation of ISO 10646".

[19] ETSI EN 300 743: "Digital Video Broadcasting (DVB); Subtitling systems".

[20] EACEM E-Book (TR-030).

[21] IETF RFC 2250: "RTP Playload Format for MPEG1/MPEG2 Video".

[22] IETF RFC 4574: "The Session Description Protocol (SDP) Label Attribute".

[23] ETSI TS 126 234: "Universal Mobile Telecommunications System (UMTS); LTE; Transparent end-to-end Packet-switched Streaming Service (PSS); Protocols and codecs (3GPP TS 26.234 version 8.1.0 Release 8)".

[24] IETF RFC 4570: "Session Description Protocol (SDP) Source Filters".

[25] IETF RFC 5053: "Raptor Forward Error Correction Scheme for Object Delivery".

2.2 Informative references

The following referenced documents are not essential to the use of the present document but they assist the user with regard to a particular subject area. For non-specific references, the latest version of the referenced document (including any amendments) applies.

[i.1] ETSI EN 302 304: "Digital Video Broadcasting (DVB); Transmission System for Handheld Terminals (DVB-H)".

[i.2] ETSI TS 102 005: "Digital Video Broadcasting (DVB); Specification for the use of Video and audio coding in DVB services delivered directly over IP protocols".

[i.3] ISO/IEC 13818-1: "Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems".

[i.4] IETF RFC 1812: "Requirements for IP Version 4 Routers".

[i.5] IETF RFC 5234: "Augmented BNF for Syntax Specifications: ABNF".

[i.6] ETSI TS 102 471: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Electronic Service Guide (ESG)".

[i.7] ETSI TS 102 832: "Digital Video Broadcasting (DVB); IP Datacast over DVB-H: Notification Framework".

3   Definitions and abbreviations

3.1   Definitions

For the purposes of the present document, the following terms and definitions apply:

associated delivery procedures: set of procedures for file repair and reception reporting which are associated to a file delivery session or a streaming session

base FLUTE channel: first channel signalled in the session description file of a file delivery session

blocking algorithm: algorithm to chop a file into source blocks and encoding symbols for transport over FLUTE

DVB-H: transmission system targeted to provide IP-based services to handheld terminals over terrestrial radio channels, as defined in "Transmission System for Handheld Terminals (DVB-H)"

encoding symbol: array of data bytes that builds up an ALC/LCT packet of a given file

FDT instance: set of files declared in an XML document and identified by a unique Instance ID that represents a subset of the file data table delivered during the file delivery session

file delivery service: set of files delivered by the server to the terminals in a time-constrained or unconstrained manner

file delivery session: instance of delivery of a file delivery service which is characterized by a start and end time and addresses of the transport flows used for the delivery of the files between the start and end time

FLUTE channel: as defined in Flute specification [2] a FLUTE channel is defined by the combination of a sender and destination IP address and port number

> NOTE: A receiver joins a channel to start receiving the data packets sent to the channel by the sender, and a receiver leaves a channel to stop receiving data packets from the channel.

IP datacast: end-to-end broadcast system for delivery of any types of digital content and services using IP-based mechanisms

> NOTE: An inherent part of the IPDC system is that it comprises a unidirectional DVB broadcast path and a bi-directional mobile/cellular interactivity path.

IP flow: flow of IP datagrams each sharing the same IP source and destination address

post-repair mechanism: set of functionalities supplied by the server and used by the terminals after end of file delivery to recover from unsuccessful reception

> NOTE   :These functionalities can be based on point-to-point or point-to-multipoint recovery.

reception reporting mechanism: mechanism that defines a request/response procedure for the server and terminals to request and send reception reports

> NOTE: Reception reports describe the status of the reception.

source block: set of encoding symbols which is used as the basis for FEC encoding/decoding operations

streaming delivery session: instance of delivery of a streaming service which is characterized by a start and end time and addresses of the Transport flows used for delivery of the media streams between start and end time

streaming service: set of synchronized media streams delivered in a time-constrained or unconstrained manner for immediate consumption (during the reception)

time slice: burst of MPE and MPE-FEC sections delivered over DVB-H using a time slicing method

Transport Object (TO): set of source blocks and potentially FEC blocks that build up a given file and which are transported during a file delivery session

transport flow: flow of IP datagrams identified by source IP-address, destination IP-address (either multicast or unicast), port and protocol in use

NOTE: IP flow is composed of one or more transport flows.

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

3GPP	3rd Generation Partnership Project
ABNF	Augmented Backus-Naur Form
ADU	Application Data Unit
ALC	Asynchronous Layered Coding
AS	Application Specific maximum bandwidth
AVP	Audio Video Profile
CCI	Congestion Control Identifier
CENC	Content ENCoding
CRLF	Carriage Return Line Feed
DVB	Digital Video Broadcasting
DVB-H	Digital Video Broadcast – Handheld
ESG	Electronic Service Guide
ESI	Encoding Symbol ID
FDT	File Delivery Table
FEC	Forward Error Correction
FLUTE	File deLivery over Unidirectional Transport
FTI	File Transfer Information
GZIP	GnuZIP
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPDC	IP DataCast
KMM	Key Management Message
KMS	Key Management System
KSM	Key Stream Message
LCT	Layered Coding Transport
MBMS	Multimedia Broadcast/Multicast Service
MIME	Multipurpose Internet Mail Extensions
MPD	MultiProtocol Decapsulation
MPE	MultiProtocol Encapsulation
MPEG-2 TS	MPEG-2 Transport Stream
MTU	Maximum Transmission Unit
NTP	Network Time Protocol
PCR	Program Clock Reference
PES	Packetized Elementary Stream
PSS	Packet-switched Streaming Service
PTS	Presentation TimeStamp
rack	reception acknowledgement
RC	Reception Report Count
RFC	Request For Comments
RR	bandwidth modifier for RTCP Reception Reports
RS	bandwidth modifier for RTCP Sender Reports
RTCP	Real-Time Transport Control Protocol
RTP	Real-Time Transport Protocol
RTSP	Real Time Streaming Protocol
SBN	Source Block Number
SCR	System Clock Reference
SDP	Session Description Protocol
SPP	Service Purchase and Protection
star	statistical reporting for successful reception
TCP	Transmission Control Protocol
TIAS	Transport Independent Application Specific maximum bandwidth

TO	Transport Object
TOI	Transport Object Identifier
TS	Transport Stream
TSI	Transport Session Identifier
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format
XML	eXtensible Markup Language

4   Delivery platform

IP Datacast system is designed to transport different types of content such as audio, video, text, pictures, and binary files. The content delivery services offered in IP Datacast can be classified in two classes: streaming and file delivery services.

In IP Datacast delivery platform, three distinct functional layers can be identified: Bearers, Delivery methods, and User Services.

- **Bearers**: bearers provide the mechanism by which IP data is transported. In IPDC over DVB-H, DVB-H is used to transport multicast and broadcast traffic in an efficient one-to-many manner and are the foundation of IP Datacast services. The DVB-H bearer may be used jointly with point-to-point bearers in offering complete service capabilities.

- **Delivery method**: when delivering content to a receiving application one or more delivery methods are used. Two delivery methods are defined, namely file delivery and streaming. The delivery layer may provide functionality such as security and key distribution, reliability control by means of forward-error-correction techniques and associated delivery procedures such as file-repair and reception reporting.

- **User service**: the IPDC User service enables applications. Different applications impose different requirements when delivering content to receivers and may use different delivery methods. As an example a software package update would use the file delivery while a TV broadcast application would use the streaming delivery.

4.1   Protocol stack

Figure 1 illustrates the protocol architecture for content delivery in IPDC over DVB-H.

Application (s)									
SPP (KSM, KMM)	Audio, Video, Subtitling, etc...	Notification	3GP file format, Binary data, still images, Text, etc...	ESG data	Post repair and reception reporting mechanisms	Notification	SPP (KMM)	Notification
	RTP payload formats							
	Streaming (RTP/RTCP)		carousel / File Delivery (FLUTE)			HTTP		
UDP						TCP		OTA-WSP
IP								

DVB-H	Point-to-Point Bearer
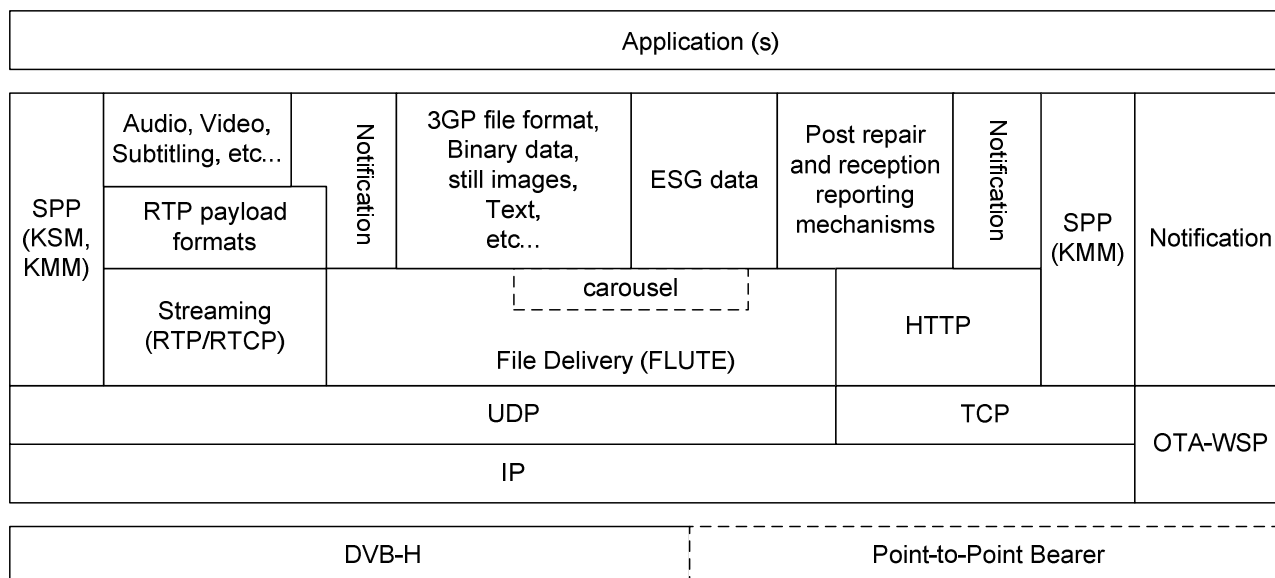
Figure 1: Baseline IPDC Protocol Stack for content delivery

The RTP protocol is used for streaming services, where audio, video and subtitling are delivered in real time. The FLUTE protocol is specified for file delivery services in which all the file data is first downloaded and stored into the terminal before being accessed by applications. Post -repair and reception report data is delivered using FLUTE (point-to-multipoint) or using HTTP and TCP for point-to-point connection. For SPP, KSM (key stream messages) and KMM (key management messages) are delivered over UDP. KMM can be also carried over TCP/IP for point-to-point case.

5   Delivery protocol for real-time streaming services

The RTP protocol is specified for Real-time streaming services in which data are played while downloaded. The supported media formats and their corresponding RTP payload formats are defined in annex B of TS 102 005 [i.2].

5.1   RTP

RTP [3] shall be used to deliver real time audio and video streaming services.

The sender shall generate and send RTCP packets as defined in [3]. The sender shall not provide any Reception Reports in its Sender Report, that is the RC (Reception Report Count) field shall be set to 0. The receivers shall not send any RTCP Receiver Reports.

5.2   Streaming session description with SDP

SDP is provided to the IPDC terminal to describe the streaming delivery session. The SDP describes one or more RTP session parts of the IPDC streaming delivery session. The SDP shall be correctly formed according to [4].

The SDP of an IPDC streaming session shall include the parameters:

- The sender IP address.

- The list of media components in the session.

- The destination IP address and port number for each and all of the media components in the IPDC streaming session.

- The start time and end time of the session.

- The transport protocol (i.e. RTP/AVP).

- Media type(s) and media formats.

- Data rate using existing SDP bandwidth modifiers.

The SDP of an IPDC streaming session may include the parameters:

- Service-language(s) per media. The language attribute is an optional media-level attribute that can be used, e.g. to indicate the spoken language of an audio stream.

5.2.1   SDP Parameters for IPDC streaming sessions

5.2.1.1   Sender IP address

The SDP file associated to a streaming session MAY provide the sender IP address using a source-filter attribute.

The IP source address shall be provided using a source-filter attribute, as defined in [24]. The following rules apply to the source-filter:

1) There shall be exactly one source-filter attribute per complete IPDC file delivery session SDP description, and this shall be in the session part of the session description (i.e. not per media).

2) <filter-mode> shall be set to "incl".

3) <nettype> shall be set to "IN".

4) The destination address <dest-address> shall be given as "*", which indicates that the source filter list applies to all destination addresses.

5.2.1.2   Destination IP address and port number for channels

Each RTP session part of an IPDC streaming session is defined by two parameters:

- IP destination address.

- Destination port number(s).

The IP destination address shall be defined according to the "connection data" field ("c=") of SDP [4]. The destination port number shall be defined according to the <port> sub-field of the media announcement field ("m=") of SDP. Multiple ports using "/" notation shall not be used. The RTCP port shall be RTP port +1.

5.2.1.3   Media description

The media description line shall be used as defined in SDP [4] for RTP [3]. The <media> part indicates the type of media: e.g. audio, video, or text. The usage of RTP and any applicable RTP profile shall be indicated by using the <proto> field of the "*m*-line". The one or more payload types that are being used in this RTP session are enumerated in the <fmt> part. Each payload type is declared using the "a=rtpmap" attribute according to SDP and use the "a=fmtp" line when required to describe the payload format parameters. A label may be assigned to each media description line, in order to uniquely identify and refer to the corresponding media session externally.

5.2.1.4   Session timing parameters

An IPDC streaming session start and end times shall be defined according to the SDP timing field ("t=") [4].

5.2.1.5   Service-language(s) per media

The existing SDP attribute "a=lang" is used to label the language of any language-specific media.

5.2.1.6   Bandwidth specification

The bit-rate required by the streaming session and its media components shall be specified using both the "AS" bandwidth modifier and the "TIAS" bandwidth modifier combined with "a=maxprate" [5] on media level in the SDP. On session level the "TIAS" bandwidth modifier combined with "a=maxprate" may be used. Where the session level expresses the aggregated peak bit-rate, which may be lower than the sum of the individual media streams.

The bandwidth required for RTCP is specified by the "RR" and "RS" bandwidth modifiers [6] on media level for each RTP session. The "RR" modifier shall be included and set to 0 to specify that RTCP receiver reports are not used. The bandwidth used for RTCP sender reports shall be specified using the "RS" bandwidth modifier.

5.2.1.7   Media Session Labelling

Media sessions described by an SDP may be identified using the label attribute [22]. This allows for external references to a specific media session. The label shall at least be unique within the context of the same service.

5.2.2   SDP example for streaming session

Here is a full example of SDP description describing a streaming session:

```
v=0
o=ghost 2890844526 2890842807 IN IP4 192.168.10.10
s=IPDC SDP Example
i=Example of IPDC streaming SDP file
u=http://www.example.com/ae600
e=ghost@mailserver.example.com
c=IN IP6 FF1E:03AD::7F2E:172A:1E24
b=TIAS:77
t=3034423619 3042462419
a=maxprate=30
a=source-filter: incl IN IP6 * 2001:210:1:2:240:96FF:FE25:8EC9
a=min-buffer-time:500
m=video 4002 RTP/AVP 96
b=TIAS:62000
b=RR:0
b=RS:600
a=maxprate:17
a=avg-br:48000
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; packetization-mode=1;sprop-parameter-sets =Z0IACpZTBYmI,aMljiA==
a=label:video_stream_1
m=audio 4004 RTP/AVP 98
b=TIAS:15120
b=RR:0
b=RS:600
a=maxprate:10
a=avg-br:14000
a=rtpmap:98 AMR/8000
a=fmtp:98 octet-align=1
a=label:audio_stream_1
```

5.3   Hypothetical receiver buffering model

5.3.1   Overview of the proposed buffering model (informative)

A hypothetical receiver buffering model is presented in figure 2.
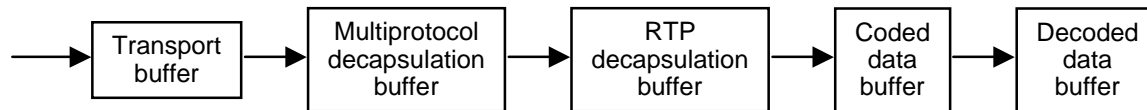
Figure 2: Hypothetical receiver buffering model

The transport buffer receives MPEG-2 TS packets and removes any duplicate packets. Its operation is described in [7] and [i.3]. The multiprotocol decapsulation buffer is used for virtual FEC decoding and decapsulation of MPE sections to IP datagrams. The RTP decapsulation buffer is used for decapsulation of RTP and RTP payload headers and for smoothing the bursty nature of time slices to constant bitrate input for the media decoders. The coded data buffer and the decoded data buffer are specified in the media decoder specifications.

There is one transport buffer per each MPEG-2 TS multiplex, one multiprotocol decapsulation buffer per each elementary stream, and one RTP decapsulation buffer per each Transport flow, one coded data buffer per each elementary media bitstream, and typically one decoded data buffer per each elementary media bitstream.

The multiprotocol decapsulation buffer and the RTP decapsulation buffer are described in the following clauses.

5.3.2   MultiProtocol Decapsulation buffer (normative)

The MultiProtocol Decapsulation (MPD) buffer model is applied to time-sliced elementary streams carrying Transport flows.

 NOTE: The value of the time_slicing element of the time slice and FEC identifier descriptor is equal to 1 for time-sliced elementary streams.

The MPD buffer model is specified as follows:

1) The MPD buffer is initially empty.

2) Data transmission starts from the first MPEG-2 TS packet in transmission order of a time slice.

3) Payload of each MPEG-2 TS packet output from the transport buffer is inserted to the MPD buffer.

4) When:

 a) the value of mpe_fec element in the Time Slice and FEC Identifier descriptor is equal to 00b; and

 b) an MPEG-2 TS packet completes an MPE section; and

 c) the completed MPE section completes a datagram (i.e. the value of last_section_number is equal to the value of section_number in the MPE section header).

then the MPE section is removed from the MPD buffer and the datagram carried in the MPE section is output.

5) When the value of mpe_fec element in the Time Slice and FEC Identifier descriptor is equal to 01b:

 a) when an MPEG-2 TS packet is the first one in a time slice, an MPE-FEC frame is formed in the MPD buffer as specified in clause 9.3.1 of [7];

 b) each MPEG-2 TS packet is inserted to the MPE-FEC frame in the MPD buffer as specified in clause 9.3.1 of [7];

 c) when an MPEG-2 TS packet is the last one containing data for the MPE-FEC frame in the MPD buffer, then the datagrams carried in the MPE sections of the MPE-FEC frame are output and the MPE-FEC frame is removed from the MPD buffer.

5.3.3 RTP Decapsulation buffer (normative)

The RTP decapsulation buffer model is applied to datagrams that are output from the multiprotocol decapsulation buffer and contain RTP packets. The RTP decapsulation buffer model is specific to a Transport flow.

1) The RTP decapsulation buffer is initially empty.

2) Each RTP packet is inserted to the RTP decapsulation buffer without UDP and IP header but including RTP header immediately when it is output from the MPD buffer.

3) RTP packets are not removed from the RTP decapsulation buffer before the signalled initial buffering delay (since the insertion of the first RTP packet) has expired. The signalling means for the initial buffering delay are specified in clause 5.3.4.

  a) Application data units (ADUs) are output from the RTP decapsulation buffer in their decoding order. The decoding order can be established from the RTP sequence numbers, in the absence of packet interleaving. The first ADU in decoding order is output immediately when the initial buffering delay expires. Each succeeding ADU in decoding order is output when it becomes available in the RTP decapsulation buffer and the following time (in seconds) since the removal of the previous ADU has elapsed:

  $8 \times$ (size of the previous ADU in bytes) / R
  where R is the average media bitrate as provided by the "a=avg-br" of the corresponding media line.

4) An RTP packet is removed from the RTP decapsulation buffer, when all the ADUs it contains are output.

5.3.4 Signalling of Hypothetical Receiver Buffer Model Parameters (normative)

The initial buffering delay signals the delay in wall clock time (in units of milliseconds) from the insertion of the RTP packet to the RTP decapsulation buffer until the first ADU in decoding order can be output from the RTP decapsulation buffer. The signalled delay guarantees pauseless decoding and playback. The value is expressed in milliseconds using values between 0 and 65 535 (inclusive).

The initial buffering delay parameter SHALL be signalled to the receiver within the session description. In SDP, the initial buffering delay is provided as session wide attribute "min-buffer-time". The syntax of the "min-buffer-time" is given in ABNF as follows:

Min-buffer-time="a=min-buffer-time:"1*5DIGIT

Additionally, the signalling of the media average bitrate is provided by a media level attribute as follows:

Average-Media-Bitrate="a=avg-br:" 1*10DIGIT

The average media bitrate is provided in bits per second and excludes all protocol headers, so that it is equivalent to the average output bitrate of the media encoder.

5.3.5 Conformance requirements (normative)

Any time-sliced elementary stream carrying Transport flows shall conform to the presented buffering model and the following requirements:

- For any elementary stream, the buffer occupancy level of the multiprotocol decapsulation buffer shall not exceed A bytes.

- A = (maxMPERows × maxMPECols+maxMPESectionHeader) × 1,2 = (1 024 × 255 + maxIPPacketsx × MPESectionHeader) × 1,2 = (1 024 × 255 + 1 024 × 255 / (40+8+12) × 16) × 1,2 = 396 903 bytes.

- For any Transport flow carried in the elementary stream, the output of the RTP decapsulation buffer shall conform to decoding specification of the media format.

- For any Transport flow carried in the elementary stream, the buffer occupancy level of the RTP decapsulation buffer shall not exceed B bytes. In the calculation of B, the assumption of 1 Transport flow per MPE-FEC frame is assumed.

- B = maxMPERows × maxMPECols × (1 – (IPUDPHdr/MaxIPSize)) × 1,2 = 1 024 × 255 × (1 – 12/4096) × 1,2 = 312 426 bytes.

A and B are proportional to the maximum MPE-FEC frame size. A marginal factor of 1,2 to smooth out variations in bitrate and time-slice interval SHOULD be assumed.

5.4   Real-Time Streaming over Unicast

5.4.1   3GPP PSS

The ESG is able to describe services delivered over unicast bearer using PSS as specified in [23].

3GPP PSS describes how terminals can initiate and control a unicast streaming session.

A PSS session can be initiated using one of the following:

- An RTSP URL

- An SDP file (either inlined in the ESG or retrieved using e.g. HTTP)

In order to ensure smooth handover to the unicast delivery session, the PSS server and receiver should support the UTC time in the Range header. If near seamless handover is desired by the receiver, the receiver shall indicate the NTP timestamp of the last correctly received or decoded media unit as the start time in the Range header. The PSS server shall be able to reconstruct the sender time line of the broadcast streaming session, in order to be able to fulfil the receiver request.

6   Delivery protocol for file delivery services

File delivery uses FLUTE [2] to deliver files and other discrete binary objects. This enables a range of file delivery services, from progressive file delivery, to background opportunistic file delivery, to Electronic Service Guide description transport.

6.1   FLUTE

IPDC file delivery method is based on the FLUTE protocol [2]. FLUTE (File deLivery over Unidirectional Transport) [2] shall be used for this function. In addition to basic protocol the proposed file delivery solution comprises parts that further specify how FLUTE is used.

FLUTE is built on top of the Asynchronous Layered Coding (ALC) protocol instantiation [8]. ALC combines the Layered Coding Transport (LCT) building block [9], a congestion control building block and the Forward Error Correction (FEC) building block [10] to provide congestion controlled reliable asynchronous delivery of content to an unlimited number of concurrent receivers from a single sender. As mentioned in [8], congestion control is not appropriate in the type of environment that IPDC system provides, and thus congestion control is not used for IPDC file delivery. See figure 3 for an illustration of FLUTE building block structure. FLUTE is carried over UDP/IP, and is independent of the IP version and the underlying link layers used.
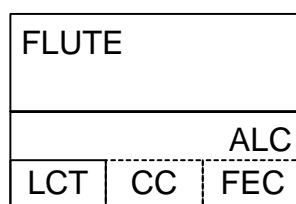
Figure 3: Building block structure of FLUTE

ALC uses the LCT building block to provide in-band session management functionality. The LCT building block has several specified and under-specified fields that are inherited and further specified by ALC. ALC uses the FEC building block to provide reliability. The FEC building block allows the choice of an appropriate FEC code to be used within ALC, including using the no-code FEC code that simply sends the original data using no FEC coding. ALC is under-specified and generally transports binary objects of finite or indeterminate length. FLUTE is a fully-specified protocol to transport files (any kind of discrete binary object), and uses special purpose objects - the File Delivery Table (FDT) Instances - to provide a running index of files and their essential reception parameters in-band of a FLUTE session.

6.1.1   FLUTE as a file delivery mechanism

The purpose of file delivery is to deliver content in files. A file contains any type of data (e.g. Audio/Video file, Binary data, Still images, Text, ESG metadata).

In the present document the term "file" is used for all objects carried by FLUTE (with the exception of the FDT Instances).

IPDC clients and servers shall implement all the mandatory parts of the FLUTE specification [2], as well as ALC [8] and LCT [9] features that FLUTE inherits. In addition, several optional and extended aspects of FLUTE, as described in the following clauses, shall be supported.

6.1.2   Segmentation of files

Segmentation of files shall be provided by a blocking algorithm (which calculates source blocks from source files) and a symbol encoding algorithm (which calculates encoding symbols from source blocks).

6.1.3   Use of multiple FLUTE channels

The use of single FLUTE channel for a FLUTE session shall be supported.

The use of multiple FLUTE channels for a FLUTE session may be supported by terminals and senders. For terminals that do not support multiple channels, it should be possible for them to receive enough data from the first channel named **base** FLUTE channel in order to declare the channel as complete. The base FLUTE channel is the channel for which the connection information appears first in the SDP session description file. This implies that FDT instances carried over the base FLUTE channel shall not reference files carried over other channels. Terminals that do not support multiple channels, shall ignore all but the base FLUTE channel declaration in the SDP session description file.

Each FLUTE channel of a session may send the data packets at a different rate so that it allows reception of faster prior channels.

6.1.4   Symbol encoding algorithm

The "Compact No-Code FEC scheme" [13] (FEC Encoding ID 0, also known as "Null-FEC") SHALL be supported. The "Raptor FEC Scheme" (FEC Encoding Id 1) is defined in clause 8. This scheme consists of two distinct components as defined in clause 8:

- Source block and source packet construction and reception.

- Repair packet construction and reception and Raptor FEC encoding and decoding.

Terminals SHALL support interpretation of source packets constructed according to the source packet construction and reception component of the Raptor FEC Scheme for the case where there is a single sub-block (i.e. N=1).

Terminals MAY support the Repair packet construction and Raptor FEC decoding component of the Raptor FEC Scheme.

In case of Service Discovery (ESG) the sender SHALL provide enough unencoded source packets of the Raptor FEC scheme such that terminals not supporting the repair packet reception and Raptor FEC decoding component are able to reconstruct the ESG data (or alternatively the sender SHALL use Compact No-Code FEC Scheme).

6.1.5   Blocking algorithm

In the case of the Compact no-Code FEC Scheme, the "Algorithm for Computing Source Block Structure" described within the FLUTE specification [2] shall be used.

In the case of the raptor FEC Scheme, the algorithm described in clause 8 shall be used.

6.1.6   Congestion control

For simplicity of congestion control, all FLUTE channels shall be fully provisioned by the datacast operator so that no transport layer congestion control is necessary. FLUTE channelization may be provided by a single FLUTE channel.

6.1.7   Content encoding of files for transport

Files may be content encoded for transport, as described in [2], in the file delivery method using the generic GZIP algorithm [11]. Terminals shall support GZIP content decoding of FLUTE files.

For GZIP-encoded files, the FDT File element attribute "Content-Encoding" SHALL be given the value "GZIP". When content encoding is used, the Transfer-Length field SHALL be present to indicate the actual size of the transport object.

6.1.8   ALC packet size considerations

In order to avoid IP-fragmentation (fragmentation of one IP datagram into several IP datagrams to changing link MTUs across an end-to-end system) it is recommended that all FLUTE packets (including IP/UDP/ALC headers and the payload of the packet itself) are no greater in size than the smallest anticipated MTU of all links end-to-end. A maximum size of such packet is 1 500 bytes as recommended in [i.4]. The overhead of protocol headers should also be considered when determining the maximal size of payload data.

6.1.9   Signalling the end of file delivery and end of file delivery session

FLUTE File Delivery Table (FDT) Instances include an "expires" attribute, which defines the expiration time of the FDT instance. The sender must use an absolute expiry time. According to FLUTE [2] "the receiver SHOULD NOT use a received FDT Instance to interpret packets received beyond the expiration time of the FDT Instance".

The terminal determines the end of file delivery based on the expiration time of the FDT instance, the end time of the session (as declared in the session description), and any end-of-object (B-flag) and end-of-session (A-flag, and SDP end time) information available.

When a particular file (URI) is present in several FDT Instances with different TOI values, then the expiration time of the FDT Instance with the highest FDT Instance ID which includes that file determines the end of file delivery for that file. A terminal shall only determine end of file delivery based only on the most up-to-date instance of the file - and shall not use FDT Instance expiry time to determine end of file delivery for any other (TOI) instances of a file (fileURI).

When a particular file (URI) is present in more than one FDT Instance with the same TOI value, then the end of file delivery is defined by the expiration time of the last FDT Instance to expire.

If an FDT Instance is received describing the file after this time (giving an FDT Instance expiry time in the future and the same or newer version), the terminal shall determine that the delivery of the file has not ended, i.e. that more packets may arrive for that file. Note that this effectively resets and stops any running timers already initiated for an associated delivery procedure for that file.

If the terminal receives an end-of-object packet (with FLUTE header B flag set true) the terminal shall determine that the delivery of that object has ended, and shall assume that file delivery is complete provided that no more recent TOIs are described for the same file (URI) in any received and unexpired FDT Instance(s).

If the terminal determines that the file delivery session has ended then it shall assume that all file deliveries for all files declared in that session have ended.

6.1.10 Files that span over several separate file delivery sessions

Spanning files over several file delivery sessions is not allowed. The use of auxiliary sessions to handle file repair is described in clause 7.

However, a file (or some encoding symbols of a file) may be sent simultaneously or at different time over multiple channels. As defined in clause 6.1.3, sufficient encoding symbols to recover a file, which is declared in an FDT Instance that is sent over the base channel, have to be sent over the base channel. When a file is declared in different FDT instances, which are sent over different channels, the expiry time of these FDT instances does not necessarily need to be the same. Instead, the terminal shall consider the most up-to-date expiry time of the corresponding FDT Instances, in order to decide whether the file is still valid or not.

6.1.11 Grouping mechanisms for FLUTE file delivery

Files downloaded as part of a multiple-file delivery are generally related to one another.

Following examples are explicitly stated for file grouping:

- Web pages are usually linked to each other. A root web page may have links to other web pages, images, or any other files. It is worthwhile to indicate to the receiver that these files constitute a file group. The receiver is then instructed to download all related files, which belong to the same group.

- Software update packages are usually composed of several files. These files usually have to be downloaded as a group because of the existing dependencies. The reception of all files of the software update package is necessary to perform the software update. Logical grouping can be used in this case to indicate the grouping of the different files of the software package. The receiver recognizes through this means that the reception of all files of the group is necessary for the file delivery to be complete.

Logical file grouping allows the server to inform the terminal about existing dependencies between objects of a file delivery session, without the need for the terminal to reconstruct these dependencies at application layer by interpreting the contents of files (or by other means).

FLUTE clients analyses the XML-encoded FDT Instances as they are received, identifies each requested file, associates it with FLUTE packets (using the TOI) and discovers the relevant in-band delivery configuration parameters of each file.

An additional "group" field in the FLUTE FDT instance and file elements enables logical grouping of related files. A FLUTE receiver should download all the files belonging to all groups where one or more of the files of those groups have been requested. (A terminal is permitted to instruct its FLUTE receiver to ignore grouping to deal with special circumstances, such as low storage availability).

The group names are allocated by the FLUTE sender and each specific group name shall group the corresponding files together as one group, including files described in the same and other FDT Instances, for a session.

Each file element of an FDT Instance may be labelled with zero, one or more group names. Each FDT Instance element may be labelled with zero, one or more group names which are inherited by all files described in that FDT Instance. The usage of the Group element in the FDT is shown in clause 6.1.15.

6.1.12 File versioning

In FLUTE, a file is uniquely identified by its "Content-Location" field, which is provided in the FDT Instance that declares that file. Using the FDT, a mapping between the "Content-Location" URI and the TOI is established. A transport object is identified by the Transport Object Identifier.

A file may be associated with several transport objects (i.e. with several TOI values) during the lifetime of the file delivery session. In this case, the transport object declared in the FDT Instance with the highest FDT Instance ID value SHALL represent the latest version of the file. Wrap-around of the FDT Instance ID values SHALL be taken into account in determining the highest FDT Instance ID value. A new FDT Instance may keep the TOI associated with a given file unchanged, which means that this is the version of the file did not change.

The FLUTE sender SHOULD stop sending FLUTE packets of a given file with an older TOI value as soon as a new FDT Instance with a different TOI value for the same file has been sent. The FLUTE sender SHOULD not assign an expiry time to a new FDT Instance that is before the expiry time of older FDT Instances. The FLUTE sender SHALL make sure that any TOI value is at most assigned to one single file unambiguously at any point of time during the lifetime of a file delivery session.

The receiver MAY stop receiving a transport object that represents an old version of a file as soon as an FDT Instance including a newer version of the file is received. The receiver may keep track of the TOI values assigned to a given file to identify the versioning history.

Note that he receiver shall not send post-repair requests for an old version of a file once a FDT Instance including a newer version of the file is received.

6.1.13   File delivery session description with SDP

The FLUTE specification [2] describes required and optional parameters for FLUTE session and media descriptors. This clause specifies SDP for FLUTE session that is used for the IPDC file delivery sessions. The formal specification of the parameters is given in ABNF [i.5].

6.1.13.1   SDP parameters for IPDC file delivery session

Session description of an IPDC file delivery session shall include the parameters:

- the sender IP address;

- the number of channels in the session;

- the destination IP address and port number for each channel in the session per media;

- the Transport Session Identifier (TSI) of the session;

- the start time and end time of the session;

- the protocol ID (i.e. FLUTE/UDP);

- media type(s) (i.e. "application") and fmt-list (i.e. "0");

- FEC capabilities and related parameters.

Session Description of an IPDC file delivery session may include the parameters:

- data rate using existing SDP bandwidth modifiers;

- service-language(s) per media;

- label identifier per media.

This list includes the parameters required by FLUTE [2].

These shall be expressed in SDP [4] syntax according to the following clauses.

6.1.13.1.1   Sender IP address

There shall be exactly one IP sender address per IPDC file delivery session, and thus there shall be exactly one IP source address per complete IPDC file delivery session SDP description.

The IP source address shall be provided using a source-filter attribute, as defined in [24]. The following rules apply to the source-filter:

1)   Exactly one source address may be specified by this attribute such that exactly one source address is given by the src-list field.

2)   There shall be exactly one source-filter attribute per complete IPDC file delivery session SDP description, and this shall be in the session part of the session description (i.e. not per media).

3)   <filter-mode> shall be set to "incl".

4)   <nettype> shall be set to "IN".

5)   The destination address <dest-address> shall be given as "*", which indicates that the source filter applies to all destination addresses.

6.1.13.1.2   Number of channels

FLUTE session channelization shall be defined according to the SDP attribute at session level as specified here.

The multiple channel attribute parameter indicates to the receiver that the sender is using multiple channels in the FLUTE session to transmit data. The attribute also indicates the number of channels used by the sender. The value specified by this descriptor may be used by the receiver to check that it has received all the *m*-lines describing the destinations.

The FLUTE number of channels SDP syntax is given below:

    sdp-flute-channel-line = "a=flute-ch:" integer CRLF integer = as defined in [4].

integer is the number of channels used by the sender to transmit data in a FLUTE session. For example, if the value of this parameter is 2, then there should be 2 channels specified by the *m*-lines.

In the absence of this descriptor, a receiver shall understand that exactly one FLUTE channel is used for the FLUTE session. As described in clause 6.1.3, the use of multiple channels is not normatively mandated but may be supported by the terminals.

6.1.13.1.3   Destination IP address and port number for channels

The FLUTE channel shall be described by the media-level channel descriptor. These channel parameters shall be per channel:

- IP destination address.

- Destination port number.

The IP destination address shall be defined according to the "connection data" field ("c=") of SDP [4]. The destination port number shall be defined according to the <port> sub-field of the media announcement field ("m=") of SDP.

The presence of a FLUTE session on a certain channel shall be indicated by using the "*m*-line" in the SDP description as shown in the following example:

    m=application 12345 FLUTE/UDP 0.

    c=IN IP6 FF1E:03AD::7F2E:172A:1E24/1.

In the above SDP attributes, the *m*-line indicates the media used and the *c*-line indicates the corresponding channel. Thus, in the above example, the *m*-line indicates that the media is transported on a channel that uses FLUTE over UDP. Further, the *c*-line indicates the channel address, which, in this case, is an IPv6 address.

6.1.13.1.4   Transport Session Identifier (TSI) of the session

The combination of the TSI and the IP source address identifies the FLUTE session. Each TSI shall uniquely identify a FLUTE session for a given IP source address during the time that the session is active, and also for a large time before and after the active session time (this is also an LCT requirement [9]).

The TSI shall be defined according to the SDP descriptor given below. There shall be exactly one occurrence of this descriptor in a complete FLUTE SDP session description and it shall appear at session level.

The syntax in ABNF is given below:

    sdp-flute-tsi-line = "a=flute-tsi:" integer CRLF.

    integer = as defined in [4].

6.1.13.1.5   Session timing parameters

A IPDC file delivery session start and end times shall be defined according to the SDP timing field ("t=") [4].

6.1.13.1.6   FEC capabilities and related parameters

A new FEC-declaration attribute is defined which results in, e.g. a=FEC-declaration:0 encoding-id=128; instance-id=0.

This can be session-level (and so the first instance (fec-ref=0) becomes the default for all media) and media-level to specify differences between media. This is optional as the information will be available elsewhere (e.g. FLUTE FDT Instances). If this attribute is not used the terminal may assume that support for FEC id 0 is sufficient capability to enter the session.

A new FEC-declaration attribute shall be defined which results in, e.g. a=FEC:0.

This is only a media-level attribute, used as a short hand to inherit one of one or more session-level FEC-declarations to a specific media.

The syntax for the attributes in ABNF [i.5] is:

- sdp-fec-declaration-line = "a=FEC-declaration:" fec-ref SP fec-enc-id ";" [SP fec-inst-id] CRLF.

- fec-ref = 1*DIGIT (value is the SDP-internal identifier for FEC-declaration).

- fec-enc-id = "encoding-id=" enc-id.

- end-id = 1*DIGIT (value is the FEC Encoding ID used, valid FEC encoding Id are specified in clause 6.1.4).

- fec-nst-id = "instance--id=" inst-id.

- inst-id = 1*DIGIT (value is the FEC Instance ID used, valid FEC encoding Id are specified in clause 6.1.4).

- sdp-fec-line = "a=FEC:" fec-ref CRLF.

- fec-ref = 1*DIGIT (value is the FEC-declaration identifier).

The SDP declares the default FEC encoding scheme (on session or media level). The FEC encoding scheme may however change from file to file and this is overwritten by declarations in the FDT, or in the EXT_FTI ALC/LCT header. It is recommended for non-FDT objects to always include the complete FEC OTI in the FDT or in the EXT_FTI header, and for FDT objects to include the complete FEC OTI in the EXT_FTI header.

6.1.13.1.7   Service-language(s) per media

The existing SDP attribute "a=lang" is used to label the language of any language-specific media. The values are taken from [12] (e.g. "a=lang:EN-US").

6.1.13.1.8   FLUTE Channel Labelling

A label attribute [22] may be assigned to each FLUTE channel of a FLUTE session. This would enable external references to a specific FLUTE channel.

6.1.13.2   Three timers

A single attribute line of SDP description might be used as described in the following example.

    EXAMPLE: a=session-timeout:100; 200; 300.

Where the first value "100" is the value of *fragment wait timer*; the second value "200" is the value of *table wait timer*; and the third value "300" is the value of *object wait timer*.

The syntax described in ABNF:

- Session timeout line =   "a=session-timeout:" ST.

- ST=1*DIGIT ";" 1*DIGIT ";" 1*DIGIT CRLF.

The above attribute shall appear at session level of SDP.

6.1.14   Signalling of parameters with FLUTE

6.1.14.1   Signalling of parameters with Basic ALC/FLUTE headers

FLUTE and ALC mandatory header fields shall be as specified in [2], [8] with the following additional specializations:

- The length of the CCI (Congestion Control Identifier) field shall be 32 bits and it is assigned a value of zero (C=0).

- The Transmission Session Identifier (TSI) field shall be of length 16 bits (S=0, H=1, 16 bits) or 32 bits (S=1, H=0) when TOI is an identifier of 32 bits.

- The Transport Object Identifier (TOI) field should be of length 16 bits (O=0, H=1) or 32 bits (O=1, H=0).

- Only Transport Object Identifier (TOI) 0 (zero) shall be used for FDT Instances.

- The T and R flags shall be set to 0. The SCT and ERT time fields shall not be present.

- The following features shall be used for signalling the end of session; the following features should be used for signalling an end of object transmission to the receiver prior to the FDT expiry date:

  - The Close Session flag (A) for indicating the end of a session as described in clause 6.1.9.

  - The Close Object flag (B) for indicating the end of an object.

In FLUTE the following applies:

- The LCT header length (HDR_LEN) shall be set to the total length of the LCT header in units of 32-bit words.

- For "Compact No-Code FEC scheme", the payload ID shall be set according to [13] such that a 16 bit SBN (Source Block Number) and then the 16 bit ESI (Encoding Symbol ID) are given.

6.1.14.2   Signalling of Parameters with FLUTE Extension Headers

FLUTE extension header fields EXT_FDT, EXT_FTI, EXT_CENC [2] shall be used as follows:

- EXT_FTI shall be included in every FLUTE packet carrying symbols belonging to any FDT Instance.

- FDT Instances shall not be content encoded and therefore EXT_CENC shall not be used.

In FLUTE the following rules apply:

- EXT_FDT is in every FLUTE packet carrying symbols belonging to any FDT Instance.

- FLUTE packets carrying symbols of files (not FDT instances) do not include the EXT_FDT.

The optional use of EXT_FTI for packets carrying symbols of files (not FDT instances) shall comply with FLUTE [2] for the signalling of FEC Object Transmission Information associated to FEC Encoding 0. When Raptor forward error correction code defined in [25] is used, the EXT_FTI format is defined in clause 8.1.3.

6.1.14.3   Signalling of parameters with FDT instances

The FLUTE FDT Instance schema defined in clause 6.1.15 shall be used. Some of the data elements can be included at the FDT-Instance or at the File level. In this case, the data element values in the File element override the same in the FDT Instance element. In addition, the following applies to both the FDT-Instance level information and all files of a FLUTE session.

The inclusion of these FDT Instance data elements is mandatory according to the FLUTE specification:

- Content-Location (URI of a file).

- TOI (Transport Object Identifier of a file instance).

- Expires (expiry data for the FDT Instance).

Additionally, the inclusion of these FDT Instance data elements is mandatory:

- Content-Length (source file length in bytes).

- Content-Type (content MIME type). This attribute shall be either in the FDT-Instance or File element or in both.

The inclusion of the following FDT Instance data elements is optional and depends on the FEC Scheme:

- FEC-OTI-Maximum-Source-Block-Length.

- FEC-OTI-Encoding-Symbol-Length.

- FEC-OTI-Max-Number-of-Encoding-Symbols.

- FEC-OTI-Scheme-Specific-Info.

These optional FDT Instance data elements may or may not be included for FLUTE in IPDC:

- Complete (the signalling that an FDT Instance provides a complete, and subsequently not modifiable, set of file parameters for a FLUTE session may or may not be performed according to this method).

- FEC-OTI-FEC-Encoding-ID (the default value is FEC Encoding ID 0).

- FEC-OTI-FEC-Instance-ID.

- Content Encoding.

- Transfer-Length (this field SHALL be present in case content encoding is applied to the transport object).

- Content-MD5 (Checksum of the file as defined in [2]).

6.1.14.4 Signalling of parameters Out-band

Support of session description as in clause 6.1.13 shall be supported. Use of other data formats and protocols for out-of-band (of a FLUTE session) signalling may be supported but not specified further by the present document.

6.1.15 FDT schema

The following XML schema shall be used for the FDT instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:dvb:ipdc:cdp:flute:fdt:2005"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:dvb:ipdc:cdp:flute:fdt::2005"
    elementFormDefault="qualified">

    <xs:element name="FDT-Instance" type="FDT-InstanceType"/>

    <xs:complexType name="FDT-InstanceType">
        <xs:sequence>
            <xs:element name="File" type="File-Type" maxOccurs="unbounded"/>
            <xs:element name="Group" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
```

```
            <xs:attribute name="Expires" type="xs:string" use="required"/>
            <xs:attribute name="Complete" type="xs:boolean" use="optional"/>
            <xs:attribute name="Content-Type" type="xs:string" use="optional"/>
            <xs:attribute name="Content-Encoding" type="xs:string" use="optional"/>

            <xs:attribute name="FEC-OTI-FEC-Encoding-ID" type="xs:unsignedLong" use="optional"/>
            <xs:attribute name="FEC-OTI-FEC-Instance-ID" type="xs:unsignedLong" use="optional"/>
            <xs:attribute name="FEC-OTI-Maximum-Source-Block-Length" type="xs:unsignedLong" use="optional"/>
            <xs:attribute name="FEC-OTI-Encoding-Symbol-Length" type="xs:unsignedLong" use="optional"/>
            <xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols" type="xs:unsignedLong" use="optional"/>
            <xs:attribute name="FEC-OTI-Scheme-Specific-Info" type="xs:base64Binary" use="optional"/>

            <xs:anyAttribute processContents="skip"/>

    </xs:complexType>

    <xs:complexType name="File-Type">
        <xs:sequence>
            <xs:element name="Group" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
            <xs:any namespace="##other" processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>

        <xs:attribute name="Content-Location" type="xs:anyURI" use="required"/>
        <xs:attribute name="TOI" type="xs:positiveInteger" use="required"/>
        <xs:attribute name="Content-Length" type="xs:unsignedLong" use="required"/>
        <xs:attribute name="Transfer-Length" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="Content-Type" type="xs:string" use="optional"/>
        <xs:attribute name="Content-Encoding" type="xs:string" use="optional"/>
        <xs:attribute name="Content-MD5" type="xs:base64Binary" use="optional"/>

        <xs:attribute name="FEC-OTI-FEC-Encoding-ID" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="FEC-OTI-FEC-Instance-ID" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="FEC-OTI-Maximum-Source-Block-Length" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="FEC-OTI-Encoding-Symbol-Length" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="FEC-OTI-Max-Number-of-Encoding-Symbols" type="xs:unsignedLong" use="optional"/>
        <xs:attribute name="FEC-OTI-Scheme-Specific-Info" type="xs:base64Binary" use="optional"/>

        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

</xs:schema>
```

6.1.15.1   FDT Schema Extensions

The following schema defines the new elements in the FDT.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns="urn:dvb:ipdc:cdp:flute:fdt:2008"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace=" urn:dvb:ipdc:cdp:flute:fdt:2008"
    elementFormDefault="qualified">

    <xs:element name="Cache-Control">
        <xs:complexType>
            <xs:choice>
                <xs:element name="no-cache" type="xs:boolean" fixed="true"/>
                <xs:element name="max-stale" type="xs:boolean" fixed="true"/>
                <xs:element name="Expires" type="xs:unsignedInt"/>
                <xs:element name="no-caching-directive" type="xs:boolean" fixed="true"/>
```

```
                </xs:choice>
                <xs:anyAttribute processContents="skip"/>
            </xs:complexType>
        </xs:element>

</xs:schema>
```

The Cache-Control element may only be present in the File element or in the FDT-Instance element of the FDT. In case it is present in the FDT-Instance element, it shall apply to all files declared in that FDT instance, unless otherwise indicated at the File element. When present at the File element, the indication shall overwrite any caching directive at the FDT instance level.

6.1.16   Caching Directives

The caching functionality defines a mechanism for signalling recommendations on the caching of a file or set of files in a FLUTE session. The receiver should follow any indicated caching directives as much as possible. A requested file should first be located in the cache, before retrieval from the FLUTE session. If no storage space is available at the receiver, files should be discarded from cache based on the priorities of the corresponding caching directives. The caching directives shall apply at the file level and not at the transport object level. In other words, when a new version of a file is available, the latest caching directives for that file should still apply. Furthermore, the terminal shall only keep the latest available version of a given file.

The following caching directives are defined:

- no-cache: this directive is used to indicate to the receiver not to cache a specific file (or set of files).

- max-stale: this directive indicates to the FLUTE receiver that a specific file (or set of files) should be cached for an indefinite period of time, if possible. The file has no associated expiry date.

- Expires: this directive is used by the server to indicate the expected expiry time of a specific file (or set of files). It indicates a date and time value formatted as an NTP timestamp.

- no-caching-directive: this is the default value and indicates that no specific caching directives can be given by the server for a given file or set of files. In case the element "Cache-Control" is not present in the FDT for a corresponding file, the terminal should assume that no caching directives can be given for that file and should handle the caching of that file in its best convenience.

The caching directive shall not change for the whole life time of a transport object. The sender is allowed to change the file caching directives from one version of a file to another. The FDT Instance expiration time should not be used as an "Expires" caching directive. Caching directives for a file are still valid after the expiration of the declaring FDT instance or after the end of the FLUTE session. The terminal is responsible for managing the cache after the end of the FLUTE session.

The syntax of the caching directives in the FDT is described in clause 6.1.15.1.

6.2   Download and carousel mechanisms

6.2.1   Types of file delivery sessions

There are five types of file delivery sessions that are specified on the basis of FLUTE:

- Static file delivery session.

- Fixed content delivery session.

- Dynamic file delivery session.

- Static file delivery carousel.

- Dynamic file delivery carousel.

The type of the file delivery session SHALL be determined from the usage of FLUTE. In the following, a description of each of the file delivery session types is given. The rules of how to use FLUTE to realize the session are also specified.

Clause 6.2.2 describes means to signal and determine session completeness for the different session types.

6.2.1.1   Static file delivery session

6.2.1.1.1   Definition

A Static file delivery session is defined as a file delivery session that carries a predefined set of files. The version of a file may change during the lifetime of the session, however only one version of a file is delivered at any point of time.

6.2.1.1.2   Implementation using FLUTE

A static file delivery session is realized with FLUTE as follows:

- At least one FDT Instance, which contains the fully exhaustive list of mappings between each TOI and the respective file parameters, SHALL be delivered. This FDT Instance sets the attribute "Complete" to "TRUE".

- Some FDT Instances may add parameters which were not present in previously delivered FDT instances (e.g. file size), further the values of some parameters may be changed (e.g. file size).

- An FDT Instance can be repeated several times during the file delivery session.

6.2.1.2   Fixed content delivery session

6.2.1.2.1   Definition

Fixed content delivery session is a special type of static file delivery session where the set of files and their version/content cannot change during a session. Figure 4 gives an example of a fixed content delivery session.

6.2.1.2.2   Implementation using FLUTE

A fixed content delivery session is realized with FLUTE as follows:

- Each FDT Instance delivered SHALL contain the fully exhaustive list of mappings between each TOI and the respective file parameters.

- An FDT Instance can be repeated several times during the file delivery session.

- Each FDT Instance sets the attribute "Complete" to "TRUE".
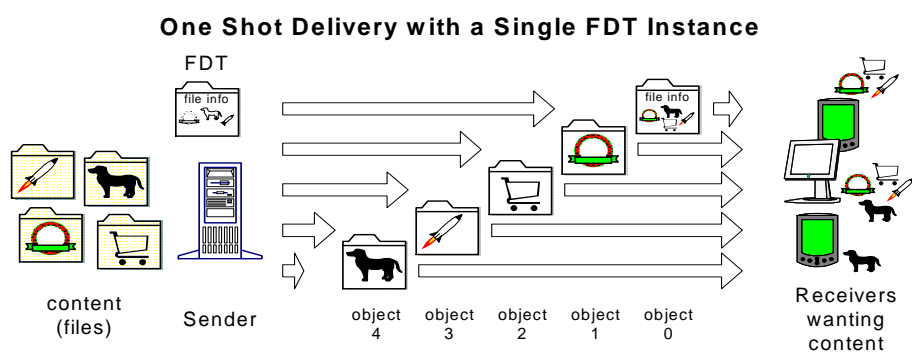
Figure 4: Example of fixed content delivery session

6.2.1.3   Dynamic file delivery session

6.2.1.3.1   Definition

Dynamic file delivery session is defined as a file delivery session, which carries a possibly changing set of files.

6.2.1.3.2   Implementation using FLUTE

In a dynamic file delivery session the basic rules of FLUTE session dynamics apply.

6.2.1.4   Static file delivery carousel

6.2.1.4.1   Definition
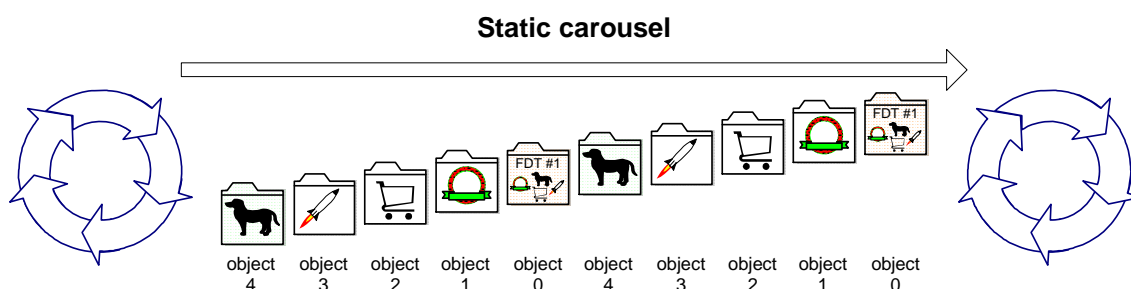
Figure 5: Example of static file delivery carousel

Static file delivery carousel is a possibly time-unbounded file delivery session in which a fixed set of unchanging files are delivered. The concept of a static file delivery carousel is illustrated in figure 5.

6.2.1.4.2   Implementation using FLUTE

A static file delivery carousel is realized as fixed content delivery session in clause 6.2.1.2. The only difference is that in the static file delivery carousel data for the FDT and the objects is sent continuously completely during the session, which is possibly unbounded in time.

In the case that the Compact No-Code FEC Scheme is used then the FDT and each object are repeated one or more times completely during the session.

In the case that the Raptor FEC Scheme is used, then data for a given object may include Raptor-encoded repair symbols in addition to the original source symbols. In particular, file reception time will be minimized if symbols are never repeated until all 65 536 possible symbols (source and repair) have been sent.

Note that packets for each file may be sent together as a block or packets from multiple files may be interleaved.

6.2.1.5   Dynamic file delivery carousel

6.2.1.5.1   Definition
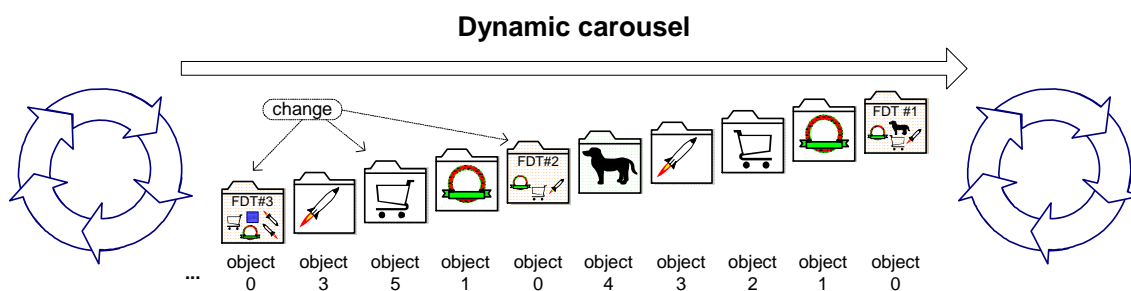
Figure 6: Example of dynamic file delivery carousel

A dynamic file delivery carousel is a possibly time-unbounded file delivery session in which a changing set of possibly changed/added/deleted files is delivered. The concept of dynamic file delivery carousel is illustrated in figure 6.

In a dynamic file delivery carousel the receiver can detect the change in carousel information by observing the FDT instance number changes.

Another example of dynamic file delivery carousel is given below.

Table 1: Example of a file delivery sequence in a dynamic file delivery carousel

Round	FDT instance number	Files being delivered	Notes
1	1	File1, File2, File3	Initial situation
2	2	File1, File2 v2, File3	File2 changed
3	3	File1, File2 v2, File3, File4	File4 added
4	3	File1, File2 v2, File3, File4	Unchanged
5	4	File1 v2, File2 v2, File4	File1 changed,File3 deleted

6.2.1.5.2   Implementation using FLUTE

A dynamic file delivery carousel is realized in the same way as dynamic file delivery session specified in clause 6.2.1.3. The only difference is that in the dynamic file delivery carousel the data for FDT and the objects is sent continuously during the session, which is possibly unbounded in time. Both the FDT Instance and the set of files and their content may change during transmission.

In the case that the Compact No-Code FEC Scheme is used then the FDT and each object are repeated one or more times completely during the session.

In the case that the Raptor FEC Scheme is used, then data for a given object may include Raptor-encoded repair symbols in addition to the original source symbols. In particular, file reception time will be minimized if symbols are never repeated until all 65 536 possible symbols (source and repair) have been sent.

Note that packets for each file may be sent together as a block or packets from multiple files may be interleaved.

6.2.2   Session completeness

It is important to the terminal to know when a given session is assumed to be complete enough for the receiver. A session is complete if the terminal does not expect further data of interest anymore. In that case the terminal SHOULD leave the file delivery session.

Session completeness is well defined in the case of fixed content sessions, where the file list is fixed and the data itself will not change during the session. However, in the cases of static file delivery session, and static file carousel, the files to be delivered may be updated at unknown points of time during the lifetime of the session. Furthermore, in the case of dynamic file delivery session, new files may be added during the lifetime of the session. Also, in the case of file carousels, the end time of the session may be unbounded or may be far in the future. In those cases, it is not possible to define absolute completeness of a session. The notion of complete enough is defined to indicate the point in time where the terminal can assume that no more data of interest will be delivered over the session.

In the following, the session completeness criteria for the different session types are defined.

6.2.2.1   Session completeness for fixed content sessions

The receiver **may** consider the session to be complete when:

- the terminal has received one FDT instance with complete-attribute set; and

    - for every file declared in that FDT instance:

        - the terminal has received all corresponding packets successfully; or

        - the terminal has received at least one packet with the B-flag for that file;

or

- the terminal receives one or more packets with A-flag set.

6.2.2.2   Session completeness for static file delivery sessions and static file delivery carousels

The receiver **may** consider the session to be complete enough when:

- the terminal has received one FDT instance with complete-attribute set; and

    - for every file declared in that FDT instance:

        ▪ the terminal has successfully received all packets of the most up-to-date version (known to the terminal) of that file; or

        ▪ the terminal has received at least one packet with the B-flag for that file;

or

- the terminal receives one or more packets with A-flag set.

6.2.2.3   Session completeness for dynamic file delivery sessions and dynamic file delivery carousels

The receiver MAY use the smart timeout algorithm to determine whether the dynamic session is complete enough.

The smart timeout algorithm is used to determine completeness of a dynamic session. The algorithm is based on using three timers (fragment wait timer, table wait timer and object wait timer) bound to the file delivery session. These parameters enable the creator and sender to determine the semantics of dynamic file delivery session. When receiving timer values, the receivers know when to assume session to be complete enough.
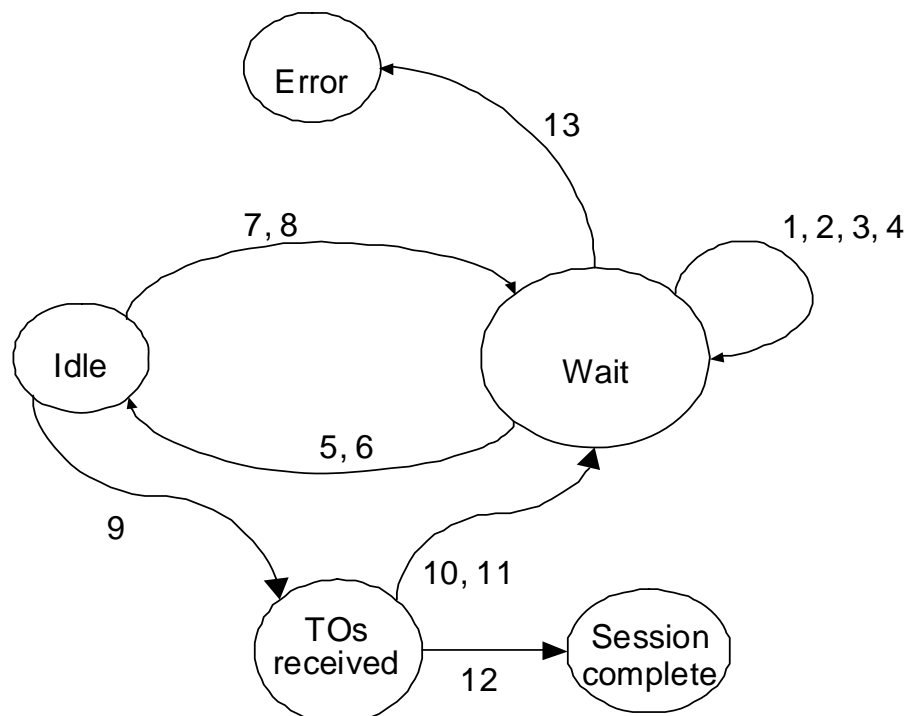

Figure 7: State machine

The state machine of figure 7 is used to specify the operation for determining the completeness. In the "Wait" state, the receiver is waiting for a TOI, or for a declaration of a TOI. In the "Idle" state, the receiver is idle and no timers are active. In the "TOs received" state, all the declared objects have been fully received. The session may be left in the error state, which indicates that an error has happened, or in "session complete" state, which indicates that a session is complete.

There are a number of events, which can trigger transition from the "Wait" state. Transition 1 is triggered when an FDT that contains one or more new TOIs, that is TOIs that have not been previously declared, is received. This triggers the setting and starting of a fragment wait timer t1 for each of the new TOIs. The transition 1 is to the "Wait" state. Transition 2 is triggered in response to the event of receiving a packet for a TOI that has an active fragment wait timer t1. The response is to stop the fragment wait timer t1 for that TOI. This transition 2 can occur only if there still are one or more active fragment wait timers t1 or table wait timers t2. The transition 2 returns to the "Wait" state. Transition 3 is triggered by the event of receiving an FDT that contains a declaration for a TOI that has an active table wait timer t2. This can occur only if there is still one or more active table wait timers t2 or fragment wait timers t1. On the transition 3 the active wait table t2 for that TOI is stopped. The third transition 3 is to the "Wait" state. Transition 4 is triggered by the event of receiving a first packet for a TOI, which is not an FDT table. This triggers the starting of a table wait timer t2 for that TOI. The transition is to the "Wait" state.

Transition 5 is made from the "Wait" state to the "Idle" state in response to the event of receiving a packet for a TOI that has an active fragment wait timer t1. Transition 5 occurs only if there are no other active table or fragment wait timers. The fragment wait timer t1 for that TOI is stopped. Transition 6 is triggered by the event of receiving an FDT that contains a declaration for a TOI that has an active table wait timer t2. Transition 6 occurs only if there are no other active table or fragment wait timers. The table wait timer for that TOI is stopped as a consequence of this transition 6. Transition 6 is from the "Wait" state to the "Idle" state.

When in the "Idle" state, there are three possible transitions. Transition 7 is in response to the event of receiving a FDT that contains one or more new TOIs. This triggers a fragment wait timer t1 to be set and start for each of those new TOIs. The transition is from the "Idle" state to the "Wait" state. Another transition 8 from the "Idle" state to the "Wait" state occurs in response to the receiving a first packet for a TOI which is not an FDT table. This triggers the starting of a table wait timer t2 for that TOI. The transition 9 from the "Idle" state is to the objects "TOs received" state. This transition 9 occurs when the last missing file fragment is received. This triggers the resetting and starting of the object wait timer t3.

When in the "TOs received" state, three transitions are possible. Transition 10 is to the "Wait" state, and occurs when an FDT, which contains one or more new TOIs, is received. This triggers the setting and starting of a fragment wait timer t1 for each of the new TOIs. Optionally, this transition 10 may cause the object wait timer t3 to be stopped. Transition 11 is to the "Wait" state, and occurs when a packet with a TOI that has not been declared in any received FDT instance so far is received. This triggers the setting and starting of a table wait timer t2 for the received new TOIs. Optionally, this transition 11 may cause the object wait timer t3 to be stopped. Transition 12 is from the "TOs received" state to the session complete state. This transition 12 occurs when the object wait timer t3 expires.

A transition from the "Wait" state to the "Error" state occurs when any of the fragment wait timers t1 or the table wait timers t2 for any TOIs expires. This transition is labelled 13 in the diagram. The parameters SHOULD be signalled in the session description as described in clause 6.1.13.2.

6.3 Delivery Protocols over Unicast

File delivery over the interactive channel allows receivers to retrieve files of a file delivery session to be retrieved alternatively over the interactive channel. This functionality addresses several use cases, such as support for roaming users, recovery from partially received FLUTE sessions, and loss of broadcast channel signal.

6.3.1 File Delivery using HTTP

Files of a file delivery session may be made accessible via the interactive channel using HTTP. The retrieval procedure is defined according to the file repair procedure request/response mechanism in clause 7.3. The URL of the HTTP server that delivers the file is indicated in the ESG by the HTTPAccessServerURL.

The HTTP GET request for the retrieval of a file is formulated (as with file repair requests) as follows:

retrieval_request = HTTPAccessServerURL "?" file_URI ["&" content_md5]

file_URI = "fileURI=" URI-reference
content_md5 = "Content-MD5=" 1*(ALPHA / DIGIT / "+" / "/" / "=")

The reply of the HTTP server shall either contain the requested file or a redirection to a different HTTP server. Redirection to a broadcast FLUTE session is not allowed.

6.3.2   File Delivery using FLUTE over Unicast

A FLUTE session over unicast may be initiated and controlled using the RTSP protocol as specified in section "FLUTE session setup and control using RTSP" [1]. This functionality is meant to be used in conjunction with a streaming session over unicast, where the file delivery session is considered as a component of the same service.

After retrieval of the SDP, the FLUTE session is initiated using the RTSP SETUP method and the URL of the FLUTE session (as indicated by the media URL in the "a=control" attributed of the SDP).

After a successful setup of the FLUTE session over unicast, the receiver can initiate data delivery by sending a PLAY request to the server. The Range header indication does not apply to FLUTE sessions.

A PAUSE request will temporarily stop the data delivery of the whole session and a TEARDOWN will close the session.

6.3.3   File Delivery using Notifications

A file delivery session over the interactive channel may be realized based on the Notification Framework [i.6]. For this purpose, a notification application is defined to deliver the FDT Instances of the FLUTE session over the interactive channel. The receiver may then retrieve a file of interest using HTTP and the retrieval URL.

7   Associated delivery procedures

7.1   Introduction

Associated delivery procedures are applicable to content delivery in IPDC over DVB-H. These facilities are especially provided to receivers that have an interactive channel.

- Post-repair of files, initially delivered as part of a FLUTE session, are specified. These repair mechanisms start with a phase where receivers request for the repair of missing elements (part of file(s), entire file(s)). The repair data may be sent either in a point-to-point or in a point-to-multipoint way.

- Reception reporting procedures are specified, as well. These procedures allow a receiver to report the complete reception of one or more files, and also to report statistics about a streaming session.

The terminal sends the repair requests and delivery confirmation reports to ad-hoc servers. To avoid network congestion in the uplink and downlink directions, and also to protect servers against overload situations, the messages from receivers shall be distributed over time and resources (network elements). The parameters of time-window and servers location shall be signalled to the receivers.

7.2   Signalling of associated delivery procedures

When associated delivery procedures are deployed with a given delivery session, a signalling shall be sent to the receivers to describe the existence and the configuration parameters of one or more associated delivery procedures.

This information may be delivered:

- within the ESG prior to the content delivery session along with the session description (out-of-band of that session); or

- in-band within the content delivery session.

The preferred format for an instance of configuration parameters of an associated delivery procedure is an XML file.

The latest version of the configuration file (as described in clause 6.1.12) shall take priority, such that configuration parameters received prior to, and out-of-band of, the content delivery session they apply to are regarded as "global defaults", and configuration parameters received during, and in-band with the content delivery session, overwrite the earlier received parameters. This provides a method to update parameters dynamically on a short time-scale, but as would be desirable where dynamics are minimal, it is not mandatory. In the ESG, the associated delivery procedure description instance is clearly identified using a URI, to enable cross-referencing of in and out-of-band configuration files.

The MIME type <text/xml> should be used for associated delivery procedure instances.

The XML schema for an instance of an associated delivery procedure configuration is defined in clause 7.5.

All configuration parameters of one associated delivery procedure are contained as attributes of an "associatedProcedureDescription" element. The elements (e.g. "postFileRepair" and "postReceptionReport") of an "associatedProcedureDescription" element identify which associated delivery procedure(s) to configure.

7.3 File repair mechanisms

7.3.1 General procedure

The purpose of the File Repair Procedure is to repair lost or corrupted file fragments from a given file delivery. Three problems must generally be avoided:

- Feedbacks implosion due to a large number of receivers requesting simultaneous file repairs. This would congest the uplink network channels.

- Downlink network channel congestion to transport the repair data, as a consequence of the simultaneous clients requests.

- Repair server overload, caused again by the incoming and outgoing traffic due to the clients' requests arriving at the server, and the server responses to serve these repair requests.

The three problems are interrelated and must be addressed at the same time, in order to guarantee a scalable and efficient solution for file repair.

The principle to protect network resources is first to spread the file repair request load in time and across multiple servers, and secondly to give the possibility to send the repaired elements to the receivers either in unicast (point-to-point) or in multicast (point-to-multipoint) depending on defined efficiency thresholds.

The overall procedure is the following:

The receiver:

1) Identifies the missing data from a file delivery.

2) Calculates a random *Back-offTime* and selects a server randomly out of a list.

3) Sends a *repair request* message to the selected server at the calculated time.

Then the server:

1) Responds with a *repair response* message either containing the requested data, or redirecting the receiver to another repair server, or information about the access to a point-to-multipoint file repair session. Error cases messages are specified, as well.

In case the repair response message redirects the receiver to another server:

1) The client reissues the request that causes the redirection to the server designated in the repair response (followed by any remaining repair requests that the client may have to submit).

2) In case the connection has been closed (e.g. due to the HTTP redirection received as a response of a query) before all the submitted requests have been answered, the client reissues any non-answered requests to the server designated in the redirection response.

7.3.2   Triggering associated delivery procedures for file delivery sessions

The identification of the end of file delivery and end of file delivery session is specified in clause 6.1.9.

The terminal SHALL not start the associated delivery procedure back-off timer for older versions of a file.

7.3.3   Identification of repair needs

At the end of a file delivery, the receivers identify their repair needs associated to that file. The session description and FLUTE provide the receiver with sufficient information to determine the source block and encoding symbol structure of each file. From this information, the receiver is able to determine set of symbols sufficient to complete reception of the file. The receiver may request a specific set of symbols from the repair server, in the case that the Raptor FEC Scheme is used, the receiver may request a number of encoding symbols sufficient to recover the file.

In the case that the raptor FEC Scheme is used, the receiver should either:

- identify a minimal set of encoding symbols to be requested that, combined with the already received symbols, allow the Raptor FEC decoder to recover the file; or

- identify a number of new repair symbols sufficient to recover the file.

7.3.4   Distribution of repair requests over time

The receivers shall send their repair requests during a defined time window.

An *offsetTime* is first signalled to the receivers as an associated delivery procedure configuration parameter. This time is the time that a receiver shall wait after the end of a given file delivery to start the file repair procedure.

The *RandomTimePeriod* that is signalled to the receivers as another associated delivery procedure configuration parameter refers to the time window length over which a receiver shall calculate a random time for the initiation of the file repair procedure. The method provides for statistically uniform distribution over a relevant period of time.

The receiver shall calculate a uniformly distributed *RandomTime* out of the interval between 0 and *RandomTimePeriod*.

The sending of the file *repair request* message shall start at *Back-offTime = offsetTime + RandomTime*, and this calculated time shall be a relative time after the file delivery has ended. The receiver shall not start sending the *repair request* message(s) before this calculated time has elapsed after the initial transmission ends.

The back-off time is expressed in seconds.

7.3.4.1   Reset of the back-off timer

The reception of an updated (higher version number) associatedProcedureDescription configuration file and/or an updated sessionDescription shall overwrite the timer parameters used in the back-off algorithm. Except in the case that the offset-time, random-time-period and session end time parameters are identical to the earlier version; the back-off time shall be recalculated. For currently running timers this requires a reset.

7.3.5   Distribution of repair requests over repair servers

The receiver randomly selects one of the service URIs from the list of repair services that is provided by the associated delivery procedure description instance.

The service URIs may also be provided as IP addresses to avoid DNS queries for address resolution. The repair service URIs of a single associated delivery procedure description should be of the same type, e.g. all IP addresses of the same version, or all domain names. The number of URIs is determined by the number of "serviceURI" elements, each of which shall be a child element of the "procedure" element. The "serviceURI" element provides the references to the file repair service via the standard XMLSchema "anyURI" type value. At least one "serviceURI" element shall be present.

Additionally, repair servers may indicate their operation mode in the AssociatedDeliveryProcedureDescription file. The repair server may operate in a P2P only mode, in which case it will always provide the repair symbols in the reply to the repair request, as long as the symbols are available. A repair server operating in hybrid mode may decide to redirect the receiver to a point-to-multipoint repair session, even if the repair symbols are available on the server. The absence of the server mode attribute shall signify that the server operates in Hybrid mode.

> NOTE: Receivers should not consider this attribute when randomly selecting a repair server, unless access to the broadcast channel is not available.

7.3.6   File repair request message

Once missing file data is identified, the receiver sends one or more messages to a repair server requesting transmission of data that allows recovery of missing file data. All point-to-point repair requests for a given file delivery shall take place in a single TCP session using the HTTP 1.1 protocol [14]. If the receiver needs repair data for more than one file received, the receiver may send separate HTTP GET requests for each file or request a group of files described in the FDT. The repair request is routed to the repair server IP address resolved from the selected "serviceURI".

If there is more than one repair request to be made for a given file, these are sent immediately after the first.

The receiver is recommended to request exactly the number of encoding symbols, per source block, that would be the minimum to complete the download/reconstruction of a file and shall not request more than this number. Where source symbols were among the transmission (i.e. Compact No-Code FEC or Raptor FEC), then only source symbols shall be requested for repair.

Alternatively, the complete file may be requested by the receiver, if reconstruction from already received encoding symbols is not possible. This is the case e.g. when no encoding symbols at all have been received by the receiver.

Moreover, the receiver may request all files from a specific FDT instance or a specific logical group of a particular IPDC user services.

7.3.6.1   File repair request message format

After the file delivery, the receiver identifies the missing file data and requests for their transmission. The requested data can be a set of files, a whole file (identified by its URI), or a list of missing file elements. The individual file elements are identified by their FEC Payload ID as used by the ALC/FLUTE. The client makes a file repair request using the HTTP request method GET. The Request URI used with the GET method shall include the service URI and a query string.

The service URI shall point to the repair service and may either be a relative path or and absolute URI. In case the service URI is the relative path of the service, the URL indicated by the Host header field shall be used as the base URI for the request. If no Host header field is present in the message and a relative service URI is used, the message shall be treated as an invalid HTTP request.

In the query string part (query_string) of the request, the file repair request shall either include the URI of the file for which it is requesting the repair data (std_query) or an identifier of a set of files (alt_query).

In the std_query case, the file URI is required to uniquely identify the file (resource). Additionally, the repair request shall contain an indication of the MD5 hash value of the file, if present in the FDT instance declaring the file from which data is being requested. The MD5 hash value is used to identify a specific version of the file. The (SBN, ESI) of encoding symbols sufficient to complete the file reception are also encoded in the std_query part of the Request URI.

> NOTE 1: The MD5 hash value is calculated on the FLUTE transport object after any content encoding (e.g. GZIP compression) has been performed (as indicated by [2] and [14]). On the terminal side, the MD5 hash value applies to the file before decompression.

A set of files may be fetched using the file repair server. A client may request all files from a specific FDT instance or a specific logical group of a particular service by alt_query. For example, when the terminal recognizes that due to some reasons such as, burst errors, the majority of files specified in a certain FDT Instance are missing or the alteration of some of them may compromise the integrity of the whole package, it may request files based on FDT Instance ID or Group ID instead of multiple individual requests.

An HTTP client implementation might limit the length of the URL to a finite value, for example 256 bytes. In the case that the length of the URL-encoded file URI and (SBN, ESI) data exceeds this limit, the receiver shall distribute the URL-encoded data into multiple HTTP GET requests, but using the same TCP connection.

In the following, the details of the general syntax used for the repair requests are given.

The HTTP GET with a normal query shall be used to request the missing data.

The HTTP URL syntax is as follows:

- repair_request_URL = repair_service_URI "?" query_string.

where:

- repair_service_URI = <the URL of the repair service selected from the associated delivery procedure description and which points to a repair service or the relative path of the repair service with respect to the repair server URL given by the Host header field>;

The provided repair service URI shall be specific to the IP platform where it is signalled.

- query_string = std_query / alt_query;

- std_query = file_URI ["&" content_md5] *("&" sbn_info);

- file_URI = "fileURI=" URI-reference; URI-reference is the file URI as indicated by the "Content-Location" field of the corresponding FDT;

- content_md5 = "Content-MD5=" 1*(ALPHA / DIGIT / "+" / "/" / "=");

- sbn_info = "SBN=" sbn_range;

- sbn_range = (sbnA ["-" sbnZ]) / (sbnA [";" esi_info]);

- esi_info = "ESI=" (esi_range *("," esi_range)) /(esiA "+ " number_symbols);

- esi_range = esiA ["-" esiZ];

- sbnA = 1*DIGIT; the SBN, or the first of a range of SBNs;

- sbnZ = 1*DIGIT; the last SBN of a range of SBNs;

- esiA = 1*DIGIT; the ESI, or the first of a range of ESIs;

- esiZ = 1*DIGIT; the last ESI of a range of ESIs;

- number_symbols = 1*DIGIT; the number of additional symbols required;

- alt_query = session_ID "&" (fdt_inst_id / fdt_group_id);

- sessionID = "sessionID=" <unique identifier of a FLUTE session as defined in clause 6.1.13.1.4. This is the format of source_IP_address+":"+FLUTE_TSI. IP Platform context information will be provided by service URI.>;

   NOTE 2: The repair service is able to uniquely identify the FLUTE session based on provided sessionID.

- fdt_inst_id = "fdtInstanceID=" <as defined in clause 3.4.1 of [2] >;

- fdt_group_id = "fdtGroupID=" <as defined in clause 6.1.11 of current document>.

For example, assume that in a FLUTE session a 3gp file with URI = www.example.com/news/latest.3gp was delivered. After the file delivery, a given receiver detects that it did not receive two packets with SBN = 5, ESI = 12 and SBN=20, ESI = 27. The URL of the repair service is "http://www.repairserver.com/ipdc_file_repair_service". Then the HTTP GET request is as follows:

- GET /ipdc_file_repair_service?fileURI=www.example.com/news/latest.3gp&Content-MD5= ODZiYTU1OTFkZGY2NWY5ODh==&SBN=5;ESI=12&SBN=20;ESI=27 HTTP/1.1.

- Host: http://ipplatform_0x1234.repairserver.com.

An example of requesting an entire file is as follows:

- GET ipplatform_0x1234.repairserver.com/ipdc_file_repair_service?fileURI=www.example.com/news/latest.3gp&Content-MD5=ODZiYTU1OTFkZGY2NWY5ODh== HTTP/1.1.

An example of requesting all files of a particular FDT instance is as follows:

- GET ipplatform_0x1234.repairserver.com/ipdc_file_repair_service?sessionID=192.168.1.1:5678&fdtInstanceID=12 HTTP/1.1.

For messaging efficiency, the formal definition enables several contiguous and non-contiguous ranges to be expressed in a single query:

- An entire file.

- A group of files.

- A symbol of a source block (like in the above example).

- A range of symbols for a certain source block (e.g....&SBN=12; ESI=23-28).

- A list of symbols for a certain source block (e.g....&SBN=12; ESI=23, 26, 28).

- All symbols of a source block (e.g....&SBN=12).

- All symbols of a range of source blocks (e.g....&SBN=12-19).

- Non-contiguous ranges (e.g.1....&SBN=12; ESI=34&SBN=20; ESI=23 also, e.g.2....&SBN=12-19&SBN=28; ESI=23-59&SBN=30; ESI=101).

- A number of additional symbols starting from a given ESI (e.g. …&SBN=12; ESI=65+20).

7.3.7   Repair server behaviour

The repair server behaviour depends on the selected repair strategy, and can be as follows:

1) point-to-point repair independently of the number of requests for encoding symbols/source blocks of a given file;

2) point-to-multipoint repair for certain encoding symbols/source blocks or files of a file delivery session;

3) on the basis of the requests that have been received, the server may decide to switch from point-to-point repair strategy to point-to-multipoint repair strategy for a given set of encoding symbols/source blocks of a given file of the file delivery session. The server may use a threshold-dependent algorithm to determine when to switch to point-to-multipoint delivery.

7.3.7.1   File repair response message

Once the repair server has assembled a set of file elements that contain sufficient data to allow the receiver to reconstruct the file data from a particular file repair request, the file repair server sends one message to the receiver. Each file repair response occurs in the same TCP and HTTP session as the repair request that initiated it.

A receiver shall be prepared for any of these 5 response scenarios:

- The server returns a repair response message where a complete file or a set of encoding symbols forms an HTTP payload as specified below.

- The server returns the requested set of files.

- The server redirects the client to a broadcast/multicast delivery (a file delivery session).

- The server redirects the client to another repair server (if a server is functioning correctly but is temporarily overloaded).

- An HTTP error code is returned. (Note that clause 7.3.10 describes the case of no server response).

In case the server responds with multiple files, the server shall encapsulate the requested files into a multipart MIME container. Each part of the multipart MIME shall contain the Content-Location of the embedded file.

For (reasonably) uniformly distributed random data losses, immediate point-to-point HTTP delivery of the repair data will generally be suitable for all clients. However, broadcast/multicast delivery of the requested data may be desirable in some cases:

- A file carousel (all or part of the files from a file delivery session) is already scheduled and the repair server prefers to handle repairs after that file carousel.

- Many terminals request the same data (over a short period of time) indicating that broadcast/multicast delivery of the repaired data would be desirable.

In this case a redirect to the broadcast/multicast repair session for terminals that have made a repair request would be advantageous.

7.3.7.2   File repair response messages codes

In the case that the file repair server receives a correctly formatted repair request which it is able to understand and properly respond to with the appropriate repair data, the file repair server shall attempt to serve that request without an error case.

For a direct point-to-point HTTP response with the requested data, the file repair response message shall report a 200 OK status code and the file repair response message shall consist of HTTP header and file repair response payload (HTTP payload), as defined in clause 7.3.7.3. If the client receives a 200 OK response with fewer than all the quantity of requested symbols it shall assume that the repair server wishes the missing symbols to be requested again (due to its choice or inability to deliver those symbols with this HTTP response).

For a redirect case the HTTP File Repair Server uses the HTTP response status code 302 (Found - Redirection) to indicate to the receiver that the resource (file repair data) is temporarily available via a different URI. The temporary URI is given by the Location field in the HTTP response. In the case of a redirect to another file repair server, this temporary URI shall be the URL of that repair server.

In the case of a redirect to a broadcast/multicast delivery, the temporary URI shall be the URI of the Session Description (SDP file) of the broadcast/multicast (repair) session as described in clause 7.3.8. Other HTTP status codes [14] shall be used to support other cases. These may include server errors, client errors (in the file repair request message), server overload and redirection to other repair servers.

In case the file repair server does not find the requested file (file with given fileURI is not found), the server shall respond with "400 Bad Request" and optionally with "0001 File not found" in the response body. As a result, the IPDC terminal may choose another file repair server as defined in clause 7.3.5.

In case the file repair server does not find the requested version of the requested file (file with given fileURI is found but Content-MD5 is not found), the server shall respond with "400 Bad Request" and optionally with "0002 Content-MD5 not valid" in the response body. As a result, the receiver may choose another file repair server as defined in clause 7.3.5. Or the receiver may request the latest version of the file and discard the previously received chunks of the file.

NOTE 1: The receiver can request the latest version of a file by using only the fileURI argument in the file repair request.

NOTE 2: In case of repetitive server errors, the client is not expected to go through the complete list of available file repair servers, and may abandon after a limited number of attempts.

In case the file repair server does not find any of the requested SBN or ESI values, it shall respond with the "400 Bad Request" and optionally with "0003 SBN or ESI out of range" in the response body. As a result, the receiver should discard all received chunks of the file and request the entire file from the file repair server.

In case the file repair server receives unknown query line arguments, it shall respond with "501 Not Implemented". As a result, the receiver should try to fetch the entire file from the file repair server.

In case the file repair server does not find the requested sessionID value, it shall respond with the "400 Bad Request" and optionally with "0004 SessionID not found" in the response body. As a result, the receiver should request the needed file separately using the fileURI query line argument.

In case the file repair server does not find the requested fdtInstanceID value, it shall respond with the "400 Bad Request" and optionally with "0005 fdtInstanceID not found" in the response body. As a result, the receiver should request the needed file separately using the fileURI query line argument.

In case the file repair server does not find the requested fdtGroupID value, it shall respond with the "400 Bad Request" and optionally with "0006 fdtGroupID not found" in the response body. As a result, the receiver should request the needed file separately using the fileURI query line argument.

HTTP responds error messages may contain a message body, which gives a more detailed error message. The MIME type of such message body shall be in text/plain. The syntax of the HTTP error message body is defined in ABNF [i.5] as follows:

> http-error-body = error-code (SP / HTAB) error-description CRLF;
>
> error-code = 4DIGIT;
>
> error-description = 1*(SP / VCHAR).

Note that the error messages in table 2 MAY be used in the message body of the HTTP response error messages.

Table 2: List of error codes

Error Codes	Semantics
0001	File is not found. The file repair server does not find the requested file.
0002	Content-MD5 is not valid. The file repair server does not find the requested version of the requested file (File with given fileURI is found but Content-MD5 is not found).
0003	SBN or ESI is out of range. The file repair server does not find any of the requested SBN or ESI values.
0004	SessionId is not found. The file repair server does not find the requested sessionID value.
0005	fdtInstanceId is not found. The file repair server does not find the requested fdtInstanceID value.
0006	fdtGroupId is not found. The file repair server does not find the requested fdtGroupID value.

7.3.7.3   Repair server response message format for HTTP carriage of repair data

The file repair response message consists of HTTP header and file repair response payload (HTTP payload).

The HTTP header shall provide:

- HTTP status code, set to 200 OK.

- Content type of the HTTP payload (see below).

The Content-Type shall either be set to the content type of the file, if a complete file is requested, or to "application/simpleSymbolContainer", which denotes that the message body is a simple container of encoding symbols as described below.

This header may look as follows:

> HTTP/1.1   200 OK
> Content-Type: application/simpleSymbolContainer.

NOTE: Other HTTP headers may also be used but are not mandated by this mechanism.

Encoding symbols are included in the response in groups. Each group is preceded by an indication of the number of symbols within the group and an FEC Payload ID coded according to the FEC scheme used for the original file delivery session. The FEC Payload ID identifies all the symbols in the group in the same way that the FEC Payload ID of an FEC source or repair packet identifies all the symbols in the packet. The file repair response payload is constructed by including each FEC Payload ID and Encoding Symbol group one after another (these are already byte aligned). The order of these pairs in the repair response payload may be in order of increasing SBN, and then increasing ESI value; however no particular order is mandated.

A single HTTP repair response message shall contain, at the most, the same number of symbols as requested by the respective HTTP repair request message.

The UE and file repair server already have sufficient information to calculate the length of each encoding symbol and each FEC Payload ID. All encoding symbols are the same length; with the possible exception of the last source encoding symbol in the repair response. All FEC Payload IDs are the same length for one file repair request-response as a single FEC Scheme is used for a single file.

Figure 8 illustrates the complete file repair response message format (box sizes are not indicative of the relative lengths of the labelled entities).

HTTP Header		
Length Indicator	FEC Payload ID	Encoding Symbols
Length Indicator	FEC Payload ID	Encoding Symbols
Length Indicator	FEC Payload ID	Encoding Symbols

NOTE 1: **Length Indicator** (2 bytes): indicates the number of encoding symbols in the group (in network byte order, i.e. high order byte first).

NOTE 2: **FEC Payload ID**: indicates which encoding symbols are included in the group. The format and interpretation of the FEC Payload ID are dependent on the FEC Scheme in use.

NOTE 3: **Encoding Symbols**: contain the encoding symbols. All the symbols shall be the same length.

Figure 8: File Repair Response Message Format

7.3.8   File repair response for broadcast/multicast of repair data

Clause 7.3.9 defines the behaviour of the terminal, in order to receive point-to-multipoint repair data. Annex B provides an algorithm for the selection of the repair mode. The FEC Object Transmission Information and Content-Encoding for files included in the broadcast/multicast session shall be the same as for the original file delivery session.

Prior to the decision to use broadcast/multicast repair, each repair response shall be provided by HTTP according to clause 7.3.7.1.

The HTTP Repair Server uses the HTTP response status code 302 (Found - Redirection) to indicate to the terminal that the resource (file repair data) is temporarily available via a different URI. The temporary URI is given by the Location field in the HTTP response and is the URI of the Session Description (SDP file) of the broadcast/multicast repair session.

Where feasible, it is recommended that the same file delivery session, that delivered the original data, be used for the broadcast/multicast repair. If this conflicts with the session end time limit of the Session Description then a new version of the Session Description shall be sent with an updated (extended) session end time. This shall be sent in-band of that file delivery session.

In some cases this may not be feasible and a different (possibly new) file delivery session may be defined for the repair. The new file delivery session for file repair shall be over the same broadcast bearer as the original file delivery session.

The SDP file for broadcast/multicast repair session may be carried as payload (entity-body) in the HTTP response which is especially useful if the broadcast/multicast repair session is a new (or recently end time modified) FLUTE file delivery session and other means of service announcement prior to this were not feasible.

The delivery method's associatedProcedureDescription may be updated and the new version transmitted in-band with the file delivery session so that currently active client back-off timers are reset, thus minimizing additional client requests until after the broadcast/multicast repair session. The server shall be prepared for additional requests in any case as successful reception of the updated associatedProcedureDescription cannot be assured in all cases.

The existence of a broadcast/multicast file repair session is signalled by the inclusion of the optional bmFileRepair procedure in the updated associatedProcedureDescription. This is signalled by the bmFileRepair element with a single "sessionDescriptionURI" attribute of type "xs:anyURI" which specifies the URI of the broadcast/multicast file repair session's SDP.

In the cases where the same IP addressing and TSI is used for the broadcast/multicast repair session as the original file delivery session, the terminal simply shall not leave the group. Otherwise, the terminal shall join to the broadcast repair session as it would for any delivery session.

A broadcast/multicast file repair session behaves just as a file delivery session, and the determination of end of files and session, and use of further associated delivery procedures uses the same techniques as specified for the file delivery method.

7.3.9   Threshold-dependent repair strategy

At the start of a file delivery session or within an update to the associatedProcedureDescription, the server indicates to the terminals the existence of a point-to-multipoint repair session. The terminal shall join the indicated point-to-multipoint repair session at the start of post-repair mechanism (as defined in clause 7.3.2), if it did not completely recover the file. The terminal may send point-to-point repair requests at a random time instant and to a randomly selected repair server as defined in clause 7.3. If the service operator decides to switch to the point-to-multipoint repair mode, this decision shall be signalled to the terminals that send point-to-point repair requests, by sending a redirect response to the repair requests. The server shall also declare the file by sending an FDT Instance with an updated and valid (in the future) expiry time to the point-to-multipoint repair session. If the service operator decides to use point-to-point repair mode for a given file, it shall not send any data or FDT Instance for that file on the point-to-multipoint session. If the terminal does not receive an FDT Instance declaring a file over the point-to-multipoint session for the total duration of the repair session (which is RandomTimePeriod+offsetTime) after the start of the repair mechanism for the given file (or after the start of the indicated repair session, if this one is in the future), it shall assume that the service operator will not use point-to-multipoint to repair that given file. In that case, the terminal may leave the point-to-multipoint repair session.

7.3.10   Server Not Responding Error Case

In the error case where a terminal determines that its selected file repair server is not responding it shall return to the serviceURI list of repair servers and uniformly randomly select another server from the list, excluding any servers it has determined are not responding. All the repair requests message(s) from that terminal shall then be immediately sent to the newly selected file repair server.

If all of the repair servers from the serviceURI list are determined to be not responding, the terminal may wait for an update of the associated delivery procedure description, in which an up to date list of the repair servers is made available. Otherwise terminal behaviour in this case is unspecified.

A terminal determines that a file repair server is not responding if any of these conditions apply:

1) The terminal is unable to establish a TCP connection to the repair server.

2) The repair server does not respond to any of the HTTP repair requests that have been sent by the terminal (it is possible that second and subsequent repair requests are sent before the first repair request is determined to be not responded to).

3) The repair server returns an unrecognized message (not a recognizable HTTP response).

4) The server returns an HTTP server error status code (in the range 500 to 505).

7.4   Reception reporting procedure

Following successful reception of content whether through the broadcast channel or the point-to-point channel, a reception reporting procedure can be initiated by the receiver to the server.

For file delivery, the reception reporting procedure is used to report the complete reception of one or more files. For streaming delivery, the reception reporting procedure is used to report statistics on the stream reception.

If the server provided parameters requiring reception reporting confirmation then the receiver shall confirm the content reception.

If reception reporting is requested for statistical purposes the server may specify the percentage subset of receivers it would like to perform reception reporting.

Transport errors can prevent a receiver from deterministically discovering whether the reception reporting associated delivery procedure is described for a session, and even if this is successful whether a sample percentage is described. A receiver shall behave according to the information it has even when it is aware that this may be incomplete.

The receiver:

1) Identifies the complete reception of a content item (e.g. a file).

2) Determines the need to report reception (see clause 7.4.3).

3) Selects a time (Request time) at which a reception report request will be sent and selects a server from a list both randomly and uniformly distributed (see clauses 7.4.4 and 7.4.5).

4) Sends a reception report request message to the selected server at the selected time.

Then the server:

5) Responds with a reception report response message either containing the requested data, or alternatively, describing an error case.

7.4.1   Identifying complete file reception from file delivery

A file is determined to be completely downloaded when it is fully received thanks to one or many delivery iterations, and/or FEC decoding and/or a subsequent File Repair Procedure. The purpose of determining file delivery completeness is to determine when it is feasible for a terminal to compile the reception report for that file.

7.4.2   Identifying complete delivery session reception

Delivery sessions (file and streaming) are considered complete when the "stop time" value of the session description (from "t=" in SDP) is reached. If the "stop time" is unbounded ("0") then this parameter is not used for identifying completed sessions.

Delivery sessions are also considered complete when the terminal decides to exit the session - where no further data from that session will be received.

For file delivery sessions, FLUTE provides a A-flag which, when used, indicates to the terminal that the session is complete.

7.4.3   Determining whether a reception report is required

Upon full reception of a content object or when a session is complete, the receiver must determine whether a reception report is required. An associated delivery procedure description indicates the parameters of a reception reporting procedure (which is transported using the same methods as the ones that describe File Repair).

A delivery method may associate zero or one associated delivery procedure descriptions with a delivery session. Where an associated delivery procedure description is associated with a session, and the description includes a postReceptionReport element, the terminal shall initiate a reception reporting procedure. Reception reporting behaviour depends on the parameters given in the description as explained below.

The Reception Reporting Procedure is initiated if:

   a)   A postReceptionReport element is present in the associated delivery procedure description instance.

One of the following will determine the terminal behaviour:

   a)   reportType is set to rack (Reception Acknowledgement). Only successful file reception is reported without reception details.

   b)   reportType is set to star (Statistical Reporting for successful reception). Successful file reception is reported (as with rack) with reception details for statistical analysis in the network.

   c)   reportType is set to star-all (Statistical Reporting for all content reception). The same as star with the addition that failed reception is also reported. star-all is relevant to both streaming and file delivery.

The reportType attribute is optional and behaviour shall default to rack when it is not present.

The samplePercentage attribute can be used to set a percentage sample of receivers which should report reception. This can be useful for statistical data analysis of large populations while increasing scalability due to reduced total uplink signalling. The samplePercentage takes on a value between 0 and 100, including the use of decimals. This attribute is of a string type and it is recommended that no more than 3 digits follow a decimal point (e.g. 67,323 is sufficient precision).

The samplePercentage attribute is optional and behaviour shall default to 100 (%) when it is not present. The samplePercentage attribute may be used with star and star-all, but shall not be used with rack.

When the samplePercentage is not present or its value is 100 each terminal which entered the associated session shall send a reception report. If the samplePercentage were provided for reportType star and star-all and the value is less than 100, the terminal generates a random number which is uniformly distributed in the range of 0 to 100. The terminal sends the reception report when the generated random number is of a lower value than samplePercentage value.

7.4.4   Request time selection

The receiver selects a time at which it is to issue a delivery confirmation request.

Back-off timing is used to spread the load of delivery confirmation requests and responses over time.

Back-off timing is performed according to the procedure described in clause 7.3.4. TheoffsetTime and randomTimePeriod used for delivery confirmation may have different values from those used for file-repair and are signalled separately in the postReceptionReport of an associated delivery procedure description instance.

In general, reception reporting procedures may be less time critical than file repair procedures. Thus, if a postFileRepair timer may expire earlier than a postReceptionReport, network resources may be saved by using the file repair point-to-point connection also for reception reporting.

The default behaviour is that a terminal shall stop its postReceptionReport timers which are active when a postFileRepair timer expires and results in the successful initiation of point-to-point communications between terminal and server.

In some circumstances, the system bottleneck may be in the server handling of reception reporting. In this case the forceTimeIndependence attribute may be used and set to true (false is the default case and would be a redundant use of this optional attribute). When forceTimeIndependence is true the terminal shall not use file repair point-to-point connections to send reception reporting messages. Instead it will allow the timers to expire and initiate point-to-point connections dedicated to reception report messaging.

For star and star-all, session completeness (according to clause 7.4.2) shall determine the back-off timer initialization time.

For rack, the complete file delivery session reception (according to clause 7.4.2) as well as completing any associated file repair delivery procedure or completing a file carousel shall determine the back-off timer initialization time. Racks shall be only sent for completely received files according to clause 7.4.1.

7.4.5 Reception report server selection

Reception report server selection is performed according to the procedure described in clause 7.3.5.

7.4.6 Reception report message

Once the need for reception reporting has been established, the receiver sends one or more Reception Report messages to the server. All Reception Report request and responses for a particular transmission should take place in a single TCP session using the HTTP protocol [14].

The Reception Report request shall include the URI of the file for which delivery is being confirmed. URI is required to uniquely identify the file (resource).

The client shall make a Reception Report request using the HTTP [14] POST request carrying XML formatted metadata for each reported received content (file). An HTTP session shall be used to confirm the successful delivery of a single file. If more than one file were downloaded in a particular download multiple descriptions shall be added in a single POST request.

Each Reception Report is formatted in XML according the following XML schema (see clause 7.5.3). An informative example of a single reception report XML object is also given (see clause 7.5.4).

The MIME type of XML reception report fragments shall be set to application/vnd.dvb.ipdc.reception-report. Multipart MIME (multipart/mixed) may be used to aggregate several small XML files of reception reports to a larger object.

For Reception Acknowledgement (rack) a receptionAcknowledgement element shall provide the relevant data.

For Statistical Reporting (star/star-all) a statisticalReporting element shall provide the relevant data.

For both rack and star/star-all (mandatory):

- For file delivery, one or more fileURI elements shall specify the set of files which are reported.

For only star/star-all (all optional):

- Each fileURI element has an optional receptionSuccess status code attribute which defaults to "true" ("1") when not used. This attribute shall be used for star-all reports. This attribute shall not be used for star reports.

- The sessionID attribute identifies the delivery session. This is of the format source_IP_address + ":" + FLUTE_TSI/RTP_source_port.

- The sessionType attribute defines the basic delivery method session type used = "download", "streaming", or "mixed".

- The serviceId attribute is value and format is taken from the respective serviceID in the ESG Service Fragment.

- The clientId attribute is unique identifier for the receiver.

- The serviceURI attribute value and format is taken from the respective associatedProcedureDescription serviceURI which was selected by the terminal for the current report. This attribute expresses the reception report server to which the reception report is addressed.

7.4.7   Reception report response message

An HTTP response is used as the Reception Report response message.

The HTTP header shall use a status code of 200 OK to signal successful processing of a reception report. Other status codes may be used in error cases as defined in [14].

7.5   XML-schema for associated delivery procedures

7.5.1   Generic associated delivery procedure description

Below is the formal XML syntax of associatedProcedureDescription instances.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:dvb:ipdc:cdp:associatedProcedures:2005"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:dvb:ipdc:cdp:associatedProcedures:2005"
        elementFormDefault="qualified">
    <xs:element name="associatedProcedureDescription" type="associatedProcedureType"/>
    <xs:complexType name="associatedProcedureType">
        <xs:sequence>
            <xs:element name="postFileRepair" type="basicProcedureType" minOccurs="0" maxOccurs="1"/>
            <xs:element name="bmFileRepair" type="bmFileRepairType" minOccurs="0" maxOccurs="1"/>
            <xs:element name="postReceptionReport" type="reportProcedureType" minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="basicProcedureType">
        <xs:sequence>
            <xs:element name="serviceURI" type="ServiceURIType" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="offsetTime" type="xs:unsignedLong" use="optional" default="0"/>
        <xs:attribute name="randomTimePeriod" type="xs:unsignedLong" use="required"/>
    </xs:complexType>

    <xs:complexType name="bmFileRepairType">
        <xs:attribute name="sessionDescriptionURI" type="xs:anyURI" use="required"/>
    </xs:complexType>

    <xs:complexType name="reportProcedureType">
        <xs:complexContent>
            <xs:extension base="basicProcedureType">
                <xs:attribute name="samplePercentage" type="xs:string" use="optional" default="100"/>
                <xs:attribute name="forceTimeIndependence" type="xs:boolean" use="optional"
        default="false"/>
                <xs:attribute name="reportType" type="Report-Type" use="optional" default="rack"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:simpleType name="Report-Type">
        <xs:restriction base="xs:string">
            <xs:enumeration value="rack"/>
            <xs:enumeration value="star"/>
            <xs:enumeration value="star-all"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="ServiceURIType">
        <xs:complexContent>
            <xs:extension base="xs:anyURI">
                <xs:attribute name="mode" type="ServerModeType" use="optional" default="Hybrid"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:simpleType name="ServerModeType">
        <xs:restriction base="xs.string">
            <xs:enumeration value="Hybrid">
            <xs:enumeration value="P2P">
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

7.5.2   Example associatedProcedureDescription instance

Below is an example of an associated ProcedureDescription instance.

```
<?xml version="1.0" encoding="UTF-8"?>
<associatedProcedureDescription xmlns="urn:dvb:ipdc:cdp:associatedProcedures:2005"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:dvb:ipdc:cdp:associatedProcedures:2005
associated-procedure-description.xsd">
<postFileRepair
offsetTime="5"
randomTimePeriod="10">
<serviceURI>http://ipplatform_0x1234.filedelivery.com/repair_script</serviceURI>
<serviceURI>http://ipplatform_0x1234.operator.umts/repair_script</serviceURI>
<serviceURI
mode="P2P">http://ipplatform_0x1234.operator.umts/p2p_repair_script</serviceURI>
</postFileRepair>
<bmFileRepair sessionDescriptionURI="http://www.example.com/ipdc/session1.sdp"/>
<postReceptionReport
offsetTime="5"
randomTimePeriod="10"
reportType="star-all"
samplePercentage="100"
forceTimeIndependence="0">
<serviceURI>http://ipplatform_0x1234.operator.umts/report_script</serviceURI>
</postReceptionReport>
</associatedProcedureDescription>
```

7.5.3   XML Syntax for a reception report request

Below is the formal XML syntax of reception report request instances.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="urn:dvb:ipdc:cdp:receptionReportRequest:2005"
     xmlns:xs="http://www.w3.org/2001/XMLSchema"
     targetNamespace="urn:dvb:ipdc:cdp:receptionReportRequest:2005"
     elementFormDefault="qualified">
  <xs:element name="receptionReport">
    <xs:complexType>
      <xs:choice>
         <xs:element name="receptionAcknowledgement" type="rackType"/>
         <xs:element name="statisticalReport" type="starType"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="rackType">
    <xs:sequence>
      <xs:element name="fileURI" type="xs:anyURI" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="starType">
    <xs:sequence>
      <xs:element name="fileURI" type="FileSuccessType" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="sessionId" type="xs:string" use="optional"/>
    <xs:attribute name="sessionType" type="Session-Type" use="optional"/>
    <xs:attribute name="serviceId" type="xs:string" use="optional"/>
    <xs:attribute name="clientId" type="xs:string" use="optional"/>
    <xs:attribute name="serviceURI" type="xs:anyURI" use="optional"/>
  </xs:complexType>

  <xs:complexType name="FileSuccessType">
    <xs:simpleContent>
      <xs:extension base="xs:anyURI">
         <xs:attribute name="receptionSuccess" type="xs:boolean" use="optional" default="true"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="Session-Type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="download"/>
      <xs:enumeration value="streaming"/>
      <xs:enumeration value="mixed"/>
```

```
            </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

7.5.4   Example XML for the Reception Report Request

```
<?xml version="1.0" encoding="UTF-8"?>
<receptionReport xmlns="urn:dvb:ipdc:cdp:receptionReportRequest:2005"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:dvb:ipdc:cdp:receptionReportRequest:2005
      receptionReportRequest.xsd">
    <statisticalReport sessionId="76298746" sessionType="download" serviceId="78237463726">
      <fileURI receptionSuccess="true">http://www.example.com/ipdc-files/file1.3gp</fileURI>
      <fileURI receptionSuccess="true">http://www.example.com/ipdc-files/file2.3gp</fileURI>
      <fileURI receptionSuccess="true">http://www.example.com/ipdc-files/file4.3gp</fileURI>
    </statisticalReport>
</receptionReport>
```

8   Application layer FEC

8.1   FEC Scheme definition

8.1.1   General

This clause defines an FEC Scheme according to [10], for the Raptor forward error correction code defined in [25] for the file delivery. This scheme is identified by FEC Encoding ID 1. The FEC Payload ID format and FEC Object Transmission Information format are as defined in the following clauses.

Functionally, this FEC Scheme consists of two components:

- Source block and source packet construction and reception.

- Repair packet construction and reception and Raptor FEC encoding and decoding.

The Source Block and Source Packet construction and reception component allows the original source data to be sent unencoded such that it may be interpreted by terminals which do not support the repair packet reception and Raptor FEC decoding component as well as by terminals which do support the repair packet reception and Raptor FEC decoding component.

Support of the Source Block and Source Packet construction component requires support of the FEC Payload ID and FEC Object Transmission Information defined in clauses 8.1.2 and 8.1.3, as well as the source packets constructed according to clauses 5.3.1 and 5.3.2 in [25]. Terminals which support only this component SHALL ignore packets with an Encoding Symbol ID which is greater than or equal to the number of source symbols in the source block.

Support of the Raptor FEC encoding and decoding component requires support of the remainder of [25].

The FEC payload ID and the FEC Object Transmission Information for the Raptor FEC scheme are defined in [25], clause 3.1 and 3.2, respectively.

9   Subtitling

For IPDC over DVB-H system two optional subtitling methods for network and terminals are defined. Either the character encoded format, or the bitmap based format or both may be supported by terminals. For the character encoded format, the 3GPP Timed Text Format and the corresponding RTP payload format SHALL be used as described in clause 9.1. For the bitmap based format, the DVB Bitmap format and the RTP payload format for MPEG-2 streams SHALL be used as described in clause 9.2.

9.1   Subtitling using 3GPP Timed Text Format

In the character encoded format subtitles, the 3GPP Timed Text Format [15] and the RTP payload format for 3GPP Timed Text [16] SHALL be used for formatting the subtitling text.

Additionally, the following restrictions and extensions apply.

9.1.1   Unicode Support

The Unicode 3.0 [17] standard shall be used. Terminals shall correctly decode UTF-8 format as specified in [18]. The support for UTF-16 is not required.

9.1.2   Support for Transparency

Colour specifications support a transparency value. A transparency value of 0 indicates a fully transparent colour, and a value of 255 indicates fully opaque. Support for full transparency (value 0) is required.

9.1.3   Text position and scaling

All text positions are specified as integer values of 16 bits resolution. Since these positions are encoded as 16.16 floating point values, the lower 16 bits of each value shall be set to 0.

The translation coordinates of the text region tx and ty are relative to the upper left corner of the display area. The receiver shall use the parameters "max-w" and "max-h" as an indication of the sender's reference display area. As a default it shall assume following values: "max-w=720" and "max-h=576". Using its own display dimensions, the terminal shall establish a scaling relationship as follows:

- W: is the current display width.

- H: is the current display height.

- Sx = W / max-w.

- Sy = H / max-h.

All position parameters shall be multiplied by the corresponding scaling factor Sx or Sy.

The font size shall be scaled accordingly and rounded to the next smaller size in order to fit within the new scaled text box.

9.1.4   Optional features

Following features are optional:

- Marquee scrolling: terminals not supporting this option shall display the text, or the portion of it that fits into the text box. All related information such as the scroll delay shall be ignored.

- Highlighting and dynamic highlighting (for closed caption and karaoke): the default value is non-highlighted text. Terminals not supporting this option shall ignore it.

- HyperText: hypertext links are optional and should be ignored if the terminal does not support them.

- Blinking text: terminals that do not support blinking shall ignore it.

9.1.5   Delivery of subtitling text

The RTP payload format for 3GPP Timed Text [16] defined by the IETF shall be used. All unit types (1 to 5) shall be supported. The sender is allowed to send new sample descriptors in-band by generating units of type 5. The default (static) sample descriptors can be sent during session announcement using the "tx3g" parameter as described in clause 9.1.6.

The sender shall use an RTP timestamp clockrate of 1 000 Hz.

9.1.6   SDP Parameters for IPDC streaming sessions

The semantics of a media type description shall include the following parameters:

- The media type, which shall be set to video.

- The media subtype "3gpp-tt" and the timestamp clockrate are declared in the "a=rtpmap" line.

- The list of supported versions of the 3GPP Timed Text Format "sver" shall contain the value 60 referring to version 6.0 and is declared in the SDP "a=fmtp" attribute.

- The parameters "tx", "ty", "layer", "tx3g", "width" and "height" are optional and if present shall be declared in the SDP "a=fmtp" attribute.

- The parameters "max-w" and "max-h" are optional and are used for scaling purposes and if present shall be declared in the SDP "a=fmtp" attribute.

- The language attribute "a=lang" is optional and can be used to indicate the human language of the subtitling stream.

Note that several subtitling streams may be present in the same session description. Furthermore, if the subtitling media stream is transported separately and/or independently of other media (such as a video stream) following parameters shall also be present:

- The sender IP address.

- The destination IP address and port number for the subtitling media component in the IPDC streaming session.

- The start time and end time of the session.

Here is a full example of SDP description describing a streaming session with a subtitling media component:

```
v=0
o=ghost 2890844526 2890842807 IN IP4 192.168.10.10
s=IPDC SDP Example
i=Example of IPDC streaming SDP file
u=http://www.example.com/ae600
e=ghost@mailserver.example.com
c=IN IP6 FF1E:03AD::7F2E:172A:1E24
b=TIAS:77
t=3034423619 3042462419
a=maxprate=20
a=source-filter: incl IN IP6 * 2001:210:1:2:240:96FF:FE25:8EC9
a=min-buffer-time:500
m=video 4002 RTP/AVP 97 96 100
b=TIAS:44000
b=RR:0
b=RS:2
a=maxprate:17
 a=avg-br:38000
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; packetization-mode=1; sprop-parameter-
sets=Z0IACpZTBYmI,aMljiA==
m=video 4006 RTP/AVP 102
b=TIAS:10000
b=RR:0
b=RS:1
a=maxprate:3
a=avg-br:5000
a=rtpmap:102 3gpp-tt/1000
a=fmtp:102 tx=20;ty=200;width=200;height=50;tx3g=Z0IACpZTBYmIaMljiA==,
L0EABpKRBYmGbMleiA==;max-w=720;max-h=576
a=lang:en
```

9.2 Bitmap based subtitling

Bitmap based subtitles in IPDC systems SHALL be coded as defined in [19] with the corrigenda and extensions specified in [20].

DVB subtitles are positioned using a "page composition segment" as defined in clause 7.2.2 in [19]. The clause states:

> "NOTE: All addressing of pixels is based on a frame of 720 pixels horizontally by 576 scan lines vertically. These numbers are independent of the aspect ratio of the picture; on a 16:9 display a pixel looks a bit wider than on a 4:3 display. In some cases, for instance a logo, this may lead to unacceptable distortion. Separate data may be provided for presentation on each of the different aspect ratios. The subtitle_descriptor signals whether the associated subtitle data can be presented on any display or on displays of specific aspect ratio only."

As IPDC systems will have a number of different resolutions some clarifications on the use of pixel addressing and scaling of the bitmaps are needed. The clause "Pixel addressing and scaling of bitmap based subtitles" based on [19] will define how pixel in non 720 by 576 systems are addressed and how bitmaps are scaled. Clause "Pixel addressing of non 720 by 576 subtitles" will define extensions to [19] for the use of subtitles which are not authored for 720 by 576 systems. Receivers SHALL support both modes.

9.2.1   Pixel addressing and scaling of bitmap based subtitles

Subtitles authored for 720 by 576 system can be displayed on IPDC systems with different resolutions without the need to do any conversion of the subtitles on the server side. In this case 720 by 576 is defined to be "full screen".

"Full screen" in this respect means that 720 by 576 have to be mapped to the real resolution of the video. In case of a CIF video, 720 by 576 would map to 352x288. Note that if the video is being scaled to a different resolution on the device, the pixel addressing and scaling of the subtitles have to be changed as well.

Table 3: Pixel addressing and scaling parameters for bitmap based subtitling

	Description
X_{d_res}	Actual display resolution in X at which the video will be shown
Y_{d_res}	Actual display resolution in Y at which the video will be shown
X_{v_res}	Resolution of the video in X
Y_{v_res}	Resolution of the video in Y
f_x	Scaling factor in X by which the pixel address and bitmaps have to be scaled
f_y	Scaling factor in Y by which the pixel address and bitmaps have to be scaled
X_D	Actual position in X for subtitles
Y_D	Actual position in Y for subtitles

If X_{d_res} is the number of pixels in X and Y_{d_res} the number of pixels in Y of a certain device and (Xs,Ys) are the coordinates specified in the subtitling stream than the resulting coordinates (Xd,Yd) on the device are:

$$X_D = f_x \times X_S \, , Y_D = f_y \times Y_S$$

with f_x and f_y being the scaling factors:

$$f_x = \frac{X_{v\_res}}{720} \times \frac{X_{d\_res}}{X_{v\_res}} = \frac{X_{d\_res}}{720} \, , f_y = \frac{Y_{v\_res}}{576} \times \frac{Y_{d\_res}}{Y_{v\_res}} = \frac{Y_{d\_res}}{576}$$

As the bitmaps in this case are authored for 720 by 576 they will have to be scaled by the factors f_x and f_y before being rendered.

The following fields of the "page composition segment" (clause 7.2.1 in [19]) are affected:

region_horizontal_address: The value of this field will be scaled by f_x.

region_vertical_address: The value of this field will be scaled by f_y.

The following fields of the "Region composition segment" (clause 7.2.2 in [19]) are affected:

region_width: The value of this field will be scaled by f_x. The sum of the scaled region_horizontal_address field and the scaled region_width SHALL not exceed X_{d_res}.

region_height: The value of this field will be scaled by fy. The sum of the scaled region_vertical_address field and the scaled region_height SHALL not exceed Y_{d_res}.

9.2.2   Pixel addressing of non "720 by 576" subtitles

If bitmap based subtitles are authored for video content which does not have the resolution of 720 by 576 it would not make sense to create the subtitles for this resolution. The page composition and region composition segments from [19] however assume subtitles which are authored for 720 by 576. In order to signal the intended resolution of the subtitles a new segment type is defined in table 4 with the segment_type id of 0x14 (see clause 7.2 in [19]).

Table 4: Syntax of an authored_subtitle_size_segment

Syntax	Size	Type
authored_subtitle_size_segment{		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
authored_width	16	uimsbf
authored_height	16	uimsbf
}		
NOTE 1: authored_width: the width the subtitles are authored for. NOTE 2: authored_height: the height the subtitles are authored for.		

This authored_subtitle_size_segment should immediately follow a page_composition_segment and there SHALL only be one authored_subtitle_size_segment between a page_composition_segment and an end_of_display_set_segment.

The authored width (X_{a_res}) and height (Y_{a_res}) signalled in the authored_subtitle_size_segment is valid for the complete display set and the following scaling factors should be used:

$$f_x = \frac{X_{v\_res}}{X_{a\_res}} \times \frac{X_{d\_res}}{X_{v\_res}} = \frac{X_{d\_res}}{X_{a\_res}}, f_y = \frac{Y_{v\_res}}{Y_{a\_res}} \times \frac{Y_{d\_res}}{Y_{v\_res}} = \frac{Y_{d\_res}}{Y_{a\_res}}$$

In the case that the authored size is the same as the actual display resolution no scaling of either the bitmaps or the pixel address is needed.

9.2.3   Carriage of DVB subtitle streams over RTP

DVB subtitles shall be carried as PES packets in a MPEG transport stream which in turn is carried via RTP. Specification [21] defines in clause 2 how a MPEG transport streams is encapsulated in RTP packets. The payload format used is MP2T and the payload id is 33.

The mapping between the PTS in the MPEG2 Transport stream and the NTP wall clock is given by the RTP timestamp in the RTP packets. The resolution of the timestamps shall be 90 kHz.

Each RTP timestamp represents the PTS of the first byte of payload data. Note that this is in contrast to [21] where it is stated:

"This clock is synchronized to the system stream Program Clock Reference (PCR) or System Clock Reference (SCR) and represents the target transmission time of the first byte of the packet payload. The RTP timestamp will not be passed to the MPEG decoder. This use of the timestamp is somewhat different than normally is the case in RTP, in that it is not considered to be the media display or presentation timestamp. The primary purposes of the RTP timestamp will be to estimate and reduce any network-induced jitter and to synchronize relative time drift between the transmitter and receiver."

9.2.4   Use of SDP to signal DVB subtitles

The following example shows how to use SDP to signal the presents of DVB subtitles in an MPEG2 transport stream carried over RTP.

m= data 4008 RTP/AVP 33
b=TIAS:10000
b=RR:0
b=RS:1
a=maxprate:3
a=avg-br:5000
a=fmtp:33 ts-content=DVB-Subtitles; max-w=720; max-h=576
a=lang:en

with max-w specifying the width the subtitles are authored for and max-h the height.

10   Description of SPP Streams using SDP

General ESG signalling to support different Service Purchase and Protection (SPP) systems is defined in [i.6] and in the present document.

This clause gives descriptions of Service Purchase and Protection streams using SDP.

Process to handle encrypted services in SPP systems and examples of referencing key stream messages in SDP media descriptions are described in annex D.

10.1   Key Stream Message (KSM) Stream

To support efficient KSM carriage, each KSM Stream is carried in its own UDP stream. The mime type ipdc-ksm is defined to signal a KSM Stream. The explicit format of the key stream is given by the IPDCKMSId parameter in the a=fmtp line.

The location of a KSM stream is signalled within the SDP file used to describe the delivery parameters for a given service. The SDP file describing the service typically contains a media announcement entry for the Video and one for the Audio. In addition, to signal KSM streams, one or more additional stream announcements are added.

A key stream is signalled in the following way:

m=data <port> UDP ipdc-ksm.

The following parameters (table 5) are defined for this mime type and are signalled in the "fmtp:" line.

Table 5: Parameters of the mime type ipdc-ksm

Parameter	Mandatory/ Optional	Type	Comments
IPDCStreamId	M	Integer	Stream identifier uniquely defined by the headend.
IPDCKMSId	M	Integer	KMS identifier.
IPDCOperatorId	M	String	Operator identifier.
IPDCAccessRights	O	String	Optional description or URL of the access rights associated with the content.
NOTE 1: IPDCStreamId uniquely identifies the key stream within the scope of the service (the SDP file) and allows later referencing.			
NOTE 2: IPDCKMSId identifies the Key Management System. This identifier is globally unique and is allocated by DVB.			
NOTE 3: IPDCOperatorId identifies the operator controlling this key stream. This identifier is unique within the scope of the IPDCKMSId and is allocated by the Key Management System. It allows differentiating between two operators using the same Key Management System. This parameter can appear multiple times for a single key stream to allow multiple operators to share a key stream.			
NOTE 4: IPDCAccessRights is an optional field that may be used by the operator to point to relevant information concerning this key stream, such as a means of acquiring the relevant access rights.			

Additional parameters can be freely added to support any specific KMS.

10.2 Key Management Message (KMM) stream

The mime type for key management message (KMM) streams (e.g. stream carrying rights objects/entitlements) is data/ipdc-kmm.

A key management message stream is signalled in the following way:

- m=data <port> UDP ipdc-kmm.

The actual format of the key management message stream is given by the IPDCKMSId and, if present, the IPDCDRMId in the "a=fmtp:ipdc-kmm" line. Every a=fmtp line should contain a parameter IPDCStreamId which identifies the particular stream.

Table 6: Parameters of the mime type ipdc-kmm

Parameter	Mandatory/ Optional	Type	Comments
IPDCStreamId	O	Integer	Stream identifier uniquely defined by the headend.
IPDCKMSId	M	Integer	KMS identifier.
IPDCDRMId	O	String	String identifying the used DRM system.
IPDCOperatorId	M	String	IPDCOperatorId of key management stream.

EXAMPLE: m=data 49230 UDP ipdc-kmm

c=IN IP4 224.2.17.12/127

a=fmtp:ipdc-kmm IPDCKMSId=0xABCD; IPDCDRMId=XRMID; IPDCStreamId=42; IPDCOperatorId=SOMEID.

10.3 KSM Stream Binding

The signalling described below allows the terminal to clearly identify which KSM streams are relevant for each media stream. Several media streams may reference the same KSM stream, thereby sharing the same Traffic Encryption Keys, but each media stream may also reference a different KSM stream.

A single media stream may reference several KSM streams, where different KMS provide secure delivery of the same Traffic Encryption Keys.

EXAMPLE: A service comprising a video stream and an audio stream, both encrypted with the same Traffic Encryption Keys, and protected by two different KMSs will make use of 4 streams: one for the video, one for the audio, one for KMS#1 KSM stream and one for KMS#2 KSM stream (figure 9).
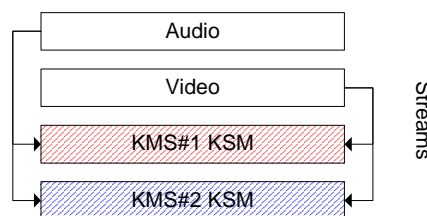
Figure 9: KSM Stream Binding

This way, the KMS will only listen to and process the KSM stream coming on the relevant IP connection. SDP [4] is used to describe the KSM stream(s) associated with each media stream. The following attribute is defined for mapping key streams to media streams in the SDP.

Table 7: Definition of IPDCKSMStream attribute

Attribute	Mandatory/ Optional	Type	Comments
IPDCKSMStream	O	Stream reference	IPDCStreamID indicating which KSM stream applies to this media stream.

The attribute can be at session level, in this case it applies to all media streams or the attribute can be at media level in this case it only applies to the specified media and would overwrite any session level attribute.

Each session or media stream can have a multiple IPDCKSMStream attributes.

Using this attribute the terminal can look up the corresponding KSM stream announcements and figure out which one to listen to and process.

Below is an example where two key streams are associated on session level with the media streams, however two other key streams (13 and 14) are associated to a second audio track. The IPDCKSMStream attribute on media level overwrites theIPDCKSMStream attribute on session level for that particular media stream. That means KSM streams 10 or 11 cannot be used to decrypt the Spanish audio track in this example.

Annex A (informative):
Overview of the blocking algorithm for FEC encoding id 0

This clause gives a brief overview on how files are constructed for and transported during a FLUTE session when using FEC encoding id 0.

The sender takes a file, e.g. a video clip or a still image, which is used as the **transport object** for FLUTE (see figure A.1). Alternatively, the file can be encoded (for example with GZIP) before using it as the transport object. One FLUTE *encoding symbol* is carried as the payload of a each FLUTE packet, thus the FLUTE *packet size* is determined by the *encoding symbol length*. Both the encoding symbols length and the *maximum allowed source block length* are configured by the server. Based on the *transport object length*, the encoding symbol length and the maximum source block length, FLUTE calculates the *source block structure* (i.e. the number of source blocks and their length).
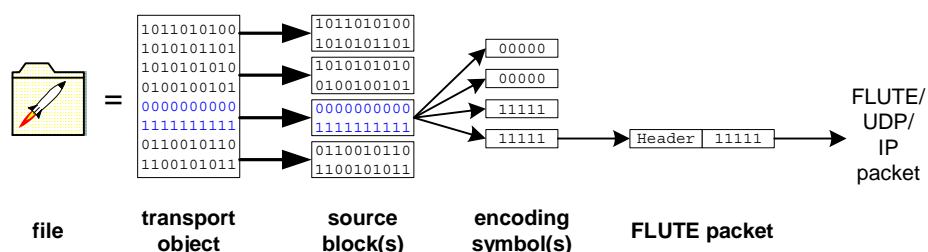
Constructing FLUTE Packets

Figure A.1: Constructing of FLUTE Packets

The server communicates the transport object length, the encoding symbol length and the maximum source block length to the receiver(s) within the FLUTE transmission. Thus the receiver can also calculate the source block structure in advance of receiving a file.

Encoding Symbols are the FLUTE packet payloads. They are taken from the source blocks in fragments according to the encoding symbol length (figure A.1 shows 4 fragments). Then the FLUTE packet is constructed from FLUTE header and encoding symbol payload.

Source blocks are the logical collection of encoding symbols on which FEC encoding and decoding operations are performed.

EXAMPLE: If there is a file of 1 000 000 bytes to transmit via FLUTE. Each FLUTE encoding symbol length = 500 bytes (only packet payload). The maximum allowed source block length = 100 encoding symbols.
This will generate 20 source blocks each long 50 000 bytes (100 symbols). Both the sender and receivers are aware of the fragmentation scheme used by FLUTE.

Annex B (informative):
Algorithm to select repair mechanism for file delivery service

This clause specifies one possible algorithm for the service operator of a file delivery service to select the repair mechanism to be used. The parameters used in this algorithm have to be estimated and adjusted by the service operator, in order to yield optimal performance of the service.

The service operator may base its choice of the repair mode on an efficiency metric. The efficiency of a repair mode can be calculated as follows:

$$E = \frac{\text{number of receivers with successful recovery}}{\text{cost of transmiss ion of repair data}}$$

The service operator estimates both parameters for the point-to-point and the point-to-multipoint repair modes separately. The service operator may then decide to schedule a point-to-multipoint repair session for a specific file, if the point-to-multipoint repair mode is more efficient.

The service operator estimates the cost for the transmission of a single octet over the cellular point-to-point network c_u and over the DVB-H broadcast network c_m.

The service operator also estimates the expected number of repair requests and the amount of data exchanged over the point-to-point and the point-to-multipoint link. The service operator estimates then the expected number of receivers, which will be able to recover the file after the post-repair session.

Estimation of the number of repair requests.

After the start of the repair session (i.e. after the file transmission has ended), terminals have to wait for an offset time and then send their repair requests randomly within the maxBackOff time window. The service operator selects a value α between 0 and 1. It then calculates a time instant t as follows:

$$t = t_{end} + t_{offsetTime} + \alpha \times T_{max\ BackOff}$$

where t_{end} is the time of the end of file delivery as defined in clause 6.1.9, $t_{offseTime}$ is the offset time, and $T_{maxBackOff}$ is the random time period window.

At time t, the service operator queries any of the declared repair servers to get information about the number of repair requests received n_{req}, the number of encoding symbols requested n_{sym}, and the number of unique receivers, which have sent repair requests n_{recv}. Given the fact that repair requests are uniformly randomly distributed over time and over the repair servers, the service operator estimates the expected total number of requests N_{req}, the expected number of requested symbols N_{sym}, and the expected total number of unique receivers sending a repair request N_{recv} for the whole repair session (i.e. over the whole maxBackOff time window) as follows:

$$N_{req} = \frac{n_{req}}{\alpha} \times r$$

$$N_{sym} = \frac{n_{sym}}{\alpha} \times r$$

$$N_{recv} = \frac{n_{recv}}{\alpha} \times r$$

where r is the number of active repair servers for the current file delivery session.

For the point-to-point repair mode the total cost can then be estimated as C_{ptp}:

$$C_{ptp} = c_u \times N_{sym} \times s_{sym} + c_u \times N_{req} \times s_{req}$$

where s_{sym} and s_{req} are the average size of an encoding symbol and the average overhead of a repair request respectively.

In the case of point-to-multipoint repair mode, the server redirects terminals to the point-to-multipoint repair session after the switching decision has been made (after time t). In this case, the repair mode will be point-to-point before time t, and point-to-multipoint after time t. The service operator should assume that terminals will still send their point-to-point repair requests up to the end of the repair time. The service operator should also assume that the point-to-multipoint repair session will contain the whole file (or equivalent data) to achieve complete reception. The cost for the point-to-multipoint repair will then be C_{ptm}:

$$C_{ptm} = c_m \times (S + s_{an}) + c_u \times N_{req} \times s_{req} + c_u \times n_{sym} \times s_{sym}$$

where S is the size of the file (or equivalent data) and s_{an} is the size of the announcement session overhead.

Estimation of number of receivers with successful reception.

The service operator should estimate the number of receivers that were able to completely recover a given file after a repair session. For the point-to-point repair case, the service operator should assume that all terminals that did send repair requests will be able to recover the file. So for the point-to-point repair mode, N_{recv} receivers will be able to recover the file.

$$\text{number of receivers with successful reception}_{ptp} = N_{recv}$$

However, there are some terminals that either do not have a point-to-point connection or are not willing to use it. The server should estimate the fraction of these terminals by **(1-β),** where β is between 0 and 1. When using the point-to-multipoint repair mode, these terminals will have the opportunity to recover the files. However, there will be a fraction of the receivers that are still not able to recover the file after the point-to-multipoint repair session, e.g. because of some packet loss, and this depends on an estimated success rate **(1-p)**. Hence, the total number of receivers recovering the file after point-to-multipoint repair should be estimated as follows:

$$\text{number of receivers with successful reception}_{ptm} = n_{recv} + (1 - p) \times \left(\frac{N_{recv}}{\beta} - n_{recv} \right)$$

Decision on the repair mode.

The service operator should use the cost and number of receivers with successful recovery to calculate the cost per satisfied receiver. The service operator decides then to use the repair mode with the least cost per satisfied receiver, i.e. the repair mode with the highest efficiency as defined in clause 7.3.9.

Implementation Issues.

The communication between the file delivery server, the file repair servers, and other service components is implementation specific. The service operator needs to indicate its repair mode decision and all related parameters (e.g. session description file for the point-to-multipoint repair session) to all repair servers.

The service operator may constantly update its estimation of the parameters **β, p, c_u, c_m, s_{req},** and **s_{an}** to achieve higher accuracy. The selection of the parameter **α,** which determines the time instant of the decision, should be so that it is small enough to allow for fast selection of the optimal mode, and high enough to account for fluctuations (due to network delays, inaccurate random number generators, inaccurate determining of end of file delivery, etc.) that may happen at the start of the repair session. An appropriate value of may be **α** 0.1, given a long enough maxBackOff window.

Annex C (normative):
Void

Annex D (informative):
Process to handle encrypted services in SPP systems

Protected services are signalled by setting the freeToAir attribute in Service fragment to false [i.6]. If only parts of a service are protected, the protected programmes are signalled by setting clearToAir in Schedule fragment to false [i.6]. The actual encryption algorithm used to protect the content/service is specified by each KMS. Independently of the encryption algorithm used traffic keys which are used to descramble the traffic have to be broadcast in parallel to the actual encrypted traffic. The KeyStream element in the Acquisition fragment lists all available key streams for a given media stream [i.6]. Based on the IPDCKMSId the terminal can decided which key stream to receive by checking whether a given KMS is supported by the terminal. There can be multiple key streams for any given media stream. Every key stream is signalled within the SDP of the media stream as a UDP data stream with the mime type of data/ipdc-ksm:

<p style="text-align:center">m=data <PORT> UDP ipdc-ksm</p>

The format of the key stream is further specified by the IPDCKMSId in the "a=fmtp: ipdc-ksm" line.

D.1   SDP examples for key streams

EXAMPLE: m=data 49230 UDP ipdc-ksm
                c=IN IP4 224.2.17.12/127
                a=fmtp: ipdc-ksm IPDCStreamId=10;
                IPDCAccessRights=https://www. IPDCshop.com/channel9.asp;
                IPDCKMSId=1559; IPDCOperatorId=1234

D.2   Examples for referencing key stream messages in SDP media descriptions

An ISMACryp encrypted video stream, could be signalled as:

m=video 41970 RTP/AVP 96
b=TIAS:120000
b=RR:0
b=RS:5
a=maxprate:50
a=avg-br:100000
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==;
ISMACRYP_CRYPTO_SUITE=AES_CTR_128; ISMACRYP_IV_LENGTH=4;
ISMACRYP_DELTA_IV_LENGTH=0; ISMACRYP_KEY_INDICATOR_LENGTH=1; ISMACRYP_SALT=base64,
AoIAE8BAQ8BAQOBSgABQKxkYXRhOmFwc;
a=**IPDCKSMStream:10**
a=IPDCKSMStream:11

An IPSec encrypted stream (e.g. a video stream) could be signalled as:

m=video 41970 RTP/AVP 96
b=TIAS:120000
b=RR:0
b=RS:5
a=maxprate:50
a=avg-br:100000
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==; a=**IPDCKSMStream:10**
a=IPDCKSMStream:11

A SRTP encrypted stream (e.g. a similar video stream as for IPSec example) could be signalled as:

m=video 41970 RTP/SAVP 96
b=TIAS:120000
b=RR:0
b=RS:5
a=maxprate:50
a=avg-br:100000
a=rtpmap:96 H264/9000
a=fmtp:96 profile-level-id=42A01E; sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==;
a=SRTPAuthentication:2
a=SRTPROCTxRate:20
a=IPDCKSMStream:10
a=IPDCKSMStream:11

In all cases, this signalling announces that to gain access to the video stream, the terminal may use either the key stream with IPDCStreamID=10, or the one with IPDCStreamID=11. The terminal can then lookup in the same SDP file both key streams (identified by their IPDCStreamID), identify the KMS and the operator each is associated with and decide, on the basis of this information and depending on which KMS it is supporting, which stream it needs to listen to in order to get the Key Stream Messages it requires.

Annex E (informative):
Example FEC decoder

An example FEC decoder is provided in clause 5.5 of [25].

History

Document history		
V1.1.1	June 2006	Publication
V1.2.1	December 2006	Publication
V1.3.1	June 2009	Publication