

lpwan Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: January 24, 2020

A. Minaburo  
Acklio  
L. Toutain  
IMT-Atlantique  
C. Gomez  
Universitat Politecnica de Catalunya  
D. Barthel  
Orange Labs  
JC. Zuniga  
SIGFOX  
July 23, 2019

**Static Context Header Compression (SCHC) and fragmentation for LPWAN,  
application to UDP/IPv6  
draft-ietf-lpwan-ipv6-static-context-hc-21**

**Abstract**

This document defines the Static Context Header Compression (SCHC) framework, which provides both header compression and fragmentation functionalities. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

SCHC compression is based on a common static context stored in both the LPWAN device and the network side. This document defines a header compression mechanism and its application to compress IPv6/UDP headers.

This document also specifies a fragmentation and reassembly mechanism that is used to support the IPv6 MTU requirement over the LPWAN technologies. Fragmentation is needed for IPv6 datagrams that, after SCHC compression or when such compression was not possible, still exceed the layer-2 maximum payload size.

The SCHC header compression and fragmentation mechanisms are independent of the specific LPWAN technology over which they are used. This document defines generic functionalities and offers flexibility with regard to parameter settings and mechanism choices. This document standardizes the exchange over the LPWAN between two SCHC entities. Settings and choices specific to a technology or a product are expected to be grouped into profiles, which are specified in other documents. Data models for the context and profiles are out of scope.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 24, 2020.

## Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction</a>	<a href="#">4</a>
<a href="#">2.</a>	<a href="#">Requirements Notation</a>	<a href="#">5</a>
<a href="#">3.</a>	<a href="#">LPWAN Architecture</a>	<a href="#">5</a>
<a href="#">4.</a>	<a href="#">Terminology</a>	<a href="#">6</a>
<a href="#">5.</a>	<a href="#">SCHC overview</a>	<a href="#">8</a>
<a href="#">5.1.</a>	<a href="#">SCHC Packet format</a>	<a href="#">10</a>
<a href="#">5.2.</a>	<a href="#">Functional mapping</a>	<a href="#">10</a>
<a href="#">6.</a>	<a href="#">Rule ID</a>	<a href="#">11</a>
<a href="#">7.</a>	<a href="#">Compression/Decompression</a>	<a href="#">12</a>
<a href="#">7.1.</a>	<a href="#">SCHC C/D Rules</a>	<a href="#">12</a>
<a href="#">7.2.</a>	<a href="#">Rule ID for SCHC C/D</a>	<a href="#">14</a>
<a href="#">7.3.</a>	<a href="#">Packet processing</a>	<a href="#">15</a>
<a href="#">7.4.</a>	<a href="#">Matching operators</a>	<a href="#">17</a>
<a href="#">7.5.</a>	<a href="#">Compression Decompression Actions (CDA)</a>	<a href="#">17</a>



7.5.1.	processing fixed-length fields . . . . .	18
7.5.2.	processing variable-length fields . . . . .	18
7.5.3.	not-sent CDA . . . . .	19
7.5.4.	value-sent CDA . . . . .	19
7.5.5.	mapping-sent CDA . . . . .	19
7.5.6.	LSB CDA . . . . .	20
7.5.7.	DevIID, AppIID CDA . . . . .	20
7.5.8.	Compute-* . . . . .	20
8.	Fragmentation/Reassembly . . . . .	21
8.1.	Overview . . . . .	21
8.2.	SCHC F/R Protocol Elements . . . . .	21
8.2.1.	Messages . . . . .	21
8.2.2.	Tiles, Windows, Bitmaps, Timers, Counters . . . . .	22
8.2.3.	Integrity Checking . . . . .	24
8.2.4.	Header Fields . . . . .	25
8.3.	SCHC F/R Message Formats . . . . .	27
8.3.1.	SCHC Fragment format . . . . .	27
8.3.2.	SCHC ACK format . . . . .	28
8.3.3.	SCHC ACK REQ format . . . . .	31
8.3.4.	SCHC Sender-Abort format . . . . .	31
8.3.5.	SCHC Receiver-Abort format . . . . .	32
8.4.	SCHC F/R modes . . . . .	33
8.4.1.	No-ACK mode . . . . .	33
8.4.2.	ACK-Always mode . . . . .	35
8.4.3.	ACK-on-Error mode . . . . .	41
9.	Padding management . . . . .	48
10.	SCHC Compression for IPv6 and UDP headers . . . . .	49
10.1.	IPv6 version field . . . . .	49
10.2.	IPv6 Traffic class field . . . . .	50
10.3.	Flow label field . . . . .	50
10.4.	Payload Length field . . . . .	50
10.5.	Next Header field . . . . .	51
10.6.	Hop Limit field . . . . .	51
10.7.	IPv6 addresses fields . . . . .	51
10.7.1.	IPv6 source and destination prefixes . . . . .	51
10.7.2.	IPv6 source and destination IID . . . . .	52
10.8.	IPv6 extensions . . . . .	52
10.9.	UDP source and destination port . . . . .	52
10.10.	UDP length field . . . . .	53
10.11.	UDP Checksum field . . . . .	53
11.	IANA Considerations . . . . .	54
12.	Security considerations . . . . .	54
12.1.	Security considerations for SCHC Compression/Decompression . . . . .	54
12.2.	Security considerations for SCHC Fragmentation/Reassembly . . . . .	55
13.	Acknowledgements . . . . .	56
14.	References . . . . .	56



<a href="#">14.1.</a>	Normative References . . . . .	<a href="#">56</a>
<a href="#">14.2.</a>	Informative References . . . . .	<a href="#">56</a>
<a href="#">Appendix A.</a>	Compression Examples . . . . .	<a href="#">57</a>
<a href="#">Appendix B.</a>	Fragmentation Examples . . . . .	<a href="#">60</a>
<a href="#">Appendix C.</a>	Fragmentation State Machines . . . . .	<a href="#">68</a>
<a href="#">Appendix D.</a>	SCHC Parameters . . . . .	<a href="#">74</a>
<a href="#">Appendix E.</a>	Supporting multiple window sizes for fragmentation .	76
<a href="#">Appendix F.</a>	Downlink SCHC Fragment transmission . . . . .	<a href="#">76</a>
	Authors' Addresses . . . . .	<a href="#">77</a>

## [1.](#) Introduction

This document defines the Static Context Header Compression (SCHC) framework, which provides both header compression and fragmentation functionalities. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

LPWAN technologies impose some strict limitations on traffic. For instance, devices sleep most of the time and may only receive data during short periods of time after transmission, in order to preserve battery. LPWAN technologies are also characterized by a greatly reduced data unit and/or payload size (see [[RFC8376](#)]).

Header compression is needed for efficient Internet connectivity to the node within an LPWAN network. The following properties of LPWAN networks can be exploited to get an efficient header compression:

- o The network topology is star-oriented, which means that all packets between the same source-destination pair follow the same path. For the needs of this document, the architecture can simply be described as Devices (Dev) exchanging information with LPWAN Application Servers (App) through a Network Gateway (NGW).
- o Because devices embed built-in applications, the traffic flows to be compressed are known in advance. Indeed, new applications are less frequently installed in an LPWAN device, than they are in a computer or smartphone.

SCHC compression uses a Context (a set of Rules) in which information about header fields is stored. This Context is static: the values of the header fields and the actions to do compression/decompression do not change over time. This avoids the need for complex resynchronization mechanisms. Indeed, a return path may be more restricted/expensive, sometimes completely unavailable [[RFC8376](#)]. A compression protocol that relies on feedback is not compatible with the characteristics of such LPWANs.



In most cases, a small Rule identifier is enough to represent the full IPv6/UDP headers. The SCHC header compression mechanism is independent of the specific LPWAN technology over which it is used.

Furthermore, some LPWAN technologies do not provide a fragmentation functionality; to support the IPv6 MTU requirement of 1280 bytes [RFC8200], they require a fragmentation protocol at the adaptation layer below IPv6. Accordingly, this document defines an optional fragmentation/reassembly mechanism for LPWAN technologies to support the IPv6 MTU requirement.

This document defines generic functionality and offers flexibility with regard to parameters settings and mechanism choices. Technology-specific settings and product-specific choices are expected to be grouped into Profiles specified in other documents.

## **2. Requirements Notation**

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#) [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## **3. LPWAN Architecture**

LPWAN technologies have similar network architectures but different terminologies. Using the terminology defined in [RFC8376], we can identify different types of entities in a typical LPWAN network, see Figure 1:

- o Devices (Dev) are the end-devices or hosts (e.g. sensors, actuators, etc.). There can be a very high density of devices per radio gateway.
- o The Radio Gateway (RGW), which is the end point of the constrained link.
- o The Network Gateway (NGW) is the interconnection node between the Radio Gateway and the Internet.
- o Application Server (App)





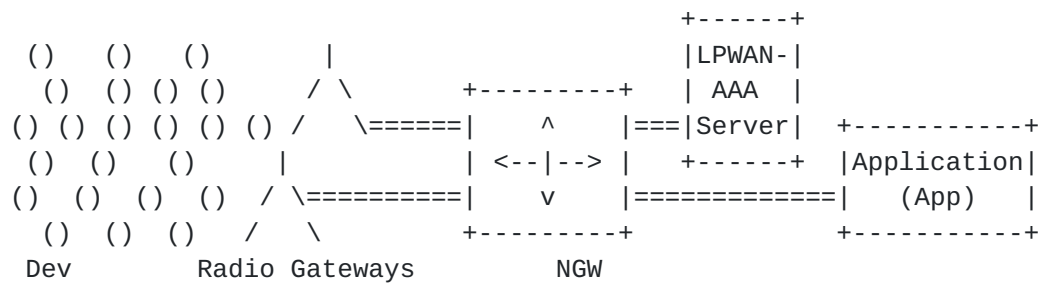


Figure 1: LPWAN Architecture as shown in [RFC8376](#)

#### 4. Terminology

This section defines the terminology and acronyms used in this document. It extends the terminology of [\[RFC8376\]](#).

The SCHC acronym is pronounced like "sheek" in English (or "chic" in French). Therefore, this document writes "a SCHC Packet" instead of "an SCHC Packet".

- o App: LPWAN Application, as defined by [\[RFC8376\]](#). An application sending/receiving packets to/from the Dev.
- o AppIID: Application Interface Identifier. The IID that identifies the application server interface.
- o Bi: Bidirectional. Characterizes a Field Descriptor that applies to headers of packets traveling in either direction (Up and Dw, see this glossary).
- o CDA: Compression/Decompression Action. Describes the pair of inverse actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field.
- o Compression Residue. The bits that remain to be sent (beyond the Rule ID itself) after applying the SCHC compression.
- o Context: A set of Rules used to compress/decompress headers.
- o Dev: Device, as defined by [\[RFC8376\]](#).
- o DevIID: Device Interface Identifier. The IID that identifies the Dev interface.



- o DI: Direction Indicator. This field tells which direction of packet travel (Up, Dw or Bi) a Field Description applies to. This allows for asymmetric processing, using the same Rule.
- o Dw: Downlink direction for compression/decompression, from SCHC C/D in the network to SCHC C/D in the Dev.
- o Field Description. A tuple containing identifier, value, matching operator and actions to be applied to a field.
- o FID: Field Identifier. This identifies the protocol and field a Field Description applies to.
- o FL: Field Length is the length of the packet header field. It is expressed in bits for header fields of fixed lengths or as a type (e.g. variable, token length, ...) for field lengths that are unknown at the time of Rule creation. The length of a header field is defined in the corresponding protocol specification (such as IPv6 or UDP).
- o FP: when a Field is expected to appear multiple times in a header, Field Position specifies the occurrence this Field Description applies to (for example, first uri-path option, second uri-path, etc. in a CoAP header).
- o IID: Interface Identifier. See the IPv6 addressing architecture [[RFC7136](#)]
- o L2: Layer two. The immediate lower layer SCHC interfaces with. It is provided by an underlying LPWAN technology. It does not necessarily correspond to the OSI model definition of Layer 2.
- o L2 Word: this is the minimum subdivision of payload data that the L2 will carry. In most L2 technologies, the L2 Word is an octet. In bit-oriented radio technologies, the L2 Word might be a single bit. The L2 Word size is assumed to be constant over time for each device.
- o MO: Matching Operator. An operator used to match a value contained in a header field with a value contained in a Rule.
- o Padding (P). Extra bits that may be appended by SCHC to a data unit that it passes to the underlying Layer 2 for transmission. SCHC itself operates on bits, not bytes, and does not have any alignment prerequisite. See [Section 9](#).
- o Profile: SCHC offers variations in the way it is operated, with a number of parameters listed in [Appendix D](#). A Profile indicates a



particular setting of all these parameters. Both ends of a SCHC communication must be provisioned with the same Profile information and with the same set of Rules before the communication starts, so that there is no ambiguity in how they expect to communicate.

- o Rule: A set of Field Descriptions.
- o Rule ID (Rule Identifier): An identifier for a Rule. SCHC C/D on both sides share the same Rule ID for a given packet. A set of Rule IDs are used to support SCHC F/R functionality.
- o SCHC C/D: SCHC Compressor/Decompressor. A mechanism used on both sides, at the Dev and at the network, to achieve Compression/Decompression of headers.
- o SCHC F/R: SCHC Fragmentation / Reassembly. A mechanism used on both sides, at the Dev and at the network, to achieve Fragmentation / Reassembly of SCHC Packets.
- o SCHC Packet: A packet (e.g. an IPv6 packet) whose header has been compressed as per the header compression mechanism defined in this document. If the header compression process is unable to actually compress the packet header, the packet with the uncompressed header is still called a SCHC Packet (in this case, a Rule ID is used to indicate that the packet header has not been compressed). See [Section 7](#) for more details.
- o TV: Target value. A value contained in a Rule that will be matched with the value of a header field.
- o Up: Uplink direction for compression/decompression, from the Dev SCHC C/D to the network SCHC C/D.

Additional terminology for the optional SCHC Fragmentation / Reassembly mechanism (SCHC F/R) is found in [Section 8.2](#).

## 5. SCHC overview

SCHC can be characterized as an adaptation layer between IPv6 and the underlying LPWAN technology. SCHC comprises two sublayers (i.e. the Compression sublayer and the Fragmentation sublayer), as shown in Figure 2.



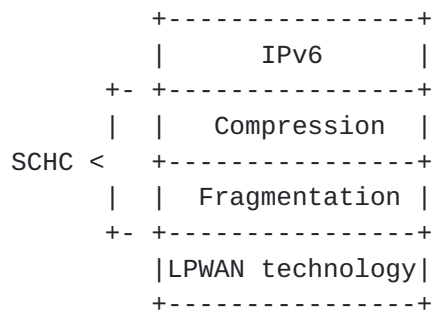
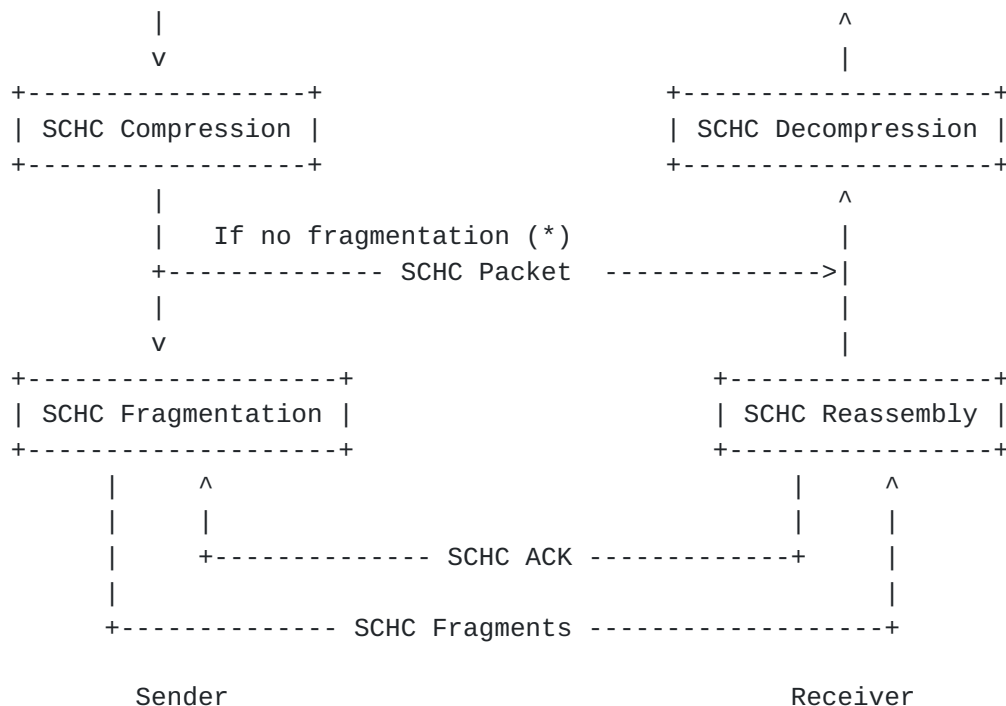


Figure 2: Protocol stack comprising IPv6, SCHC and an LPWAN technology

Before a packet (e.g. an IPv6 packet) is transmitted, header compression is first applied. The resulting packet is called a SCHC Packet, whether or not any compression is performed. If the SCHC Packet is to be fragmented, the optional SCHC Fragmentation MAY be applied to the SCHC Packet. The inverse operations take place at the receiver. This process is illustrated in Figure 3.

A packet (e.g. an IPv6 packet)



\*: the decision to use Fragmentation or not is left to each Profile.

Figure 3: SCHC operations at the SENDER and the RECEIVER





### 5.1. SCHC Packet format

The SCHC Packet is composed of the Compressed Header followed by the payload from the original packet (see Figure 4). The Compressed Header itself is composed of the Rule ID and a Compression Residue, which is the output of compressing the packet header with that Rule (see [Section 7](#)). The Compression Residue may be empty. Both the Rule ID and the Compression Residue potentially have a variable size, and are not necessarily a multiple of bytes in size.

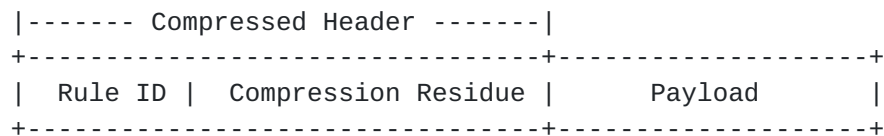


Figure 4: SCHC Packet

## 5.2. Functional mapping

Figure 5 below maps the functional elements of Figure 3 onto the LPWAN architecture elements of Figure 1.

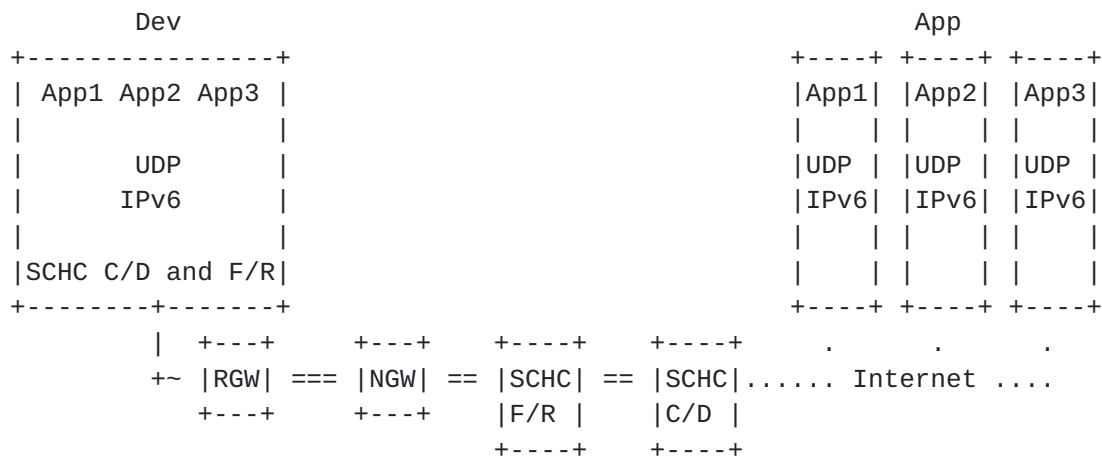


Figure 5: Architecture

SCHC C/D and SCHC F/R are located on both sides of the LPWAN transmission, i.e. on the Dev side and on the Network side.

The operation in the Uplink direction is as follows. The Device application uses IPv6 or IPv6/UDP protocols. Before sending the packets, the Dev compresses their headers using SCHC C/D and, if the SCHC Packet resulting from the compression needs to be fragmented by SCHC, SCHC F/R is performed (see [Section 8](#)). The resulting SCHC Fragments are sent to an LPWAN Radio Gateway (RGW) which forwards



them to a Network Gateway (NGW). The NGW sends the data to a SCHC F/R for re-assembly (if needed) and then to the SCHC C/D for decompression. After decompression, the packet can be sent over the Internet to one or several LPWAN Application Servers (App).

The SCHC F/R and C/D on the Network side can be located in the NGW, or somewhere else as long as a tunnel is established between them and the NGW. For some LPWAN technologies, it may be suitable to locate the SCHC F/R functionality nearer the NGW, in order to better deal with time constraints of such technologies.

The SCHC C/Ds on both sides MUST share the same set of Rules. So MUST the SCHC F/Rs on both sides.

The operation in the Downlink direction is similar to that in the Uplink direction, only reverting the order in which the architecture elements are traversed.

## **6. Rule ID**

Rule IDs identify the Rules used for Compression/Decompression or for Fragmentation/Reassembly.

The scope of a Rule ID is the link between the SCHC Compressor and the SCHC Decompressor, or between the SCHC Fragmenter and the SCHC Reassembler.

The size of the Rule IDs is not specified in this document, as it is implementation-specific and can vary according to the LPWAN technology and the number of Rules, among others. It is defined in Profiles.

The Rule IDs are used:

- o For SCHC C/D, to identify the Rule (i.e., the set of Field Descriptions) that is used to compress a packet header.
  - \* At least one Rule ID MUST be allocated to tagging packets for which SCHC compression was not possible (no matching Rule was found).
- o In SCHC F/R, to identify the specific mode and settings of F/R for one direction of traffic (Up or Dw).
  - \* When F/R is used for both communication directions, at least two Rule ID values are needed for F/R, one per direction of traffic.



## **7. Compression/Decompression**

Compression with SCHC is based on using a set of Rules, called the Context, to compress or decompress headers. SCHC avoids Context synchronization traffic, which consumes considerable bandwidth in other header compression mechanisms such as RoHC [[RFC5795](#)]. Since the content of packets is highly predictable in LPWAN networks, static Contexts may be stored beforehand. The Contexts **MUST** be stored at both ends, and they can be learned by a provisioning protocol or by out of band means, or they can be pre-provisioned. The way the Contexts are provisioned is out of the scope of this document.

### **7.1. SCHC C/D Rules**

The main idea of the SCHC compression scheme is to transmit the Rule ID to the other end instead of sending known field values. This Rule ID identifies a Rule that matches the original packet values. Hence, when a value is known by both ends, it is only necessary to send the corresponding Rule ID over the LPWAN network. The manner by which Rules are generated is out of the scope of this document. The Rules **MAY** be changed at run-time but the mechanism is out of scope of this document.

The Context is a set of Rules. See Figure 6 for a high level, abstract representation of the Context. The formal specification of the representation of the Rules is outside the scope of this document.

Each Rule itself contains a list of Field Descriptions composed of a Field Identifier (FID), a Field Length (FL), a Field Position (FP), a Direction Indicator (DI), a Target Value (TV), a Matching Operator (MO) and a Compression/Decompression Action (CDA).



```

/-----\
|                                     |
|                                     | Rule N
|                                     |
/-----\
|                                     | Rule i
/-----\
| (FID)          Rule 1              ||| | | | | | | |
|+-----+---+---+---+-----+-----+-----+-----+|||
||Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|||
|+-----+---+---+---+-----+-----+-----+-----+|||
||Field 2|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|||
|+-----+---+---+---+-----+-----+-----+-----+|||
||...   |..|..|..|   ...   | ...   | ...   |||
|+-----+---+---+---+-----+-----+-----+-----+||/
||Field N|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|||
|+-----+---+---+---+-----+-----+-----+-----+|/
|
\-----/

```

Figure 6: A Compression/Decompression Context

A Rule does not describe how the compressor parses a packet header to find and identify each field (e.g. the IPv6 Source Address, the UDP Destination Port or a CoAP URI path option). It is assumed that there is a protocol parser alongside SCHC that is able to identify all the fields encountered in the headers to be compressed, and to label them with a Field ID. Rules only describe the compression/decompression behavior for each header field, after it has been identified.

In a Rule, the Field Descriptions are listed in the order in which the fields appear in the packet header. The Field Descriptions describe the header fields with the following entries:

- o Field ID (FID) designates a protocol and field (e.g. UDP Destination Port), unambiguously among all protocols that a SCHC compressor processes. In the presence of protocol nesting, the Field ID also identifies the nesting.
- o Field Length (FL) represents the length of the field. It can be either a fixed value (in bits) if the length is known when the Rule is created or a type if the length is variable. The length of a header field is defined by its own protocol specification (e.g. IPv6 or UDP). If the length is variable, the type defines the process to compute the length and its unit (bits, bytes...).
- o Field Position (FP): most often, a field only occurs once in a packet header. However, some fields may occur multiple times. An





example is the uri-path of CoAP. FP indicates which occurrence this Field Description applies to. If FP is not specified in the Field Description, it takes the default value of 1. The value 1 designates the first occurrence. The value 0 is special. It means "don't care", see [Section 7.3](#).

- o A Direction Indicator (DI) indicates the packet direction(s) this Field Description applies to. Three values are possible:
  - \* UPLINK (Up): this Field Description is only applicable to packets sent by the Dev to the App,
  - \* DOWNLINK (Dw): this Field Description is only applicable to packets sent from the App to the Dev,
  - \* BIDIRECTIONAL (Bi): this Field Description is applicable to packets traveling both Up and Dw.
- o Target Value (TV) is the value used to match against the packet header field. The Target Value can be a scalar value of any type (integer, strings, etc.) or a more complex structure (array, list, etc.). The types and representations are out of scope for this document.
- o Matching Operator (MO) is the operator used to match the Field Value and the Target Value. The Matching Operator may require some parameters. MO is only used during the compression phase. The set of MOs defined in this document can be found in [Section 7.4](#).
- o Compression Decompression Action (CDA) describes the compression and decompression processes to be performed after the MO is applied. Some CDAs might use parameter values for their operation. CDAs are used in both the compression and the decompression functions. The set of CDAs defined in this document can be found in [Section 7.5](#).

## **[7.2](#). Rule ID for SCHC C/D**

Rule IDs are sent by the compression function in one side and are received for the decompression function in the other side. In SCHC C/D, the Rule IDs are specific to the Context related to one Dev. Hence, multiple Dev instances, which refer to different header compression Contexts, MAY reuse the same Rule ID for different Rules. On the network side, in order to identify the correct Rule to be applied, the SCHC Decompressor needs to associate the Rule ID with the Dev identifier. Similarly, the SCHC Compressor on the network side first identifies the destination Dev before looking for the



appropriate compression Rule (and associated Rule ID) in the Context of that Dev.

### **7.3. Packet processing**

The compression/decompression process follows several steps:

- o Compression Rule selection: the set of Rules is browsed to identify which Rule will be used to compress the packet header. The Rule is selected by matching the Fields Descriptions to the packet header. The detailed steps are the following:
  - \* The first step is to check the Field Identifiers (FID). If any header field of the packet being examined cannot be matched with a Field Description with the correct FID, the Rule MUST be disregarded. If any Field Description in the Rule has a FID that cannot be matched to one of the header fields of the packet being examined, the Rule MUST be disregarded.
  - \* The next step is to match the Field Descriptions by their direction, using the Direction Indicator (DI). If any field of the packet header cannot be matched with a Field Description with the correct FID and DI, the Rule MUST be disregarded.
  - \* Then the Field Descriptions are further selected according to Field Position (FP). If any field of the packet header cannot be matched with a Field Description with the correct FID, DI and FP, the Rule MUST be disregarded.

The value 0 for FP means "don't care", i.e. the comparison of this Field Description's FP with the position of the field of the packet header being compressed returns True, whatever that position. FP=0 can be useful to build compression Rules for protocols headers in which some fields order is irrelevant. An example could be uri-queries in CoAP. Care needs to be exercised when writing Rules containing FP=0 values. Indeed, it may result in decompressed packets having fields ordered differently compared to the original packet.

- \* Once each header field has been associated with a Field Description with matching FID, DI and FP, each packet field's value is then compared to the corresponding Target Value (TV) stored in the Rule for that specific field, using the matching operator (MO). If every field in the packet header satisfies the corresponding matching operators (MO) of a Rule (i.e. all MO results are True), that Rule is used for compressing the header. Otherwise, the Rule MUST be disregarded.



- \* If no eligible compression Rule is found, then the header **MUST** be sent in its entirety using the Rule ID of the "default" Rule dedicated to this purpose. Sending an uncompressed header may require SCHC F/R.
- o Compression: each field of the header is compressed according to the Compression/Decompression Actions (CDAs). The fields are compressed in the order that the Field Descriptions appear in the Rule. The compression of each field results in a residue, which may be empty. The Compression Residue for the packet header is the concatenation of the non-empty residues for each field of the header, in the order the Field Descriptions appear in the Rule.

```

|----- Compression Residue -----|
+-----+-----+-----+-----+
| field 1 residue | field 2 residue | ... | field N residue |
+-----+-----+-----+-----+

```

Figure 7: Compression Residue structure

- o Sending: The Rule ID is sent to the other end followed by the Compression Residue (which could be empty) or the uncompressed header, and directly followed by the payload (see Figure 4). The way the Rule ID is sent will be specified in the Profile and is out of the scope of the present document. For example, it could be included in an L2 header or sent as part of the L2 payload.
- o Decompression: when decompressing, on the network side the SCHC C/D needs to find the correct Rule based on the L2 address; in this way, it can use the DevIID and the Rule ID. On the Dev side, only the Rule ID is needed to identify the correct Rule since the Dev typically only holds Rules that apply to itself.

The receiver identifies the sender through its device-id or source identifier (e.g. MAC address, if it exists) and selects the Rule using the Rule ID. This Rule describes the compressed header format and associates the received residues to each of the header fields. For each field in the header, the receiver applies the CDA action associated to that field in order to reconstruct the original header field value. The CDA application order can be different from the order in which the fields are listed in the Rule. In particular, Compute-\* **MUST** be applied after the application of the CDAs of all the fields it computes on.



#### 7.4. Matching operators

Matching Operators (MOs) are functions used by both SCHC C/D endpoints. They are not typed and can be applied to integer, string or any other data type. The result of the operation can either be True or False. MOs are defined as follows:

- o equal: The match result is True if the field value in the packet matches the TV.
- o ignore: No matching is attempted between the field value in the packet and the TV in the Rule. The result is always true.
- o MSB(x): A match is obtained if the most significant x bits of the packet header field value are equal to the TV in the Rule. The x parameter of the MSB MO indicates how many bits are involved in the comparison. If the FL is described as variable, the length must be a multiple of the unit. For example, x must be multiple of 8 if the unit of the variable length is in bytes.
- o match-mapping: With match-mapping, the Target Value is a list of values. Each value of the list is identified by an index. Compression is achieved by sending the index instead of the original header field value. This operator matches if the header field value is equal to one of the values in the target list.

#### 7.5. Compression Decompression Actions (CDA)

The Compression Decompression Action (CDA) describes the actions taken during the compression of header fields and the inverse action taken by the decompressor to restore the original value.

Action	Compression	Decompression
not-sent	elided	use TV stored in Rule
value-sent	send	use received value
mapping-sent	send index	retrieve value from TV list
LSB	send LSB	concat. TV and received value
compute-*	elided	recompute at decompressor
DevIID	elided	build IID from L2 Dev addr
AppIID	elided	build IID from L2 App addr

Table 1: Compression and Decompression Actions





Table 1 summarizes the basic actions that can be used to compress and decompress a field. The first column shows the action's name. The second and third columns show the compression and decompression behaviors for each action.

#### **7.5.1. processing fixed-length fields**

If the field is identified in the Field Description as being of fixed length, then applying the CDA to compress this field results in a fixed amount of bits. The residue for that field is simply the bits resulting from applying the CDA to the field. This value may be empty (e.g. not-sent CDA), in which case the field residue is absent from the Compression Residue.

```
| - field residue - |
+-----+
|   value   |
+-----+
```

Figure 8: fixed sized field residue structure

#### **7.5.2. processing variable-length fields**

If the field is identified in the Field Description as being of variable length, then applying the CDA to compress this field may result in a value of fixed size (e.g. not-sent or mapping-sent) or of variable size (e.g. value-sent or LSB). In the latter case, the residue for that field is the bits that result from applying the CDA to the field, preceded with the size of the value. The most significant bit of the size is stored first (left of the residue bit field).

```
| --- field residue --- |
+-----+-----+
| size | value |
+-----+-----+
```

Figure 9: variable sized field residue structure

The size (using the unit defined in the FL) is encoded on 4, 12 or 28 bits as follows:

- o If the size is between 0 and 14, it is encoded as a 4 bits unsigned integer.



- o Sizes between 15 and 254 are encoded as 0b1111 followed by the 8 bits unsigned integer.
- o Larger sizes are encoded as 0xffff followed by the 16 bits unsigned integer.

If the field is identified in the Field Description as being of variable length and this field is not present in the packet header being compressed, size 0 MUST be sent to denote its absence.

#### **7.5.3. not-sent CDA**

The not-sent action can be used when the field value is specified in a Rule and therefore known by both the Compressor and the Decompressor. This action SHOULD be used with the "equal" MO. If MO is "ignore", there is a risk to have a decompressed field value different from the original field that was compressed.

The compressor does not send any residue for a field on which not-sent compression is applied.

The decompressor restores the field value with the Target Value stored in the matched Rule identified by the received Rule ID.

#### **7.5.4. value-sent CDA**

The value-sent action can be used when the field value is not known by both the Compressor and the Decompressor. The value is sent in its entirety.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in [Section 7.5.2](#).

This action is generally used with the "ignore" MO.

#### **7.5.5. mapping-sent CDA**

The mapping-sent action is used to send an index (the index into the Target Value list of values) instead of the original value. This action is used together with the "match-mapping" MO.

On the compressor side, the match-mapping Matching Operator searches the TV for a match with the header field value. The mapping-sent CDA then sends the corresponding index as the field residue. The most significant bit of the index is stored first (left of the residue bit field).



On the decompressor side, the CDA uses the received index to restore the field value by looking up the list in the TV.

The number of bits sent is the minimal size for coding all the possible indices.

#### **7.5.6. LSB CDA**

The LSB action is used together with the "MSB(x)" MO to avoid sending the most significant part of the packet field if that part is already known by the receiving end.

The compressor sends the Least Significant Bits as the field residue value. The number of bits sent is the original header field length minus the length specified in the MSB(x) MO.

The decompressor concatenates the x most significant bits of Target Value and the received residue value.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in [Section 7.5.2](#).

#### **7.5.7. DevIID, AppIID CDA**

These actions are used to process respectively the Dev and the App Interface Identifiers (DevIID and AppIID) of the IPv6 addresses. AppIID CDA is less common since most current LPWAN technologies frames contain a single L2 address, which is the Dev's address.

The IID value MAY be computed from the Device ID present in the L2 header, or from some other stable identifier. The computation is specific to each Profile and MAY depend on the Device ID size.

In the downlink direction (Dw), at the compressor, the DevIID CDA may be used to generate the L2 addresses on the LPWAN, based on the packet's Destination Address.

#### **7.5.8. Compute-\***

Some fields can be elided at the compressor and recomputed locally at the decompressor.

Because the field is uniquely identified by its Field ID (e.g. UDP length), the relevant protocol specification unambiguously defines the algorithm for such computation.



Examples of fields that know how to recompute themselves are UDP length, IPv6 length and UDP checksum.

## **8. Fragmentation/Reassembly**

### **8.1. Overview**

In LPWAN technologies, the L2 MTU typically ranges from tens to hundreds of bytes. Some of these technologies do not have an internal fragmentation/reassembly mechanism.

The optional SCHC Fragmentation/Reassembly (SCHC F/R) functionality enables such LPWAN technologies to comply with the IPv6 MTU requirement of 1280 bytes [[RFC8200](#)]. It is optional to implement. If it is not needed, its description can be safely ignored.

This specification includes several SCHC F/R modes, which allow for a range of reliability options such as optional SCHC Fragment retransmission. More modes may be defined in the future.

The same SCHC F/R mode MUST be used for all SCHC Fragments of a SCHC Packet. This document does not specify which mode(s) are to be used over a specific LPWAN technology. That information will be given in Profiles.

The L2 Word size (see [Section 4](#)) determines the encoding of some messages. SCHC F/R usually generates SCHC Fragments and SCHC ACKs that are multiples of L2 Words.

### **8.2. SCHC F/R Protocol Elements**

This subsection describes the different elements that are used to enable the SCHC F/R functionality defined in this document. These elements include the SCHC F/R messages, tiles, windows, bitmaps, counters, timers and header fields.

The elements are described here in a generic manner. Their application to each SCHC F/R mode is found in [Section 8.4](#).

#### **8.2.1. Messages**

SCHC F/R defines the following messages:

- o SCHC Fragment: A message that carries part of a SCHC Packet from the sender to the receiver.
- o SCHC ACK: An acknowledgement for fragmentation, by the receiver to the sender. This message is used to indicate whether or not the





reception of pieces of, or the whole of the fragmented SCHC Packet, was successful.

- o SCHC ACK REQ: A request by the sender for a SCHC ACK from the receiver.
- o SCHC Sender-Abort: A message by the sender telling the receiver that it has aborted the transmission of a fragmented SCHC Packet.
- o SCHC Receiver-Abort: A message by the receiver to tell the sender to abort the transmission of a fragmented SCHC Packet.

### 8.2.2. Tiles, Windows, Bitmaps, Timers, Counters

### 8.2.2.1. Tiles

The SCHC Packet is fragmented into pieces, hereafter called tiles. The tiles MUST be non-empty and pairwise disjoint. Their union MUST be equal to the SCHC Packet.

See Figure 10 for an example.

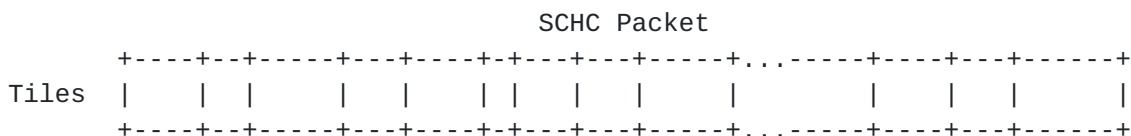


Figure 10: a SCHC Packet fragmented in tiles

Each SCHC Fragment message carries at least one tile in its Payload, if the Payload field is present.

#### 8.2.2.2. Windows

Some SCHC F/R modes may handle successive tiles in groups, called windows.

If windows are used

- o all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW\_SIZE.
- o WINDOW\_SIZE MUST be specified in a Profile.
- o the windows are numbered.



- o their numbers MUST increase from 0 upward, from the start of the SCHC Packet to its end.
- o the last window MUST contain WINDOW\_SIZE tiles or less.
- o tiles are numbered within each window.
- o the tile indices MUST decrement from WINDOW\_SIZE - 1 downward, looking from the start of the SCHC Packet toward its end.
- o each tile of a SCHC Packet is therefore uniquely identified by a window number and a tile index within this window.

See Figure 11 for an example.

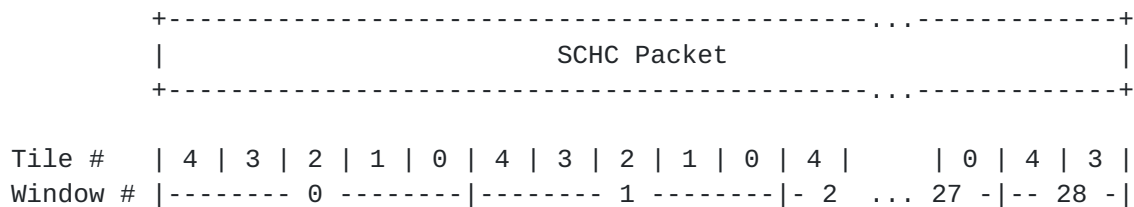


Figure 11: a SCHC Packet fragmented in tiles grouped in 28 windows, with WINDOW\_SIZE = 5

When windows are used

- o Bitmaps (see [Section 8.2.2.3](#)) MAY be sent back by the receiver to the sender in a SCHC ACK message.
- o A Bitmap corresponds to exactly one Window.

#### **[8.2.2.3. Bitmaps](#)**

Each bit in the Bitmap for a window corresponds to a tile in the window. Each Bitmap has therefore WINDOW\_SIZE bits. The bit at the left-most position corresponds to the tile numbered WINDOW\_SIZE - 1. Consecutive bits, going right, correspond to sequentially decreasing tile indices. In Bitmaps for windows that are not the last one of a SCHC Packet, the bit at the right-most position corresponds to the tile numbered 0. In the Bitmap for the last window, the bit at the right-most position corresponds either to the tile numbered 0 or to a tile that is sent/received as "the last one of the SCHC Packet" without explicitly stating its number (see [Section 8.3.1.2](#)).

At the receiver



- o a bit set to 1 in the Bitmap indicates that a tile associated with that bit position has been correctly received for that window.
- o a bit set to 0 in the Bitmap indicates that no tile associated with that bit position has been correctly received for that window.

#### **8.2.2.4. Timers and counters**

Some SCHC F/R modes can use the following timers and counters

- o Inactivity Timer: a SCHC Fragment receiver uses this timer to abort waiting for a SCHC F/R message.
- o Retransmission Timer: a SCHC Fragment sender uses this timer to abort waiting for an expected SCHC ACK.
- o Attempts: this counter counts the requests for SCHC ACKs, up to MAX\_ACK\_REQUESTS.

#### **8.2.3. Integrity Checking**

The integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receive end. By default, integrity checking is performed by computing a Reassembly Check Sequence (RCS) of the SCHC Packet at the sender side before fragmentation and transmitting it to the receiver for comparison with the RCS locally computed after reassembly.

The RCS supports UDP checksum elision by SCHC C/D (see [Section 10.11](#)).

The CRC32 polynomial 0xEDB88320 (i.e. the reverse representation of the polynomial used e.g. in the Ethernet standard [[RFC3385](#)]) is RECOMMENDED as the default algorithm for computing the RCS. Nevertheless, other RCS lengths or other algorithms MAY be required by the Profile.

The RCS MUST be computed on the full SCHC Packet concatenated with the padding bits, if any, of the SCHC Fragment carrying the last tile. The rationale is that the SCHC reassembler has no way of knowing the boundary between the last tile and the padding bits. Indeed, this requires decompressing the SCHC Packet, which is out of the scope of the SCHC reassembler.

Note that the concatenation of the complete SCHC Packet and the potential padding bits of the last SCHC Fragment does not generally constitute an integer number of bytes. For implementers to be able



to use byte-oriented CRC libraries, it is RECOMMENDED that the concatenation of the complete SCHC Packet and the last fragment potential padding bits be zero-extended to the next byte boundary and that the RCS be computed on that byte array. A Profile MAY specify another behavior.

#### **8.2.4. Header Fields**

The SCHC F/R messages contain the following fields (see the formats in [Section 8.3](#)):

- o Rule ID: this field is present in all the SCHC F/R messages. It is used to identify
  - \* that a SCHC F/R message is being carried, as opposed to an unfragmented SCHC Packet,
  - \* which SCHC F/R mode is used
  - \* and for this mode
    - + if windows are used and what the value of WINDOW\_SIZE is,
    - + what other optional fields are present and what the field sizes are.

The Rule ID allows SCHC F/R interleaving non-fragmented SCHC Packets and SCHC Fragments that carry other SCHC Packets, or interleaving SCHC Fragments that use different SCHC F/R modes or different parameters.

- o Datagram Tag (DTag). Its size (called T, in bits) is defined by each Profile for each Rule ID. When T is 0, the DTag field does not appear in the SCHC F/R messages and the DTag value is defined as 0.

When T is 0, there can be only one fragmented SCHC Packet in transit for a given Rule ID.

If T is not 0, DTag

- \* MUST be set to the same value for all the SCHC F/R messages related to the same fragmented SCHC Packet,
- \* MUST be set to different values for SCHC F/R messages related to different SCHC Packets that are being fragmented under the same Rule ID and the transmission of which may overlap.





A sequence counter that is incremented for each new fragmented SCHC Packet, counting from 0 to up to  $(2^T)-1$  and wrapping back to 0 is RECOMMENDED for maximum traceability and avoidance of ambiguity.

A flow of SCHC F/R messages with a given Rule ID and DTag value pair MUST NOT interfere with the operation of a SCHC F/R instance that uses another Rule ID and DTag value pair.

- o W: The W field is optional. It is only present if windows are used. Its presence and size (called M, in bits) is defined by each SCHC F/R mode and each Profile for each Rule ID.

This field carries information pertaining to the window a SCHC F/R message relates to. If present, W MUST carry the same value for all the SCHC F/R messages related to the same window. Depending on the mode and Profile, W may carry the full window number, or just the least significant bit or any other partial representation of the window number.

- o Fragment Compressed Number (FCN). The FCN field is present in the SCHC Fragment Header. Its size (called N, in bits) is defined by each Profile for each Rule ID.

This field conveys information about the progress in the sequence of tiles being transmitted by SCHC Fragment messages. For example, it can contain a partial, efficient representation of a larger-sized tile index. The description of the exact use of the FCN field is left to each SCHC F/R mode. However, two values are reserved for special purposes. They help control the SCHC F/R process:

- \* The FCN value with all the bits equal to 1 (called All-1) signals the very last tile of a SCHC Packet. By extension, if windows are used, the last window of a packet is called the All-1 window.
- \* If windows are used, the FCN value with all the bits equal to 0 (called All-0) signals the last tile of a window that is not the last one of the SCHC packet. By extension, such a window is called an All-0 window.
- o Reassembly Check Sequence (RCS). This field only appears in the All-1 SCHC Fragments. Its size (called U, in bits) is defined by each Profile for each Rule ID.

See [Section 8.2.3](#) for the RCS default size, default polynomial and details on RCS computation.



- o C (integrity Check): C is a 1-bit field. This field is used in the SCHC ACK message to report on the reassembled SCHC Packet integrity check (see [Section 8.2.3](#)).

A value of 1 tells that the integrity check was performed and is successful. A value of 0 tells that the integrity check was not performed, or that it was a failure.

- o Compressed Bitmap. The Compressed Bitmap is used together with windows and Bitmaps (see [Section 8.2.2.3](#)). Its presence and size is defined for each F/R mode for each Rule ID.

This field appears in the SCHC ACK message to report on the receiver Bitmap (see [Section 8.3.2.1](#)).

### 8.3. SCHC F/R Message Formats

This section defines the SCHC Fragment formats, the SCHC ACK format, the SCHC ACK REQ format and the SCHC Abort formats.

#### 8.3.1. SCHC Fragment format

A SCHC Fragment conforms to the general format shown in Figure 12. It comprises a SCHC Fragment Header and a SCHC Fragment Payload. The SCHC Fragment Payload carries one or several tile(s).

```
+-----+-----+-----+
| Fragment Header | Fragment Payload | padding (as needed) |
+-----+-----+-----+
```

Figure 12: SCHC Fragment general format

##### 8.3.1.1. Regular SCHC Fragment

The Regular SCHC Fragment format is shown in Figure 13. Regular SCHC Fragments are generally used to carry tiles that are not the last one of a SCHC Packet. The DTag field and the W field are optional.

```
|--- SCHC Fragment Header ---|
      |-- T --|-M-|-- N --|
+-- ... --+ ... -+---+ ... -+-----+-----+
| Rule ID | DTag | W | FCN | Fragment Payload | padding (as needed) |
+-- ... --+ ... -+---+ ... -+-----+-----+
```

Figure 13: Detailed Header Format for Regular SCHC Fragments

The FCN field MUST NOT contain all bits set to 1.



The Fragment Payload of a SCHC Fragment with FCN equal to 0 (called an All-0 SCHC Fragment) MUST be distinguishable by size from a SCHC ACK REQ message (see [Section 8.3.3](#)) that has the same T, M and N values, even in the presence of padding. This condition is met if the Payload is at least the size of an L2 Word. This condition is also met if the SCHC Fragment Header is a multiple of L2 Words.

#### [8.3.1.2](#). All-1 SCHC Fragment

The All-1 SCHC Fragment format is shown in Figure 14. The sender generally uses the All-1 SCHC Fragment format for the message that completes the emission of a fragmented SCHC Packet. The DTag field, the W field, the RCS field and the Payload are optional. At least one of RCS field or Payload MUST be present. The FCN field is all ones.

```
|----- SCHC Fragment Header -----|
|-- T --|-M-|- N --|- U --|
+-- ... --+ ... -+---+ ... -+ ... -+-----+-----+-----+
| Rule ID | DTag | W | 11..1 | RCS | Frag Payload | pad. (as needed)
+-- ... --+ ... -+---+ ... -+ ... -+-----+-----+-----+
                                   (FCN)
```

Figure 14: Detailed Header Format for the All-1 SCHC Fragment

The All-1 SCHC Fragment message MUST be distinguishable by size from a SCHC Sender-Abort message (see [Section 8.3.4](#)) that has the same T, M and N values, even in the presence of padding. This condition is met if the RCS is present and is at least the size of an L2 Word, or if the Payload is present and at least the size an L2 Word. This condition is also met if the SCHC Sender-Abort Header is a multiple of L2 Words.

#### [8.3.2](#). SCHC ACK format

The SCHC ACK message is shown in Figure 15. The DTag field, the W field and the Compressed Bitmap field are optional. The Compressed Bitmap field can only be present in SCHC F/R modes that use windows.



```

|---- SCHC ACK Header ----|
      |-- T --|-M-| 1 |
+--- ... +- ... -+---+---+~~~~~
| Rule ID | DTag | W |C=1| padding as needed          (success)
+--- ... +- ... -+---+---+~~~~~

+--- ... +- ... -+---+---+----- ... -----+~~~~~
| Rule ID | DTag | W |C=0|Compressed Bitmap| pad. as needed (failure)
+--- ... +- ... -+---+---+----- ... -----+~~~~~

```

Figure 15: Format of the SCHC ACK message

The SCHC ACK Header contains a C bit (see [Section 8.2.4](#)).

If the C bit is set to 1 (integrity check successful), no Bitmap is carried.

If the C bit is set to 0 (integrity check not performed or failed) and if windows are used, a Compressed Bitmap for the window referred to by the W field is transmitted as specified in [Section 8.3.2.1](#).

#### **8.3.2.1. Bitmap Compression**

For transmission, the Compressed Bitmap in the SCHC ACK message is defined by the following algorithm (see Figure 16 for a follow-along example):

- o Build a temporary SCHC ACK message that contains the Header followed by the original Bitmap (see [Section 8.2.2.3](#) for a description of Bitmaps).
- o Position scissors at the end of the Bitmap, after its last bit.
- o While the bit on the left of the scissors is 1 and belongs to the Bitmap, keep moving left, then stop. When this is done,
- o While the scissors are not on an L2 Word boundary of the SCHC ACK message and there is a Bitmap bit on the right of the scissors, keep moving right, then stop.
- o At this point, cut and drop off any bits to the right of the scissors

When one or more bits have effectively been dropped off as a result of the above algorithm, the SCHC ACK message is a multiple of L2 Words, no padding bits will be appended.





Because the SCHC Fragment sender knows the size of the original Bitmap, it can reconstruct the original Bitmap from the Compressed Bitmap received in the SCH ACK message.

Figure 16 shows an example where L2 Words are actually bytes and where the original Bitmap contains 17 bits, the last 15 of which are all set to 1.

```
|---- SCHC ACK Header ----|----- Bitmap -----|
      |-- T --|-M-| 1 |
+--- ... -+- ... -+---+---+-----+
| Rule ID | DTag | W |C=0|1 0 1 1 1 1 1 1 1 1 1 1 1 1 1|
+--- ... -+- ... -+---+---+-----+
      next L2 Word boundary ->|
```

Figure 16: SCHC ACK Header plus uncompressed Bitmap

Figure 17 shows that the last 14 bits are not sent.

```
|---- SCHC ACK Header ----|CpBmp|
      |-- T --|-M-| 1 |
+--- ... -+- ... -+---+---+-----+
| Rule ID | DTag | W |C=0|1 0 1|
+--- ... -+- ... -+---+---+-----+
      next L2 Word boundary ->|
```

Figure 17: Resulting SCHC ACK message with Compressed Bitmap

Figure 18 shows an example of a SCHC ACK with tile indices ranging from 6 down to 0, where the Bitmap indicates that the second and the fourth tile of the window have not been correctly received.

```
|---- SCHC ACK Header ----|--- Bitmap --|
      |-- T --|-M-| 1 |6 5 4 3 2 1 0| (tile #)
+-----+-----+---+---+-----+
| Rule ID | DTag | W |C=0|1 0 1 0 1 1 1|      uncompressed Bitmap
+-----+-----+---+---+-----+
      next L2 Word boundary ->|<-- L2 Word -->|

+-----+-----+---+---+-----+~~~~+
| Rule ID | DTag | W |C=0|1 0 1 0 1 1 1|Pad| transmitted SCHC ACK
+-----+-----+---+---+-----+~~~~+
      next L2 Word boundary ->|<-- L2 Word -->|
```

Figure 18: Example of a SCHC ACK message, missing tiles



Figure 19 shows an example of a SCHC ACK with FCN ranging from 6 down to 0, where integrity check has not been performed or has failed and the Bitmap indicates that there is no missing tile in that window.

```
|---- SCHC ACK Header ----|--- Bitmap --|
      |-- T --|-M-| 1 |6 5 4 3 2 1 0| (tile #)
+-----+-----+-----+-----+
| Rule ID | DTag | W |C=0|1 1 1 1 1 1| with uncompressed Bitmap
+-----+-----+-----+-----+
      next L2 Word boundary ->|

+--- ... -+- ... -+---+---+
| Rule ID | DTag | W |C=0|1| transmitted SCHC ACK
+--- ... -+- ... -+---+---+
      next L2 Word boundary ->|
```

Figure 19: Example of a SCHC ACK message, no missing tile

### 8.3.3. SCHC ACK REQ format

The SCHC ACK REQ is used by a sender to request a SCHC ACK from the receiver. Its format is shown in Figure 20. The DTag field and the W field are optional. The FCN field is all zero.

```
|---- SCHC ACK REQ Header ----|
      |-- T --|-M-|-- N --|
+-- ... -+- ... -+---+---+ ... -+-----+
| Rule ID | DTag | W | 0..0 | padding (as needed) (no payload)
+-- ... -+- ... -+---+---+ ... -+-----+
```

Figure 20: SCHC ACK REQ format

### 8.3.4. SCHC Sender-Abort format

When a SCHC Fragment sender needs to abort an on-going fragmented SCHC Packet transmission, it sends a SCHC Sender-Abort message to the SCHC Fragment receiver.

The SCHC Sender-Abort format is shown in Figure 21. The DTag field and the W field are optional. The FCN field is all ones.

```
|---- Sender-Abort Header ----|
      |-- T --|-M-|-- N --|
+-- ... -+- ... -+---+---+ ... -+-----+
| Rule ID | DTag | W | 11..1 | padding (as needed)
+-- ... -+- ... -+---+---+ ... -+-----+
```

Figure 21: SCHC Sender-Abort format



If the W field is present,

- o the fragment sender MUST set it to all ones. Other values are RESERVED.
- o the fragment receiver MUST check its value. If the value is different from all ones, the message MUST be ignored.

The SCHC Sender-Abort MUST NOT be acknowledged.

#### **8.3.5. SCHC Receiver-Abort format**

When a SCHC Fragment receiver needs to abort an on-going fragmented SCHC Packet transmission, it transmits a SCHC Receiver-Abort message to the SCHC Fragment sender.

The SCHC Receiver-Abort format is shown in Figure 22. The DTag field and the W field are optional.

```
|--- Receiver-Abort Header ---|
|--- T ---|-M-| 1 |
+--- ... ---+--- ... ---+---+---+---+---+---+---+---+---+---+
| Rule ID | DTag | W |C=1| 1..1|      1..1      |
+--- ... ---+--- ... ---+---+---+---+---+---+---+---+---+---+
                        next L2 Word boundary ->|<-- L2 Word -->|
```

Figure 22: SCHC Receiver-Abort format

If the W field is present,

- o the fragment receiver MUST set it to all ones. Other values are RESERVED.
- o if the value is different from all ones, the fragment sender MUST ignore the message.

The SCHC Receiver-Abort has the same header as a SCHC ACK message. The bits that follow the SCHC Receiver-Abort Header MUST be as follows

- o if the Header does not end at an L2 Word boundary, append bits set to 1 as needed to reach the next L2 Word boundary
- o append exactly one more L2 Word with bits all set to ones

Such a bit pattern never occurs in a regular SCHC ACK. This is how the fragment sender recognizes a SCHC Receiver-Abort.



The SCHC Receiver-Abort MUST NOT be acknowledged.

#### **8.4. SCHC F/R modes**

This specification includes several SCHC F/R modes, which

- o allow for a range of reliability options, such as optional SCHC Fragment retransmission
- o support various LPWAN characteristics, including variable MTU.

More modes may be defined in the future.

##### **8.4.1. No-ACK mode**

The No-ACK mode has been designed under the assumption that data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly. This mode supports LPWAN technologies that have a variable MTU.

In No-ACK mode, there is no communication from the fragment receiver to the fragment sender. The sender transmits all the SCHC Fragments without expecting acknowledgement.

In No-ACK mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

The tile sizes are not required to be uniform. Windows are not used. The Retransmission Timer is not used. The Attempts counter is not used.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST NOT be present in the SCHC F/R messages. SCHC ACK MUST NOT be sent. SCHC ACK REQ MUST NOT be sent. SCHC Sender-Abort MAY be sent. SCHC Receiver-Abort MUST NOT be sent.

The value of N (size of the FCN field) is RECOMMENDED to be 1.

Each Profile, for each Rule ID value, MUST define

- o the size of the DTag field,
- o the size and algorithm for the RCS field,
- o the expiration time of the Inactivity Timer





Each Profile, for each Rule ID value, MAY define

- o a value of N different from the recommended one,
- o the meaning of values sent in the FCN field, for values different from the All-1 value.

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer.

#### **8.4.1.1. Sender behavior**

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. The tile MUST be at least the size of an L2 Word. The sender MUST transmit the SCHC Fragments messages in the order that the tiles appear in the SCHC Packet. Except for the last tile of a SCHC Packet, each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits. Except for the last one, the SCHC Fragments MUST use the Regular SCHC Fragment format specified in [Section 8.3.1.1](#). The last SCHC Fragment MUST use the All-1 format specified in [Section 8.3.1.2](#).

The sender MAY transmit a SCHC Sender-Abort.

Figure 37 shows an example of a corresponding state machine.

#### **8.4.1.2. Receiver behavior**

Upon receiving each Regular SCHC Fragment,

- o the receiver MUST reset the Inactivity Timer,
- o the receiver assembles the payloads of the SCHC Fragments

On receiving an All-1 SCHC Fragment,

- o the receiver MUST append the All-1 SCHC Fragment Payload and the padding bits to the previously received SCHC Fragment Payloads for this SCHC Packet
- o the receiver MUST perform the integrity check



- o if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet
- o the reassembly operation concludes.

On expiration of the Inactivity Timer, the receiver MUST drop the SCHC Packet being reassembled.

On receiving a SCHC Sender-Abort, the receiver MAY drop the SCHC Packet being reassembled.

Figure 38 shows an example of a corresponding state machine.

#### **8.4.2. ACK-Always mode**

The ACK-Always mode has been designed under the following assumptions

- o Data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly
- o The L2 MTU value does not change while the fragments of a SCHC Packet are being transmitted.

In ACK-Always mode, windows are used. An acknowledgement, positive or negative, is transmitted by the fragment receiver to the fragment sender at the end of the transmission of each window of SCHC Fragments.

The tiles are not required to be of uniform size. In ACK-Always mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

Briefly, the algorithm is as follows: after a first blind transmission of all the tiles of a window, the fragment sender iterates retransmitting the tiles that are reported missing until the fragment receiver reports that all the tiles belonging to the window have been correctly received, or until too many attempts were made. The fragment sender only advances to the next window of tiles when it has ascertained that all the tiles belonging to the current window have been fully and correctly received. This results in a per-window lock-step behavior between the sender and the receiver.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present and its size M MUST be 1 bit.



Each Profile, for each Rule ID value, MUST define

- o the value of N (size of the FCN field),
- o the value of WINDOW\_SIZE, which MUST be strictly less than  $2^N$ ,
- o the size and algorithm for the RCS field,
- o the size of the DTag field,
- o the value of MAX\_ACK\_REQUESTS,
- o the expiration time of the Retransmission Timer
- o the expiration time of the Inactivity Timer

For each active pair of Rule ID and DTag values, the sender MUST maintain

- o one Attempts counter
- o one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer.

#### **8.4.2.1. Sender behavior**

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. All tiles with the index 0, as well as the last tile, MUST be at least the size of an L2 Word.

In all SCHC Fragment messages, the W field MUST be filled with the least significant bit of the window number that the sender is currently processing.

For a SCHC Fragment that carries a tile other than the last one of the SCHC Packet,

- o the Fragment MUST be of the Regular type specified in [Section 8.3.1.1](#)
- o the FCN field MUST contain the tile index



- o each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits.

The SCHC Fragment that carries the last tile MUST be an All-1 SCHC Fragment, described in [Section 8.3.1.2](#).

The fragment sender MUST start by transmitting the window numbered 0.

The sender starts by a "blind transmission" phase, in which it MUST transmit all the tiles composing the window, in decreasing tile index order.

Then, it enters a "retransmission phase" in which it MUST initialize an Attempts counter to 0, it MUST start a Retransmission Timer and it MUST await a SCHC ACK. Then,

- o upon receiving a SCHC ACK,
  - \* if the SCHC ACK indicates that some tiles are missing at the receiver, then the sender MUST transmit all the tiles that have been reported missing, it MUST increment Attempts, it MUST reset the Retransmission Timer and MUST await the next SCHC ACK.
  - \* if the current window is not the last one and the SCHC ACK indicates that all tiles were correctly received, the sender MUST stop the Retransmission Timer, it MUST advance to the next fragmentation window and it MUST start a blind transmission phase as described above.
  - \* if the current window is the last one and the SCHC ACK indicates that more tiles were received than the sender sent, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - \* if the current window is the last one and the SCHC ACK indicates that all tiles were correctly received yet integrity check was a failure, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - \* if the current window is the last one and the SCHC ACK indicates that integrity checking was successful, the sender exits successfully.
- o on Retransmission Timer expiration,





- \* if Attempts is strictly less than MAX\_ACK\_REQUESTS, the fragment sender MUST send a SCHC ACK REQ and MUST increment the Attempts counter.
- \* otherwise the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

At any time,

- o on receiving a SCHC Receiver-Abort, the fragment sender MAY exit with an error condition.
- o on receiving a SCHC ACK that bears a W value different from the W value that it currently uses, the fragment sender MUST silently discard and ignore that SCHC ACK.

Figure 39 shows an example of a corresponding state machine.

#### **8.4.2.2. Receiver behavior**

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

- o the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.
- o the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair.
- o the receiver MUST start an Inactivity Timer. It MUST initialize an Attempts counter to 0. It MUST initialize a window counter to 0.

In the rest of this section, "local W bit" means the least significant bit of the window counter of the receiver.

On reception of any SCHC F/R message, the receiver MUST reset the Inactivity Timer.

Entering an "acceptance phase", the receiver MUST first initialize an empty Bitmap for this window, then

- o on receiving a SCHC Fragment or SCHC ACK REQ with the W bit different from the local W bit, the receiver MUST silently ignore and discard that message.



- o on receiving a SCHC Fragment with the W bit equal to the local W bit, the receiver MUST assemble the received tile based on the window counter and on the FCN field in the SCHC Fragment and it MUST update the Bitmap.
- \* if the SCHC Fragment received is an All-0 SCHC Fragment, the current window is determined to be a not-last window, and the receiver MUST send a SCHC ACK for this window. Then,
  - + If the Bitmap indicates that all the tiles of the current window have been correctly received, the receiver MUST increment its window counter and it enters the "acceptance phase" for that new window.
  - + If the Bitmap indicates that at least one tile is missing in the current window, the receiver enters the "retransmission phase" for this window.
- \* if the SCHC Fragment received is an All-1 SCHC Fragment, the padding bits of the All-1 SCHC Fragment MUST be assembled after the received tile, the current window is determined to be the last window, the receiver MUST perform the integrity check and it MUST send a SCHC ACK for this window. Then,
  - + If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase".
  - + If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for this window.
- o on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver has not yet determined if the current window is a not-last one or the last one, the receiver MUST send a SCHC ACK for this window, and it keeps accepting incoming messages.

In the "retransmission phase":

- o if the window is a not-last window
  - \* on receiving a SCHC Fragment or SCHC ACK REQ with a W bit different from the local W bit the receiver MUST silently ignore and discard that message.
  - \* on receiving a SCHC ACK REQ with a W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.



- \* on receiving a SCHC Fragment with a W bit equal to the local W bit,
  - + if the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST silently ignore it and discard it.
  - + otherwise, the receiver MUST update the Bitmap and it MUST assemble the tile received.
- \* on the Bitmap becoming fully populated with 1's, the receiver MUST send a SCHC ACK for this window, it MUST increment its window counter and it enters the "acceptance phase" for the new window.
- o if the window is the last window
  - \* on receiving a SCHC Fragment or SCHC ACK REQ with a W bit different from the local W bit the receiver MUST silently ignore and discard that message.
  - \* on receiving a SCHC ACK REQ with a W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.
  - \* on receiving a SCHC Fragment with a W bit equal to the local W bit,
    - + if the SCHC Fragment received is an All-0 SCHC Fragment, the receiver MUST silently ignore it and discard it.
    - + otherwise, the receiver MUST update the Bitmap and it MUST assemble the tile received. If the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST assemble the padding bits of the All-1 SCHC Fragment after the received tile. It MUST perform the integrity check. Then
      - if the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST send a SCHC ACK and it enters the "clean-up phase".
      - if the integrity check indicates that the full SCHC Packet has not been correctly reassembled,
        - o if the SCHC Fragment received was an All-1 SCHC Fragment, the receiver MUST send a SCHC ACK for this window
        - o it keeps accepting incoming messages.



In the "clean-up phase":

- o Any received SCHC F/R message with a W bit different from the local W bit MUST be silently ignored and discarded.
- o Any received SCHC F/R message different from an All-1 SCHC Fragment or a SCHC ACK REQ MUST be silently ignored and discarded.
- o On receiving an All-1 SCHC Fragment or a SCHC ACK REQ, the receiver MUST send a SCHC ACK.

At any time, on expiration of the Inactivity Timer, on receiving a SCHC Sender-Abort or when Attempts reaches MAX\_ACK\_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit the receive process for that SCHC Packet.

Figure 40 shows an example of a corresponding state machine.

#### **8.4.3. ACK-on-Error mode**

The ACK-on-Error mode supports LPWAN technologies that have variable MTU and out-of-order delivery.

In ACK-on-Error mode, windows are used. All tiles MUST be of equal size, except for the last one, which MUST be of the same size or smaller than the regular ones. If allowed in a Profile, the penultimate tile MAY be exactly one L2 Word smaller than the regular tile size.

A SCHC Fragment message carries one or more tiles, which may span multiple windows. A SCHC ACK reports on the reception of exactly one window of tiles.

See Figure 23 for an example.

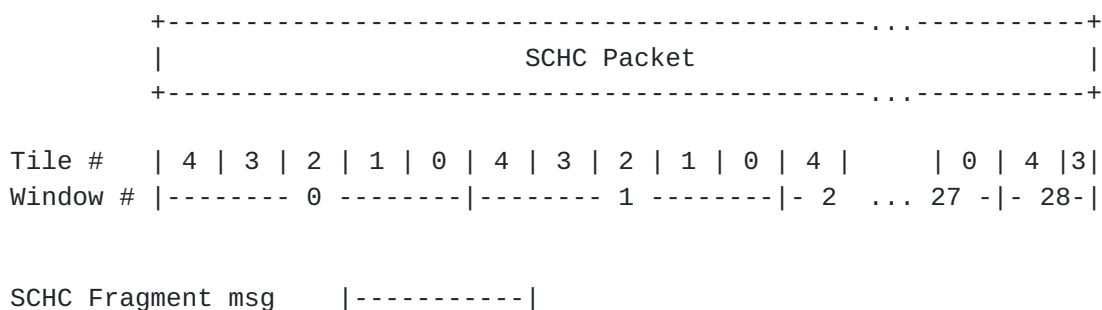


Figure 23: a SCHC Packet fragmented in tiles, Ack-on-Error mode





The W field is wide enough that it unambiguously represents an absolute window number. The fragment receiver sends SCHC ACKs to the fragment sender about windows for which tiles are missing. No SCHC ACK is sent by the fragment receiver for windows that it knows have been fully received.

The fragment sender retransmits SCHC Fragments for tiles that are reported missing. It can advance to next windows even before it has ascertained that all tiles belonging to previous windows have been correctly received, and can still later retransmit SCHC Fragments with tiles belonging to previous windows. Therefore, the sender and the receiver may operate in a decoupled fashion. The fragmented SCHC Packet transmission concludes when

- o integrity checking shows that the fragmented SCHC Packet has been correctly reassembled at the receive end, and this information has been conveyed back to the sender,
- o or too many retransmission attempts were made,
- o or the receiver determines that the transmission of this fragmented SCHC Packet has been inactive for too long.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present in the SCHC F/R messages.

Each Profile, for each Rule ID value, MUST define

- o the tile size (a tile does not need to be multiple of an L2 Word, but it MUST be at least the size of an L2 Word)
- o the value of M (size of the W field),
- o the value of N (size of the FCN field),
- o the value of WINDOW\_SIZE, which MUST be strictly less than  $2^N$ ,
- o the size and algorithm for the RCS field,
- o the size of the DTag field,
- o the value of MAX\_ACK\_REQUESTS,
- o the expiration time of the Retransmission Timer
- o the expiration time of the Inactivity Timer



- o if the last tile is carried in a Regular SCHC Fragment or an All-1 SCHC Fragment (see [Section 8.4.3.1](#))
- o if the penultimate tile MAY be one L2 Word smaller than the regular tile size. In this case, the regular tile size MUST be at least twice the L2 Word size.

For each active pair of Rule ID and DTag values, the sender MUST maintain

- o one Attempts counter
- o one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer.

#### **8.4.3.1. Sender behavior**

At the beginning of the fragmentation of a new SCHC Packet,

- o the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet. A Rule MUST NOT be selected if the values of M and WINDOW\_SIZE for that Rule are such that the SCHC Packet cannot be fragmented in  $(2^M) * \text{WINDOW\_SIZE}$  tiles or less.
- o the fragment sender MUST initialize the Attempts counter to 0 for that Rule ID and DTag value pair.

A Regular SCHC Fragment message carries in its payload one or more tiles. If more than one tile is carried in one Regular SCHC Fragment

- o the selected tiles MUST be consecutive in the original SCHC Packet
- o they MUST be placed in the SCHC Fragment Payload adjacent to one another, in the order they appear in the SCHC Packet, from the start of the SCHC Packet toward its end.

Tiles that are not the last one MUST be sent in Regular SCHC Fragments specified in [Section 8.3.1.1](#). The FCN field MUST contain the tile index of the first tile sent in that SCHC Fragment.

In a Regular SCHC Fragment message, the sender MUST fill the W field with the window number of the first tile sent in that SCHC Fragment.

Depending on the Profile, the last tile of a SCHC Packet MUST be sent either



- o in a Regular SCHC Fragment, alone or as part of a multi-tiles Payload
- o alone in an All-1 SCHC Fragment

In an All-1 SCHC Fragment message, the sender MUST fill the W field with the window number of the last tile of the SCHC Packet.

The fragment sender MUST send SCHC Fragments such that, all together, they contain all the tiles of the fragmented SCHC Packet.

The fragment sender MUST send at least one All-1 SCHC Fragment.

The fragment sender MUST listen for SCHC ACK messages after having sent

- o an All-1 SCHC Fragment
- o or a SCHC ACK REQ.

A Profile MAY specify other times at which the fragment sender MUST listen for SCHC ACK messages. For example, this could be after sending a complete window of tiles.

Each time a fragment sender sends an All-1 SCHC Fragment or a SCHC ACK REQ,

- o it MUST increment the Attempts counter
- o it MUST reset the Retransmission Timer

On Retransmission Timer expiration

- o if Attempts is strictly less than MAX\_ACK\_REQUESTS, the fragment sender MUST send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window,
- o otherwise the fragment sender MUST send a SCHC Sender-Abort and it MAY exit with an error condition.

On receiving a SCHC ACK,

- o if the W field in the SCHC ACK corresponds to the last window of the SCHC Packet,
  - \* if the C bit is set, the sender MAY exit successfully
  - \* otherwise,



- + if the Profile mandates that the last tile be sent in an All-1 SCHC Fragment,
  - if the SCHC ACK shows no missing tile at the receiver, the sender
    - o MUST send a SCHC Sender-Abort
    - o MAY exit with an error condition
  - otherwise
    - o the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.
    - o if the last message in this sequence of SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MUST in addition send a SCHC ACK REQ with the W field corresponding to the last window, after the sequence.
- + otherwise,
  - if the SCHC ACK shows no missing tile at the receiver, the sender MUST send the All-1 SCHC Fragment
  - otherwise
    - o the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.
    - o the fragment sender MUST then send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window.
- o otherwise, the fragment sender
  - \* MUST send SCHC Fragment messages containing the tiles that are reported missing in the SCHC ACK
  - \* then it MAY send a SCHC ACK REQ with the W field corresponding to the last window

See Figure 41 for one among several possible examples of a Finite State Machine implementing a sender behavior obeying this specification.





#### **8.4.3.2. Receiver behavior**

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

- o the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.
- o the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair.
- o the receiver MUST start an Inactivity Timer. It MUST initialize an Attempts counter to 0.

On receiving any SCHC F/R message, the receiver MUST reset the Inactivity Timer.

On receiving a SCHC Fragment message, the receiver determines what tiles were received, based on the payload length and on the W and FCN fields of the SCHC Fragment.

- o if the FCN is All-1, if a Payload is present, the full SCHC Fragment Payload MUST be assembled including the padding bits. This is because the size of the last tile is not known by the receiver, therefore padding bits are indistinguishable from the tile data bits, at this stage. They will be removed by the SCHC C/D sublayer. If the size of the SCHC Fragment Payload exceeds or equals the size of one regular tile plus the size of an L2 Word, this SHOULD raise an error flag.
- o otherwise, tiles MUST be assembled based on the a priori known tile size.
  - \* If allowed by the Profile, the end of the payload MAY contain the last tile, which may be shorter. Padding bits are indistinguishable from the tile data bits, at this stage.
  - \* the payload may contain the penultimate tile that, if allowed by the Profile, MAY be exactly one L2 Word shorter than the regular tile size.
  - \* Otherwise, padding bits MUST be discarded. The latter is possible because
    - + the size of the tiles is known a priori,



- + tiles are larger than an L2 Word
- + padding bits are always strictly less than an L2 Word

On receiving a SCHC ACK REQ or an All-1 SCHC Fragment,

- o if the receiver has at least one window that it knows has tiles missing, it MUST return a SCHC ACK for the lowest-numbered such window,
- o otherwise,
  - \* if it has received at least one tile, it MUST return a SCHC ACK for the highest-numbered window it currently has tiles for
  - \* otherwise it MUST return a SCHC ACK for window numbered 0

A Profile MAY specify other times and circumstances at which a receiver sends a SCHC ACK, and which window the SCHC ACK reports about in these circumstances.

Upon sending a SCHC ACK, the receiver MUST increase the Attempts counter.

After receiving an All-1 SCHC Fragment, a receiver MUST check the integrity of the reassembled SCHC Packet at least every time it prepares for sending a SCHC ACK for the last window.

Upon receiving a SCHC Sender-Abort, the receiver MAY exit with an error condition.

Upon expiration of the Inactivity Timer, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

On the Attempts counter exceeding MAX\_ACK\_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

Reassembly of the SCHC Packet concludes when

- o a Sender-Abort has been received
- o or the Inactivity Timer has expired
- o or the Attempts counter has exceeded MAX\_ACK\_REQUESTS
- o or when at least an All-1 SCHC Fragment has been received and integrity checking of the reassembled SCHC Packet is successful.



See Figure 42 for one among several possible examples of a Finite State Machine implementing a receiver behavior obeying this specification, and that is meant to match the sender Finite State Machine of Figure 41.

## **9. Padding management**

SCHC C/D and SCHC F/R operate on bits, not bytes. SCHC itself does not have any alignment prerequisite. The size of SCHC Packets can be any number of bits.

If the layer below SCHC constrains the payload to align to some boundary, called L2 Words (for example, bytes), the SCHC messages **MUST** be padded. When padding occurs, the number of appended bits **MUST** be strictly less than the L2 Word size.

If a SCHC Packet is sent unfragmented (see Figure 24), it is padded as needed for transmission.

If a SCHC Packet needs to be fragmented for transmission, it is not padded in itself. Only the SCHC F/R messages are padded as needed for transmission. Some SCHC F/R messages are intrinsically aligned to L2 Words.



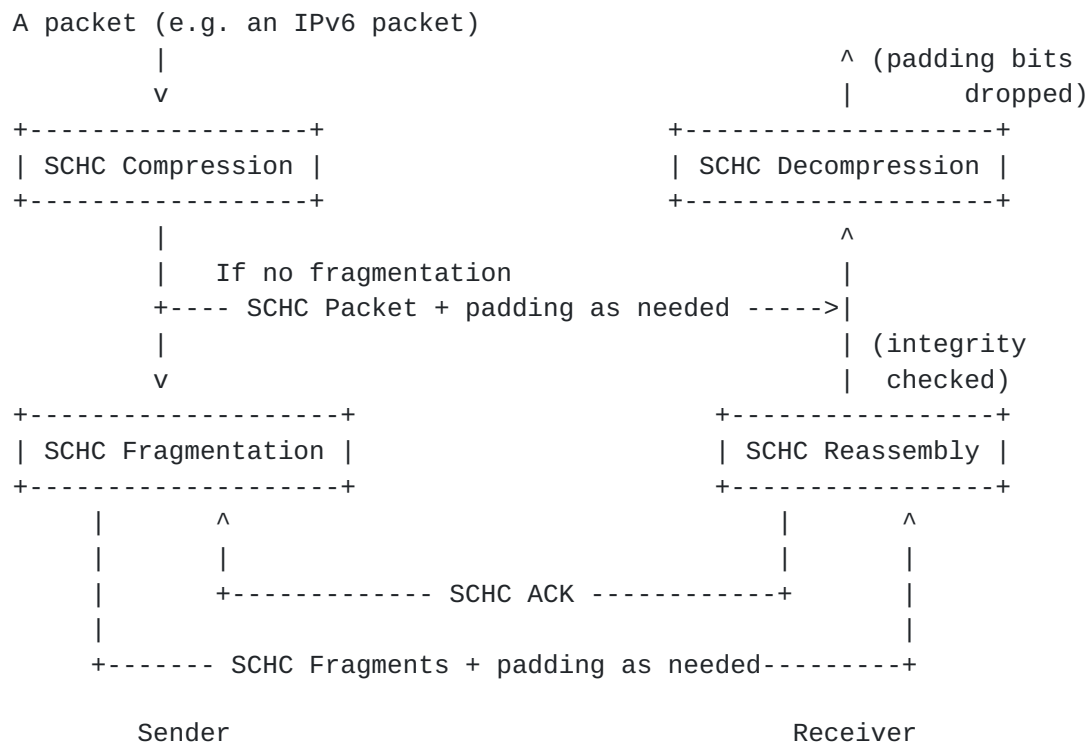


Figure 24: SCHC operations, including padding as needed

Each Profile MUST specify the size of the L2 Word. The L2 Word might actually be a single bit, in which case no padding will take place at all.

A Profile MAY define the value of the padding bits. The RECOMMENDED value is 0.

## 10. SCHC Compression for IPv6 and UDP headers

This section lists the IPv6 and UDP header fields and describes how they can be compressed.

### 10.1. IPv6 version field

The IPv6 version field is labeled by the protocol parser as being the "version" field of the IPv6 protocol. Therefore, it only exists for IPv6 packets. In the Rule, TV is set to 6, MO to "ignore" and CDA to "not-sent".





### **10.2. IPv6 Traffic class field**

If the DiffServ field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this well-known value, an "equal" MO and a "not-sent" CDA.

Otherwise (e.g. ECN bits are to be transmitted), two possibilities can be considered depending on the variability of the value:

- o One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".
- o If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

### **10.3. Flow label field**

If the Flow Label field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this well-known value, an "equal" MO and a "not-sent" CDA.

Otherwise, two possibilities can be considered:

- o One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".
- o If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

ECN functionality depends on both bits of the ECN field, which are the 2 LSBs of this field, hence sending only a single LSB of this field is NOT RECOMMENDED.

### **10.4. Payload Length field**

This field can be elided for the transmission on the LPWAN network. The SCHC C/D recomputes the original payload length value. In the Field Descriptor, TV is not set, MO is set to "ignore" and CDA is "compute-\*".



### **10.5. Next Header field**

If the Next Header field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this Next Header value, the MO SHOULD be "equal" and the CDA SHOULD be "not-sent".

Otherwise, TV is not set in the Field Descriptor, MO is set to "ignore" and CDA is set to "value-sent". Alternatively, a matching-list MAY also be used.

### **10.6. Hop Limit field**

The field behavior for this field is different for uplink (Up) and downlink (Dw). In Up, since there is no IP forwarding between the Dev and the SCHC C/D, the value is relatively constant. On the other hand, the Dw value depends on Internet routing and can change more frequently. The Direction Indicator (DI) can be used to distinguish both directions:

- o in the Up, elide the field: the TV in the Field Descriptor is set to the known constant value, the MO is set to "equal" and the CDA is set to "not-sent".
- o in the Dw, the Hop Limit is elided for transmission and forced to 1 at the receiver, by setting TV to 1, MO to "ignore" and CDA to "not-sent". This prevents any further forwarding.

### **10.7. IPv6 addresses fields**

As in 6LoWPAN [[RFC4944](#)], IPv6 addresses are split into two 64-bit long fields; one for the prefix and one for the Interface Identifier (IID). These fields SHOULD be compressed. To allow for a single Rule being used for both directions, these values are identified by their role (Dev or App) and not by their position in the header (source or destination).

#### **10.7.1. IPv6 source and destination prefixes**

Both ends MUST be configured with the appropriate prefixes. For a specific flow, the source and destination prefixes can be unique and stored in the Context. In that case, the TV for the source and destination prefixes contain the values, the MO is set to "equal" and the CDA is set to "not-sent".

If the Rule is intended to compress packets with different prefix values, match-mapping SHOULD be used. The different prefixes are



listed in the TV, the MO is set to "match-mapping" and the CDA is set to "mapping-sent". See Figure 26.

Otherwise, the TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

#### **10.7.2. IPv6 source and destination IID**

If the Dev or App IID are based on an LPWAN address, then the IID can be reconstructed with information coming from the LPWAN header. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "DevIID" or "AppIID". On LPWAN technologies where the frames carry a single identifier (corresponding to the Dev.), AppIID cannot be used.

As described in [[RFC8065](#)], it may be undesirable to build the Dev IPv6 IID out of the Dev address. Another static value is used instead. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDA is set to "not-sent". [[RFC7217](#)] provides some methods to derive this static identifier.

If several IIDs are possible, then the TV contains the list of possible IIDs, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

It may also happen that the IID variability only expresses itself on a few bytes. In that case, the TV is set to the stable part of the IID, the MO is set to "MSB" and the CDA is set to "LSB".

Finally, the IID can be sent in its entirety on the LPWAN. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

#### **10.8. IPv6 extensions**

This document does not provide recommendations on how to compress IPv6 extensions.

#### **10.9. UDP source and destination port**

To allow for a single Rule being used for both directions, the UDP port values are identified by their role (Dev or App) and not by their position in the header (source or destination). The SCHC C/D MUST be aware of the traffic direction (Uplink, Downlink) to select the appropriate field. The following Rules apply for Dev and App port numbers.



If both ends know the port number, it can be elided. The TV contains the port number, the MO is set to "equal" and the CDA is set to "not-sent".

If the port variation is on few bits, the TV contains the stable part of the port number, the MO is set to "MSB" and the CDA is set to "LSB".

If some well-known values are used, the TV can contain the list of these values, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

Otherwise the port numbers are sent over the LPWAN. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

#### **10.10. UDP length field**

The UDP length can be computed from the received data. The TV is not set, the MO is set to "ignore" and the CDA is set to "compute-\*".

#### **10.11. UDP Checksum field**

The UDP checksum operation is mandatory with IPv6 for most packets but there are exceptions [[RFC8200](#)].

For instance, protocols that use UDP as a tunnel encapsulation may enable zero-checksum mode for a specific port (or set of ports) for sending and/or receiving. [[RFC8200](#)] requires any node implementing zero-checksum mode to follow the requirements specified in "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums" [[RFC6936](#)].

6LoWPAN Header Compression [[RFC6282](#)] also specifies that a UDP checksum can be elided by the compressor and re-computed by the decompressor when an upper layer guarantees the integrity of the UDP payload and pseudo-header. A specific example of this is when a Message Integrity Check protects the compressed message between the compressor that elides the UDP checksum and the decompressor that computes it, with a strength that is identical or better to the UDP checksum.

Similarly, a SCHC compressor MAY elide the UDP checksum when another layer guarantees at least equal integrity protection for the UDP payload and the pseudo-header. In this case, the TV is not set, the MO is set to "ignore" and the CDA is set to "compute-\*".

In particular, when SCHC fragmentation is used, a fragmentation RCS of 2 bytes or more provides equal or better protection than the UDP





checksum; in that case, if the compressor is collocated with the fragmentation point and the decompressor is collocated with the packet reassembly point, and if the SCHC Packet is fragmented even when it would fit unfragmented in the L2 MTU, then the compressor MAY verify and then elide the UDP checksum. Whether and when the UDP Checksum is elided is to be specified in the Profile.

Since the compression happens before the fragmentation, implementors should understand the risks when dealing with unprotected data below the transport layer and take special care when manipulating that data.

In other cases, the checksum SHOULD be explicitly sent. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

## **11. IANA Considerations**

This document has no request to IANA.

## **12. Security considerations**

Wireless networks are subjects to various sorts of attacks, which are not specific to SCHC. In this section, we'll assume that an attacker was able to break into the network despite the latter's security measures and that it can now send packets to a target node. What is specific to SCHC is the amplification of the effects that this break-in could allow. Our analysis equally applies to legitimate nodes "going crazy".

### **12.1. Security considerations for SCHC Compression/Decompression**

Let's assume that an attacker is able to send a forged SCHC Packet to a SCHC Decompressor.

Let's first consider the case where the Rule ID contained in that forged SCHC Packet does not correspond to a Rule allocated in the Rule table. An implementation should detect that the Rule ID is invalid and should silently drop the offending SCHC Packet.

Let's now consider that the Rule ID corresponds to a Rule in the table. With the CDAs defined in this document, the reconstructed packet is at most a constant number of bits bigger than the SCHC Packet that was received. This assumes that the compute-\* decompression actions produce a bounded number of bits, irrespective of the incoming SCHC Packet. This property is true for IPv6 Length, UDP Length and UDP Checksum, for which the compute-\* CDA is recommended by this document.



As a consequence, SCHC Decompression does not amplify attacks, beyond adding a bounded number of bits to the SCHC Packet received. This bound is determined by the Rule stored in the receiving device.

As a general safety measure, a SCHC Decompressor should never re-construct a packet larger than MAX\_PACKET\_SIZE (defined in a Profile, with 1500 bytes as generic default).

## **12.2. Security considerations for SCHC Fragmentation/Reassembly**

Let's assume that an attacker is able to send to a forged SCHC Fragment to a SCHC Reassembler.

A node can perform a buffer reservation attack: the receiver will reserve buffer space for the SCHC Packet. If the implementation has only one buffer, other incoming fragmented SCHC Packets will be dropped while the reassembly buffer is occupied during the reassembly timeout. Once that timeout expires, the attacker can repeat the same procedure, and iterate, thus creating a denial of service attack. An implementation may have multiple reassembly buffers. The cost to mount this attack is linear with the number of buffers at the target node. Better, the cost for an attacker can be increased if individual fragments of multiple SCHC Packets can be stored in the reassembly buffer. The finer grained the reassembly buffer (down to the smallest tile size), the higher the cost of the attack. If buffer overload does occur, a smart receiver could selectively discard SCHC Packets being reassembled based on the sender behavior, which may help identify which SCHC Fragments have been sent by the attacker. Another mild counter-measure is for the target to abort the fragmentation/reassembly session as early as it detects a non-identical SCHC Fragment duplicate, anticipating for an eventual corrupt SCHC Packet, so as to save the sender the hassle of sending the rest of the fragments for this SCHC Packet.

In another type of attack, the malicious node is additionally assumed to be able to hear an incoming communication destined to the target node. It can then send a forged SCHC Fragment that looks like it belongs to a SCHC Packet already being reassembled at the target node. This can cause the SCHC Packet to be considered corrupt and be dropped by the receiver. The amplification happens here by a single spoofed SCHC Fragment rendering a full sequence of legit SCHC Fragments useless. If the target uses ACK-Always or ACK-on-Error mode, such a malicious node can also interfere with the acknowledgement and repetition algorithm of SCHC F/R. A single spoofed ACK, with all bitmap bits set to 0, will trigger the repetition of WINDOW\_SIZE tiles. This protocol loop amplification depletes the energy source of the target node and consumes the channel bandwidth. Similarly, a spoofed ACK REQ will trigger the



sending of a SCHC ACK, which may be much larger than the ACK REQ if WINDOW\_SIZE is large. These consequences should be borne in mind when defining profiles for SCHC over specific LPWAN technologies.

### **13. Acknowledgements**

Thanks to Sergio Aguilar Romero, Carsten Bormann, David Black, Philippe Clavier, Daniel Ducuara Beltran Diego Dujovne, Eduardo Ingles Sanchez, Arunprabhu Kandasamy, Suresh Krishnan, Rahul Jadhav, Sergio Lopez Bernal, Antony Markovski, Alexander Pelov, Charles Perkins, Edgar Ramos, Shoichi Sakane, and Pascal Thubert for useful design consideration and comments.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336, and by the ERDF and the Spanish Government through project TEC2016-79988-P. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

### **14. References**

#### **14.1. Normative References**

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", [RFC 6936](#), DOI 10.17487/RFC6936, April 2013, <<https://www.rfc-editor.org/info/rfc6936>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, [RFC 8200](#), DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.

#### **14.2. Informative References**



- [RFC3385] Sheinwald, D., Satran, J., Thaler, P., and V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations", [RFC 3385](#), DOI 10.17487/RFC3385, September 2002, <<https://www.rfc-editor.org/info/rfc3385>>.
- [RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", [RFC 4944](#), DOI 10.17487/RFC4944, September 2007, <<https://www.rfc-editor.org/info/rfc4944>>.
- [RFC5795] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObusT Header Compression (ROHC) Framework", [RFC 5795](#), DOI 10.17487/RFC5795, March 2010, <<https://www.rfc-editor.org/info/rfc5795>>.
- [RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", [RFC 6282](#), DOI 10.17487/RFC6282, September 2011, <<https://www.rfc-editor.org/info/rfc6282>>.
- [RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6 Interface Identifiers", [RFC 7136](#), DOI 10.17487/RFC7136, February 2014, <<https://www.rfc-editor.org/info/rfc7136>>.
- [RFC7217] Gont, F., "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", [RFC 7217](#), DOI 10.17487/RFC7217, April 2014, <<https://www.rfc-editor.org/info/rfc7217>>.
- [RFC8065] Thaler, D., "Privacy Considerations for IPv6 Adaptation-Layer Mechanisms", [RFC 8065](#), DOI 10.17487/RFC8065, February 2017, <<https://www.rfc-editor.org/info/rfc8065>>.
- [RFC8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", [RFC 8376](#), DOI 10.17487/RFC8376, May 2018, <<https://www.rfc-editor.org/info/rfc8376>>.

## **[Appendix A](#). Compression Examples**

This section gives some scenarios of the compression mechanism for IPv6/UDP. The goal is to illustrate the behavior of SCHC.

The mechanisms defined in this document can be applied to a Dev that embeds some applications running over CoAP. In this example, three flows are considered. The first flow is for the device management based on CoAP using Link Local IPv6 addresses and UDP ports 123 and





124 for Dev and App, respectively. The second flow will be a CoAP server for measurements done by the Dev (using ports 5683) and Global IPv6 Address prefixes alpha::IID/64 to beta::1/64. The last flow is for legacy applications using different ports numbers, the destination IPv6 address prefix is gamma::1/64.

Figure 25 presents the protocol stack. IPv6 and UDP are represented with dotted lines since these protocols are compressed on the radio link.

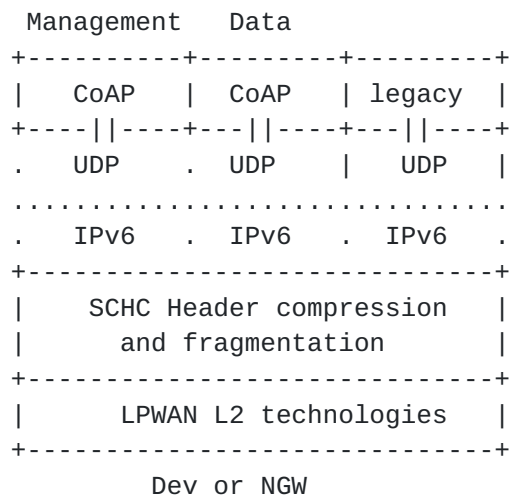


Figure 25: Simplified Protocol Stack for LP-WAN

In some LPWAN technologies, only the Devs have a device ID. When such technologies are used, it is necessary to statically define an IID for the Link Local address for the SCHC C/D.

#### Rule 0

Field	FL	FP	DI	Value	Match	Comp	Decomp	Sent
					Opera.	Action		[bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent		
IPv6 DiffServ	8	1	Bi	0	equal	not-sent		
IPv6 Flow Label	20	1	Bi	0	equal	not-sent		
IPv6 Length	16	1	Bi		ignore	compute-*		
IPv6 Next Header	8	1	Bi	17	equal	not-sent		
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent		
IPv6 DevPrefix	64	1	Bi	FE80::/64	equal	not-sent		
IPv6 DevIID	64	1	Bi		ignore	DevIID		
IPv6 AppPrefix	64	1	Bi	FE80::/64	equal	not-sent		
IPv6 AppIID	64	1	Bi	::1	equal	not-sent		



UDP DevPort	16 1 Bi 123	equal	not-sent		
UDP AppPort	16 1 Bi 124	equal	not-sent		
UDP Length	16 1 Bi	ignore	compute-*		
UDP checksum	16 1 Bi	ignore	compute-*		
+=====+=====+=====+=====+=====+					

## Rule 1

Field	FL FP DI	Value	Match	Action	Sent
			Opera.	Action	[bits]
+-----+-----+-----+-----+-----+					
IPv6 Version	4 1 Bi 6	ignore	not-sent		
IPv6 DiffServ	8 1 Bi 0	equal	not-sent		
IPv6 Flow Label	20 1 Bi 0	equal	not-sent		
IPv6 Length	16 1 Bi	ignore	compute-*		
IPv6 Next Header	8 1 Bi 17	equal	not-sent		
IPv6 Hop Limit	8 1 Bi 255	ignore	not-sent		
IPv6 DevPrefix	64 1 Bi [alpha/64,	match-	mapping-sent		1
		fe80::/64]	mapping		
IPv6 DevIID	64 1 Bi	ignore	DevIID		
IPv6 AppPrefix	64 1 Bi [beta/64,	match-	mapping-sent		2
		alpha/64,	mapping		
		fe80::64]			
IPv6 AppIID	64 1 Bi ::1000	equal	not-sent		
+=====+=====+=====+=====+=====+					
UDP DevPort	16 1 Bi 5683	equal	not-sent		
UDP AppPort	16 1 Bi 5683	equal	not-sent		
UDP Length	16 1 Bi	ignore	compute-*		
UDP checksum	16 1 Bi	ignore	compute-*		
+=====+=====+=====+=====+=====+					

## Rule 2

Field	FL FP DI	Value	Match	Action	Sent
			Opera.	Action	[bits]
+-----+-----+-----+-----+-----+					
IPv6 Version	4 1 Bi 6	ignore	not-sent		
IPv6 DiffServ	8 1 Bi 0	equal	not-sent		
IPv6 Flow Label	20 1 Bi 0	equal	not-sent		
IPv6 Length	16 1 Bi	ignore	compute-*		
IPv6 Next Header	8 1 Bi 17	equal	not-sent		
IPv6 Hop Limit	8 1 Up 255	ignore	not-sent		
IPv6 Hop Limit	8 1 Dw	ignore	value-sent		8
IPv6 DevPrefix	64 1 Bi alpha/64	equal	not-sent		
IPv6 DevIID	64 1 Bi	ignore	DevIID		
IPv6 AppPrefix	64 1 Bi gamma/64	equal	not-sent		
IPv6 AppIID	64 1 Bi ::1000	equal	not-sent		
+=====+=====+=====+=====+=====+					



UDP DevPort	16 1  Bi 8720	MSB(12)  LSB	4
UDP AppPort	16 1  Bi 8720	MSB(12)  LSB	4
UDP Length	16 1  Bi	ignore   compute-*	
UDP checksum	16 1  Bi	ignore   compute-*	
+=====+==+==+==+=====+=====+=====+=====+			

Figure 26: Context Rules

All the fields described in the three Rules depicted on Figure 26 are present in the IPv6 and UDP headers. The DevIID-DID value is found in the L2 header.

The second and third Rules use global addresses. The way the Dev learns the prefix is not in the scope of the document.

The third Rule compresses each port number to 4 bits.

## [Appendix B](#). Fragmentation Examples

This section provides examples for the various fragment reliability modes specified in this document. In the drawings, Bitmaps are shown in their uncompressed form.

Figure 27 illustrates the transmission in No-ACK mode of a SCHC Packet that needs 11 SCHC Fragments. FCN is 1 bit wide.

```

Sender                                Receiver
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=0----->|
|-----FCN=1 + RCS ---->| Integrity check: success
(End)

```

Figure 27: No-ACK mode, 11 SCHC Fragments

In the following examples, N (the size of the FCN field) is 3 bits. The All-1 FCN value is 7.



Figure 28 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW\_SIZE=7 and no lost SCHC Fragment.

Sender	Receiver
-----W=0, FCN=6----->	
-----W=0, FCN=5----->	
-----W=0, FCN=4----->	
-----W=0, FCN=3----->	
-----W=0, FCN=2----->	
-----W=0, FCN=1----->	
-----W=0, FCN=0----->	
(no ACK)	
-----W=1, FCN=6----->	
-----W=1, FCN=5----->	
-----W=1, FCN=4----->	
--W=1, FCN=7 + RCS-->	Integrity check: success
<-- ACK, W=1, C=1 ---	C=1
(End)	

Figure 28: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, no lost SCHC Fragment.

Figure 29 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW\_SIZE=7 and three lost SCHC Fragments.





```

Sender              Receiver
|-----W=0, FCN=6----->|
|-----W=0, FCN=5----->|
|-----W=0, FCN=4--X-->|
|-----W=0, FCN=3----->|
|-----W=0, FCN=2--X-->|
|-----W=0, FCN=1----->|
|-----W=0, FCN=0----->|          6543210
|<-- ACK, W=0, C=0 ---| Bitmap:1101011
|-----W=0, FCN=4----->|
|-----W=0, FCN=2----->|
(no ACK)
|-----W=1, FCN=6----->|
|-----W=1, FCN=5----->|
|-----W=1, FCN=4--X-->|
|- W=1, FCN=7 + RCS ->| Integrity check: failure
|<-- ACK, W=1, C=0 ---| C=0, Bitmap:1100001
|-----W=1, FCN=4----->| Integrity check: success
|<-- ACK, W=1, C=1 ---| C=1
(End)

```

Figure 29: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, lost SCHC Fragments.

Figure 30 shows an example of a transmission in ACK-on-Error mode of a SCHC Packet fragmented in 73 tiles, with N=5, WINDOW\_SIZE=28, M=2 and 3 lost SCHC Fragments.



Sender	Receiver
-----W=0, FCN=27----->	4 tiles sent
-----W=0, FCN=23----->	4 tiles sent
-----W=0, FCN=19----->	4 tiles sent
-----W=0, FCN=15--X-->	4 tiles sent (not received)
-----W=0, FCN=11----->	4 tiles sent
-----W=0, FCN=7 ----->	4 tiles sent
-----W=0, FCN=3 ----->	4 tiles sent
-----W=1, FCN=27----->	4 tiles sent
-----W=1, FCN=23----->	4 tiles sent
-----W=1, FCN=19----->	4 tiles sent
-----W=1, FCN=15----->	4 tiles sent
-----W=1, FCN=11----->	4 tiles sent
-----W=1, FCN=7 ----->	4 tiles sent
-----W=1, FCN=3 --X-->	4 tiles sent (not received)
-----W=2, FCN=27----->	4 tiles sent
-----W=2, FCN=23----->	4 tiles sent
^  -----W=2, FCN=19----->	1 tile sent
-----W=2, FCN=18----->	1 tile sent
-----W=2, FCN=17----->	1 tile sent
-----W=2, FCN=16----->	1 tile sent
s  -----W=2, FCN=15----->	1 tile sent
m  -----W=2, FCN=14----->	1 tile sent
a  -----W=2, FCN=13--X-->	1 tile sent (not received)
l  -----W=2, FCN=12----->	1 tile sent
l  ---W=2, FCN=31 + RCS->	Integrity check: failure
e  <--- ACK, W=0, C=0 ---	C=0, Bitmap:1111111111110000111111111111
r  -----W=0, FCN=15----->	1 tile sent
-----W=0, FCN=14----->	1 tile sent
L  -----W=0, FCN=13----->	1 tile sent
2  -----W=0, FCN=12----->	1 tile sent
<--- ACK, W=1, C=0 ---	C=0, Bitmap:11111111111111111111111111110000
M  -----W=1, FCN=3 ----->	1 tile sent
T  -----W=1, FCN=2 ----->	1 tile sent
U  -----W=1, FCN=1 ----->	1 tile sent
-----W=1, FCN=0 ----->	1 tile sent
<--- ACK, W=2, C=0 ---	C=0, Bitmap:11111111111111111010000000000001
-----W=2, FCN=13----->	Integrity check: success
V  <--- ACK, W=2, C=1 ---	C=1

(End)

Figure 30: ACK-on-Error mode, variable MTU.

In this example, the L2 MTU becomes reduced just before sending the "W=2, FCN=19" fragment, leaving space for only 1 tile in each forthcoming SCHC Fragment. Before retransmissions, the 73 tiles are carried by a total of 25 SCHC Fragments, the last 9 being of smaller size.



Note: other sequences of events (e.g. regarding when ACKs are sent by the Receiver) are also allowed by this specification. Profiles may restrict this flexibility.

Figure 31 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, with  $N=3$ ,  $WINDOW\_SIZE=7$  and no loss.

Sender	Receiver
-----W=0, FCN=6----->	
-----W=0, FCN=5----->	
-----W=0, FCN=4----->	
-----W=0, FCN=3----->	
-----W=0, FCN=2----->	
-----W=0, FCN=1----->	
-----W=0, FCN=0----->	
<-- ACK, W=0, C=0 ---	Bitmap:1111111
-----W=1, FCN=6----->	
-----W=1, FCN=5----->	
-----W=1, FCN=4----->	
--W=1, FCN=7 + RCS-->	Integrity check: success
<-- ACK, W=1, C=1 ---	C=1
(End)	

Figure 31: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, no loss.

Figure 32 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment,  $N=3$ ,  $WINDOW\_SIZE=7$  and three lost SCHC Fragments.



```

Sender                      Receiver
|-----W=0, FCN=6----->|
|-----W=0, FCN=5----->|
|-----W=0, FCN=4--X-->|
|-----W=0, FCN=3----->|
|-----W=0, FCN=2--X-->|
|-----W=0, FCN=1----->|
|-----W=0, FCN=0----->|          6543210
|<-- ACK, W=0, C=0 ---| Bitmap:1101011
|-----W=0, FCN=4----->|
|-----W=0, FCN=2----->|
|<-- ACK, W=0, C=0 ---| Bitmap:1111111
|-----W=1, FCN=6----->|
|-----W=1, FCN=5----->|
|-----W=1, FCN=4--X-->|
|--W=1, FCN=7 + RCS-->| Integrity check: failure
|<-- ACK, W=1, C=0 ---| C=0, Bitmap:11000001
|-----W=1, FCN=4----->| Integrity check: success
|<-- ACK, W=1, C=1 ---| C=1
(End)

```

Figure 32: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 33 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3, WINDOW\_SIZE=7, three lost SCHC Fragments and only one retry needed to recover each lost SCHC Fragment.

```

Sender                      Receiver
|-----W=0, FCN=6----->|
|-----W=0, FCN=5----->|
|-----W=0, FCN=4--X-->|
|-----W=0, FCN=3--X-->|
|-----W=0, FCN=2--X-->|
|--W=0, FCN=7 + RCS-->| Integrity check: failure
|<-- ACK, W=0, C=0 ---| C=0, Bitmap:1100001
|-----W=0, FCN=4----->| Integrity check: failure
|-----W=0, FCN=3----->| Integrity check: failure
|-----W=0, FCN=2----->| Integrity check: success
|<-- ACK, W=0, C=1 ---| C=1
(End)

```

Figure 33: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 34 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3,





WINDOW\_SIZE=7, three lost SCHC Fragments, and the second SCHC ACK lost.

Sender	Receiver
-----W=0, FCN=6----->	
-----W=0, FCN=5----->	
-----W=0, FCN=4--X-->	
-----W=0, FCN=3--X-->	
-----W=0, FCN=2--X-->	
--W=0, FCN=7 + RCS-->	Integrity check: failure
<-- ACK, W=0, C=0 ---	C=0, Bitmap:1100001
-----W=0, FCN=4----->	Integrity check: failure
-----W=0, FCN=3----->	Integrity check: failure
-----W=0, FCN=2----->	Integrity check: success
<-X-ACK, W=0, C=1 ---	C=1
timeout	
--- W=0, ACK REQ --->	ACK REQ
<-- ACK, W=0, C=1 ---	C=1
(End)	

Figure 34: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, SCHC ACK loss.

Figure 35 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with N=3, WINDOW\_SIZE=7, with three lost SCHC Fragments, and one retransmitted SCHC Fragment lost again.

Sender	Receiver
-----W=0, FCN=6----->	
-----W=0, FCN=5----->	
-----W=0, FCN=4--X-->	
-----W=0, FCN=3--X-->	
-----W=0, FCN=2--X-->	
--W=0, FCN=7 + RCS-->	Integrity check: failure
<-- ACK, W=0, C=0 ---	C=0, Bitmap:1100001
-----W=0, FCN=4----->	Integrity check: failure
-----W=0, FCN=3----->	Integrity check: failure
-----W=0, FCN=2--X-->	
timeout	
--- W=0, ACK REQ --->	ACK REQ
<-- ACK, W=0, C=0 ---	C=0, Bitmap: 1111101
-----W=0, FCN=2----->	Integrity check: success
<-- ACK, W=0, C=1 ---	C=1
(End)	

Figure 35: ACK-Always mode, 6 tiles, retransmitted SCHC Fragment lost again.



Figure 36 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 28 tiles, with one tile per SCHC Fragment, N=5, WINDOW\_SIZE=24 and two lost SCHC Fragments.

Sender	Receiver
-----W=0, FCN=23----->	
-----W=0, FCN=22----->	
-----W=0, FCN=21--X-->	
-----W=0, FCN=20----->	
-----W=0, FCN=19----->	
-----W=0, FCN=18----->	
-----W=0, FCN=17----->	
-----W=0, FCN=16----->	
-----W=0, FCN=15----->	
-----W=0, FCN=14----->	
-----W=0, FCN=13----->	
-----W=0, FCN=12----->	
-----W=0, FCN=11----->	
-----W=0, FCN=10--X-->	
-----W=0, FCN=9 ----->	
-----W=0, FCN=8 ----->	
-----W=0, FCN=7 ----->	
-----W=0, FCN=6 ----->	
-----W=0, FCN=5 ----->	
-----W=0, FCN=4 ----->	
-----W=0, FCN=3 ----->	
-----W=0, FCN=2 ----->	
-----W=0, FCN=1 ----->	
-----W=0, FCN=0 ----->	
<--- ACK, W=0, C=0 ---	Bitmap:11011111111111101111111111
-----W=0, FCN=21----->	
-----W=0, FCN=10----->	
<--- ACK, W=0, C=0 ---	Bitmap:11111111111111111111111111
-----W=1, FCN=23----->	
-----W=1, FCN=22----->	
-----W=1, FCN=21----->	
--W=1, FCN=31 + RCS-->	Integrity check: success
<--- ACK, W=1, C=1 ---	C=1

(End)

Figure 36: ACK-Always mode, 28 tiles, one tile per SCHC Fragment, lost SCHC Fragments.



## Appendix C. Fragmentation State Machines

The fragmentation state machines of the sender and the receiver, one for each of the different reliability modes, are described in the following figures:

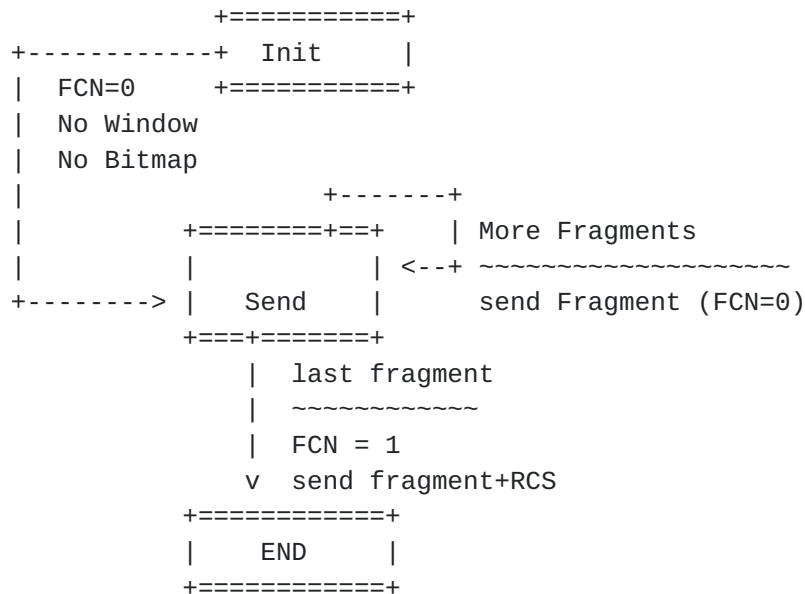


Figure 37: Sender State Machine for the No-ACK Mode

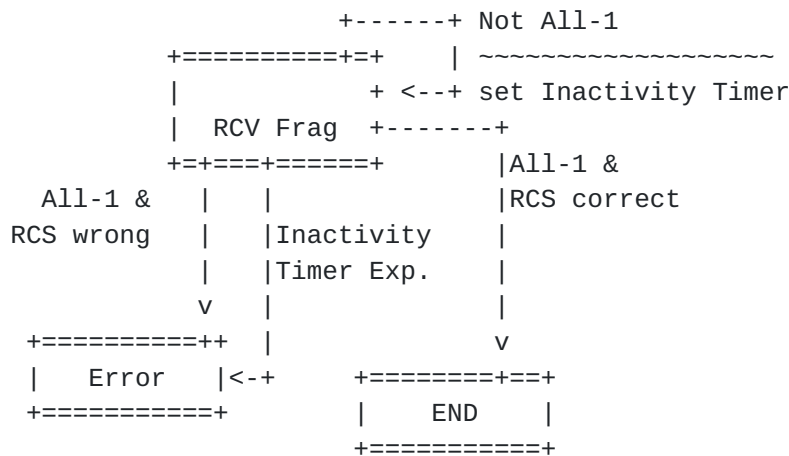


Figure 38: Receiver State Machine for the No-ACK Mode



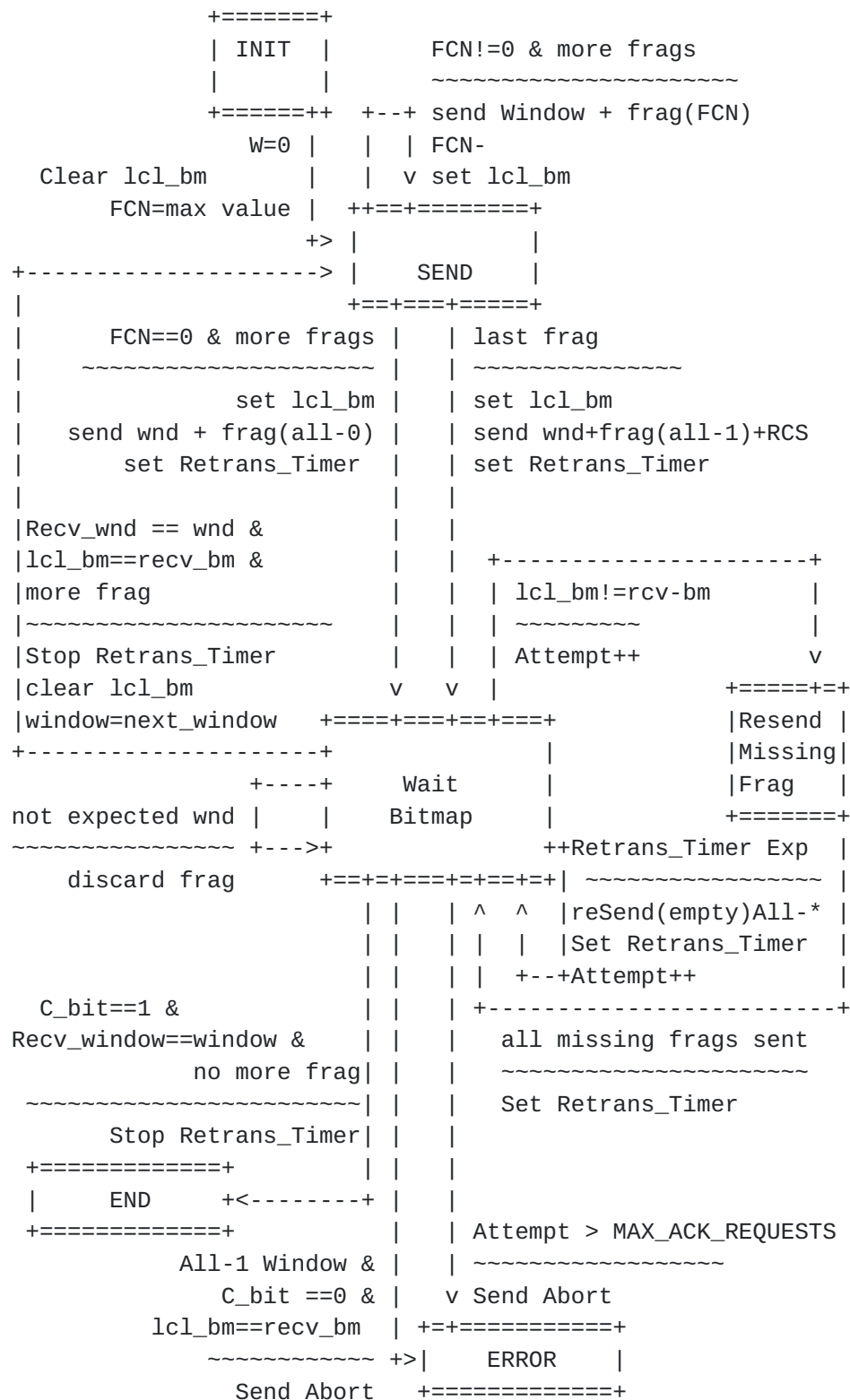


Figure 39: Sender State Machine for the ACK-Always Mode





```

Not All- & w=expected +---+ +---+w = Not expected
~~~~~| | | |~~~~~
Set lcl_bm(FCN) | v v |discard
          +---+
+-----+ Rcv +--->* ABORT
| +-----+ Window |
| | +-----+
| | All-0 & w=expect | ^ w =next & not-All
| | ~~~~~| |~~~~~
| | set lcl_bm(FCN) | |expected = next window
| | send lcl_bm | |Clear lcl_bm
| | | |
| | w=expected & not-All | |
| | ~~~~~| |
| | set lcl_bm(FCN)+-+ | | +---+ w=next & All-0
| | if lcl_bm full | | | |~~~~~
| | send lcl_bm | | | |expected = nxt wnd
| | v | v | | |Clear lcl_bm
| |w=expected& All-1 +---+ +---+ | set lcl_bm(FCN)
| | ~~~~~ +->+ Wait +<+ send lcl_bm
| | discard +-+ Next |
| | All-0 +-----+ Window +--->* ABORT
| | ~~~~~ +----->+-----+
| | snd lcl_bm All-1 & w=next| | All-1 & w=nxt
| | & RCS wrong| | & RCS right
| | ~~~~~| |~~~~~
| | set lcl_bm(FCN)| |set lcl_bm(FCN)
| | send lcl_bm| |send lcl_bm
| | |
| | +-----+
| | All-1 & w=expected | | | |
| | & RCS wrong v +---+ w=expected & |
| | ~~~~~ +-----+ | RCS wrong |
| | set lcl_bm(FCN) | | +<+ ~~~~~ |
| | send lcl_bm | Wait End | set lcl_bm(FCN)|
| | +----->+ +--->* ABORT |
| | +-----+ All-1&RCS wrong|
| | | ^ | ~~~~~|
| | w=expected & RCS right | +---+ send lcl_bm |
| | ~~~~~| |
| | set lcl_bm(FCN) | +-+ Not All-1 |
| | send lcl_bm | | |~~~~~|
| | | | discard |
| | All-1&w=expected & RCS right | | |
| | ~~~~~ v | v +-----+All-1 |
| | set lcl_bm(FCN) +---+ +---+ |~~~~~|
| | send lcl_bm | +<+Send lcl_bm |
| | +----->+ END |
| | +-----+<-----+

```



```

--->* ABORT
~~~~~
Inactivity_Timer = expires
When DWL
  IF Inactivity_Timer expires
    Send DWL Request
    Attempt++

```

Figure 40: Receiver State Machine for the ACK-Always Mode

```

+=====+
|      |
| INIT  |
|      |
+=====+
|      | FCN!=0 & more frags
+=====+
Frag RuleID trigger | +--+ Send cur_W + frag(FCN);
~~~~~| | FCN--;
cur_W=0; FCN=max_value;| | | set [cur_W, cur_Bmp]
clear [cur_W, Bmp_n];| | v
clear rcv_Bmp | ++++++=====+ **BACK_TO_SEND
+-->+ | cur_W==rcv_W &
**BACK_TO_SEND | SEND | [cur_W,Bmp_n]==rcv_Bmp
+----->+ | & more frags
| +----->+ | ~~~~~
| | ++++++=====+ cur_W++;
| | FCN==0 & more frags| |last frag clear [cur_W, Bmp_n]
| | ~~~~~
| | set cur_Bmp; | |set [cur_W, Bmp_n];
| |send cur_W + frag(All-0);| |send cur_W + frag(All-1)+RCS;
| | set Retrans_Timer| |set Retrans_Timer
| | +-----+
| |Retrans_Timer expires & | |cur_W==rcv_W&[cur_W,Bmp_n]!=rcv_Bmp|
| |more Frags | | ~~~~~
| | ~~~~~
| | Attempts++; W=cur_W
| |stop Retrans_Timer; | | +-----+ rcv_W==Wn &
| |[cur_W,Bmp_n]==cur_Bmp; v v | | [Wn,Bmp_n]!=rcv_Bmp|
| |cur_W++ ++++++=====+ ++++++=====+ ~~~~~
| +-----+ | | Resend | Attempts++;|
+-----+ Wait x ACK | | Missing | W=Wn |
+----->+ | | Frags(W) +<-----+
| rcv_W==Wn &+--+ ++++++=====+
| [Wn,Bmp_n]!=rcv_Bmp| ++++++=====+
| ~~~~~ ^ | | ^ |
| send (cur_W,+--+ | | +-----+
| ALL-0-empty) | | | all missing frag sent(W)
| ~~~~~
| Retrans_Timer expires &| | | set Retrans_Timer

```



```

|           No more Frags|   |   |
|           ~~~~~~|   |   |
|       stop Retrans_Timer;|   |   |
|(re)send frag(All-1)+RCS |   |   |
+-----+   |   |
|           cur_W==rcv_W&|   |   |
|   [cur_W,Bmp_n]==rcv_Bmp&|   | Attempts > MAX_ACK_REQUESTS
| No more Frags & RCS flag==OK|   | ~~~~~~
|           ~~~~~~|   | send Abort
+=====+stop Retrans_Timer|   | +=====+
|   END   +<-----+   |   | +-->+   ERROR   |
+=====+   |   |   |   | +=====+

```

Figure 41: Sender State Machine for the ACK-on-Error Mode

This is an example only. It is not normative. The specification in [Section 8.4.3.1](#) allows for sequences of operations different from the one shown here.



```

+=====+          New frag RuleID received
|          | ~~~~~~
| INIT +-----+cur_W=0;clear([cur_W,Bmp_n]);
+=====+          |sync=0
|
Not All* & rcv_W==cur_W+---+ | +---+
~~~~~ | | | (E)
set[cur_W,Bmp_n(FCN)] | v v v |
+=====+
+-----+ +---+ All-0&Full[cur_W,Bmp_n]
|          ABORT *<---+ Rcv Window | | ~~~~~~
| +-----+ +<--+ cur_W++;set Inact_timer;
| |          +->+=====+ clear [cur_W,Bmp_n]
| | All-0 empty(Wn)| | | ^ ^
| | ~~~~~~ +-----+ | | | rcv_W==cur_W & sync==0;
| | sendACK([Wn,Bmp_n]) | | | |& Full([cur_W,Bmp_n])
| | | | |& All* || last_miss_frag
| | | | ~~~~~~
| | All* & rcv_W==cur_W|(C)| |sendACK([cur_W,Bmp_n]);
| | & sync==0| | | |cur_W++; clear([cur_W,Bmp_n])
| |&no_full([cur_W,Bmp_n])| |(E)| | |
| | ~~~~~~ | | | |
| | sendACK([cur_W,Bmp_n])| | | | Error/ |
| | | | | +-----+ | Abort |
| | | | v v | | | +====+=====+
| | | | +=====+ (D) ^
| | | | +---+ Wait x | | |
| | All-0 empty(Wn)+->| Missing Frags |<--+
| | ~~~~~~ +=====+ |
| | sendACK([Wn,Bmp_n]) +-----+
| | |
v v
(A)(B)

```

(C) All\* & sync>0

~~~~~

```

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n])

```

(D) All\* || last\_miss\_frag

& rcv\_W!=cur\_W & sync>0

& Full([rcv\_W,Bmp\_n])

~~~~~

```

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n]);sync--

```

ABORT-->\* Uplink Only &

Inact\_Timer expires

(E) Not All\* & rcv\_W!=cur\_W

|| Attempts > MAX\_ACK\_REQUESTS

~~~~~

```

sync++; cur_W=rcv_W;
set[cur_W,Bmp_n(FCN)]

```

send Abort





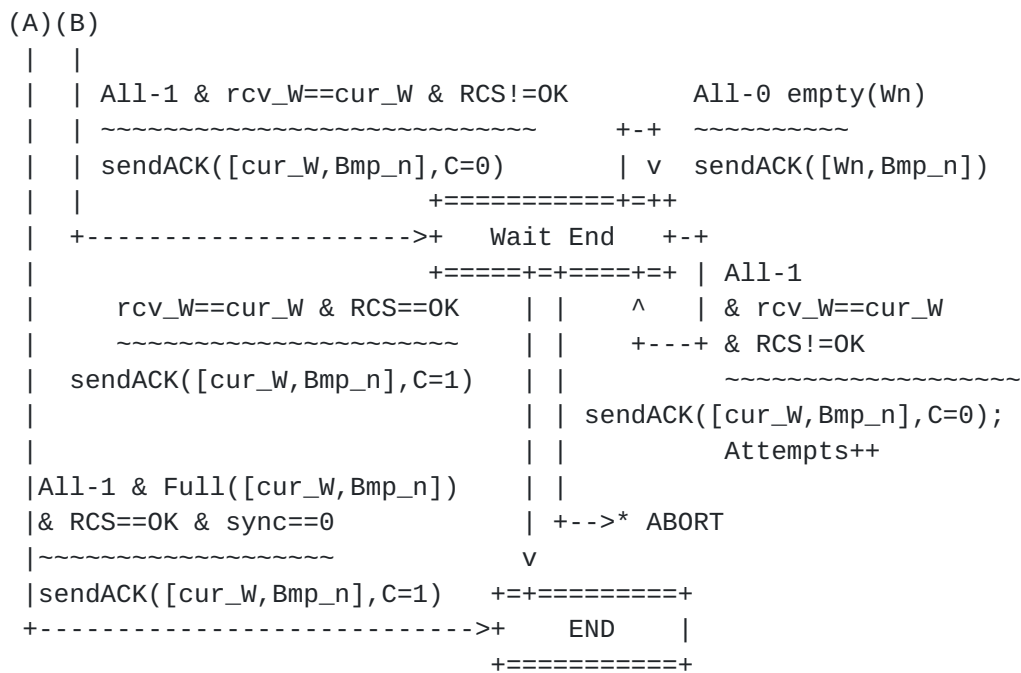


Figure 42: Receiver State Machine for the ACK-on-Error Mode

#### Appendix D. SCHC Parameters

This section lists the information that needs to be provided in the LPWAN technology-specific documents.

- o Most common uses cases, deployment scenarios
- o Mapping of the SCHC architectural elements onto the LPWAN architecture
- o Assessment of LPWAN integrity checking
- o Various potential channel conditions for the technology and the corresponding recommended use of SCHC C/D and F/R

This section lists the parameters that need to be defined in the Profile.

- o Rule ID numbering scheme, fixed-sized or variable-sized Rule IDs, number of Rules, the way the Rule ID is transmitted
- o maximum packet size that should ever be reconstructed by SCHC Decompression (MAX\_PACKET\_SIZE). See [Section 12](#).



- o Padding: size of the L2 Word (for most LPWAN technologies, this would be a byte; for some technologies, a bit)
- o Decision to use SCHC fragmentation mechanism or not. If yes:
  - \* reliability mode(s) used, in which cases (e.g. based on link channel condition)
  - \* Rule ID values assigned to each mode in use
  - \* presence and number of bits for DTag (T) for each Rule ID value
  - \* support for interleaved packet transmission, to what extent
  - \* WINDOW\_SIZE, for modes that use windows
  - \* number of bits for W (M) for each Rule ID value, for modes that use windows
  - \* number of bits for FCN (N) for each Rule ID value
  - \* size of RCS and algorithm for its computation, for each Rule ID, if different from the default CRC32. Byte fill-up with zeroes or other mechanism, to be specified.
  - \* Retransmission Timer duration for each Rule ID value, if applicable to the SCHC F/R mode
  - \* Inactivity Timer duration for each Rule ID value, if applicable to the SCHC F/R mode
  - \* MAX\_ACK\_REQUEST value for each Rule ID value, if applicable to the SCHC F/R mode
- o if L2 Word is wider than a bit and SCHC fragmentation is used, value of the padding bits (0 or 1). This is needed because the padding bits of the last fragment are included in the RCS computation.

A Profile may define a delay to be added after each SCHC message transmission for compliance with local regulations or other constraints imposed by the applications.

- o In some LPWAN technologies, as part of energy-saving techniques, downlink transmission is only possible immediately after an uplink transmission. In order to avoid potentially high delay in the downlink transmission of a fragmented SCHC Packet, the SCHC Fragment receiver may perform an uplink transmission as soon as



possible after reception of a SCHC Fragment that is not the last one. Such uplink transmission may be triggered by the L2 (e.g. an L2 ACK sent in response to a SCHC Fragment encapsulated in a L2 PDU that requires an L2 ACK) or it may be triggered from an upper layer.

- o the following parameters need to be addressed in documents other than this one but not necessarily in the LPWAN technology-specific documents:

- \* The way the Contexts are provisioned
- \* The way the Rules are generated

#### **Appendix E. Supporting multiple window sizes for fragmentation**

For ACK-Always or ACK-on-Error, implementers may opt to support a single window size or multiple window sizes. The latter, when feasible, may provide performance optimizations. For example, a large window size should be used for packets that need to be split into a large number of tiles. However, when the number of tiles required to carry a packet is low, a smaller window size, and thus a shorter Bitmap, may be sufficient to provide reception status on all tiles. If multiple window sizes are supported, the Rule ID may signal the window size in use for a specific packet transmission.

The same window size MUST be used for the transmission of all tiles that belong to the same SCHC Packet.

#### **Appendix F. Downlink SCHC Fragment transmission**

For downlink transmission of a fragmented SCHC Packet in ACK-Always mode, the SCHC Fragment receiver may support timer-based SCHC ACK retransmission. In this mechanism, the SCHC Fragment receiver initializes and starts a timer (the Inactivity Timer is used) after the transmission of a SCHC ACK, except when the SCHC ACK is sent in response to the last SCHC Fragment of a packet (All-1 fragment). In the latter case, the SCHC Fragment receiver does not start a timer after transmission of the SCHC ACK.

If, after transmission of a SCHC ACK that is not an All-1 fragment, and before expiration of the corresponding Inactivity timer, the SCHC Fragment receiver receives a SCHC Fragment that belongs to the current window (e.g. a missing SCHC Fragment from the current window) or to the next window, the Inactivity timer for the SCHC ACK is stopped. However, if the Inactivity timer expires, the SCHC ACK is resent and the Inactivity timer is reinitialized and restarted.



The default initial value for the Inactivity Timer, as well as the maximum number of retries for a specific SCHC ACK, denoted MAX\_ACK\_RETRIES, are not defined in this document, and need to be defined in a Profile. The initial value of the Inactivity timer is expected to be greater than that of the Retransmission timer, in order to make sure that a (buffered) SCHC Fragment to be retransmitted can find an opportunity for that transmission. One exception to this recommendation is the special case of the All-1 SCHC Fragment transmission.

When the SCHC Fragment sender transmits the All-1 SCHC Fragment, it starts its Retransmission Timer with a large timeout value (e.g. several times that of the initial Inactivity Timer). If a SCHC ACK is received before expiration of this timer, the SCHC Fragment sender retransmits any lost SCHC Fragments reported by the SCHC ACK, or if the SCHC ACK confirms successful reception of all SCHC Fragments of the last window, the transmission of the fragmented SCHC Packet is considered complete. If the timer expires, and no SCHC ACK has been received since the start of the timer, the SCHC Fragment sender assumes that the All-1 SCHC Fragment has been successfully received (and possibly, the last SCHC ACK has been lost: this mechanism assumes that the Retransmission Timer for the All-1 SCHC Fragment is long enough to allow several SCHC ACK retries if the All-1 SCHC Fragment has not been received by the SCHC Fragment receiver, and it also assumes that it is unlikely that several ACKs become all lost).

#### Authors' Addresses

Ana Minaburo  
Acklio  
1137A avenue des Champs Blancs  
35510 Cesson-Sevigne Cedex  
France

Email: ana@ackl.io

Laurent Toutain  
IMT-Atlantique  
2 rue de la Chataigneraie  
CS 17607  
35576 Cesson-Sevigne Cedex  
France

Email: Laurent.Toutain@imt-atlantique.fr





Carles Gomez  
Universitat Politecnica de Catalunya  
C/Esteve Terradas, 7  
08860 Castelldefels  
Spain

Email: carlesgo@entel.upc.edu

Dominique Barthel  
Orange Labs  
28 chemin du Vieux Chene  
38243 Meylan  
France

Email: dominique.barthel@orange.com

Juan Carlos Zuniga  
SIGFOX  
425 rue Jean Rostand  
Labège 31670  
France

Email: JuanCarlos.Zuniga@sigfox.com

