lpwan Working Group   A. Minaburo
Internet-Draft   Acklio
Intended status: Informational   L. Toutain
Expires: September 1, 2018   IMT-Atlantique
                                                                C. Gomez
                                    Universitat Politecnica de Catalunya
                                                       February 28, 2018

LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP
draft-ietf-lpwan-ipv6-static-context-hc-10

Abstract

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

SCHC compression is based on a common static context stored in LPWAN devices and in the network. This document applies SCHC compression to IPv6/UDP headers. This document also specifies a fragmentation and reassembly mechanism that is used to support the IPv6 MTU requirement over LPWAN technologies. Fragmentation is mandatory for IPv6 datagrams that, after SCHC compression or when it has not been possible to apply such compression, still exceed the layer two maximum payload size.

The SCHC header compression mechanism is independent of the specific LPWAN technology over which it will be used. Note that this document defines generic functionality. This document purposefully offers flexibility with regard to parameter settings and mechanism choices, that are expected to be made in other, technology-specific, documents.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a header compression scheme and fragmentation functionality, both specially tailored for Low Power Wide Area Networks (LPWAN).

Header compression is needed to efficiently bring Internet connectivity to the node within an LPWAN network. Some LPWAN networks properties can be exploited to get an efficient header compression:

o The topology is star-oriented which means that all packets follow the same path. For the necessity of this draft, the architecture is simple and is described as Devices (Dev) exchanging information

with LPWAN Application Servers (App) through Network Gateways (NGW).

- o The traffic flows can be known in advance since devices embed built-in applications. New applications cannot be easily installed in LPWAN devices, as they would in computers or smartphones.

The Static Context Header Compression (SCHC) is defined for this environment. SCHC uses a context, where header information is kept in the header format order. This context is static: the values of the header fields do not change over time. This avoids complex resynchronization mechanisms, that would be incompatible with LPWAN characteristics. In most cases, a small context 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.

LPWAN technologies impose some strict limitations on traffic. For instance, devices are sleeping most of the time and MAY receive data during short periods of time after transmission to preserve battery. LPWAN technologies are also characterized, among others, by a very reduced data unit and/or payload size [I-D.ietf-lpwan-overview]. However, some of these technologies do not provide fragmentation functionality, therefore the only option for them to support the IPv6 MTU requirement of 1280 bytes [RFC2460] is to use a fragmentation protocol at the adaptation layer, below IPv6. In response to this need, this document also defines a fragmentation/reassembly mechanism, which supports the IPv6 MTU requirement over LPWAN technologies. Such functionality has been designed under the assumption that data unit out-of-sequence delivery will not happen between the entity performing fragmentation and the entity performing reassembly.

Note that this document defines generic functionality and purposefully offers flexibility with regard to parameter settings and mechanism choices, that are expected to be made in other, technology-specific documents.

2. LPWAN Architecture

LPWAN technologies have similar network architectures but different terminology. 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 LPWAN-AAA Server, which controls the user authentication and the applications.

o Application Server (App)

```
                                              +------+
    ()   ()   ()       |                      |LPWAN-|
     ()  () () ()     / \       +---------+   | AAA  |
   () () () () () () /   \======|    ^    |===|Server|  +-----------+
    ()  ()   ()     |           | <--|--> |   +------+  |APPLICATION|
   ()  ()  ()  ()  / \==========|    v    |=============|   (App)   |
     ()  ()  ()   /   \         +---------+             +-----------+
    Dev        Radio Gateways         NGW
```

Figure 1: LPWAN Architecture

3. Terminology

This section defines the terminology and acronyms used in this document.

o Abort. A SCHC fragment format to signal the other end-point that the on-going fragment transmission is stopped and finished.

o ACK (Acknowledgment). A SCHC fragment format used to report the success or unsuccess reception of a set of SCHC fragments.

o All-0. The SCHC fragment format for the last frame of a window that is not the last one of a packet (see Window in this glossary).

o All-1. The SCHC fragment format for the last frame of the packet.

o All-0 empty. An All-0 SCHC fragment without a payload. It is used to request the ACK with the encoded Bitmap when the Retransmission Timer expires, in a window that is not the last one of a packet.

o All-1 empty. An All-1 SCHC fragment without a payload. It is used to request the ACK with the encoded Bitmap when the Retransmission Timer expires in the last window of a packet.

- App: LPWAN Application. An application sending/receiving IPv6 packets to/from the Device.

- APP-IID: Application Interface Identifier. Second part of the IPv6 address that identifies the application server interface.

- Bi: Bidirectional, a rule entry that applies to headers of packets travelling in both directions (Up and Dw).

- Bitmap: a field of bits in an acknowledgment message that tells the sender which SCHC fragments of a window were correctly received.

- C: Checked bit. Used in an acknowledgment (ACK) header to determine if the MIC locally computed by the receiver matches (1) the received MIC or not (0).

- CDA: Compression/Decompression Action. Describes the reciprocal pair of actions that are performed at the compressor to compress a header field and at the decompressor to recover the original header field value.

- Compress Residue. The bytes that need to be sent after applying the SCHC compression over each header field

- Context: A set of rules used to compress/decompress headers.

- Dev: Device. A node connected to the LPWAN. A Dev SHOULD implement SCHC.

- Dev-IID: Device Interface Identifier. Second part of the IPv6 address that identifies the device interface.

- DI: Direction Indicator. This field tells which direction of packet travel (Up, Dw or Bi) a rule applies to. This allows for assymmetric processing.

- DTag: Datagram Tag. This SCHC fragmentation header field is set to the same value for all SCHC fragments carrying the same IPv6 datagram.

- Dw: Dw: Downlink direction for compression/decompression in both sides, from SCHC C/D in the network to SCHC C/D in the Dev.

- FCN: Fragment Compressed Number. This SCHC fragmentation header field carries an efficient representation of a larger-sized fragment number.

- Field Description. A line in the Rule Table.

- FID: Field Identifier. This is an index to describe the header fields in a Rule.

- FL: Field Length is the length of the field in bits for fixed values or a type (variable, token length, ...) for length unknown at the rule creation. The length of a header field is defined in the specific protocol standard.

- FP: Field Position is a value that is used to identify the position where each instance of a field appears in the header.

- SCHC Fragment: A data unit that carries a subset of a SCHC packet. SCHC Fragmentation is needed when the size of a SCHC packet exceeds the available payload size of the underlying L2 technology data unit.

- IID: Interface Identifier. See the IPv6 addressing architecture [RFC7136]

- Inactivity Timer. A timer used after receiving a SCHC fragment to detect when there is an error and there is no possibility to continue an on-going SCHC fragmented packet transmission.

- L2: Layer two. The immediate lower layer SCHC interfaces with. It is provided by an underlying LPWAN technology.

- MIC: Message Integrity Check. A SCHC fragmentation header field computed over an IPv6 packet before fragmentation, used for error detection after IPv6 packet reassembly.

- MO: Matching Operator. An operator used to match a value contained in a header field with a value contained in a Rule.

- Retransmission Timer. A timer used by the SCHC fragment sender during an on-going SCHC fragmented packet transmission to detect possible link errors when waiting for a possible incoming ACK.

- Rule: A set of header field values.

- Rule entry: A row in the rule that describes a header field.

- Rule ID: An identifier for a rule, SCHC C/D in both sides share the same Rule ID for a specific packet. A set of Rule IDs are used to support SCHC fragmentation functionality.

o SCHC C/D: Static Context Header Compression Compressor/ Decompressor. A mechanism used in both sides, at the Dev and at the network to achieve Compression/Decompression of headers. SCHC C/D uses SCHC rules to perform compression and decompression.

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).

o TV: Target value. A value contained in the Rule that will be matched with the value of a header field.

o Up: Uplink direction for compression/decompression in both sides, from the Dev SCHC C/D to the network SCHC C/D.

o W: Window bit. A SCHC fragment header field used in Window mode ({Frag}), which carries the same value for all SCHC fragments of a window.

o Window: A subset of the SCHC fragments needed to carry a packet ({Frag}).

4. SCHC overview

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

```
             +----------------+
             |      IPv6      |
          +- +----------------+
          |  |   Compression  |
    SCHC <   +----------------+
          |  |  Fragmentation |
          +- +----------------+
             |LPWAN technology|
             +----------------+
```

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

As per this document, when a packet (e.g. an IPv6 packet) needs to be transmitted, header compression is first applied to the packet. The resulting packet after header compression (whose header MAY actually be smaller than that of the original packet or not) is called a SCHC packet. Subsequently, and if the SCHC packet size exceeds the layer 2 (L2) MTU, fragmentation is then applied to the SCHC packet. This process is illustrated by Figure 3

```
       A packet (e.g. an IPv6 packet)
                  |
                  V
     +------------------------------+
     |SCHC Compression/Decompression|
     +------------------------------+
                  |
              SCHC packet
                  |
                  V
         +------------------+
         |SCHC Fragmentation|  (if needed)
         +------------------+
                  |
                  V
            SCHC Fragment(s) (if needed)
```

Figure 3: SCHC operations from a sender point of view: header compression and fragmentation

5. Rule ID

Rule ID are identifiers used to select either the correct context to be used for Compression/Decompression functionalities or for SCHC Fragmentation or after trying to do SCHC C/D and SCHC fragmentation the packet is sent as is. The size of the Rule ID 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.

The Rule IDs identifiers are: * In the SCHC C/D context the Rule used to keep the Field Description of the header packet.

o In SCHC Fragmentation to identify the specific modes and settings. In bidirectional SCHC fragmentation at least two Rules ID are needed.

   o And at least one Rule ID MAY be reserved to the case where no SCHC C/D nor SCHC fragmentation were possible.

6. Static Context Header Compression

In order to perform header compression, this document defines a mechanism called Static Context Header Compression (SCHC), which is based on using context, i.e. a set of rules to compress or decompress headers. SCHC avoids context synchronization, which is the most bandwidth-consuming operation in other header compression mechanisms such as RoHC [RFC5795]. Since the nature of packets are highly predictable in LPWAN networks, static contexts MAY be stored beforehand to omit transmitting some information over the air. The contexts MUST be stored at both ends, and they can either be learned by a provisioning protocol, by out of band means, or they can be pre-provisioned. The way the contexts are provisioned on both ends is out of the scope of this document.

```
        Dev                                               App
   +----------------+                                +--------------+
   | APP1 APP2 APP3 |                                |APP1 APP2 APP3|
   |                |                                |              |
   |       UDP      |                                |     UDP      |
   |      IPv6      |                                |    IPv6      |
   |                |                                |              |
   |SCHC Comp / Frag|                                |              |
   +--------+-------+                                +-------+------+
            |   +--+     +----+     +-----------+              .
            +~~ |RG| === |NGW | === |   SCHC    |... Internet ..
                +--+     +----+     |Comp / Frag|
                                    +-----------+
```

Figure 4: Architecture

Figure 4 The figure represents the architecture for SCHC (Static Context Header Compression) Compression / Fragmentation where SCHC C/D (Compressor/Decompressor) and SCHC Fragmentation are performed. It is based on [I-D.ietf-lpwan-overview] terminology. SCHC Compression / Fragmentation is located on both sides of the transmission in the Dev and in the Network side. In the Uplink direction, the Device application packets use IPv6 or IPv6/UDP protocols. Before sending these packets, the Dev compresses their headers using SCHC C/D and if the SCHC packet resulting from the compression exceeds the maximum payload size of the underlying LPWAN technology, SCHC fragmentation is performed, see Section 7. The resulting SCHC fragments are sent as one or more L2 frames to an LPWAN Radio Gateway (RG) which forwards the frame(s) to a Network Gateway (NGW).

The NGW sends the data to an SCHC Fragmentation and then to the SCHC C/D for decompression. The SCHC C/D in the Network side can be located in the Network Gateway (NGW) or somewhere else as long as a tunnel is established between the NGW and the SCHC Compression / Fragmentation. Note that, for some LPWAN technologies, it MAY be suitable to locate SCHC fragmentation and reassembly 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. After decompression, the packet can be sent over the Internet to one or several LPWAN Application Servers (App).

The SCHC Compression / Fragmentation process is symmetrical, therefore the same description applies to the reverse direction.

6.1. 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 provides the closest match to 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. How Rules are generated is out of the scope of this document. The rule MAY be changed but it will be specified in another document.

The context contains a list of rules (cf. Figure 5). Each Rule contains itself a list of Fields 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 5: Compression/Decompression Context

The Rule does not describe how to delineate each field in the original packet header. This MUST be known from the compressor/ decompressor. The rule only describes the compression/decompression behavior for each header field. In the rule, the Fields Descriptions are listed in the order in which the fields appear in the packet header.

The Rule also describes the Compression Residue sent regarding the order of the Fields Descriptions in the Rule.

The Context describes the header fields and its values with the following entries:

- o Field ID (FID) is a unique value to define the header field.

- o Field Length (FL) represents the length of the field in bits for fixed values or a type (variable, token length, ...) for Field Description length unknown at the rule creation. The length of a header field is defined in the specific protocol standard.

- o Field Position (FP): indicating if several instances of a field exist in the headers which one is targeted. The default position is 1.

- o A direction indicator (DI) indicating 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 travelling both Up and Dw.

- Target Value (TV) is the value used to make the match with the packet header field. The Target Value can be of any type (integer, strings, etc.). For instance, it can be a single value or a more complex structure (array, list, etc.), such as a JSON or a CBOR structure.

- 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 6.4.

- Compression Decompression Action (CDA) describes the compression and decompression processes to be performed after the MO is applied. The CDA MAY require some parameters to be processed. CDAs are used in both the compression and the decompression functions. The set of CDAs defined in this document can be found in Section 6.5.

6.2. 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 a Dev. Hence, multiple Dev instances MAY use the same Rule ID to define different header compression contexts. To identify the correct Rule ID, the SCHC C/D needs to correlate the Rule ID with the Dev identifier to find the appropriate Rule to be applied.

6.3. Packet processing

The compression/decompression process follows several steps:

- Compression Rule selection: The goal is to identify which Rule(s) will be used to compress the packet's headers. When doing decompression, in the network side the SCHC C/D needs to find the correct Rule based on the L2 address and in this way, it can use the Dev-ID and the Rule-ID. In the Dev side, only the Rule ID is needed to identify the correct Rule since the Dev only

holds Rules that apply to itself. The Rule will be selected by matching the Fields Descriptions to the packet header as described below. When the selection of a Rule is done, this Rule is used to compress the header. The detailed steps for compression Rule selection are the following:

* The first step is to choose the Fields Descriptions by their direction, using the direction indicator (DI). A Field Description that does not correspond to the appropriate DI will be ignored, if all the fields of the packet do not have a Field Description with the correct DI the Rule is discarded and SCHC C/D proceeds to explore the next Rule.

* When the DI has matched, then the next step is to identify the fields according to Field Position (FP). If the Field Position does not correspond, the Rule is not used and the SCHC C/D proceeds to consider the next Rule.

* Once the DI and the FP correspond to the header information, each field's value of the packet is then compared to the corresponding Target Value (TV) stored in the Rule for that specific field using the matching operator (MO).

* If all the fields in the packet's header satisfy all the matching operators (MO) of a Rule (i.e. all MO results are True), the fields of the header are then compressed according to the Compression/Decompression Actions (CDAs) and a compressed header (with possibly a Compressed Residue) SHOULD be obtained. Otherwise, the next Rule is tested.

* If no eligible Rule is found, then the header MUST be sent without compression, depending on the L2 PDU size, this is one of the case that MAY require the use of the SCHC fragmentation process.

o Sending: If an eligible Rule is found, the Rule ID is sent to the other end followed by the Compression Residue (which could be empty) and directly followed by the payload. The product of the Compression Residue is sent in the order expressed in the Rule for all the fields. The way the Rule ID is sent depends on the specific LPWAN layer two technology. For example, it can be either included in a Layer 2 header or sent in the first byte of the L2 payload. (Cf. Figure 6). This process will be specified in the LPWAN technology-specific document and is out of the scope of the present document. On LPWAN technologies that are byte-oriented, the compressed header concatenated with the original packet payload is padded to a multiple of 8 bits, if needed. See Section 8 for details.

o Decompression: When doing decompression, in the network side the SCHC C/D needs to find the correct Rule based on the L2 address and in this way, it can use the Dev-ID and the Rule-ID. In the Dev side, only the Rule ID is needed to identify the correct Rule since the Dev only holds Rules that apply to itself.

   The receiver identifies the sender through its device-id (e.g. MAC address, if exists) and selects the appropriate Rule from the Rule ID. If a source identifier is present in the L2 technology, it is used to select the Rule ID. This Rule describes the compressed header format and associates the values to the header fields. The receiver applies the CDA action to reconstruct the original header fields. The CDA application order can be different from the order given by the Rule. For instance, Compute-* SHOULD be applied at the end, after all the other CDAs.

```
+--- ... --+------- ... -------+------------------+~~~~~~~
|  Rule ID |Compression Residue|  packet payload  |padding
+--- ... --+------- ... -------+------------------+~~~~~~~
                                                   (optional)
<----- compressed header ------>
```

Figure 6: SCHC C/D Packet Format

6.4. Matching operators

Matching Operators (MOs) are functions used by both SCHC C/D endpoints involved in the header compression/decompression. They are not typed and can be indifferently 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 a field value in a packet and the value in the TV are equal.

o ignore: No check is done between a field value in a packet and a TV in the Rule. The result of the matching is always true.

o MSB(x): A match is obtained if the most significant x bits of the field value in the header packet are equal to the TV in the Rule. The x parameter of the MSB Matching Operator indicates how many bits are involved in the comparison.

o match-mapping: With match-mapping, the Target Value is a list of values. Each value of the list is identified by a short ID (or 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.

6.5. Compression Decompression Actions (CDA)

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

Action	Compression	Decompression
not-sent	elided	use value stored in ctxt
value-sent	send	build from received value
mapping-sent	send index	value from index on a table
LSB(y)	send LSB	TV, received value
compute-length	elided	compute length
compute-checksum	elided	compute UDP checksum
Deviid	elided	build IID from L2 Dev addr
Appiid	elided	build IID from L2 App addr

y=size of the transmitted bits

Figure 7: Compression and Decompression Functions

Figure 7 summarizes the basic functions that can be used to compress and decompress a field. The first column lists the actions name. The second and third columns outline the reciprocal compression/ decompression behavior for each action.

Compression is done in order that Fields Descriptions appear in the Rule. The result of each Compression/Decompression Action is appended to the working Compression Residue in that same order. The receiver knows the size of each compressed field which can be given by the rule or MAY be sent with the compressed header.

If the field is identified as being variable in the Field Description, then the size of the Compression Residue value in bytes MUST be sent first using the following coding:

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

- o For values between 15 and 255, the first 4 bits sent are set to 1 and the size is sent using 8 bits integer.

- For higher values of size, the first 12 bits are set to 1 and the next two bytes contain the size value as a 16 bits integer.

- If a field does not exist in the packet but in the Rule and its FL is variable, the size zero MUST be used.

6.5.1. not-sent CDA

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

The compressor does not send any value in the Compressed 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.

6.5.2. value-sent CDA

The value-sent action is generally used when the field value is not known by both Compressor and Decompressor. The value is sent in the compressed message header. Both Compressor and Decompressor MUST know the size of the field, either implicitly (the size is known by both sides) or explicitly in the compression residue by indicating the length, as defined in Section 6.5. This function is generally used with the "ignore" MO.

6.5.3. mapping-sent CDA

The mapping-sent is used to send a smaller index (the index into the Target Value list of values) instead of the original value. This function 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 and the mapping-sent CDA appends the corresponding index to the Compression Residue to be sent. 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.

6.5.4. LSB(y) CDA

The LSB(y) action is used together with the "MSB(x)" MO to avoid sending the higher part of the packet field if that part is already known by the receiving end. A length can be specified in the rule to indicate how many bits have to be sent. If the length is not specified, the number of bits sent is the original header field length minus the length specified in the MSB(x) MO.

The compressor sends the Least Significant Bits (e.g. LSB of the length field). The decompressor combines the value received with the Target Value depending on the field type.

If this action needs to be done on a variable length field, the size of the Compressed Residue in bytes MUST be sent as described in Section 6.5.

6.5.5. DEViid, APPiid CDA

These functions 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 current LPWAN technologies frames contain a single address, which is the Dev's address.

The IID value MAY be computed from the Device ID present in the Layer 2 header, or from some other stable identifier. The computation is specific for each LPWAN technology and MAY depend on the Device ID size.

In the Downlink direction, these Deviid CDA is used to determine the L2 addresses used by the LPWAN.

6.5.6. Compute-*

Some fields are elided during compression and reconstructed during decompression. This is the case for length and Checksum, so:

- compute-length: computes the length assigned to this field. This CDA MAY be used to compute IPv6 length or UDP length.

- compute-checksum: computes a checksum from the information already received by the SCHC C/D. This field MAY be used to compute UDP checksum.

7. Fragmentation

7.1. Overview

In LPWAN technologies, the L2 data unit size typically varies from tens to hundreds of bytes. The SCHC fragmentation MAY be used either because after applying SCHC C/D or when SCHC C/D is not possible the entire SCHC packet still exceeds the L2 data unit.

The SCHC fragmentation functionality defined in this document has been designed under the assumption that data unit out-of- sequence delivery will not happen between the entity performing fragmentation and the entity performing reassembly. This assumption allows reducing the complexity and overhead of the SCHC fragmentation mechanism.

To adapt the SCHC fragmentation to the capabilities of LPWAN technologies is required to enable optional SCHC fragment retransmission and to allow a stepper delivery for the reliability of SCHC fragments. This document does not make any decision with regard to which SCHC fragment delivery reliability mode will be used over a specific LPWAN technology. These details will be defined in other technology-specific documents.

7.2. Fragmentation Tools

This subsection describes the different tools that are used to enable the SCHC fragmentation functionality defined in this document, such as fields in the SCHC fragmentation header frames (see the related formats in Section 7.4), and the different parameters supported in the reliability modes such as timers and parameters.

o Rule ID. The Rule ID is present in the SCHC fragment header and in the ACK header format. The Rule ID in a SCHC fragment header is used to identify that a SCHC fragment is being carried, which SCHC fragmentation reliability mode is used and which window size is used. The Rule ID in the SCHC fragmentation header also allows interleaving non-fragmented packets and SCHC fragments that carry other SCHC packets. The Rule ID in an ACK identifies the message as an ACK.

o Fragment Compressed Number (FCN). The FCN is included in all SCHC fragments. This field can be understood as a truncated, efficient representation of a larger-sized fragment number, and does not carry an absolute SCHC fragment number. There are two FCN reserved values that are used for controlling the SCHC fragmentation process, as described next:

   * The FCN value with all the bits equal to 1 (All-1) denotes the last SCHC fragment of a packet. The last window of a packet is called an All-1 window.

   * The FCN value with all the bits equal to 0 (All-0) denotes the last SCHC fragment of a window that is not the last one of the packet. Such a window is called an All-0 window.

   The rest of the FCN values are assigned in a sequentially decreasing order, which has the purpose to avoid possible ambiguity for the receiver that might arise under certain conditions. In the SCHC fragments, this field is an unsigned integer, with a size of N bits. In the No-ACK mode, it is set to 1 bit (N=1), All-0 is used in all SCHC fragments and All-1 for the last one. For the other reliability modes, it is recommended to use a number of bits (N) equal to or greater than 3. Nevertheless, the appropriate value of N MUST be defined in the corresponding technology-specific profile documents. For windows that are not the last one from a SCHC fragmented packet, the FCN for the last SCHC fragment in such windows is an All-0. This indicates that the window is finished and communication proceeds according to the reliability mode in use. The FCN for the last SCHC fragment in the last window is an All-1, indicating the last SCHC fragment of the SCHC packet. It is also important to note that, in the No-ACK mode or when N=1, the last SCHC fragment of the packet will carry a FCN equal to 1, while all previous SCHC fragments will carry a FCN of 0. For further details see Section 7.5. The highest FCN in the window, denoted MAX_WIND_FCN, MUST be a value equal to or smaller than 2^N-2. (Example for N=5, MAX_WIND_FCN MAY be set to 23, then subsequent FCNs are set sequentially and in decreasing order, and the FCN will wrap from 0 back to 23).

o Datagram Tag (DTag). The DTag field, if present, is set to the same value for all SCHC fragments carrying the same SCHC packet, and to different values for different datagrams. Using this field, the sender can interleave fragments from different SCHC packets, while the receiver can still tell them apart. In the SCHC fragment formats, the size of the DTag field is T bits, which MAY be set to a value greater than or equal to 0 bits. For each new SCHC packet processed by the sender, DTag MUST be sequentially increased, from 0 to 2^T - 1 wrapping back from 2^T - 1 to 0. In the ACK format, DTag carries the same value as the DTag field in the SCHC fragments for which this ACK is intended.

o W (window): W is a 1-bit field. This field carries the same value for all SCHC fragments of a window, and it is complemented for the next window. The initial value for this field is 0. In the ACK

format, this field also has a size of 1 bit. In all ACKs, the W bit carries the same value as the W bit carried by the SCHC fragments whose reception is being positively or negatively acknowledged by the ACK.

- Message Integrity Check (MIC). This field, which has a size of M bits, is computed by the sender over the complete SCHC packet before SCHC fragmentation. The MIC allows the receiver to check errors in the reassembled packet, while it also enables compressing the UDP checksum by use of SCHC compression. The CRC32 as 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 MIC. Nevertheless, other algorithms MAY be required and are defined in the technology-specific documents.

- C (MIC checked): C is a 1-bit field. This field is used in the ACK packets to report the outcome of the MIC check, i.e. whether the reassembled packet was correctly received or not. A value of 1 represents a positive MIC check at the receiver side (i.e. the MIC computed by the receiver matches the received MIC).

- Retransmission Timer. A SCHC fragment sender uses it after the transmission of a window to detect a transmission error of the ACK corresponding to this window. Depending on the reliability mode, it will lead to a request an ACK retransmission (in ACK-Always mode) or it will trigger the transmission of the next window (in ACK-on-Error mode). The duration of this timer is not defined in this document and MUST be defined in the corresponding technology documents.

- Inactivity Timer. A SCHC fragment receiver uses it to take action when there is a problem in the transmission of SCHC fragments. Such a problem could be detected by the receiver not getting a single SCHC fragment during a given period of time or not getting a given number of packets in a given period of time. When this happens, an Abort message will be sent (see related text later in this section). Initially, and each time a SCHC fragment is received, the timer is reinitialized. The duration of this timer is not defined in this document and MUST be defined in the specific technology document.

- Attempts. This counter counts the requests for a missing ACK. When it reaches the value MAX_ACK_REQUESTS, the sender assume there are recurrent SCHC fragment transmission errors and determines that an Abort is needed. The default value offered MAX_ACK_REQUESTS is not stated in this document, and it is expected to be defined in the specific technology document. The

Attempts counter is defined per window. It is initialized each time a new window is used.

- o Bitmap. The Bitmap is a sequence of bits carried in an ACK. Each bit in the Bitmap corresponds to a SCHC fragment of the current window, and provides feedback on whether the SCHC fragment has been received or not. The right-most position on the Bitmap reports if the All-0 or All-1 fragment has been received or not. Feedback on the SCHC fragment with the highest FCN value is provided by the bit in the left-most position of the Bitmap. In the Bitmap, a bit set to 1 indicates that the SCHC fragment of FCN corresponding to that bit position has been correctly sent and received. The text above describes the internal representation of the Bitmap. When inserted in the ACK for transmission from the receiver to the sender, the Bitmap MAY be truncated for energy/ bandwidth optimisation, see more details in Section 7.4.3.1.

- o Abort. On expiration of the Inactivity timer, or when Attempts reached MAX_ACK_REQUESTS or upon an occurrence of some other error, the sender or the receiver MUST use the Abort. When the receiver needs to abort the on-going SCHC fragmented packet transmission, it sends the Receiver-Abort format. When the sender needs to abort the transmission, it sends the Sender-Abort format. None of the Abort are acknowledged.

- o Padding (P). If it is needed, the number of bits used for padding is not defined and depends on the size of the Rule ID, DTag and FCN fields, and on the L2 payload size (see Section 8). Some ACKs are byte-aligned and do not need padding (see Section 7.4.3.1).

7.3. Reliability modes

This specification defines three reliability modes: No-ACK, ACK-Always and ACK-on-Error. ACK-Always and ACK-on-Error operate on windows of SCHC fragments. A window of SCHC fragments is a subset of the full set of SCHC fragments needed to carry a packet or an SCHC packet.

- o No-ACK. No-ACK is the simplest SCHC fragment reliability mode. The receiver does not generate overhead in the form of acknowledgments (ACKs). However, this mode does not enhance reliability beyond that offered by the underlying LPWAN technology. In the No-ACK mode, the receiver MUST NOT issue ACKs. See further details in Section 7.5.1.

- o ACK-Always. The ACK-Always mode provides flow control using a window scheme. This mode is also able to handle long bursts of lost SCHC fragments since detection of such events can be done

before the end of the SCHC packet transmission as long as the window size is short enough. However, such benefit comes at the expense of ACK use. In ACK-Always the receiver sends an ACK after a window of SCHC fragments has been received, where a window of SCHC fragments is a subset of the whole number of SCHC fragments needed to carry a complete SCHC packet. The ACK is used to inform the sender if a SCHC fragment in the actual window has been lost or well received. Upon an ACK reception, the sender retransmits the lost SCHC fragments. When an ACK is lost and the sender has not received it before the expiration of the Inactivity Timer, the sender uses an ACK request by sending the All-1 empty SCHC fragment. The maximum number of ACK requests is MAX_ACK_REQUESTS. If the MAX_ACK_REQUEST is reached the transmission needs to be Aborted. See further details in Section 7.5.2.

o ACK-on-Error. The ACK-on-Error mode is suitable for links offering relatively low L2 data unit loss probability. In this mode, the SCHC fragment receiver reduces the number of ACKs transmitted, which MAY be especially beneficial in asymmetric scenarios. Because the SCHC fragments use the uplink of the underlying LPWAN technology, which has higher capacity than downlink. The receiver transmits an ACK only after the complete window transmission and if at least one SCHC fragment of this window has been lost. An exception to this behavior is in the last window, where the receiver MUST transmit an ACK, including the C bit set based on the MIC checked result, even if all the SCHC fragments of the last window have been correctly received. The ACK gives the state of all the SCHC fragments (received or lost). Upon an ACK reception, the sender retransmits the lost SCHC fragments. If an ACK is not transmitted back by the receiver at the end of a window, the sender assumes that all SCHC fragments have been correctly received. When the ACK is lost, the sender assumes that all SCHC fragments covered by the lost ACK have been successfully delivered, so the sender continues transmitting the next window of SCHC fragments. If the next SCHC fragments received belong to the next window, the receiver will abort the on-going fragmented packet transmission. See further details in {{ACK-on-Error- subsection}}.

The same reliability mode MUST be used for all SCHC fragments of an SCHC packet. The decision on which reliability mode will be used and whether the same reliability mode applies to all SCHC packets is an implementation problem and is out of the scope of this document.

Note that the reliability mode choice is not necessarily tied to a particular characteristic of the underlying L2 LPWAN technology, e.g. the No-ACK mode MAY be used on top of an L2 LPWAN technology with symmetric characteristics for uplink and downlink. This document

does not make any decision as to which SCHC fragment reliability mode(s) are supported by a specific LPWAN technology.

Examples of the different reliability modes described are provided in Appendix B.

7.4. Fragmentation Formats

This section defines the SCHC fragment format, the All-0 and All-1 formats, the ACK format and the Abort formats.

7.4.1. Fragment format

A SCHC fragment comprises a SCHC fragment header, a SCHC fragment payload and padding bits (if needed). A SCHC fragment conforms to the general format shown in Figure 8. The SCHC fragment payload carries a subset of SCHC packet. A SCHC fragment is the payload of the L2 protocol data unit (PDU). Padding MAY be added in SCHC fragments and in ACKs if necessary, therefore a padding field is optional (this is explicitly indicated in Figure 8 for the sake of illustration clarity.

```
      +-----------------+-----------------------+~~~~~~~~~~~~~~~
      | Fragment Header |   Fragment payload    | padding (opt.)
      +-----------------+-----------------------+~~~~~~~~~~~~~~~
```

Figure 8: Fragment general format. Presence of a padding field is optional

In ACK-Always or ACK-on-Error, SCHC fragments except the last one SHALL conform the detailed format defined in {{Fig- NotLastWin}}. The total size of the fragment header is R bits. Where is R is not a multiple of 8 bits.

```
 <------------ R ----------->
            <--T--> 1 <--N-->
 +-- ... --+- ... -+-+- ... -+--------...-------+
 | Rule ID | DTag  |W|  FCN  | Fragment payload |
 +-- ... --+- ... -+-+- ... -+--------...-------+
```

Figure 9: Fragment Detailed Format for Fragments except the Last One, Window mode

In the No-ACK mode, SCHC fragments except the last one SHALL conform to the detailed format defined in Figure 10. The total size of the fragment header is R bits.

```
   <------------ R ----------->
                <--T--> <--N-->
    +-- ... --+- ...  -+- ... -+--------...-------+
    | Rule ID |  DTag  |  FCN  | Fragment payload |
    +-- ... --+- ...  -+- ... -+--------...-------+
```

Figure 10: Fragment Detailed Format for Fragments except the Last One, No-ACK mode

In all these cases, R may not be a multiple of 8 bits.

7.4.2. All-1 and All-0 formats

The All-0 format is used for sending the last SCHC fragment of a window that is not the last window of the packet.

```
        <------------ R ----------->
                   <- T -> 1 <- N ->
        +-- ... --+- ... -+-+- ... -+--- ... ---+
        | Rule ID | DTag  |W|  0..0 |  payload  |
        +-- ... --+- ... -+-+- ... -+--- ... ---+
```

Figure 11: All-0 fragment detailed format

The All-0 empty fragment format is used by a sender to request the retransmission of an ACK by the receiver. It is only used in ACK-Always mode.

```
    <------------ R ----------->
               <- T -> 1 <- N ->
    +-- ... --+- ... -+-+- ... -+
    | Rule ID | DTag  |W|  0..0 | (no payload)
    +-- ... --+- ... -+-+- ... -+
```

Figure 12: All-0 empty fragment detailed format

In the No-ACK mode, the last SCHC fragment of an IPv6 datagram SHALL contain a SCHC fragment header that conforms to the detaield format shown in Figure 13. The total size of this SCHC fragment header is R+M bits.

```
   <------------ R ----------->
                 <- T -> <N=1> <---- M ---->
   +---- ... ---+- ... -+-----+---- ... ----+---...---+
   |   Rule ID  | DTag  |  1  |     MIC     | payload |
   +---- ... ---+- ... -+-----+---- ... ----+---...---+
```

   Figure 13: All-1 Fragment Detailed Format for the Last Fragment, No-
                                 ACK mode

   In any of the Window modes, the last fragment of an IPv6 datagram
   SHALL contain a SCHC fragment header that conforms to the detailed
   format shown in Figure 14. The total size of the SCHC fragment
   header in this format is R+M bits.

```
   <------------ R ----------->
              <- T -> 1 <- N -> <---- M ---->
   +-- ... --+- ... -+-+- ... -+---- ... ----+---...---+
   | Rule ID | DTag  |W| 11..1 |     MIC     | payload |
   +-- ... --+- ... -+-+- ... -+---- ... ----+---...---+
                          (FCN)
```

   Figure 14: All-1 Fragment Detailed Format for the Last Fragment, ACK-
                          Always or ACK-on-Error

   In either ACK-Always or ACK-on-Error, in order to request a
   retransmission of the ACK for the All-1 window, the fragment sender
   uses the format shown in Figure 15. The total size of the SCHC
   fragment header in this format is R+M bits.

```
   <------------ R ----------->
              <- T -> 1 <- N -> <---- M ---->
   +-- ... --+- ... -+-+- ... -+---- ... ----+
   | Rule ID | DTag  |W|  1..1 |     MIC     | (no payload)
   +-- ... --+- ... -+-+- ... -+---- ... ----+
```

       Figure 15: All-1 for Retries format, also called All-1 empty

   The values for R, N, T and M are not specified in this document, and
   SHOULD be determined in other documents (e.g. technology-specific
   profile documents).

7.4.3. ACK format

   The format of an ACK that acknowledges a window that is not the last
   one (denoted as All-0 window) is shown in Figure 16.

```
     <--------- R -------->
                 <- T -> 1
     +---- ... --+-... -+-+---- ... -----+
     |  Rule ID  | DTag |W|encoded Bitmap| (no payload)
     +---- ... --+-... -+-+---- ... -----+
```

Figure 16: ACK format for All-0 windows

To acknowledge the last window of a packet (denoted as All-1 window), a C bit (i.e. MIC checked) following the W bit is set to 1 to indicate that the MIC check computed by the receiver matches the MIC present in the All-1 fragment. If the MIC check fails, the C bit is set to 0 and the Bitmap for the All-1 window follows.

```
   <---------- R --------->
               <- T -> 1 1
   +---- ... --+-... -+-+-+
   |  Rule ID  | DTag |W|1| (MIC correct)
   +---- ... --+-... -+-+-+

   +---- ... --+-... -+-+-+----- ... -----+
   |  Rule ID  | DTag |W|0|encoded Bitmap |(MIC Incorrect)
   +---- ... --+-... -+-+-+----- ... -----+
                        C
```

Figure 17: Format of an ACK for All-1 windows

7.4.3.1. **Bitmap Encoding**

The Bitmap is transmitted by a receiver as part of the ACK format. An ACK message MAY include padding at the end to align its number of transmitted bits to a multiple of 8 bits.

Note that the ACK sent in response to an All-1 fragment includes the C bit. Therefore, the window size and thus the encoded Bitmap size need to be determined taking into account the available space in the layer two frame payload, where there will be 1 bit less for an ACK sent in response to an All-1 fragment than in other ACKs. Note that the maximum number of SCHC fragments of the last window is one unit smaller than that of the previous windows.

When the receiver transmits an encoded Bitmap with a SCHC fragment that has not been sent during the transmission, the sender will Abort the transmission.

```
                    <----         Bitmap bits      ---->
| Rule ID | DTag |W|1|0|1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|
|--- byte boundary ----| 1 byte  next  |  1 byte next  |
```

Figure 18: A non-encoded Bitmap

In order to reduce the resulting frame size, the encoded Bitmap is shortened by applying the following algorithm: all the right-most contiguous bytes in the encoded Bitmap that have all their bits set to 1 MUST NOT be transmitted. Because the SCHC fragment sender knows the actual Bitmap size, it can reconstruct the original Bitmap with the trailing 1 bit optimized away. In the example shown in Figure 19, the last 2 bytes of the Bitmap shown in Figure 18 comprise bits that are all set to 1, therefore they are not sent.

```
     <-------   R  ------->
                 <- T -> 1
     +---- ... --+-... -+-+-+-+
     |  Rule ID  | DTag |W|1|0|
     +---- ... --+-... -+-+-+-+
     |---- byte boundary -----|
```

Figure 19: Optimized Bitmap format

Figure 20 shows an example of an ACK with FCN ranging from 6 down to 0, where the Bitmap indicates that the second and the fifth SCHC fragments have not been correctly received.

```
<------   R  ------>6 5 4 3 2 1   0 (*)
          <- T -> 1
+---------+------+-+-+-+-+-+-+-+------+
| Rule ID | DTag |W|1|0|1|1|0|1|all-0| Bitmap(before tx)
+---------+------+-+-+-+-+-+-+-+------+
|<-- byte boundary ->|<---- 1 byte---->|
    (*)=(FCN values)

+---------+------+-+-+-+-+-+-+-+------+~~
| Rule ID | DTag |W|1|0|1|1|0|1|all-0|Padding(opt.) encoded Bitmap
+---------+------+-+-+-+-+-+-+-+------+~~
|<-- byte boundary ->|<---- 1 byte---->|
```

Figure 20: Example of a Bitmap before transmission, and the transmitted one, in any window except the last one

Figure 21 shows an example of an ACK with FCN ranging from 6 down to 0, where the Bitmap indicates that the MIC check has failed but there are no missing SCHC fragments.

```
 <-------   R  ------->  6 5 4 3 2 1 7 (*)
            <- T -> 1 1
 |  Rule ID  | DTag |W|0|1|1|1|1|1|1|1|padding|  Bitmap (before tx)
 |---- byte boundary -----|  1 byte next |
                       C
 +---- ... --+-... -+-+-+-+
 |  Rule ID  | DTag |W|0|1| encoded Bitmap
 +---- ... --+-... -+-+-+-+
 |<--- byte boundary ---->|
   (*) = (FCN values indicating the order)
```

Figure 21: Example of the Bitmap in ACK-Always or ACK-on-Error for the last window, for N=3)

7.4.4. Abort formats

Abort are coded as exceptions to the previous coding, a specific format is defined for each direction. When a SCHC fragment sender needs to abort the transmission, it sends the Sender-Abort format Figure 22, that is an All-1 fragment with no MIC or payload. In regular cases All-1 fragment contains at least a MIC value. This absence of the MIC value indicates an Abort.

When a SCHC fragment receiver needs to abort the on-going SCHC fragmented packet transmission, it transmits the Receiver- Abort format Figure 23, creating an exception in the encoded Bitmap coding. Encoded Bitmap avoid sending the rigth most bits of the Bitmap set to 1. Abort is coded as an ACK message with a Bitmap set to 1 until the byte boundary, followed by an extra 0xFF byte. Such message never occurs in a regular acknowledgement and is view as an abort.

None of these messages are not acknowledged nor retransmitted.

The sender uses the Sender-Abort when the MAX_ACK_REQUEST is reached. The receiver uses the Receiver-Abort when the Inactivity timer expires, or in the ACK-on-Error mode, ACK is lost and the sender transmits SCHC fragments of a new window. Some other cases for Abort are explained in the Section 7.5 or Appendix C.

```
   <------------- R ----------->< --- 1 byte --->
   +--- ... ---+- ... -+-+-...-+-+-+-+-+-+-+-+-+
   |  Rule ID  | DTag  |W| FCN |       FF      | (no MIC & no payload)
   +--- ... ---+- ... -+-+-...-+-+-+-+-+-+-+-+-+
```

    Figure 22: Sender-Abort format. All FCN fields in this format are
                                 set to 1

```
    <----- byte boundary ------><--- 1 byte --->

    +---- ... --+-... -+-+-+-+-+-+-+-+-+-+-+-+-+
    |  Rule ID  | DTag |W| 1..1|       FF      |
    +---- ... --+-... -+-+-+-+-+-+-+-+-+-+-+-+-+
```

                    Figure 23: Receiver-Abort format

7.5. Baseline mechanism

If after applying SCHC header compression (or when SCHC header compression is not possible) the SCHC packet does not fit within the payload of a single L2 data unit, the SCHC packet SHALL be broken into SCHC fragments and the fragments SHALL be sent to the fragment receiver. The fragment receiver needs to identify all the SCHC fragments that belong to a given SCHC packet. To this end, the receiver SHALL use:

o The sender's L2 source address (if present),

o The destination's L2 address (if present),

o Rule ID,

o DTag (if present).

Then, the fragment receiver MAY determine the SCHC fragment reliability mode that is used for this SCHC fragment based on the Rule ID in that fragment.

After a SCHC fragment reception, the receiver starts constructing the SCHC packet. It uses the FCN and the arrival order of each SCHC fragment to determine the location of the individual fragments within the SCHC packet. For example, the receiver MAY place the fragment payload within a payload datagram reassembly buffer at the location determined from the FCN, the arrival order of the SCHC fragments, and the fragment payload sizes. In Window mode, the fragment receiver also uses the W bit in the received SCHC fragments. Note that the

size of the original, unfragmented packet cannot be determined from fragmentation headers.

Fragmentation functionality uses the FCN value to transmit the SCHC fragments. It has a length of N bits where the All-1 and All-0 FCN values are used to control the fragmentation transmission. The rest of the FCN numbers MUST be assigned sequentially in a decreasing order, the first FCN of a window is RECOMMENDED to be MAX_WIND_FCN, i.e. the highest possible FCN value depending on the FCN number of bits.

In all modes, the last SCHC fragment of a packet MUST contain a MIC which is used to check if there are errors or missing SCHC fragments and MUST use the corresponding All-1 fragment format. Note that a SCHC fragment with an All-0 format is considered the last SCHC fragment of the current window.

If the receiver receives the last fragment of a datagram (All-1), it checks for the integrity of the reassembled datagram, based on the MIC received. In No-ACK, if the integrity check indicates that the reassembled datagram does not match the original datagram (prior to fragmentation), the reassembled datagram MUST be discarded. In Window mode, a MIC check is also performed by the fragment receiver after reception of each subsequent SCHC fragment retransmitted after the first MIC check.

There are three reliability modes: No-ACK, ACK-Always and ACK-on-Error. In ACK-Always and ACK-on-Error, a jumping window protocol uses two windows alternatively, identified as 0 and 1. A SCHC fragment with all FCN bits set to 0 (i.e. an All-0 fragment) indicates that the window is over (i.e. the SCHC fragment is the last one of the window) and allows to switch from one window to the next one. The All-1 FCN in a SCHC fragment indicates that it is the last fragment of the packet being transmitted and therefore there will not be another window for this packet.

7.5.1. No-ACK

In the No-ACK mode, there is no feedback communication from the fragment receiver. The sender will send all the SCHC fragments of a packet without any possibility of knowing if errors or losses have occurred. As, in this mode, there is no need to identify specific SCHC fragments, a one-bit FCN MAY be used. Consequently, the FCN All-0 value is used in all SCHC fragments except the last one, which carries an All-1 FCN and the MIC. The receiver will wait for SCHC fragments and will set the Inactivity timer. The receiver will use the MIC contained in the last SCHC fragment to check for errors. When the Inactivity Timer expires or if the MIC check indicates that

the reassembled packet does not match the original one, the receiver will release all resources allocated to reassembling this packet. The initial value of the Inactivity Timer will be determined based on the characteristics of the underlying LPWAN technology and will be defined in other documents (e.g. technology-specific profile documents).

7.5.2. ACK-Always

In ACK-Always, the sender transmits SCHC fragments by using the two-jumping-windows procedure. A delay between each SCHC fragment can be added to respect local regulations or other constraints imposed by the applications. Each time a SCHC fragment is sent, the FCN is decreased by one. When the FCN reaches value 0 and there are more SCHC fragments to be sent after, the sender transmits the last SCHC fragment of this window using the All-0 fragment format, it starts the Retransmission Timer and waits for an ACK. On the other hand, if the FCN has reached 0 and the SCHC fragment to be transmitted is the last SCHC fragment of the SCHC packet, the sender uses the All-1 fragment format, which includes a MIC. The sender sets the Retransmission Timer and waits for the ACK to know if transmission errors have occured.

The Retransmission Timer is dimensioned based on the LPWAN technology in use. When the Retransmission Timer expires, the sender sends an All-0 empty (resp. All-1 empty) fragment to request again the ACK for the window that ended with the All-0 (resp. All-1) fragment just sent. The window number is not changed.

After receiving an All-0 or All-1 fragment, the receiver sends an ACK with an encoded Bitmap reporting whether any SCHC fragments have been lost or not. When the sender receives an ACK, it checks the W bit carried by the ACK. Any ACK carrying an unexpected W bit value is discarded. If the W bit value of the received ACK is correct, the sender analyzes the rest of the ACK message, such as the encoded Bitmap and the MIC. If all the SCHC fragments sent for this window have been well received, and if at least one more SCHC fragment needs to be sent, the sender advances its sending window to the next window value and sends the next SCHC fragments. If no more SCHC fragments have to be sent, then the SCHC fragmented packet transmission is finished.

However, if one or more SCHC fragments have not been received as per the ACK (i.e. the corresponding bits are not set in the encoded Bitmap) then the sender resends the missing SCHC fragments. When all missing SCHC fragments have been retransmitted, the sender starts the Retransmission Timer, even if an All-0 or an All-1 has not been sent as part of this retransmission and waits for an ACK. Upon receipt of

the ACK, if one or more SCHC fragments have not yet been received, the counter Attempts is increased and the sender resends the missing SCHC fragments again. When Attempts reaches MAX_ACK_REQUESTS, the sender aborts the on-going SCHC fragmented packet transmission by sending an Abort message and releases any resources for transmission of the packet. The sender also aborts an on-going SCHC fragmented packet transmission when a failed MIC check is reported by the receiver or when a SCHC fragment that has not been sent is reported in the encoded Bitmap.

On the other hand, at the beginning, the receiver side expects to receive window 0. Any SCHC fragment received but not belonging to the current window is discarded. All SCHC fragments belonging to the correct window are accepted, and the actual SCHC fragment number managed by the receiver is computed based on the FCN value. The receiver prepares the encoded Bitmap to report the correctly received and the missing SCHC fragments for the current window. After each SCHC fragment is received the receiver initializes the Inactivity timer, if the Inactivity Timer expires the transmission is aborted.

When an All-0 fragment is received, it indicates that all the SCHC fragments have been sent in the current window. Since the sender is not obliged to always send a full window, some SCHC fragment number not set in the receiver memory SHOULD not correspond to losses. The receiver sends the corresponding ACK, the Inactivity Timer is set and the transmission of the next window by the sender can start.

If an All-0 fragment has been received and all SCHC fragments of the current window have also been received, the receiver then expects a new Window and waits for the next SCHC fragment. Upon receipt of a SCHC fragment, if the window value has not changed, the received SCHC fragments are part of a retransmission. A receiver that has already received a SCHC fragment SHOULD discard it, otherwise, it updates the encoded Bitmap. If all the bits of the encoded Bitmap are set to one, the receiver MUST send an ACK without waiting for an All-0 fragment and the Inactivity Timer is initialized.

On the other hand, if the window value of the next received SCHC fragment is set to the next expected window value, this means that the sender has received a correct encoded Bitmap reporting that all SCHC fragments have been received. The receiver then updates the value of the next expected window.

When an All-1 fragment is received, it indicates that the last SCHC fragment of the packet has been sent. Since the last window is not always full, the MIC will be used to detect if all SCHC fragments of the packet have been received. A correct MIC indicates the end of the transmission but the receiver MUST stay alive for an Inactivity

Timer period to answer to any empty All-1 fragments the sender MAY send if ACKs sent by the receiver are lost. If the MIC is incorrect, some SCHC fragments have been lost. The receiver sends the ACK regardless of successful SCHC fragmented packet reception or not, the Inactitivity Timer is set. In case of an incorrect MIC, the receiver waits for SCHC fragments belonging to the same window. After MAX_ACK_REQUESTS, the receiver will abort the on-going SCHC fragmented packet transmission by transmitting a the Receiver-Abort format. The receiver also aborts upon Inactivity Timer expiration.

7.5.3. ACK-on-Error

The senders behavior for ACK-on-Error and ACK-Always are similar. The main difference is that in ACK-on-Error the ACK with the encoded Bitmap is not sent at the end of each window but only when at least one SCHC fragment of the current window has been lost. Excepts for the last window where an ACK MUST be sent to finish the transmission.

In ACK-on-Error, the Retransmission Timer expiration will be considered as a positive acknowledgment. This timer is set after sending an All-0 or an All-1 fragment. When the All-1 fragment has been sent, then the on-going SCHC fragmentation process is finished and the sender waits for the last ACK. If the Retransmission Timer expires while waiting for the ACK for the last window, an All-1 empty MUST be sent to request the last ACK by the sender to complete the SCHC fragmented packet transmission. When it expires the sender continue sending SCHC fragments of the next window.

If the sender receives an ACK, it checks the window value. ACKs with an unexpected window number are discarded. If the window number on the received encoded Bitmap is correct, the sender verifies if the receiver has received all SCHC fragments of the current window. When at least one SCHC fragment has been lost, the counter Attempts is increased by one and the sender resends the missing SCHC fragments again. When Attempts reaches MAX_ACK_REQUESTS, the sender sends an Abort message and releases all resources for the on-going SCHC fragmented packet transmission. When the retransmission of the missing SCHC fragments is finished, the sender starts listening for an ACK (even if an All-0 or an All-1 has not been sent during the retransmission) and initializes the Retransmission Timer. After sending an All-1 fragment, the sender listens for an ACK, initializes Attempts, and starts the Retransmission Timer. If the Retransmission Timer expires, Attempts is increased by one and an empty All-1 fragment is sent to request the ACK for the last window. If Attempts reaches MAX_ACK_REQUESTS, the sender aborts the on-going SCHC fragmented packet transmission by transmitting the Sender-Abort fragment.

Unlike the sender, the receiver for ACK-on-Error has a larger amount of differences compared with ACK-Always. First, an ACK is not sent unless there is a lost SCHC fragment or an unexpected behavior. With the exception of the last window, where an ACK is always sent regardless of SCHC fragment losses or not. The receiver starts by expecting SCHC fragments from window 0 and maintains the information regarding which SCHC fragments it receives. After receiving an SCHC fragment, the Inactivity Timer is set. If no further SCHC fragment are received and the Inactivity Timer expires, the SCHC fragment receiver aborts the on-going SCHC fragmented packet transmission by transmitting the Receiver-Abort data unit.

Any SCHC fragment not belonging to the current window is discarded. The actual SCHC fragment number is computed based on the FCN value. When an All-0 fragment is received and all SCHC fragments have been received, the receiver updates the expected window value and expects a new window and waits for the next SCHC fragment.
If the window value of the next SCHC fragment has not changed, the received SCHC fragment is a retransmission. A receiver that has already received an SCHC fragment discard it. If all SCHC fragments of a window (that is not the last one) have been received, the receiver does not send an ACK. While the receiver waits for the next window and if the window value is set to the next value, and if an All-1 fragment with the next value window arrived the receiver knows that the last SCHC fragment of the packet has been sent. Since the last window is not always full, the MIC will be used to detect if all SCHC fragments of the window have been received. A correct MIC check indicates the end of the SCHC fragmented packet transmission. An ACK is sent by the SCHC fragment receiver. In case of an incorrect MIC, the receiver waits for SCHC fragments belonging to the same window or the expiration of the Inactivity Timer. The latter will lead the receiver to abort the on-going SCHC fragmented packet transmission.

If after receiving an All-0 fragment the receiver missed some SCHC fragments, the receiver uses an ACK with the encoded Bitmap to ask the retransmission of the missing fragments and expect to receive SCHC fragments with the actual window. While waiting the retransmission an All-0 empty fragment is received, the receiver sends again the ACK with the encoded Bitmap, if the SCHC fragments received belongs to another window or an All-1 fragment is received, the transmission is aborted by sending a Receiver-Abort fragment. Once it has received all the missing fragments it waits for the next window fragments.

7.6. Supporting multiple window sizes

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 carried by a large number of SCHC fragments. However, when the number of SCHC fragments required to carry a packet is low, a smaller window size, and thus a shorter Bitmap, MAY be sufficient to provide feedback on all SCHC fragments. If multiple window sizes are supported, the Rule ID MAY be used to signal the window size in use for a specific packet transmission.

Note that the same window size MUST be used for the transmission of all SCHC fragments that belong to the same SCHC packet.

7.7. Downlink SCHC fragment transmission

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 SCHC fragmented datagram, 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 frame that requires an L2 ACK) or it MAY be triggered from an upper layer.

For downlink transmission of a SCHC fragmented packet in ACK-Always mode, the SCHC fragment receiver MAY support timer-based ACKretransmission. In this mechanism, the SCHC fragment receiver initializes and starts a timer (the Inactivity Timer is used) after the transmission of an ACK, except when the 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 ACK.

If, after transmission of an 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 ACK is stopped. However, if the Inactivity timer expires, the 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 ACK, denoted MAX_ACK_RETRIES, are not defined in this document, and need to be

defined in other documents (e.g. technology-specific profiles). 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.

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

8. Padding management

Default padding is defined for L2 frame with a variable length of bytes. Padding is done twice, after compression and in the all-1 fragmentation.

In compression, the rule and the compression residues are not aligned on a byte, but payload following the residue is always a multiple of 8 bits. In that case, padding bits can be added after the payload to reach the first byte boundary. Since the rule and the residue give the length of the SCHC header and payload is always a multiple of 8 bits, the receiver can without ambiguity remove the padding bits which never excide 7 bits.

SCHC fragmentation works on a byte aligned (i.e. padded SCHC packet). Fragmentation header may not be aligned on byte boundary, but each fragment except the last one (All-1 fragment) must sent the maximum bits as possible. Only the last fragment need to introduce padding to reach the next boundary limit. Since the SCHC is known to be a multiple of 8 bits, the receiver can remove the extra bit to reach this limit.

Default padding mechanism do not need to send the padding length and can lead to a maximum of 14 bits of padding.

9. SCHC Compression for IPv6 and UDP headers

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

9.1. IPv6 version field

This field always holds the same value. Therefore, in the rule, TV is set to 6, MO to "equal" and CDA to "not-sent".

9.2. 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, 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(y).

9.3. 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(y).

9.4. 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-IPv6-length".

If the payload length needs to be sent and does not need to be coded in 16 bits, the TV can be set to 0x0000, the MO set to MSB(16-s) where 's' is the number of bits to code the maximum length, and CDA is set to LSB(s).

9.5. 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.

9.6. Hop Limit field

The field behavior for this field is different for Uplink and Downlink. In Uplink, since there is no IP forwarding between the Dev and the SCHC C/D, the value is relatively constant. On the other hand, the Downlink value depends of Internet routing and MAY change more frequently. One neat way of processing this field is to use the Direction Indicator (DI) to distinguish both directions:

- o in the Uplink, 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 Downlink, send the value: TV is not set, MO is set to "ignore" and CDA is set to "value-sent".

9.7. 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 frame (source or destination).

9.7.1. IPv6 source and destination prefixes

Both ends MUST be synchronized with the appropriate prefixes. For a specific flow, the source and destination prefixes can be unique and stored in the context. It can be either a link-local prefix or a global prefix. 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 25

Otherwise, the TV contains the prefix, the MO is set to "equal" and the CDA is set to "value-sent".

9.7.2. 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". Note that the LPWAN technology generally carries a single identifier corresponding to the DEV. Therefore Appiid cannot be used.

For privacy reasons or if the DEV address is changing over time, a static value that is not equal to the DEV address SHOULD be used. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDF is set to "not-sent". [RFC7217] provides some methods that MAY be used 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 extenso 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".

9.8. IPv6 extensions

No rule is currently defined that processes IPv6 extensions. If such extensions are needed, their compression/decompression rules can be based on the MOs and CDAs described above.

9.9. 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 frame (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".

9.10. UDP length field

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

If the payload is small, the TV can be set to 0x0000, the MO set to "MSB" and the CDA to "LSB".

In other cases, the length SHOULD be sent and the CDA is replaced by "value-sent".

9.11. UDP Checksum field

IPv6 mandates a checksum in the protocol above IP. Nevertheless, if a more efficient mechanism such as L2 CRC or MIC is carried by or over the L2 (such as in the LPWAN SCHC fragmentation process (see Section 7)), the UDP checksum transmission can be avoided. In that

case, the TV is not set, the MO is set to "ignore" and the CDA is set to "compute-checksum".

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

10. Security considerations

10.1. Security considerations for header compression

A malicious header compression could cause the reconstruction of a wrong packet that does not match with the original one. Such a corruption MAY be detected with end-to-end authentication and integrity mechanisms. Header Compression does not add more security problem than what is already needed in a transmission. For instance, to avoid an attack, never re-construct a packet bigger than some configured size (with 1500 bytes as generic default).

10.2. Security considerations for SCHC fragmentation

This subsection describes potential attacks to LPWAN SCHC fragmentation and suggests possible countermeasures.

A node can perform a buffer reservation attack by sending a first SCHC fragment to a target. Then, the receiver will reserve buffer space for the IPv6 packet. Other incoming SCHC fragmented 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. The (low) cost to mount this attack is linear with the number of buffers at the target node. However, the cost for an attacker can be increased if individual SCHC fragments of multiple packets can be stored in the reassembly buffer. To further increase the attack cost, the reassembly buffer can be splitted into SCHC fragment-sized buffer slots. Once a packet is complete, it is processed normally. If buffer overload occurs, a receiver can discard packets based on the sender behavior, which MAY help identify which SCHC fragments have been sent by an attacker.

In another type of attack, the malicious node is required to have overhearing capabilities. If an attacker can overhear a SCHC fragment, it can send a spoofed duplicate (e.g. with random payload) to the destination. If the LPWAN technology does not support suitable protection (e.g. source authentication and frame counters to prevent replay attacks), a receiver cannot distinguish legitimate from spoofed SCHC fragments. Therefore, the original IPv6 packet will be considered corrupt and will be dropped. To protect resource-

constrained nodes from this attack, it has been proposed to establish a binding among the SCHC fragments to be transmitted by a node, by applying content-chaining to the different SCHC fragments, based on cryptographic hash functionality. The aim of this technique is to allow a receiver to identify illegitimate SCHC fragments.

Further attacks MAY involve sending overlapped fragments (i.e. comprising some overlapping parts of the original IPv6 datagram). Implementers SHOULD make sure that the correct operation is not affected by such event.

In Window mode - ACK on error, a malicious node MAY force a SCHC fragment sender to resend a SCHC fragment a number of times, with the aim to increase consumption of the SCHC fragment sender's resources. To this end, the malicious node MAY repeatedly send a fake ACK to the SCHC fragment sender, with a Bitmap that reports that one or more SCHC fragments have been lost. In order to mitigate this possible attack, MAX_ACK_RETRIES MAY be set to a safe value which allows to limit the maximum damage of the attack to an acceptable extent. However, note that a high setting for MAX_ACK_RETRIES benefits SCHC fragment reliability modes, therefore the trade-off needs to be carefully considered.

11. Acknowledgements

Thanks to Dominique Barthel, Carsten Bormann, Philippe Clavier, Eduardo Ingles Sanchez, Arunprabhu Kandasamy, Rahul Jadhav, Sergio Lopez Bernal, Antony Markovski, Alexander Pelov, Pascal Thubert, Juan Carlos Zuniga, Diego Dujovne, Edgar Ramos, and Shoichi Sakane for useful design consideration and comments.

12. References

12.1. Normative References

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460, December 1998, <https://www.rfc-editor.org/info/rfc2460>.

[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>.

[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>.

12.2. **Informative References**

[I-D.ietf-lpwan-overview] Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-10 (work in progress), February 2018.

Appendix A. **SCHC Compression Examples**

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

The most common case using the mechanisms defined in this document will be a LPWAN 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 Device (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 24 presents the protocol stack for this Device. IPv6 and UDP are represented with dotted lines since these protocols are compressed on the radio link.

```
     Management   Data
    +----------+---------+---------+
    |   CoAP   |  CoAP   | legacy  |
    +----||----+---||----+---||----+
    .   UDP    .  UDP    |   UDP   |
    ................................
    .   IPv6   .  IPv6   .  IPv6   .
    +------------------------------+
    |    SCHC Header compression   |
    |      and fragmentation       |
    +------------------------------+
    |      LPWAN L2 technologies   |
    +------------------------------+
             DEV or NGW
```

Figure 24: Simplified Protocol Stack for LP-WAN

Note that in some LPWAN technologies, only the Devs have a device ID. Therefore, 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	equal	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	comp-length	
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	comp-length	
UDP checksum	16	1	Bi		ignore	comp-chk	

Rule 1

Field	FL	FP	DI	Value	Match	Action	Sent
					Opera.	Action	[bits]

IPv6 version	4	1	Bi	6	equal	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	comp-length	
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, fe80::/64]	match-mapping	mapping-sent	[1]
IPv6 DEViid	64	1	Bi		ignore	DEViid	
IPv6 APPprefix	64	1	Bi	[beta/64, alpha/64, fe80::64]	match-mapping	mapping-sent	[2]
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	comp-length	
UDP checksum	16	1	Bi		ignore	comp-chk	

Rule 2

Field	FL	FP	DI	Value	Match	Action	Sent
					Opera.	Action	[bits]
IPv6 version	4	1	Bi	6	equal	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	comp-length	
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)	[4]
UDP APPport	16	1	Bi	8720	MSB(12)	LSB(4)	[4]
UDP Length	16	1	Bi		ignore	comp-length	
UDP checksum	16	1	Bi		ignore	comp-chk	

Figure 25: Context rules

All the fields described in the three rules depicted on Figure 25 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 port numbers to 4 bits.

Appendix B. Fragmentation Examples

This section provides examples for the different fragment reliability modes specified in this document.

Figure 26 illustrates the transmission in No-ACK mode of an IPv6 packet that needs 11 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=1 + MIC --->|MIC checked: success =>
```

Figure 26: Transmission in No-ACK mode of an IPv6 packet carried by 11 fragments

In the following examples, N (i.e. the size if the FCN field) is 3 bits. Therefore, the All-1 FCN value is 7.

Figure 27 illustrates the transmission in ACK-on-Error of an IPv6 packet that needs 11 fragments, with MAX_WIND_FCN=6 and no fragment 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----->|
        (no ACK)
            |-----W=1, FCN=6----->|
            |-----W=1, FCN=5----->|
            |-----W=1, FCN=4----->|
            |--W=1, FCN=7 + MIC-->|MIC checked: success =>
            |<---- ACK, W=1 ------|
```

Figure 27: Transmission in ACK-on-Error mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6 and no loss.

Figure 28 illustrates the transmission in ACK-on-Error mode of an IPv6 packet that needs 11 fragments, with MAX_WIND_FCN=6 and three lost 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-->|             7
            |-----W=0, FCN=1----->|             /
            |-----W=0, FCN=0----->|       6543210
            |<-----ACK, W=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 + MIC ->|MIC checked: failed
            |<-----ACK, W=1-------|C=0 Bitmap:1100001
            |-----W=1, FCN=4----->|MIC checked: success =>
            |<---- ACK, W=1 ------|C=1, no Bitmap
```

Figure 28: Transmission in ACK-on-Error mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6 and three lost fragments.

Figure 29 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 11 fragments, with MAX_WIND_FCN=6 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-------| Bitmap:1111111
          |-----W=1, FCN=6----->|
          |-----W=1, FCN=5----->|
          |-----W=1, FCN=4----->|
          |--W=1, FCN=7 + MIC-->|MIC checked: success =>
          |<-----ACK, W=1-------| C=1 no Bitmap
        (End)
```

Figure 29: Transmission in ACK-Always mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6 and no lost fragment.

Figure 30 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 11 fragments, with MAX_WIND_FCN=6 and three lost fragments.

```
           Sender               Receiver
             |-----W=1, FCN=6----->|
             |-----W=1, FCN=5----->|
             |-----W=1, FCN=4--X-->|
             |-----W=1, FCN=3----->|
             |-----W=1, FCN=2--X-->|             7
             |-----W=1, FCN=1----->|             /
             |-----W=1, FCN=0----->|       6543210
             |<-----ACK, W=1-------|Bitmap:1101011
             |-----W=1, FCN=4----->|
             |-----W=1, FCN=2----->|
             |<-----ACK, W=1-------|Bitmap:
             |-----W=0, FCN=6----->|
             |-----W=0, FCN=5----->|
             |-----W=0, FCN=4--X-->|
             |--W=0, FCN=7 + MIC-->|MIC checked: failed
             |<-----ACK, W=0-------| C= 0 Bitmap:11000001
             |-----W=0, FCN=4----->|MIC checked: success =>
             |<-----ACK, W=0-------| C= 1 no Bitmap
           (End)
```

Figure 30: Transmission in ACK-Always mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6 and three lost fragments.

Figure 31 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 6 fragments, with MAX_WIND_FCN=6, three lost fragments and only one retry needed to recover each lost 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 + MIC-->|MIC checked: failed
                |<-----ACK, W=0-------|C= 0 Bitmap:1100001
                |-----W=0, FCN=4----->|MIC checked: failed
                |-----W=0, FCN=3----->|MIC checked: failed
                |-----W=0, FCN=2----->|MIC checked: success
                |<-----ACK, W=0-------|C=1 no Bitmap
              (End)
```

Figure 31: Transmission in ACK-Always mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6, three lost framents and only one retry needed for each lost fragment.

Figure 32 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 6 fragments, with MAX_WIND_FCN=6, three lost fragments, and the second 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 + MIC-->|MIC checked: failed
             |<-----ACK, W=0-------|C=0  Bitmap:1100001
             |-----W=0, FCN=4----->|MIC checked: failed
             |-----W=0, FCN=3----->|MIC checked: failed
             |-----W=0, FCN=2----->|MIC checked: success
             |  X---ACK, W=0-------|C= 1 no Bitmap
    timeout  |                     |
             |--W=0, FCN=7 + MIC-->|
             |<-----ACK, W=0-------|C= 1 no Bitmap

           (End)
```

Figure 32: Transmission in ACK-Always mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6, three lost fragments, and the second ACK lost.

Figure 33 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 6 fragments, with MAX_WIND_FCN=6, with three lost fragments, and one retransmitted 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 + MIC-->|MIC checked: failed
                |<-----ACK, W=0-------|C=0 Bitmap:1100001
                |-----W=0, FCN=4----->|MIC checked: failed
                |-----W=0, FCN=3----->|MIC checked: failed
                |-----W=0, FCN=2--X-->|
         timeout|                     |
                |--W=0, FCN=7 + MIC-->|All-0 empty
                |<-----ACK, W=0-------|C=0 Bitmap: 1111101
                |-----W=0, FCN=2----->|MIC checked: success
                |<-----ACK, W=0-------|C=1 no Bitmap
              (End)
```

Figure 33: Transmission in ACK-Always mode of an IPv6 packet carried by 11 fragments, with MAX_WIND_FCN=6, with three lost fragments, and one retransmitted fragment lost again.

Figure 34 illustrates the transmission in ACK-Always mode of an IPv6 packet that needs 28 fragments, with N=5, MAX_WIND_FCN=23 and two lost fragments. Note that MAX_WIND_FCN=23 may be useful when the maximum possible Bitmap size, considering the maximum lower layer technology payload size and the value of R, is 3 bytes. Note also that the FCN of the last fragment of the packet is the one with FCN=31 (i.e. FCN=2^N-1 for N=5, or equivalently, all FCN bits set to 1).

```
         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 ----->|
           |                      |lcl-Bitmap:110111111111101111111111
           |<------ACK, W=0-------|encoded Bitmap:1101111111111011
           |-----W=0, FCN=21----->|
           |-----W=0, FCN=10----->|
           |<------ACK, W=0-------|no Bitmap
           |-----W=1, FCN=23----->|
           |-----W=1, FCN=22----->|
           |-----W=1, FCN=21----->|
           |--W=1, FCN=31 + MIC-->|MIC checked: sucess =>
           |<------ACK, W=1-------|no Bitmap
         (End)
```

Figure 34: Transmission in ACK-Always mode of an IPv6 packet carried by 28 fragments, with N=5, MAX_WIND_FCN=23 and two lost 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:

```
                +===========+
   +------------+  Init     |
   |  FCN=0     +===========+
   |  No Window
   |  No Bitmap
   |                   +-------+
   |          +========+==+    | More Fragments
   |          |           | <--+ ~~~~~~~~~~~~~~~~~~~~
   +--------> |   Send    |      send Fragment (FCN=0)
              +===+=======+
                  |  last fragment
                  |  ~~~~~~~~~~~~
                  |  FCN = 1
                  v  send fragment+MIC
              +============+
              |    END     |
              +============+
```

Figure 35: Sender State Machine for the No-ACK Mode

```
                         +------+ Not All-1
              +==========+=+    | ~~~~~~~~~~~~~~~~~~~
              |            + <--+ set Inactivity Timer
              |  RCV Frag  +-------+
              +=+===+======+       |All-1 &
      All-1 &   |   |              |MIC correct
    MIC wrong   |   |Inactivity    |
                |   |Timer Exp.    |
                v   |              |
     +==========++  |              v
     |   Error   |<-+     +========+==+
     +===========+        |    END    |
                          +===========+
```

Figure 36: Receiver State Machine for the No-ACK Mode

```
                    +=======+
                    | INIT  |       FCN!=0 & more frags
                    |       |       ~~~~~~~~~~~~~~~~~~~~~~
                    +======++  +--+ send Window + frag(FCN)
                       W=0 |   |  | FCN-
        Clear local Bitmap |   |  v set local Bitmap
             FCN=max value |  ++==+========+
                           +> |            |
      +---------------------> |    SEND    |
      |                       +==+===+=====+
      |      FCN==0 & more frags |   | last frag
      |    ~~~~~~~~~~~~~~~~~~~~~ |   | ~~~~~~~~~~~~~~~
      |         set local-Bitmap |   | set local-Bitmap
      |   send wnd + frag(all-0) |   | send wnd+frag(all-1)+MIC
      |       set Retrans_Timer  |   | set Retrans_Timer
      |                          |   |
      |Recv_wnd == wnd &         |   |
      |Lcl_Bitmap==recv_Bitmap&  |   |  +----------------------+
      |more frag                 |   |  |lcl-Bitmap!=rcv-Bitmap|
      |~~~~~~~~~~~~~~~~~~~~~~    |   |  | ~~~~~~~~~            |
      |Stop Retrans_Timer        |   |  | Attemp++             v
      |clear local_Bitmap        v   v  |                +=====+=+
      |window=next_window   +====+===+==+===+            |Resend |
      +---------------------+               |            |Missing|
                       +----+     Wait      |            |Frag   |
      not expected wnd |    |    Bitmap     |            +=======+
      ~~~~~~~~~~~~~~~  +--->+               ++Retrans_Timer Exp  |
          discard frag      +==+=+===+=+==+=+| ~~~~~~~~~~~~~~~~~ |
                               | |   | ^  ^  |reSend(empty)All-* |
                               | |   | |  |  |Set Retrans_Timer  |
      MIC_bit==1 &             | |   | |  +--+Attemp++           |
      Recv_window==window &    | |   | +-------------------------+
      Lcl_Bitmap==recv_Bitmap &| |   |   all missing frag sent
                   no more frag| |   |   ~~~~~~~~~~~~~~~~~~~~~~
       ~~~~~~~~~~~~~~~~~~~~~~~~| |   |   Set Retrans_Timer
             Stop Retrans_Timer| |   |
       +=============+         | |   |
       |     END     +<--------+ |   | Attemp > MAX_ACK_REQUESTS
       +=============+           |   | ~~~~~~~~~~~~~~~~~~
                    All-1 Window |   v Send Abort
                    ~~~~~~~~~~~~ | +=+===========+
                   MIC_bit ==0 & +>|    ERROR    |
          Lcl_Bitmap==recv_Bitmap  +=============+
```

Figure 37: Sender State Machine for the ACK-Always Mode

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

```
          --->* ABORT
                ~~~~~~~
                Inactivity_Timer = expires
           When DWN_Link
             IF Inactivity_Timer expires
                 Send DWL Request
                 Attemp++
```

Figure 38: Receiver State Machine for the ACK-Always Mode

```
                      +=======+
                      |       |
                      | INIT  |
                      |       |        FCN!=0 & more frags
                      +======++  +--+  ~~~~~~~~~~~~~~~~~~~~~~~
                         W=0 |   |  |  send Window + frag(FCN)
          ~~~~~~~~~~~~~~~~~~ |   |  |  FCN-
          Clear local Bitmap |   |  v  set local Bitmap
               FCN=max value |  ++=============+
                             +> |              |
                                |     SEND     |
    +-------------------------> |              |
    |                           ++=====+=======+
    |         FCN==0 & more frags|     |last frag
    |     ~~~~~~~~~~~~~~~~~~~~~~~|     |~~~~~~~~~~~~~~~~~~
    |            set local-Bitmap|     |set local-Bitmap
    |      send wnd + frag(all-0)|     |send wnd+frag(all-1)+MIC
    |           set Retrans_Timer|     |set Retrans_Timer
    |                            |     |
    |Retrans_Timer expires &     |     |   lcl-Bitmap!=rcv-Bitmap
    |more fragments              |     |   ~~~~~~~~~~~~~~~~~~~~~~~
    |~~~~~~~~~~~~~~~~~~~~~       |     |   Attemp++
    |stop Retrans_Timer          |     |  +-----------------+
    |clear local-Bitmap          v     v  |                 v
    |window = next window  +=====+=====+==+==+         +====+====+
    +----------------------+                 +         | Resend  |
    +--------------------->+    Wait Bitmap  |         | Missing |
    |                  +-- +                 |         | Frag    |
    | not expected wnd |   ++=+===+===+===+==+         +======+==+
    | ~~~~~~~~~~~~~~~~ |    ^ |   |   |   ^                   |
    |    discard frag  +----+ |   |   |   +-------------------+
    |                         |   |   |     all missing frag sent
    |Retrans_Timer expires &  |   |   |     ~~~~~~~~~~~~~~~~~~~~~
    |       No more Frag      |   |   |     Set Retrans_Timer
    | ~~~~~~~~~~~~~~~~~~~~~~~ |   |   |
    |  Stop Retrans_Timer     |   |   |
    |  Send ALL-1-empty       |   |   |
    +-------------------------+   |   |
                                  |   |
         Local_Bitmap==Recv_Bitmap|   |
         ~~~~~~~~~~~~~~~~~~~~~~~~~|   |Attemp > MAX_ACK_REQUESTS
    +=========+Stop Retrans_Timer |   |~~~~~~~~~~~~~~~~~~~~~~~~
    |   END   +<------------------+   v  Send Abort
    +=========+                     +=+=========+
                                    |   ERROR   |
                                    +===========+
```

Figure 39: Sender State Machine for the ACK-on-Error Mode

```
      Not All- & w=expected +---+   +---+w = Not expected
      ~~~~~~~~~~~~~~~~~~~~~ |   |   |   |~~~~~~~~~~~~~~~~~
      Set local_Bitmap(FCN) |   v   v   |discard
                           ++===+===+===+=+
   +-----------------------+              +--+ All-0 & full
   |            ABORT *<---+  Rcv Window  |  | ~~~~~~~~~~~~
   |  +--------------------+              +<-+ w =next
   |  |     All-0 empty +->+=+=+===+======+ clear lcl_Bitmap
   |  |     ~~~~~~~~~~~ |    | |   ^
   |  |     send bitmap +----+ |   |w=expct & not-All & full
   |  |                        |   |~~~~~~~~~~~~~~~~~~~~~~~~
   |  |                        |   |set lcl_Bitmap; w =nxt
   |  |                        |   |
   |  |      All-0 & w=expect  |   |     w=next
   |  |      & no_full Bitmap  |   |    ~~~~~~~~  +========+
   |  |      ~~~~~~~~~~~~~~~~~ |   |    Send abort| Error/ |
   |  |      send local_Bitmap |   |  +---------->+ Abort  |
   |  |                        |   |  | +-------->+========+
   |  |                        v   |  | |   all-1       ^
   |  |    All-0 empty    +====+===+==+=+=+ ~~~~~~~     |
   |  |  ~~~~~~~~~~~~~ +--+    Wait       | Send abort  |
   |  |  send lcl_btmp +->| Missing Fragm.|             |
   |  |                   +==============++             |
   |  |                                  +--------------+
   |  |                                   Uplink Only &
   |  |                             Inactivity_Timer = expires
   |  |                             ~~~~~~~~~~~~~~~~~~~~~~~~~~
   |  |                              Send Abort
   |  |All-1 & w=expect & MIC wrong
   |  |~~~~~~~~~~~~~~~~~~~~~~~~~~~~      +-+  All-1
   |  |set local_Bitmap(FCN)             | v  ~~~~~~~~~~
   |  |send local_Bitmap     +===========+==+ snd lcl_btmp
   |  +--------------------->+   Wait End   +-+
   |                         +=====+=+====+=+ | w=expct &
   |       w=expected & MIC right  | |    ^   | MIC wrong
   |       ~~~~~~~~~~~~~~~~~~~~~~  | |    +---+ ~~~~~~~~~
   |  set & send local_Bitmap(FCN) | | set lcl_Bitmap(FCN)
   |                               | |
   |All-1 & w=expected & MIC right | +-->* ABORT
   |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ v
   |set & send local_Bitmap(FCN) +=+==========+
   +---------------------------->+     END    |
                                 +============+
               --->* ABORT
                    Only Uplink
                    Inactivity_Timer = expires
                    ~~~~~~~~~~~~~~~~~~~~~~~~~~
                    Send Abort
```

Figure 40: Receiver State Machine for the ACK-on-Error Mode

Appendix D. Note

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.

Authors' Addresses

Ana Minaburo
Acklio
2bis rue de la Chataigneraie
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