IP Datacast over DVB-H: Content Delivery Protocols (CDP)

DVB Document A101

December 2005

Contents

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 of 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).

Harmonisation 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 [2]. 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

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- 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.
- For a non-specific reference, the latest version applies.

[1] 3GPP TS 26.346 v6.1.0: “Multimedia Broadcast/Multicast Service; Protocols and Codecs (Release 6)”.

[2] ETSI EN 302 304: “Transmission System for Handheld Terminals”.

[3] IETF RFC 3926: “FLUTE – File Delivery over Unidirectional Transport”, T. Paila et al., October 2004.

[4] ETSI TS 102 005: "Digital Video Broadcasting (DVB); Implementation Guidelines for the use of Audio-Visual Content in DVB services delivered over IP".

[5] IETF RFC 3550: “RTP: A Transport Protocol for Real-Time Applications”, Schulzrinne H. et al., July 2003.

[6] IETF RFC 2327:"SDP: Session Description Protocol", M. Handley, V. Jacobson April 1998.

[7] IETF RFC 3266: "Support for IPv6 in Session Description Protocol (SDP)", S. Olson, G. Camarillo, A. B. Roach, June 2002.

[8] IETF I-D: “Session Description Protocol (SDP) Source Filters”, B. Quinn, R. Finlayson, June 2005.

[9] IETF RFC 3890: “A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP) ”, Westerlund M., September 2004.

[10] IETF RFC 3556, “Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth”, S. Casner, July 2003.

[11] ETSI EN 301 192: “Digital Video Broadcasting; DVB specification for data broadcasting”, v1.4.1.

[12] ISO/IEC 13818-1: “Information technology – Generic coding of moving pictures and associated audio information – Part1: Systems.

[13] IETF RFC 3450: “Asynchronous Layered Coding (ALC) Protocol Instantiation”, M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, J. Crowcroft, December 2002.

[14] IETF RFC 3451: “Layered Coding Transport (LCT) Building Block”, M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, M. Handley, J. Crowcroft, December 2002.

[15] IETF RFC 3452: “Forward Error Correction (FEC) Building Block”, M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, J. Crowcroft December 2002.

[16] IETF RFC 1952: “GZIP file format specification version 4.3” P. Deutsch, May 1996.

[17] IETF RFC 1812: “Requirements for IP Version 4 Routers”, F. Baker, June 1995.

[18] IETF RFC 2234: "Augmented BNF for Syntax Specifications: ABNF", D. Crocker, P. Overell, November 1997.

[19] IETF RFC 3066: “Tags for the identification of languages”, H. Alvestrand, Jan 2001.

[20] IETF RFC 3695: “Compact Forward Error Correction (FEC) Schemes”, M. Luby, L. Vicisano, February 2004.

[21] IETF RFC 2616: “Hypertext Transfer Protocol -- HTTP/1.1”.

[22] 3GPP TS 26.245 V6.1.0: “3rd Generation Partnership; Technical Specification Group Services and System Aspects Transparent End-to-End Packet switched Streaming Service (PSS); Timed text format (Release 6)”, December 2004.

[23] IETF I-D: “RTP Payload Format for 3GPP Timed Text”, J. Rey and Y. Matsui, June 2005.

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

[25] RFC 3629: "UTF-8, a transformation of ISO 10646", F. Yergeau, November 2003.

[26] ETSI EN 300 743: ‘Digital Video Broadcasting (DVB); Subtitling systems’.

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

[28] IETF RFC 2250: “RTP Format for MPEG1/MPEG2 Video”.

[29] ETSI TS YYY YYY: “IP Datacast over DVB-H: Electronic Service Guide (ESG)”.

3 Definitions and abbreviations

3.1 Definitions

Associated Delivery Procedures: A set of procedures for file repair and reception reporting which are associated to a file delivery session or a streaming session.

Base FLUTE Channel: The first channel signalled in the session description file of a file delivery session.
Blocking Algorithm: An algorithm to chop a file into source blocks and encoding symbols for transport over FLUTE.

DVB-H: A 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: An array of data bytes that builds up an ALC/LCT packet of a given file.

FDT Instance: A 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: A set of files delivered by the server to the terminals in a time-constrained or unconstrained manner.

File Delivery Session: An instance of delivery of a file delivery service which is characterized by a start and end time and addresses of the IP flows used for the delivery of the files between the start and end time.

FLUTE channel: As defined in Flute specification [3] a FLUTE channel is defined by the combination of a sender and destination IP address and port number. 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: An end-to-end broadcast system for delivery of any types of digital content and services using IP-based mechanisms. An inherent part of the IPDC system is that it comprises of a unidirectional DVB broadcast path and a bi-directional mobile/cellular interactivity path.

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

Post-repair Mechanism: A set of functionalities supplied by the server and used by the terminals after end of file delivery to recover from unsuccessful reception. These functionalities can be based on point-to-point or point-to-multipoint recovery.

Reception Reporting Mechanism: A mechanism that defines a request/response procedure for the server and terminals to request and send reception reports. Reception reports described the status of the reception.

Source Block: A set of encoding symbols which is used as the basis for FEC encoding/decoding operations.

Streaming Delivery Session: An instance of delivery of a streaming service which is characterized by a start and end time and addresses of the IP flows used for delivery of the media streams between start and end time.

Streaming Service: A set of synchronized media streams delivered in a time-constrained or unconstrained manner for immediate consumption (during the reception).

Time Slice: A burst of MPE and MPE-FEC sections delivered over DVB-H using a time slicing method.

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

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
AL-FEC	Application Layer Forward Error Correction
AS	bandwidth modifier
AVP	Audio Video Profile
CC	Congestion Control
CCI	Congestion Control Indentifier
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 Unit
MPE	Multiprotocol encapsulation
MPEG-2 TS	MPEG-2 Transport Stream
MTU	Maximum Transmission Unit
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
RTP	Real-Time Transport Protocol
RTCP	Real-Time Transport Control Protocol
SBN	Source Block Number
SDP	Session Description Protocol
SPP	Service Purchase and Protection
StaR	Statistical Reporting for successful reception
TCP	Transmission Control Protocol
TIAS	bandwidth modifier
TO	Transport Object
TOI	Transport Object Identifier
TS	Transport Stream
TSI	Transport Session Identifier
UDP	User Datagram Protocol
URI	Uniform Resource Indentifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
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.

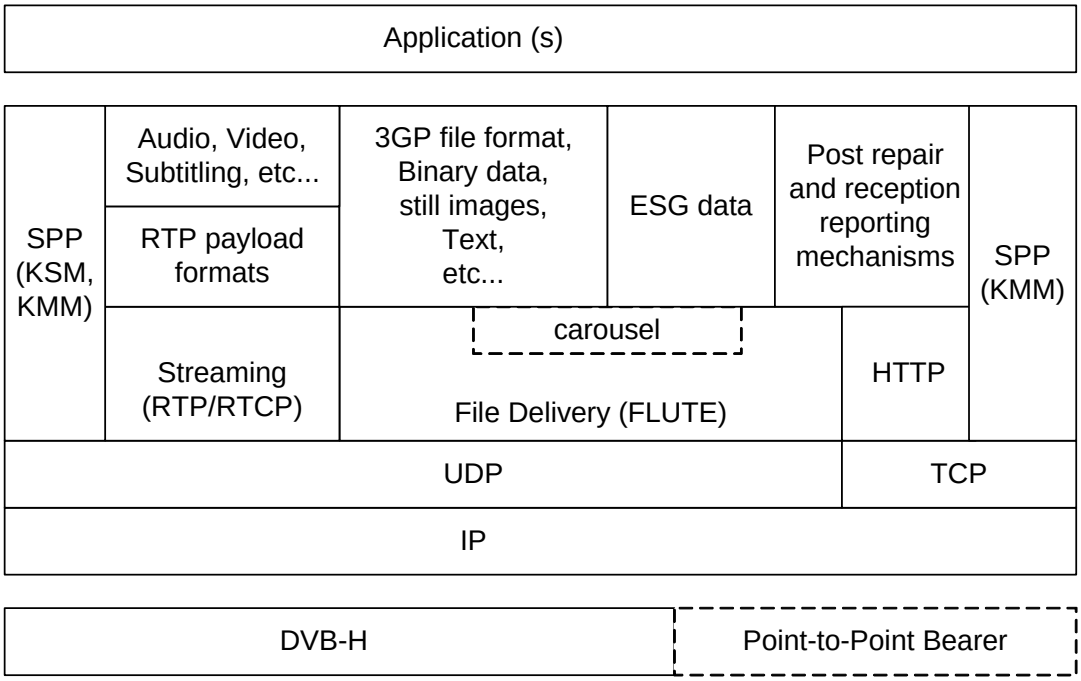


Figure 1: Baseline IPDC Protocol Stack for content delivery.

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 Specification [4].

5.1 RTP

RTP [5] shall be used to deliver real time audio and video streaming services.

The sender shall generate and send RTCP packets as defined in [5]. 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 [6], [7].

5.2.1 SDP Parameters for IPDC streaming sessions

Session Description 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.

Session Description 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.

5.2.1.1 Sender IP address

The SDP file associated to a streaming session MAY provide IP source address filtering information. In that case,the IP source address filtering shall be defined according to the source-filter attribute (“a=source-filter:”) [8] for both IPv4 and IPv6 sources, with the following exceptions:

1. The source-filter attribute shall only be in the session part of the session description (i.e., not per media).
2. The * value shall be used for the <dest-address> subfield.

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 [6], [7]. 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 [6] for RTP [5]. 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.

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=”) [6].

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" [9] 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 [10] 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.

5.2.2 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
t=3034423619 3042462419
b=AS:77
a=source-filter: incl IN IP6 * 2001:210:1:2:240:96FF:FE25:8EC9
m=video 4002 RTP/AVP  96
b=TIAS:62000
b=RR:0
b=RS:600
a=maxprate:17
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; packetization-mode=1; sprop-parameter-
sets=Z0IACpZTBYmI,aMljiA==
m=audio 4004 RTP/AVP  98
b=TIAS:15120
b=RR:0
b=RS:600
a=maxprate:10
a=rtpmap:98 AMR/8000
a=fmtp:98 octet-align=1
```

5.3 Hypothetical Receiver Buffering Model

5.3.1 Overview of the Proposed Buffering Model (INFORMATIVE)

A hypothetical receiver buffering modelis presented in Figure 2.

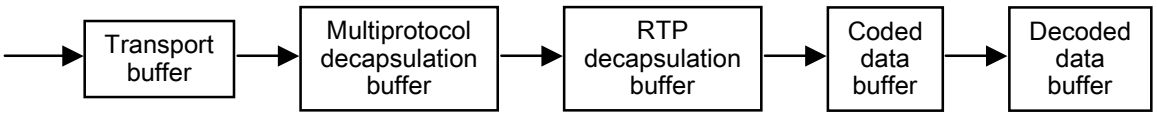


Figure 2: Hypothetical receiver buffering model.

The transport buffer receives MPEG-2 TS packets and removes any duplicate packets. Its operation is described in [11] and [12]. 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, one RTP decapsulation buffer per each IP stream, 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.

5.3.2 Multiprotocol Decapsulation Buffer (NORMATIVE)

The multiprotocol decapsulation (MPD) buffer model is applied to time-sliced elementary streams carrying IP streams.

Informative note: The value of the time_slicing element of the time slice and FEC indentifier 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 [11].
    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 [11].
    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 an IP stream.

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 signaled initial buffering delay (since the insertion of the first RTP packet) has expired. The signaling means for the initial buffering delay are specified in section 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 * (size of the previous ADU in bytes) / (1000 * (value of “b=AS” SDP attribute for the stream))

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

5.3.4 Signaling of Initial Buffering Delay (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 signaled delay guarantees pauseless decoding and playback. The value is expressed in milliseconds using an unsigned 16-bit integer in network byte order.

The initial buffering delay parameter SHALL be signaled to the receiver within the session description. In SDP, the initial buffering delay is provided a 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*16DIGIT

5.3.5 Conformance Requirements (NORMATIVE)

Any time-sliced elementary stream carrying IP streams 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 * 1.2 = 1024 * 255 * 1.2 = 313344 Bytes.*

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

- For any IP stream 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 IP stream per MPE-FEC frame is assumed.

  *B = maxMPERows * maxMPECols * (1 – (IPUDPHdr/MaxIPSize)) *1.2 = 1024 * 255 * (1 – 12/4096) * 1.2 = 312426 Byte.*

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.

6 Delivery Protocol for File Delivery Services

File delivery uses FLUTE [3] 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 [3]. FLUTE (File Delivery over Unidirectional Transport) [3] shall be used for this function. In addition to basic protocol the proposed file delivery solution is comprised of parts that further specify how FLUTE is used.

FLUTE is built on top of the Asynchronous Layered Coding (ALC) protocol instantiation [13]. ALC combines the Layered Coding Transport (LCT) building block [14], a congestion control building block and the Forward Error Correction (FEC) building block [15] to provide congestion controlled reliable asynchronous delivery of content to an

unlimited number of concurrent receivers from a single sender. As mentioned in [13], congestion control is not appropriate in the type of environment that IPDC system is 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.

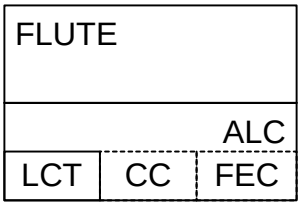


Figure 3: Building block structure of FLUTE.

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 this specification 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 [3], as well as ALC [13] and LCT [14] 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 to receive faster prior channels.

6.1.4 Symbol Encoding Algorithm

The "Compact No-Code FEC scheme" [20] (FEC Encoding ID 0, also known as "Null-FEC") SHALL be supported. The "Raptor FEC Scheme" (FEC Encoding Id 1) is defined in Section 8. This scheme consists of two distinct components as defined in Section 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 source 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 [3] shall be used.

In the case of the Raptor FEC Scheme, the algorithm described in Section 8 shall be used.

The values of N, Z, T and A shall be set such that the sub-block size is less than 256kB.

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 channelisation 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 [3], in the file delivery method using the generic GZip algorithm [16]. 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”.

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 1500 bytes as recommended in [17]. 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 [3] “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, 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 if 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 Section 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 section 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 explicitely 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 analyse the XML-encoded FDT Instances as they are received, identifies each requested file, associates it with FLUTE packets (using the TOI) and discover 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 section 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: The 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 [3] 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 [18].

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.

This list includes the parameters required by FLUTE [3].

These shall be expressed in SDP [6],[7],[8] 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 defined according to the source-filter attribute ("a=source-filter:") [6],[7],[8] for both IPv4 and IPv6 sources, with the following exceptions:

1. Exactly one source address may be specified by this attribute such that exclusive-mode shall not be used and inclusive-mode shall use exactly one source address in the <src-list>.
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. The * value shall be used for the <dest-address> subfield, even when the IPDC file delivery session employs only a single LCT (multicast) channel.

6.1.13.1.2 Number of channels

FLUTE session channelisation is the use of multiple LCT channels (multicast groups) to transport the files of a single FLUTE session. FLUTE session channelisation 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 [6] **.**

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 section 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 [6],[7]. 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 [14]).

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[6].

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=") [6].

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 [18] 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 6.1.4).

fec-inst-id = "instance-id=" inst-id

inst-id = 1*DIGIT (value is the FEC Instance ID used, valid FEC encoding Id are specified in 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 forFDT 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 [19] (e.g. "a=lang:EN-US").

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 [3],[13] 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 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 section 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 [20] 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 [3] 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 applies:

- 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 to FLUTE [3] for the signalling of FEC Object Transmission Information associated to FEC Encoding 0. When Raptor forward error correction code defined in Annex C is used, the EXT_FTI format is defined in section 8.1.3.

6.1.14.3 Signalling of Parameters with FDT Instances

The FLUTE FDT Instance schema defined in Section 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 unmodifiable, 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 ;
- Content-MD5 (Checksum of the file as defined in [3].)

6.1.14.4 Signalling of Parameters Out-band

Support of session description as in section 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 this 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>
```

6.2 Download & 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.

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

6.2.1.1 Static file delivery session

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.

Implementation using FLUTE

A static file delivery session is realised 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

Definition

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

Implementation using FLUTE

A fixed content delivery session is realised 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”

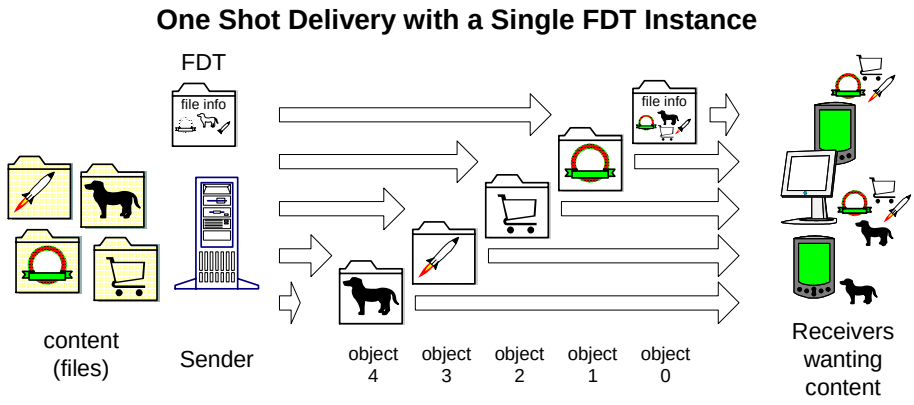


Figure 4: Example of fixed content delivery session.

6.2.1.3 Dynamic file delivery session

Definition

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

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

Definition

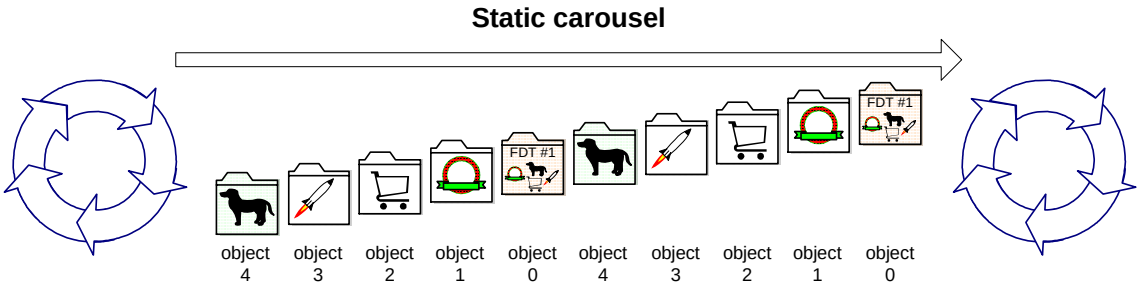


Figure 5: Example of static file delivery carousel.

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.

Implementation using FLUTE

A static file delivery carousel is realized as fixed content delivery session in section 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 minimised 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

Definition

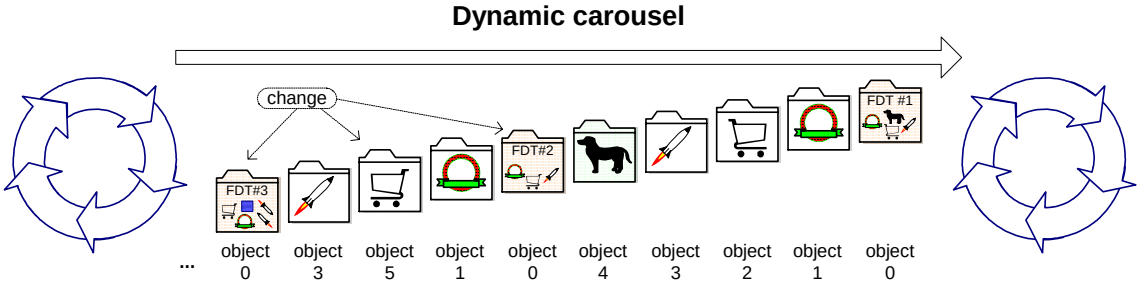


Figure 6: Example of dynamic file delivery carousel.

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

Implementation using FLUTE

A dynamic file delivery carousel is realized in the same way as dynamic file delivery session specified in section 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 minimised 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.

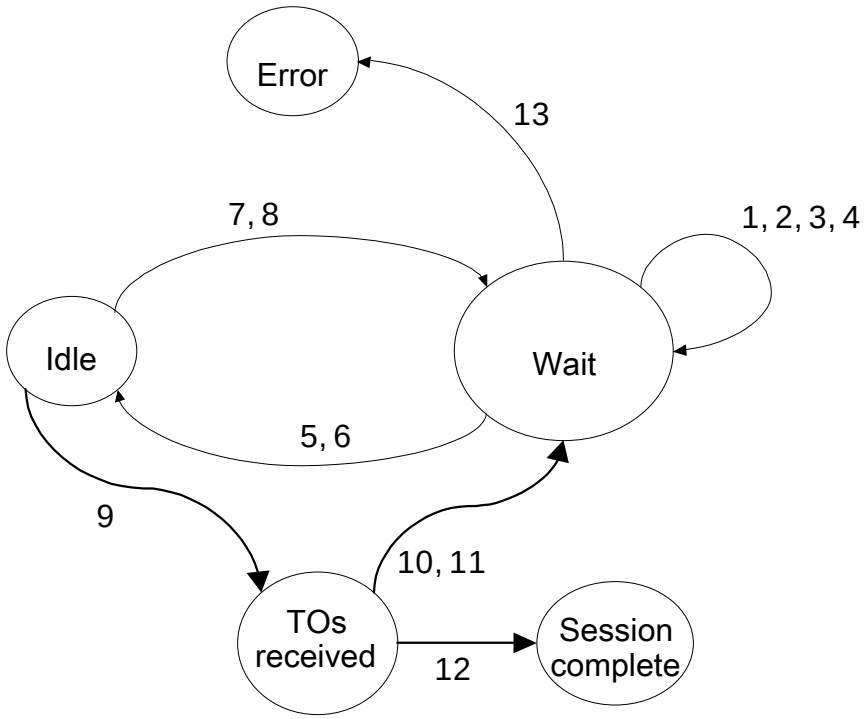


Figure 7: State machine.

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. 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 weight timer t1 for that TOI. This transition 2 can occur only if there still are one or more active fragment wait timers t1. 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 are still one or more active table wait timers t2. 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. 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. 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 from 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. A transition 9 from the wait state to the "TOs received" state occurs when the last missing file fragment is received. This resets and starts the object wait timer t3.

The parameters SHOULD be signalled in the session description as described in 6.1.13.2.

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 section 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 section 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.

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 section 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 server URIs from the list of repair servers that is provided by the associated delivery procedure description instance.

The server URIs may also be provided as IP addresses to avoid DNS queries for address resolution. The repair server 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 “serverURI” elements, each of which shall be a child element of the “procedure” element. The “serverURI” element provides the references to the file repair server via the standard XMLSchema "anyURI" type value. At least one “serverURI” element shall be present.

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 [21]. If the receiver needs repair data for more than one file received, the receiver shall send separate HTTP GET requests for each file. The repair request is routed to the repair server IP address resolved from the selected “serverURI”.

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. null FEC or systematic FEC), then only source symbols shall be requested for repair.

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 are either the 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 file repair request shall include the URI of the file for which it is requesting the repair data. The URI is required to uniquely identify the file (resource). The (SBN, ESI) of elements sufficient to complete the file reception are URL-encoded and included in the HTTP GET request.

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 (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 general HTTP URL syntax is as follows:

http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

For File Repair Request:

query = application *("&" [sbn_info])

application = "ipdc-flute-repair"

For FEC encoding ID value 128:

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 SBNs

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

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

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. Then the HTTP GET request is as follows:

```
GET www.example.com/news/latest.3gp?ipdc-flute-
repair&SBN=5;ESI=12+SBN=20;ESI=27 HTTP/1.1
```

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

- 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 behavior 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 4 response scenarios:

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

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 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 [21] 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.

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)
- Content transfer encoding, set to binary

The Content-Type shall be set to “application/simpleSymbolContainer”, which denotes that the message body is a simple container of encoding symbols as described below.

This header is as follows:

```
HTTP/1.1 200 OK
Content-Type: application/simpleSymbolContainer
Content-Transfer-Encoding: binary
```

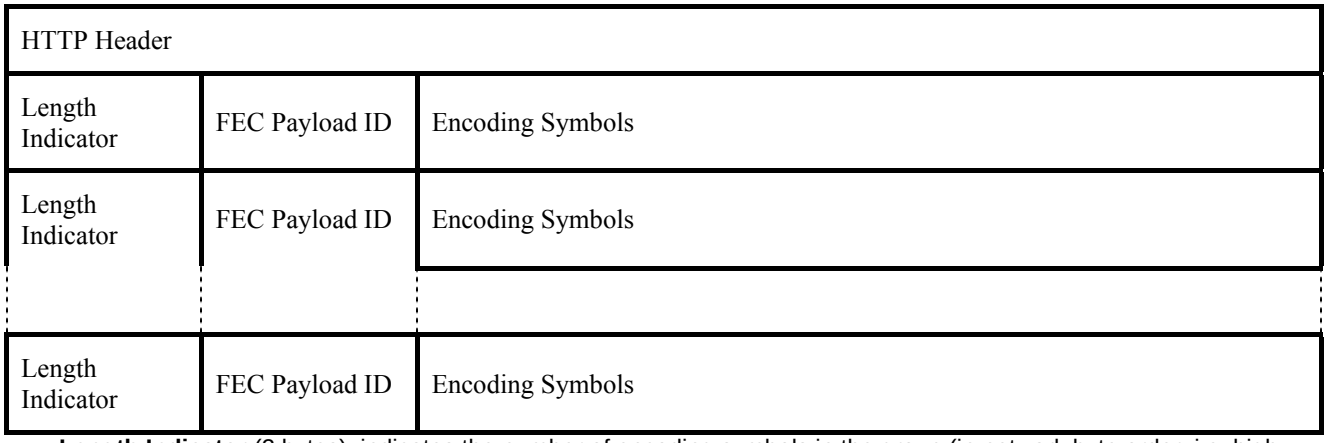
Note, other HTTP headers may also be used but are not mandated by this mechanism.

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).



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

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.

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

Section 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 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 minimising 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 can not 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 session description.

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 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 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 serverURI 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 serverURI 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 unrecognised message (not a recognisable HTTP response)
4. The server returns an HTTP server error status code (in the range 500-505)

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:

4. Identifies the complete reception of a content item (e.g. a file).
5. Determines the need to report reception 7.4.3.
6. 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 clause 7.4.4 and 7.4.5.
7. Sends a reception report request message to the selected server at the selected time.

Then the server

2. 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:

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

c. 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.

d. 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 to100. 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 initialisation 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 initialisation 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 [21].

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 [21] 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 (clause 7.5.3). An informative example of a single reception report XML object is also given (clause 7.5.4).

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” || “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 serverURI attribute value and format is taken from the respective associatedProcedureDescription serverURI 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 [21].

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 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="serverURI" type="xs:anyURI" minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="offsetTime" type="xs:unsignedLong" use="optional"/>
        <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"/>
                <xs:attribute name="forceTimingIndependence" type="xs:boolean" use="optional"/>
                <xs:attribute name="reportType" type="Report-Type" use="optional"/>
            </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: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">
        <serverURI>"http://ipdcrepair.operator.umts/"</serverURI>
        <serverURI>"http://ipdcrepair1.operator.umts/"</serverURI>
        <serverURI>"http://ipdcrepair2.operator.umts/"</serverURI>
    </postFileRepair>
    <bmFileRepair sessionDescriptionURI="http://www.example.com/ipdc/session1.sdp"/>
    <postReceptionReport
                offsetTime="5"
                randomTimePeriod="10"
                reportType="star-all"
                samplePercentage="100"
                forceTimingIndependence="0">
        <serverURI>"http://ipdcrepair.operator.umts/"</serverURI>
    </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="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="starType">
<xs:sequence>
<xs:element name="fileURI" type="FileSuccessType" minOccurs="0" 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="serverURI" 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"/>
</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 [15], for the Raptor forward error correction code defined in Annex C 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 8.1.2 and 8.1.3 as well as the source packets constructed according to C.3.1 and C.3.2.1. 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 Annex C.

8.1.2 FEC payload ID

The FEC Payload ID shall be a 4 octet field defined as follows in Figure 9:

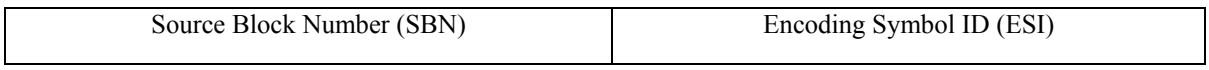


Figure 9: FEC Payload ID format.

Source Block Number (SBN), (16 bits): An integer identifier for the source block that the encoding symbols within the packet relate to.

Encoding Symbol ID (ESI), (16 bits): An integer identifier for the encoding symbols within the packet.

The interpretation of the Source Block Number and Encoding Symbol Identifier is defined in Annex C.

8.1.3 FEC Object Transmission Information

The FEC Object Transmission information shall consist of:

- The FEC Encoding ID
- The Transfer Length (*F*)
- The parameters *T*, *Z*, *N* and *A* defined in Annex C.

When EXT_FTI is used to communicate the Object Transmission Information, the FEC Encoding ID and Transfer Length shall be coded according to FLUTE [3]. The other parameters shall be encoded in the FEC Encoding ID specific portion of the EXT_FTI field as shown in Figure 10 below.

General EXT_FTI format	Encoding Symbol Length (*T*)	
Number of Source Blocks (*Z*)	Number of Sub-Blocks (*N*)	Symbol Alignment Parameter (*A*)

Figure 10: FEC Encoding ID-specific EXT_FTI format.

The parameters *T* and *Z* are 16 bit unsigned integers, *N* and *A* are 8 bit unsigned integers.

When the FDT is used to deliver the FEC Object Transmission Information, then the FEC Encoding ID, Transfer Length (F) and Encoding Symbol Length (*T*) shall be encoded using the Transfer-Length, FEC-OTI-Encoding-ID and FEC-OTI-Encoding-Symbol-Length elements defined in the FDT Schema. The remaining parameters *Z*, *N*, *A*, shall be encoded as a 4 byte field within the FEC-OTI-Scheme-Specific-Info field, according to the format specified in Figure 10 above, excepting the Encoding Symbol Length field.

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 section 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 section 9.2.

9.1 Subtitling using 3GPP Timed Text Format

In the character encoded format subtitles, the 3GPP Timed Text Format [22] and the RTP payload format for 3GPP Timed Text [23] SHALL be used for formatting the subtitling text.

Additionally, the following restrictions and extensions apply.

9.1.1 Unicode Support

The Unicode 3.0 [24] standard shall be used. Terminals shall correctly decode UTF-8 format as specified in [25]. 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 bit 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 don't support blinking shall ignore it.

9.1.5 Delivery of subtitling text

The RTP payload format for 3GPP Timed Text [23] 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 section 9.1.6.

The sender shall use an RTP timestamp clockrate of 1000Hz.

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 text
- 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
t=3034423619 3042462419
b=AS:77
a=source-filter: incl IN IP6 * 2001:210:1:2:240:96FF:FE25:8EC9
m=video 4002 RTP/AVP 97 96 100
a=maxprate:17
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=42A01E; packetization-mode=1; sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==
m=text 4006 RTP/AVP 102
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 [26] with the corrigenda and extensions specified in [27].

DVB subtitles are positioned using a 'page composition segment' as defined in chapter 7.2.1 in [26]. The chapter 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 chapter 'Pixel addressing and scaling of bitmap based subtitles based on [26] will define how pixel in non 720 by 576 systems are addressed and how bitmaps are scaled. Chapter 'Pixel addressing of non 720 by 576 subtitles' will define extensions to [26] for the use of subtitles which aren't authored for 720x576 systems. Receivers SHALL support both modes.

9.2.1 Pixel addressing and scaling of bitmap based subtitles based on [26].

Subtitles authored for 720x576 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 720x576 is defined to be 'full screen'.

'Full screen' in this respect means that 720x576 have to be mapped to the real resolution of the video. In case of a CIF video, 720x576 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 2: 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

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

with f_x and f_y being the scaling factors:

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

As the bitmaps in this case are authored for 720x576 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' (chapter 7.2.1 in [26]) 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' (chapter 7.2.2 in [26]) 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 doesn't have the resolution of 720x576 it wouldn't make sense to create the subtitles for this resolution. The page composition and region composition segments from [26] however assume subtitles which are authored for 720x576. In order to signal the intended resolution of the subtitles a new segment type is defined in Table 3 with the segment_type id of 0x14 (see chapter 7.2 in [26]).

Table 3: 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
}		

authored_width The width the subtitles are authored for.

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:

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

In the case that the authored size is the same as the actual display resolution no scaling of neither the bitmaps nor 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 [28] defines in chapter 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 [28] 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

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 [29] and in this specification.

This section 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 the Informative 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 4) are defined for this mime type and are signalled in the "fmtp:" line:

Table 4: Parameters of the mime type ipdc-ksm.

Parameter	Mand / Opt	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.

IPDCStreamId uniquely identifies the key stream within the scope of the service (the SDP file) and allows later referencing.

IPDCKMSId identifies the Key Management System. This identifier is globally unique and is allocated by DVB.

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.

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 5: Parameters of the mime type ipdc-kmm.

Parameter	Mand / Opt	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 11).

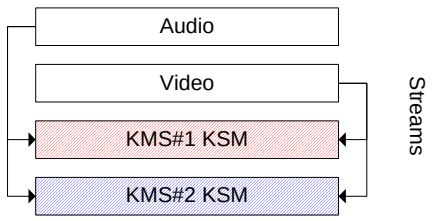


Figure 11: KSM Stream Binding.

This way, the KMS will only listen to and process the KSM stream coming on the relevant IP connection. SDP [6] 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 6: Definition of IPDCKSMStream attribute.

Attribute	Mand / Opt	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 lookup 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.

```
v=0
o=IPDC 2890844526 2890842807 IN IP4 126.16.64.4
s=A SPP stream
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
a=IPDCKSMStream:10
a=IPDCKSMStream:11
m=audio 49170 RTP/AVP 0
a=lang:en
m=video 51372 RTP/AVP 31
m=audio 52002 RTP/AVP 0
a=lang:ES
a=IPDCKSMStream:13
a=IPDCKSMStream:14
```

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).

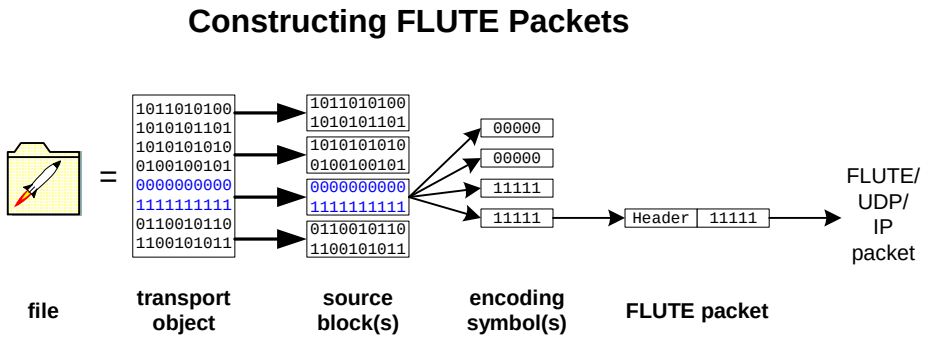


Figure A-1: Constructing of FLUTE Packets.

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 (the 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 1000000 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 50000 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 effeciency metric. The effeciency of a repair mode can be calculated as follows: $E = \frac{\text{number of receivers with successful recovery}}{\text{cost of transmission of repair data}}$

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 octect 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 section 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.

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

However, there are some terminals that either don't 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:

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

Decision on the repair mode

The service operator should uses 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 section 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 netowork 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: FEC encoder and decoder specification

This Section specifies the systematic Raptor forward error correction code and its application to CBMS. Raptor is a fountain code, i.e., as many encoding symbols as needed, up to 65536, can be generated by the encoder on-the-fly from the source symbols of a block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

The code described in this document is a systematic code, that is, the original source symbols are sent unmodified from sender to receiver, as well as a number of repair symbols. The specification is technically identical to the one in 3GPP MBMS [1].

C.1 Definitions, Symbols and abbreviations

C.1.1 Definitions

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

Source block: a block of *K* source symbols which are considered together for Raptor encoding purposes.

Source symbol: the smallest unit of data used during the encoding process. All source symbols within a source block have the same size.

Encoding symbol: a symbol that is included in a data packet. The encoding symbols consist of the source symbols and the repair symbols. Repair symbols generated from a source block have the same size as the source symbols of that source block.

Systematic code: a code in which the source symbols are included as part of the encoding symbols sent for a source block.

Repair symbol: the encoding symbols sent for a source block that are not the source symbols. The repair symbols are generated based on the source symbols.

Intermediate symbols: symbols generated from the source symbols using an inverse encoding process . The repair symbols are then generated directly from the intermediate symbols. The encoding symbols do not include the intermediate symbols, i.e., intermediate symbols are not included in data packets.

Symbol: a unit of data. The size, in bytes, of a symbol is known as the symbol size.

Encoding symbol group: a group of encoding symbols that are sent together, i.e., within the same packet whose relationship to the source symbols can be derived from a single Encoding Symbol ID.

Encoding Symbol ID: information that defines the relationship between the symbols of an encoding symbol group and the source symbols.

Encoding packet: data packets that contain encoding symbols

Sub-block: a source block is sometime broken into sub-blocks, each of which is sufficiently small to be decoded in working memory. For a source block consisting of K source symbols, each sub-block consists of K sub-symbols, each symbol of the source block being composed of one sub-symbol from each sub-block.

Sub-symbol: part of a symbol. Each source symbol is composed of as many sub-symbols as there are sub-blocks in the source block.

Source packet: data packets that contain source symbols.

Repair packet: data packets that contain repair symbols.

C.1.2. Symbols

i, j, x, h, a, b, d, v, m represent positive integers

ceil(x)	denotes the smallest positive integer which is greater than or equal to x
choose(i,j)	denotes the number of ways j objects can be chosen from among i objects without repetition
floor(x)	denotes the largest positive integer which is less than or equal to x
i % j	denotes i modulo j
X ^ Y	denotes, for equal-length bit strings X and Y, the bitwise exclusive-or of X and Y
A	denote a symbol alignment parameter. Symbol and sub-symbol sizes are restricted to be multiples of A.
$\mathbf{A}^T$	denotes the transposed matrix of matrix **A**
$\mathbf{A}^{-1}$	denotes the inverse matrix of matrix **A**
K	denotes the number of symbols in a single source block
K_{MAX}	denotes the maximum number of source symbols that can be in a single source block. Set to 8192.
L	denotes the number of pre-coding symbols for a single source block
S	denotes the number of LDPC symbols for a single source block
H	denotes the number of Half symbols for a single source block
C	denotes an array of intermediate symbols, $C[0]$, $C[1]$, $C[2]$,…, $C[L-1]$
C’	denotes an array of source symbols, $C'[0]$, $C'[1]$, $C'[2]$,…, $C'[K-1]$
X	a non-negative integer value
V_0, V_1	two arrays of 4-byte integers, $V_0[0]$, $V_0[1]$,…, $V_0[255]$ and $V_1[0]$, $V_1[1]$,…, $V_1[255]$
Rand[X, i, m]	a pseudo-random number generator

Deg[v] a degree generator

LTEnc[K, **C** ,(d, a, b)] a LT encoding symbol generator

Trip[K, X] a triple generator function

G the number of symbols within an encoding symbol group

N the number of sub-blocks within a source block

T the symbol size in bytes. If the source block is partitioned into sub-blocks, then $T = T' \cdot N$.

T' the sub-symbol size, in bytes. If the source block is not partitioned into sub-blocks then T' is not relevant.

F the file size, for file delivery, in bytes

I the sub-block size in bytes

P for file delivery, the payload size of each packet, in bytes, that is used in the recommended derivation of the file delivery transport parameters. For streaming, the payload size of each repair packet, in bytes, that is used in the recommended derivation of the streaming transport parameters.

Q $Q = 65521$, i.e., Q is the largest prime smaller than 2^{16}

Z the number of source blocks, for file delivery

$J(K)$ the systematic index associated with K

G denotes any generator matrix

$\mathbf{I}_S$ denotes the SxS identity matrix

$\mathbf{0}_{SxH}$ denotes the SxH zero matrix

C.1.3 Abbreviations

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

ESI	Encoding Symbol ID
LDPC	Low Density Parity Check
LT	Luby Transform
SBN	Source Block Number
SBL	Source Block Length (in units of symbols)

C.2. Overview

The Raptor forward error correction code can be applied to the CBMS file delivery application.

The principle component of the systematic Raptor code is the basic encoder described in Section C.4. First, it is described how to derive values for a set of intermediate symbols from the original source symbols such that knowledge of the intermediate symbols is sufficient to reconstruct the source symbols. Secondly, the encoder produces repair symbols which are each the exclusive OR of a number of the intermediate symbols. The encoding symbols are the combination of the source and repair symbols. The repair symbols are produced in such a way that the intermediate symbols and therefore also the source symbols can be recovered from any sufficiently large set of encoding symbols.

This document defines the systematic Raptor code encoder. A number of different decoding algorithms are possible. An efficient decoding algorithm is provided in Section C.7.

The construction of the intermediate and repair symbols is based in part on a pseudo-random number generator described in Section C.4. This generator is based on a fixed set of 512 random numbers which must be available to both sender and receiver. These are provided in Section C.6.

Finally, the construction of the intermediate symbols from the source symbols is governed by a 'systematic index', values of which are provided in Section C.5 for source block sizes from 4 source symbols to K_{MAX} = 8192 source symbols.

C.3. File Delivery

C.3.1. Source block construction

C.3.1.1. General

In order to apply the Raptor encoder to a source file, the file may be broken into $Z \geq 1$ blocks, known as *source blocks*. The Raptor encoder is applied independently to each source block. Each source block is identified by a unique integer Source Block Number (SBN), where the first source block has SBN zero, the second has SBN one, etc. Each source block is divided into a number, K, of *source symbols* of size T bytes each. Each source symbol is identified by a unique integer Encoding Symbol Identifier (ESI), where the first source symbol of a source block has ESI zero, the second has ESI one, etc.

Each source block with K source symbols is divided into $N \geq 1$ sub-blocks, which are small enough to be decoded in the working memory. Each sub-block is divided into K sub-symbols of size T'.

Note that the value of K is not necessarily the same for each source block of a file and the value of T' may not necessarily be the same for each sub-block of a source block. However, the symbol size T is the same for all source blocks of a file and the number of symbols, K is the same for every sub-block of a source block. Exact partitioning of the file into source blocks and sub-blocks is described in C.3.1.2 below.

Figure C.3.1.1.-1 shows an example source block placed into a two dimensional array, where each entry is a T'-byte sub-symbol, each row is a sub-block and each column is a source symbol. In this example, the value of T' is the same for every sub-block. The number shown in each sub-symbol entry indicates their original order within the source block. For example, the sub-symbol numbered K contains bytes $T' \cdot K$ through $T' \cdot (K+1)-1$ of the source block. Then, source symbol i is the concatenation of the ith sub-symbol from each of the sub-blocks, which corresponds to the sub-symbols of the source block numbered i, $K+i$, $2 \cdot K+i$,…, $(N-1) \cdot K+i$.

0	1	2	…	…	…	…	K-1
K	K+1	K+2	…	…	…	…	$2 \cdot K$-1
$2 \cdot K$	$2 \cdot K$+1	$2 \cdot K$+2	…	…	…	…	$3 \cdot K$-1
…	…	…	…	…	…	…	
$(N-1) \cdot K$	…	…	…	…	…	…	$N \cdot K$-1

Figure C.3.1.1-1: Source symbols from sub-symbols– the 3 highlighted columns show source symbols 0, 2 and *K*-1.

C.3.1.2 Source block and sub-block partitioning

The construction of source blocks and sub-blocks is determined based on five input parameters, F, A, T, Z and N and a function Partition[]. The five input parameters are defined as follows:

- F the size of the file, in bytes
- A a symbol alignment parameter, in bytes
- T the symbol size, in bytes, which must be a multiple of A
- Z the number of source blocks
- N the number of sub-blocks in each source block

These parameters shall be set so that ceil(ceil(F/T)/Z) ≤ K_{MAX}. Recommendations for derivation of these parameters are provided in Section C.3.4.

The function Partition[] takes a pair of integers (I, J) as input and derives four integers (I_L, I_S, J_L, J_S) as output. Specifically, the value of Partition[I, J] is a sequence of four integers (I_L, I_S, J_L, J_S), where I_L = ceil(I/J), I_S = floor(I/J), $J_L = I - I_S \cdot J$ and $J_S = J - J_L$. Partition[] derives parameters for partitioning a block of size I into J approximately equal sized blocks. Specifically, J_L blocks of length I_L and J_S blocks of length I_S.

The source file shall be partitioned into source blocks and sub-blocks as follows:

Let,

K_t = ceil(F/T)

(K_L, K_S, Z_L, Z_S) = Partition[K_t, Z]

(T_L, T_S, N_L, N_S) = Partition[T/A, N]

Then, the file shall be partitioned into $Z = Z_L + Z_S$ contiguous source blocks, the first Z_L source blocks each having length $K_L \cdot T$ bytes and the remaining Z_S source blocks each having $K_S \cdot T$ bytes.

If $K_t \cdot T > F$ then for encoding purposes, the last symbol shall be padded at the end with $K_t \cdot T - F$ zero bytes.

Next, each source block shall be divided into $N = N_L + N_S$ contiguous sub-blocks, the first N_L sub-blocks each consisting of K contiguous sub-symbols of size of $T_L \cdot A$ and the remaining N_S sub-blocks each consisting of K contiguous sub-symbols of size of $T_S \cdot A$. The symbol alignment parameter A ensures that sub-symbols are always a multiple of A bytes.

Finally, the mth symbol of a source block consists of the concatenation of the mth sub-symbol from each of the N sub-blocks.

C.3.2. Encoding packet construction

C.3.2.1. General

Each encoding packet contains the following information:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- encoding symbol(s)

Each source block is encoded independently of the others. Source blocks are numbered consecutively from zero.

Encoding Symbol ID values from 0 to K-1 identify the source symbols. Encoding Symbol IDs from K onwards identify repair symbols.

C.3.2.2 Encoding packet construction

Each encoding packet either consists entirely of source symbols (source packet) or entirely of repair symbols (repair packet). A packet may contain any number of symbols from the same source block. In the case that the last symbol in the packet includes padding bytes added for FEC encoding purposes then these bytes need not be included in the packet. Otherwise, only whole symbols shall be included.

The Encoding Symbol ID, X, carried in each source packet is the Encoding Symbol ID of the first source symbol carried in that packet. The subsequent source symbols in the packet have Encoding Symbol IDs, X+1 to X+G-1, in sequential order, where G is the number of symbols in the packet.

Similarly, the Encoding Symbol ID, X, placed into a repair packet is the Encoding Symbol ID of the first repair symbol in the repair packet and the subsequent repair symbols in the packet have Encoding Symbol IDs X+1 to X+G-1 in sequential order, where G is the number of symbols in the packet.

Note that it is not necessary for the receiver to know the total number of repair packets. The G repair symbol triples ($d[0]$, $a[0]$, $b[0]$),…, ($d[G\text{-}1]$, $a[G\text{-}1]$, $b[G\text{-}1]$) for the repair symbols placed into a repair packet with ESI X are computed using the Triple generator defined in C.5.3.4 as follows:

For each $i = 0, \ldots, G\text{-}1$

$(d[i], a[i], b[i]) = \text{Trip}[K,X+i]$

The G repair symbols to be placed in repair packet with ESI X are calculated based on the repair symbol triples as described in Section C.5.3 using the intermediate symbols $\mathbf{C}$ and the LT encoder $\text{LTenc}[K, \mathbf{C}, (d[i], a[i], b[i])]$.

C.3.3. Transport

This section describes the information exchange between the Raptor encoder/decoder and any transport protocol making use of Raptor forward error correction for file delivery.

The Raptor encoder and decoder for file delivery require the following information from the transport protocol:

- The file size, F, in bytes
- The symbol alignment parameter, A
- The symbol size, T, in bytes, which must be a multiple of A
- The number of source blocks, Z
- The number of sub-blocks in each source block, N

The Raptor encoder for file delivery additionally requires:

- the file to be encoded, F bytes

The Raptor encoder supplies the transport protocol with encoding packet information consisting, for each packet, of:

- Source Block Number (SBN)
- Encoding Symbol ID (ESI)
- encoding symbol(s)

The transport protocol shall communicate this information transparently to the Raptor decoder.

Suitable transport protocols based on FLUTE/ALC and HTTP are defined in this specification.

C.3.4. Example Parameters

C.3.4.1 Parameter derivation algorithm

This section provides recommendations for the derivation of the four transport parameters, A, T, Z and N. This recommendation is based on the following input parameters:

- F the file size, in bytes
- W a target on the sub-block size, in bytes
- P the maximum packet payload size, in bytes, which is assumed to be a multiple of A
- A the symbol alignment factor, in bytes
- K_{MAX} the maximum number of source symbols per source block.
- K_{MIN} a minimum target on the number of symbols per source block
- G_{MAX} a maximum target number of symbols per packet

Based on the above inputs, the transport parameters T, Z and N are calculated as follows:

Let,

$G = \min\{\text{ceil}(P \cdot K_{MIN}/F), P/A, G_{MAX}\}$ - the approximate number of symbols per packet

$T = \text{floor}(P/(A \cdot G)) \cdot A$

$K_t = \text{ceil}(F/T)$ - the total number of symbols in the file

$Z = \text{ceil}(K_t / K_{MAX})$

$N = \min\{\text{ceil}(\text{ceil}(K_t/Z) \cdot T/W), T/A\}$

The values of G and N derived above should be considered as lower bounds. It may be advantageous to increase these values, for example to the nearest power of two. In particular, the above algorithm does not guarantee that the symbol size, T, divides the maximum packet size, P, and so it may not be possible to use the packets of size exactly P. If, instead, G is chosen to be a value which divides P/A, then the symbol size, T, will be a divisor of P and packets of size P can be used.

Recommended settings for the input parameters, W, A, K_{MIN} and G_{MAX} are as follows:

$W = 256$ KB $A = 4$ $K_{MIN} = 1024$ $G_{MAX} = 10$

C.3.4.2 Examples

The above algorithm leads to transport parameters as shown in Table C.3.4.2-1 below, assuming the recommended values for W, A, K_{MIN} and G_{MAX} and $P = 512$:

Table C.3.4.2-1: Examples of transport parameters.

File size F	G	Symbol size T	$G*T$	K_t	Source blocks Z	Sub-blocks N	K_L	K_S	$T_L \cdot A$	$T_S \cdot A$
100 KB	6	84	504	1,220	1	1	1,220	1,220	N/A	N/A
100 KB	8	64	512	1,600	1	1	1,600	1,600	N/A	N/A
300 KB	2	256	512	1,200	1	2	1,200	1,200	128	128
1,000 KB	1	512	512	2,000	1	5	2,000	2,000	104	100
3,000 KB	1	512	512	6,000	1	12	6,000	6,000	44	40
10,000 KB	1	512	512	20,000	3	14	6,666	6,667	40	36

C.4. Systematic Raptor encoder

C.4.1. Encoding overview

The systematic Raptor encoder is used to generate *repair symbols* from a source block that consists of *K source symbols*.

Symbols are the fundamental data units of the encoding and decoding process. For each source block (sub-block) all symbols (sub-symbols) are the same size. The atomic operation performed on symbols (sub-symbols) for both encoding and decoding is the exclusive-or operation.

Let $C'[0],\ldots, C'[K-1]$ denote the K source symbols.

Let $C[0],\ldots, C[L-1]$ denote L intermediate symbols.

The first step of encoding is to generate a number, $L > K$, of intermediate symbols from the K source symbols. In this step, K source triples $(d[0], a[0], b[0]), \ldots, (d[K-1], a[K-1], b[K-1])$ are generated using the Trip[] generator as described in Section C.5.4.4. The K source triples are associated with the K source symbols and are then used to determine the L intermediate symbols $C[0],\ldots, C[L-1]$ from the source symbols using an inverse encoding process. This process can be can be realized by a Raptor decoding process.

Certain "pre-coding relationships" must hold within the L intermediate symbols. Section C.5.2 describes these relationships and how the intermediate symbols are generated from the source symbols.

Once the intermediate symbols have been generated, repair symbols are produced and one or more repair symbols are placed as a group into a single data packet. Each repair symbol group is associated with an Encoding Symbol ID (ESI) and a number, G, of encoding symbols. The ESI is used to generate a triple of three integers, (d, a, b) for each repair symbol, again using the Trip[] generator as described in Section C.5.4.4. This is done as described in Sections C.3 and C.4 using the generators described in Section C.5.4 . Then, each (d,a,b)-triple is used to generate the corresponding repair symbol from the intermediate symbols using the LTEnc[K, $C[0],\ldots, C[L-1]$, (d,a,b)] generator described in Section C.5.4.3.

C.4.2. First encoding step: Intermediate Symbol Generation

C.4.2.1 General

The first encoding step is a pre-coding step to generate the L intermediate symbols $C[0], \ldots, C[L-1]$ from the source symbols $C'[0], \ldots, C'[K-1]$. The intermediate symbols are uniquely defined by two sets of constraints:

1. The intermediate symbols are related to the source symbols by a set of *source symbol triples.* The generation of the source symbol triples is defined in Section C.5.2.2 using the the Trip[] generator as described in Section C.5.4.4.

2. A set of pre-coding relationships hold within the intermediate symbols themselves. These are defined in Section C.5.2.3.

The generation of the L intermediate symbols is then defined in Section C.5.2.4.

C.4.2.2 Source symbol triples

Each of the K source symbols is associated with a triple $(d[i], a[i], b[i])$ for $0 \le i < K$. The source symbol triples are determined using the Triple generator defined in Section C.5.4.4 as:

For each i, $0 \le i < K$

$(d[i], a[i], b[i]) = \text{Trip}[K, i]$

C.4.2.3 Pre-coding relationships

The pre-coding relationships amongst the L intermediate symbols are defined by expressing the last L-K intermediate symbols in terms of the first K intermediate symbols.

The last L-K intermediate symbols $C[K],\ldots,C[L-1]$ consist of *S LDPC symbols* and *H Half symbols* The values of S and H are determined from K as described below. Then $L = K+S+H$.

Let

X be the smallest positive integer such that $X\cdot(X-1) \ge 2\cdot K$.

S be the smallest prime integer such that $S \ge \text{ceil}(0.01\cdot K) + X$

H be the smallest integer such that $\text{choose}(H,\text{ceil}(H/2)) \ge K + S$

$H' = \text{ceil}(H/2)$

$L = K+S+H$

$C[0],\ldots, C[K-1]$ denote the first K intermediate symbols

$C[K],\ldots, C[K+S-1]$ denote the S LDPC symbols, initialised to zero

$C[K+S],\ldots, C[L-1]$ denote the H Half symbols, initialised to zero

The S LDPC symbols are defined to be the values of $C[K],\ldots, C[K+S-1]$ at the end of the following process:

For $i = 0,\ldots,K-1$ do

$a = 1 + (\text{floor}(i/S) \% (S-1))$

$b = i \% S$

$C[K + b] = C[K + b] \wedge C[i]$

$b = (b + a) \% S$

$C[K + b] = C[K + b] \wedge C[i]$

$b = (b + a) \% S$

$C[K + b] = C[K + b] \wedge C[i]$

The H Half symbols are defined as follows:

Let

$g[i] = i \wedge (\text{floor}(i/2))$ for all positive integers i

Note: $g[i]$ is the Gray sequence, in which each element differs from the previous one in a single bit position

$g[j,k]$ denote the j^{th} element, j=0, 1, 2, …, of the subsequence of $g[i]$ whose elements have exactly k non-zero bits in their binary representation

Then, the Half symbols are defined as the values of $C[K+S],\ldots, C[L-1]$ after the following process:

For $h = 0,\ldots,H-1$ do

For $j = 0,\ldots,K+S-1$ do

If bit h of $g[j,H']$ is equal to 1 then $C[h+K+S] = C[h+K+S] \wedge C[j]$.

C.4.2.4 Intermediate symbols

C.5.2.4.1 Definition

Given the K source symbols $C'[0], C'[1],\ldots, C'[K-1]$ the L intermediate symbols $C[0], C[1],\ldots, C[L-1]$ are the uniquely defined symbol values that satisfy the following conditions:

1. The K source symbols $C'[0], C'[1],\ldots, C'[K-1]$ satisfy the K constraints

$$C'[i] \equiv \text{LTEnc}[K, (C[0],\ldots, C[L-1]), (d[i], a[i], b[i])], \text{ for all } i,\ 0 \le i < K.$$

2. The L intermediate symbols $C[0], C[1],\ldots, C[L-1]$ satisfy the pre-coding relationships defined in C.5.2.3.

C.5.2.4.2 Example method for calculation of intermediate symbols

This subsection describes a possible method for calculation of the L intermediate symbols $C[0], C[1],\ldots, C[L-1]$ satisfying the constraints in C.5.2.4.1

The generator matrix **G** for a code which generates N output symbols from K input symbols is an NxK matrix over $GF(2)$, where each row corresponds to one of the output symbols and each column to one of the input symbols and where the i^{th} output symbol is equal to the sum of those input symbols whose column contains a non-zero entry in row i.

Then, the L intermediate symbols can be calculated as follows:

Let

C denote the column vector of the L intermediate symbols, $C[0], C[1],\ldots, C[L-1]$.

D denote the column vector consisting of $S+H$ zero symbols followed by the K source symbols $C'[0], C'[1], \ldots, C'[K-1]$

Then the above constraints define an LxL matrix over $GF(2)$, **A**, such that:

$$\mathbf{A}\cdot\mathbf{C} = \mathbf{D}$$

The matrix **A** can be constructed as follows:

Let:

$\mathbf{G}_{\text{LDPC}}$ be the S x K generator matrix of the LDPC symbols. So,

$$\mathbf{G}_{\text{LDPC}} \cdot (C[0], \ldots, C[K-1])^T = (C[K], \ldots, C[K+S-1])^T$$

$\mathbf{G}_{\text{Half}}$ be the H x (K+S) generator matrix of the Half symbols, So,

$$\mathbf{G}_{\text{Half}} \cdot (C[0], \ldots, C[S+K-1])^T = (C[K+S], \ldots, C[K+S+H-1])^T$$

$\mathbf{I}_S$ be the S x S identity matrix

$\mathbf{I}_H$ be the H x H identity matrix

$\mathbf{0}_{SxH}$ be the S x H zero matrix

$\mathbf{G}_{\text{LT}}$ be the KxL generator matrix of the encoding symbols generated by the LT Encoder.

So,

$$\mathbf{G}_{\text{LT}} \cdot (C[0], \ldots, C[L-1])^T = (C'[0], C'[1],\ldots, C'[K-1])^T$$

i.e. $\mathbf{G}_{\text{LT}i,j} = 1$ if and only if $C[j]$ is included in the symbols which are XORed to produce LTEnc[K, ($C[0]$, …, $C[L-1]$), ($d[i]$, $a[i]$, $b[i]$)].

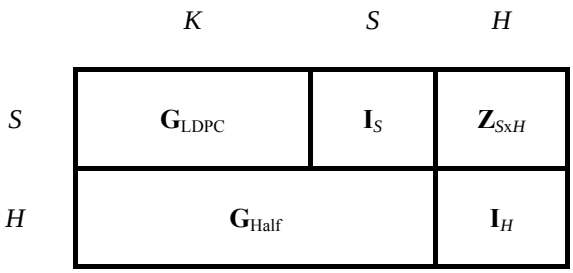
Then:

The first S rows of **A** are equal to $\mathbf{G}_{\text{LDPC}} \mid \mathbf{I}_S \mid \mathbf{Z}_{SxH}$.

The next H rows of **A** are equal to $\mathbf{G}_{\text{Half}} \mid \mathbf{I}_H$.

The remaining K rows of **A** are equal to $\mathbf{G}_{\text{LT}}$.

The matrix **A** is depicted in the figure below:



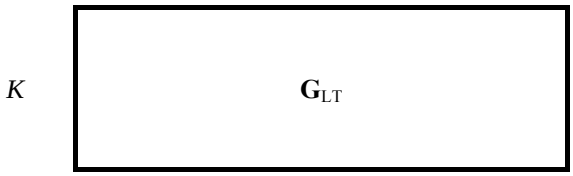


Figure C.5.2.4.2-1: The matrix A.

The intermediate symbols can then be calculated as:

$\mathbf{C} = \mathbf{A}^{-1} \cdot \mathbf{D}$

The source triples are generated such that for any K matrix **A** has full rank and is therefore invertible. This calculation can be realized by applying a Raptor decoding process to the K source symbols $C'[0], C'[1],\ldots, C'[K-1]$ to produce the L intermediate symbols $C[0], C[1],\ldots, C[L-1]$.

To efficiently generate the intermediate symbols from the source symbols, it is recommended that an efficient decoder implementation such as that described in Section C.8 be used. The source symbol triples are designed to facilitate efficient decoding of the source symbols using that algorithm.

C.4.3. Second encoding step: LT encoding

In the second encoding step, the repair symbol with ESI X is generated by applying the generator LTEnc[K, ($C[0]$, $C[1],\ldots, C[L-1]$), (d, a, b)] defined in Section C.5.4 to the L intermediate symbols $C[0], C[1],\ldots, C[L-1]$ using the triple (d, a, b)=Trip[K,X] generated according to Sections C.3.2.2 and C.4.2.

C.4.4. Generators

C.4.4.1 Random Generator

The random number generator Rand[X, i, m] is defined as follows, where X is a non-negative integer, i is a non-negative integer and m is a positive integer and the value produced is an integer between 0 and m-1. Let V_0 and V_1 be arrays of 256 entries each, where each entry is a 4-byte unsigned integer. These arrays are provided in Section C.7.

Then,

$\text{Rand}[X, i, m] = (V_0[(X + i) \% 256] \wedge V_1[(\text{floor}(X/256) + i) \% 256]) \% m$

C.4.4.2 Degree Generator

The degree generator Deg[v] is defined as follows, where v is an integer that is at least 0 and less than $2^{20} = 1048576$.

In Table C.4.4.2-1, find the index j such that $f[j-1] \le v < f[j]$

Deg[v] = $d[j]$

Table C.4.4.2-1: Defines the degree distribution for encoding symbols.

Index j	$f[j]$	$d[j]$
0	0	--
1	10241	1
2	491582	2
3	712794	3
4	831695	4
5	948446	10
6	1032189	11
7	1048576	40

C.4.4.3 LT Encoding Symbol Generator

The encoding symbol generator LTEnc[K, (C[0], C[1],…, C[L-1]), (d, a, b)] takes the following inputs:

K is the number of source symbols (or sub-symbols) for the source block (sub-block). Let L be derived from K as described in Section C.5.2, and let L' be the smallest prime integer greater than or equal to L.

(C[0], C[1],…, C[L-1]) is the array of L intermediate symbols (sub-symbols) generated as described in Section C.5.2

(d, a, b) is a source triple determined using the Triple generator defined in Section C.5.3.4, whereby

d is an integer denoting an encoding symbol degree

a is an integer between 1 and L'-1 inclusive

b is an integer between 0 and L'-1 inclusive

The encoding symbol generator produces a single encoding symbol as output, according to the following algorithm:

While ($b \geq L$) do $b = (b + a)$ % L'

LTEnc[K, (C[0], C[1],…, C[L-1]), (d, a, b)] = C[b].

For j = 1,…,min(d-1,L-1) do

$b = (b + a)$ % L'

While ($b \geq L$) do $b = (b + a)$ % L'

LTEnc[K, (C[0], C[1],…, C[L-1]), (d, a, b)] = *LTEnc*[K, (C[0], C[1],…, C[L-1]), (d, a, b)] ^ C[b]

C.4.4.4 Triple generator

The triple generator Trip[K,X] takes the following inputs:

K The number of source symbols

X An encoding symbol ID

Let

L be determined from K as described in Section C.5.2

L' be the smallest prime that is greater than or equal to L

Q = 65521, the largest prime smaller than 2^{16}.

$J(K)$ be the systematic index associated with K, as defined in Section C.7

The output of the triple generator is a triples, (d, a, b) determined as follows:

1. $A = (53591 + J(K)\cdot 997)$ % Q
2. $B = 10267\cdot(J(K)+1)$ % Q
3. $Y = (B + X\cdot A)$ % Q
4. v = Rand[Y, 0, 2^{20}]
5. d = Deg[v]
6. a = 1 + Rand[Y, 1, L'-1]
7. b = Rand[Y, 2, L']

C.5 Systematic Indices *J*(*K*)

For each value of *K* the systematic index *J*(*K*) is designed to have the property that the set of source symbol triples (*d*[0], *a*[0], *b*[0]), …, (*d*[*L*-1], *a*[*L*-1], *b*[*L*-1]) are such that the *L* intermediate symbols are uniquely defined, i.e. the matrix **A** in Section C.5.2.4.2 has full rank and is therefore invertible.

The following is the list of the systematic indices for values of *K* between 4 and 8192 inclusive,

18, 14, 61, 46, 14, 22, 20, 40, 48, 1, 29, 40, 43, 46, 18, 8, 20, 2, 61, 26, 13, 29, 36, 19, 58, 5, 58, 0, 54, 56, 24, 14, 5, 67, 39, 31, 25, 29, 24, 19, 14, 56, 49, 49, 63, 30, 4, 39, 2, 1, 20, 19, 61, 4, 54, 70, 25, 52, 9, 26, 55, 69, 27, 68, 75, 19, 64, 57, 45, 3, 37, 31, 100, 41, 25, 41, 53, 23, 9, 31, 26, 30, 30, 46,
90, 50, 13, 90, 77, 61, 31, 54, 54, 3, 21, 66, 21, 11, 23, 11, 29, 21, 7, 1, 27, 4, 34, 17, 85, 69, 17, 75, 93, 57, 0, 53, 71, 88, 119, 88, 90, 22, 0, 58, 41, 22, 96, 26, 79, 118, 19, 3, 81, 72, 50, 0, 32, 79, 28, 25, 12, 25, 29, 3, 37, 30, 30, 41, 84, 32, 31, 61, 32, 61, 7, 56, 54, 39, 33, 66, 29, 3, 14, 75, 75, 78, 84,
75, 84, 25, 54, 25, 25, 107, 78, 27, 73, 0, 49, 96, 53, 50, 21, 10, 73, 58, 65, 27, 3, 27, 18, 54, 45, 69, 29, 3, 65, 31, 71, 76, 56, 54, 76, 54, 13, 5, 18, 142, 17, 3, 37, 114, 41, 25, 56, 0, 23, 3, 41, 22, 22, 31, 18, 48, 31, 58, 37, 75, 88, 3, 56, 1, 95, 19, 73, 52, 52, 4, 75, 26, 1, 25, 10, 1, 70, 31, 31, 12, 10, 54,
46, 11, 74, 84, 74, 8, 58, 23, 74, 8, 36, 11, 16, 94, 76, 14, 57, 65, 8, 22, 10, 36, 36, 96, 62, 103, 6, 75, 103, 58, 10, 15, 41, 75, 125, 58, 15, 10, 34, 29, 34, 4, 16, 29, 18, 18, 28, 71, 28, 43, 77, 18, 41, 41, 41, 62, 29, 96, 15, 106, 43, 15, 3, 43, 61, 3, 18, 103, 77, 29, 103, 19, 58, 84, 58, 1, 146, 32, 3, 70, 52,
54, 29, 70, 69, 124, 62, 1, 26, 38, 26, 3, 16, 26, 5, 51, 120, 41, 16, 1, 43, 34, 34, 29, 37, 56, 29, 96, 86, 54, 25, 84, 50, 34, 34, 93, 84, 96, 29, 29, 50, 50, 6, 1, 105, 78, 15, 37, 19, 50, 71, 36, 6, 54, 8, 28, 54, 75, 75, 16, 75, 131, 5, 25, 16, 69, 17, 69, 6, 96, 53, 96, 41, 119, 6, 6, 88, 50, 88, 52, 37, 0, 124,
73, 73, 7, 14, 36, 69, 79, 6, 114, 40, 79, 17, 77, 24, 44, 37, 69, 27, 37, 29, 33, 37, 50, 31, 69, 29, 101, 7, 61, 45, 17, 73, 37, 34, 18, 94, 22, 22, 63, 3, 25, 25, 17, 3, 90, 34, 34, 41, 34, 41, 54, 41, 54, 41, 41, 41, 163, 143, 96, 18, 32, 39, 86, 104, 11, 17, 17, 11, 86, 104, 78, 70, 52, 78, 17, 73, 91, 62, 7, 128,
50, 124, 18, 101, 46, 10, 75, 104, 73, 58, 132, 34, 13, 4, 95, 88, 33, 76, 74, 54, 62, 113, 114, 103, 32, 103, 69, 54, 53, 3, 11, 72, 31, 53, 102, 37, 53, 11, 81, 41, 10, 164, 10, 41, 31, 36, 113, 82, 3, 125, 62, 16, 4, 41, 41, 4, 128, 49, 138, 128, 74, 103, 0, 6, 101, 41, 142, 171, 39, 105, 121, 81, 62, 41, 81,
37, 3, 81, 69, 62, 3, 69, 70, 21, 29, 4, 91, 87, 37, 79, 36, 21, 71, 37, 41, 75, 128, 128, 15, 25, 3, 108, 73, 91, 62, 114, 62, 62, 36, 36, 15, 58, 114, 61, 114, 58, 105, 114, 41, 61, 176, 145, 46, 37, 30, 220, 77, 138, 15, 1, 128, 53, 50, 50, 58, 8, 91, 114, 105, 63, 91, 37, 37, 13, 169, 51, 102, 6, 102, 23, 105,
23, 58, 6, 29, 29, 19, 82, 29, 13, 36, 27, 29, 61, 12, 18, 127, 127, 12, 44, 102, 18, 4, 15, 206, 53, 127, 53, 17, 69, 69, 69, 29, 29, 109, 25, 102, 25, 53, 62, 99, 62, 62, 29, 62, 62, 45, 91, 125, 29, 29, 29, 4, 117, 72, 4, 30, 71, 71, 95, 79, 179, 71, 30, 53, 32, 32, 49, 25, 91, 25, 26, 26, 103, 123, 26, 41, 162,
78, 52, 103, 25, 6, 142, 94, 45, 45, 94, 127, 94, 94, 94, 47, 209, 138, 39, 39, 19, 154, 73, 67, 91, 27, 91, 84, 4, 84, 91, 12, 14, 165, 142, 54, 69, 192, 157, 185, 8, 95, 25, 62, 103, 103, 95, 71, 97, 62, 128, 0, 29, 51, 16, 94, 16, 16, 51, 0, 29, 85, 10, 105, 16, 29, 29, 13, 29, 4, 4, 132, 23, 95, 25, 54, 41, 29,
50, 70, 58, 142, 72, 70, 15, 72, 54, 29, 22, 145, 29, 127, 29, 85, 58, 101, 34, 165, 91, 46, 46, 25, 185, 25, 77, 128, 46, 128, 46, 188, 114, 46, 25, 45, 45, 114, 145, 114, 15, 102, 142, 8, 73, 31, 139, 157, 13, 79, 13, 114, 150, 8, 90, 91, 123, 69, 82, 132, 8, 18, 10, 102, 103, 114, 103, 8, 103, 13, 115, 55,
62, 3, 8, 154, 114, 99, 19, 8, 31, 73, 19, 99, 10, 6, 121, 32, 13, 32, 119, 32, 29, 145, 30, 13, 13, 114, 145, 32, 1, 123, 39, 29, 31, 69, 31, 140, 72, 72, 25, 25, 123, 25, 123, 8, 4, 85, 8, 25, 39, 25, 39, 85, 138, 25, 138, 25, 33, 102, 70, 25, 25, 31, 25, 25, 192, 69, 69, 114, 145, 120, 120, 8, 33, 98, 15, 212,
155, 8, 101, 8, 8, 98, 68, 155, 102, 132, 120, 30, 25, 123, 123, 101, 25, 123, 32, 24, 94, 145, 32, 24, 94, 118, 145, 101, 53, 53, 25, 128, 173, 142, 81, 81, 69, 33, 33, 125, 4, 1, 17, 27, 4, 17, 102, 27, 13, 25, 128, 71, 13, 39, 53, 13, 53, 47, 39, 23, 128, 53, 39, 47, 39, 135, 158, 136, 36, 36, 27, 157, 47, 76,
213, 47, 156, 25, 25, 53, 25, 53, 25, 86, 27, 159, 25, 62, 79, 39, 79, 25, 145, 49, 25, 143, 13, 114, 150, 130, 94, 102, 39, 4, 39, 61, 77, 228, 22, 25, 47, 119, 205, 122, 119, 205, 119, 22, 119, 258, 143, 22, 81, 179, 22, 22, 143, 25, 65, 53, 168, 36, 79, 175, 37, 79, 70, 79, 103, 70, 25, 175, 4, 96, 96, 49,
128, 138, 96, 22, 62, 47, 95, 105, 95, 62, 95, 62, 142, 103, 69, 103, 30, 103, 34, 173, 127, 70, 127, 132, 18, 85, 22, 71, 18, 206, 206, 18, 128, 145, 70, 193, 188, 8, 125, 114, 70, 128, 114, 145, 102, 25, 12, 108, 102, 94, 10, 102, 1, 102, 124, 22, 22, 118, 132, 22, 116, 75, 41, 63, 41, 189, 208, 55, 85, 69,
8, 71, 53, 71, 69, 102, 165, 41, 99, 69, 33, 33, 29, 156, 102, 13, 251, 102, 25, 13, 109, 102, 164, 102, 164, 102, 25, 29, 228, 29, 259, 179, 222, 95, 94, 30, 30, 30, 142, 55, 142, 72, 55, 102, 128, 17, 69, 164, 165, 3, 164, 36, 165, 27, 27, 45, 21, 21, 237, 113, 83, 231, 106, 13, 154, 13, 154, 128, 154, 148,
258, 25, 154, 128, 3, 27, 10, 145, 145, 21, 146, 25, 1, 185, 121, 0, 1, 95, 55, 95, 95, 30, 0, 27, 95, 0, 95, 8, 222, 27, 121, 30, 95, 121, 0, 98, 94, 131, 55, 95, 95, 30, 98, 30, 0, 91, 145, 66, 179, 66, 58, 175, 29, 0, 31, 173, 146, 160, 39, 53, 28, 123, 199, 123, 175, 146, 156, 54, 54, 149, 25, 70, 178, 128,
25, 70, 70, 94, 224, 54, 4, 54, 54, 25, 228, 160, 206, 165, 143, 206, 108, 220, 234, 160, 13, 169, 103, 103, 103, 91, 213, 222, 91, 103, 91, 103, 31, 30, 123, 13, 62, 103, 50, 106, 42, 13, 145, 114, 220, 65, 8, 8, 175, 11, 104, 94, 118, 132, 27, 118, 193, 27, 128, 127, 127, 183, 33, 30, 29, 103, 128, 61,
234, 165, 41, 29, 193, 33, 207, 41, 165, 165, 55, 81, 157, 157, 8, 81, 11, 27, 8, 8, 98, 96, 142, 145, 41, 179, 112, 62, 180, 206, 206, 165, 39, 241, 45, 151, 26, 197, 102, 192, 125, 128, 67, 128, 69, 128, 197, 33, 125, 102, 13, 103, 25, 30, 12, 30, 12, 30, 25, 77, 12, 25, 180, 27, 10, 69, 235, 228, 343, 118,
69, 41, 8, 69, 175, 25, 69, 25, 125, 41, 25, 41, 8, 155, 146, 155, 146, 155, 206, 168, 128, 157, 27, 273, 211, 211, 168, 11, 173, 154, 77, 173, 77, 102, 102, 102, 8, 85, 95, 102, 157, 28, 122, 234, 122, 157, 235, 222, 241, 10, 91, 179, 25, 13, 25, 41, 25, 206, 41, 6, 41, 158, 206, 206, 33, 296, 296, 33, 228,
69, 8, 114, 148, 33, 29, 66, 27, 27, 30, 233, 54, 173, 108, 106, 108, 108, 53, 103, 33, 33, 33, 176, 27, 27, 205, 164, 105, 237, 41, 27, 72, 165, 29, 29, 259, 132, 132, 132, 364, 71, 71, 27, 94, 160, 127, 51, 234, 55, 27, 95, 94, 165, 55, 55, 41, 0, 41, 128, 4, 123, 173, 6, 164, 157, 121, 121, 154, 86, 164,
164, 25, 93, 164, 25, 164, 210, 284, 62, 93, 30, 25, 25, 30, 30, 260, 130, 25, 125, 57, 53, 166, 166, 166, 185, 166, 158, 94, 113, 215, 159, 62, 99, 21, 172, 99, 184, 62, 259, 4, 21, 21, 77, 62, 173, 41, 146, 6, 41, 128, 121, 41, 11, 121, 103, 159, 164, 175, 206, 91, 103, 164, 72, 25, 129, 72, 206, 129, 33,
103, 102, 102, 29, 13, 11, 251, 234, 135, 31, 8, 123, 65, 91, 121, 129, 65, 243, 10, 91, 8, 65, 70, 228, 220, 243, 91, 10, 10, 30, 178, 91, 178, 33, 21, 25, 235, 165, 11, 161, 158, 27, 27, 30, 128, 75, 36, 30, 36, 36, 173, 25, 33, 178, 112, 162, 112, 112, 112, 162, 33, 33, 178, 123, 123, 39, 106, 91, 106,
106, 158, 106, 106, 284, 39, 230, 21, 228, 11, 21, 228, 159, 241, 62, 10, 62, 10, 68, 234, 39, 39, 138, 62, 22, 27, 183, 22, 215, 10, 175, 175, 353, 228, 42, 193, 175, 175, 27, 98, 27, 193, 150, 27, 173, 17, 233, 233, 25, 102, 123, 152, 242, 108, 4, 94, 176, 13, 41, 219, 17, 151, 22, 103, 103, 53, 128, 233,
284, 25, 265, 128, 39, 39, 138, 42, 39, 21, 86, 95, 127, 29, 91, 46, 103, 103, 215, 25, 123, 123, 230, 25, 193, 180, 30, 60, 30, 242, 136, 180, 193, 30, 206, 180, 60, 165, 206, 193, 165, 123, 164, 103, 68, 25, 70, 91, 25, 82, 53, 82, 186, 53, 82, 53, 25, 30, 282, 91, 13, 234, 160, 160, 126, 149, 36, 36, 160,
149, 178, 160, 39, 294, 149, 149, 160, 39, 95, 221, 186, 106, 178, 316, 267, 53, 53, 164, 159, 164, 165, 94, 228, 53, 52, 178, 183, 53, 294, 128, 55, 140, 294, 25, 95, 366, 15, 304, 13, 183, 77, 230, 6, 136, 235, 121, 311, 273, 36, 158, 235, 230, 98, 201, 165, 165, 165, 91, 175, 248, 39, 185, 128, 39, 39,
128, 313, 91, 36, 219, 130, 25, 130, 234, 234, 130, 234, 121, 205, 304, 94, 77, 64, 259, 60, 60, 60, 77, 242, 60, 145, 95, 270, 18, 91, 199, 159, 91, 235, 58, 249, 26, 123, 114, 29, 15, 191, 15, 30, 55, 55, 347, 4, 29, 15, 4, 341, 93, 7, 30, 23, 7, 121, 266, 178, 261, 70, 169, 25, 25, 158, 169, 25, 169, 270,
270, 13, 128, 327, 103, 55, 128, 103, 136, 159, 103, 327, 41, 32, 111, 111, 114, 173, 215, 173, 25, 173, 180, 114, 173, 173, 98, 93, 25, 160, 157, 159, 160, 159, 159, 160, 320, 35, 193, 221, 33, 36, 136, 248, 91, 215, 125, 215, 156, 68, 125, 125, 1, 287, 123, 94, 30, 184, 13, 30, 94, 123, 206, 12, 206,
289, 128, 122, 184, 128, 289, 178, 29, 26, 206, 178, 65, 206, 128, 192, 102, 197, 36, 94, 94, 155, 10, 36, 121, 280, 121, 368, 192, 121, 121, 179, 121, 36, 54, 192, 121, 192, 197, 118, 123, 224, 118, 10, 192, 10, 91, 269, 91, 49, 206, 184, 185, 62, 8, 49, 289, 30, 5, 55, 30, 42, 39, 220, 298, 42, 347, 42,
234, 42, 70, 42, 55, 321, 129, 172, 173, 172, 13, 98, 129, 325, 235, 284, 362, 129, 233, 345, 175, 261, 175, 60, 261, 58, 289, 99, 99, 99, 206, 99, 36, 175, 29, 25, 432, 125, 264, 168, 173, 69, 158, 273, 179, 164, 69, 158, 69, 8, 95, 192, 30, 164, 101, 44, 53, 273, 335, 273, 53, 45, 128, 45, 234, 123, 105,
103, 103, 224, 36, 90, 211, 282, 264, 91, 228, 91, 166, 264, 228, 398, 50, 101, 91, 264, 73, 36, 25, 73, 50, 50, 242, 36, 36, 58, 165, 204, 353, 165, 125, 320, 128, 298, 298, 180, 128, 60, 102, 30, 30, 53, 179, 234, 325, 234, 175, 21, 250, 215, 103, 21, 21, 250, 91, 211, 91, 313, 301, 323, 215, 228, 160,
29, 29, 81, 53, 180, 146, 248, 66, 159, 39, 98, 323, 98, 36, 95, 218, 234, 39, 82, 82, 230, 62, 13, 62, 230, 13, 30, 98, 0, 8, 98, 8, 98, 91, 267, 121, 197, 30, 78, 27, 78, 102, 27, 298, 160, 103, 264, 264, 264, 175, 17, 273, 273, 165, 31, 160, 17, 99, 17, 99, 234, 31, 17, 99, 36, 26, 128, 29, 214, 353, 264,
102, 36, 102, 264, 264, 273, 273, 4, 16, 138, 138, 264, 128, 313, 25, 420, 60, 10, 280, 264, 60, 60, 103, 178, 125, 178, 29, 327, 29, 36, 30, 36, 4, 52, 183, 183, 173, 52, 31, 173, 31, 158, 31, 158, 31, 9, 31, 31, 353, 31, 353, 173, 415, 9, 17, 222, 31, 103, 31, 165, 27, 31, 31, 165, 27, 27, 206, 31, 31, 4, 4,
30, 4, 4, 264, 185, 159, 310, 273, 310, 173, 40, 4, 173, 4, 173, 4, 250, 250, 62, 188, 119, 250, 233, 62, 121, 105, 105, 54, 103, 111, 291, 236, 236, 103, 297, 36, 26, 316, 69, 183, 158, 206, 129, 160, 129, 184, 55, 179, 279, 11, 179, 347, 160, 184, 129, 179, 351, 179, 353, 179, 129, 129, 351, 11, 111,
93, 93, 235, 103, 173, 53, 93, 50, 111, 86, 123, 94, 36, 183, 60, 55, 55, 178, 219, 253, 321, 178, 235, 235, 183, 183, 204, 321, 219, 160, 193, 335, 121, 70, 69, 295, 159, 297, 231, 121, 231, 136, 353, 136, 121, 279, 215, 366, 215, 353, 159, 353, 353, 103, 31, 31, 298, 298, 30, 30, 165, 273, 25, 219, 35,
165, 259, 54, 36, 54, 54, 165, 71, 250, 327, 13, 289, 165, 196, 165, 165, 94, 233, 165, 94, 60, 165, 96, 220, 166, 271, 158, 397, 122, 53, 53, 137, 280, 272, 62, 30, 30, 30, 105, 102, 67, 140, 8, 67, 21, 270, 298, 69, 173, 298, 91, 179, 327, 86, 179, 88, 179, 179, 55, 123, 220, 233, 94, 94, 175, 13, 53, 13,
154, 191, 74, 83, 83, 325, 207, 83, 74, 83, 325, 74, 316, 388, 55, 55, 364, 55, 183, 434, 273, 273, 273, 164, 213, 11, 213, 327, 321, 21, 352, 185, 103, 13, 13, 55, 30, 323, 123, 178, 435, 178, 30, 175, 175, 30, 481, 527, 175, 125, 232, 306, 232, 206, 306, 364, 206, 270, 206, 232, 10, 30, 130, 160, 130,
347, 240, 30, 136, 130, 347, 136, 279, 298, 206, 30, 103, 273, 241, 70, 206, 306, 434, 206, 94, 94, 156, 161, 321, 321, 64, 161, 13, 183, 183, 83, 161, 13, 169, 13, 159, 36, 173, 159, 36, 36, 230, 235, 235, 159, 159, 335, 312, 42, 342, 264, 39, 39, 39, 34, 298, 36, 36, 252, 164, 29, 493, 29, 387, 387,
435, 493, 132, 273, 105, 132, 74, 73, 206, 234, 273, 206, 95, 15, 280, 280, 280, 280, 397, 273, 273, 242, 397, 280, 397, 397, 397, 273, 397, 280, 230, 137, 353, 67, 81, 137, 137, 353, 259, 312, 114, 164, 164, 25, 77, 21, 77, 165, 30, 30, 231, 234, 121, 234, 312, 121, 364, 136, 123, 123, 136, 123, 136,
150, 264, 285, 30, 166, 93, 30, 39, 224, 136, 39, 355, 355, 397, 67, 67, 25, 67, 25, 298, 11, 67, 264, 374, 99, 150, 321, 67, 70, 67, 295, 150, 29, 321, 150, 70, 29, 142, 355, 311, 173, 13, 253, 103, 114, 114, 70, 192, 22, 128, 128, 183, 184, 70, 77, 215, 102, 292, 30, 123, 279, 292, 142, 33, 215, 102,
468, 123, 468, 473, 30, 292, 215, 30, 213, 443, 473, 215, 234, 279, 279, 279, 279, 265, 443, 206, 66, 313, 34, 30, 206, 30, 51, 15, 206, 41, 434, 41, 398, 67, 30, 301, 67, 36, 3, 285, 437, 136, 136, 22, 136, 145, 365, 323, 323, 145, 136, 22, 453, 99, 323, 353, 9, 258, 323, 231, 128, 231, 382, 150, 420,
39, 94, 29, 29, 353, 22, 22, 347, 353, 39, 29, 22, 183, 8, 284, 355, 388, 284, 60, 64, 99, 60, 64, 150, 95, 150, 364, 150, 95, 150, 6, 236, 383, 544, 81, 206, 388, 206, 58, 159, 99, 231, 228, 363, 363, 121, 99, 121, 121, 99, 422, 544, 273, 173, 121, 427, 102, 121, 235, 284, 179, 25, 197, 25, 179, 511, 70,
368, 70, 25, 388, 123, 368, 159, 213, 410, 159, 236, 127, 159, 21, 373, 184, 424, 327, 250, 176, 176, 175, 284, 316, 176, 284, 327, 111, 250, 284, 175, 175, 264, 111, 176, 219, 111, 427, 427, 176, 284, 427, 353, 428, 55, 184, 493, 158, 136, 99, 287, 264, 334, 264, 213, 213, 292, 481, 93, 264, 292,
295, 295, 6, 367, 279, 173, 308, 285, 158, 308, 335, 299, 137, 137, 572, 41, 137, 137, 41, 94, 335, 220, 36, 224, 420, 36, 265, 265, 91, 91, 71, 123, 264, 91, 91, 123, 107, 30, 22, 292, 35, 241, 356, 298, 14, 298, 441, 35, 121, 71, 63, 130, 63, 488, 363, 71, 63, 307, 194, 71, 71, 220, 121, 125, 71, 220,
71, 71, 71, 71, 235, 265, 353, 128, 155, 128, 420, 400, 130, 173, 183, 183, 184, 130, 173, 183, 13, 183, 130, 130, 183, 183, 353, 353, 183, 242, 183, 183, 306, 324, 324, 321, 306, 321, 6, 6, 128, 306, 242, 242, 306, 183, 183, 6, 183, 321, 486, 183, 164, 30, 78, 138, 158, 138, 34, 206, 362, 55, 70, 67,
21, 375, 136, 298, 81, 298, 298, 298, 230, 121, 30, 230, 311, 240, 311, 311, 158, 204, 136, 136, 184, 136, 264, 311, 311, 312, 312, 72, 311, 175, 264, 91, 175, 264, 121, 461, 312, 312, 238, 475, 350, 512, 350, 312, 313, 350, 312, 366, 294, 30, 253, 253, 253, 388, 158, 388, 22, 388, 22, 388, 103, 321,
321, 253, 7, 437, 103, 114, 242, 114, 114, 242, 114, 114, 242, 242, 242, 306, 242, 114, 7, 353, 335, 27, 241, 299, 312, 364, 506, 409, 94, 462, 230, 462, 243, 230, 175, 175, 462, 461, 230, 428, 426, 175, 175, 165, 175, 175, 372, 183, 572, 102, 85, 102, 538, 206, 376, 85, 85, 284, 85, 85, 284, 398, 83,
160, 265, 308, 398, 310, 583, 289, 279, 273, 285, 490, 490, 211, 292, 292, 158, 398, 30, 220, 169, 368, 368, 368, 169, 159, 368, 93, 368, 368, 93, 169, 368, 368, 443, 368, 298, 443, 368, 298, 538, 345, 345, 311, 178, 54, 311, 215, 178, 175, 222, 264, 475, 264, 264, 475, 478, 289, 63, 236, 63, 299,
231, 296, 397, 299, 158, 36, 164, 164, 21, 492, 21, 164, 21, 164, 403, 26, 26, 588, 179, 234, 169, 465, 295, 67, 41, 353, 295, 538, 161, 185, 306, 323, 68, 420, 323, 82, 241, 241, 36, 53, 493, 301, 292, 241, 250, 63, 63, 103, 442, 353, 185, 353, 321, 353, 185, 353, 353, 185, 409, 353, 589, 34, 271, 271,
34, 86, 34, 34, 353, 353, 39, 414, 4, 95, 95, 4, 225, 95, 4, 121, 30, 552, 136, 159, 159, 514, 159, 159, 54, 514, 206, 136, 206, 159, 74, 235, 235, 312, 54, 312, 42, 156, 422, 629, 54, 465, 265, 165, 250, 35, 165, 175, 659, 175, 175, 8, 8, 8, 8, 206, 206, 206, 50, 435, 206, 432, 230, 230, 234, 230, 94, 299,
299, 285, 184, 41, 93, 299, 299, 285, 41, 285, 158, 285, 206, 299, 41, 36, 396, 364, 364, 120, 396, 514, 91, 382, 538, 807, 717, 22, 93, 412, 54, 215, 54, 298, 308, 148, 298, 148, 298, 308, 102, 656, 6, 148, 745, 128, 298, 64, 407, 273, 41, 172, 64, 234, 250, 398, 181, 445, 95, 236, 441, 477, 504, 102,
196, 137, 364, 60, 453, 137, 364, 367, 334, 364, 299, 196, 397, 630, 589, 589, 196, 646, 337, 235, 128, 128, 343, 289, 235, 324, 427, 324, 58, 215, 215, 461, 425, 461, 387, 440, 285, 440, 440, 285, 387, 632, 325, 325, 440, 461, 425, 425, 387, 627, 191, 285, 440, 308, 55, 219, 280, 308, 265, 538, 183,
121, 30, 236, 206, 30, 455, 236, 30, 30, 705, 83, 228, 280, 468, 132, 8, 132, 132, 128, 409, 173, 353, 132, 409, 35, 128, 450, 137, 398, 67, 432, 423, 235, 235, 388, 306, 93, 93, 452, 300, 190, 13, 452, 388, 30, 452, 13, 30, 13, 30, 306, 362, 234, 721, 635, 809, 784, 67, 498, 498, 67, 353, 635, 67, 183,
159, 445, 285, 183, 53, 183, 445, 265, 432, 57, 420, 432, 420, 477, 327, 55, 60, 105, 183, 218, 104, 104, 475, 239, 582, 151, 239, 104, 732, 41, 26, 784, 86, 300, 215, 36, 64, 86, 86, 675, 294, 64, 86, 528, 550, 493, 565, 298, 230, 312, 295, 538, 298, 295, 230, 54, 374, 516, 441, 54, 54, 323, 401, 401,
382, 159, 837, 159, 54, 401, 592, 159, 401, 417, 610, 264, 150, 323, 452, 185, 323, 323, 185, 403, 185, 423, 165, 425, 219, 407, 270, 231, 99, 93, 231, 631, 756, 71, 364, 434, 213, 86, 102, 434, 102, 86, 23, 71, 335, 164, 323, 409, 381, 4, 124, 41, 424, 206, 41, 124, 41, 41, 703, 635, 124, 493, 41, 41,
487, 492, 124, 175, 124, 261, 600, 488, 261, 488, 261, 206, 677, 261, 308, 723, 908, 704, 691, 723, 488, 488, 441, 136, 476, 312, 136, 550, 572, 728, 550, 22, 312, 312, 22, 55, 413, 183, 280, 593, 191, 36, 36, 427, 36, 695, 592, 19, 544, 13, 468, 13, 544, 72, 437, 321, 266, 461, 266, 441, 230, 409, 93,
521, 521, 345, 235, 22, 142, 150, 102, 569, 235, 264, 91, 521, 264, 7, 102, 7, 498, 521, 235, 537, 235, 6, 241, 420, 420, 631, 41, 527, 103, 67, 337, 62, 264, 527, 131, 67, 174, 263, 264, 36, 36, 263, 581, 253, 465, 160, 286, 91, 160, 55, 4, 4, 631, 631, 608, 365, 465, 294, 427, 427, 335, 669, 669, 129,
93, 93, 93, 93, 74, 66, 758, 504, 347, 130, 505, 504, 143, 505, 550, 222, 13, 352, 529, 291, 538, 50, 68, 269, 130, 295, 130, 511, 295, 295, 130, 486, 132, 61, 206, 185, 368, 669, 22, 175, 492, 207, 373, 452, 432, 327, 89, 550, 496, 611, 527, 89, 527, 496, 550, 516, 516, 91, 136, 538, 264, 264, 124,
264, 264, 264, 264, 264, 535, 264, 150, 285, 398, 285, 582, 398, 475, 81, 694, 694, 64, 81, 694, 234, 607, 723, 513, 234, 64, 581, 64, 124, 64, 607, 234, 723, 717, 367, 64, 513, 607, 488, 183, 488, 450, 183, 550, 286, 183, 363, 286, 414, 67, 449, 449, 366, 215, 235, 95, 295, 295, 41, 335, 21, 445, 225,
21, 295, 372, 749, 461, 53, 481, 397, 427, 427, 427, 714, 481, 714, 427, 717, 165, 245, 486, 415, 245, 415, 486, 274, 415, 441, 456, 300, 548, 300, 422, 422, 757, 11, 74, 430, 430, 136, 409, 430, 749, 191, 819, 592, 136, 364, 465, 231, 231, 918, 160, 589, 160, 160, 465, 465, 231, 157, 538, 538, 259,
538, 326, 22, 22, 22, 179, 22, 22, 550, 179, 287, 287, 417, 327, 498, 498, 287, 488, 327, 538, 488, 583, 488, 287, 335, 287, 335, 287, 327, 41, 287, 335, 287, 327, 441, 335, 287, 488, 538, 327, 498, 8, 8, 374, 8, 64, 427, 8, 374, 417, 760, 409, 373, 160, 423, 206, 160, 106, 499, 160, 271, 235, 160, 590, 353,
695, 478, 619, 590, 353, 13, 63, 189, 420, 605, 427, 643, 121, 280, 415, 121, 415, 595, 417, 121, 398, 55, 330, 463, 463, 123, 353, 330, 582, 309, 582, 582, 409, 582, 582, 405, 330, 550, 405, 582, 353, 309, 308, 60, 353, 7, 60, 71, 353, 189, 183, 183, 183, 582, 755, 189, 437, 287, 189, 183, 668, 481, 384, 384, 481,
481, 481, 477, 582, 582, 499, 650, 481, 121, 461, 231, 36, 235, 36, 413, 235, 209, 36, 689, 114, 353, 353, 235, 592, 36, 353, 413, 209, 70, 308, 70, 699, 308, 70, 213, 292, 86, 689, 465, 55, 508, 128, 452, 29, 41, 681, 573, 352, 21, 21, 648, 648, 69, 509, 409, 21, 264, 21, 509, 514, 514, 409, 21, 264,
443, 443, 427, 160, 433, 663, 433, 231, 646, 185, 482, 646, 433, 13, 398, 172, 234, 42, 491, 172, 234, 234, 832, 775, 172, 196, 335, 822, 461, 298, 461, 364, 1120, 537, 169, 169, 364, 694, 219, 612, 231, 740, 42, 235, 321, 279, 960, 279, 353, 492, 159, 572, 321, 159, 287, 353, 287, 287, 206, 206,
321, 287, 159, 321, 492, 159, 55, 572, 600, 270, 492, 784, 173, 91, 91, 443, 443, 582, 261, 497, 572, 91, 555, 352, 206, 261, 555, 285, 91, 555, 497, 83, 91, 619, 353, 488, 112, 4, 592, 295, 295, 488, 235, 231, 769, 568, 581, 671, 451, 451, 483, 299, 1011, 432, 422, 207, 106, 701, 508, 555, 508, 555,
125, 870, 555, 589, 508, 125, 749, 482, 125, 125, 130, 544, 643, 643, 544, 488, 22, 643, 130, 335, 544, 22, 130, 544, 544, 488, 426, 426, 4, 180, 4, 695, 35, 54, 433, 500, 592, 433, 262, 94, 401, 401, 106, 216, 216, 106, 521, 102, 462, 518, 271, 475, 365, 193, 648, 206, 424, 206, 193, 206, 206, 424,
299, 590, 590, 364, 621, 67, 538, 488, 567, 51, 51, 513, 194, 81, 488, 486, 289, 567, 563, 749, 563, 338, 338, 502, 563, 822, 338, 563, 338, 502, 201, 230, 201, 533, 445, 175, 201, 175, 13, 85, 960, 103, 85, 175, 30, 445, 445, 175, 573, 196, 877, 287, 356, 678, 235, 489, 312, 572, 264, 717, 138, 295,
6, 295, 523, 55, 165, 165, 295, 138, 663, 6, 295, 6, 353, 138, 6, 138, 169, 129, 784, 12, 129, 194, 605, 784, 445, 234, 627, 563, 689, 627, 647, 570, 627, 570, 647, 206, 234, 215, 234, 816, 627, 816, 234, 627, 215, 234, 627, 264, 427, 427, 30, 424, 161, 161, 916, 740, 180, 616, 481, 514, 383, 265, 481,
164, 650, 121, 582, 689, 420, 669, 589, 420, 788, 549, 165, 734, 280, 224, 146, 681, 788, 184, 398, 784, 4, 398, 417, 417, 398, 636, 784, 417, 81, 398, 417, 81, 185, 827, 420, 241, 420, 41, 185, 185, 718, 241, 101, 185, 185, 241, 241, 241, 241, 241, 185, 324, 420, 420, 1011, 420, 827, 241, 184, 563,
241, 183, 285, 529, 285, 808, 822, 891, 822, 488, 285, 486, 619, 55, 869, 39, 567, 39, 289, 203, 158, 289, 710, 818, 158, 818, 355, 29, 409, 203, 308, 648, 792, 308, 308, 91, 308, 6, 592, 792, 106, 106, 308, 41, 178, 91, 751, 91, 259, 734, 166, 36, 327, 166, 230, 205, 205, 172, 128, 230, 432, 623, 838,
623, 432, 278, 432, 42, 916, 432, 694, 623, 352, 452, 93, 314, 93, 93, 641, 88, 970, 914, 230, 61, 159, 270, 159, 493, 159, 755, 159, 409, 30, 30, 836, 128, 241, 99, 102, 984, 538, 102, 102, 273, 639, 838, 102, 102, 136, 637, 508, 627, 285, 465, 327, 327, 21, 749, 327, 749, 21, 845, 21, 21, 409, 749,
1367, 806, 616, 714, 253, 616, 714, 714, 112, 375, 21, 112, 375, 375, 51, 51, 51, 51, 393, 206, 870, 713, 193, 802, 21, 1061, 42, 382, 42, 543, 876, 42, 876, 382, 696, 543, 635, 490, 353, 353, 417, 64, 1257, 271, 64, 377, 127, 127, 537, 417, 905, 353, 538, 465, 605, 876, 427, 324, 514, 852, 427, 53,
427, 557, 173, 173, 7, 1274, 563, 31, 31, 31, 745, 392, 289, 230, 230, 230, 91, 218, 327, 420, 420, 128, 901, 552, 420, 230, 608, 552, 476, 347, 476, 231, 159, 137, 716, 648, 716, 627, 740, 718, 679, 679, 6, 718, 740, 6, 189, 679, 125, 159, 757, 1191, 409, 175, 250, 409, 67, 324, 681, 605, 550, 398,
550, 931, 478, 174, 21, 316, 91, 316, 654, 409, 425, 425, 699, 61, 699, 321, 698, 321, 698, 61, 425, 699, 321, 409, 699, 299, 335, 321, 335, 61, 698, 699, 654, 698, 299, 425, 231, 14, 121, 515, 121, 14, 165, 81, 409, 189, 81, 373, 465, 463, 1055, 507, 81, 81, 189, 1246, 321, 409, 886, 104, 842, 689,
300, 740, 380, 656, 656, 832, 656, 380, 300, 300, 206, 187, 175, 142, 465, 206, 271, 468, 215, 560, 83, 215, 83, 215, 215, 83, 175, 215, 83, 83, 111, 206, 756, 559, 756, 1367, 206, 559, 1015, 559, 559, 946, 1015, 548, 559, 756, 1043, 756, 698, 159, 414, 308, 458, 997, 663, 663, 347, 39, 755, 838,
323, 755, 323, 159, 159, 717, 159, 21, 41, 128, 516, 159, 717, 71, 870, 755, 159, 740, 717, 374, 516, 740, 51, 148, 335, 148, 335, 791, 120, 364, 335, 335, 51, 120, 251, 538, 251, 971, 1395, 538, 78, 178, 538, 538, 918, 129, 918, 129, 538, 538, 656, 129, 538, 538, 129, 538, 1051, 538, 128, 838, 931,
998, 823, 1095, 334, 870, 334, 367, 550, 1061, 498, 745, 832, 498, 745, 716, 498, 498, 128, 997, 832, 716, 832, 130, 642, 616, 497, 432, 432, 432, 432, 642, 159, 432, 46, 230, 788, 160, 230, 478, 46, 693, 103, 920, 230, 589, 643, 160, 616, 432, 165, 165, 583, 592, 838, 784, 583, 710, 6, 583, 583, 6,
35, 230, 838, 592, 710, 6, 589, 230, 838, 30, 592, 583, 6, 583, 6, 6, 583, 30, 30, 6, 375, 375, 99, 36, 1158, 425, 662, 417, 681, 364, 375, 1025, 538, 822, 669, 893, 538, 538, 450, 409, 632, 527, 632, 563, 632, 527, 550, 71, 698, 550, 39, 550, 514, 537, 514, 537, 111, 41, 173, 592, 173, 648, 173, 173,
173, 1011, 514, 173, 173, 514, 166, 648, 355, 161, 166, 648, 497, 327, 327, 550, 650, 21, 425, 605, 555, 103, 425, 605, 842, 836, 1011, 636, 138, 756, 836, 756, 756, 353, 1011, 636, 636, 1158, 741, 741, 842, 756, 741, 1011, 677, 1011, 770, 366, 306, 488, 920, 920, 665, 775, 502, 500, 775, 775, 648,
364, 833, 207, 13, 93, 500, 364, 500, 665, 500, 93, 295, 183, 1293, 313, 272, 313, 279, 303, 93, 516, 93, 1013, 381, 6, 93, 93, 303, 259, 643, 168, 673, 230, 1261, 230, 230, 673, 1060, 1079, 1079, 550, 741, 741, 590, 527, 741, 741, 442, 741, 442, 848, 741, 590, 925, 219, 527, 925, 335, 442, 590, 239,
590, 590, 590, 239, 527, 239, 1033, 230, 734, 241, 741, 230, 549, 548, 1015, 1015, 32, 36, 433, 465, 724, 465, 73, 73, 73, 465, 808, 73, 592, 1430, 250, 154, 154, 250, 538, 353, 353, 353, 353, 353, 175, 194, 206, 538, 632, 1163, 960, 175, 175, 538, 452, 632, 1163, 175, 538, 960, 194, 175, 194, 632,
960, 632, 94, 632, 461, 960, 1163, 1163, 461, 632, 960, 755, 707, 105, 382, 625, 382, 382, 784, 707, 871, 559, 387, 387, 871, 784, 559, 784, 88, 36, 570, 314, 1028, 975, 335, 335, 398, 573, 573, 573, 21, 215, 562, 738, 612, 424, 21, 103, 788, 870, 912, 23, 186, 757, 73, 818, 23, 73, 563, 952, 262,
563, 137, 262, 1022, 952, 137, 1273, 442, 952, 604, 137, 308, 384, 913, 235, 325, 695, 398, 95, 668, 776, 713, 309, 691, 22, 10, 364, 682, 682, 578, 481, 1252, 1072, 1252, 825, 578, 825, 1072, 1149, 592, 273, 387, 273, 427, 155, 1204, 50, 452, 50, 1142, 50, 367, 452, 1142, 611, 367, 50, 50, 367, 50,
1675, 99, 367, 50, 1501, 1099, 830, 681, 689, 917, 1089, 453, 425, 235, 918, 538, 550, 335, 161, 387, 859, 324, 21, 838, 859, 1123, 21, 723, 21, 335, 335, 206, 21, 364, 1426, 21, 838, 838, 335, 364, 21, 21, 859, 920, 838, 838, 397, 81, 639, 397, 397, 588, 933, 933, 784, 222, 830, 36, 36, 222, 1251,
266, 36, 146, 266, 366, 581, 605, 366, 22, 966, 681, 681, 433, 730, 1013, 550, 21, 21, 938, 488, 516, 21, 21, 656, 420, 323, 323, 323, 327, 323, 918, 581, 581, 830, 361, 830, 364, 259, 364, 496, 496, 364, 691, 705, 691, 475, 427, 1145, 600, 179, 427, 527, 749, 869, 689, 335, 347, 220, 298, 689, 1426,
183, 554, 55, 832, 550, 550, 165, 770, 957, 67, 1386, 219, 683, 683, 355, 683, 355, 355, 738, 355, 842, 931, 266, 325, 349, 256, 1113, 256, 423, 960, 554, 554, 325, 554, 508, 22, 142, 22, 508, 916, 767, 55, 1529, 767, 55, 1286, 93, 972, 550, 931, 1286, 1286, 972, 93, 1286, 1392, 890, 93, 1286, 93,
1286, 972, 374, 931, 890, 808, 779, 975, 975, 175, 173, 4, 681, 383, 1367, 173, 383, 1367, 383, 173, 175, 69, 238, 146, 238, 36, 148, 888, 238, 173, 238, 148, 238, 888, 185, 925, 925, 797, 925, 815, 925, 469, 784, 289, 784, 925, 797, 925, 925, 1093, 925, 925, 925, 1163, 797, 797, 815, 925, 1093,
784, 636, 663, 925, 187, 922, 316, 1380, 709, 916, 916, 187, 355, 948, 916, 187, 916, 916, 948, 948, 916, 355, 316, 316, 334, 300, 1461, 36, 583, 1179, 699, 235, 858, 583, 699, 858, 699, 1189, 1256, 1189, 699, 797, 699, 699, 699, 699, 427, 488, 427, 488, 175, 815, 656, 656, 150, 322, 465, 322, 870,
465, 1099, 582, 665, 767, 749, 635, 749, 600, 1448, 36, 502, 235, 502, 355, 502, 355, 355, 355, 172, 355, 355, 95, 866, 425, 393, 1165, 42, 42, 42, 393, 939, 909, 909, 836, 552, 424, 1333, 852, 897, 1426, 1333, 1446, 1426, 997, 1011, 852, 1198, 55, 32, 239, 588, 681, 681, 239, 1401, 32, 588, 239,
462, 286, 1260, 984, 1160, 960, 960, 486, 828, 462, 960, 1199, 581, 850, 663, 581, 751, 581, 581, 1571, 252, 252, 1283, 264, 430, 264, 430, 430, 842, 252, 745, 21, 307, 681, 1592, 488, 857, 857, 1161, 857, 857, 857, 138, 374, 374, 1196, 374, 1903, 1782, 1626, 414, 112, 1477, 1040, 356, 775, 414,
414, 112, 356, 775, 435, 338, 1066, 689, 689, 1501, 689, 1249, 205, 689, 765, 220, 308, 917, 308, 308, 220, 327, 387, 838, 917, 917, 917, 220, 662, 308, 220, 387, 387, 220, 220, 308, 308, 308, 387, 1009, 1745, 822, 279, 554, 1129, 543, 383, 870, 1425, 241, 870, 241, 383, 716, 592, 21, 21, 592, 425,
550, 550, 550, 427, 230, 57, 483, 784, 860, 57, 308, 57, 486, 870, 447, 486, 433, 433, 870, 433, 997, 486, 443, 433, 433, 997, 486, 1292, 47, 708, 81, 895, 394, 81, 935, 81, 81, 81, 374, 986, 916, 1103, 1095, 465, 495, 916, 667, 1745, 518, 220, 1338, 220, 734, 1294, 741, 166, 828, 741, 741, 1165,
1371, 1371, 471, 1371, 647, 1142, 1878, 1878, 1371, 1371, 822, 66, 327, 158, 427, 427, 465, 465, 676, 676, 30, 30, 676, 676, 893, 1592, 93, 455, 308, 582, 695, 582, 629, 582, 85, 1179, 85, 85, 1592, 1179, 280, 1027, 681, 398, 1027, 398, 295, 784, 740, 509, 425, 968, 509, 46, 833, 842, 401, 184,
401, 464, 6, 1501, 1501, 550, 538, 883, 538, 883, 883, 883, 1129, 550, 550, 333, 689, 948, 21, 21, 241, 2557, 2094, 273, 308, 58, 863, 893, 1086, 409, 136, 1086, 592, 592, 830, 830, 883, 830, 277, 68, 689, 902, 277, 453, 507, 129, 689, 630, 664, 550, 128, 1626, 1626, 128, 902, 312, 589, 755, 755,
589, 755, 407, 1782, 589, 784, 1516, 1118, 407, 407, 1447, 589, 235, 755, 1191, 235, 235, 407, 128, 589, 1118, 21, 383, 1331, 691, 481, 383, 1129, 1129, 1261, 1104, 1378, 1129, 784, 1129, 1261, 1129, 947, 1129, 784, 784, 1129, 1129, 35, 1104, 35, 866, 1129, 1129, 64, 481, 730, 1260, 481, 970,
481, 481, 481, 481, 863, 481, 681, 699, 863, 486, 681, 481, 481, 55, 55, 235, 1364, 944, 632, 822, 401, 822, 952, 822, 822, 99, 550, 2240, 550, 70, 891, 860, 860, 550, 550, 916, 1176, 1530, 425, 1530, 916, 628, 1583, 916, 628, 916, 916, 628, 628, 425, 916, 1062, 1265, 916, 916, 916, 280, 461, 916,
916, 1583, 628, 1062, 916, 916, 677, 1297, 924, 1260, 83, 1260, 482, 433, 234, 462, 323, 1656, 997, 323, 323, 931, 838, 931, 1933, 1391, 367, 323, 931, 1391, 1391, 103, 1116, 1116, 1116, 769, 1195, 1218, 312, 791, 312, 741, 791, 997, 312, 334, 334, 312, 287, 287, 633, 1397, 1426, 605, 1431, 327,
592, 705, 1194, 592, 1097, 1118, 1503, 1267, 1267, 1267, 618, 1229, 734, 1089, 785, 1089, 1129, 1148, 1148, 1089, 915, 1148, 1129, 1148, 1011, 1011, 1229, 871, 1560, 1560, 1560, 563, 1537, 1009, 1560, 632, 985, 592, 1308, 592, 882, 145, 145, 397, 837, 383, 592, 592, 832, 36, 2714, 2107, 1588,
1347, 36, 36, 1443, 1453, 334, 2230, 1588, 1169, 650, 1169, 2107, 425, 425, 891, 891, 425, 2532, 679, 274, 274, 274, 325, 274, 1297, 194, 1297, 627, 314, 917, 314, 314, 1501, 414, 1490, 1036, 592, 1036, 1025, 901, 1218, 1025, 901, 280, 592, 592, 901, 1461, 159, 159, 159, 2076, 1066, 1176, 1176,
516, 327, 516, 1179, 1176, 899, 1176, 1176, 323, 1187, 1229, 663, 1229, 504, 1229, 916, 1229, 916, 1661, 41, 36, 278, 1027, 648, 648, 648, 1626, 648, 646, 1179, 1580, 1061, 1514, 1008, 1741, 2076, 1514, 1008, 952, 1089, 427, 952, 427, 1083, 425, 427, 1089, 1083, 425, 427, 425, 230, 920, 1678,
920, 1678, 189, 189, 953, 189, 133, 189, 1075, 189, 189, 133, 1264, 725, 189, 1629, 189, 808, 230, 230, 2179, 770, 230, 770, 230, 21, 21, 784, 1118, 230, 230, 230, 770, 1118, 986, 808, 916, 30, 327, 918, 679, 414, 916, 1165, 1355, 916, 755, 733, 433, 1490, 433, 433, 433, 605, 433, 433, 433, 1446,
679, 206, 433, 21, 2452, 206, 206, 433, 1894, 206, 822, 206, 2073, 206, 206, 21, 822, 21, 206, 206, 21, 383, 1513, 375, 1347, 432, 1589, 172, 954, 242, 1256, 1256, 1248, 1256, 1256, 1248, 1248, 1256, 842, 13, 592, 13, 842, 1291, 592, 21, 175, 13, 592, 13, 13, 1426, 13, 1541, 445, 808, 808, 863,
647, 219, 1592, 1029, 1225, 917, 1963, 1129, 555, 1313, 550, 660, 550, 220, 660, 552, 663, 220, 533, 220, 383, 550, 1278, 1495, 636, 842, 1036, 425, 842, 425, 1537, 1278, 842, 554, 1508, 636, 554, 301, 842, 792, 1392, 1021, 284, 1172, 997, 1021, 103, 1316, 308, 1210, 848, 848, 1089, 1089, 848,
848, 67, 1029, 827, 1029, 2078, 827, 1312, 1029, 827, 590, 872, 1312, 427, 67, 67, 67, 67, 872, 827, 872, 2126, 1436, 26, 2126, 67, 1072, 2126, 1610, 872, 1620, 883, 883, 1397, 1189, 555, 555, 563, 1189, 555, 640, 555, 640, 1089, 1089, 610, 610, 1585, 610, 1355, 610, 1015, 616, 925, 1015, 482,
230, 707, 231, 888, 1355, 589, 1379, 151, 931, 1486, 1486, 393, 235, 960, 590, 235, 960, 422, 142, 285, 285, 327, 327, 442, 2009, 822, 445, 822, 567, 888, 2611, 1537, 323, 55, 1537, 323, 888, 2611, 323, 1537, 323, 58, 445, 593, 2045, 593, 58, 47, 770, 842, 47, 47, 842, 842, 648, 2557, 173, 689,
2291, 1446, 2085, 2557, 2557, 2291, 1780, 1535, 2291, 2391, 808, 691, 1295, 1165, 983, 948, 2000, 948, 983, 983, 2225, 2000, 983, 983, 705, 948, 2000, 1795, 1592, 478, 592, 1795, 1795, 663, 478, 1790, 478, 592, 1592, 173, 901, 312, 4, 1606, 173, 838, 754, 754, 128, 550, 1166, 551, 1480, 550,
550, 1875, 1957, 1166, 902, 1875, 550, 550, 551, 2632, 551, 1875, 1875, 551, 2891, 2159, 2632, 3231, 551, 815, 150, 1654, 1059, 1059, 734, 770, 555, 1592, 555, 2059, 770, 770, 1803, 627, 627, 627, 2059, 931, 1272, 427, 1606, 1272, 1606, 1187, 1204, 397, 822, 21, 1645, 263, 263, 822, 263, 1645,
280, 263, 605, 1645, 2014, 21, 21, 1029, 263, 1916, 2291, 397, 397, 496, 270, 270, 1319, 264, 1638, 264, 986, 1278, 1397, 1278, 1191, 409, 1191, 740, 1191, 754, 754, 387, 63, 948, 666, 666, 1198, 548, 63, 1248, 285, 1248, 169, 1248, 1248, 285, 918, 224, 285, 1426, 1671, 514, 514, 717, 514, 51,
1521, 1745, 51, 605, 1191, 51, 128, 1191, 51, 51, 1521, 267, 513, 952, 966, 1671, 897, 51, 71, 592, 986, 986, 1121, 592, 280, 2000, 2000, 1165, 1165, 1165, 1818, 222, 1818, 1165, 1252, 506, 327, 443, 432, 1291, 1291, 2755, 1413, 520, 1318, 227, 1047, 828, 520, 347, 1364, 136, 136, 452, 457, 457,
132, 457, 488, 1087, 1013, 2225, 32, 1571, 2009, 483, 67, 483, 740, 740, 1013, 2854, 866, 32, 2861, 866, 887, 32, 2444, 740, 32, 32, 866, 2225, 866, 32, 1571, 2627, 32, 850, 1675, 569, 1158, 32, 1158, 1797, 2641, 1565, 1158, 569, 1797, 1158, 1797, 55, 1703, 42, 55, 2562, 675, 1703, 42, 55, 749,
488, 488, 347, 1206, 1286, 1286, 488, 488, 1206, 1286, 1206, 1286, 550, 550, 1790, 860, 550, 2452, 550, 550, 2765, 1089, 1633, 797, 2244, 1313, 194, 2129, 194, 194, 194, 818, 32, 194, 450, 1313, 2387, 194, 1227, 2387, 308, 2232, 526, 476, 278, 830, 830, 194, 830, 194, 278, 194, 714, 476, 830,
714, 830, 278, 830, 2532, 1218, 1759, 1446, 960, 1747, 187, 1446, 1759, 960, 105, 1446, 1446, 1271, 1446, 960, 960, 1218, 1446, 1446, 105, 1446, 960, 488, 1446, 427, 534, 842, 1969, 2460, 1969, 842, 842, 1969, 427, 941, 2160, 427, 230, 938, 2075, 1675, 1675, 895, 1675, 34, 129, 1811, 239, 749,
1957, 2271, 749, 1908, 129, 239, 239, 129, 129, 2271, 2426, 1355, 1756, 194, 1583, 194, 194, 1583, 194, 1355, 194, 1628, 2221, 1269, 2425, 1756, 1355, 1355, 1583, 1033, 427, 582, 30, 582, 582, 935, 1444, 1962, 915, 733, 915, 938, 1962, 767, 353, 1630, 1962, 1962, 563, 733, 563, 733, 353, 822,
1630, 740, 2076, 2076, 2076, 589, 589, 2636, 866, 589, 947, 1528, 125, 273, 1058, 1058, 1161, 1635, 1355, 1161, 1161, 1355, 1355, 650, 1206, 1206, 784, 784, 784, 784, 784, 412, 461, 412, 2240, 412, 679, 891, 461, 679, 679, 189, 189, 1933, 1651, 2515, 189, 1386, 538, 1386, 1386, 1187, 1386,
2423, 2601, 2285, 175, 175, 2331, 194, 3079, 384, 538, 2365, 2294, 538, 2166, 1841, 3326, 1256, 3923, 976, 85, 550, 550, 1295, 863, 863, 550, 1249, 550, 1759, 146, 1069, 920, 2633, 885, 885, 1514, 1489, 166, 1514, 2041, 885, 2456, 885, 2041, 1081, 1948, 362, 550, 94, 324, 2308, 94, 2386, 94,
550, 874, 1329, 1759, 2280, 1487, 493, 493, 2099, 2599, 1431, 1086, 1514, 1086, 2099, 1858, 368, 1330, 2599, 1858, 2846, 2846, 2907, 2846, 713, 713, 1854, 1123, 713, 713, 3010, 1123, 3010, 538, 713, 1123, 447, 822, 555, 2011, 493, 508, 2292, 555, 1736, 2135, 2704, 555, 2814, 555, 2000, 555,
555, 822, 914, 327, 679, 327, 648, 537, 2263, 931, 1496, 537, 1296, 1745, 1592, 1658, 1795, 650, 1592, 1745, 1745, 1658, 1592, 1745, 1592, 1745, 1658, 1338, 2124, 1592, 1745, 1745, 1745, 837, 1726, 2897, 1118, 1118, 230, 1118, 1118, 1118, 1388, 1748, 514, 128, 1165, 931, 514, 2974, 2041,
2387, 2041, 979, 185, 36, 1269, 550, 173, 812, 36, 1165, 2676, 2562, 1473, 2885, 1982, 1578, 1578, 383, 383, 2360, 383, 1578, 2360, 1584, 1982, 1578, 1578, 1578, 2019, 1036, 355, 724, 2023, 205, 303, 355, 1036, 1966, 355, 1036, 401, 401, 401, 830, 401, 849, 578, 401, 849, 849, 578, 1776, 1123,
552, 2632, 808, 1446, 1120, 373, 1529, 1483, 1057, 893, 1284, 1430, 1529, 1529, 2632, 1352, 2063, 1606, 1352, 1606, 2291, 3079, 2291, 1529, 506, 838, 1606, 1606, 1352, 1529, 1529, 1483, 1529, 1606, 1529, 259, 902, 259, 902, 612, 612, 284, 398, 2991, 1534, 1118, 1118, 1118, 1118, 1118, 734,
284, 2224, 398, 734, 284, 734, 398, 3031, 398, 734, 1707, 2643, 1344, 1477, 475, 1818, 194, 1894, 691, 1528, 1184, 1207, 1501, 6, 2069, 871, 2069, 3548, 1443, 2069, 2685, 3265, 1350, 3265, 2069, 2069, 128, 1313, 128, 663, 414, 1313, 414, 2000, 128, 2000, 663, 1313, 699, 1797, 550, 327, 550,
1526, 699, 327, 1797, 1526, 550, 550, 327, 550, 1426, 1426, 1426, 2285, 1123, 890, 728, 1707, 728, 728, 327, 253, 1187, 1281, 1364, 1571, 2170, 755, 3232, 925, 1496, 2170, 2170, 1125, 443, 902, 902, 925, 755, 2078, 2457, 902, 2059, 2170, 1643, 1129, 902, 902, 1643, 1129, 606, 36, 103, 338,
338, 1089, 338, 338, 338, 1089, 338, 36, 340, 1206, 1176, 2041, 833, 1854, 1916, 1916, 1501, 2132, 1736, 3065, 367, 1934, 833, 833, 833, 2041, 3017, 2147, 818, 1397, 828, 2147, 398, 828, 818, 1158, 818, 689, 327, 36, 1745, 2132, 582, 1475, 189, 582, 2132, 1191, 582, 2132, 1176, 1176, 516,
2610, 2230, 2230, 64, 1501, 537, 1501, 173, 2230, 2988, 1501, 2694, 2694, 537, 537, 173, 173, 1501, 537, 64, 173, 173, 64, 2230, 537, 2230, 537, 2230, 2230, 2069, 3142, 1645, 689, 1165, 1165, 1963, 514, 488, 1963, 1145, 235, 1145, 1078, 1145, 231, 2405, 552, 21, 57, 57, 57, 1297, 1455, 1988,
2310, 1885, 2854, 2014, 734, 1705, 734, 2854, 734, 677, 1988, 1660, 734, 677, 734, 677, 677, 734, 2854, 1355, 677, 1397, 2947, 2386, 1698, 128, 1698, 3028, 2386, 2437, 2947, 2386, 2643, 2386, 2804, 1188, 335, 746, 1187, 1187, 861, 2519, 1917, 2842, 1917, 675, 1308, 234, 1917, 314, 314, 2339,
2339, 2592, 2576, 902, 916, 2339, 916, 2339, 916, 2339, 916, 1089, 1089, 2644, 1221, 1221, 2446, 308, 308, 2225, 2225, 3192, 2225, 555, 1592, 1592, 555, 893, 555, 550, 770, 3622, 2291, 2291, 3419, 465, 250, 2842, 2291, 2291, 2291, 935, 160, 1271, 308, 325, 935, 1799, 1799, 1891, 2227, 1799,
1598, 112, 1415, 1840, 2014, 1822, 2014, 677, 1822, 1415, 1415, 1822, 2014, 2386, 2159, 1822, 1415, 1822, 179, 1976, 1033, 179, 1840, 2014, 1415, 1970, 1970, 1501, 563, 563, 563, 462, 563, 1970, 1158, 563, 563, 1541, 1238, 383, 235, 1158, 383, 1278, 383, 1898, 2938, 21, 2938, 1313, 2201,
2059, 423, 2059, 1313, 872, 1313, 2044, 89, 173, 3327, 1660, 2044, 1623, 173, 1114, 1114, 1592, 1868, 1651, 1811, 383, 3469, 1811, 1651, 869, 383, 383, 1651, 1651, 3223, 2166, 3469, 767, 383, 1811, 767, 2323, 3355, 1457, 3341, 2640, 2976, 2323, 3341, 2323, 2640, 103, 103, 1161, 1080, 2429,
370, 2018, 2854, 2429, 2166, 2429, 2094, 2207, 871, 1963, 1963, 2023, 2023, 2336, 663, 2893, 1580, 691, 663, 705, 2046, 2599, 409, 2295, 1118, 2494, 1118, 1950, 549, 2494, 2453, 2046, 2494, 2453, 2046, 2453, 2046, 409, 1118, 4952, 2291, 2225, 1894, 1423, 2498, 567, 4129, 1475, 1501, 795,
463, 2084, 828, 828, 232, 828, 232, 232, 1818, 1818, 666, 463, 232, 220, 220, 2162, 2162, 833, 4336, 913, 35, 913, 21, 2927, 886, 3037, 383, 886, 876, 1747, 383, 916, 916, 916, 2927, 916, 1747, 837, 1894, 717, 423, 481, 1894, 1059, 2262, 3206, 4700, 1059, 3304, 2262, 871, 1831, 871, 3304, 1059,
1158, 1934, 1158, 756, 1511, 41, 978, 1934, 2603, 720, 41, 756, 41, 325, 2611, 1158, 173, 1123, 1934, 1934, 1511, 2045, 2045, 2045, 1423, 3206, 3691, 2512, 3206, 2512, 2000, 1811, 2504, 2504, 2611, 2437, 2437, 2437, 1455, 893, 150, 2665, 1966, 605, 398, 2331, 1177, 516, 1962, 4241, 94, 1252,
760, 1292, 1962, 1373, 2000, 1990, 3684, 42, 1868, 3779, 1811, 1811, 2041, 3010, 5436, 1780, 2041, 1868, 1811, 1780, 1811, 1868, 1811, 2041, 1868, 1811, 5627, 4274, 1811, 1868, 4602, 1811, 1811, 1474, 2665, 235, 1474, 2665

C.6. Random Numbers

The two tables V_0 and V_1 described in Section C.5.4.1 are given below. Each entry is a 32-bit integer in decimal representation.

C.6.1. The table V_0

251291136, 3952231631, 3370958628, 4070167936, 123631495, 3351110283, 3218676425, 2011642291, 774603218, 2402805061, 1004366930, 1843948209, 428891132, 3746331984, 1591258008, 3067016507, 1433388735, 504005498, 2032657933, 3419319784, 2805686246, 3102436986, 3808671154, 2501582075, 3978944421, 246043949, 4016898363, 649743608, 1974987508, 2651273766, 2357956801, 689605112, 715807172, 2722736134, 191939188, 3535520147, 3277019569, 1470435941, 3763101702, 3232409631, 122701163, 3920852693, 782246947, 3721213310, 2995604341, 2045698575, 2332962102, 4005368743, 218596347, 3415381967, 4207612806, 861117671, 3676575285, 2581671944, 3312220480, 681232419, 3073068666, 4112503940, 1158111502, 709227802, 2724140433, 4201101115, 4215970289, 4048876515, 3031661061, 1909085522, 510985033, 1361682810, 129243379, 3142379587, 2569842483, 3033268270, 1658118006, 932109358, 1982290045, 2983082771, 3007670818, 3448104768, 683749698, 778296777, 1399125101, 1939403708, 1692176003, 3868299200, 1422476658, 593093658, 1878973865, 2526292949, 1591602827, 3986158854, 3964389521, 2695031039, 1942050155, 424618399, 1347204291, 2669179716, 2434425874, 2540801947, 1384069776, 4123580443, 1523670218, 2708475297, 1046771089, 2229796016, 1255426612, 4213663089, 1521339547, 3041843489, 420130494, 10677091, 515623176, 3457502702, 2115821274, 2720124766, 3242576090, 854310108, 425973987, 325832382, 1796851292, 2462744411, 1976681690, 1408671665, 1228817808, 3917210003, 263976645, 2593736473, 2471651269, 4291353919, 650792940, 1191583883, 3046561335, 2466530435, 2545983082, 969168436, 2019348792, 2268075521, 1169345068, 3250240009, 3963499681, 2560755113, 911182396, 760842409, 3569308693, 2687243553, 381854665, 2613828404, 2761078866, 1456668111, 883760091, 3294951678, 1604598575, 1985308198, 1014570543, 2724959607, 3062518035, 3115293053, 138853680, 4160398285, 3322241130, 2068983570, 2247491078, 3669524410, 1575146607, 828029864, 3732001371, 3422026452, 3370954177, 4006626915, 543812220, 1243116171, 3928372514, 2791443445, 4081325272, 2280435605, 885616073, 616452097, 3188863436, 2780382310, 2340014831, 1208439576, 258356309, 3837963200, 2075009450, 3214181212, 3303882142, 880813252, 1355575717, 207231484, 2420803184, 358923368, 1617557768, 3272161958, 1771154147, 2842106362, 1751209208, 1421030790, 658316681, 1940658839, 3241510581, 38625260, 3018875395, 4176141739, 2973129930, 2137802113, 1502984205, 3669376622, 3728477036, 234652930, 2213589897, 2734638912, 1129721478, 3187422815, 2859178611, 3284308411, 3819792700, 3557526733, 451874476, 1740576081, 3592838701, 1709429513, 3702918379, 3533351328, 1641660745, 179350258, 2380520112, 3936163904, 3685256204, 3156252216, 1854258901, 2861641019, 3176611298, 834787554, 331353807, 517858103, 3010168884, 4012642001, 2217188075, 3756943137, 3077882590, 2054995199, 3081443129, 3895398812, 1141097543, 2376261053, 2626898255, 2554703076, 401233789, 1460049922, 678083952, 1064990737, 940909784, 1673396780, 528881783, 1712547446, 3629685652, 1358307511

C.6.2. The table V_1

807385413, 2043073223, 3336749796, 1302105833, 2278607931, 541015020, 1684564270, 372709334, 3508252125, 1768346005, 1270451292, 2603029534, 2049387273, 3891424859, 2152948345, 4114760273, 915180310, 3754787998, 700503826, 2131559305, 1308908630, 224437350, 4065424007, 3638665944, 1679385496, 3431345226, 1779595665, 3068494238, 1424062773, 1033448464, 4050396853, 3302235057, 420600373, 2868446243, 311689386, 259047959, 4057180909, 1575367248, 4151214153, 1102497841, 3006865921, 4293710613, 3501256572, 998007483, 499288295, 1205710710, 2997199489, 640417429, 3044194711, 486690751, 2686640734, 2394526209, 2521660077, 49993987, 3843885867, 4201106668, 415906198, 19296841, 2402488407, 2137119134, 1744097284, 579965637, 2037662632, 852173610, 2681403713, 1047144830, 2982173936, 910285038, 4187576520, 2589870048, 989448887, 3292758024, 506322719, 176010738, 1865471968, 2619324712, 564829442, 1996870325, 339697593, 4071072948, 3618966336, 2111320126, 1093955153, 957978696, 892010560, 1854601078, 1873407527, 2498544695, 2694156259, 1927339682, 1650555729, 183933047, 3061444337, 2067387204, 228962564, 3904109414, 1595995433, 1780701372, 2463145963, 307281463, 3237929991, 3852995239, 2398693510, 3754138664, 522074127, 146352474, 4104915256, 3029415884, 3545667983, 3320389010, 976628269, 3123492423, 3041418372, 2258059298, 2139377204, 3243642973, 3226247917, 3674004636, 2698992189, 3453843574, 1963216666, 3509855005, 2358481858, 747331248, 1957348676, 1097574450, 2435697214, 3870972145, 1888833893, 2914085525, 4161315584, 1273113343, 3269644828, 3681293816, 412536684, 1156034077, 3823026442, 1066971017, 3598330293, 1979273937, 2079029895, 1195045909, 1071986421, 2712821515, 3377754595, 2184151095, 750918864, 2585729879, 4249895712, 1832579367, 1192240192, 946734366, 31230688, 3174399083, 3549375728, 1642430184, 1904857554, 861877404, 3277825584, 4267074718, 3122860549, 666423581, 644189126, 226475395, 307789415, 1196105631, 3191691839, 782852669, 1608507813, 1847685900, 4069766876, 3931548641, 2526471011, 766865139, 2115084288, 4259411376, 3323683436, 568512177, 3736601419, 1800276898, 4012458395, 1823982, 27980198, 2023839966, 869505096, 431161506, 1024804023, 1853869307, 3393537983, 1500703614, 3019471560, 1351086955, 3096933631, 3034634988, 2544598006, 1230942551, 3362230798, 159984793, 491590373, 3993872886, 3681855622, 903593547, 3535062472, 1799803217, 772984149, 895863112, 1899036275, 4187322100, 101856048, 234650315, 3183125617, 3190039692, 525584357, 1286834489, 455810374, 1869181575, 922673938, 3877430102, 3422391938, 1414347295, 1971054608, 3061798054, 830555096, 2822905141, 167033190, 1079139428, 4210126723, 3593797804, 429192890, 372093950, 1779187770, 3312189287, 204349348, 452421568, 2800540462, 3733109044, 1235082423, 1765319556, 3174729780, 3762994475, 3171962488, 442160826, 198349622, 45942637, 1324086311, 2901868599, 678860040, 3812229107, 19936821, 1119590141, 3640121682, 3545931032, 2102949142, 2828208598, 3603378023, 4135048896

C.7 Example FEC decoder

C.7.1 General

This section describes an efficient decoding algorithm for the Raptor codes described in this specification. Note that each received encoding symbol can be considered as the value of an equation amongst the intermediate symbols. From these simultaneous equations, and the known pre-coding relationships amongst the intermediate symbols, any algorithm for solving simultaneous equations can successfully decode the intermediate symbols and hence the source symbols. However, the algorithm chosen has a major effect on the computational efficiency of the decoding.

C.7.2 Decoding a source block

C.7.2.1 General

It is assumed that the decoder knows the structure of the source block it is to decode, including the symbol size, T, and the number K of symbols in the source block.

From the algorithms described in Sections C.5, the Raptor decoder can calculate the total number $L = K+S+H$ of pre-coding symbols and determine how they were generated from the source block to be decoded. In this description it is assumed that the received encoding symbols for the source block to be decoded are passed to the decoder. Furthermore, for each such encoding symbol it is assumed that the number and set of intermediate symbols whose exclusive-or is equal to the encoding symbol is passed to the decoder. In the case of source symbols, the source symbol triples described in Section C.5.2.2 indicate the number and set of intermediate symbols which sum to give each source symbol.

Let $N \geq K$ be the number of received encoding symbols for a source block and let $M = S+H+N$. The following M by L bit matrix **A** can be derived from the information passed to the decoder for the source block to be decoded. Let **C** be the column vector of the L intermediate symbols, and let **D** be the column vector of M symbols with values known to the receiver, where the first $S+H$ of the M symbols are zero-valued symbols that correspond to LDPC and Half symbols (these are check symbols for the LDPC and Half symbols, and not the LDPC and Half symbols themselves), and the remaining N of the M symbols are the received encoding symbols for the source block. Then, **A** is the bit matrix that satisfies $\mathbf{A} \cdot \mathbf{C} = \mathbf{D}$, where here $\cdot$ denotes matrix multiplication over GF[2]. In particular, $A[i,j] = 1$ if the intermediate symbol corresponding to index j is exclusive-ORed into the LDPC, Half or encoding symbol corresponding to index i in the encoding, or if index i corresponds to a LDPC or Half symbol and index j corresponds to the same LDPC or Half symbol. For all other i and j, $A[i,j] = 0$.

Decoding a source block is equivalent to decoding **C** from known **A** and **D**. It is clear that **C** can be decoded if and only if the rank of **A** over GF[2] is L. Once **C** has been decoded, missing source symbols can be obtained by using the source symbol triples to determine the number and set of intermediate symbols which must be exclusive-ORed to obtain each missing source symbol.

The first step in decoding **C** is to form a decoding schedule. In this step **A** is converted, using Gaussian elimination (using row operations and row and column reorderings) and after discarding $M - L$ rows, into the L by L identity matrix. The decoding schedule consists of the sequence of row operations and row and column re-orderings during the Gaussian elimination process, and only depends on **A** and not on **D**. The decoding of **C** from **D** can take place concurrently with the forming of the decoding schedule, or the decoding can take place afterwards based on the decoding schedule.

The correspondence between the decoding schedule and the decoding of **C** is as follows. Let $c[0] = 0$, $c[1] = 1 \ldots, c[L-1] = L-1$ and $d[0] = 0$, $d[1] = 1 \ldots, d[M-1] = M-1$ initially.

- Each time row i of **A** is exclusive-ORed into row i' in the decoding schedule then in the decoding process symbol $D[d[i]]$ is exclusive-ORed into symbol $D[d[i']]$.
- Each time row i is exchanged with row i' in the decoding schedule then in the decoding process the value of $d[i]$ is exchanged with the value of $d[i']$.
- Each time column j is exchanged with column j' in the decoding schedule then in the decoding process the value of $c[j]$ is exchanged with the value of $c[j']$.

From this correspondence it is clear that the total number of exclusive-ORs of symbols in the decoding of the source block is the number of row operations (not exchanges) in the Gaussian elimination. Since **A** is the L by L identity matrix after the Gaussian elimination and after discarding the last $M - L$ rows, it is clear at the end of successful decoding that the L symbols $D[d[0]]$, $D[d[1]],\ldots,$ $D[d[L-1]]$ are the values of the L symbols $C[c[0]]$, $C[c[1]],\ldots,$ $C[c[L-1]]$.

The order in which Gaussian elimination is performed to form the decoding schedule has no bearing on whether or not the decoding is successful. However, the speed of the decoding depends heavily on the order in which Gaussian elimination is performed. (Furthermore, maintaining a sparse representation of **A** is crucial, although this is not described here). The remainder of this section describes an order in which Gaussian elimination could be performed that is relatively efficient.

C.7.2.2 First Phase

The first phase of the Gaussian elimination the matrix **A** is conceptually partitioned into submatrices. The submatrix sizes are parameterized by non-negative integers i and u which are initialized to 0. The submatrices of **A** are:

(1) The submatrix **I** defined by the intersection of the first i rows and first i columns. This is the identity matrix at the end of each step in the phase.

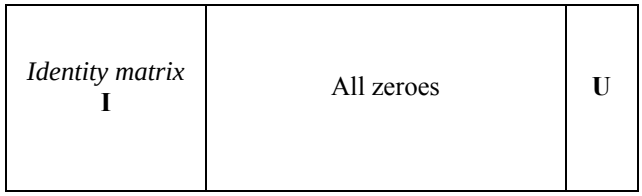
(2) The submatrix defined by the intersection of the first i rows and all but the first i columns and last u columns. All entries of this submatrix are zero.

(3) The submatrix defined by the intersection of the first i columns and all but the first i rows. All entries of this submatrix are zero.

(4) The submatrix **U** defined by the intersection of all the rows and the last u columns.

(5) The submatrix **V** formed by the intersection of all but the first i columns and the last u columns and all but the first i rows.

Figure C.7.2.2-1 illustrates the submatrices of **A**. At the beginning of the first phase **V** = **A**. In each step, a row of **A** is chosen.



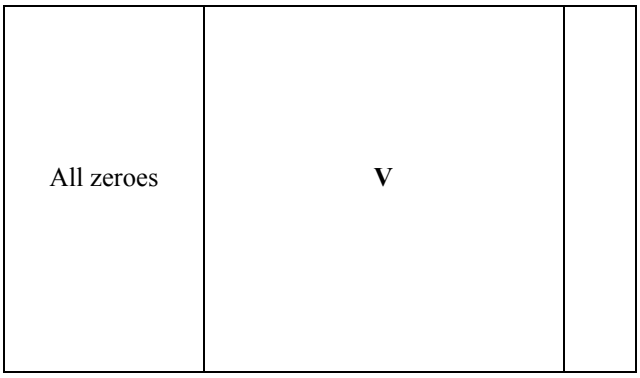


Figure C.7.2.2-1: Submatrices of A in the first phase.

The following graph defined by the structure of **V** is used in determining which row of **A** is chosen. The columns that intersect **V** are the nodes in the graph, and the rows that have exactly 2 ones in **V** are the edges of the graph that connect the two columns (nodes) in the positions of the two ones. A component in this graph is a maximal set of nodes (columns) and edges (rows) such that there is a path between each pair of nodes/edges in the graph. The size of a component is the number of nodes (columns) in the component.

There are at most L steps in the first phase. The phase ends successfully when $i + u = L$, i.e., when **V** and the all zeroes submatrix above **V** have disappeared and A consists of **I**, the all zeroes submatrix below **I**, and **U**. The phase ends unsuccessfully in decoding failure if at some step before **V** disappears there is no non-zero row in **V** to choose in that step. In each step, a row of **A** is chosen as follows:

- If all entries of **V** are zero then no row is chosen and decoding fails.
- Let r be the minimum integer such that at least one row of **A** has exactly r ones in **V**.
- If $r \neq 2$ then choose a row with exactly r ones in **V** with minimum original degree among all such rows.
- If $r = 2$ then choose any row with exactly 2 ones in **V** that is part of a maximum size component in the graph defined by X.

After the row is chosen in this step the first row of **A** that intersects **V** is exchanged with the chosen row so that the chosen row is the first row that intersects **V**. The columns of **A** among those that intersect **V** are reordered so that one of the r ones in the chosen row appears in the first column of **V** and so that the remaining r-1 ones appear in the last columns of **V**. Then, the chosen row is exclusive-ORed into all the other rows of **A** below the chosen row that have a one in the first column of **V**. Finally, i is incremented by 1 and u is incremented by r-1, which completes the step.

C.7.2.3 Second Phase

The submatrix **U** is further partitioned into the first i rows, $\mathbf{U}_{upper}$, and the remaining $M - i$ rows, $\mathbf{U}_{lower}$. Gaussian elimination is performed in the second phase on $\mathbf{U}_{lower}$ to either determine that its rank is less than u (decoding failure) or to convert it into a matrix where the first u rows is the identity matrix (success of the second phase). Call this u by u identity matrix $\mathbf{I}_u$. The $M - L$ rows of A that intersect $\mathbf{U}_{lower} - \mathbf{I}_u$ are discarded. After this phase **A** has L rows and L columns.

C.7.2.4 Third Phase

After the second phase the only portion of **A** which needs to be zeroed out to finish converting **A** into the L by L identity matrix is $\mathbf{U}_{upper}$. The number of rows i of the submatrix $\mathbf{U}_{upper}$ is generally much larger than the number of columns u of $\mathbf{U}_{upper}$. To zero out $\mathbf{U}_{upper}$ efficiently, the following precomputation matrix **U'** is computed based on $\mathbf{I}_u$ in the third phase and then **U'** is used in the fourth phase to zero out $\mathbf{U}_{upper}$. The u rows of $\mathbf{I}_u$ are partitioned into ceil(u/8) groups of 8 rows each. Then, for each group of 8 rows all non-zero combinations of the 8 rows are computed, resulting in 2^8 - 1 = 255 rows (this can be done with 2^8-8-1 = 247 exclusive-ors of rows per group, since the combinations of Hamming weight one that appear in $\mathbf{I}_u$ do not need to be recomputed). Thus, the resulting precomputation matrix **U'** has ceil(u/8) ·255 rows and u columns. Note that **U'** is not formally a part of matrix **A**, but will be used in the fourth phase to zero out $\mathbf{U}_{upper}$.

C.7.2.5 Fourth Phase

For each of the first *i* rows of **A**, for each group of 8 columns in the $\mathbf{U}_{upper}$submatrix of this row, if the set of 8 column entries in $\mathbf{U}_{upper}$ are not all zero then the row of the precomputation matrix **U'** that matches the pattern in the 8 columns is exclusive-ORed into the row, thus zeroing out those 8 columns in the row at the cost of exclusive-oring one row of **U'** into the row.

After this phase **A** is the *L* by *L* identity matrix and a complete decoding schedule has been successfully formed. Then, as explained in Section C.2.1, the corresponding decoding consisting of exclusive-ORing known encoding symbols can be executed to recover the intermediate symbols based on the decoding schedule.

The triples associated with all source symbols are computed according to C.5.2.2.The triples for received source symbols are used in the decoding.The triples for missing source symbols are used to determine which intermediate symbols need to be exclusive-ORed to recover the missing source symbols.

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 [29]. If only parts of a service are protected the protected programmes are signalled by setting clearToAir in Schedule fragment to false [29]. 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 [29]. 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:

m=data <PORT> UDP ipdc-ksm

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

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;

Examples for referencing key stream messages in SDP media descriptions

An ISMACryp encrypted video stream, could be signalled as:

```
m=video 41970 rtp/savp 96
a=rtpmap:96 enc-generic-mp4/16000/1
a=fmtp:96 mode=video; 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
a=rtpmap:96 H264/9000
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 96a=rtpmap:96 H264/9000a=fmtp:96 profile-level-id=42A01E;
sprop-parameter-sets=Z0IACpZTBYmI,aMljiA==;

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.