ETSI TS 102 821 V1.4.1 (2012-10)

Digital Radio Mondiale (DRM); Distribution and Communications Protocol (DCP)

Reference

RTS/JTC-DRM-27

Keywords

broadcasting, digital, DRM, radio

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

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2012.
© European Broadcasting Union 2012.
All rights reserved.

DECTTM, **PLUGTESTS**TM, **UMTS**TM and the ETSI logo are Trade Marks of ETSI registered for the benefit of its Members.
3GPPTM and **LTE**™ are Trade Marks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.
GSM® and the GSM logo are Trade Marks registered and owned by the GSM Association.

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 Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

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

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

Introduction

A large number of communications protocols have been developed to allow reliable exchange of data using a wide variety of different techniques. Some have relied on two-way communication to allow requests for re-tries of missing or corrupted messages, while others have relied on Forward Error Correcting codes such as Reed Solomon to rebuild the original message. Unfortunately most of the protocols are tightly coupled to the application they were originally developed for, do not scale well in multicast networks or are unsuitable for use over the uni-directional circuits often found in distribution systems. When the development of a distribution protocol for Digital Radio Mondiale broadcasts was considered, none of the available protocols was deemed suitable and so it was decided to develop a general purpose, low-level, reliable communications protocol suitable for both uni-directional and bi-directional data links which would meet the needs of DRM but would also hopefully be flexible enough to meet the needs of other applications as well.

1 Scope

The present document gives the specification for a general purpose distribution link suitable for multicasting data to many recipients using a uni-directional communications network.

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 reference 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] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

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.

Not applicable.

3 Definitions, symbols, abbreviations and conventions

3.1 Definitions

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

AF Packet: collection of TAG Items with a header carrying a cohesive and self-contained block of data

application: any DCP-based protocol (specified outside the scope of the present document)

Application Framing (AF): layer of the DCP providing a logical grouping of a number of TAG Items

byte: collection of 8-bits

Distribution and Communication Protocol (DCP): transport layer communications protocol providing fragmentation, addressing and/or reliable data transmission over errored channels using a Reed Solomon code to provide Forward Error Correction

TAG Item: DCP elemental type combining in a single logical data the name, length and value of the data

TAG Name: name field within an individual TAG Item used to identify an individual piece of information

TAG Packet: collection of TAG Items with a header carrying a cohesive and self-contained block of data

TAG Value: payload of a TAG Item

3.2 Symbols

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

N_x	The value *N* is expressed in radix *x*. The radix of *x* is be decimal, thus $2A_{16}$ is the hexadecimal representation of the decimal number 42.
$\lceil x \rceil$	The smallest integral value numerically greater than *x*. Sometimes known as the "ceiling" function.
$\lfloor x \rfloor$	The largest integral value numerically less than *x*. Sometimes known as the "floor" function.
$\frac{x}{y}$	The result of dividing the value *x* by the value *y*.
$MIN\{a,...,z\}$	The smallest value in the list.

3.3 Abbreviations

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

AF	Application Framing
CRC	Cyclic Redundancy Check
DCP	Distribution and Communication Protocol
DRM	Digital Radio Mondiale
FEC	Forward Error Correction
IP	Internet Protocol
LSb	Least Significant bit
LSB	Least Significant Byte
MSb	Most Significant bit
MSB	Most Significant Byte
MTU	Maximum Transmit Unit
PFT	Protection, Fragmentation and Transportation
RS	Reed Solomon
TAG	Tag, Length, Value
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
LAN	Local Area Network
MDI	Multiplex Distribution Interface
UTF	Unicode Transformation Format

3.4 Conventions

The order of bits and bytes within each description use the following notation unless otherwise stated:

- in figures, the bit or byte shown in the left hand position is considered to be first;
- in tables, the bit or byte shown in the left hand position is considered to be first;
- in byte fields, the Most Significant bit (MSb) is considered to be first and denoted by the higher number. For example, the MSb of a single byte is denoted "b_7" and the Least Significant bit (LSb) is denoted "b_0";
- in vectors (mathematical expressions), the bit with the lowest index is considered to be first.

The order of transmission (MSb-first or LSb-first) is the conventional order for the physical link in use. Where both orders are common, MSb-first is used.

4 General description

4.1 System overview

The Distribution and Communication Protocol is specifically designed to permit reliable multicast communication from a central server to a number of receivers. Errors on the communications link(s) can be detected and corrected using a Reed-Solomon Forward Error Correction code. As a result the communications links employed can be uni-directional, offering a potentially huge cost saving.

NOTE: This general-purpose DCP protocol does not define any transmission timing rules or restrictions. The definition of timing rules is highly application specific and is therefore handled individually within each application protocol definition.

4.2 System architecture

The application data is carried from the server to the receiver through a number of layers as shown in figure 1. The data at each layer is encapsulated in a series of packets. The TAG layer encapsulates the elementary arbitrary length data items, while the AF Layer combines the elementary data into a cohesive block of related data. The optional PFT layer allows fragmentation of the potentially large AF packets, and adds the possibility of having addressing and FEC. The AF Packets or the PFT Fragments can then be transported by any one of a number of physical links, including (but not limited to) asynchronous serial, UDP/IP and even stored as a file on a disc. Some examples of the possible link layer options are shown in figure 2.

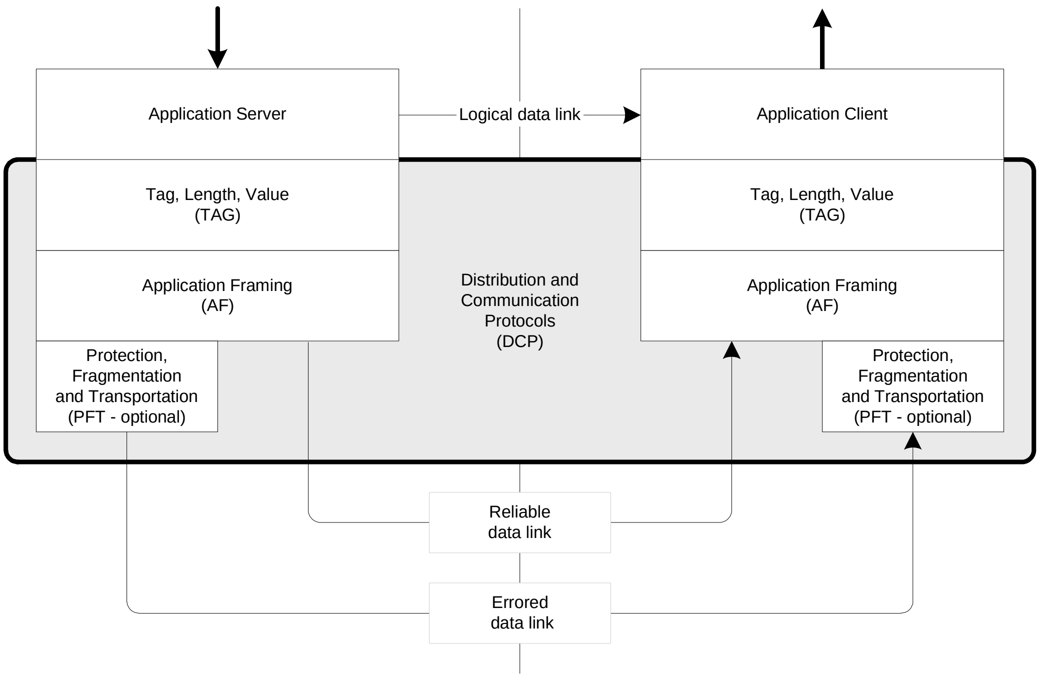


Figure 1: DCP protocol stack

Although shown used with errored data links, the PFT layer is also helpful when using reliable data links which do not provide a transport addressing function.

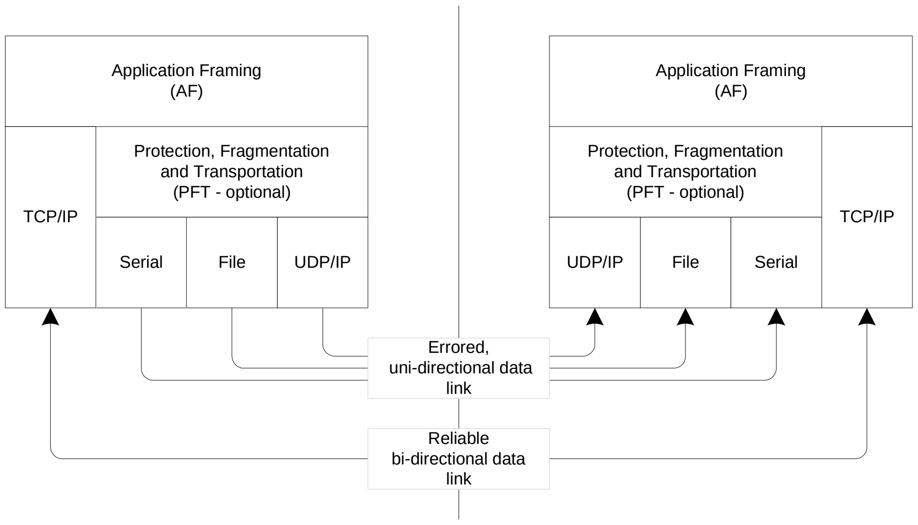


Figure 2: Example DCP link layers

4.2.1 TAG Items and AF packets and PFT fragments

A very brief overview of the structure of the data at the various layers is shown in figure 3.

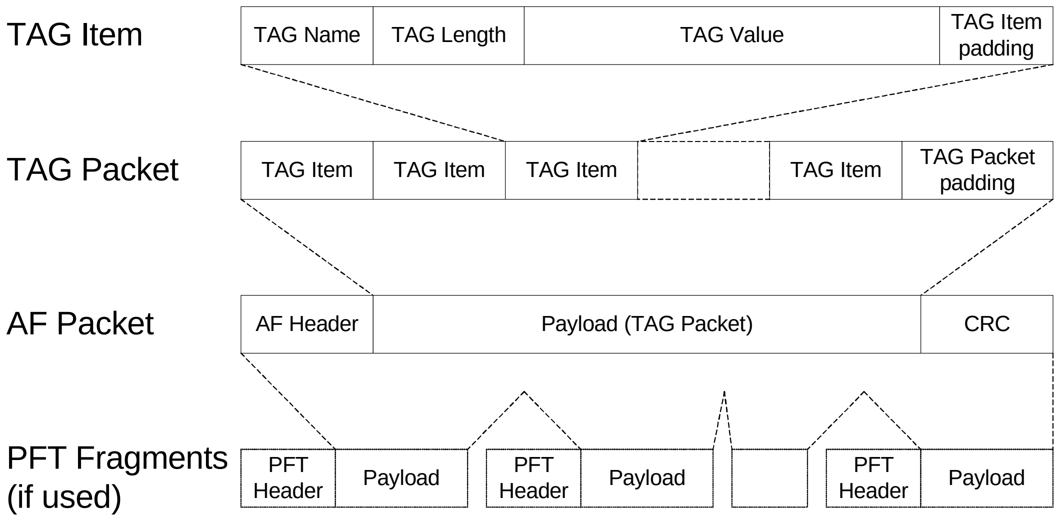


Figure 3: TAG Items, AF packets and PFT fragments

5 TAG Layer

The TAG Layer forms the interface between the application and the DCP. Within the TAG Layer, TAG Items encapsulate individual data items. TAG Items are combined to form in a TAG Packet to form a logically cohesive block of data to the application. Thus to define a new application it is merely necessary to define a series of TAG Items and to impose any necessary limits on the features of the DCP, for example specifying that the PFT Layer shall always be used or that the physical link shall always be Ethernet, etc.

5.1 TAG packet

A TAG Packet is the name given to a cohesive group of TAG Items that have some significance to the overall application. TAG Packets do not contain any synchronization or error correction and so will typically only exist within equipment.

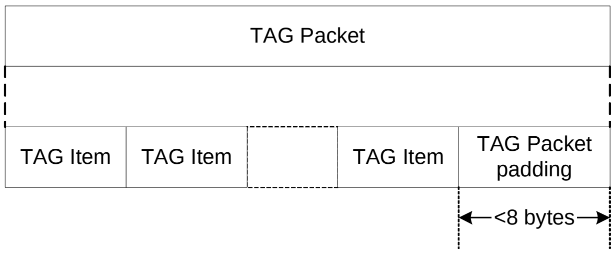


Figure 4: TAG packet

A TAG Packet may include up to 7 bytes of padding after the last TAG Item - the data carried in the padding shall be undefined. Such padding shall be ignored by all receivers. Since the shortest length of a TAG Item is 8 bytes, TAG Packet Padding can be easily identified. If more than 7 bytes of padding are required, the **dmy* special TAG Item shall be used, see clause 5.2.2.2.

It should be noted that the TAG Packet itself has no header, and there is no way of determining the total length of the TAG Items in the packet: these functions are achieved using the AF Layer described later. As a result, the TAG Packet is not a suitable structure for passing between pieces of equipment, but is a convenient abstraction that can be used in implementations if desired.

5.1.1 General rules

The application may determine whether the order of TAG Items within a TAG Packet has any significance. It is very strongly recommended that the order of TAG Items shall not be significant.

The application may determine whether a single TAG Packet may contain multiple TAG Items with the same TAG Name value.

Implementations shall ignore any TAG Items included in a TAG Packet that are not recognized. This allows proprietary extensions to be made to existing protocols in a backwards compatible manner.

TAG Item Name may comprise any four bytes, and need not be restricted to ASCII characters. One general restriction is given in clause 5.2.2. Applications may define additional restrictions.

5.2 TAG item

The structure of a single TAG Item is as shown in figure 5.

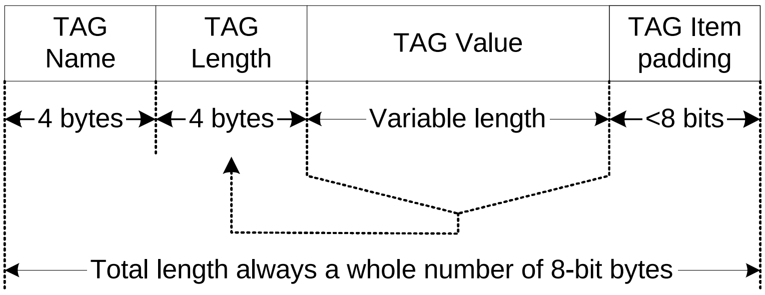


Figure 5: TAG Item

TAG Name: A four-byte name used to identify the data value carried in the TAG Item.

TAG Length: A four-byte value representing the number of bits in the TAG Value field.

TAG Value: Any value as required by the application.

TAG Item padding: Up to seven bits of undefined value as required to make the total length of the TAG Item a whole number of bytes.

5.2.1 Hierarchical TAG items - Coding example

If required by the application, a single TAG Item may encapsulate further TAG Items, as illustrated in figure 6. The depth of the hierarchy may be limited by the application if appropriate.

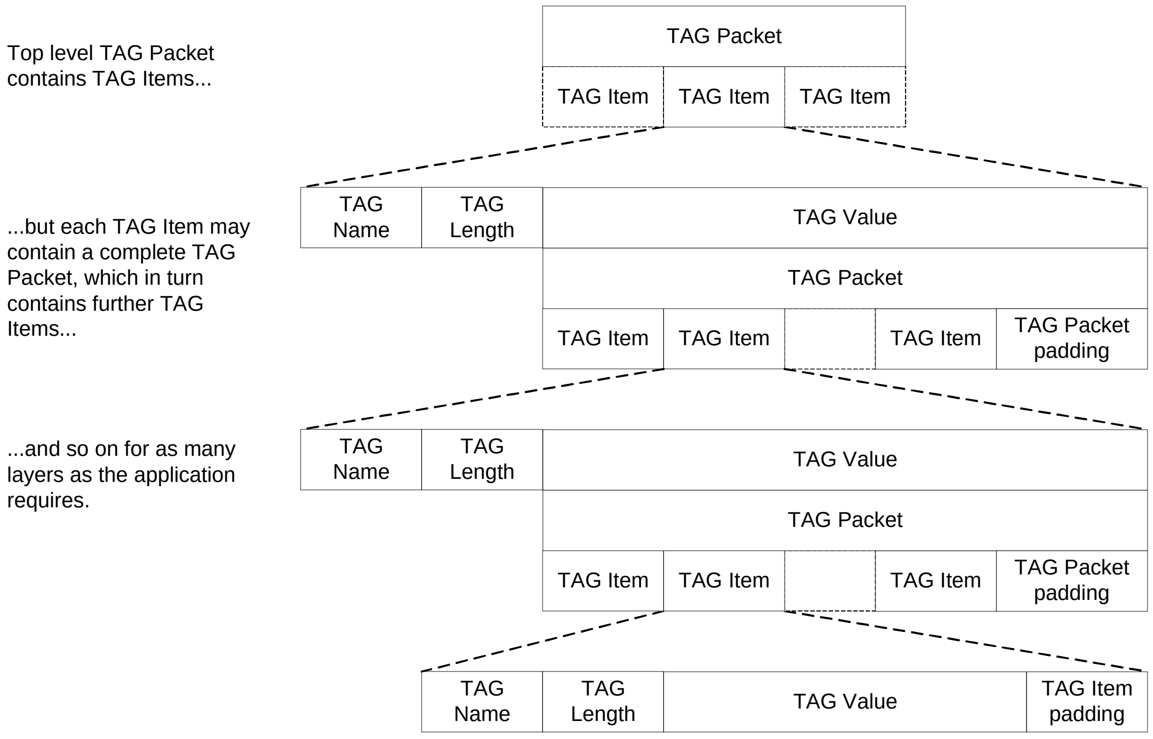


Figure 6: Hierarchical TAG Items

Every lowest-layer TAG Item shall contain TAG Item padding if needed to ensure it is a whole number of 8-bit bytes long. Higher layer TAG Items will therefore never require TAG Item padding.

5.2.2 Special TAG items

Each application is largely free to define TAG Item names as appropriate, the only exception being that all names beginning with the ASCII "*" character, 42 (decimal) or $2A_{16}$, are reserved as control TAG Items.

5.2.2.1 Protocol type and revision, *ptr

It is highly recommended that every application using the DCP should declare a protocol type and revision in every TAG Packet using the *ptr TAG Item as shown in figure 8.

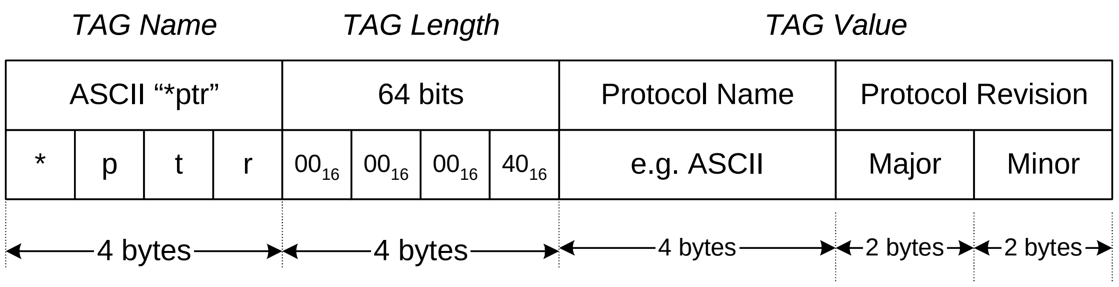


Figure 7: Protocol type and revision

Protocol type: The name of the protocol. Typically this will be encoded using ASCII values in the range 20_{00} to $7F_{16}$, but values outside this range may be used if desired.

Major revision: A binary counter representing the major version number of the protocol, starting from 0000_{16}.

Minor revision: A binary counter representing the minor version number of the protocol, starting from 0000_{16}.

This TAG Item requires no TAG Item padding.

5.2.2.1.1 Revision numbering

Each application is permitted to use any revision numbering scheme as required, however it is highly recommended that the minor revisions are backwards compatible, thus an application implementing version 4.3 of protocol WXYZ should be able to decode versions 4.0, 4.1 and 4.2 in addition to 4.3, but may not necessarily support versions 3.1 or 5.0. Additionally, version 4.5 packets shall be processed as if they were version 4.3 with any new features added in version 4.4 or 4.5 being ignored.

As a general principal, the addition of new TAG Items or the definition of previously reserved bits should be reflected by a minor version number change. Non-backwards compatible re-definition (including lengthening) of existing TAG Items should be reflected by a major version number change.

5.2.2.2 Dummy padding, *dmy

To allow word alignment to be achieved when necessary, the *dmy TAG Item allows more than 8 padding bytes (of undefined data) to be inserted into a TAG Packet: if less than 8 bytes of padding are required, the TAG Packet Padding shall be used instead.

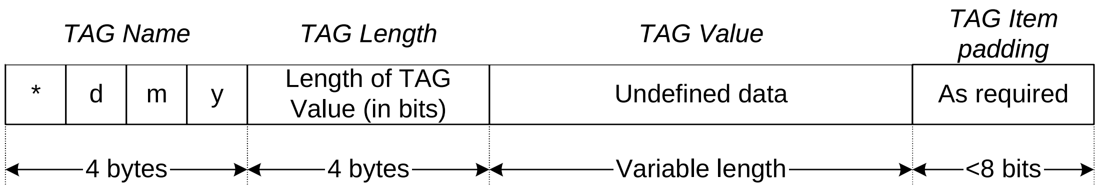


Figure 8: Dummy (padding) TAG Item

6 Application Framing (AF) layer

The AF layer encapsulates a single TAG Packet in a simple structure that is suitable for passing between equipment connected via error-free links. Such links may be provided by existing networks, such as TCP/IP, or by the PFT Layer described in clause 7.

6.1 AF packet structure

The basic structure of an AF Packet is as shown in figure 9.

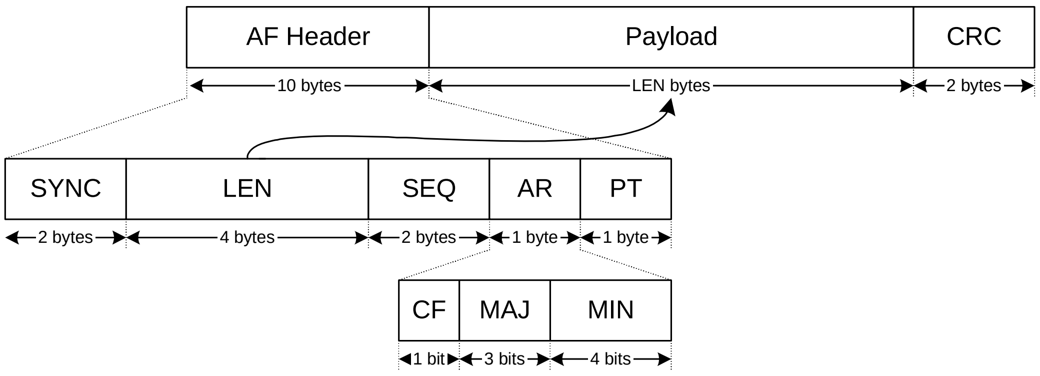


Figure 9: AF Layer

SYNC: two-byte ASCII representation of "AF".

LEN: length of the payload, in bytes.

SEQ: sequence number. Each AF Packet shall increment the sequence number by one for each packet sent, regardless of content. There shall be no requirement that the first packet received shall have a specific value. The counter shall wrap from $FFFF_{16}$ to 0000_{16}, thus the value shall count, $FFFE_{16}$, $FFFF_{16}$, 0000_{16}, 0001_{16}, etc.

AR: AF protocol Revision - a field combining the ***CF***, ***MAJ*** and ***MIN*** fields.

CF: CRC Flag, 0 if the ***CRC*** field is not used (CRC value shall be 0000_{16}) or 1 if the ***CRC*** field contains a valid CRC.

MAJ: major revision of the AF protocol in use, see clause 6.2.

MIN: minor revision of the AF protocol in use, see clause 6.2.

Protocol Type (PT): single byte encoding the protocol of the data carried in the payload. For TAG Packets, the value shall be the ASCII representation of "T".

CRC: CRC calculated as described in annex A over the AF Header and Payload field if the ***CF*** field is 1, otherwise 0000_{16}.

6.2 Revision history

Table 1: Revision history

Major revision	Minor revision	Date	Changes
01_{16}	00_{16}	2003-01-28	Initial public release

7 PFT layer

The optional Protection, Fragmentation and Transport, or PFT Layer provides, as its name suggests, three separate functions. The first is error protection using a Reed-Solomon code that can detect and correct individual bit errors and also rebuild entire lost packets. The second is fragmentation, splitting large packets into smaller units suitable for data links that enforce lower MTU. Finally, the PFT layer allows a limited form of transport addressing so that those lower layers that do not include addressing (for example RS232 serial links) can be used with multiple transport streams. It is not compulsory that all three functions are used simultaneously, and the use of optional header fields minimizes the overhead when a specific feature is not required. The following combinations are permitted:

- Encapsulation.
- Simple fragmentation.
- Reed Solomon FEC and fragmentation.

In addition, each of the above three options can be combined with transport addressing if desired. These options are summarized in figure 10.

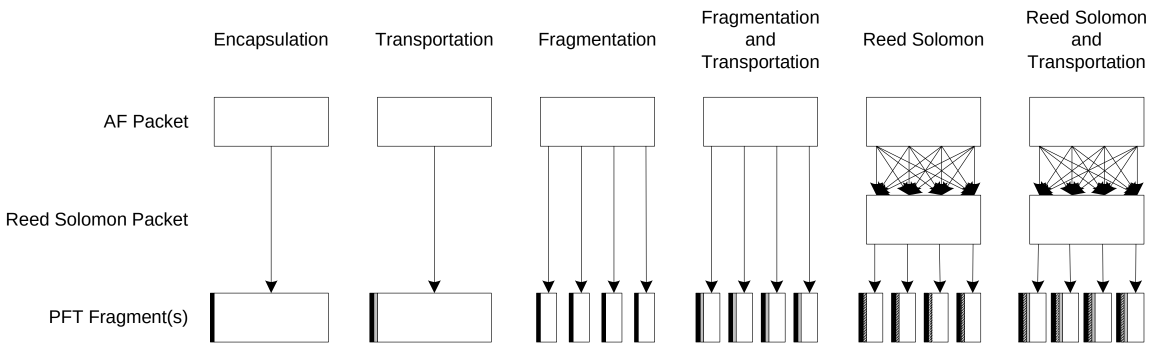


Figure 10: Options available when using PFT Layer

7.1 PFT fragment structure

The structure of a PFT Fragment is as shown in figure 11.

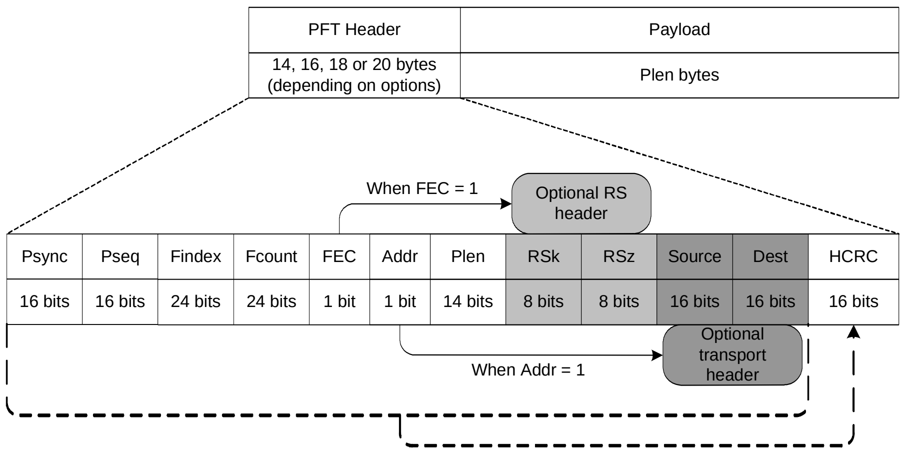


Figure 11: PFT Fragment

Psync: the ASCII string "PF" is used as the synchronization word for the PFT Layer.

Pseq: 16-bit counter incremented by one for each AF Packet. The value shall wrap around from 2^{16}-1 to 0, e.g. …, 2^{16}-2, 2^{16}-1, 0, 1, … The receiver shall not expect a specific value in the first fragment received. The value of the ***Pseq*** field has no link to the value of the AF Packet's ***SEQ*** field.

Findex: 24-bit counter incremented by one for each fragment that forms part of a single AF Packet. The first fragment of each AF Packet shall have the value zero. The value shall not wrap, thus imposing a maximum limit on the AF Packet size that can be carried. The maximum value varies depending on the MTU of the link, but is typically several gigabytes.

Fcount: number of fragments produced from this AF Packet, in the range 1 to 2^{24}-1. The value zero shall not be used.

FEC: when this single-bit flag is set (1), the ***Optional RS Header*** is present.

Addr: when this single-bit flag is set (1), the ***Optional Transport Header*** is present.

Plen: the length, in bytes, of the payload of this fragment.

RSk: the length of the Reed Solomon data word - see clause 7.2.2. Only present when the ***FEC*** field is 1.

RSz: the number of padding bytes in the last Reed Solomon block - see clause 7.2.2. Only present when the ***FEC*** field is 1.

Source: free-format 16-bit source identifier. Only present when the ***Addr*** field is 1.

Dest: free-format 16-bit destination identifier. Only present when the ***Addr*** field is 1.

HCRC: PFT Header CRC calculated over the PFT Header fields from ***Psync*** including any optional headers present. The CRC shall be calculated as described in annex A.

When both ***FEC*** and ***Addr*** are set (i.e. when both optional headers are present), the two headers shall appear in the order indicated in figure 11.

7.2 Definitions

Throughout the following text, the following definitions shall apply.

7.2.1 Known values

- l is the total length of the original AF Packet, including the header and CRC.
- k_{max} is the maximum value of k and has the value 207.
- p is the number of bytes of Reed Solomon parity per chunk and has the value 48.
- m is the maximum number of fragment losses per packet that the Reed Solomon is to be able to recover. When recovery after fragment loss is not required, or when Reed Solomon is not used, m shall be zero. Values of m greater than 5 are not recommended due to the overhead of sending many small fragments.
- MTU is the Maximum Transmit Unit size (in bytes) for the underlying transport layer. When the transport layer has no MTU and when the MTU is greater than 2^{14}, then the value of MTU shall be 2^{14}.
- h is the PFT header length in bytes. The value shall be 12, 14, 16 or 18 bytes depending on the options in use.

7.2.2 Calculated values

- c is the number of Reed Solomon chunks (fixed as zero if Reed Solomon not in use).
- k is the data length of each chunk and is carried in the ***RSk*** field of the PFT Header (zero if Reed Solomon not used).
- z is the number of zero-bytes added to the last Reed Solomon chunk and is carried in the ***RSz*** field of the PFT Header (zero if Reed Solomon not in use).
- s_{max} is an intermediate result representing the maximum payload size, in bytes, for a single fragment.
- f is the number of fragments and is carried in the ***Fcount*** field of the PFT Header.

- s is the actual size of the fragment(s), in bytes.
- L is the length (in bytes) of the packet to be fragmented. When Reed Solomon is used, L has the value $f \times s$, otherwise l.

$$c = \left\lceil \frac{l}{k_{\max}} \right\rceil$$

$$k = \left\lceil \frac{l}{c} \right\rceil$$

$$z = c \times k - l$$

$$s_{\max} = MIN\left\{ c \times \left\lfloor \frac{p}{m} \right\rfloor, MTU - h \right\} \qquad \text{for } m > 0$$

$$s_{\max} = MTU - h \qquad \text{for } m = 0$$

$$f = \left\lceil \frac{l + c \times p + z}{s_{\max}} \right\rceil$$

$$s = \left\lceil \frac{l + c \times p + z}{f} \right\rceil$$

NOTE: The algorithm to calculate $s_{\max}$ (for the case m>0) has been revised compared to previous versions of the present document, see clause G.1.

7.3 Encoding

The following steps shall be completed in order to encode a single AF Packet. In the event that an option (e.g. Reed Solomon) is not enabled, that step is simply not performed.

7.3.1 Reed Solomon

The original packet is first broken down into c Reed Solomon Chunks of k bytes each - z zero-bytes of padding are added to the last chunk if necessary. The Reed Solomon parity bytes are then calculated and appended to each chunk, and the resulting RS Block is interleaved to form an RS Packet. This is shown diagrammatically in figure 12.

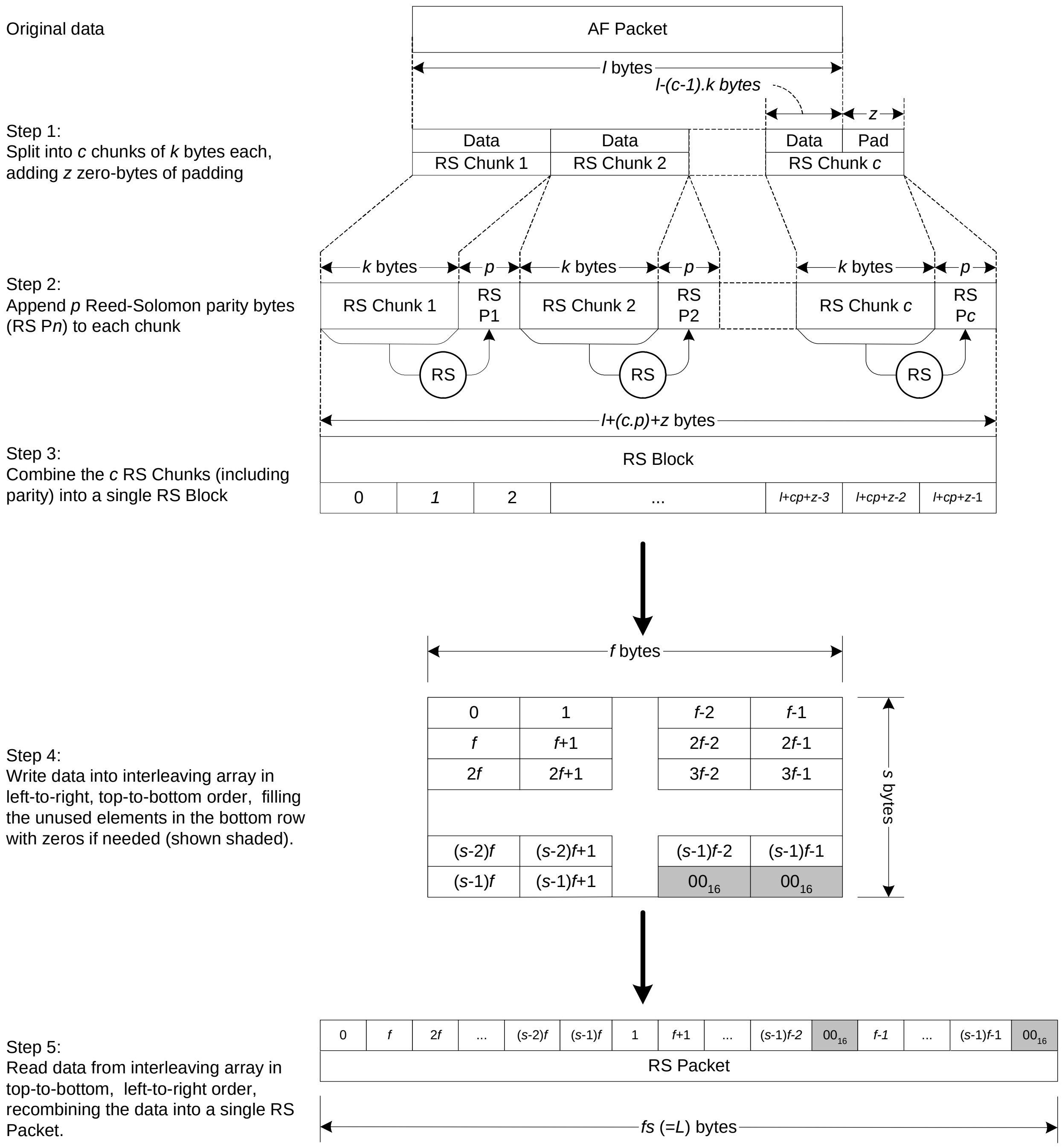

Original data
AF Packet
l bytes
l-(c-1).k bytes
z
Step 1:
Split into c chunks of k bytes each, adding z zero-bytes of padding
Data
RS Chunk 1
Data
RS Chunk 2
Data
Pad
RS Chunk c
Step 2:
Append p Reed-Solomon parity bytes (RS Pn) to each chunk
k bytes
p
RS Chunk 1
RS P1
RS Chunk 2
RS P2
RS Chunk c
RS Pc
RS
RS
RS
l+(c.p)+z bytes
Step 3:
Combine the c RS Chunks (including parity) into a single RS Block
RS Block
0
1
2
...
l+cp+z-3
l+cp+z-2
l+cp+z-1
f bytes
Step 4:
Write data into interleaving array in left-to-right, top-to-bottom order, filling the unused elements in the bottom row with zeros if needed (shown shaded).
0
1
f-2
f-1
f
f+1
2f-2
2f-1
2f
2f+1
3f-2
3f-1
(s-2)f
(s-2)f+1
(s-1)f-2
(s-1)f-1
(s-1)f
(s-1)f+1
00₁₆
00₁₆
s bytes
Step 5:
Read data from interleaving array in top-to-bottom, left-to-right order, recombining the data into a single RS Packet.
0
f
2f
...
(s-2)f
(s-1)f
1
f+1
...
(s-1)f-2
00₁₆
f-1
...
(s-1)f-1
00₁₆
RS Packet
fs (=L) bytes

Figure 12: Generation of Reed Solomon Packet

The full Reed Solomon code used shall be RS(255,207) calculated over the Galois Field GF(2^8) using the generator polynomial $P(x)=x^8 + x^4 + x^3 + x^2 + 1$.

When the calculated value for k is less than 207, bytes k to 206 (inclusive) encoded by the RS(255,207) code shall all be zero and shall not be included in the resulting RS Block, thus producing an RS(k+p,k) code.

The code polynomial shall be $G(x) = \prod_{i=1}^{48} (x - \alpha^i)$.

The k data bytes and p Reed-Solomon check-bytes are mapped to the coefficients of the corresponding polynomials in order of decreasing power of x. If the order of the data bytes in the RS block is d_0 to d_{k-1} followed by the check-bytes rs_0 to rs_{p-1}, then d_0 to d_{k-1} are the coefficients of x^{254} to x^{255-k} respectively and rs_0 to rs_{p-1} are the coefficients of x^{p-1} to x^0 in the codeword polynomial, i.e. the polynomial that has $G(x)$ as a factor.

7.3.2 Fragmentation

Fragmentation may be applied directly to an AF Packet or to a pre-processed RS Packet.

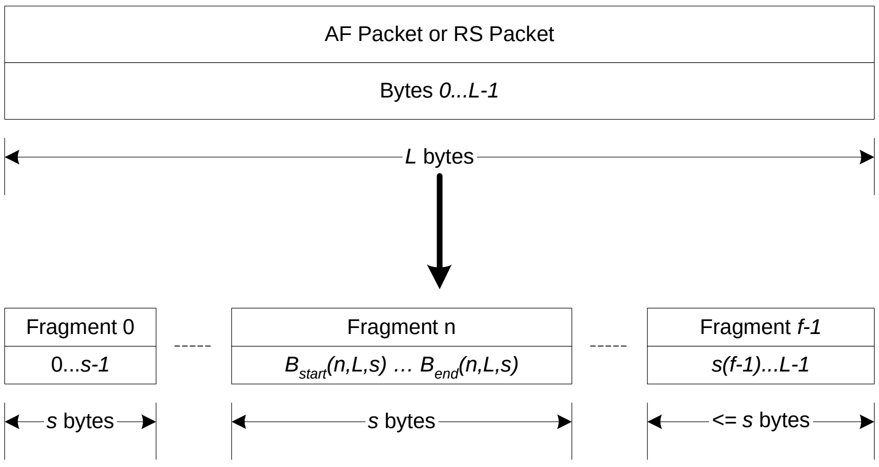
The fragmentation splits the data from the original AF or RS Packet into a number of separate fragments. When carrying an interleaved Reed Solomon Packet, up to m of these fragments can be lost from each packet without losing data. The fragmenting process is shown diagrammatically in figure B.3.

Every PFT Fragment produced from a single AF or RS Packet shall have the same values in all of the PFT Header fields except for the ***Findex***, ***Plen*** and ***HCRC*** fields.

The ***Findex*** field shall contain a count that shall start with zero and increment by one for each fragment.

The ***Plen*** field of all fragments shall be the s for the initial $f-1$ fragments and $s - (L\%f)$ (modulus operator) for the final fragment. Note that when Reed Solomon has been used, all fragments will be of length s.

The ***HCRC*** field shall be calculated correctly for each PFT Fragment.



$$B_{start}(n,L,s) = n.s$$
$$B_{end}(n,L,s) = MIN[s(n+1)-1, L-1]$$

Figure 13: Fragmentation

7.3.3 Transport addressing

The PFT Layer addressing fields ***Source*** and ***Dest*** are intended to be used to identify the sender (***Source***) and recipient (***Dest***) of a packet. The value $FFFF_{16}$ shall be used to indicate "broadcast", all other values indicate a specific address. If a device is configured with specific source and/or destination addresses, it shall ignore all PFT Fragments received with an incorrect non-broadcast address.

7.4 Decoding process

Decoding the PFT Fragments is a four stage process. The four stages are, in order:

- synchronize;
- discard incorrectly addressed fragments (if transport addressing enabled);
- de-fragment (if either Reed Solomon or Simple Fragmentation has been used);
- Reed Solomon error detect and correct (if Reed Solomon enabled).

7.4.1 Synchronization

For streaming links (e.g. asynchronous serial or TCP/IP) the following process shall be used for synchronization of the incoming stream. Synchronization may also be applicable when reading a file.

i) Detect the bit-pattern 0101000001000110_2 (5046_{16}) corresponding to the ASCII sync-word "PF" to find the start of a candidate PFT Header.

ii) Calculate the CRC over the candidate header - this may be performed efficiently by a continuous byte-wide CRC implementation utilizing the fact that the CRC of the PFT Header including the ***CRC*** field itself will result in the constant value $1D0F_{16}$.

iii) Check that the header length matches the options selected: if too short, continue from step (ii), if too long, return to step (i), if correct, synchronization has been achieved and the ***Plen*** field can be used to determine the length of the fragment.

For packet-based links (e.g. UDP/IP), the concept of synchronization is unnecessary as the transport protocol will present complete fragments to the PFT layer. It is only necessary to check that the CRC and length are correct before passing the packet on for further processing. In the event that the CRC or length are incorrect, the packet may be assumed to be corrupted and discarded.

7.4.2 Transport addressing

Each PFT Fragment may contain a source and destination address. If present, the address fields shall be examined and if they do not match the configuration of the unit, the entire fragment shall be discarded.

PFT Fragments which do not contain the optional Transport header shall never be discarded.

7.4.3 Defragmentation

Defragmentation is the reverse process to fragmentation. Memory management is kept simple since it is known that all fragments (except the last one) are the same size. The last fragment will be the same size or smaller than the preceding fragments.

If Reed Solomon is not in use, every fragment need to be received correctly and completely in order to re-construct the original AF Packet.

When Reed Solomon is in use, the Forward Error Correction code can be used to attempt recovery of the original AF Packet before all of the fragments have been received, see clause 7.4.4.

NOTE 1: Not all PFT Fragments need to arrive in order. This behaviour can result from special conditions on the transport layer (e.g. re-ordering of packets sent over public Internet), or can be initiated willingly by the transmitting side by interleaving PFT Fragments belonging to different AF Packets. Interleaving may be applied to help recovery from longer link dropouts.

NOTE 2: PFT Fragments that are identical to previously received PFT Fragments in a currently reassembled AF Packet (i.e. duplicate PFT Fragments) should be silently discarded.

7.4.4 Reed-Solomon decoding

Reed Solomon decoding is the reverse process to encoding.

The length of the RS Packet can be calculated as follows, where:

- f is the number of fragments produced from this packet, carried in the ***Fcount*** header field.
- s is the size in bytes of the PFT fragments as carried in the ***Plen*** header field of all fragments except for the last fragment (where ***Findex*** equals ***[Fcount - 1]***).
- k is the data size in bytes of the Reed Solomon code as carried in the ***RSk*** header field.
- z is the number of Reed Solomon padding bytes as carried in the ***RSz*** header field.
- p is the number of Reed Solomon parity bytes and shall have the value 48.
- c_{max} is the maximum number of Reed Solomon chunks that may have been sent.
- Rx_{min} is the minimum number of fragments which need to be received before Reed Solomon decoding can be attempted. Additional fragments may be required if errors have occurred or if one of the fragments received is the last fragment.

$$c_{\max} = \left\lfloor \frac{f \times s}{k + p} \right\rfloor$$

$$Rx_{\min} = f - \left\lfloor \frac{c_{\max} \times p}{s} \right\rfloor$$

Once Rx_{min} PFT Fragments have been received, the remaining bytes can be filled with zeros and Reed Solomon decoding attempted, with success being indicated by a valid AF Packet being produced with a correct CRC. In the case that bit-errors have occurred, more PFT Fragments will be needed before the original AF Packet can be correctly decoded.

Note that the padding bytes added during the Reed Solomon interleaving process may result in more data being recovered than was originally sent for very large packets. This additional data will be all zeros, and can be differentiated from the z zero bytes added during Reed Solomon encoding using the value of the ***RSz*** header field. The size of the final AF Packet can be determined from the AF ***LEN*** field.

NOTE: The algorithm to calculate $Rx_{\min}$ has been revised compared to previous versions of the present document, see clause G.2.

Annex A (normative): Calculation of the CRC word

The implementation of Cyclic Redundancy Check codes (CRC-codes) allows the detection of transmission errors at the receiver side. These CRC words shall be defined by the result of the procedure described in this annex.

A CRC code is defined by a polynomial of degree *n*:

$$G(x) = x^n + g_{n-1}x^{n-1} + \ldots + g_2x^2 + g_1x + 1$$

with $n \geq 1$

and $g_i \in \{0,1\}, \quad i = 1.....n-1$

The CRC calculation may be performed by means of a shift register containing *n* register stages, equivalent to the degree of the polynomial (see figure B.3). The stages are denoted by b_0 to b_{n-1}, where b_0 corresponds to 1, b_1 to x, b_2 to x^2, b_{n-1} to x^{n-1}. The shift register is tapped by inserting XORs at the input of those stages, where the corresponding coefficients g_i of the polynomial are "1".

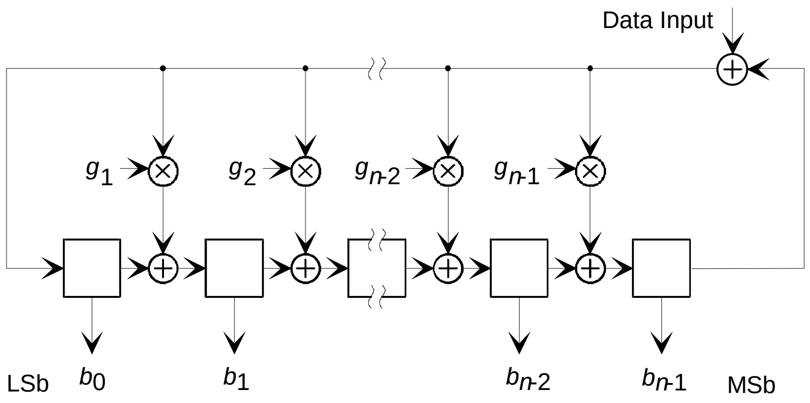


Figure A.1: CRC Generator

At the beginning of the CRC calculation, all register stage contents are initialized to all ones.

After applying the first bit of the data block (MSb first) to the input, the shift clock causes the register to shift its content by one stage towards the MSb stage (b_{n-1}), while loading the tapped stages with the result of the appropriate XOR operations. The procedure is then repeated for each data bit. Following the shift after applying the last bit (LSb) of the data block to the input, the shift register contains the CRC word, which is then read out. The data and CRC words are transmitted MSb first.

The CRC shall be inverted (1's complemented) prior to transmission.

The generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$ shall be used.

If the CRC is appended to the original data, a second CRC calculated over the entire length will result in the constant value $1D0F_{16}$.

Annex B (normative): Physical mapping

The actual transmission of PFT (or AF) Packets shall be possible over as many kinds of existing transmission infrastructure as possible. Three possible physical mappings are defined in this annex, however this list is neither exhaustive nor prescriptive. New mappings may be defined in the future.

B.1 Packet links

Packet switching networks are becoming very common, and in many cases either PFT Fragments or AF Packets may be mapped one-for-one onto link packets. The MTU for the link layer needs to be observed and PFT Fragmentation or PFT Reed Solomon Fragmentation should used to ensure that a single source packet will not be fragmented by the link layer.

B.1.1 UDP/IP

UDP/IP is one of the most popular packet-switching networks at present. Both PFT Fragments and AF Packets may be mapped one-for-one onto UDP/IP packets provided due care is taken to observe the MTU of the UDP/IP layer. PFT Fragmentation or PFT Reed Solomon Fragmentation should be used to ensure that a single source packet would not be fragmented. Any defined IP transport interface for UDP/IP may be used, including (but not limited to) 10Base-T Ethernet or PPP connections.

UDP/IP provides source and destination port numbers, hence the use of the optional PFT Transport Header is not likely to be necessary, however its use is not prohibited.

UDP/IP does not guarantee delivery of packets, hence the use of the PFT Reed Solomon Header is strongly recommended, but is not mandated.

B.2 Streaming links

A streaming connection in this context describes any type of non-packetized connections. Examples of such connections include asynchronous serial links, synchronous serial links and TCP/IP links. The distinguishing feature of such links is that the higher level layers (e.g. AF or PFT layer) is required to establish packet-synchronization before attempting to decode the stream.

The use of the PFT layer is strongly recommended as this has been designed to offer improved synchronization reliability compared to the raw AF layer, however the raw AF layer may be used directly if required.

For physical links which do not guarantee error-free reception, it is recommended that the optional Reed Solomon FEC mechanism provided by the PFT Layer is used. Links such as TCP/IP are guaranteed to provide in-order error-free links and so do not need the Reed Solomon protection, however RS-232 links may be subject to errors and hence benefit greatly from the addition of Reed Solomon FEC.

B.3 File

PFT Fragments or AF Packets may be stored in a file for offline distribution, archiving or any other purpose. A standard mapping has been defined using the Hierarchical TAG Item option available, however other mappings may be defined (or this one extended) in the future. The top level TAG Item has the TAG Name *fio_* and is used to encapsulate a TAG packet, one part of which is the AF Packet or PFT Fragment in an *afpf* TAG Item. Additional TAG Items have been defined to monitor the reception or control the replay of the packets.

B.3.1 File IO (fio_)

The fio_ TAG Item is the highest layer TAG Item in the file TAG Item hierarchy.

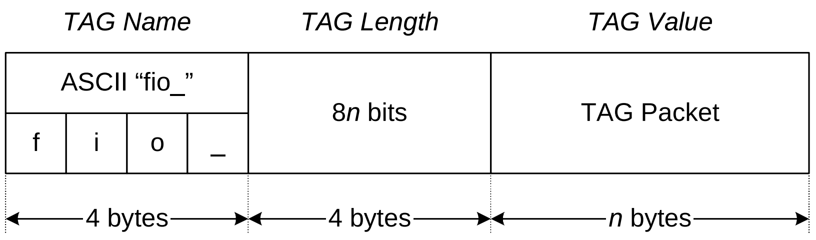


Figure B.1: File Input/Output

This TAG Item acts as a container for an *afpf* TAG Item. A *time* TAG Item may optionally be present.

B.3.1.1 AF Packet/PFT Fragment (afpf)

The *afpf* TAG Item contains an entire AF Packet or PFT Fragment as the TAG Value.

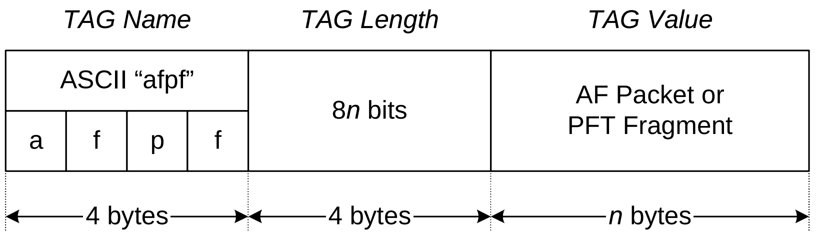


Figure B.2: AF Packet or PFT Fragment

B.3.1.2 Timestamp (time)

The *time* TAG Item may occur in the payload of any *fio_* TAG Item. It may record the time of reception of the payload, or it may indicate the intended time of replay. The time value given in the TI_SEC and TI_NSEC fields may be relative to the start of the file, or any other reference desired.

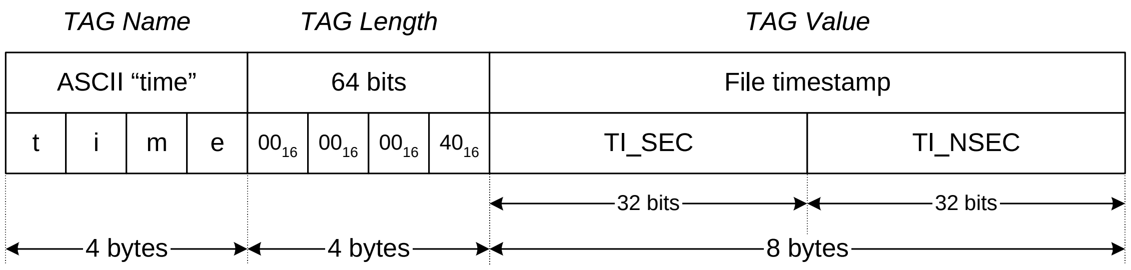


Figure B.3: File timestamp

TI_SEC: the number of whole SI seconds, in the range 0 to (2^{32}-1).

TI_NSEC: the number of whole SI nanoseconds, in the range 0 to 999 999 999. Values outside of this range are not defined and shall not be used.

Annex C (informative): Signalling of basic transmission layers and parameters

Since several basic transmission protocols for PFT (or AF) Packets are defined, it is sensible to also define a common method for addressing and identifying each of these protocols together with their individual parameters. These informative definitions are intended as suggestions for organizations using the DCP protocol to be able to describe DCP sources and targets (DCP protocol addresses) in a common way.

The following definitions are in the general style of a uniform resource identifier (URI) as defined in RFC 3986 [1] but are not entirely consistent with this definition and should not be syntactically validated against the URI specifications. Semantically also the strings specified in this annex do not identify unique resources; rather they provide one possible specification for an application to be informed of the required DCP protocol elements to implement on a given communications link.

The syntax used to describe the DCP protocol addresses is as follows:

- items in square brackets identify optional elements;
- items in angle brackets identify named elements;
- scheme strings and parameter names/values should be treated as case-insensitive;
- most parameters are optional; if such a parameter is not specified, the default value is assumed; if no (default) value is specified, the parameter is NOT optional;
- additional parameters may be defined by applications using the Distribution and Communication Protocol (e.g. maximum transmission time or maximum number of bytes recorded);
- parameters can appear in any order in the parameter section (see below);
- unknown parameters and their values are ignored and may be reported to the user; unknown values of known parameters should be reported to the user.

General DCP protocol address layout:

- <scheme>:<target>[:[<src-addr>:]<dst-addr>]] [?<param>=<val>[&<param>=<val>[&…]]].

General items:

- <scheme> specifies the basic protocol to be used, for example "*dcp.udp.pft*" or "*dcp.tcp*". All DCP-based schemes begin with the string "dcp.". If ".pft" is the suffix of the <scheme> string, the PFT protocol is used together with the specified basic transport protocol; if absent, AF Packets are directly transmitted using the specified basic transport protocol;
- <target> has different precise semantics depending on the scheme used.
  Since the colon character (":") is used as item delimiter, it should not be used as part of the <target> string;
- <src-addr> and <dst-addr> have different precise semantics depending on the scheme used.

NOTE: DCP protocol addresses which omit the src-addr component are syntactically compliant with the URI specification in RFC 3986 [1].

General AF Layer parameters (available identically for all basic transport layers):

- "crc" (disabled: "f", "false", "0"; enabled: "t", "true", "1"; default: "1") enables or disables the CRC calculation for the AF Layer.

General PFT Layer parameters (available identically for all basic transport layers if and only if the <scheme> section contains the suffix ".pft"):

- "saddr", "daddr" (both default to 0). If either or both "saddr" and "daddr" parameters are present, then the "saddr" and "daddr" parameters are carried in the address header of the PFT Packet Header. If both are omitted, it is recommended that the optional address header is not included in the PFT Packet Header. For backwards compatibility, if neither "saddr" nor "daddr" are present, an application may send the address header using the values in the <src-addr> and <dst-addr> sections, or 0 if the <src-addr> and/or <dst-addr> sections are absent;
- "fec" (disabled: "0"; enabled: "sp", "1"…"9"; default: "0") enables/disables the FEC protection mechanism of the PFT Layer; value "sp" indicates the "single packet FEC mode" (FEC enabled, but no FEC driven fragmentation; however, fragmentation may still result from "maxpaklen" parameter!); values 1..9 define the strength of the protection (may e.g. indicate the number of recoverable fragment losses);
- "maxpaklen" (positive integer number; no limit: "0"; default: "0") specifies the MTU of the link for the PFT layer in bytes.

DCP will often be used to communicate the same information over multiple instances of transports. For example, the same data stream may be sent to one destination via a serial port and to another using UDP/IP. Even over one transport type, for example UDP/IP, one connection may be over a local LAN and be highly reliable whilst another may be over the public internet and require forward error correction. If it is desired to support different possible AF CRC and/or PFT options for the same logical DCP connection then these are specified as separate DCP addresses at all connecting entities. Currently there is no protocol defined within DCP to negotiate a specific AF/PFT profile at the time of establishing the connection.

C.1 DCP over UDP/IP

This DCP protocol address format supports both multicast and unicast delivery. The DCP protocol address relates to the destination of a DCP transmission, i.e. the target host for unicast and the target multicast group for multicast, or to the local reception of a DCP transmission.

- <scheme> has the value "dcp.udp[.pft]".
- <target> is a hostname or IP address (as defined in clause 3.2.2 of RFC 3986 [1]) preceded by "//".
  For compatibility, applications are not required to support the IP-literal syntax.
- <src-addr> is the UDP/IP port number at the source host in the range 0…(2^{16}-1).
  If <src-addr> is omitted or "0" then the application is free to use any port number.
- <dst-addr> is the UDP/IP port number at the destination host in the range 0…(2^{16}-1).
  It is not possible to omit the <dst-addr>.
  0 is a valid port number according to the UDP specification but is practically excluded on real systems.

If the DCP protocol address describes the local reception parameters of a DCP transmission, <target> identifies the multicast group address in case of multicast reception or the local host ("localhost", "127.0.0.1", etc.) for unicast reception; <dst-addr> identifies the local port number which is used to receive incoming UDP packets.

The following additional parameters are defined for this scheme:

- "interface" (ip address or system specific device name, e.g. "192.168.0.2", "eth0"; default: use system routing tables). This parameter indicates that the platform routing tables should be bypassed and the datagrams is sent out or received via the specified network interface. Specifically for multicast addresses it relates to the IP_MULTICAST_IF socket option.
- "ttl" (numeric, 0 (restricted to same host) to 255 (unrestricted); defaults to the platform default). Defines the value of the IP_MULTICAST_TTL (time-to-live) socket option. Meaningful for multicast only, and relevant if administrative scoping is not defined for the specified multicast group.

EXAMPLE 1: dcp.udp.pft://192.168.0.1:3002?fec=9&crc=0&saddr=7&daddr=6.

EXAMPLE 2: dcp.udp://224.10.1.20:3002?interface=192.168.0.2.

EXAMPLE 3: dcp.udp://transmitter2.drm.org:1234:3114.

EXAMPLE 4: dcp.udp.pft://192.168.0.1:3002?fec=sp&crc=0.

C.2 DCP over transparent (serial) links

- <scheme> has the value "dcp.ser[.pft]".
- <target> is a system specific device identifier, e.g. "COM4" or "/dev/ttyS1".
- <src-addr> has the value of the PFT Packet Header field SRC, in the range 0…(2^{16}-1).
- <dst-addr> has the value of the PFT Packet Header field DST, in the range 0…(2^{16}-1).

The following additional parameters are defined for this scheme:

- "bitrate" (numeric, e.g. "115200").
- "flowctrl" ("xonxoff", "rtscts"/"hw" or "none" (default)).

EXAMPLE 1: dcp.ser.pft:/dev/ttyS3:1:2?bitrate=4800&fec=4&flowctrl=hw.

EXAMPLE 2: dcp.ser:COM2:200?bitrate=115200.

C.3 DCP to/from a file using File IO

- <scheme> has the value "dcp.file[.pft]".
- <target> is a system specific file name, including any path required.
  Since the colon character (":") is used as item delimiter, it should not be used as part of the <target> string.
  A scenario where it cannot be omitted are DOS compliant path strings including a drive letter ("C:\temp\test.mdi"). In this particular case the colon character should always be followed by a non-numeric character (e.g. a backslash character or the non-numeric start character of a file name).
  In case of ambiguous statements like "C:5:6" (where the parser cannot clearly decide whether "C" is the file name, "5" the source- and "6" the destination address, or whether the source address is omitted and the file name is "5" on drive "C:"), the parser assumes that both the source and destination address are defined.
  If a statement may become ambiguous, the use of the PFT Layer's "saddr" and "daddr" parameters are recommended instead of the <src-addr> and <dst-addr> address items.
- <src-addr> has the value of the PFT Packet Header field SRC, in the range 0…(2^{16}-1).
- <dst-addr> has the value of the PFT Packet Header field DST, in the range 0…(2^{16}-1).

EXAMPLE 1: dcp.file:/tmp/record_1/test.dcp.

EXAMPLE 2: dcp.file.pft:c:\temp\test.dcp:99:100.

EXAMPLE 3: dcp.file.pft:\\myhost\myshare\temp\test.dcp?saddr=99.

C.4 DCP over TCP/IP

For DCP over TCP/IP there is an (initiating) active end (a "client") and a passive end (a "server" waiting for connection requests). The passive end may optionally support multiple DCP sessions over TCP/IP sequentially or simultaneously. The DCP protocol address format defines the passive end of the connection either locally (on the "server") or remotely (remote address a "client" can connect to reach the "server").

- <scheme> has the value "dcp.tcp[.pft]".
- <target> is a hostname or IP address (as defined in clause 3.2.2 of RFC 3986 [1]) preceded by "//".
  For compatibility, applications are not required to support the IP-literal syntax.
  If the <target> defines the local reception parameters of a DCP transmission (on the "server"), it describes the local host ("localhost", "127.0.0.1", etc.).
- <src-addr> is the TCP/IP port number at the active host (the "client") in the range 0…(2^{16}-1).
  If specified and non-zero at the active host (the "client") then this port number is used as the local outgoing port number for the connection; if <src-addr> is omitted or zero then the application is free to use any port number.
  If specified at the passive host (the "server") then requests with other source port numbers will be rejected; a value of zero is equivalent to omitting this parameter.
- <dst-addr> is the TCP/IP port number at the passive host (the "server") in the range 0…(2^{16}-1).
  0 is a valid port number but is practically excluded on real systems.
  If specified and non-zero at the active host (the "client") then connections should be initiated towards this port number.

  If specified at the passive host (the "server") then the host listens for incoming connections on this port.

The following additional parameters are defined for this scheme:

- "interface" (ip address or system specific device name, e.g. "192.168.0.2", "eth0"; default: listen on all interfaces for incoming connections). This parameter is only relevant for the passive host (the "server"). If it carries the address of one of the interfaces of a host with multiple interfaces then the host should listen only on that interface.

EXAMPLE 1 (on the "server"): dcp.tcp://localhost:3002?interface=eth0.

EXAMPLE 2 (on the "client"): dcp.tcp://yourserver:3002.

Annex D (normative): DCP Profiles and DCP Parameters

Not all areas of use and thus not all implementations of the Distribution & Communication Protocol may need to support all functionality provided by the DCP. Therefore this annex lists the three available levels of DCP feature support, called DCP Profiles A, B and C.
The definition of DCP Profiles is a convenient way to clearly indicate which functionality is provided by a specific DCP implementation.

In addition, optional DCP Parameters provide an easy way to further detail some implementation aspects of a particular DCP implementation.

The DCP Profiles and DCP Parameters listed below apply to both DCP encoding and DCP decoding implementations.

Any DCP encoding or decoding implementation NOT supporting DCP Profile A (full support of all DCP features) shall state this fact!

D.1 DCP Profile definitions

The following DCP Profiles A to C are defined.

D.1.1 DCP Profile A

The DCP implementation supports all features provided by DCP:

- AF layer.
- PFT layer with addressing, fragmentation and forward error correction.

This is the default DCP Profile and implicitly assumed if not indicated otherwise by a specific DCP implementation.

D.1.2 DCP Profile B

The DCP implementation supports the following features provided by DCP:

- AF layer;
- PFT layer with addressing and fragmentation.

The DCP implementation does NOT support the following DCP features:

- PFT layer with forward error correction.

A DCP decoding implementation supporting Profile B shall be able to successfully decode a PFT protected AF packet with forward error correction applied to it, as long as all PFT fragments could successfully be received.

D.1.3 DCP Profile C

The DCP implementation supports the following features provided by DCP:

- AF layer.

The DCP implementation does NOT support the following DCP features:

- PFT layer (neither addressing or fragmentation nor forward error correction).

D.2 DCP Parameter definition

The following DCP Parameters may be used to optionally describe some implementation aspects of a specific DCP encoder or DCP decoder implementation in more detail. These DCP Parameters, if stated, typically accompany the specification of the supported DCP Profile.

NOTE: If a DCP implementation is more restricted regarding a particular parameter than the maximum technical restriction imposed by the DCP protocol (where mentioned below), it should state this fact.

D.2.1 AFRevision

This parameter indicates the major and minor revision number of the protocol supported by the DCP encoder or DCP decoder implementation in the form "<major>.<minor>" (numeric values).

For DCP decoders:
If the DCP implementation supports a range of revisions, it may indicate this by using the "-" sign: "<major_first>.<minor_first>-<major_last>.<minor_last>" and/or by replacing the <minor> revision number by an "x" character (which indicates that at least <minor> revision 0 of the particular <major> revision number is fully supported).

Note that higher <minor> revisions than indicated for the same <major> revision number are automatically supported (downward compatibility).

EXAMPLE: AFRevision 1.x-2.x

D.2.2 AFMaxLen

This parameter describes the maximum byte length of the (AF) payload that can be handled by the DCP implementation (excluding the AF header).

NOTE: The maximum size of the AF payload that can technically be signalled in the AF header LEN field is 2^{32}-1.

D.2.3 PFTMaxLen

This parameter describes the maximum byte length of the payload of a PFT Packet that can be handled by the DCP implementation.

NOTE: The maximum size of the PFT payload that can technically be signalled in the PFT header PLEN field is 2^{14}-1.

D.2.4 PFTMaxFragCnt

This parameter describes the maximum number of PFT fragments per AF Packet that can be handled by the DCP implementation.

NOTE: The maximum number of PFT fragments that can technically be signalled in the PFT header is 2^{24}-2.

D.2.5 PFTMaxAFFragCache

This parameter only applies to DCP decoding implementations.

It describes the maximum number of AF Packets of which PFT fragments can be handled simultaneously when received out of order.

Annex E (informative): DCP implementation hints

This annex gives some hints regarding implementation details and error behaviour of DCP decoder implementations.

E.1 Application interface

The interface between a DCP implementation and the application using this DCP implementation is not specified within the scope of the present document.

It is up to the DCP implementation to provide a sufficiently featured and documented interface to the application. The term "sufficiently" may vary strongly depending on the individual requirements of a particular application type and environment.

- Typically a **DCP encoding implementation** will accept the data to be transmitted along with required parameters and target addresses, build the required AF, PFT and probably FileFraming packets and finally output these resulting packets via the specified basic transmission layer.
- A **DCP decoding implementation** may typically handle DCP Packets received via a specified basic transmission layer and forward successfully received AF Packets (or their content plus additional management information respectively) to the application.
  In most cases the AF layer decoding (AF header, CRC calculation, etc.), the PFT layer (re-ordering, FEC decoding/correction, etc.) as well as the FileFraming layer (if applicable) will fully be handled internally by the DCP implementation.

E.2 Reordering of AF Packets

Reordering of received AF Packets may typically not be handled by the DCP decoding implementation (in contrast to PFT fragment reordering). Instead, the DCP implementation may choose to forward any successfully received AF Packet's content (along with the AF Packet's sequence number and other relevant management information) immediately to the application.

This behaviour is recommended due to the fact that some applications may depend on real time input processing and may therefore consider it more appropriate to replace missing AF Packets immediately than to introduce any delay by waiting for a later arrival of missing AF Packets.
Even if a particular application is able to handle a certain amount of AF Packet reordering (e.g. due to input buffering), it is up to the application to decide about the size of the input buffer and related parameters.

Leaving the handling of AF Packet reordering to the application allows a more general DCP decoding implementation to be used in combination with any application type - both real time applications with/without input data buffering or non-real time applications.

E.3 Error behaviour

This DCP specification does not include any rules for handling error situations.

- In a **DCP encoding implementation**, error situations will typically be reported to the application (invalid content, invalid target parameters, target not reachable, etc.).
- In a **DCP decoding implementation**, it is up to the particular implementation to implement (and document) its error behaviour and how error information is forwarded to the application.
  Typically a DCP decoding implementation will report errors related to the basic transmission layer, which was indicated by the application, back to the application (e.g. "serial port cannot be opened", "invalid UDP port number", etc.).

  However with respect to handling received input data via a successfully operating basic transmission layer, a DCP decoding implementation may choose to only forward successfully decoded AF Packets (or their content plus additional management information respectively) to the application and simply discard any invalidly formed data (which may in fact not even be related to DCP) without any further notice.

Annex F (informative): DCP bi-directional communication framework

This annex outlines a proposal for a general-purpose bi-directional communication framework based on the Distribution and Communication Protocol (DCP).

NOTE 1: The framework for a DCP framework for bi-directional communication presented in this clause is only **informative** and available to be referred to and used by any DCP based application.
However, any application based on DCP may choose to implement its own handling of bi-directional communication via DCP, just rely on the basic transmission layer in use (e.g. TCP/IP), or extend this framework definition according to its individual requirements.

NOTE 2: The framework for bi-directional communication based on DCP does not give any recommendations regarding **timing** (re-transmission of un-answered or un-confirmed requests, maximum timeout between request and arrival of answer, etc.).
The definition of timeouts is highly dependent on these particular application and environment. Since the DCP framework for bi-directional communication is suitable for a wide range of applications, the definition of timeouts is left to the specification of a particular application making use of this DCP framework for bi-directional communication.

F.1 Typical Communication Requirements

This clause outlines some general approaches for DCP based communication on which the following DCP framework for bi-directional communication is based.

F.1.1 Transactions

The communication approach is **transaction** based. A transaction comprises:

- one **Request Message** (a special DCP Packet) sent from Client to Server;
- a variable amount of **Response Messages** (special DCP Packets) sent from Server to Client, referencing the Request Message.

F.1.2 Classes of transactions

- **Send Transaction:**
  A Client sends information to a Server.
- **Fetch Transaction:**
  A Client requests information from a Server.
- **Subscribe/Unsubscribe Transaction; Delivery Transaction:**
  A request for server initiated information delivery.
  A Client subscribes to information offered by a Server;
  the Server henceforth delivers information to the Client whenever (new) data is available.
  The Server stops data delivery on explicit Unsubscribe command or for example on missing re-subscription (timeout).

F.1.2.1 Send Transaction

Typical examples for Send Transaction types:

- commands:
  set frequency, shut down in 5 minutes, water the flowers, etc.
- notifications:
  new user just logged in, fire on fifth floor, self destruction is activated and cannot be stopped.

Typical communication flow examples:

- Client sends command to Server and does not request any confirmation; Communication is finished immediately, successful command reception cannot be verified.
- Client sends command to Server and requests reception confirmation;
  Server sends reception confirmation notification.
- Client sends command to Server and requests processing confirmation; Application on server processes command and notifies client.

Potential response types:

- no response;
- confirmation of reception of Request Message;
- notification of processing statuses by Server (command accepted, command processed, execution error/OK, etc.) - available statuses defined by application definition.

F.1.2.2 Fetch Transaction

Typical examples for Fetch Transaction types:
get current frequency, get list of available services, get reception quality indicator, etc.

Typical communication flow examples:

- Client requests data from Server which is immediately available.
- Client requests data from Server which requires action (pre/post processing) by the Server.

Potential response types:

- confirmation of Request Message reception;
- notification of processing statuses by Server (command accepted, command processed, execution error/OK, etc.) - available statuses defined by application definition;
- delivery of requested information.

F.1.2.3 Subscribe/Unsubscribe Transactions

Typical examples for Subscribe Transaction types:
notify me about all future user logins; deliver data stream X

Typical communication flow examples:

- Client subscribes to service X; Server automatically delivers data to Client whenever new data for service X becomes available.
- Client subscribes to all warning messages of Server;
  Server automatically informs Client about all occurring warnings.

Potential response types:

- no response;
- confirmation of Request Message reception;
- notification of processing statuses by Server (command accepted, command processed, execution error/OK, etc.) - available statuses defined by application definition;
- delivery of requested information initiated by the Server as an individual Delivery Transaction; a Delivery Transaction is carried out just like a Send Transaction (in this case originating from the Server instead of the Client).

F.2 DCP message packet

A DCP message packet is a specially formed DCP Packet carrying information used within a transaction from a Client to a Server or vice versa. The parts of which such a DCP Packet is composed will be explained in the following clauses.

F.2.1 General structure

A DCP message packet has the structure shown in figure F.1.

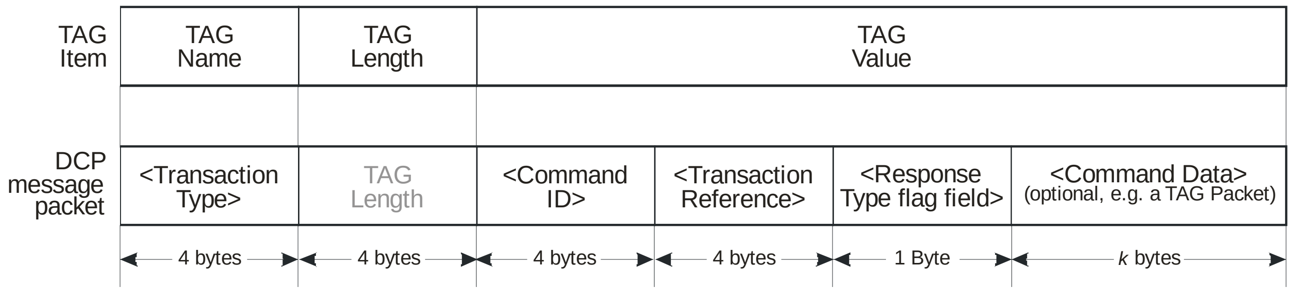


Figure F.1: Structure of a DCP message packet

Transaction Type: one of the special TAG Names (4 ASCII characters) indicating a DCP message and its type (see clause F.2.2 for details):

	Request Message Type	**Response Message Type**
Send Transaction	`*tsq`	`*tss`
Fetch Transaction	`*tfq`	`*tfs`
Subscribe Transaction	`*tuq`	`*tus`
Unsubscribe Transaction	`*tnq`	`*tns`
Delivery Transaction	`*tdq`	`*tds`

TAG Length: length of the TAG Value section in bits (as for every TAG Packet).

Command ID: indicates the individual command; may be coded as 4 "human readable" ASCII characters or as a binary number; values are assigned by the application which makes use of this DCP framework for bi-directional communication.

Transaction Reference: a reference number chosen be the transaction initiator with a new Request command (e.g. a random number) and used in any subsequent Response command to identify the individual transaction (communication thread);
no two Requests Messages sent between one Client and one Server should use the same combination of Command ID value and Transaction Reference value as far as possible;
an application should specify the minimum interval before reuse of Transaction References.

Response Type flag field: for a Request Messages:
indicates which type of responses the sender of the particular Request Message wishes to receive; one or several flags may be enabled (set to "1");
for a Response Message:
indicates which types of responses of the previously requested responses are handled within the current Response Message; each response may be sent as an individual Response Message or several responses may be contained in the same Response Message;
flags unknown to the receiver of the packet are treated as "0" (and thus be ignored);
see clause F.2.3 for details.

Command Data: optional - e.g. a data field or a full TAG Packet;
presence, layout and content depend on application specification as well as individual Command ID;
if present, may carry detailed information required e.g. for a fetch or subscribe message;
if formatted as a full TAG Packet, may always carry the pre-defined TAG Items defined in clause F.2.4.

NOTE: There is no restriction on the number of Request or Response Messages carried in one DCP Packet.

F.2.2 Transaction Types

The following Transaction Types are defined to be used within the DCP framework for bi-directional communication.

F.2.2.1 Send Transaction (request "*tsq", response "*tss")

A Client sends information to a Server (Request Message). Optionally one or more Response Messages may be requested.

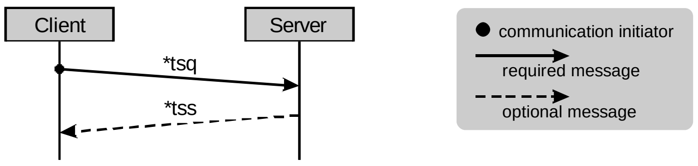


Figure F.2: Send Transaction

F.2.2.2 Fetch Transaction (request "*tfq", response "*tfs")

A Client requests information from a Server (Request Message). At least one Response Messages is sent back, containing the requested information.

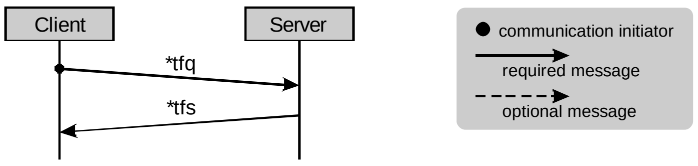


Figure F.3: Fetch Transaction

F.2.2.3 Subscribe/Unsubscribe Transaction (request "*tuq"/"*tnq", response "*tus"/"*tns"); Delivery Transaction (request "*tdq", response "*tds")

A Client subscribes to probably non-immediate information offered by a Server (Subscribe Transaction); the Server henceforth delivers information to the Client whenever (new) data is available (Delivery Transaction).
The Server stops data delivery upon reception of an Unsubscribe Transaction or for example after a missing re-subscription (timeout).

Any transaction above consists of at least a Request Message and optionally one or more Response Messages (e.g. as reception confirmation).

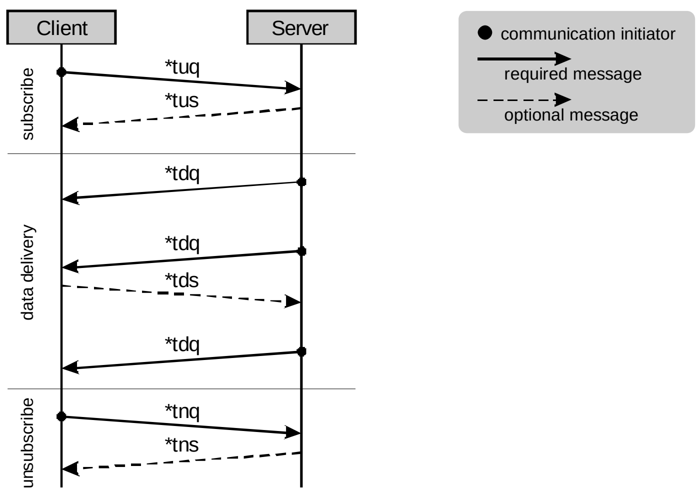


Figure F.4: Subscribe/Unsubscribe Transaction; Delivery Transaction

F.2.3 Response Type flag field

The Response Type flag field consists of 8 bits. Each bit represents a special type of requested Response Message (see below).

In a Request Message, a Response Message is requested for all types of responses indicated by the flags.

In a Response Message, one or more of the bits set in the related Request Message can be set to indicate that the flagged responses are handled by the current Response Message. The number of individual Response Messages following a particular Request Message is therefore between 0 (no flag was set) up to the number of flags set in the Request Message (each type of Response Message is sent as a separate DCP Packet). In the latter case, only one bit of the flag field is set per Response Message.

The Response Type flag field has the following structure:

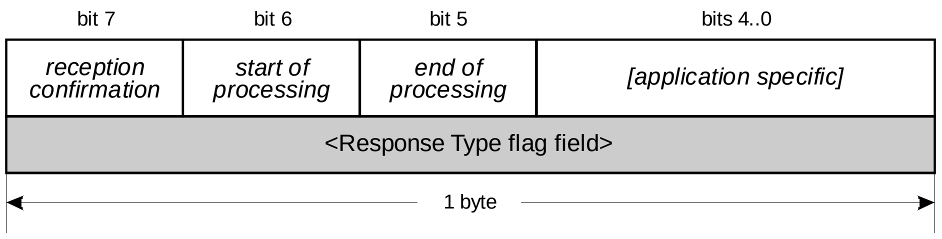


Figure F.5: Structure of the Response Type flag field

The standardized response types have the following meaning:

reception confirmation:	The Request Message was successfully received (transmission related confirmation).
start of processing:	The Command ID is known and can technically be fulfilled; optionally provided data related to the command is within a valid range (content related general confirmation).
end of processing:	The command was finally executed (content related specific confirmation). The optional status TAG Item (part of the Command Data) may provide information about success or failure.

The definition and use of all other flags is up to the application using the DCP framework for bi-directional communication.

NOTE: Any application using this DCP framework for bi-directional communication should at least support the reception confirmation. Support and exact meaning of other two standardized flags may vary from application to application, however it should be conform with the meaning assigned above.

F.2.4 Command Data

Command data is information required to process the current message. It may be a parameter value required for a particular command within a Send Request Message, the value that is returned within the Response message of a Fetch Transaction, or a data delivery a client previously subscribed to.

The optional Command Data section of the Message Packet may carry a value or a full TAG Packet in its own. The presence, layout and content of this field depends on the application and the particular Command ID (see clause F.2.4.1).

In addition, command related data may also be carried in the top hierarchy level of the message DCP Packet, for example to maintain compatibility of subscription delivery data (see clause F.2.4.2).

F.2.4.1 Command Data field carries a full TAG Packet

If the Command Data field carries a full TAG Packet, it may be used to transport parameters related to a particular Command ID or to transport any other form of information. The definition of the TAG Items contained in the TAG Packet is up to the application using the DCP framework for bi-directional communication and may also depend on the Command ID value.

However some TAG Items are predefined in this place to provide a uniform solution for general-purpose data encoding (only applicably if the Command Data field carries a full TAG Packet).

TAG Name	TAG Value	description

`*sta`	2 bytes <status indicator> +
	n bytes status description;
	<status indicator>:
	4 bits status class +
	12 bits status code
		This TAG Items carries the status e.g. as response to a command (success/failure etc.);
		status description is a plain UTF-8 formatted text;
		status class:
		0 = OK;
		1 = warning;
		2 = error;
		3 to 14: application specific;
		15: undefined/unknown;
		status code: application specific.

`*rqu`	m bytes	Format and length are application specific and depend on the Command ID.
`*rsp`	p bytes	Format and length are application specific and depend on the Command ID.

The "*rqu" and "*rsp" TAG Items may be used to transport additional information belonging to a Subscribe or Response Message respectively.

F.2.4.2 Command related data in top hierarchy level of message packet

In some cases it is sensible to transmit information in the top level hierarchy of the message DCP Packet.

An example for this is an MDI stream previously subscribed to by a Client and now delivered by the Server: To be processable by a standard MDI decoder, the actual MDI data is located in the top hierarchy level. This MDI data is just accompanied by a Delivery Message.

Another example is the downward compatible extension of protocols currently carrying commands and command-related data on top level. Any request or response (currently carried on top level) could just additionally be accompanied by a Send or Fetch Message respectively (one additional TAG Item per command). This Send or Fetch Message uses for example the same command name as defined for the command's TAG Name (or the command's ID where applicable) and adds all the features of the DCP bi-directional communication framework, without duplicating the actual useful data from the top level into the Message's Command Data section.

Annex G (informative): Revision history of changes to formulas and consequences

G.1 Clause 7.2.2, $s_{\max}$

In versions 1.1.1 and 1.2.1 of the present document, the algorithm to calculate $s_{\max}$ (for the case m>0) was defined as:

$$s_{\max} = MIN\left\{\left\lfloor \frac{c \times p}{m+1} \right\rfloor, MTU - h\right\}$$

The consequence of this formula was that even with the lowest protection strength m=1 (i.e. intended protection against a single fragment loss), more fragments than technically required were generated by the DCP encoder, and two instead of just one lost fragment(s) could be corrected by a DCP decoder. Also for every higher value of *m*, one more fragment loss than indicated by the value of *m* could be corrected by a DCP decoder at the cost of an unnecessarily high number of fragments being generated. To duplicate the exact behaviour of a DCP encoder implementation according to the old version of the algorithm, the value of *m* may be chosen one higher than actually intended. For example, to duplicate the former "protection against one single packet loss" (m=1), *m* would need to be set to 2 instead of 1 with the new version of the algorithm – however then actually protecting against two fragment losses.

In version 1.3.1 of the present document, the algorithm to calculate $s_{\max}$ (for the case m>0) was defined as:

$$s_{\max} = MIN\left\{\left\lfloor \frac{c \times p}{m} \right\rfloor, MTU - h\right\}$$

The consequence of this formula was that in some rare situations fewer than *m* lost fragments could be corrected by the decoder in cases where a specific set of fragments was lost. The new formula ensures that in any case at least *m* lost fragments can be corrected by the decoder, at the cost of slight over-protection for the cases where *m* is either 5, 7, or 9 (i.e. not a factor of $p = 48$).

An example configuration for such a special situation is l=1233, m=5 (c=6, k=206, z=3) → $s_{\max,\ old\_algorithm}$=57; $s_{\max,\ new\_algorithm}$=54. In this case the FEC correction of the second Reed-Solomon code word (calculated according to the old algorithm) fails, if at least 4 of the 5 lost fragments are from the fragment index range 11 to 21 (inclusive).

G.2 Clause 7.4.4, $Rx_{\min}$

In versions 1.1.1 and 1.2.1 of the present document, the algorithm to calculate the minimum number of required fragments to start the FEC correction process was specified as:

$$Rx_{\min} = \left\lceil \frac{c_{\max} \times k}{s} \right\rceil$$

In some cases this resulted in too few fragments, leading the DCP decoder to attempt an FEC correction too early (and thus wasting CPU power). An example configuration for this behaviour is l=379, m=5 → k=190, z=1, s=19, f=26: $Rx_{\min,new\_algorithm}$ = 21 (correct); $Rx_{\min,old\_algorithm}$ = 20 (incorrect)

Annex H (informative): Bibliography

IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".

IETF RFC 2718: "Guidelines for new URL Schemes".

History

Document history		
V1.1.1	December 2003	Publication
V1.2.1	October 2005	Publication
V1.3.1	December 2010	Publication
V1.4.1	October 2012	Publication