ETSI TS 103 179 V1.1.1 (2013-08)

Satellite Earth Stations and Systems (SES);
Return Link Encapsulation (RLE) protocol

Reference
DTS/SES-00339

Keywords
MSS, protocol, satellite

ETSI

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

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

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

Important notice

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

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

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

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

Copyright Notification

Contents

Intellectual Property Rights

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

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

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Satellite Earth Stations and Systems (SES).

1 Scope

The present document specifies the Return Link Encapsulation (RLE) Protocol, which is used to encapsulate and if necessary fragment network layer packets such as for example IP datagrams to allow their transmission over the return link of an interactive satellite network.

RLE has been derived from the Generic Stream Encapsulation (GSE) protocol [1], used in the forward links of interactive and broadcasting satellite networks, which are normally characterised by continuous transmission, limited variability in the size of network layer packets, and large physical layer frames typically capable of carrying more than one network layer packet. RLE was designed to maximise the system efficiency on the return channel, which is in turn characterised by bursty traffic, highly variable size of network layer packets, smaller physical layer bursts, and multiple access constraints.

The RLE protocol is designed to provide three main functionalities which are fully specified in the present document, namely:

- encapsulation
- fragmentation
- frame packing

RLE is today used in DVB-RCS2 [2] as well as in the S-MIM [3] standards.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

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

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

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 102 606: "Digital Video Broadcasting (DVB); Generic Stream Encapsulation (GSE) Protocol".

[2] ETSI EN 301 545-2: "Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard".

[3] ETSI TS 102 721-5: "Satellite Earth Stations and Systems; Air Interface for S-band Mobile Interactive Multimedia (S-MIM); Part 5: Protocol Specifications, Link Layer".

[4] IETF RFC 4326: "Unidirectional Lightweight Encapsulation (ULE) for Transmission of IP Datagrams over an MPEG-2 Transport Stream (TS)".

[5] IEEE 802.3- 2012: IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks-- Specific requirements; Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications".

[6] IETF RFC 4944: "Transmission of IPv6 Packets over IEEE 802.15.4 Networks".

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] IETF RFC 5163: "Extension Formats for Unidirectional Lightweight Encapsulation (ULE) and the Generic Stream Encapsulation (GSE)".

[i.2] ETSI EN 301 790: "Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems".

[i.3] DVB Document A155-2: "Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard"; January 2013.

[i.4] ETSI TS 102 721-3: "Satellite Earth Stations and Systems (SES); Air Interface for S-band Mobile Interactive Multimedia (S-MIM); Part 3: Physical Layer Specification, Return Link Asynchronous Access".

[i.5] ETSI TS 102 721-4: "Satellite Earth Stations and Systems (SES); Air Interface for S-band Mobile Interactive Multimedia (S-MIM); Part 4: Physical Layer Specification, Return Link Synchronous Access".

3 Definitions and abbreviations

3.1 Definitions

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

Protocol Data Unit (PDU): unit of data produced by a layer of the network stack and given to the next lower layer for transport to the remote peer layer instance

NOTE 1: It contains control information from the current layer and may contain user data from the layer above.

NOTE 2: The SDU of a given layer is the PDU of the layer above.

RLE receiver: entity processing received physical layer frame or burst payloads to reconstruct original network layer packets.

RLE transmitter: entity processing network layer packets to produce physical layer frame or burst payloads to be transmitted through the corresponding physical layer.

Service Data Unit (SDU): unit of data that is passed from one layer of the network stack to the next lower layer which has not yet been encapsulated into a PDU by that lower layer

NOTE: It is the set of data given by the user of a layer service to that layer to be transmitted semantically unchanged to the peer service user. The SDU of a given layer is the PDU of the layer above.

3.2 Abbreviations

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

ACM	Adaptive Coding and Modulation
ALPDU	Addressed Link PDU
ARQ	Automatic Repeat reQuest
bslbf	bit string, left bit first
CDMA	Code Division Muliple Access
CoS	Class of Service
CRC	Cyclic Redundancy Check
CRDSA	Contention Resolution Diversity Slotted ALOHA

DAMA	Demand Assignement Muliple Access
DCH	Dedicated CHannel
DVB-RCS	Digital Video Broadcasting - Return Channel via Satellite
FEC	Forward Error Correction
FPDU	Frame PDU
GSE	Generic Stream Encapsulation
IP	Internet Protocol
MAC	Medium Access Control
NCC	Network Control Center
PDU	Protocol Data Unit
PPDU	Payload-adapted PDU
QEF	Quasi Error Free
QS-CDMA	Quasi Sychnronous Code Division Multiple Access
RACH	Random Access CHannel
RLE	Return Link Encapsulation
ROHC	RObust Header Compression
rpchof	remainder polynomial coefficients, highest order first
SDU	Service Data Unit
S-MIM	S-Band Mobile Interactive Multimedia
SNDU	SubNetwork Data Unit
SSA	Spread Spectrum ALOHA
SVN	Switched Virtual Network
TCP	Transmission Control Protocol
uimsbf	unsigned integer, most significant bit first

4 System Aspects

The main use of the RLE protocol is the transport of data on the return link of transparent or regenerative interactive satellite networks and on the direct links between terminals in fully meshed systems (see Figure 4.1).

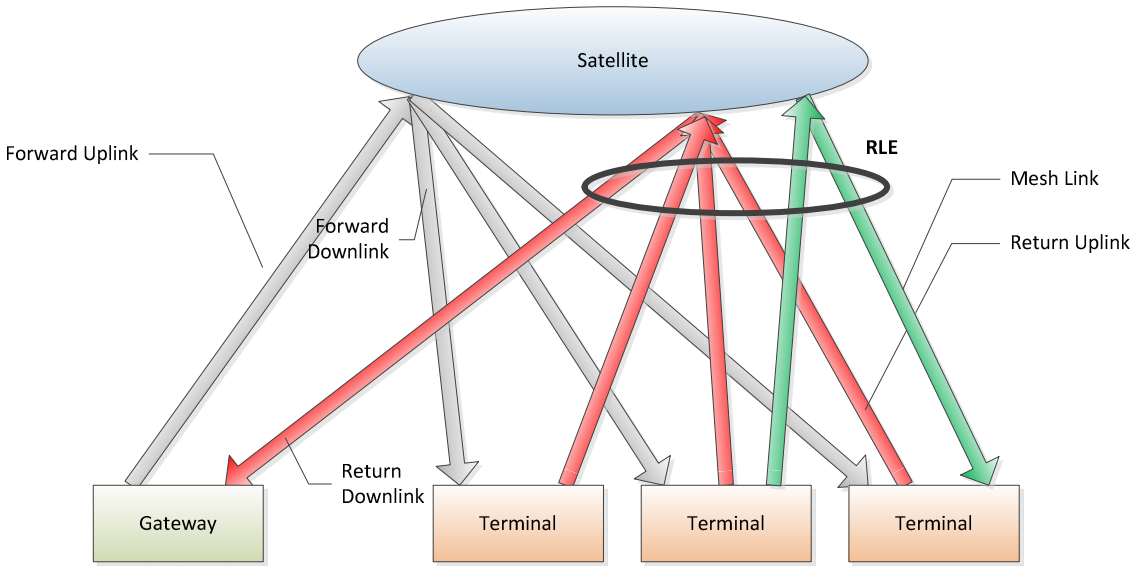

Figure 4.1: Interactive Satellite Network

The terminal transmitter contains the transmit side of the RLE protocol (RLE transmitter) which processes network layer packets and produces physical layer frame or burst payloads. The gateway receiver or the terminal receiver (in the meshed case) contain the receiving side of the RLE protocol (RLE receiver) which reconstructs network layer packets from the received burst or frame payloads. In regenerative satellite systems the RLE receiver may also partly or completely be located in the satellite.

4.1 Protocol Stack

The RLE protocol resides in the Data Link Layer of the communication system (see Figure 4.2). The RLE transmitter gets network layer packets from the network layer (for example IP datagrams) and produces burst or frame payloads that are delivered to the physical layer for transmission. On the receiver side burst or frame payloads are processed to extract the network layer packets and deliver them to the network layer.

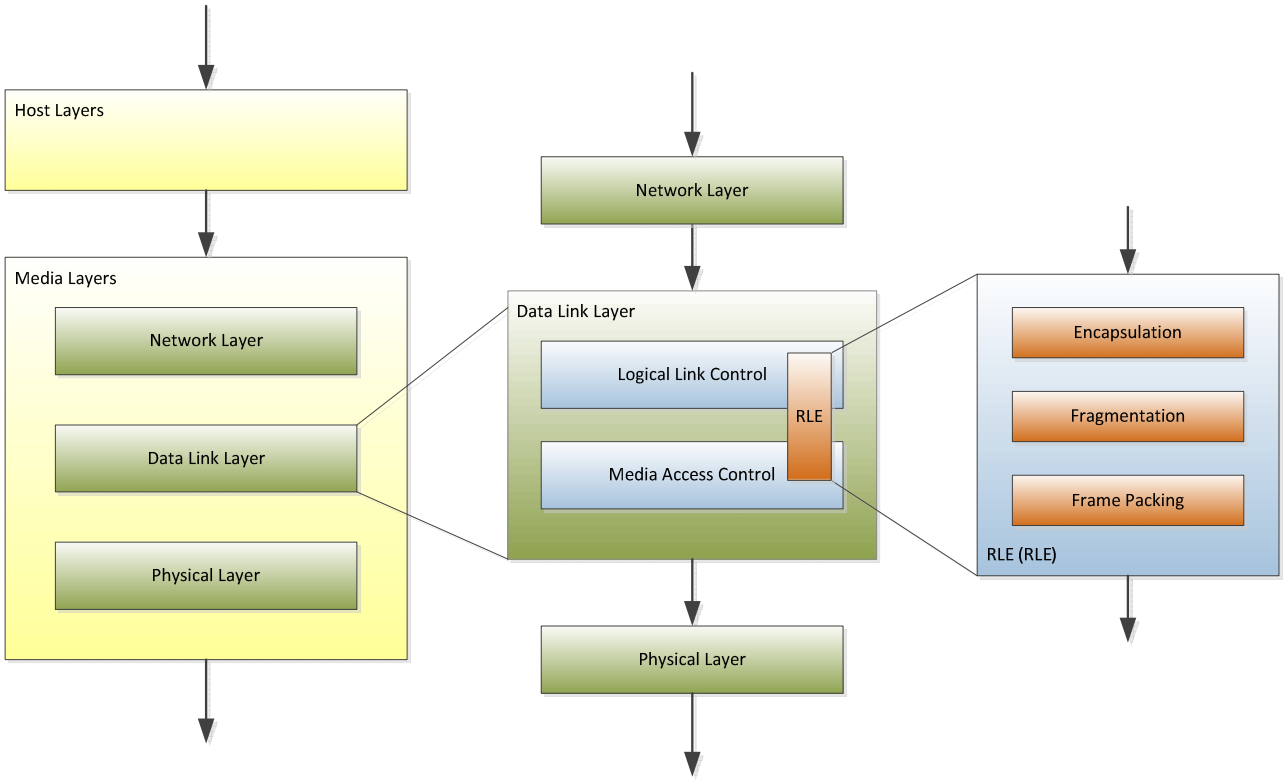

Figure 4.2: Network Protocol Stack

RLE has three distinct functions:

1) The Encapsulation function which is partly related to the Logical Link Control Layer takes higher layer packets and associated protocol type and label information and packs these into Addressed Link PDUs (ALPDUs). On the receiver side the ALPDUs are unpacked and dispatched to the network layer instance handling the given protocol type. The Encapsulation function realizes the protocol multiplexing function of the Logical Link Control Layer.

2) The Fragmentation function fragments ALPDUs into smaller units producing Payload-adapted PDUs (PPDUs), which are then multiplexed into a single stream. The fragmentation takes into account size information about the physical layer payload fields and may take into account priority information and other system-specific constraints. This sublayer normally includes a scheduler selecting packets to be processed based on this information. On the receiving side PPDUs are reassembled into ALPDUs, checked for errors and, in the success case, delivered upstreams. The fragmentation process is controlled by the current payload sizes required by the physical layer and the current ALPDU sizes. The physical layer payload sizes may change on a frame by frame or burst by burst basis (for example in systems using ACM) and the ALPDU sizes will adapt to the variable size of the higher layer protocols (TCP/IP, for example).

3) The Frame Packing function takes the stream of PPDUs and packs one or more of these into a Frame PDU (FPDU). It may optionally add signaling information, a label and provide additional error detection capabilities if the physical layer does not provide those. On the receiver side the Frame Packing function optionally checks for transmission errors, extracts all PPDUs from each FPDU and dispatches them to the right instance in the Fragmentation layer for reassembly of ALPDUs. Both the Fragmentation and the Frame Packing function are related to the Medium Access Control sublayer.

The protocol can be employed not only in single hop (direct transmitter to receiver) configurations, but also in multi-hop scenarios using switching. The switching can occur between any of the sub-layers of the protocol in which case the intermediate node implements only the sub-layers below the switching point. Examples are provided in Annex E.

4.2 Protocol Tailoring and Configuration

The RLE protocol can be tailored and configured to fit the transmission system requirements as good as possible. This involves two steps:

1) Selection of the capabilities and options to be implemented in the transmission system. This is done based on the system requirements and on the characteristics of the surrounding layers (physical layer and network layer). Features than are not needed and options that can be set to fixed values reduce the implementation and testing effort. The remaining options shall be provided with either configuration or signaling (or both) support in the system.

2) Configure the system operation. The limits of this configurability are defined by the previous step.

Complete information on protocol tailoring and configuration is provided in clause 6.

4.3 Physical Layer Requirements

RLE can operate over physical layers with fixed burst or frame sizes as well as over physical layers with burst or frame sizes varying from burst to burst or frame to frame as they occur in systems like DVB-RCS2 when using adaptive coding and modulation [2]. The RLE transmitter dynamically adapts the fragmentation to the available burst or frame size.

The following conditions shall be met by the physical layer:

1) The physical layer burst or frame payload shall have a certain minimum size. This size depends on the configuration of the FPDU (see clause 5.4) and the PPDU (see clause 5.3). The burst or frame payload size shall be large enough for the FPDU to be able to carry at least one PPDU. The space for the PPDU shall be large enough so that a PPDU with the `start_indicator` set to "1" and the `end_indicator` set to "0" fits into that space.

2) The physical layer shall either never reorder physical layer bursts or frames or it shall do so only in exceptional cases. In the later case the `use_alpdu_crc` option shall be used for reassembly error checking. If reordering occurs than all the higher layer PDUs data of which is carried in the reordered frames or bursts are lost.

3) The physical layer shall either operate quasi-error-free or the `use_frame_protection` option of the FPDU shall be used with a protection scheme that reduces the residual errors to quasi-error-free operation.

5 RLE Data Format

The RLE transmitter transforms the network layer PDU into an Addressed Link PDU (ALPDU), sections the ALPDU into one or more Payload-adapted PDUs (PPDUs) as required, and assembles PPDUs into FPDUs that fit into burst payload.

On the receiving side the FPDUs from the physical layer are unpacked, the resulting PPDUs are reassembled into ALPDUs. These in turn are decapsulated and the contained network layer PDUs are sent to the next upper layer (see Figure 5.1).

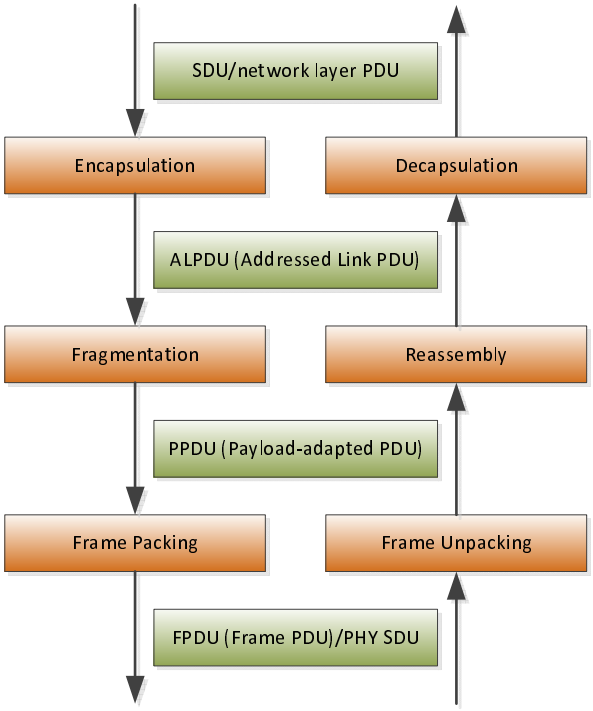

Figure 5.1: PDUs within RLE stack

5.1 Higher Layer SDU

The SDU is constructed from extension headers and higher layer PDUs, for example IP packets according to the rules in [4]. Each of the two parts, but not both at the same time, is optional. Associated to the SDU is a 16-bit protocol type value which is either the type value of the outermost extension header or the actual protocol type of the higher layer PDU if there are no extension headers. Also associated to the SDU may be a label of up to 15 bytes carrying address or other information.

5.2 The Addressed Link PDU (ALPDU)

The RLE transmitter shall build ALPDUs that, in addition to the SDU, may include an explicit protocol type indication and an explicit address tag in a similar structure as for GSE [1]. When both fields are included the label field is appended after the protocol type field and before the SDU. Both fields are optional. The ALPDU may have a non-zero size protection field (called PRO in the figure). This is illustrated in Figure 5.2.

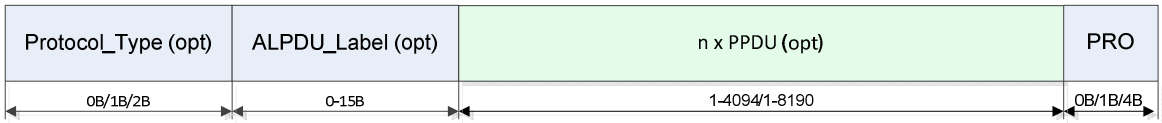

Figure 5.2: Addressed Link PDU Format

The ALPDU provides limited explicit integrity protection and thus relies on the integrity protection provided by the lower protocol layers. If the ALPDU fits into a single PPDU it is not provided with a protection (PRO) parameter field. When fragmented into multiple PPDUs, the ALPDU contains an integrity protection parameter field of either 1 byte or 4 bytes.

5.2.1 Addressed Link PDU Format and Syntax

The ALPDU format and syntax are defined in Table 5.1:

Table 5.1: Addressed Link PDU Format and Syntax

Syntax	No. of bits		Mnemonic
	Reserved	Information	
addressed_link_pdu() {			
if (implied_protocol_type[alpdu_label_type] <> protocol_type) {			
if (protocol_type_compressed[alpdu_label_type] = 1){			
compressed_protocol_type		8	uimsbf
for (i = 0; i < length[alpdu_label_type]; i++) {			
alpdu_label_byte		8	bslbf
}			
if (compressed_protocol_type = 0xff) {			
protocol_type		16	uimsbf
}			
}			
else {			
protocol_type		16	uimsbf
for (i = 0; i < length[alpdu_label_type]; i++) {			
alpdu_label_byte		8	bslbf
}			
}			
}			
else {			
for (i = 0; i < length[alpdu_label_type]; i++) {			
alpdu_label_byte		8	bslbf
}			
}			
for (i = 0; i < N1; i++) {			
sdu_byte		8	bslbf
}			
if (fragmented_alpdu) {			
if (use_alpdu_crc = 1) {			
alpdu_crc		32	rpchof
}			
else {			
sequence_number		8	uimsbf
}			
}			
}			

NOTE 1: `length[alpdu_label_type]` is the number of bytes in the ALPDU label (configuration parameter). The maximum length is 15 bytes.
NOTE 2: `implied_protocol_type[alpdu_label_type]` is the implied protocol type for the given label type (configuration parameter). There may be no implied protocol type.
NOTE 3: `protocol_type_compressed[alpdu_label_type]` is "1" if the label type allows protocol type compression (configuration parameter).
NOTE 4: `N1` is the number of bytes in the SDU.

The semantics for the `addressed_link_pdu` (ALPDU) parameters are specified in the following clauses. The specific values for the control parameters for the ALPDU structure may be partly given by the specific START PPDU for the respective ALPDU, partly by system signalling or specification and partly by more dynamic conditions like e.g. contiguous payload availability and SDU size.

5.2.1.1 `compressed_protocol_type` Field (optional)

This 8-bit field may be present in the ALPDU as defined by system specification or instructed by system signalling, if not explicitly suppressed by the Protocol Suppressed flag.

If present, the field shall explicitly indicate the protocol type of the SDU by its compressed equivalent or it shall alternatively indicate the presence of a trailing full size protocol type field. The compressed protocol type field can have these different lengths as indicated by system specification and signalling and the value of the Protocol Type Suppressed flag (provided by PPDU):

1) **1 byte.** This is a compressed protocol type.

2) **0 byte.** The protocol type of the SDU is the implicit protocol type implied by the specific ALPDU Label Type value used (as provided by PPDU).

The compressed protocol type value 0xFF is reserved to indicate a construction with insertion of a 2 byte protocol type field after the ALPDU label, intended to support indication of any SDU which immediately follow this protocol type field, and to support the utilization of extension headers together with the compressed protocol type.

The mapping between compressed and uncompressed protocol types has to be defined in the system specification or by system signalling.

5.2.1.2 `protocol_type` Field (optional)

This 16 bit field is present in the ALPDU if not explicitly indicated to be suppressed by the Protocol Type Suppressed flag (provided by PPDU) or excluded by use of other compressed protocol types than one indicating inclusion of the full size protocol type.

The `protocol_Type` field may explicitly indicate the protocol type of the SDU or the presence of header extensions. The field may have these different lengths as indicated by system specification and signalling and the value of the Protocol Type Suppressed flag (provided by PPDU):

1) **2 bytes in network byte order.** A supported SDU protocol type value.

2) **0 byte.** The protocol type of the SDU is the implicit protocol type implied by the specific ALPDU Label Type value used (as provided by the PPDU).

5.2.1.3 `alpdu_label_byte` Field (optional)

The `alpdu_label_byte` field is one byte of the ALPDU label.

The length of the ALPDU label is indicated in the PPDU by the ALPDU Label Type value. The correspondence between label types and ALPDU label lengths is defined in the system specification or established via signalling. See the specification of the ALPDU Label Type in clause 5.3.1.4.

5.2.1.4 `sdu_byte` Field

This 8 bit field holds one byte from the complete contiguous sequence of SDU bytes. The maximum number of SDU bytes supported depends the length of the protection field, the label length and the length of the protocol type field. The maximum size of the entire ALPDU is 4 095 bytes.

5.2.1.5 `fragmenting_aldpdu`

This control parameter is local to the RLE transmitter and reflects whether the ALPDU is mapped into one payload adapted PDU or fragmented into multiple such PDUs. A trailing field for integrity protection is included when the ALPDU is fragmented for adaptation to the next layer, and excluded when the ALPDU is contained within a single next layer PDU.

5.2.1.6 `sequence_number` Field (optional)

The 8 bit `sequence_number` field may be included at the end of the ALPDU and it is mutually exclusive with use of the `alpdu_crc` field.

The presence of the sequence number in the ALPDU is either indicated by the value 0 in the `use_alpdu_crc` field of the corresponding START PPDU if this field is present, or is given by the system specification or signalling. Allowance to apply the ALPDU Sequence Number is indicated in the system specification or by signalling. The transmitter shall not apply the ALPDU Sequence Number if not explicitly allowed.

The transmitter shall use an incrementing sequence number independently per `fragment_id` (see clause 5.3.1.3). The first sequence number used for a `fragment_id` after receiver start or reset shall be the value zero. It shall be incremented by one for each ALPDU sent using the respective `fragment_id`, and it shall be calculated modulo 256.

5.2.1.7 `alpdu_crc` Field (optional)

This 32 bit field may be included in the ALPDU and it is mutually exclusive with the `sequence_number` field. It carries the ALPDU CRC.

The presence of the ALPDU CRC is indicated by the value 1 in the `use_alpdu_crc` field of the corresponding START PPDU if this field is present, or is given by the system specification or signalling. Allowance to apply the ALPDU CRC is explicitly indicated by the system specification or signalling. The transmitter shall not apply the ALPDU CRC if not explicitly allowed.

The CRC is calculated using the same fields (in the same order) and algorithm as GSE [1]. In the case of the use of implied or compressed protocol types the protocol type field shall be correctly expanded and the protocol type expansion field shall be removed before calculating the CRC. The following fields are used for CRC calculation in the given order:

1) A 16-bit length field calculated as the sum of:
   - 2 (length of the uncompressed protocol type in bytes).
   - The length of the ALPDU label in bytes.
   - The length of the SDU in bytes.
2) The 16-bit uncompressed protocol type.
3) The ALPDU label.
4) The SDU.

The CRC algorithm is described in Annex A.

5.2.2 The ALPDU Label

The format and length of the ALPDU label is specified in the system specification of the transmission system. For each transmission context used in the system up to four different label lengths and formats shall be specified for the four different ALPDU label types to be used at the same time.

5.2.3 Mapping the ALPDU to Available Payload

The transmitter shall fragment an ALPDU into Payload-adapted PDUs as necessary to fit these PPDUs into transmission frames, aimed for the receivers targeted by the ALPDU.

5.2.3.1 Forwarding the ALPDU in One Payload-adapted PDU

The ALPDU may be transmitted in full in a single PPDU if the payload can hold the PPDU.

5.2.3.2 Forwarding the ALPDU Using Several Payload-adapted PDUs

The ALPDU may be fragmented and mapped into a sequence of a start PPDU, optionally a number of intermediate PPDUs and an end PPDU finalizing the transport of the ALPDU. These PPDUs shall all be tagged with the same Fragment ID value.

The Fragment ID values shall be managed in the transmission context of the transmitter. Each fragment of an ALPDU shall be transmitted in a PPDU tagged with the same Fragment ID value as the PPDUs carrying other fragments of the same ALPDU. The fragments of an ALPDU shall be transmitted in their natural sequence, with the header fragment first.

5.2.3.3 Integrity Protection of a Fragmented ALPDU

A transmitter shall use either the ALPDU CRC method or the ALPDU Sequence Number method for integrity protection at fragmentation of the ALPDU onto several PPDUs. The transmitter shall use a method explicitly allowed by the system specification or signalling.

A transmitter shall associate an independent 8 bit wrapping counter with each Fragment ID value. The counter shall initialise to zero at transmitter initialization time. The counter value shall be appended as the ALPDU Sequence Number to the END PPDU before incrementing the counter by one to provide the value for the next end PPDU tagged with the same Fragment ID value.

The 32-bit ALPDU CRC shall be calculated using the algorithm specified in Annex A. It is to be computed over a possibly expanded ALPDU constituted by the concatenation of the corresponding 16-bit SDU protocol type (even if suppressed or compressed) and the optional ALPDU label if present, all extension headers and the higher layer SDU.

5.2.3.4 Multiplexing Payload-adapted PDUs used for Different ALPDUs

A transmitter may multiplex PPDUs associated to different ALPDUs even if all of the PPDUs carry fragments of different ALPDUs, as long as each of the ALPDUs not yet finalized by an end PPDU is associated with a Fragment ID value that is not associated to any other ALPDU in progress from the transmitter. An ALPDU in progress is associated to a start PPDU transmitted earlier but the ALPDU is not yet finalized by transmission of an END PPDU.

A PPDU both starting and finalizing a complete ALPDU may be sent anywhere in a sequence of PPDUs.

5.3 The Payload-adapted PDU (PPDU)

The RLE transmitter shall use a Payload-adapted PDU (PPDU) format complying with this clause as specified in the system specification or by signalling.

An ALPDU shall be transported by using one or more PPDUs.

This set of 4 PPDU types is defined:

1) FULL PPDU: the PPDU type for an unfragmented ALPDU.

2) START PPDU: the first PPDU for an ALPDU that utilizes several PPDUs.

3) CONTINUATION PPDU: the PPDU for an ALPDU fragment that is a continuation of an ALPDU, following in time the preceding adjacent ALPDU fragment transmitted in an earlier PPDU.

4) END PPDU: the PPDU that finalizes an ALPDU.

The different PPDU formats are illustrated in Figure 5.3.

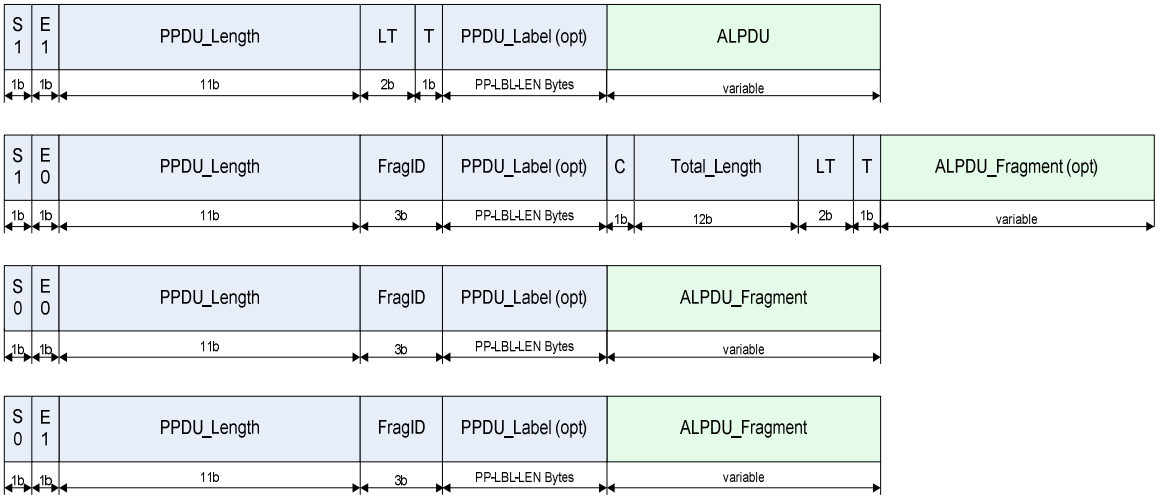

Figure 5.3: Payload-adapted PDU Formats

The PPDU has a first header of two bytes which is present in all PPDU types. These two bytes start with two bits that indicate the type of the PPDU and that have the same semantics as for the protocol specified for the GSE protocol in [1]. These two bits are followed by a length field of 11 bits that indicates the length of the varying length part of the PPDU. The meaning of the remaining three bits differ: for a PPDU with an unfragmented ALPDU these bits contain the ALPDU Label Type field (two bits) and the Protocol Type Suppression flag. For the other PPDU types these three bits contain the Fragmentation ID field.

The two-byte first header of the PPDU is immediately followed by the optional PPDU label, if this is present.

The remainder of the PPDU content depends on the values of the `start_indicator` and `end_indicator`. This is described in the following clauses.

The START PPDU is never less than 4 bytes due to a second header of 2 bytes following after the optional PPDU label. The presence of an ALPDU section is optional both in the START PPDU and the END PPDU as well as in a FULL PPDU, but an ALPDU section shall be present in a CONTINUATION PPDU to avoid that this PPDU resembles the start of payload padding in the position of the first header.

NOTE: There may be implementations of the specifications in the present document that require the transmitter to avoid splitting the 4 last bytes in the ALPDU across FPDUs, when these 4 bytes are used for the CRC32.

The 2 byte PPDU header containing the value 0x0000 indicates start of FPDU padding.

5.3.1 The Payload-adapted PDU Format and Syntax

The format and syntax of the PPDU is specified in Table 5.2.

Table 5.2: Payload-adapted PDU Format and Syntax

Syntax	No. of bits		Mnemonic
	Reserved	Information	
`payload_adapted_pdu() {`			
`  start_indicator`		1	bslbf
`  end_indicator`		1	bslbf
`  ppdu_length`		11	uimsbf
`  if (start_indicator = 1 and end_indicator = 1) {`			
`    alpdu_label_type`		2	uimsbf
`    protocol_type_suppressed`		1	bslbf
`  }`			
`  else {`			
`    fragment_id`		3	uimsbf
`  }`			
`  for (i = 0; i < M; i++) {`			
`    ppdu_label_byte`		8	uimsbf
`  }`			

Syntax	No. of bits		Mnemonic
	Reserved	Information	
if (start_indicator = 1 and end_indicator = 0) {			
if (large_alpdus)			
total_length		13	uimsbf
}			
else {			
use_alpdu_crc		1	bslbf
total_length		12	uimsbf
}			
alpdu_label_type		2	uimsbf
protocol_type_suppressed		1	bslbf
}			
for (i = 0; i < N; i++) {			
alpdu_byte		8	bslbf
}			
}			

NOTE 1: M is the number of PPDU label bytes that applies for all the PPDUs in the payload carrying this PPDU.
NOTE 2: N is the number of bytes in the ALPDU section carried by the specific PPDU, and the section may be any fragment of the ALPDU or the complete ALPDU.

The semantic of the fields of the payload_adapted_pdu and the corresponding rules are explained in the following clauses.

5.3.1.1 start_indicator and end_indicator Fields

These are both 1 bit fields and appear in all PPDUs.

A value of "1" in the Start Indicator position indicates that the PPDU contains the initiation of transport of an ALPDU. A value of "0" indicates that the PPDU either contains an intermediate section or finalizes an ALPDU where earlier sections of the ALPDU are contained in PPDUs transmitted earlier in the PPDU sequence, if not indicating start of padding.

A value of "1" in the End Indicator position indicates that the PPDU contains the finalization of an ALPDU. A value of "0" indicates either an intermediate section of an ALPDU or the initialisation of an ALPDU, if not indicating padding.

If both start and end indicators are "0", the PPDU Length is "0" and the Fragment ID is "0" this is not a PPDU but instead the start of padding filling the rest of the available transmission frame payload space.

5.3.1.2 ppdu_length Field

This 11 bit field is present in all PPDU types.

The ppdu_length field contains the length of the PPDU exclusive of the two byte PPDU header and exclusive of the PPDU label.

A value of 0 in the PPDU Length field position shall only occur if both the Start Indicator field and the Start Indicator field are 0. This condition together with zero in the Fragment ID field indicates that these fields are not indicating the start of a PPDU but the start of payload padding.

5.3.1.3 fragment_id Field

This 3 bit field is present in all PPDU types but the FULL PPDU.

A receiver shall be able to concurrently receive PPDUs for of up to 8 ALPDUs from each possible transmitter (then all possible Fragment ID values are in use by all possible transmitters). All PPDUs carrying data from the same ALPDU shall use the same fragment_id. PPDUs from different ALPDUs that are fragmented at the same time shall use different fragment_ids.

5.3.1.4 `alpdu_label_type` Field

This 2 bit field is present in the START PPDU and in the FULL PPDU. It indicates either the length and contents of the ALPDU label and the implicit protocol type (if any). There are four possible values allowing four different label formats and implicit protocol types. The correspondence between the label type value and the label length and format has to be defined in the system specification. The implicit protocol type for each label type can also be specified in the system specification or can be signalled.

Annex D provides a generic algorithm for the selection of the most appropriate label type.

5.3.1.5 `protocol_type_suppressed` Field

This 1 bit field is present in the START PPDU and in the FULL PPDU.

The inclusion of the `protocol_type` field in the ALPDU is indicated by the Protocol Type Suppressed flag set to "0".

The omission of the `protocol_type` field, and thus the use of an implicit protocol type for the SDU is indicated by the Protocol Type Suppressed flag set to "1".

5.3.1.6 `ppdu_label_byte` Field (optional)

This one byte field holds one byte of the PPDU label and may be present in any PPDU type.

The size and format of the PPDU labels applicable for the PPDUs in a given payload is indicated either explicitly in the optional FPDU header or the size that applies for a given payload is specified in the system specification or by signalling. The explicit indication in the FPDU header takes precedence.

The maximum supported PPDU label size is 15 bytes.

5.3.1.7 `large_alpdus` Field

This is an internal configuration variable for both the transmitter and receiver which can assume the following two values:

1) **Value "0".** The maximum ALPDU length is 4 095 bytes. The `total_length` field of the PPDU is 12 bits long and the use of the `alpdu_crc` field or the `sequence_number` field is signalled for each single ALPDU in the `use_alpdu_crc` field of the PPDU header.

2) **Value "1".** The maximum ALPDU length is 8 191 bytes. The total_length field of the PPDU is 13 bits long and the use of the `alpdu_crc` field or the `sequence_number` field is either fixed for the given transmission system or signalled by out-of-band signalling.

5.3.1.8 `use_alpdu_crc` Field (optional)

This one-bit field is present in the START PPDU when the `large_alpdus` field has a value of "0". It indicates whether the ALPDU Sequence Number or the ALPDU CRC is used. If the bit is set, the ALPDU CRC is included. If the bit is cleared, the ALPDU Sequence Number is included. If `large_alpdus` is set to "1" then the `use_alpdu_crc` is conceptually still present, but its value is fixed and not transmitted in the PPDU.

5.3.1.9 `total_length` Field

The 12 or 13 bit `total_length` field is present in the START PPDU. It indicates the size of the ALPDU. The field size allows a maximum size of 4 095 or 8 191 bytes. The maximum ALPDU size may be further restricted by the system specification or signalling.

5.3.1.10 `alpdu_byte` Field

This field represents one single byte of the ALPDU from the section of contiguous ALPDU bytes contained in the given ALPDU. The ALPDU is specified in clause 5.2. An ALPDU may be fragmented into contiguous sections where the first section is put into one PPDU, the next section into another PPDU transmitted later and so forth until the transmission of the ALPDU is finalized.

5.3.2 The PPDU Label

The PPDU label conveys information attached to the PPDU. Its size, format and syntax is defined in the system specification for each transmission context. When the `ppdu_label_length` field is not used in the FPDU (see clause 5.4.1.3) only a single size, format and syntax is available per transmission context. If the `ppdu_label_length` field is used different label lengths, and correspondingly formats and syntaxes, may be defined for each transmission context.

5.4 The Frame PDU (FPDU)

The transmitter shall build transmission frame payloads as required by the underlying layer. Different transmission contexts may use different variants of frame payload structure. The configurable options for each transmission context are specified in the system specification and may additionally be modified dynamically by signalling.

The generic structure of the FPDU for a frame that supports transport of user traffic is illustrated in Figure 5.4.

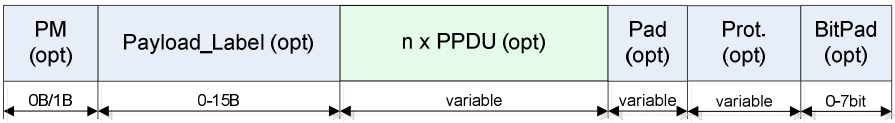

Figure 5.4: FPDU Format

5.4.1 The FPDU Format and Syntax

The transmission FPDU format and syntax for a frame for user traffic shall be in accordance with Table 5.3. Depending on the context this specifies the payload of a transmission burst or a transmission frame.

Table 5.3: FPDU Format and Syntax

Syntax	No. of bits		Mnemonic
	Reserved	Information	
`frame_pdu() {`			
`  if (use_explicit_payload_header_map = 1) {`			
`    payload_label_length`		4	uismbf
`    ppdu_label_length`		4	uismbf
`  }`			
`  for (b = 0; b < payload_label_length; b++) {`			
`    payload_label_byte`		8	bslbf
`  }`			
`  for (j = 0; j < N1; j++) {`			
`    for (i = 0; i < X1(j); i++) {`			
`      ppdu_byte`		8	bslbf
`    }`			
`  }`			
`  for (p = 0; p < P1; p++) {`			
`    padding_byte`		8	bslbf
`  }`			
`  if (use_frame_protection) {`			
`    for (p = 0; p < T1; p++) {`			
`      protection_byte`		8	bslbf
`    }`			
`  }`			
`  if (not_byte_aligned) {`			

Syntax	No. of bits		Mnemonic
	Reserved	Information	
for (k = 0; k < K1; k++) {			
padding_bit		1	bslbf
}			
}			
}			

NOTE 1: P1 is the number of complete padding bytes required to fill the frame payload.
NOTE 2: N1 is the number of PPDUs in the payload, X1(j) is the number of bytes occupied by PPDU 'j'.
NOTE 3: T1 is the number of bytes in the frame protection field.
NOTE 4: The values of use_frame_protection and use_explicit_payload_header_map are specified in the system specification or by signalling.
NOTE 5: K1 is less than 8 and is the number of bits required to fill the payload.

The size of the transmission frame payload is given by the construction of the physical layer, and may differ between the different transmission contexts and modes.

The semantics of the fields of the frame_pdu and the corresponding rules are explained in the following clauses.

5.4.1.1 use_explicit_payload_header_map Field

The use_explicit_payload_header_map is an internal field for both the transmitter and the receiver for a given transmission context. It has the same value for both the transmitter and the receiver. This value is defined in the system specification or by signalling. The field can have one of the following two values:

1) **Value "0".** The payload_label_length and ppdu_label_length fields are not present in the FPDU and the lengths of both labels shall be given by the system specification or signalling. This can be used if the lengths do not change for a given transmission context.

2) **Value "1".** The payload_label_length and ppdu_label_length fields are present in the FPDU and their values overwrite the lengths given by the system specification or signalling for this FPDU and all Payload-adapted PDUs in it.

5.4.1.2 payload_label_length Field (optional)

This field holds the length of the payload label for this FPDU in bytes. The supported range is 0 to 15 bytes. If this fields is present the ppdu_label_length shall also be present. The presence of both fields is controlled by the use_explicit_payload_header_map flag.

If this field is not present the label length defined in the system specification or by signalling shall be used.

5.4.1.3 ppdu_label_length Field (optional)

This field holds the length of the Payload-adapted PDU label in bytes. The supported range is 0 to 15 bytes. If this fields is present the payload_label_length shall also be present. The presence of both fields is controlled by the use_explicit_payload_header_map flag.

If this field is not present the label length defined in the system specification or by signalling shall be used.

5.4.1.4 payload_label_byte Field (optional)

One byte of the Payload Label. The payload label size is specified by the payload_label_length field or the system specification and signalling if this fields is not present. The Payload Label can be used to transport information associated to the FPDU.

5.4.1.5 ppdu_byte (optional)

A sequence of X1(n) of this 8 bit field contains one PPDU with structure and semantics as specified in clause 5.3. The contiguous section of ppdu_bytes holds N1 complete PPDUs.

5.4.1.6 padding_byte (optional)

One byte from a variable size padding field. Any bytes from this position and up to the FPDU end, but not including the optional protection bytes, are padding bytes and shall be set to zero. A single remaining byte following in the payload after the last PPDU is padding.

5.4.1.7 use_frame_protection field

This is an internal field for both the transmitter and the receiver for a given transmission context. It has the same value for both the transmitter and the receiver. This value is defined in the system specification or by signalling. The field can have one of the following two values:

1) **Value "0".** No frame protection used. It is assumed that the underlying physical layer is quasi-error-free.

2) **Value "1".** Frame protection used for error detection. The undelying physical layer may provide error correction but does not provide error detection. The kind of error detection and the length of the protection field is system-specific and is specified in the system specification.

5.4.1.8 protection_byte (optional)

Contains the error detection bytes for FPDU protection. This field is only present if use_frame_protection is set to "1". The length and kind of error detection is system specific.

5.4.1.9 padding_bit (optional)

One bit out of 0-7 padding bits that all shall be set to zero. This field is present only when the payload size of the frame or burst carrying the FPDU is not a multiple of 8 bits.

5.4.2 The FPDU Payload Label

The FPDU Payload Label conveys information attached to the FPDU. Its size, format and syntax is defined in the system specification for each transmission context. When the payload_label_length field is not used in the FPDU only a single size, format and syntax is available per transmission context. If the payload_label_length field is used different label lengths, and correspondingly formats and syntaxes, shall be defined for each transmission context.

5.4.3 FPDU error protection

The RLE protocol assumes a quasi-error-free physical layer. If the physical layer is not quasi-error-free error an optional error detection mechanism can be used to protect the entire FPDU (excluding the padding bits). The kind of error detection mechanism (CRC, for example) and the length of the information used for error detection is system specific and needs to be selected based on the properties of the physical layer. The additional information is inserted into the protection bytes at the end of the FPDU by the transmitter and are used for error detection by the receiver.

6 RLE Configuration and Tailoring

The present clause lists all the configuration options of the protocol.

6.1 System Specification and Signalling

RLE offers a number of options that can be used to adapt the protocol to the concrete transmission context. A number of options is signalled in-band an thus can be changed on the fly. Other options are more static in nature and their use shall be either fixed by means of system specification or static configuration or shall be signalled by out-of-band means (with signalling SDUs, for example). Even the dynamic options have conceptually attached to it a static option that allows or disallows their use. The static options can be specified at different specification levels for a concrete transmission system:

1) **System Specification.** This is the kind of document that describes a certain kind of transmission system. Examples are the DVB-RCS2 standard [2] and the S-MIM standard [3]. If an option value is specified there then all system implementations will be interoperable with regard to this option. The system specification may disallow the use of certain RLE features for all systems conforming to that specification or may force certain values for RLE options that all systems shall support. The specification may also allow the system implementation to decide on features and option values.

2) **System Implementation.** A system specification may keep the choice of the option value to the system designer. In this case the option value is fixed for an implementation of the system specification. All options and option values not fixed or disallowed by the system specification or the system implementation shall be supported by all systems produced according to the system specification and implementation documents.

3) **System Configuration.** The options and option values that are not fixed in the system specification and the system implementation can be configured by the operator of the concrete system installation.

4) **System Operation.** The option value may be changed during system operation dynamically by signalling. Options that shall have the same value in the transmitter and the receiver shall be carefully changed to avoid loss of synchronization between the transmitter and the receiver. For systems using logon procedures these changes could be limited to logon time, for example. All options and values allowed by the system specification and the system implementation shall be supported.

Options and option values excluded by the System Specification or the System Implementation do not need to be implemented in the RLE implementation used for the system. This allows limiting the development and testing effort of the protocol by carefully choosing options at a higher specification level.

6.2 Higher layer SDU

6.2.1 Maximum SDU size

The maximum size of the ALPDU is 4 095 or 8 191 bytes depending on `large_alpdus` configuration setting. Depending on the use of compressed and implied protocol types and the length of the ALPDU labels the maximum SDU size is:

$$L_{SDU,Max} \leq L_{SDU,Max,RLE} = L_{ALPDU,Max} - L_{proto} - L_{label} \quad (1)$$

Where $L_{SDU,Max}$ is the maximum supported SDU size in bytes of the transmission system, $L_{SDU,Max,RLE}$ is the maximum SDU size supported by the protocol, L_{proto} is the size of the protocol type which is in the range from 0 to 3 bytes and L_{label} is the ALPDU label size in the range from 0 to 15 bytes. $L_{ALPDU,Max}$ is either 4 095 or 8 191 bytes. $L_{SDU,Max,RLE}$ may be further restricted by upper layer considerations to $L_{SDU,Max}$.

6.3 Addressed Link PDU (ALPDU)

6.3.1 Protocol type table

The protocol type table provides the protocol types and extension header types supported by the transmission system. If protocol type compression is used, the compressed values are also provided. The protocol type configuration includes the following information:

1) **16-bit protocol type.** This provides the uncompressed value which is required for communication with the layers above RLE and for calculation of the ALPDU CRC if CRC protection is used. The protocol type may directly use official EtherTypes (see [5]), privately defined protocol types or extension header types (see [4]).

2) **8-bit compressed protocol type.** If protocol type compression is to be used for a given 16-bit protocol type then the 8-bit compressed protocol type corresponding to it shall be defined. Compressed protocol type 0xFF is reserved to signal the presence of the uncompressed type after the ALPDU label.

3) **Protocol type.** Specifies whether the protocol type designates a real higher layer SDU type or an extension header. This flag is necessary in order to support SDU protocol types defined in the space reserved by RFC 4326 [4] for extension header types.

Table B.1 provides as an example an excerpt of the protocol type table for DVB-RCS2 [2].

Both the RLE transmitter and receiver may drop SDUs with protocol types other than those configured in the protocol type table. An RLE implementation may also choose to support all possible protocol types (uncompressed).

6.3.2 Label type table

For each of the four label types supported by the `alpdu_label_type` the following values shall be configured:

1) **Length and syntax of the ALPDU label.** The length shall be in the range from 0 to 15 bytes.

2) **Protocol type compression.** This configuration variable can have two values:

   - **Value "0".** The protocol type cannot be compressed for this label type.
   - **Value "1".** The protocol type shall be compressed for this label type.

3) **Use of implied protocol type.** This configuration variable can have two values:

   - **Value "0".** The protocol type cannot be omitted and has a length of 2 byte if compression is not allowed and 1 or 3 bytes if compression is allowed.
   - **Value "1".** If the protocol type of the SDU equals the implied protocol type then the protocol type field in the ALPDU can be omitted. In this case the `protocol_type_suppressed` shall be set to one in the header of the first PPDU created from the ALPDU.

4) **Implied protocol type.** If the use of implied protocol types is allowed this is the implied protocol type. If the use of implied protocol types is not allowed this value is not used.

Table B.2 provides as an example a possible label type configuration for a DVB-RCS2 system [2].

6.3.3 Extension headers

Extension headers are used to tag arbitrary data on higher layer PDUs or to add additional processing in the different stack layers (FEC, encryption) [4]. Because of the way extension headers work it is necessary to carefully specify the list of extension headers the transmitter may produce and the list of extension headers the receiver may process. The system specification of the transmission system should provide a list of mandatory and optional extension headers to be allowed for the RLE.

An implementation may chose to disallow fragmenting of all or certain extension headers. This is necessary for interworking with strong readings of the GSE standard ([1]) and may be required for certain types of extension headers that shall be processed before ALPDU reassembly (FEC or encryption, for example). This is a transmitter-only configuration option.

6.3.4 Maximum ALPDU Length

The configuration variable `large_alpdus` selects the maximum size of the ALPDU to be 4 095 (if the value is "0") or 8 191 bytes (if the value is "1").

6.3.5 Integrity Protection

ALPDUs that are fragmented into more than one PPDU have an integrity protection that uses either a 1-byte sequence number or a 4-byte CRC. Two configuration flags are conceptually attached to the two methods:

1) **Allow sequence number.** If set to "1" the sequence number method can be used; if set to "0" it cannot be used.

2) **Allow CRC.** If set to "1" the CRC methods can be used; if set to "0" it cannot be used.

At least one of these flags shall be set to "1". If both are set to "1" the transmitter can choose one of the two methods. This choice can be either done at pre-defined times, for example, at transmitter initialization, or independently for each PDU. Which methods is actually used is signalled in-band in the `use_alpdu_crc` of the ALPDU.

If the `large_alpdus` configuration variable is set to "1" then exactly one of "Allow sequence number" and "Allow CRC" shall be set to "1". This setting shall be fixed for the system or shall be modified at the transmitter and the receiver at the same time (for example by reinitialization).

6.4 Payload-adapted PDU (PPDU)

Most of the PPDU configuration options are shared with the ALPDU (label types, integrity protection and others).

6.4.1 PPDU Label

For each transmission context the length, format and syntax of the PPDU label shall be defined. The protocol supports a limited kind of flexibility of the PPDU label. Its length may be signalled in the FPDU, but the length is restricted to change from FPDU to FPDU. All PPDUs in a single FPDU shall use the same PPDU label length. The PPDU label length shall be in the range from 0 to 15 bytes.

6.4.2 ALPDU Fragmentation

An implementation may chose to disallow fragmentation for certain types of ALPDUs or label types. This can be necessary for ALPDUs that carry timing information which shall be accessed at layers below RLE, for example.

This is a transmitter-only option.

6.5 Frame PDU (FPDU)

6.5.1 Payload Header Map

If there is more than one possible lengths of the PPDU label or more than one possible length of the Payload label and these lengths are chosen by the transmitter on a FPDU by FPDU basis for a given transmission context, then the Payload Header Map field consisting of the `payload_label_length` field and the `ppdu_label_length` field shall be used to signal the actual lengths of the labels for each FPDU. If the labels have a constant length the Payload Header Map may be omitted. The `use_explicit_payload_header_map` setting shall be synchronized between transmitter and receiver.

6.5.2 Payload Label

The possible Payload Label lengths, formats and syntaxes shall be defined. If more than one label length needs to be supported for a given transmission context and that length can change from FPDU to FPDU the Payload Header Map field shall be used to signal the actual label length. The label length is restricted to the range of 0 to 15 bytes.

6.5.3 Frame Protection

Based on the characteristics of the underlying physical layer of the given transmission context the kind of frame protection shall be chosen. If the physical layer supports quasi-error-free transmission the frame protection may be omitted. Otherwise an appropriate error detection scheme shall be selected (CRC, for example). The length, format and syntax of the protection field as well as the algorithm used to calculate it shall be selected the same for both the transmitter and receiver of a given transmission link.

7 RLE Reassembly Error Check

The present clause describes details of the transmitter and receiver processing of the RLE protocol for the reassembly error check.

7.1 Principles

The RLE protocol allows parallel fragmentation and reassembly of several higher layer SDUs from the same sender at the same time. This is achieved by combining the information in the `fragment_id` header field of the PPDU with either the `sequence_number` field (if the `use_alpdu_crc` header field is set to "0"), or with the `alpdu_crc` field (if the `use_alpdu_crc` header field is set to "1"). The option to use a 1 byte sequence number instead of a 4 byte CRC32 is provided to reduce the packet overhead from 4 bytes to 1 byte in the cases that the environment allows for that. CRC32 shall be used in environments with very large drop-outs (more than 255 bursts in a row, mobile environments, for example) or for interoperability with other protocols.

It is recommended not to mix both options at a sender, although it is supported. On the one hand, the suitability of one or the other method is a trade-off between propagation impairments, system design and overhead reduction; this allows selecting one method or the other for the sender once. On the other side, letting the sender switch from one to the other option includes higher complexity for scheduling RLE packets in a burst, as the minimum length of the last PPDU of a fragmented ALPDU varies for different setting of the `use_alpdu_crc` header field.

The methods and rules to apply the sequence number and the CRC32 for reassembly error checking are described in the following clauses.

7.2 Reassembly error check algorithm with sequence number

If the sequence number is applied for reassembly error check, the following rules shall be applied:

1) The transmitter has a variable `next_sequence_number` for every `fragment_id` from 0 to 7. When emitting an END PPDU the fragmentation process appends the value of `next_sequence_number` to the encapsulated higher layer packet and increments the variable. When the value reaches 255, the next increment cycles back to 0.

2) The receiver has a variable `next_sequence_number` for every `fragment_id`. This variable contains the `sequence_number` expected in the next END PPDU that has the `use_alpdu_crc` bit set to "1". When the receiver receives such a PPDU it does the following:

   - It compares the `sequence_number` from the fragment to the `next_sequence_number` for the given `fragment_id`. If they differ, the reassembled ALPDU is thrown away. If they are equal the ALPDU is processed further.

   - It sets the `next_sequence_number` for the `fragment_id` to the value of `sequence_number` + 1 independently of whether the ALPDU was thrown away in the previous step or not. If the new value is 256 it cycles back to 0.

Both the fragmentation process and the defragmentation process initialize their next_sequence_number values for all fragment_ids to 0 at initialization time. If the transmitter sends a packet with an alpdu_crc the next_sequence_number is not incremented. When the receiver receives a packet with an alpdu_crc, its next_sequence_number is not incremented. This is done so that in the case of the loss of an ALPDU with an alpdu_crc the next packet with a sequence number is not lost too (an ALPDU with an alpdu_crc cannot resynchronize the next expected sequence number in the receiver).

Figure C.1 illustrates how the reassembly error detection mechanism with sequence numbers works.

Additional rules for reassembly error detection are:

1) If a PPDU arrives which has a start_indicator value of "0", but no ALPDU is currently reassembled for the given fragment_id, then a START PPDU has been lost and the received PPDU is discarded. If the PPDU has the end_indicator set equal to "1", the next_sequence_number of the receiver for this fragment_id is updated with the sequence_number value from the received PPDU plus 1.

2) If a PPDU arrives which has a start_indicator value of "1", and an ALPDU is currently reassembled for the given fragment_id, an END PPDU has been lost. The currently reassembled ALPDU is discarded, the next_sequence_number of the receiver for this fragment_id is incremented and the received PPDU packet is processed normally.

3) If the sum of the packet_length field of the received PPDU and the already reassembled length of the ALPDU is larger than the value in the total_length field of the first PPDU of that ALPDU then at least one START PPDU and one END PPDU have been lost. Both the currently reassembled ALPDU and the received PPDU are discarded. If the PPDU has the end_indicator value set to "1" and the start_indicator set to "0", the next_sequence_number is updated with the sequence_number value from the received PPDU plus 1.

4) If an END PPDU is received and the sum of the packet_length and the length of the already buffered ALPDU part does not match the value in the total_length field of the first PPDU of the reassembled ALPDU, the buffered ALPDU and the received PPDU are discarded. The next_sequence_number is set to the sequence_number value from the received PPDU plus one.

7.3 Reassembly error checking algorithm with CRC-32

If the CRC-32 is applied for reassembly error check, the following rules shall be applied:

1) When emitting an END PPDU the fragmentation process includes the alpdu_crc field calculated according to clause 5.2.1.7 into the transmitted PPDU.

2) When the receiver receives an END PPDU and the corresponding START PPDU had the use_alpdu_crc field set to "1" it does the following:

   - It performs an integrity check by calculating the CRC value according to Annex A over the fields of the reassembled ALPDU.

   - It compares the calculated CRC with the alpdu_crc value of the reassembled ALPDU. If they are not identical (integrity check fails), the reassembled ALPDU is discarded. Otherwise, if the integrity check is successful, the reassembled ALPDU is further processed.

Figure C.2 illustrates how the reassembly error detection mechanism with CRC-32 works:

1) If a PPDU arrives which has a start_indicator value of "0", but no ALPDU is currently reassembled for the given fragment_id, then a START PPDU has been lost and the received PPDU is discarded.

2) If a PPDU arrives which has a start_indicator value of "1", and an ALPDU is currently reassembled for the given fragment_id, an END PPDU has been lost. The currently reassembled ALPDU is discarded and the received PPDU is processed normally.

3) If the sum of the `packet_length` field of the received PPDU and the already reassembled length of the ALPDU is larger than the value in the `total_length` field of the first PPDU of that ALPDU then at least START PPDU and one END PPDU have been lost. Both the currently reassembled ALPDU and the received PPDU are discarded.

4) If an END PPDU is received and the sum of the `packet_length` and the length of the already buffered ALPDU part does not match the value in the `total_length` field of the first PPDU of the reassembled ALPDU, the buffered ALPDU and the received PPDU are discarded.

Annex A (normative): ALPDU CRC-32 Calculation

This annex defines the CRC-32 algorithm to be used for the `alpdu_crc` field.

The generator polynom for the CRC-32 calculation shall be:

x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0

represented in hexadecimal as 0x104C11DB7.

The CRC accumulator shall be initialized to 0xFFFFFFFF then data shall be fed into the algorithm according to the following rules:

1) 16-bit integer values shall be processed with the most significant byte first;
2) the label, the extension headers and the SDU shall be processed byte by byte starting at their resp. first byte;
3) each byte shall be processed starting with the most significant bit first.

The resulting 32-bit value shall be inserted into the 4 byte `alpdu_crc` field most significant byte first and the most significant CRC-bit mapped to the most significant bit of the output byte.

The Rocksoft specification of the algorithm shall be:

Width:	32
Poly:	0x04C11DB7
Init:	0xFFFFFFFF
RefIn:	FALSE
RefOut:	FALSE
XorOut:	0x00000000
Check:	0x0376E6E7

Annex B (informative): RLE Configuration

This annex provides examples for some of the configuration tables of RLE.

B.1 Protocol Type Table

The protocol type table lists the protocol types allowed by an implementation, the mapping between compressed and uncompressed protocol types (if appropriate) and whether a protocol type with a value less than 1 536 designates a real protocol type or an extension header (if appropriate). The protocol type table may be fixed, modifiable (by configuration or signalling) or may be implicit.

Table B.1 provides an excerpt of the DVB-RCS2 protocol type table taken from [2] as an example. Further examples are provided in Annex E for S-MIM.

Table B.1: Protocol type table

Protocol type value	Compressed protocol type value	P/E	Protocol or extension header
0x0000	0x00	E	RFC 4326 Test-SNDU Extension Header [4]
0x0001	0x01	E	RFC 4326 Bridged-SNDU Extension Header [4]
0x0002	0x02	E	RFC 5163 TS-Concat Extension Header [i.1]
0x0003	0x03	E	RFC 5163 PDU-Concat Extension Header [i.1]
0x00C8	0x04	E	EN 301 790 [i.2] V1.5.1 LL_RCS_FEC_FDT Extension Header
0x0100	0x05	E	RFC 4326 Padding Extension Header [4]
…	…	…	…
0x0800	0x0D	P	Ipv4
…	…	…	…
0x86DD	0x11	P	Ipv6
…	…	…	…
0x0082	0x42	P	Internal DVB-RCS2 M&C signalling (L2S) [2]
…	…	…	…
value in adjacent 2 byte protocol type field	0xFF	-	according to protocol type indicated

B.2 Label Type Table

Table B.2 provides an example configuration table which is partly derived from the DVB-RCS2 standard [2] and partly is set up by the NCC to provide optimum performance for the concrete system.

Table B.2: Label type table (transparent star, DAMA context)

Label type	Label length	Label syntax	Protocol type compression	Implied protocol type
0	1	Most significant byte of MAC24	Yes	0x0800
1	3	Destination MAC24	Yes	0x0800
2	0	-	Yes	0x0800
3	0	-	Yes	0x0082

In this configuration label type 0 is used for transport of DAMA data on non-default SVNs. Label type 1 is used for mesh overlay or regenerative mesh data and type 2 for DAMA data on the default SVN. Label type 3 is used for internal system signalling.

Protocol type compression has been enabled by the NCC via signalling. In DVB-RCS2 compression can be enabled or disabled only for all label types together.

The implied protocol type for label type 3 is internal signalling. This assignment is defined in the system specification [2] and cannot be changed by the NCC while the default protocol types for the other three label types has been set to the EtherType for Ipv4 by the NCC under the assumption that the concrete system will be used mainly for Ipv4 traffic. The implied protocol type can only changed for all three label types 0 to 2 together in DVB-RCS2.

Annex C (informative): Reassembly Error Check Examples

This annex provides examples of the error checking mechanism specified in clause 5.2.3.3.

C.1 Error check with sequence number

Figure C.1 provides an example sequence of ALPDUs with sequence numbers showing the processing at the sender and receiver when FPDUs are lost.

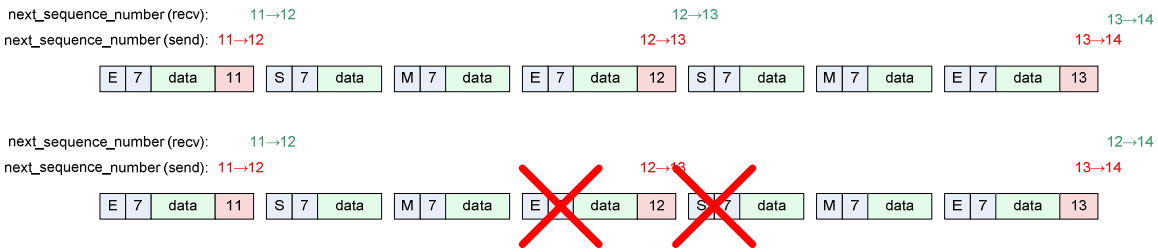

Figure C.1: Error Check with Sequence Number Example

In figure C.2, the fragmentation and reassembly of several ALPDUs using the same `fragment_id` is shown. The upper data flow shows the process without transmission errors and the lower one with two PPDUs lost (as denoted by the red crosses). The red (lower) numbers show the changes of the `next_sequence_number` in the fragmentation process, the green (upper) numbers the changes in the defragmentation process.

In the error-less case the first PPDU shown is the END PPDU of a the first ALPDU. The current `next_sequence_number` in the transmitter is 11 so this value is inserted into the packet trailer and the variable is incremented to 12. The receiver compares the trailer value (11) to its `next_sequence_number` (11). It finds them to be equal and delivers the ALPDU to the next processing step. Then it sets its `next_sequence_number` to the trailer value plus 1 (12). The transmitter then fragments the next ALPDU: a START PPDU, an intermediate PPDU and an END PPDU. It inserts its current `next_sequence_number` (12) into the trailer and increments the variable to 13. The receiver finds the trailer (12) to equal the `next_sequence_number` value (12), delivers the ALPDU and sets its `next_sequence_number` to 13. Then the transmitter processes the next packet the same way resulting in the `next_sequence_numbers` on both sides to become 14.

In the error case a START PPDU and an END PPDU are lost. It is assumed, that the sum of the sizes of the other PPDU payloads happen to sum up to the correct `total_length` value (otherwise the error is detected already by the length check). When the transmitter sends the second END PPDU it inserts 12 into the `sequence_number` field and sets its `next_sequence_number` to 13. This PPDU is lost so the receiver still has value 12. Now the transmitter sends the PPDUs for the ALPDU. The `sequence_number` field now contains 13 and the `next_sequence_number` is 14. The START PPDU is also lost (otherwise the error would be detected earlier). When the END PPDU finally arrives at the receiver it has a `sequence_number` of 13 which is different from the receiver's `next_sequence_number` (12). Therefore the reassembled ALPDU and the received PPDU are discarded and the receiver's `next_sequence_number` is set to the receiver `sequence_number` value plus 1 (14).

C.2 Error check with CRC-32

Figure C.2 provides an example sequence of ALPDUs with CRC-32 showing the processing at the sender and receiver when FPDUs are lost.

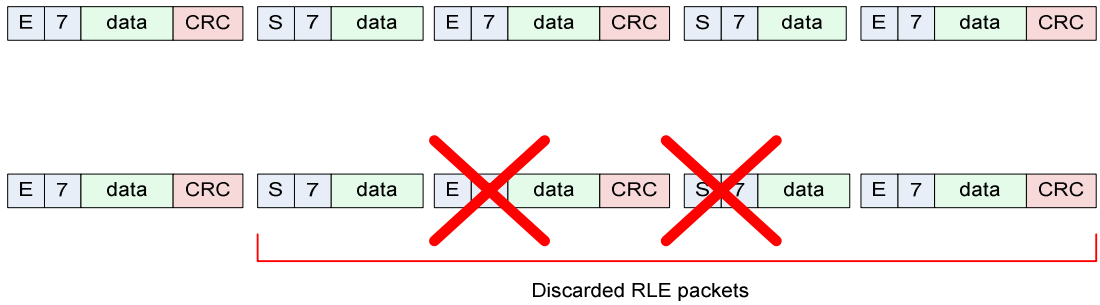

Figure C.2: Error Check with Sequence Number Example

In figure C.2 the fragmentation and reassembly of several ALPDUs using the same `fragment_id` is shown. The upper data flow shows the process without transmission errors and the lower one with two PPDUs lost (as denoted by the red crosses).

In the error-less case the first PPDU shown is the END PPDU of the first ALPDU (it is assumed that the previous START PPDU and intermediate PPDUs have been received correctly). The receiver will calculate the CRC and compare it successfully with the received one in the END PPDU packet. The same will happen with the two following pairs of START PPDUs and END PPDUs, so that the reassembled ALPDUs will be handed over to the decapsulator for further processing.

In the error case, the END PPDU of the second ALPDU and the START PPDU of the third ALPDU are lost. This means, that eventually the reassembly function could wrongly interpret that the consecutively received START and END PPDUs belong to the same ALPDU. If the sizes of the correctly received START and END PPDUs do not match the `total_length` field in the START PPDU, the error would be detected before even calculating the CRC. Otherwise, the CRC mechanism should detect the error by comparing the locally calculated CRC over the reassembled ALPDU with the CRC contained in the `alpdu_crc` field of the END PPDU. If they are not identical, the reassembly function at the receiver will discard both, the START and the END PPDU.

Annex D (informative): Generic Label Type Selection Algorithm

This Annex provides a generic algorithm that selects the most appropriate ALPDU label type and protocol type encoding given an RLE transmitter configuration, an SDU protocol type and an ALPDU label.

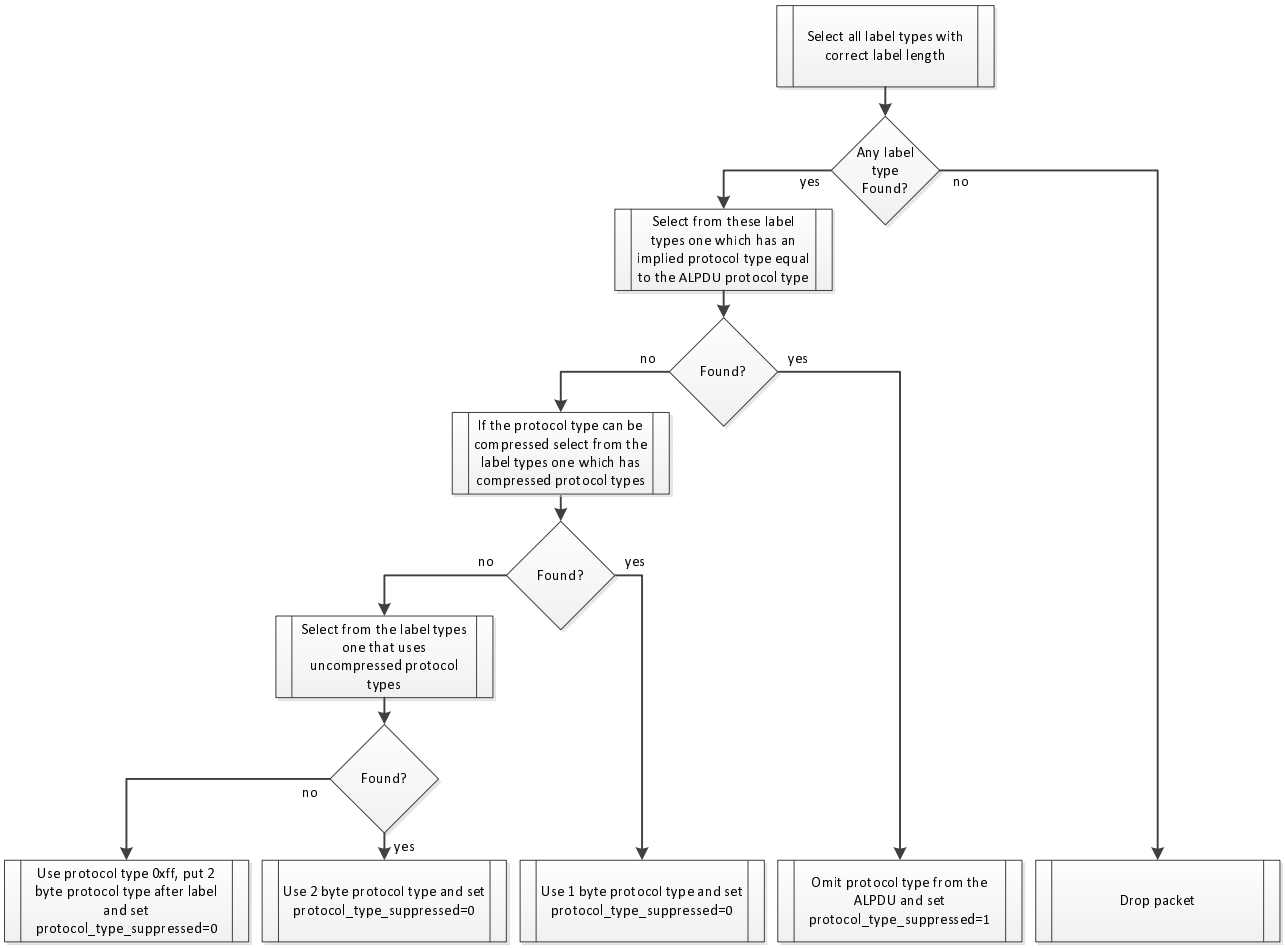

Figure D.1: Label type selection algorithm

The algorithm works as follows:

a) From all four label types a subset S1 is select with all label types that have the correct ALPDU label length. If that set is empty, this ALPDU label length is not supported and the ALPDU can be dropped.

b) Otherwise select from S1 all the label types into subset S2 that have an implied protocol type that equals the protocol type of the ALPDU. If S2 is not empty chose one of the types in S2, set the `protocol_type_suppressed` flag in the PPDU header and omit the protocol type when constructing the ALPDU.

c) Otherwise check whether the protocol type has a compressed variant and if it has select a subset S3 from S1 with all the label types that use compressed protocol types. If S3 is not empty chose one of the types in S3, clear the `protocol_type_suppressed` flag and use the compressed protocol type when constructing the ALPDU.

d) Otherwise select from S1 all label types that use uncompressed protocol types into subset S4. If S4 is not empty chose one of the label types in S4, clear the `protocol_type_suppressed` flag and use the uncompressed protocol type when constructing the ALPDU.

e) Otherwise select one label type from S1, clear the `protocol_type_suppressed` flag, use a compressed protocol type of 0xFF and append the uncompressed protocol type to the ALPDU label when constructing the ALPDU.

Depending on the concrete system specification the algorithm can be simplified. When only uncompressed protocol types are supported by the system the algorithm gets:

a) From all four label types a subset S1 is select with all label types that have the correct ALPDU label length. If that set is empty, this ALPDU label length is not supported and the ALPDU can be dropped.

b) Otherwise select from S1 all the label types into subset S2 that have an implied protocol type that equals the protocol type of the ALPDU. If S2 is not empty chose one of the types in S2, set the `protocol_type_suppressed` flag in the PPDU header and omit the protocol type when constructing the ALPDU.

c) Otherwise select one label type from S1, clear the `protocol_type_suppressed` flag, use a compressed protocol type of 0xFF and append the uncompressed protocol type to the ALPDU label when constructing the ALPDU.

Corresponding simplifications are possible when only a single label length is supported, no implied protocol types exist or only compressed protocol types are available.

Annex E (normative): RLE Example Scenarios

This Annex provides specific configurations for RLE usage in different scenarios and within different standards.

E.1 DVB-RCS2

The configuration of RLE for the DVB-RCS2 transparent star scenario is described in [2] while the transparent or regenerative mesh is not yet completely specified in the ETSI version of the document, but only in the DVB version [i.3].

The selected options for the DVB systems are chosen to optimize performance for the range of standard return link burst sizes (38 to 599 bytes) and for applications ranging from header-compressed VoIP with packet sizes of 10 or even less bytes to full Ethernet frames and to support all available access schemes and system scenarios.

The DVB-RCS2 standard generally describes minimum requirements for the supported configurations. Concrete systems and specification can use additional options and configurations but this requires some form of (administrative) agreement between the terminal implementation and the NCC implementation. This clause describes only the required configurations.

E.1.1 DVB-RCS2 Common Configuration

Clauses E.1.1 to E.1.3 provide the requirements that are common for all DVB-RCS2 scenarios.

E.1.1.1 Maximum SDU Size

The required minimum maximum SDU size shall be 1 500 bytes.

E.1.1.2 Protocol Type Table

The minimum supported protocol types are specified in DVB-RCS2 [2]. Allowance of protocol type compression and the default protocol type (if any) shall be signaled by the NCC to the terminal during the logon process for each transmission context. The default protocol type shall be always enabled for label type 3 and shall be equal to the protocol type for DVB-RCS2 lower layer signaling. Thus signaling packets can be sent with minimum overhead (the label length is 0).

E.1.1.3 Label Type Table

The label types are defined in DVB-RCS2 [2] and are fixed except for the length of the ALPDU label of label type 0. This length is signalled by the NCC during logon and shall be at least 1 byte. This label type shall be used for transparent star traffic. Three byte labels shall be supported by label type 1 but they are used for the transparent mesh and regenerative traffic. Zero byte labels for traffic shall be supported by label type 2 and are used for transparent star traffic on the default SVN.

E.1.1.4 Maximum ALPDU Length

The maximum ALPDU length shall be 4 095 and the `large_alpdus` configuration variable shall have the value "0".

E.1.1.5 Extension Headers

The supported extension headers are explicitly listed in the standard. Extension headers may be fragmented.

E.1.1.6 Integrity Protection

Support of the sequence number is mandatory, support of the CRC method is optional. The methods that a terminal may use shall be signalled by the NCC at logon time. At least one method shall be enabled by the NCC.

E.1.1.7 PPDU Label

The PPDU label length shall be always zero.

E.1.1.8 ALPDU Fragmentation

All ALPDUs may be fragmented if necessary.

E.1.1.9 Payload Header Map

The Payload Header Map shall be omitted. The use of an explicit Payload Header Map may be request by the NCC from the terminal, but the support of it is optional. The implicit Payload Label and the PPDU Label Length shall be signalled by the NCC at logon time but the only required supported value for the PPDU Label Length shall be zero. The required supported values for the Payload Label Length shall be listed in clause E.1.2.2 for the transparent star and in clause E.1.3.3 for transparent mesh and regenerative mesh.

E.1.1.10 Frame protection

The physical layer is assumed to be QEF and no additional frame protection is required.

E.1.2 DVB-RCS2 Transparent Star Configuration

The transparent star is the configuration that is supported by all DVB-RCS2 implementations. There are up to four transmission modes that are related to RLE: DAMA, contiguous carrier, slotted ALOHA and CRDSA with only DAMA being mandatory.

E.1.2.1 Addressing requirements

Although there is only a single physical destination for the return link bursts-the gateway-there may be more than one logical destination by using SVNs (Satellite Virtual Networks). Traffic sent from the terminal for non-default SVNs needs to be tagged with the SVN number. This is done in the ALPDU label by using a one byte SVN tag. For higher efficiency there is the notion of a default SVN for which the tag can be omitted.

For the random access methods the source of the burst shall be identified in the gateway. While DAMA bursts may be identified by processing the burst allocations, random slots shall be identified explicitly. For both transmission modes the payload label contains the source address in form of the group Id and the logon Id.

E.1.2.2 Payload Label

The following payload label formats and lengths shall be supported, given that the corresponding transmission mode is supported:

1) DAMA bursts and contiguous carrier transmission: payload label length shall be zero.

2) Slotted ALOHA: 3 byte payload label; concatenated source Group ID and Logon ID.

3) CRDSA: 5 byte payload label; concatenated source Group ID and Logon ID and CRDSA tag.

E.1.3 DVB-RCS2 Transparent and Regenerative Mesh

A transparent mesh is an overlay onto a transparent star configuration. Bursts sent by one terminal can directly be received by other terminals. This requires receivers for the DVB-RCS2 return link waveform in the terminals. In a regenerative mesh the return link bursts are demodulated and decoded in the satellite and resent on a DVB-S2 forward link. Optionally the higher layer PDUs may also be reassembled and decapsulated.

A transparent mesh system needs to supported the following configuration in addition to the configuration for the transparent star.

E.1.3.1 Addressing requirements

In a mesh each terminal can sent to a potential large number of destinations. In this case the destination and source addresses of the bursts shall be know in the receiving terminals. For the transparent mesh the terminal can deduce the source address from the time-burst plan. In the regenerative case there is only one sender in the downlink that a given terminal can receive-the satellite, so no explicit indication is required. Because of the broadcast nature of the satellite downlink in certain cases it is possible to multiplex higher layer PDUs for several destinations into a single frame, especially for the regenerative mesh with the large DVB-S2 frame sizes. Therefore the ALPDU label shall contain the destination address.

E.1.3.2 ALPDU label

The ALPDU label contains the 3 byte MAC24 destination address. For control packets the label length is zero.

E.1.3.3 Payload Label

The following payload label formats and lengths shall be supported in addition to the transparent star formats for transparent mesh terminals:

1) DAMA bursts for transparent mesh: 2 byte transmitter identification.

The following payload formats and lengths shall be supported by the regenerative mesh terminals:

2) DAMA bursts for regenerative mesh: 2 byte received identification.

E.2 S-MIM

In the S-MIM standard, the RLE protocol shall be used in for both Spead Spectrum Aloha (SSA) [i.4] and the Quasi-Synchronous CDMA (QS-CDMA) [i.5] radio interfaces.

E.2.1 Common Configuration

Clauses E.1.1.1 to E.1.1.10 provide the common RLE configuration settings that are common for both SSA and QS-CDMA radio interfaces as currently defined in [3].

E.2.1.1 Maximum SDU Size

The maximum SDU size shall be 1 500 bytes.

E.2.1.2 Protocol Type Table

The Protocol_Type field shall be used in its compressed mode (1 byte) to identify the protocol of the encapsulated PDU. In particular, the default protocol shall be IPv6. In this case, the `Protocol_Type` field shall be omitted by setting the `Protocol_Type_Suppressed` flag set to 1. Otherwise, the following configuration of the `Compressed_Protocol_Type` field shall be used.

Table E.1

Compressed protocol type value	Protocol
0x00 to 0x2f	reserved
0x30	Ipv4/Ipv6 detected by introspection of the first 4 bits of the first SDU byte
0x31	internal QS-CDMA signalling
0x32	authentication signalling for SSA (LLC-SNAP)
0x33	initial authentication signaling
0x34 to 0x41	reserved
0x42	ROHC bidirectional for QS-CDMA
0x43	ROHC feedback signalling for QS-CDMA
0x44	modified RFC 4944 [6]
0x45 to 0x7f	reserved
0x80 to 0xfe	user defined
0xff	reserved

Table E.2: Compressed protocol type values and supported protocols

Compressed protocol type value	Protocol
0x00 to 0x2F	Reserved
0x30	Ipv4
0x31	Internal QS-CDMA signalling
0x32	Authentication signalling for SSA (LLC-SNAP)
0x33	Initial Authentication signalling
0x33 to 0x41	Reserved
0x42	ROHC bidirectional for QS-CDMA
0x43	ROHC feedback signalling for QS-CDMA
0x44	Modified RFC 4944 [6]
0x45 to 0x7F	Reserved
0x80 to 0xFE	User Defined
0xFF	Reserved

E.2.1.3 Label Type Table

The `alpdu_label_type field`, present in the START PPDU and the FULL PPDU, shall be interpreted as follows:

Table E.3: alpdu_label_type values

Alpdu_label_type value	ALPDU label size (bytes)
0	2 bytes
1	1byte
2	0 bytes
3	0 bytes

E.2.1.4 Maximum ALPDU Length

The Maximum ALPDU length is 4 095 Bytes and the `large_alpdus configuration` variable shall have the value "0".

E.2.1.5 Extension Headers

Extension headers shall not be used.

E.2.1.6 Integrity Protection

`use_alpdu_crc` field shall be set to 1; therefore, the Integrity Protection of a Fragmented ALPDU shall be the ALPDU CRC method. The CRC shall be calculated as defined in clause 4.2.2 of [1].

E.2.1.7 ALPDU Fragmentation

All ALPDUs can be fragmented if necessary. The `fragment_id` field shall be applied to identify fragments of the same packet. When a PPDU is not a FULL PPDU, all fragments corresponding to the same packet are assigned the same `fragment_id`. For the sake of avoiding ambiguities, the RLE transmitter shall not assign the same `fragment_id` to PPDUs corresponding to different packets at the same time. This requirement might limit the transmitter throughput as described next.

Since fragmented PPDUs of a single ALPDU cannot be guaranteed to be received in the same order they were sent if interference cancellation is applied to enhance the demodulation, their order is only uniquely specified if the number of fragments is below 4. If the number of fragments is equal or higher than 4, the continuation PPDUs cannot be sorted with the information contained in the RLE headers. Therefore, if interference cancellation is applied at the SSA demodulator and ALPDUs may be fragmented in more than 3 fragments, the following additional mechanism is required at the receiver to allow unequivocal PPDUs sorting: the RLE transmitter shall limit its rate to N ALPDUs every Δ seconds, where N = 8 is the number of fragment_id values available for encapsulating data and Δ is the difference between the maximum and minimum processing delay at the SSA demodulator.

Moreover, in case an aggressive frequency reuse pattern is applied in the SSA return link (e.g. frequency reuse factor 1 or 2), an additional mechanism shall be present to handle reception of duplicated PPDUs through different demodulators (corresponding to beams or cells using the same frequency slot).

E.2.1.8 Payload Header Map

The Payload Header Map shall be omitted. The required supported value for the Payload Label Length shall be 6 bytes (the source MAC address is transmitted in this field).

E.2.1.9 PPDU Label

The PPDU label length shall be always zero.

E.2.2 RLE Configuration for SSA

Four different protocols shall be encapsulated in RLE to access SSA:

- Ipv6
- Ipv4
- RFC 4944 [6] (header-compressed IP)
- LLC-SNAP (for authentication signalling)

Additionally, SSA requires the reservation of 1 byte for ARQ management, the reservation of 6 bytes to carry the MAC address of the source and CoS differentiation (1 byte). Therefore, the following configuration shall be applied:

- `alpdu_label_type` = 0 (2 bytes long ALPDU label)
- `protocol_type_suppressed` flag = 1 if the SDU to be encapsulated is Ipv6
- `protocol_type_suppressed` flag = 0 if the SDU to be encapsulated is other than Ipv6. In this case, the `compressed_protocol_type` field is present and shall be set according to clause E.2.1.2 , depending on the encapsulated protocol.

The ALPDU label shall carry the following information:

- ARQ (1 byte): information format as specified in clause 8 of [3].
- CoS (1 byte): this field shall be interpreted as follows: 0x01 corresponds to the highest priority. The level of priority decreases as the CoS field value increases, 0xFF corresponds to the lowest priority and 0x00 remains reserved.

Additionally, the following shall be fulfilled at FPDU:

- it shall set the `use_explicit_payload_header_map` to 0;
- the payload_label shall have a fixed length of 6 bytes instead of 8 (this is a fixed configuration to transport the MAC address of the source, as specified in clause 7 of [3]);
- the field `use_frame_protection` shall be configurable. By default, is shall be set to 0, assuming that it is not required to add a CRC in FPDU since the optional CRC defined in clause 5.1.1 of [i.4] will be added by the physical layer. However, if FPDU protection is required, the field `use_frame_protection` shall be set to 1 and the 8 bit CRC specified in clause 5.1.1 of [i.4] shall be used.

E.2.3 QS-CDMA Configuration

Four different protocols shall be encapsulated in RLE to access QS-CDMA:

1) Ipv6
2) Ipv4
3) ROHC - bidirectional mode (header-compressed IP)
4) Internal MAC signalling

QS-CDMA has two types of transport channels, namely DCH and RACH. Different RLE configurations will be applied to access each transport channel.

E.2.3.1 QS-CDMA Configuration for DCH Transport Channel

- `alpdu_label_type` = 1 (1 byte long ALPDU label)
- `protocol_type_suppressed` flag = 1 if the SDU to be encapsulated is Ipv6
- `protocol_type_suppressed` flag = 0 if the SDU to be encapsulated is other than Ipv6. In this case, the `compressed_protocol_type` field shall be present and shall be set according to clause E.2.1.2 , depending on the encapsulated protocol

The ALPDU label shall carry the CoS information (1 byte): this field shall be interpreted as follows: 0x01 corresponds to the highest priority. The level of priority decreases as the CoS field value increases, 0xFF corresponds to the lowest priority and 0x00 remains reserved.

Additionally, the following shall be fulfilled at FPDU:

- it shall set the `use_explicit_payload_header_map` to 0 (since the physical layer header provides means to signal the payload length);
- the field `use_frame_protection` shall be set to 0 since the physical layer CRC is mandatory in [i.5].

E.2.3.2 QS-CDMA Configuration for RACH Transport Channel

- `alpdu_label_type` = 2 (no ALPDU label).
- `protocol_type_suppressed` flag = 0, since the RACH channel is used only for signalling.

- `compressed_protocol_type` = 0x31.

Additionally, the following shall be fulfilled at FPDU:

- it shall set the `use_explicit_payload_header_map` to 0 (since the physical layer header provides means to signal the payload length);
- the payload_label shall have a fixed length of 6 bytes instead of 8 (this is a fixed configuration to transport the MAC address of the source, as specified in clause 7 of [3]);
- the field `use_frame_protection` shall be set to 0 since the physical layer CRC is mandatory in [i.5].

E.3 Other Regenerative Satellite Systems

RLE is designed to support a very broad range of possible regenerative systems. In general there is a large number of options in designing regenerative systems with regard to (at least):

1) The switching layer. On-board switching can take place at the network layer (IP routing) or inside the physical layer (fast burst switching) or at almost any layer or sublayer inbetween. With regard to RLE it is possible to switch on ALPDUs (by means of the ALPDU label), on PPDUs (by means of the PPDU label or cached ALPDU labels) or on FPDUs (by means of the payload label). The label mechanism is flexible enough to put any kind of information required for the on-board switch into one or more labels. The label positions are fixed with regard to their PDU to allow efficient hardware implementation of switching.

2) Stateless/statefull switching. Switching can be connection-oriented, connection-less or cut-through switching. All of these variants can be used with RLE.

3) Address entities and address sizes. There may be different addressable entities: in a system with PPDU switching, for example, the PPDU label may just contain the address of the destination downlink beam (which is expected to be short), while the ALPDU label contains the address of the destination terminal. Because of the wide range of label sizes very small systems and large systems may be supported efficiently. Depending on the communication architecture the labels may be required to carry destination addresses and/or source addresses. This is supported by the flexible labels.

4) Additional information may be required for a given PDU type. This includes, for example, the required downlink modulation and coding codepoint or quality of service tags which otherwise cannot be obtained on-board when, for example, a PDU carries only a fragment of a higher layer PDU.

An example system using connection-less PPDU switching could use the following label configuration:

1) The ALPDU label contains the destination terminal address.

2) The PPDU label contains the destination downlink address, the source terminal address and a QoS codepoint. The source terminal address is required to be able to associate PPDUs for reassembly in the destination terminal. Only the destination downlink address is used by the switch. The QoS codepoint is used by the downlink scheduler.

3) The FPDU label is empty.

For this system efficiency could be enhanced by using the following additional rules:

1) Both the link address and the QoS codepoint could be stripped from the PPDU before sending the PPDU on the downlink (in this case the downlink and uplink RLE configurations are slightly different). Because the PPDU label is not part of the PPDU length field this requires just removing the bytes of the two fields from the label.

2) The source terminal address could instead be contained in the uplink FPDU label and be duplicated by the on-board processor into the PPDU labels of the PPDUs contained in the given FPDU. This also requires no modification to the PPDU length field.

History

Document history		
V1.1.1	August 2013	Publication