           LPWAN Static Context Header Compression (SCHC) and fragmentation for
                              IPv6 and UDP
                draft-ietf-lpwan-ipv6-static-context-hc-15

Abstract

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

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

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

   The SCHC header compression and fragmentation mechanisms are
   independent of the specific LPWAN technology over which they are
   used.  Note that this document defines generic functionalities and
   advisedly offers flexibility with regard to parameter settings and
   mechanism choices.  Such settings and choices are expected to be made
   in other technology-specific documents.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF).  Note that other groups may also distribute

working documents as Internet-Drafts.  The list of current Internet-
Drafts is at https://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on December 31, 2018.

Copyright Notice

Table of Contents

1.  Introduction

   This document defines the Static Context Header Compression (SCHC)
   framework, which provides both header compression and fragmentation
   functionalities.  SCHC has been 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 network topology is star-oriented, which means that all
      packets follow the same path.  For the needs of this document, the
      architecture can simply be described as Devices (Dev) exchanging
      information with LPWAN Application Servers (App) through Network
      Gateways (NGW).

   o  Because devices embed built-in applications, the traffic flows to
      be compressed are known in advance.  Indeed, 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, in which information about header
   fieds is stored.  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 (see [RFC8376]).  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 [RFC8200] 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 there is no out-of-sequence delivery of data units
   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.  Such settings and choices are expected to be made
in other, technology-specific documents.

## 2.  Requirements Notation

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

## 3.  LPWAN Architecture

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

o Devices (Dev) are the end-devices or hosts (e.g. sensors,
actuators, etc.).  There can be a very high density of devices per
radio gateway.

o The Radio Gateway (RGW), which is the end point of the constrained
link.

o The Network Gateway (NGW) is the interconnection node between the
Radio Gateway and the Internet.

o 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
```
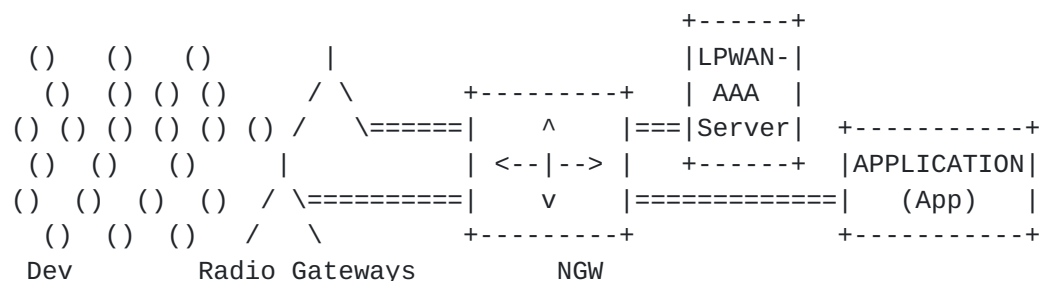
Figure 1: LPWAN Architecture

4.  **Terminology**

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

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

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

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

   o  All-1.  The SCHC Fragment format for the last fragment of the SCHC
      Packet.

   o  All-0 empty.  An All-0 SCHC Fragment without payload.  It is used
      to request the SCHC 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 payload.  It is used
      to request the SCHC ACK with the encoded Bitmap when the
      Retransmission Timer expires in the last window of a packet.

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

   o  AppIID: Application Interface Identifier.  The IID that identifies
      the application server interface.

   o  Bi: Bidirectional.  Characterises a Rule Entry that applies to
      headers of packets travelling in either direction (Up and Dw, see
      this glossary).

   o  Bitmap: a bit field in the SCHC ACK message that tells the sender
      which SCHC Fragments in a window of fragments were correctly
      received.

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

   o  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.

   o  Compression Residue.  The bits that need to be sent (beyond the
      Rule ID itself) after applying the SCHC compression over each
      header field.

   o  Context: A set of Rules used to compress/decompress headers.

   o  Dev: Device.  A node connected to an LPWAN.  A Dev SHOULD
      implement SCHC.

   o  DevIID: Device Interface Identifier.  The IID that identifies the
      Dev interface.

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

   o  DTag: Datagram Tag. This SCHC F/R header field is set to the same
      value for all SCHC Fragments carrying the same SCHC Packet.

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

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

   o  Field Description.  A line in the Rule table.

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

   o  FL: Field Length is the length of the packet header field.  It is
      expressed in bits for header fields of fixed lengths or as a type
      (e.g. variable, token length, ...) for field lengths that are
      unknown at the time of Rule creation.  The length of a header
      field is defined in the corresponding protocol specification.

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

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

o  Inactivity Timer.  A timer used after receiving a SCHC Fragment to
   detect when, due to a communication error, there is no possibility
   to continue an on-going fragmented SCHC Packet transmission.

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

o  L2 Word: this is the minimum subdivision of payload data that the
   L2 will carry.  In most L2 technologies, the L2 Word is an octet.
   In bit-oriented radio technologies, the L2 Word might be a single
   bit.  The L2 Word size is assumed to be constant over time for
   each device.

o  MIC: Message Integrity Check.  A SCHC F/R header field computed
   over the fragmented SCHC Packet and potential fragment padding,
   used for error detection after SCHC Packet reassembly.

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

o  Padding (P).  Extra bits that may be appended by SCHC to a data
   unit that it passes to the underlying Layer 2 for transmission.
   SCHC itself operates on bits, not bytes, and does not have any
   alignment prerequisite.  See Section 9.

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

o  Rule: A set of header field values.

o  Rule entry: A column in a Rule that describes a parameter of the
   header field.

o  Rule ID: An identifier for a Rule.  SCHC C/D on both sides share
   the same Rule ID for a given packet.  A set of Rule IDs are used
   to support SCHC F/R functionality.

o  SCHC ACK: A SCHC acknowledgement for fragmentation.  This message
   is used to report on the success of reception of a set of SCHC
   Fragments.  See Section 8 for more details.

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

o  SCHC F/R: Static Context Header Compression Fragmentation/
   Reassembly.  A protocol used on both sides, at the Dev and at the
   network, to achieve Fragmentation/Reassembly of SCHC Packets.
   SCHC F/R has three reliability modes.

o  SCHC Fragment: A data unit that carries a subset of a SCHC Packet.
   SCHC F/R is needed when the size of a SCHC packet exceeds the
   available payload size of the underlying L2 technology data unit.
   See Section 8.

o  SCHC Packet: A packet (e.g. an IPv6 packet) whose header has been
   compressed as per the header compression mechanism defined in this
   document.  If the header compression process is unable to actually
   compress the packet header, the packet with the uncompressed
   header is still called a SCHC Packet (in this case, a Rule ID is
   used to indicate that the packet header has not been compressed).
   See Section 7 for more details.

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

o  Up: Uplink direction for compression/decompression 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 ACK-on-Error
   or ACK-Always mode Section 8, which carries the same value for all
   SCHC Fragments of a window.

o  Window: A subset of the SCHC Fragments needed to carry a SCHC
   Packet (see Section 8).

## 5.  SCHC overview

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

```
            +----------------+
            |      IPv6      |
       +- +----------------+
       |  |   Compression  |
  SCHC <   +----------------+
       |  |  Fragmentation |
       +- +----------------+
          |LPWAN technology|
          +----------------+
```
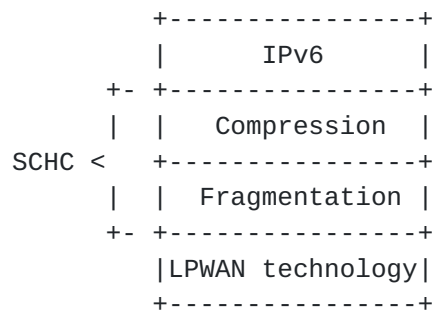
Figure 2: Protocol stack comprising IPv6, SCHC and an LPWAN
technology

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 or may
not actually be smaller than that of the original packet) is called a
SCHC Packet.  If the SCHC Packet size exceeds the layer 2 (L2) MTU,
fragmentation is then applied to the SCHC Packet.  The SCHC Packet or
the SCHC Fragments are then transmitted over the LPWAN.  The
reciprocal operations take place at the receiver.  This process is
illustrated in Figure 3.

```
A packet (e.g. an IPv6 packet)
          |                                            ^
          v                                            |
+------------------+                        +--------------------+
| SCHC Compression |                        | SCHC Decompression |
+------------------+                        +--------------------+
          |                                            ^
          |    If no fragmentation (*)                 |
          +------------- SCHC Packet  -------------->|
          |                                            |
          v                                            |
+--------------------+                      +-----------------+
| SCHC Fragmentation |                      | SCHC Reassembly |
+--------------------+                      +-----------------+
        |      ^                                  |     ^
        |      |                                  |     |
        |      +------------- SCHC ACK -------------+     |
        |                                          |
        +------------- SCHC Fragments ------------------+


        SENDER                                  RECEIVER
```
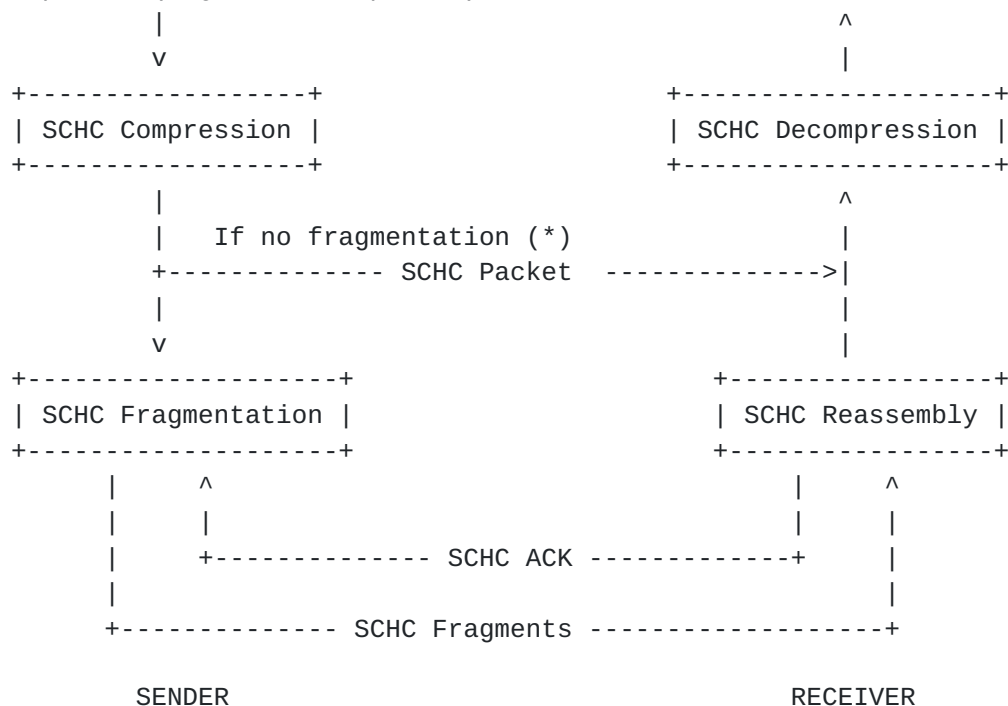
*: the decision to use Fragmentation or not is left to each LPWAN technology
   over which SCHC is applied. See LPWAN technology-specific documents.


   Figure 3: SCHC operations taking place at the sender and the receiver

   The SCHC Packet is composed of the Compressed Header followed by the
   payload from the original packet (see Figure 4).  The Compressed
   Header itself is composed of a Rule ID and a Compression Residue.
   The Compression Residue may be absent, see [Section 7](#).  Both the Rule
   ID and the Compression Residue potentially have a variable size, and
   generally are not a mutiple of bytes in size.

```
   |  Rule ID +  Compression Residue |
   +---------------------------------+-------------------+
   |        Compressed Header         |      Payload      |
   +---------------------------------+-------------------+
```
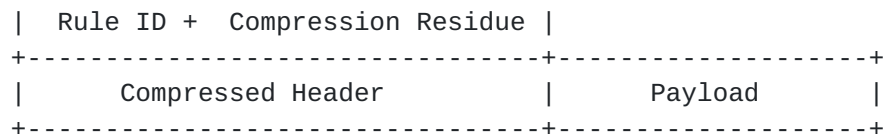

                        Figure 4: SCHC Packet

   The Fragment Header size is variable and depends on the Fragmentation
   parameters.  The Fragment payload contains a part of the SCHC Packet
   Compressed Header, a part of the SCHC Packet Payload or both.  Its

size depends on the L2 data unit, see Section 8.  The SCHC Fragment
has the following format:

```
| Rule ID + DTAG + W + FCN [+ MIC ] |   Partial  SCHC Packet  |
+----------------------------------+-------------------------+
|          Fragment Header          |   Fragment  Payload     |
+----------------------------------+-------------------------+
```
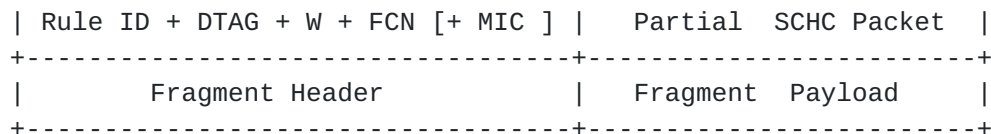
              Figure 5: SCHC Fragment

The SCHC ACK is only used for Fragmentation.  It has the following
format:

```
|Rule ID + DTag + W|
+-----------------+-------- ... ---------+
|    ACK Header    |    encoded Bitmap    |
+-----------------+-------- ... ---------+
```
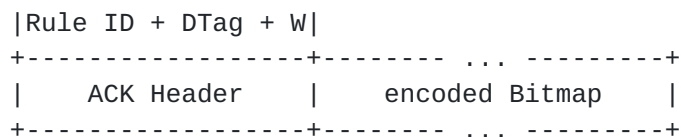
                Figure 6: SCHC ACK

The SCHC ACK Header and the encoded Bitmap both have variable size.

Figure 7 below maps the functional elements of Figure 3 onto the
LPWAN architecture elements of Figure 1.

```
      Dev                                         App
+----------------+                         +--------------+
| APP1 APP2 APP3 |                         |APP1 APP2 APP3|
|                |                         |              |
|      UDP       |                         |     UDP      |
|      IPv6      |                         |     IPv6     |
|                |                         |              |
|SCHC C/D and F/R|                         |              |
+--------+-------+                         +-------+------+
       |   +--+     +----+     +-----------+          .
       +~~ |RG| === |NGW | === |   SCHC    |... Internet ..
           +--+     +----+     |F/R and C/D|
                               +-----------+
```

              Figure 7: Architecture

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

Let's describe the operation 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 F/R is
performed (see Section 8).  The resulting SCHC Fragments are sent as
one or more L2 frames to an LPWAN Radio Gateway (RG) which forwards
them to a Network Gateway (NGW).  The NGW sends the data to a SCHC F/
R and then to the SCHC C/D for decompression.  The SCHC F/R and C/D
on the Network side can be located in the NGW or somewhere else as
long as a tunnel is established between them and the NGW.  Note that,
for some LPWAN technologies, it MAY be suitable to locate the SCHC F/
R functionality nearer the NGW, in order to better deal with time
constraints of such technologies.  The SCHC C/D and F/R 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 C/D and F/R process is symmetrical, therefore the
description of the Downlink direction trivially derives from the one
above.

## 6. Rule ID

Rule IDs are identifiers used to select the correct context either
for Compression/Decompression or for Fragmentation/Reassembly.

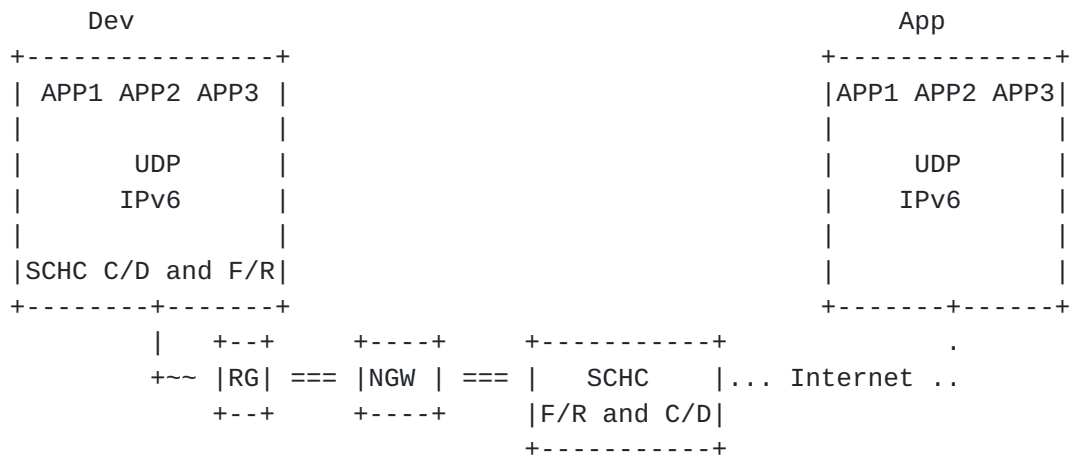The size of the Rule IDs is not specified in this document, as it is
implementation-specific and can vary according to the LPWAN
technology and the number of Rules, among others.

The Rule IDs are used:

o  In the SCHC C/D context, to identify the Rule (i.e., the set of
   Field Descriptions) that is used to compress a packet header.

o  At least one Rule ID MAY be allocated to tagging packets for which
   SCHC compression was not possible (no matching Rule was found).

o  In SCHC F/R, to identify the specific modes and settings of SCHC
   Fragments being transmitted, and to identify the SCK ACKs,
   including their modes and settings.  Note that in the case of
   bidirectional communication, at least two Rule ID values are
   therefore needed for F/R.

## 7. 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 is 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 be learned by a
provisioning protocol or 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.

## 7.1.  SCHC C/D Rules

The main idea of the SCHC compression scheme is to transmit the Rule
ID to the other end instead of sending known field values.  This Rule
ID identifies a Rule that 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
Rules MAY be changed at run-time but the way to do this will be
specified in another document.

The context contains a list of Rules (cf.  Figure 8).  Each Rule
itself contains a list of Field Descriptions composed of a Field
Identifier (FID), a Field Length (FL), a Field Position (FP), a
Direction Indicator (DI), a Target Value (TV), a Matching Operator
(MO) and a Compression/Decompression Action (CDA).

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

Figure 8: Compression/Decompression Context

A Rule does not describe how to parse a packet header to find each
field.  This MUST be known from the compressor/decompressor.  Rules
only describe the compression/decompression behavior for each header
field.  In a Rule, the Field Descriptions are listed in the order in
which the fields appear in the packet header.

A Rule also describes what Compression Residue is sent.  The
Compression Residue is assembled by concatenating the residues for
each field, in the order the Field Descriptions appear 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.  It can be
   either a fixed value (in bits) if the length is known when the
   Rule is created or a type if the length is variable.  The length
   of a header field is defined in the corresponding protocol
   specification.  The type defines the process to compute the
   length, its unit (bits, bytes,...) and the value to be sent before
   the Compression Residue.

o  Field Position (FP): most often, a field only occurs once in a
   packet header.  Some fields may occur multiple times in a header.
   FP indicates which occurrence this Field Description applies to.
   The default value is 1 (first occurence).

o  A Direction Indicator (DI) indicates the packet direction(s) this
   Field Description applies to.  Three values are possible:

   *  UPLINK (Up): this Field Description is only applicable to
      packets sent by the Dev to the App,

   *  DOWNLINK (Dw): this Field Description is only applicable to
      packets sent from the App to the Dev,

   *  BIDIRECTIONAL (Bi): this Field Description is applicable to
      packets travelling both Up and Dw.

o  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.

o  Matching Operator (MO) is the operator used to match the Field
   Value and the Target Value.  The Matching Operator may require

some parameters.  MO is only used during the compression phase.
The set of MOs defined in this document can be found in
Section 7.4.

o  Compression Decompression Action (CDA) describes the compression
   and decompression processes to be performed after the MO is
   applied.  Some CDAs MAY require parameter values for their
   operation.  CDAs are used in both the compression and the
   decompression functions.  The set of CDAs defined in this document
   can be found in Section 7.5.

## 7.2.  Rule ID for SCHC C/D

Rule IDs are sent by the compression function in one side and are
received for the decompression function in the other side.  In SCHC
C/D, the Rule IDs are specific to 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.

## 7.3.  Packet processing

The compression/decompression process follows several steps:

o  Compression Rule selection: The goal is to identify which Rule(s)
   will be used to compress the packet's headers.  When doing
   decompression, on 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 DevIID and the Rule ID.  On 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 Field 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 FP 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 packet field's value is then compared to the corresponding
      Target Value (TV) stored in the Rule for that specific field
      using the matching operator (MO).

      If 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 Compression Residue) SHOULD
      be obtained.  Otherwise, the next Rule is tested.

   *  If no eligible Rule is found, then the header MUST be sent
      without compression.  This MAY require the use of the SCHC F/R
      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 Compression
      Residue is the concatenation of the Compression Residues for each
      field according to the CDAs for that Rule.  The way the Rule ID is
      sent depends on the specific underlying LPWAN technology.  For
      example, it can be either included in an L2 header or sent in the
      first byte of the L2 payload.  (Cf.  Figure 9).  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 9 for details.

   o  Decompression: When doing decompression, on 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 DevIID and the Rule ID.  On 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 |
+--- ... --+------- ... -------+-----------------+

|----- compressed header ------|
```

                    Figure 9: SCHC C/D Packet Format

## 7.4.  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
      packet header field value are equal to the TV in the Rule.  The x
      parameter of the MSB MO indicates how many bits are involved in
      the comparison.  If the FL is described as variable, the length
      must be a multiple of the unit.  For example, x must be multiple
      of 8 if the unit of the variable length is in bytes.

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

## 7.5.  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 context |
|value-sent         |send        |build from received value   |
|mapping-sent       |send index  |value from index on a table |
|LSB                |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  |
\-------------------+------------+---------------------------/
```

Figure 10: Compression and Decompression Actions

Figure 10 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 a
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 (using
the unit defined in the FL) MUST be sent first using the following
coding:

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

o  For values between 15 and 254, the first 4 bits sent are set to 1
   and the size is sent using 8 bits integer.

o  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 is not present in the packet but exists in the Rule and
its FL is specified as being variable, size 0 MUST be sent to denote
its absence.

### 7.5.1.  not-sent CDA

The not-sent function is generally used when the field value is
specified in a 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 original field that was compressed.

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

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

### 7.5.2.  value-sent CDA

The value-sent action is generally used when the field value is not
known by both the Compressor and the Decompressor.  The value is sent
as a residue 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 by explicitly indicating the length
in the Compression Residue, as defined in Section 7.5.  This function
is generally used with the "ignore" MO.

### 7.5.3.  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.

### 7.5.4.  LSB CDA

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

If this action needs to be done on a variable length field, the size
of the Compression Residue in bytes MUST be sent as described in
Section 7.5.

### 7.5.5.  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 L2
header, or from some other stable identifier.  The computation is
specific to each LPWAN technology and MAY depend on the Device ID
size.

In the downlink direction (Dw), at the compressor, this DevIID CDA
may be used to generate the L2 addresses on the LPWAN, based on the
packet destination address.

### 7.5.6.  Compute-*

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

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

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

## 8.  Fragmentation

### 8.1.  Overview

In LPWAN technologies, the L2 data unit size typically varies from
tens to hundreds of bytes.  The SCHC F/R (Fragmentation /Reassembly)
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 F/R 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 F/R mechanism.

This document also assumes that the L2 data unit size does not vary
while a fragmented SCHC Packet is being transmitted.

To adapt the SCHC F/R to the capabilities of LPWAN technologies, it
is required to enable optional SCHC Fragment retransmission and to
allow for a range of reliability options for sending the 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.

SCHC F/R uses the knowledge of the L2 Word size (see Section 4) to
encode some messages.  Therefore, SCHC MUST know the L2 Word size.
SCHC F/R generates SCHC Fragments and SCHC ACKs that are, for most of
them, multiples of L2 Words.  The padding overhead is kept to the
absolute minimum.  See Section 9.

## 8.2.  Fragmentation Tools

This subsection describes the different tools that are used to enable
the SCHC F/R functionality defined in this document, such as fields
in the SCHC F/R header frames (see the related formats in
Section 8.4), windows and timers.

o  Rule ID.  The Rule ID is present in the SCHC Fragment header and
   in the SCHC ACK header formats.  The Rule ID in a SCHC Fragment
   header is used to identify that a SCHC Fragment is being carried,
   which SCHC F/R reliability mode is used and which window size is
   used.  The Rule ID in the SCHC Fragment header also allows
   interleaving non-fragmented SCHC Packets and SCHC Fragments that
   carry other SCHC Packets.  The Rule ID in a SCHC ACK identifies
   the message as a SCHC 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 F/R
   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, the size 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 of a fragmented SCHC 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 to 0.  For further details see
   Section 8.5.  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 SCHC Packets.  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 SCHC ACK format, DTag carries the same value as
      the DTag field in the SCHC Fragments for which this SCHC ACK is
      intended.  When there is no Dtag, there can be only one SCHC
      Packet in transit.  Only after all its fragments have been
      transmitted can another SCHC Packet be sent.  The length of DTag,
      denoted T, is not specified in this document because it is
      technology dependant.  It will be defined in the corresponding
      technology-specific documents, based on the number of simultaneous
      packets that are to be supported.

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 SCHC
   ACK format, this field also has a size of 1 bit.  In all SCHC
   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 SCHC ACK.

o  Message Integrity Check (MIC).  This field is computed by the
   sender over the complete SCHC Packet and 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 as well as the length in bits of the MIC used.

o  C (MIC checked): C is a 1-bit field.  This field is used in the
   SCHC 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).

o  Retransmission Timer.  A SCHC Fragment sender uses it after the
   transmission of a window to detect a transmission error of the
   SCHC ACK corresponding to this window.  Depending on the
   reliability mode, it will lead to a request a SCHC 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-specific documents.

o  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.  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
   corresponding technology-specific document.

o  Attempts.  This counter counts the requests for a missing SCHC
   ACK.  When it reaches the value MAX_ACK_REQUESTS, the sender
   assumes there are recurrent SCHC Fragment transmission errors and
   determines that an Abort is needed.  The default value

MAX_ACK_REQUESTS is not stated in this document, and it is
expected to be defined in the corresponding technology-specific
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 a SCHC 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 SCHC ACK for transmission from
   the receiver to the sender, the Bitmap is shortened for energy/
   bandwidth optimisation, see more details in Section 8.4.3.1.

o  Abort.  On expiration of the Inactivity timer, or when Attempts
   reaches MAX_ACK_REQUESTS or upon occurrence of some other error,
   the sender or the receiver may use the Abort.  When the receiver
   needs to abort the on-going fragmented SCHC 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 Aborts are acknowledged.

## 8.3.  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 SCHC Packet.

o  No-ACK.  No-ACK is the simplest SCHC Fragment reliability mode.
   The receiver does not generate overhead in the form of
   acknowledgements (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 SCHC
   ACKs.  See further details in Section 8.5.1.

o  ACK-Always.  The ACK-Always mode provides flow control using a
   windowing 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 SCHC ACK use.  In ACK-Always, the receiver sends a SCHC
   ACK after a window of SCHC Fragments has been received.  The SCHC

ACK is used to inform the sender which SCHC Fragments in the
current window have been well received.  Upon a SCHC ACK
reception, the sender retransmits the lost SCHC Fragments.  When a
SCHC ACK is lost and the sender has not received it by the
expiration of the Retransmission Timer, the sender uses a SCHC ACK
request by sending the All-0 empty SCHC Fragment when it is not
the last window and the All-1 empty Fragment when it is the last
window.  The maximum number of SCHC ACK requests is
MAX_ACK_REQUESTS.  If MAX_ACK_REQUESTS is reached, the
transmission needs to be aborted.  See further details in
Section 8.5.2.

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 SCHC ACKs
   transmitted, which MAY be especially beneficial in asymmetric
   scenarios.  The receiver transmits a SCHC 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 a SCHC 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 SCHC ACK gives the state of all the SCHC Fragments
   of the current window (received or lost).  Upon a SCHC ACK
   reception, the sender retransmits any lost SCHC Fragments based on
   the SCHC ACK.  If a SCHC 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 a SCHC ACK is lost,
   the sender assumes that all SCHC Fragments covered by the lost
   SCHC 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 and it is still
   expecting fragments from the previous window, the receiver will
   abort the on-going fragmented packet transmission.  See further
   details in Section 8.5.3.

The same reliability mode MUST be used for all SCHC Fragments of a
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
modes are relevant for a specific LPWAN technology.

Examples of the different reliability modes described are provided in
[Appendix B](#).

## 8.4.  Fragmentation Formats

This section defines the SCHC Fragment format, including the All-0
and All-1 formats and their "empty" variations, the SCHC ACK format
and the Abort formats.

A SCHC Fragment conforms to the general format shown in Figure 11.
It comprises a SCHC Fragment Header and a SCHC Fragment Payload.  In
addition, the last SCHC Fragment carries as many padding bits as
needed to fill up an L2 Word.  The SCHC Fragment Payload carries a
subset of the SCHC Packet.  The SCHC Fragment is the data unit passed
on to the L2 for transmission.

```
+-----------------+-----------------------+~~~~~~~~~~~~~~~~~~~~
| Fragment Header |   Fragment payload    | padding (as needed)
+-----------------+-----------------------+~~~~~~~~~~~~~~~~~~~~
```

Figure 11: SCHC Fragment general format.  Presence of a padding field
is optional

### 8.4.1.  Fragments that are not the last one

In ACK-Always or ACK-on-Error, SCHC Fragments except the last one
SHALL conform to the detailed format defined in Figure 12.

```
 |----- Fragment Header -----|
          |-- T --|1|-- N --|
 +-- ... --+- ... -+-+- ... -+--------..-------+
 | Rule ID | DTag  |W|  FCN  | Fragment payload |
 +-- ... --+- ... -+-+- ... -+--------..-------+
```

Figure 12: Fragment Detailed Format for Fragments except the Last
One, ACK-Always and ACK-on-Error

In the No-ACK mode, SCHC Fragments except the last one SHALL conform
to the detailed format defined in Figure 13.

```
 |---- Fragment Header ----|
          |-- T --|-- N --|
  +-- ... --+- ... -+- ... -+--------..-------+
  | Rule ID |  DTag |  FCN  | Fragment payload |
  +-- ... --+- ... -+- ... -+--------..-------+
```

Figure 13: Fragment Detailed Format for Fragments except the Last
One, No-ACK mode

The total size of the fragment header is not necessarily a multiple
of the L2 Word size.  To build the fragment payload, SCHC F/R MUST
take from the SCHC Packet a number of bits that makes the SCHC
Fragment an exact multiple of L2 Words.  As a consequence, no padding
bit is used for these fragments.

### 8.4.1.1.  All-0 fragment

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

```
 |----- Fragment Header -----|
          |-- T --|1|-- N --|
  +-- ... --+- ... -+-+- ... -+--------..-------+
  | Rule ID | DTag  |W|  0..0 | Fragment payload |
  +-- ... --+- ... -+-+- ... -+--------..-------+
```

Figure 14: All-0 fragment detailed format

This is simply an instance of the format described in Figure 12.  An
All-0 fragment payload MUST be at least the size of an L2 Word.  The
rationale is that the All-0 empty fragment (see Section 8.4.1.2)
needs to be distinguishable from the All-0 regular fragment, even in
the presence of padding.

### 8.4.1.2.  All-0 empty fragment

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

```
|----- Fragment Header -----|
        |-- T --|1|-- N --|
+-- ... --+- ... -+-+- ... -+~~~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  |W|  0..0 | padding (as needed)     (no payload)
+-- ... --+- ... -+-+- ... -+~~~~~~~~~~~~~~~~~~~~
```

Figure 15: All-0 empty fragment detailed format

The size of the All-0 fragment header is generally not a multiple of
the L2 Word size.  Therefore, an All-0 empty fragment generally needs
padding bits.  The padding bits are always less than an L2 Word.

Since an All-0 payload MUST be at least the size of an L2 Word, a
receiver can distinguish an All-0 empty fragment from a regular All-0
fragment, even in the presence of padding.

## 8.4.2. All-1 fragment

In the No-ACK mode, the last SCHC Fragment of a SCHC Packet SHALL
contain a SCHC Fragment header that conforms to the detailed format
shown in Figure 16.

```
|---------- Fragment Header ----------|
           |-- T --|-N=1-|
+---- ... ---+- ... -+-----+-- ... --+---...---+~~~~~~~~~~~~~~~~~~~~
|   Rule ID | DTag  | 1 |   MIC   | payload | padding (as needed)
+---- ... ---+- ... -+-----+-- ... --+---...---+~~~~~~~~~~~~~~~~~~~~
```

Figure 16: All-1 Fragment Detailed Format for the Last Fragment, No-
ACK mode

In ACK-Always or ACK-on-Error mode, the last fragment of a SCHC
Packet SHALL contain a SCHC Fragment header that conforms to the
detailed format shown in Figure 17.

```
|---------- Fragment Header ----------|
         |-- T --|1|-- N --|
+-- ... --+- ... -+-+- ... -+-- ... --+---...---+~~~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  |W| 11..1 |   MIC   | payload | padding (as needed)
+-- ... --+- ... -+-+- ... -+-- ... --+---...---+~~~~~~~~~~~~~~~~~~~~
                  (FCN)
```

Figure 17: All-1 Fragment Detailed Format for the Last Fragment, ACK-
Always or ACK-on-Error

The total size of the All-1 SCHC Fragment header is generally not a
multiple of the L2 Word size.  The All-1 fragment being the last one
of the SCHC Packet, SCHC F/R cannot freely choose the payload size to
align the fragment to an L2 Word.  Therefore, padding bits are
generally appended to the All-1 fragment to make it a multiple of L2
Words in size.

The MIC MUST be computed on the payload and the padding bits.  The
rationale is that the SCHC Reassembler needs to check the correctness
of the reassembled SCHC packet but has no way of knowing where the
payload ends.  Indeed, the latter requires decompressing the SCHC
Packet.

An All-1 fragment payload MUST be at least the size of an L2 Word.
The rationale is that the All-1 empty fragment (see Section 8.4.2.1)
needs to be distinguishable from the All-1 fragment, even in the
presence of padding.  This may entail saving an L2 Word from the
previous fragment payload to make the payload of this All-1 fragment
big enough.

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

The length of the MIC MUST be at least an L2 Word size.  The
rationale is to be able to distinguish a Sender-Abort (see
Section 8.4.4) from an All-1 Fragment, even in the presence of
padding.

### 8.4.2.1.  All-1 empty fragment

The All-1 empty fragment format is an All-1 fragment format without a
payload (see Figure 18).  It is used by a fragment sender, in either
ACK-Always or ACK-on-Error, to request a retransmission of the SCHC
ACK for the All-1 window.

The size of the All-1 empty fragment header is generally not a
multiple of the L2 Word size.  Therefore, an All-1 empty fragment
generally needs padding bits.  The padding bits are always less than
an L2 Word.

Since an All-1 payload MUST be at least the size of an L2 Word, a
receiver can distinguish an All-1 empty fragment from a regular All-1
fragment, even in the presence of padding.

```
|---------- Fragment Header --------|
          |-- T --|1|-- N --|
+-- ... --+- ... -+-+- ... -+- ... -+~~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  |W|  1..1 |  MIC  | padding as needed (no payload)
+-- ... --+- ... -+-+- ... -+- ... -+~~~~~~~~~~~~~~~~~~~
```

       Figure 18: All-1 for Retries format, also called All-1 empty

### 8.4.3. SCHC ACK format

The format of a SCHC ACK that acknowledges a window that is not the
last one (denoted as All-0 window) is shown in Figure 19.

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

                Figure 19: 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.

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

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

          Figure 20: Format of a SCHC ACK for All-1 windows

The Rule ID and Dtag values in the SCHC ACK messages MUST be
identical to the ones used in the SCHC Fragments that are being
acknowledged.  This allows matching the SCHC ACK and the
corresponding SCHC Fragments.

The Bitmap carries information on the reception of each fragment of
the window as described in Section 8.2.

See [Appendix D](#) for a discussion on the size of the Bitmaps.

In order to reduce the SCK ACK size, the Bitmap that is actually
transmitted is shortened ("encoded") as explained in [Section 8.4.3.1](#).

## 8.4.3.1.  Bitmap Encoding

The SCHC ACK that is transmitted is truncated by applying the
following algorithm: the longest contiguous sequence of bits that
starts at an L2 Word boundary of the SCHC ACK, where the bits of that
sequence are all set to 1, are all part of the Bitmap and finish
exactly at the end of the Bitmap, if one such sequence exists, MUST
NOT be transmitted.  Because the SCHC Fragment sender knows the
actual Bitmap size, it can reconstruct the original Bitmap from the
shortened bitmap.

When shortening effectively takes place, the SCHC ACK is a multiple
of L2 Words, and padding MUST NOT be appended.  When shortening does
not happen, padding bits MUST be appended as needed to fill up the
last L2 Word.

Figure 21 shows an example where L2 Words are actually bytes and
where the original Bitmap contains 17 bits, the last 15 of which are
all set to 1.

```
|--  SCHC ACK Header --|--------      Bitmap    --------|
|  Rule ID  |  DTag  |W|1|0|1|1|1|1|1|1|1|1|1|1|1|1|1|1|1|
     next L2 Word boundary ->|  next L2 Word |  next L2 Word |
```

                   Figure 21: A non-encoded Bitmap

Figure 22 shows that the last 14 bits are not sent.

```
           |-- T --|1|
+---- ... --+- ... -+-+-+-+-+-+
|  Rule ID  |  DTag |W|1|0|1|
+---- ... --+- ... -+-+-+-+-+-+
    next L2 Word boundary ->|
```

                  Figure 22: Optimized Bitmap format

Figure 23 shows an example of a SCHC 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.

```
                       6 5 4 3 2 1 0 (*)
                 |-- T --|1|
        +-----------+-------+-+-+-+-+-+-+-+
        | Rule ID   | DTag  |W|1|0|1|1|0|1|1|        Bitmap before tx
        +-----------+-------+-+-+-+-+-+-+-+
        next L2 Word boundary ->|<-- L2 Word -->|
            (*)=(FCN values)


        +-----------+-------+-+-+-+-+-+-+-+~~~+
        | Rule ID   | DTag  |W|1|0|1|1|0|1|1|Pad|       Encoded Bitmap
        +-----------+-------+-+-+-+-+-+-+-+~~~+
        next L2 Word boundary ->|<-- L2 Word -->|
```

      Figure 23: Example of a Bitmap before transmission, and the
           transmitted one, for a window that is not the last one

   Figure 24 shows an example of a SCHC ACK with FCN ranging from 6 down
   to 0, where MIC check has failed but the Bitmap indicates that there
   is no missing SCHC Fragment.

```
   |- Fragmentation Header-|6 5 4 3 2 1 7 (*)
                 |-- T --|1|
   | Rule ID   | DTag  |W|0|1|1|1|1|1|1|1|        Bitmap before tx
     next L2 Word boundary ->|<-- L2 Word -->|
                            C
   +---- ... --+- ... -+-+-+-+
   | Rule ID   | DTag  |W|0|1|                     Encoded Bitmap
   +---- ... --+- ... -+-+-+-+
     next L2 Word boundary ->|
       (*) = (FCN values indicating the order)
```

     Figure 24: Example of the Bitmap in ACK-Always or ACK-on-Error for
                              the last window

## 8.4.4.  Abort formats

   When a SCHC Fragment sender needs to abort the on-going fragmented
   SCHC Packet transmission, it sends a Sender-Abort.  The Sender-Abort
   format (see Figure 25) is a variation of the All-1 fragment, with
   neither a MIC nor a payload.  All-1 fragments contain at least a MIC.
   The absence of the MIC indicates a Sender-Abort.

```
|--- Sender-Abort Header ---|
+--- ... ---+- ... -+-+-....+~~~~~~~~~~~~~~~~~~~~
|  Rule ID  | DTag  |W| FCN | padding (as needed)
+--- ... ---+- ... -+-+-....+~~~~~~~~~~~~~~~~~~~~
```

Figure 25: Sender-Abort format.  All FCN field bits in this format
are set to 1

The size of the Sender-Abort header is generally not a multiple of
the L2 Word size.  Therefore, a Sender-Abort generally needs padding
bits.

Since an All-1 fragment MIC MUST be at least the size of an L2 Word,
a receiver can distinguish a Sender-Abort from an All-1 fragment,
even in the presence of padding.

When a SCHC Fragment receiver needs to abort the on-going fragmented
SCHC Packet transmission, it transmits a Receiver-Abort.  The
Receiver-Abort format is a variation on the SCHC ACK format, creating
an exception in the encoded Bitmap algorithm.  As shown in Figure 26,
a Receiver-Abort is coded as a SCHC ACK message with a shortened
Bitmap set to 1 up to the first L2 Word boundary, followed by an
extra L2 Word full of 1's.  Such a message never occurs in a regular
acknowledgement and is detected as a Receiver-Abort.

The Rule ID and Dtag values in the Receive-Abort message MUST be
identical to the ones used in the fragments of the SCHC Packet the
transmission of which is being aborted.

A Receiver-Abort is aligned to L2 Words, by design.  Therefore,
padding MUST not be appended.

```
|- Receiver-Abort Header -|

+---- ... ----+-- ... --+-+-+-+-+-+-+-+-+-+-+-+-+
|   Rule ID   |  DTag   |W| 1..1|     1..1      |
+---- ... ----+-- ... --+-+-+-+-+-+-+-+-+-+-+-+-+
        next L2 Word boundary ->|<-- L2 Word -->|
```

Figure 26: Receiver-Abort format

Neither the Sender-Abort nor the Receiver-Abort messages are ever
acknowledged or retransmitted.

Use cases for the Sender-Abort and Receiver-Abort messages are
explained in Section 8.5 or Appendix C.

## 8.5. 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 reassembly buffer at the location determined
from the FCN, the arrival order of the SCHC Fragments, and the
fragment payload sizes.  In ACK-on-Error or ACK-Always, 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 SCHC Packet (All-1),
it checks for the integrity of the reassembled SCHC Packet, based on
the MIC received.  In No-ACK, if the integrity check indicates that
the reassembled SCHC Packet does not match the original SCHC Packet
(prior to fragmentation), the reassembled SCHC Packet MUST be
discarded.  In ACK-on-Error or ACK-Always, a MIC check is also
performed by the fragment receiver after reception of each subsequent
SCHC Fragment retransmitted after the first MIC check.

Notice that the SCHC ACK for the All-1 window carries one more bit
(the C bit) compared to the SCHC ACKs for the previous windows.  See
Appendix D for a discussion on various options to deal with this
"bump" in the SCHC ACK.

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.

### 8.5.1.  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).

### 8.5.2.  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, if there are more
SCHC Fragments remaining to be sent, the sender transmits the last
SCHC Fragment of this window using the All-0 fragment format.  It
then starts the Retransmission Timer and waits for a SCHC ACK.
Otherwise, if FCN reaches 0 and the sender transmits 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 SCHC ACK to know if transmission errors have
occurred.

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 SCHC
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 a SCHC
ACK with an encoded Bitmap reporting whether any SCHC fragments have
been lost or not.  When the sender receives a SCHC ACK, it checks the
W bit carried by the SCHC ACK.  Any SCHC ACK carrying an unexpected W
bit value is discarded.  If the W bit value of the received SCHC ACK
is correct, the sender analyzes the rest of the SCHC 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 fragmented SCHC
Packet transmission is finished.

However, if one or more SCHC Fragments have not been received as per
the SCHC 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 a SCHC ACK.  Upon
receipt of the SCHC 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 fragmented SCHC
Packet transmission by sending a Sender-Abort message and releases
any resources for transmission of the packet.  The sender also aborts
an on-going fragmented SCHC 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.  When the Inactivity Timer expires, the transmission is
aborted by the receiver sending a Receiver-Abort message.

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 SCHC 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 a SCHC 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 by the receiver 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 SCHC ACKs sent by the receiver are
lost.  If the MIC is incorrect, some SCHC Fragments have been lost.
The receiver sends the SCHC ACK regardless of successful fragmented
SCHC 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 fragmented SCHC Packet transmission by transmitting a
the Receiver-Abort format.  The receiver also aborts upon Inactivity
Timer expiration by sending a Receiver-Abort message.

If the sender receives a SCK ACK with a Bitmap containing a bit set
for a SCHC Fragment that it has not sent during the transmission
phase of this window, it MUST abort the whole fragmentation and
transmission of this SCHC Packet.

### 8.5.3.  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 SCHC 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.  Except
for the last window where a SCHC ACK MUST be sent to finish the
transmission.

In ACK-on-Error, the Retransmission Timer expiration is considered as
a positive acknowledgement for all windows but the last one.  This
timer is set after sending an All-0 or an All-1 fragment.  For an
All-0 fragment, on timer expiration, the sender resumes operation and
sends the SCHC Fragments of the next window.

If the sender receives a SCHC ACK, it checks the window value.  SCHC
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 a Sender-Abort message and releases all resources for the on-
going fragmented SCHC Packet transmission.  When the retransmission
of the missing SCHC Fragments is finished, the sender starts
listening for a SCHC 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 a SCHC 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 SCHC ACK for the last
window.  If Attempts reaches MAX_ACK_REQUESTS, the sender aborts the
on-going fragmented SCHC Packet transmission by transmitting the
Sender-Abort fragment.

At the end of any window, if the sender receives a SCK ACK with a
Bitmap containing a bit set for a SCHC Fragment that it has not sent
during the transmission phase of that window, it MUST abort the whole
fragmentation and transmission of this SCHC Packet.

Unlike the sender, the receiver for ACK-on-Error has a larger amount
of differences compared with ACK-Always.  First, a SCHC ACK is not
sent unless there is a lost SCHC Fragment or an unexpected behavior.
With the exception of the last window, where a SCHC 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 a 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 fragmented SCHC 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 a Fragment discard it.  If all SCHC Fragments of a
window (that is not the last one) have been received, the receiver
does not send a SCHC 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 fragmented SCHC 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 by
transmitting the Receiver-Abort message.

If, after receiving an All-0 fragment the receiver missed some SCHC
Fragments, the receiver uses a SCHC 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 SCHC 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.

## 8.6.  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.

## 8.7.  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 fragmented SCHC Packet, the SCHC Fragment
receiver MAY perform an uplink transmission as soon as possible after
reception of a SCHC Fragment that is not the last one.  Such uplink
transmission MAY be triggered by the L2 (e.g. an L2 ACK sent in
response to a SCHC Fragment encapsulated in a L2 frame that requires
an L2 ACK) or it MAY be triggered from an upper layer.

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

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

The default initial value for the Inactivity timer, as well as the
maximum number of retries for a specific SCHC ACK, denoted
MAX_ACK_RETRIES, are not defined in this document, and need to be

defined in 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 a SCHC ACK is
received before expiration of this timer, the SCHC Fragment sender
retransmits any lost SCHC Fragments reported by the SCHC ACK, or if
the SCHC ACK confirms successful reception of all SCHC Fragments of
the last window, the transmission of the fragmented SCHC Packet is
considered complete.  If the timer expires, and no SCHC ACK has been
received since the start of the timer, the SCHC Fragment sender
assumes that the All-1 fragment has been successfully received (and
possibly, the last SCHC ACK has been lost: this mechanism assumes
that the retransmission timer for the All-1 fragment is long enough
to allow several SCHC 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).

## 9.  Padding management

SCHC C/D and SCHC F/R operate on bits, not bytes.  SCHC itself does
not have any alignment prerequisite.  If the Layer 2 below SCHC
constrains the L2 Data Unit to align to some boundary, called L2
Words (for example, bytes), SCHC will meet that constraint and
produce messages with the correct alignement.  This may entail adding
extra bits (called padding bits).

When padding occurs, the number of appended bits is strictly less
than the L2 Word size.

Padding happens at most once for each Packet going through the full
SCHC chain, i.e. Compression and (optionally) SCHC Fragmentation (see
Figure 2).  If a SCHC Packet is sent unfragmented (see Figure 27), it
is padded as needed.  If a SCHC Packet is fragmented, only the last
fragment is padded as needed.

```
A packet (e.g. an IPv6 packet)
        |                                        ^ (padding bits
        v                                        |       dropped)
+------------------+                  +--------------------+
| SCHC Compression |                  | SCHC Decompression |
+------------------+                  +--------------------+
        |                                        ^
        |    If no fragmentation                 |
        +---- SCHC Packet + padding as needed ----->|
        |                                        | (MIC checked
        v                                        |  and removed)
+-------------------+                  +-----------------+
| SCHC Fragmentation |                 | SCHC Reassembly |
+------------------+                   +-----------------+
     |        ^                           |        ^
     |        |                           |        |
     |        +------------- SCHC ACK ------------+        |
     |                                            |
     +--------------- SCHC Fragments -------------------+
     +--- last SCHC Frag with MIC + padding as needed ---+

        SENDER                              RECEIVER
```
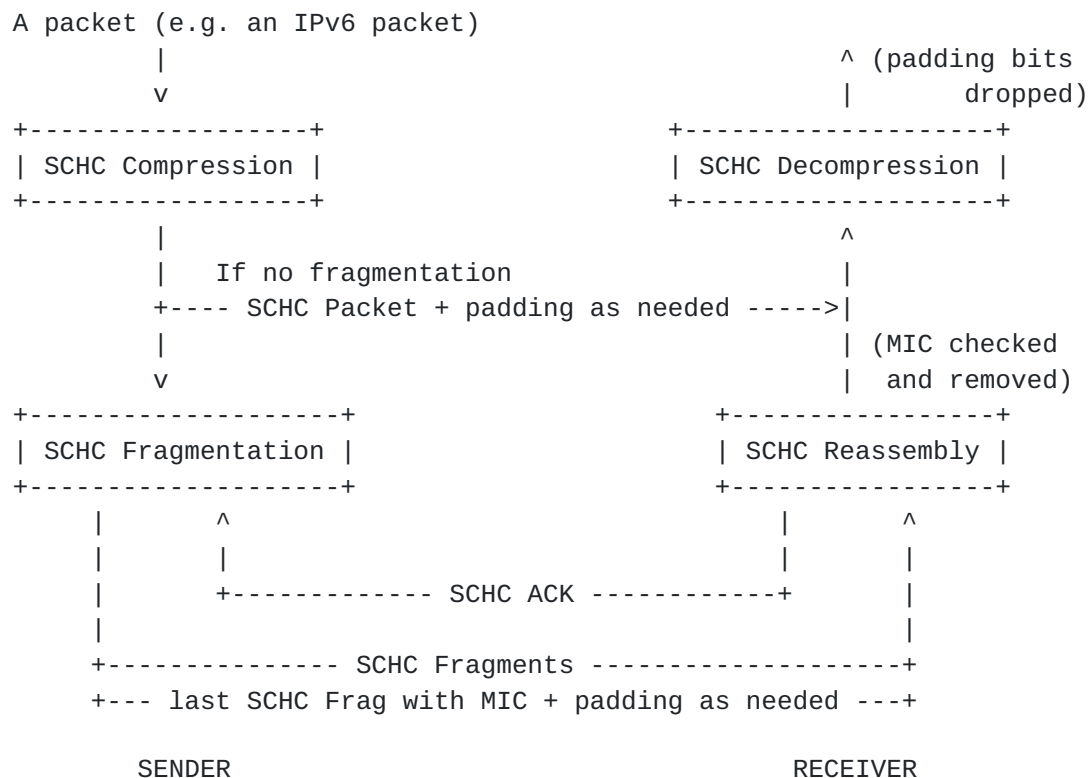
Figure 27: SCHC operations, including padding as needed

Each technology-specific document MUST specify the size of the L2
Word.  The L2 Word might actually be a single bit, in which case at
most zero bits of padding will be appended to any message, i.e. no
padding will take place at all.

## 10.  SCHC Compression for IPv6 and UDP headers

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

### 10.1.  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".

### 10.2.  IPv6 Traffic class field

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

Otherwise, 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).

## 10.3.  Flow label field

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

Otherwise, two possibilities can be considered:

o  One possibility is to not compress the field and send the original
   value.  In the Rule, TV is not set to any particular value, MO is
   set to "ignore" and CDA is set to "value-sent".

o  If some upper bits in the field are constant and known, a better
   option is to only send the LSBs.  In the Rule, TV is set to a
   value with the stable known upper part, MO is set to MSB(x) and
   CDA to LSB(y).

## 10.4.  Payload Length field

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

## 10.5.  Next Header field

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

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

## 10.6.  Hop Limit field

The field behavior for this field is different for Uplink 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".

## 10.7.  IPv6 addresses fields

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

## 10.7.1.  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 29

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

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

   If the DEV or APP IID are based on an LPWAN address, then the IID can
   be reconstructed with information coming from the LPWAN header.  In
   that case, the TV is not set, the MO is set to "ignore" and the CDA
   is set to "DevIID" or "AppIID".  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".

**10.8**.  **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.

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

   To allow for a single Rule being used for both directions, the UDP
   port values are identified by their role (DEV or APP) and not by
   their position in the 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".

## 10.10.  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".

## 10.11.  UDP Checksum field

The UDP checksum operation is mandatory with IPv6 [RFC8200] for most
packets but recognizes that there are exceptions to that default
behavior.

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

6LoWPAN Header Compression [RFC6282] also authorizes to send UDP
datagram that are deprived of the checksum protection when an upper
layer guarantees the integrity of the UDP payload and pseudo-header
all the way between the compressor that elides the UDP checksum and
the decompressor that computes again it.  A specific example of this
is when a Message Integrity Check (MIC) protects the compressed
message all along that path with a strength that is identical or
better to the UDP checksum.

In a similar fashion, this specification allows a SCHC compressor to
elide the UDP checks when another layer guarantees an identical or
better integrity protection for the UDP payload and the pseudo-

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

In particular, when SCHC fragmentation is used, a fragmentation MIC
of 2 bytes or more provides equal or better protection than the UDP
checksum; in that case, if the compressor is collocated with the
fragmentation point and the decompressor is collocated with the
packet reassembly point, then compressor MAY elide the UDP checksum.
Whether and when the UDP Checksum is elided is to be specified in the
technology-specific documents.

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

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

## 11.  IANA Considerations

This document has no request to IANA.

## 12.  Security considerations

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

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).

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

This subsection describes potential attacks to LPWAN SCHC F/R 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 fragmented SCHC Packets
will be dropped while the reassembly buffer is occupied during the
reassembly timeout.  Once that timeout expires, the attacker can
repeat the same procedure, and iterate, thus creating a denial of
service attack.  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 split 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 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.

## 13.  Acknowledgements

## 14. References

### 14.1. Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119,
            DOI 10.17487/RFC2119, March 1997,
            <https://www.rfc-editor.org/info/rfc2119>.

[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>.

[RFC8174]   Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
            2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
            May 2017, <https://www.rfc-editor.org/info/rfc8174>.

### 14.2. Informative References

[RFC3385]   Sheinwald, D., Satran, J., Thaler, P., and V. Cavanna,
            "Internet Protocol Small Computer System Interface (iSCSI)
            Cyclic Redundancy Check (CRC)/Checksum Considerations",
            RFC 3385, DOI 10.17487/RFC3385, September 2002,
            <https://www.rfc-editor.org/info/rfc3385>.

[RFC4944]   Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler,
            "Transmission of IPv6 Packets over IEEE 802.15.4
            Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007,
            <https://www.rfc-editor.org/info/rfc4944>.

[RFC5795]   Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust
            Header Compression (ROHC) Framework", RFC 5795,
            DOI 10.17487/RFC5795, March 2010,
            <https://www.rfc-editor.org/info/rfc5795>.

[RFC6282]   Hui, J., Ed. and P. Thubert, "Compression Format for IPv6
            Datagrams over IEEE 802.15.4-Based Networks", RFC 6282,
            DOI 10.17487/RFC6282, September 2011,
            <https://www.rfc-editor.org/info/rfc6282>.

[RFC6936]   Fairhurst, G. and M. Westerlund, "Applicability Statement
            for the Use of IPv6 UDP Datagrams with Zero Checksums",
            RFC 6936, DOI 10.17487/RFC6936, April 2013,
            <https://www.rfc-editor.org/info/rfc6936>.

   [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>.

   [RFC8200]   Deering, S. and R. Hinden, "Internet Protocol, Version 6
               (IPv6) Specification", STD 86, RFC 8200,
               DOI 10.17487/RFC8200, July 2017,
               <https://www.rfc-editor.org/info/rfc8200>.

   [RFC8376]   Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN)
               Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018,
               <https://www.rfc-editor.org/info/rfc8376>.

## Appendix A.  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 28 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
```
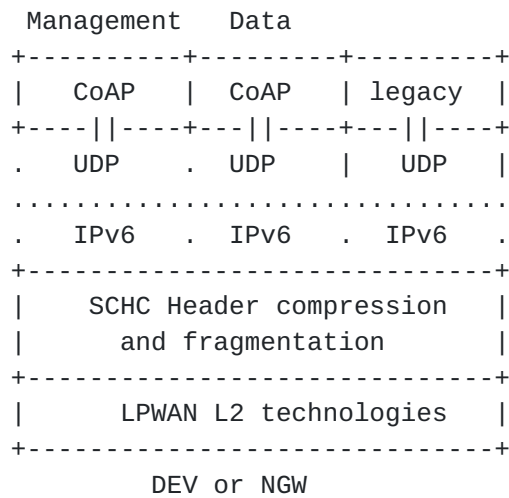
Figure 28: Simplified Protocol Stack for LP-WAN

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 Opera. | Comp Decomp Action | Sent [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 Opera. | Action Action | Sent [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, match- |mapping-sent|| [1] |
|                |  |  |  | |fe80::/64] mapping|          ||      |
|IPv6 DevIID     |64|1 |Bi|         | ignore| DevIID    ||      |
|IPv6 APPprefix  |64|1 |Bi|[beta/64,| match- |mapping-sent|| [2] |
|                |  |  |  | |alpha/64,| mapping|          ||      |
|                |  |  |  | |fe80::64]|        |          ||      |
|IPv6 AppIID     |64|1 |Bi|::1000   | equal | not-sent  ||      |
+===============+==+==+==+=========+========+============++======+
|UDP DEVport     |16|1 |Bi|5683     | equal | not-sent  ||      |
|UDP APPport     |16|1 |Bi|5683     | equal | not-sent  ||      |
|UDP Length      |16|1 |Bi|         | ignore| 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] |
|UDP APPport     |16|1 |Bi|8720     | MSB(12)| LSB        || [4] |
|UDP Length      |16|1 |Bi|         | ignore | comp-length||     |
|UDP checksum    |16|1 |Bi|         | ignore | comp-chk   ||      |
+===============+==+==+==+=========+========+============++======+
```

Figure 29: Context Rules

All the fields described in the three Rules depicted on Figure 29 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 30 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 30: 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 31 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 31: Transmission in ACK-on-Error mode of an IPv6 packet
carried by 11 fragments, with MAX_WIND_FCN=6 and no loss.

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

Figure 33 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 33: Transmission in ACK-Always mode of an IPv6 packet carried
      by 11 fragments, with MAX_WIND_FCN=6 and no lost fragment.

Figure 34 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 34: Transmission in ACK-Always mode of an IPv6 packet carried
    by 11 fragments, with MAX_WIND_FCN=6 and three lost fragments.

Figure 35 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 35: 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 36 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 36: 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 37 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 37: 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 38 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 38: 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 39: 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 40: 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  |
                       | |   | |  +--+Attemp++          |
      MIC_bit==1 &       | |   | +------------------------+
      Recv_window==window &  | |   |  all missing frags sent
              no more frag| |   | ~~~~~~~~~~~~~~~~~~~~~~
       ~~~~~~~~~~~~~~~~~~~~| |   |  Set Retrans_Timer
           Stop Retrans_Timer| |   |
        +=============+      | |   |
        |    END    +<--------+ |   |
        +=============+          |   | Attemp > MAX_ACK_REQUESTS
             All-1 Window & |    | ~~~~~~~~~~~~~~~~~~
               MIC_bit ==0 & |    v Send Abort
        Lcl_Bitmap==recv_Bitmap | +=+==========+
             ~~~~~~~~~~~~ +>|    ERROR    |
               Send Abort   +=============+
```

                 Figure 41: 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++
```

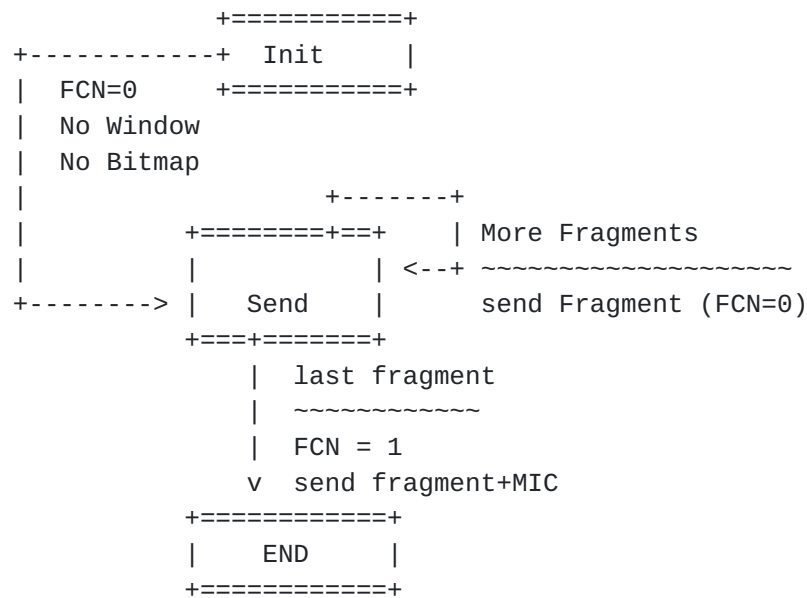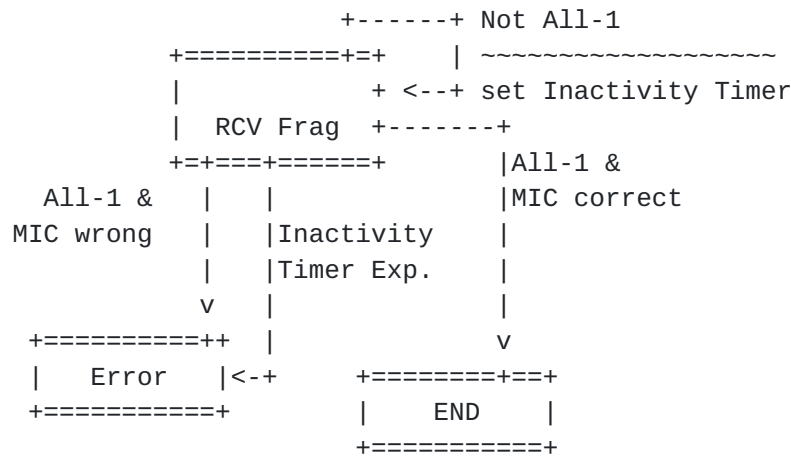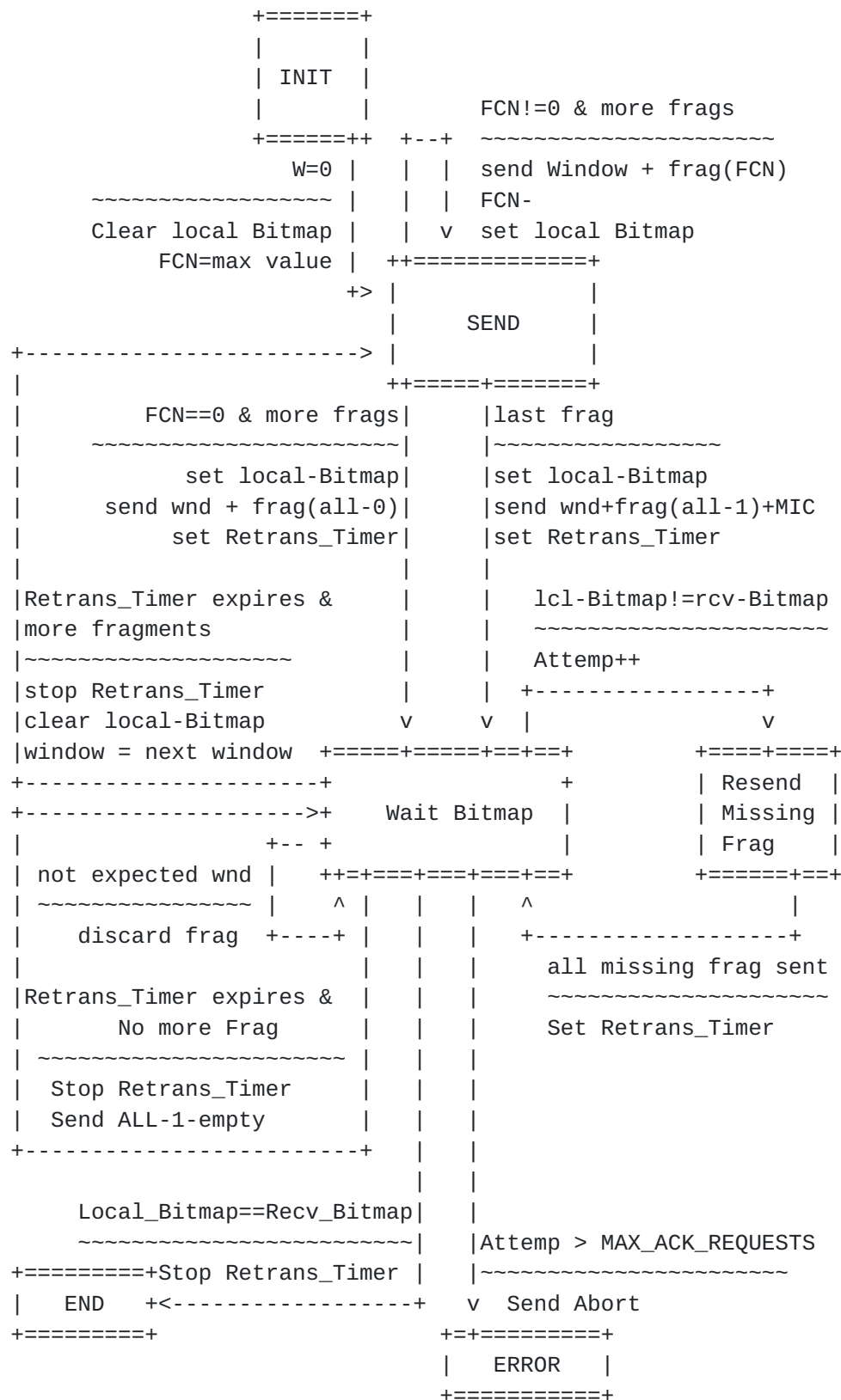                Figure 42: Receiver State Machine for the ACK-Always Mode

```
                        +=======+
                        |       |
                        | 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 43: 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 44: Receiver State Machine for the ACK-on-Error Mode

**Appendix D**.  **SCHC Parameters - Ticket #15**

This section gives the list of parameters that need to be defined in
the technology-specific documents.

o  Define the most common uses case and how SCHC may be deployed.

o  LPWAN Architecture.  Explain the SCHC entities (Compression and
   Fragmentation), how/where they are represented in the
   corresponding technology architecture.  If applicable, explain the
   various potential channel conditions for the technology and the
   corresponding recommended use of C/D and F/R.

o  L2 fragmentation decision

o  Technology developers must evaluate that L2 has strong enough
   integrity checking to match SCHC's assumption.

o  Rule ID numbering system, number of Rules

o  Size of the Rule IDs

o  The way the Rule ID is sent (L2 or L3) and how (describe)

o  Fragmentation delivery reliability mode used in which cases (e.g.
   based on link channel condition)

o  Define the number of bits for FCN (N) and DTag (T)

o  in particular, is interleaved packet transmission supported and to
   what extent

o  The MIC algorithm to be used and the size, if different from the
   default CRC32

o  Retransmission Timer duration

o  Inactivity Timer duration

o  Define MAX_ACK_REQUEST (number of attempts)

o  Padding: size of the L2 Word (for most technologies, a byte; for
   some technologies, a bit).  Value of the padding bits (1 or 0).
   The value of the padding bits needs to be specified because the
   padding bits are included in the MIC calculation.

   o  Take into account that the length of Rule ID + N + T + W when
      possible is good to have a multiple of 8 bits to complete a byte
      and avoid padding

   o  In the ACK format to have a length for Rule ID + T + W bit into a
      complete number of byte to do optimization more easily

   o  The technology documents will describe if Rule ID is constrained
      by any alignment

   o  When fragmenting in ACK-on-Error or ACK-Always mode, it is
      expected that the last window (called All-1 window) will not be
      fully utilised, i.e. there won't be fragments with all FCN values
      from MAX_WIND_FCN downto 1 and finally All-1.  It is worth noting
      that this document does not mandate that other windows (called
      All-0 windows) are fully utilised either.  This document purposely
      does not specify that All-1 windows use Bitmaps with the same
      number of bits as All-0 windows do.  By default, Bitmaps for All-0
      and All-1 windows are of the same size MAX_WIND_FCN + 1.  But a
      technology-specific document MAY revert that decision.  The
      rationale for reverting the decision could be the following: Note
      that the SCHC ACK sent as a response to an All-1 fragment includes
      a C bit that SCHC ACK for other windows don't have.  Therefore,
      the SCHC ACK for the All-1 window is one bit bigger.  An L2
      technology with a severely constrained payload size might decide
      that this "bump" in the SCHC ACK for the last fragment is a bad
      resource usage.  It could thus mandate that the All-1 window is
      not allowed to use the FCN value 1 and that the All-1 SCHC ACK
      Bitmap size is reduced by 1 bit.  This provides room for the C bit
      without creating a bump in the SCHC ACK.

   And the following parameters need to be addressed in another document
   but not forcely in the technology-specific one:

   o  The way the contexts are provisioning

   o  The way the Rules as generated

## Appendix E.  Note

Authors' Addresses

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

    Email: ana@ackl.io


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

    Email: Laurent.Toutain@imt-atlantique.fr


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

    Email: carlesgo@entel.upc.edu


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

    Email: dominique.barthel@orange.com