Internet Engineering Task Force (IETF)
Request for Comments: 8724
Category: Standards Track
ISSN: 2070-1721

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

April 2020

SCHC: Generic Framework for Static Context Header Compression and Fragmentation

Abstract

This document defines the Static Context Header Compression and fragmentation (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. SCHC has been designed with Low-Power Wide Area Networks (LPWANs) in mind.

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

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

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

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8724.

Table of Contents

1. Introduction

This document defines the Static Context Header Compression and fragmentation (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. SCHC has been designed with Low-Power Wide Area Networks (LPWANs) in mind.

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

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

* The network topology is star-oriented, which means that all packets between the same source-destination pair follow the same path. For the needs of this document, the architecture can simply be described as Devices (Dev) exchanging information with LPWAN Application Servers (Apps) through a Network Gateway (NGW).

* Because devices embed built-in applications, the traffic flows to be compressed are known in advance. Indeed, new applications are less frequently installed in an LPWAN device than they are in a general-purpose computer or smartphone.

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

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

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

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

2. Requirements Notation

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

3. LPWAN Architecture

LPWAN architectures are similar among them, but each LPWAN technology names architecture elements differently. In this document, we use terminology from [RFC8376], which identifies the following entities in a typical LPWAN (see Figure 1):

* 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.

* The Radio Gateway (RGW) is the endpoint of the constrained link.

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

* The Application Server (App) is the endpoint of the application-level protocol on the Internet side.

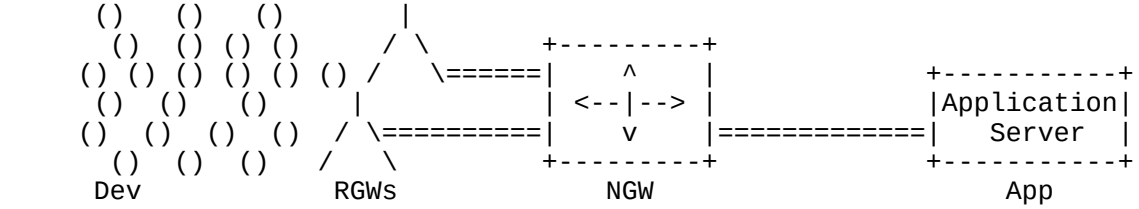


Figure 1: LPWAN Architecture (Simplified from That Shown in RFC 8376)

4. Terminology

This section defines terminology and abbreviations used in this document. It extends the terminology of [RFC8376].

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

App: LPWAN Application Server, as defined by [RFC8376]. It runs an application sending/receiving packets to/from the Dev.

AppIID: Application Interface Identifier. The IID that identifies the App interface.

Compression Residue: The bits that remain to be sent (beyond the RuleID itself) after applying the SCHC compression.

Context: A set of Rules used to compress/decompress headers, or to fragment/reassemble a packet.

Dev: Device, as defined by [RFC8376].

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

Downlink: From the App to the Dev.

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

L2: Layer 2. The immediate lower layer that SCHC interfaces with, for example an underlying LPWAN technology. It does not necessarily correspond to the OSI model definition of Layer 2.

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.

Padding: Extra bits that may be appended by SCHC to a data unit that it passes down to L2 for transmission. SCHC itself operates on bits, not bytes, and does not have any alignment prerequisite. See Section 9.

Profile: SCHC offers variations in the way it is operated, with a number of parameters listed in Appendix D. A Profile indicates a particular setting of all these parameters. Both ends of a SCHC communication must be provisioned with the same Profile information and with the same set of Rules before the communication starts, so that there is no ambiguity in how they expect to communicate.

Rule: Part of the Context that describes how a packet is compressed/decompressed or fragmented/reassembled.

RuleID: Rule Identifier. An identifier for a Rule.

SCHC: Static Context Header Compression and fragmentation (SCHC), a generic framework.

SCHC C/D: SCHC Compressor/Decompressor, or SCHC Compression/ Decompression. The SCHC entity or mechanism used on both sides, at the Dev and at the network, to achieve compression/decompression of headers.

SCHC F/R: SCHC Fragmenter/Reassembler or SCHC Fragmentation/ Reassembly. The SCHC entity or mechanism used on both sides, at the Dev and at the network, to achieve fragmentation/reassembly of SCHC Packets.

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 RuleID is used to indicate that the packet header has not been compressed). See Section 7 for more details.

Uplink: From the Dev to the App.

Additional terminology for the optional SCHC F/R is found in Section 8.2.

Additional terminology for SCHC C/D is found in Section 7.1.

5. SCHC Overview

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

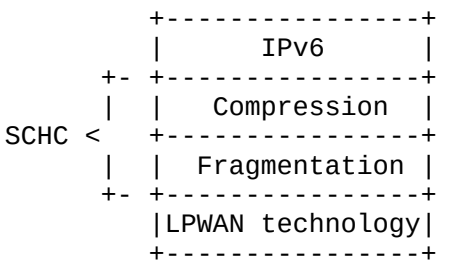
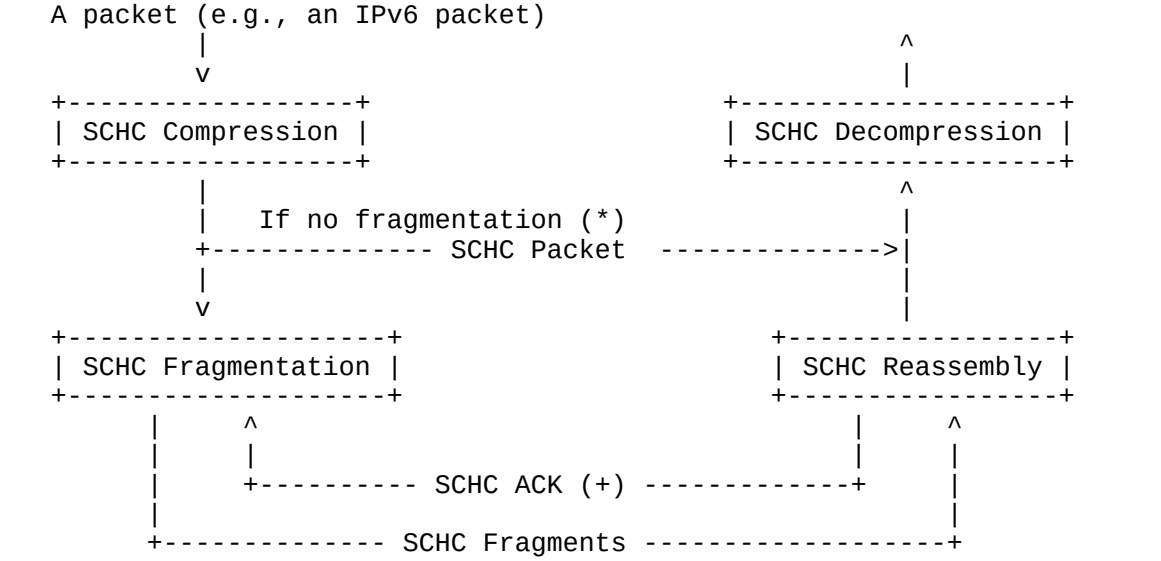


Figure 2: Example of Protocol Stack Comprising IPv6, SCHC, and an LPWAN Technology

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



*: the decision not to use SCHC fragmentation is left to each Profile
+: optional, depends on Fragmentation mode

Figure 3: SCHC Operations at the Sender and the Receiver

5.1. SCHC Packet Format

The SCHC Packet is composed of the Compressed Header followed by the payload from the original packet (see Figure 4). The Compressed

Header itself is composed of the RuleID and a Compression Residue, which is the output of compressing the packet header with the Rule identified by that RuleID (see Section 7). The Compression Residue may be empty. Both the RuleID and the Compression Residue potentially have a variable size, and are not necessarily a multiple of bytes in size.

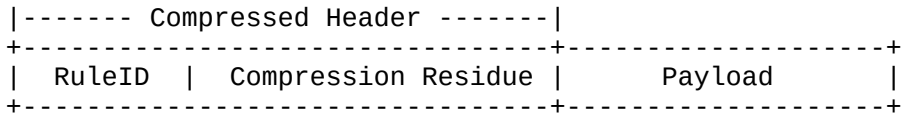


Figure 4: SCHC Packet

5.2. Functional Mapping

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

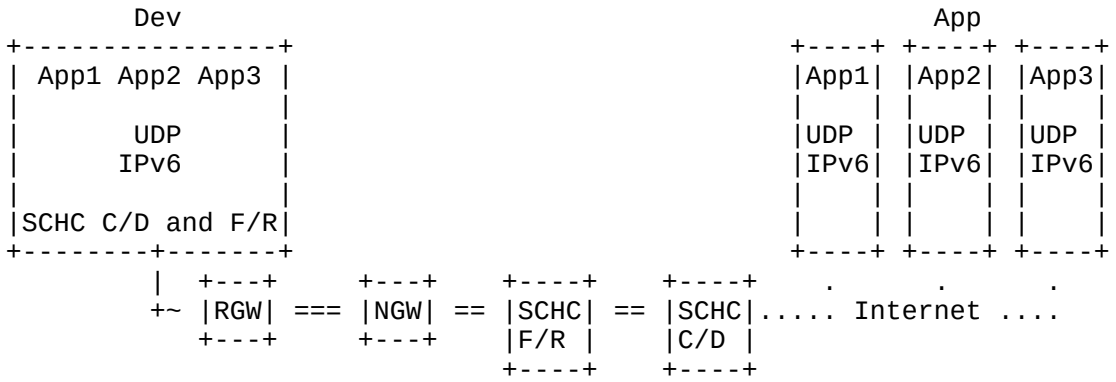


Figure 5: Architectural Mapping

SCHC C/D and SCHC F/R are located on both sides of the LPWAN transmission, hereafter called the "Dev side" and the "Network Infrastructure side".

The operation in the Uplink direction is as follows. The Device application uses IPv6 or IPv6/UDP protocols. Before sending the packets, the Dev compresses their headers using SCHC C/D; if the SCHC Packet resulting from the compression needs to be fragmented by SCHC, SCHC F/R is performed (see Section 8). The resulting SCHC Fragments are sent to an LPWAN Radio Gateway (RGW), which forwards them to a Network Gateway (NGW). The NGW sends the data to a SCHC F/R for reassembly (if needed) and then to the SCHC C/D for decompression. After decompression, the packet can be sent over the Internet to one or several Apps.

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

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

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

6. RuleID

RuleIDs identify the Rules used for compression/decompression or for fragmentation/reassembly.

The scope of the RuleID of a compression/decompression Rule is the link between the SCHC C/D in a given Dev and the corresponding SCHC C/D in the Network Infrastructure side. The scope of the RuleID of a fragmentation/reassembly Rule is the link between the SCHC F/R in a given Dev and the corresponding SCHC F/R in the Network Infrastructure side. If such a link is bidirectional, the scope includes both directions.

The RuleIDs are therefore specific to the Context related to one Dev. Hence, multiple Dev instances, which refer to different Contexts, MAY reuse the same RuleID for different Rules. On the Network Infrastructure side, in order to identify the correct Rule to be applied to Uplink traffic, the SCHC C/D or SCHC F/R needs to associate the RuleID with the Dev identifier. Similarly, for Downlink traffic, the SCHC C/D or SCHC F/R on the Network Infrastructure side first needs to identify the destination Dev before looking for the appropriate Rule (and associated RuleID) in the Context of that Dev.

Inside their scopes, Rules for compression/decompression and Rules for fragmentation/reassembly share the same RuleID space.

The size of the RuleIDs 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 other things. It is defined in Profiles.

The RuleIDs are used:

* For SCHC C/D, to identify the Rule that is used to compress a packet header.

  - At least one RuleID MUST be allocated to tagging packets for which SCHC compression was not possible (i.e., no matching compression Rule was found).

* In SCHC F/R, to identify the specific mode and settings of fragmentation/reassembly for one direction of data traffic (Uplink or Downlink).

  - When SCHC F/R is used for both communication directions, at least two RuleID values are needed for fragmentation/reassembly: one per direction of data traffic. This is because fragmentation/reassembly may entail control messages flowing in the reverse direction compared to data traffic.

7. Compression/Decompression

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

7.1. SCHC C/D Rules

The main idea of the SCHC compression scheme is to transmit the RuleID to the other end instead of sending known field values. This

RuleID identifies a Rule that matches the original packet values. Hence, when a value is known by both ends, it is only necessary to send the corresponding RuleID over the LPWAN. The manner by which Rules are generated is out of the scope of this document. The Rules MAY be changed at run-time, but the mechanism is out of scope of this document.

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

Each Rule itself contains a list of Field Descriptors 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).

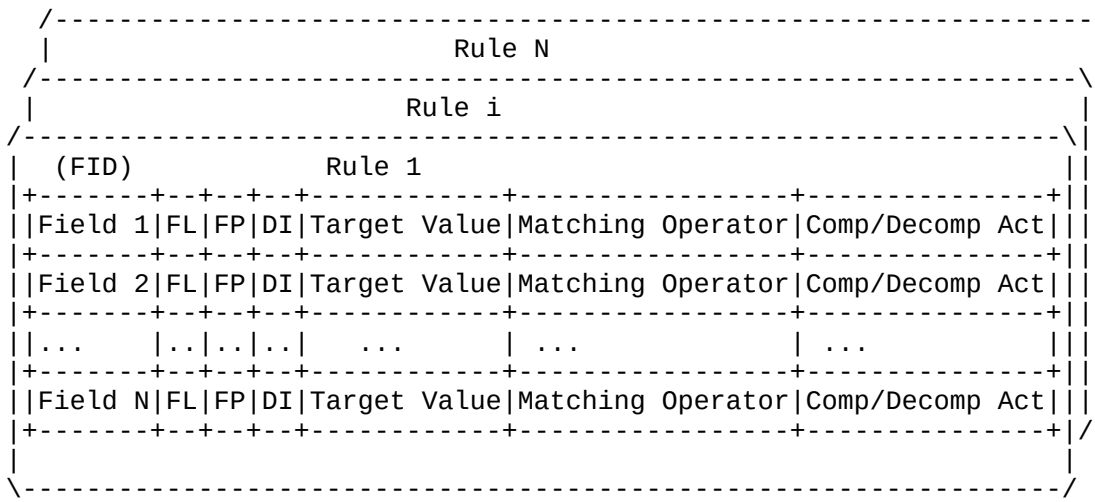


Figure 6: A SCHC C/D Context

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

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

* Field Identifier (FID) designates a protocol and field (e.g., UDP Destination Port), unambiguously among all protocols that a SCHC compressor processes. In the presence of protocol nesting, the Field ID also identifies the nesting.

* Field Length (FL) represents the length of the original field. It can be either a fixed value (in bits) if the length is known when the Rule is created or a type if the length is variable. The length of a header field is defined by its own protocol specification (e.g., IPv6 or UDP). If the length is variable, the type defines the process to compute the length and its unit (bits, bytes...).

* Field Position (FP): most often, a field only occurs once in a packet header. However, some fields may occur multiple times. An example is the uri-path of CoAP. FP indicates which occurrence

this Field Descriptor applies to. The default value is 1. The value 1 designates the first occurrence. The value 0 is special. It means "don't care", see Section 7.2.

* A Direction Indicator (DI) indicates the packet direction(s) this Field Descriptor applies to. It allows for asymmetric processing, using the same Rule. Three values are possible:

  Up: this Field Descriptor is only applicable to packets traveling Uplink.

  Dw: this Field Descriptor is only applicable to packets traveling Downlink.

  Bi: this Field Descriptor is applicable to packets traveling Uplink or Downlink.

* Target Value (TV) is the value used to match against the packet header field. The Target Value can be a scalar value of any type (integer, strings, etc.) or a more complex structure (array, list, etc.). The types and representations are out of scope for this document.

* Matching Operator (MO) is the operator used to match the field value and the Target Value. The Matching Operator may require some parameters. The set of MOs defined in this document can be found in Section 7.3.

* Compression/Decompression Action (CDA) describes the pair of actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field. Some CDAs might use parameter values for their operation. The set of CDAs defined in this document can be found in Section 7.4.

7.2. Packet Processing

The compression/decompression process follows several phases:

Compression Rule selection: the general idea is to browse the Rule set to find a Rule that has a matching Field Descriptor (given the DI and FP) for all and only those header fields that appear in the packet being compressed. The detailed algorithm is the following:

* The first step is to check the FIDs. If any header field of the packet being examined cannot be matched with a Field Descriptor with the correct FID, the Rule MUST be disregarded. If any Field Descriptor in the Rule has a FID that cannot be matched to one of the header fields of the packet being examined, the Rule MUST be disregarded.

* The next step is to match the Field Descriptors by their direction, using the DI. If any field of the packet header cannot be matched with a Field Descriptor with the correct FID and DI, the Rule MUST be disregarded.

* Then, the Field Descriptors are further selected according to FP. If any field of the packet header cannot be matched with a Field Descriptor with the correct FID, DI and FP, the Rule MUST be disregarded.

  The value 0 for FP means "don't care", i.e., the comparison of this Field Descriptor's FP with the position of the field of the packet header being compressed returns True, whatever that position. FP=0 can be useful to build compression Rules for protocol headers in which some fields order is irrelevant. An

example could be uri-queries in CoAP. Care needs to be exercised when writing Rules containing FP=0 values. Indeed, it may result in decompressed packets having fields ordered differently compared to the original packet.

* Once each header field has been associated with a Field Descriptor with matching FID, DI, and FP, each packet field's value is then compared to the corresponding TV stored in the Rule for that specific field, using the MO. If every field in the packet header satisfies the corresponding MOs of a Rule (i.e., all MO results are True), that Rule is valid for use to compress the header. Otherwise, the Rule MUST be disregarded.

  This specification does not prevent multiple Rules from matching the above steps and, therefore, being valid for use. Which Rule to use among multiple valid Rules is left to the implementation. As long as the same Rule set is installed at both ends, this degree of freedom does not constitute an interoperability issue.

* If no valid compression Rule is found, then the packet MUST be sent uncompressed using the RuleID dedicated to this purpose (see Section 6). The entire packet header is the Compression Residue (see Figure 4). Sending an uncompressed header is likely to require SCHC F/R.

Compression: if a valid Rule is found, each field of the header is compressed according to the CDAs of the Rule. The fields are compressed in the order that the Field Descriptors appear in the Rule. The compression of each field results in a residue, which may be empty. The Compression Residue for the packet header is the concatenation of the non-empty residues for each field of the header, in the order the Field Descriptors appear in the Rule. The order in which the Field Descriptors appear in the Rule is therefore semantically important.

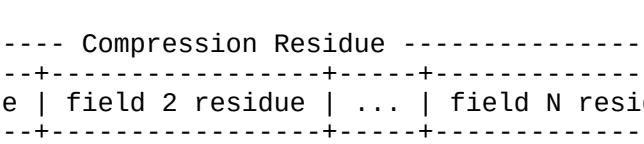


Figure 7: Compression Residue Structure

Sending: The RuleID is sent to the other end jointly with the Compression Residue (which could be empty) or the uncompressed header, and directly followed by the payload (see Figure 4). The way the RuleID is sent will be specified in the Profile and is out of the scope of the present document. For example, it could be included in an L2 header or sent as part of the L2 payload.

Decompression: when decompressing, on the Network Infrastructure side, the SCHC C/D needs to find the correct Rule based on the L2 address of the Dev. On the Dev side, only the RuleID is needed to identify the correct Rule since the Dev typically only holds Rules that apply to itself.

This Rule describes the compressed header format. From this, the decompressor determines the order of the residues, the fixed-size or variable-size nature of each residue (see Section 7.4.2), and the size of the fixed-size residues.

Therefore, from the received compressed header, it can retrieve all the residue values and associate them to the corresponding header fields.

For each field in the header, the receiver applies the CDA action

associated with that field in order to reconstruct the original header field value. The CDA application order can be different from the order in which the fields are listed in the Rule. In particular, Compute-* MUST be applied after the application of the CDAs of all the fields it computes on.

7.3. Matching Operators

MOs are functions used at the compression side of SCHC C/D. They are not typed and can be applied to integer, string or any other data type. The result of the operation can either be True or False. The following MOs are defined:

equal: The match result is True if the field value in the packet matches the TV.

ignore: No matching is attempted between the field value in the packet and the TV in the Rule. The result is always True.

MSB(x): A match is obtained if the most significant (leftmost) 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 x parameter must be a multiple of the FL unit. For example, x must be multiple of 8 if the unit of the variable length is bytes.

match-mapping: With match-mapping, TV is a list of values. Each value of the list is identified by an index. Compression is achieved by sending the index instead of the original header field value. This operator matches if the header field value is equal to one of the values in the target list.

7.4. Compression/Decompression Actions (CDA)

The CDA specifies the actions taken during the compression of header fields and the inverse action taken by the decompressor to restore the original value. The CDAs defined by this document are described in detail in Section 7.4.3 to Section 7.4.8. They are summarized in Table 1.

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

Table 1: Compression and Decompression Actions

The first column shows the action’s name. The second and third

columns show the compression and decompression behaviors for each action.

7.4.1. Processing Fixed-Length Fields

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

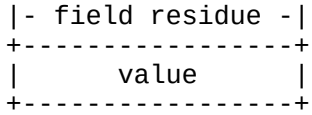


Figure 8: Fixed-Size Field Residue Structure

7.4.2. Processing Variable-Length Fields

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

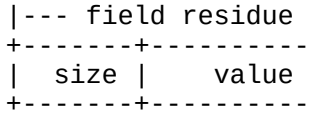


Figure 9: Variable-Size Field Residue Structure

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

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

* Sizes between 15 and 254 are encoded as 0b1111 followed by the 8-bit unsigned integer.

* Larger sizes are encoded as 0xfff followed by the 16-bit unsigned integer.

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

7.4.3. Not-Sent CDA

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

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

The decompressor restores the field value with the TV stored in the matched Rule identified by the received RuleID.

7.4.4. Value-Sent CDA

The value-sent action can be used when the field value is not known by both the Compressor and the Decompressor. The field is sent in its entirety, using the same bit order as in the original packet header.

If this action is performed on a variable-length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.4.2.

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

7.4.5. Mapping-Sent CDA

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

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

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

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

The first element in the list MUST be represented by index value 0, and successive elements in the list MUST have indices incremented by 1.

7.4.6. LSB CDA

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

The compressor sends the LSBs as the field residue value. The number of bits sent is the original header field length minus the length specified in the MSB(x) MO. The bits appear in the residue in the same bit order as in the original packet header.

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

If this action is performed on a variable-length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.4.2.

7.4.7. DevIID, AppIID CDA

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

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

In the Downlink direction, at the compressor, the DevIID CDA may be

used to generate the L2 addresses on the LPWAN, based on the packet's Destination Address.

7.4.8. Compute-*

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

Because the field is uniquely identified by its FID (e.g., IPv6 length), the relevant protocol specification unambiguously defines the algorithm for such computation.

An example of a field that knows how to recompute itself is IPv6 length.

8. Fragmentation/Reassembly

8.1. Overview

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

The optional SCHC F/R functionality enables such LPWAN technologies to comply with the IPv6 MTU requirement of 1280 bytes [RFC8200]. It is OPTIONAL to implement per this specification, but Profiles may specify that it is REQUIRED.

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

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

SCHC allows transmitting non-fragmented SCHC Packet concurrently with fragmented SCHC Packets. In addition, SCHC F/R provides protocol elements that allow transmitting several fragmented SCHC Packets concurrently, i.e., interleaving the transmission of fragments from different fragmented SCHC Packets. A Profile MAY restrict the latter behavior.

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

8.2. SCHC F/R Protocol Elements

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

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

8.2.1. Messages

SCHC F/R defines the following messages:

SCHC Fragment: A message that carries part of a SCHC Packet from the sender to the receiver.

SCHC ACK: An acknowledgement for fragmentation, by the receiver to

the sender. This message is used to indicate whether or not the reception of pieces of, or the whole of, the fragmented SCHC Packet was successful.

SCHC ACK REQ: A request by the sender for a SCHC ACK from the receiver.

SCHC Sender-Abort: A message by the sender telling the receiver that it has aborted the transmission of a fragmented SCHC Packet.

SCHC Receiver-Abort: A message by the receiver to tell the sender to abort the transmission of a fragmented SCHC Packet.

The format of these messages is provided in Section 8.3.

8.2.2. Tiles, Windows, Bitmaps, Timers, Counters

8.2.2.1. Tiles

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

See Figure 10 for an example.

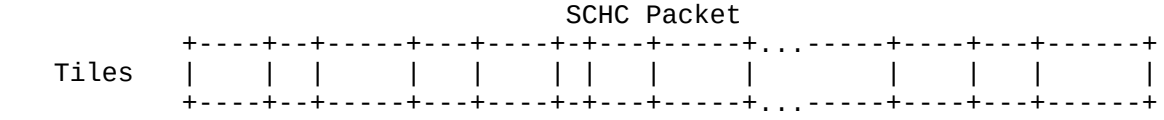


Figure 10: SCHC Packet Fragmented in Tiles

Modes (see Section 8.4) MAY place additional constraints on tile sizes.

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

8.2.2.2. Windows

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

If windows are used:

* all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW_SIZE.

* WINDOW_SIZE MUST be specified in a Profile.

* the windows are numbered.

* their numbers MUST increment by 1 from 0 upward, from the start of the SCHC Packet to its end.

* the last window MUST contain WINDOW_SIZE tiles or less.

* tiles are numbered within each window.

* the tile indices MUST decrement by 1 from WINDOW_SIZE - 1 downward, looking from the start of the SCHC Packet toward its end.

* therefore, each tile of a SCHC Packet is uniquely identified by a window number and a tile index within this window.

See Figure 11 for an example.

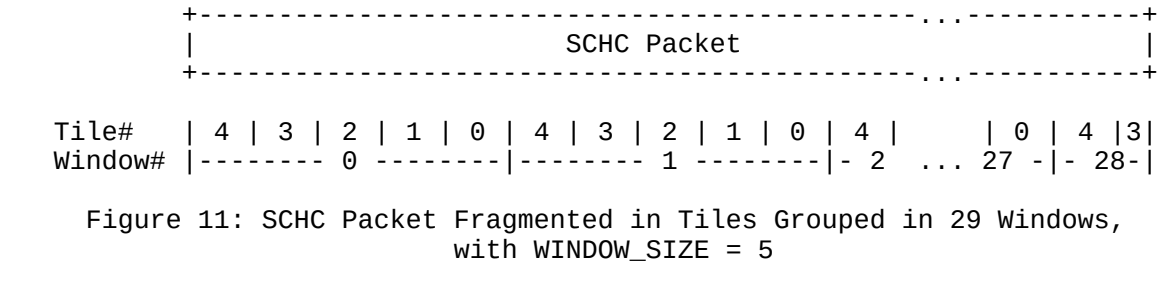


Figure 11: SCHC Packet Fragmented in Tiles Grouped in 29 Windows, with WINDOW_SIZE = 5

Appendix E discusses the benefits of selecting one among multiple window sizes depending on the size of the SCHC Packet to be fragmented.

When windows are used:

* Bitmaps (see Section 8.2.2.3) MAY be sent back by the receiver to the sender in a SCHC ACK message.

* A Bitmap corresponds to exactly one Window.

8.2.2.3. Bitmaps

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

At the receiver:

* a bit set to 1 in the Bitmap indicates that a tile associated with that bit position has been correctly received for that window.

* a bit set to 0 in the Bitmap indicates that there has been no tile correctly received, associated with that bit position, for that window. Possible reasons include that the tile was not sent at all, not received, or received with errors.

8.2.2.4. Timers and Counters

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

Inactivity Timer: a SCHC Fragment receiver uses this timer to abort waiting for a SCHC F/R message.

Retransmission Timer: a SCHC Fragment sender uses this timer to abort waiting for an expected SCHC ACK.

Attempts: this counter counts the requests for SCHC ACKs, up to MAX_ACK_REQUESTS.

8.2.3. Integrity Checking

The integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receive end. A Profile MUST specify how integrity checking is performed.

It is RECOMMENDED that integrity checking be performed by computing a

Reassembly Check Sequence (RCS) based on the SCHC Packet at the sender side and transmitting it to the receiver for comparison with the RCS locally computed after reassembly.

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

The CRC32 polynomial 0xEDB88320 (i.e., the reversed polynomial representation, which is used in the Ethernet standard [ETHERNET]) is RECOMMENDED as the default algorithm for computing the RCS.

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

The concatenation of the complete SCHC Packet and any padding bits, if present, of the last SCHC Fragment does not generally constitute an integer number of bytes. CRC libraries are usually byte oriented. It is RECOMMENDED that the concatenation of the complete SCHC Packet and any last fragment padding bits be zero-extended to the next byte boundary and that the RCS be computed on that byte array.

8.2.4. Header Fields

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

RuleID: this field is present in all the SCHC F/R messages. The Rule identifies:

* that a SCHC F/R message is being carried, as opposed to an unfragmented SCHC Packet,

* which SCHC F/R mode is used,

* in case this mode uses windows, what the value of WINDOW_SIZE is, and

* what other optional fields are present and what the field sizes are.

The Rule tells apart a non-fragmented SCHC Packet from SCHC Fragments. It will also tell apart SCHC Fragments of fragmented SCHC Packets that use different SCHC F/R modes or different parameters. Therefore, interleaved transmission of these is possible.

All SCHC F/R messages pertaining to the same SCHC Packet MUST bear the same RuleID.

Datagram Tag (DTag): This field allows differentiating SCHC F/R messages belonging to different SCHC Packets that may be using the same RuleID simultaneously. Hence, it allows interleaving fragments of a new SCHC Packet with fragments of a previous SCHC Packet under the same RuleID.

The size of the DTag field (called "T", in bits) is defined by each Profile for each RuleID. When T is 0, the DTag field does not appear in the SCHC F/R messages and the DTag value is defined as 0.

When T is 0, there can be no more than one fragmented SCHC Packet in transit for each fragmentation RuleID.

If T is not 0, DTag:

* MUST be set to the same value for all the SCHC F/R messages related to the same fragmented SCHC Packet, and

* MUST be set to different values for SCHC F/R messages related to different SCHC Packets that are being fragmented under the same RuleID and whose transmission may overlap.

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

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

Fragment Compressed Number (FCN): The FCN field is present in the SCHC Fragment Header. Its size (called "N", in bits) is defined by each Profile for each RuleID.

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

* The FCN value with all the bits equal to 1 (called "All-1") signals that the very last tile of a SCHC Packet has been transmitted. By extension, if windows are used, the last window of a packet is called the "All-1" window.

* If windows are used, the FCN value with all the bits equal to 0 (called "All-0") signals the last tile of a window that is not the last one of the SCHC packet. By extension, such a window is called an "All-0 window".

Reassembly Check Sequence (RCS): This field only appears in the All-1 SCHC Fragments. Its size (called "U", in bits) is defined by each Profile for each RuleID.

See Section 8.2.3 for the RCS default size, default polynomial and details on RCS computation.

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

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

Compressed Bitmap: The Compressed Bitmap is used together with windows and Bitmaps (see Section 8.2.2.3). Its presence and size is defined for each SCHC F/R mode for each RuleID.

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

8.3. SCHC F/R Message Formats

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

8.3.1. SCHC Fragment Format

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

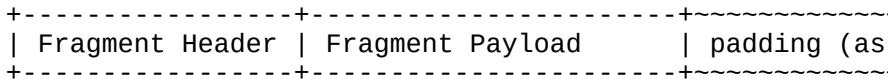


Figure 12: SCHC Fragment General Format

8.3.1.1. Regular SCHC Fragment

The Regular SCHC Fragment format is shown in Figure 13. Regular SCHC Fragments are generally used to carry tiles that are not the last one of a SCHC Packet. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

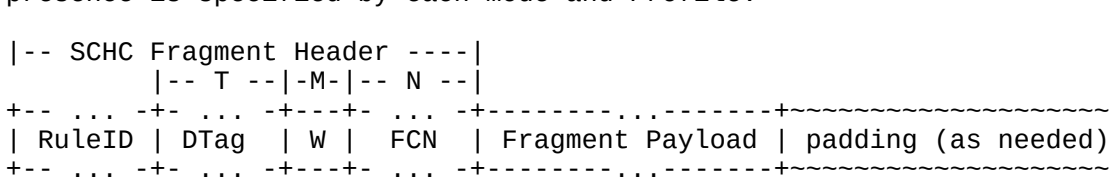


Figure 13: Detailed Header Format for Regular SCHC Fragments

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

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

8.3.1.2. All-1 SCHC Fragment

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

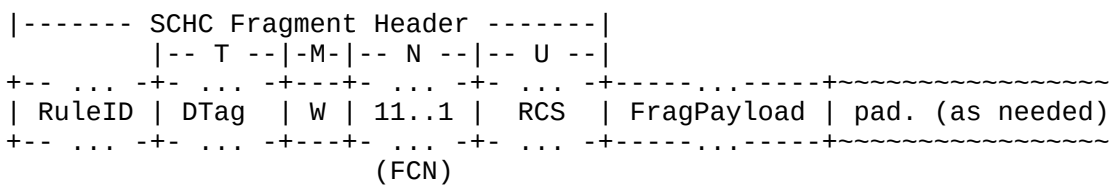


Figure 14: Detailed Header Format for the All-1 SCHC Fragment

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

of L2 Words.

8.3.2. SCHC ACK Format

The SCHC ACK message is shown in Figure 15. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The Compressed Bitmap field MUST be present in SCHC F/R modes that use windows and MUST NOT be present in other modes.

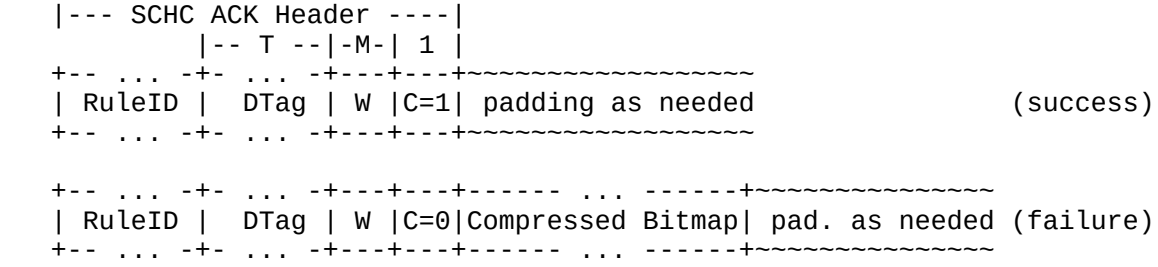


Figure 15: Format of the SCHC ACK Message

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

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

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

8.3.2.1. Bitmap Compression

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

* Build a temporary SCHC ACK message that contains the Header followed by the original Bitmap (see Section 8.2.2.3 for a description of Bitmaps).

* Position scissors at the end of the Bitmap, after its last bit.

* While the bit on the left of the scissors is 1 and belongs to the Bitmap, keep moving left, then stop.

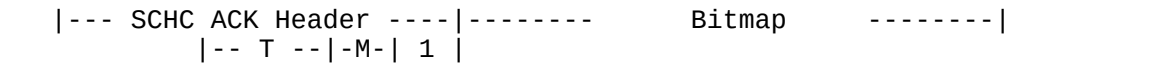
* Then, while the scissors are not on an L2 Word boundary of the SCHC ACK message and there is a Bitmap bit on the right of the scissors, keep moving right, then stop.

* At this point, cut and drop off any bits to the right of the scissors.

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

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

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



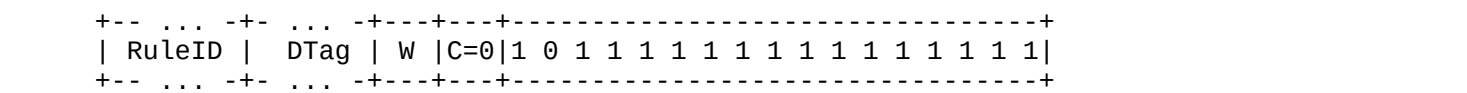
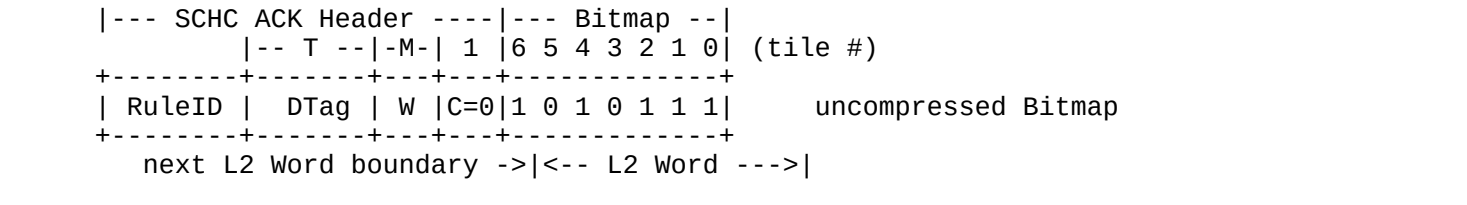


Figure 16: SCHC ACK Header Plus Uncompressed Bitmap

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

Figure 17: Resulting SCHC ACK Message with Compressed Bitmap

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



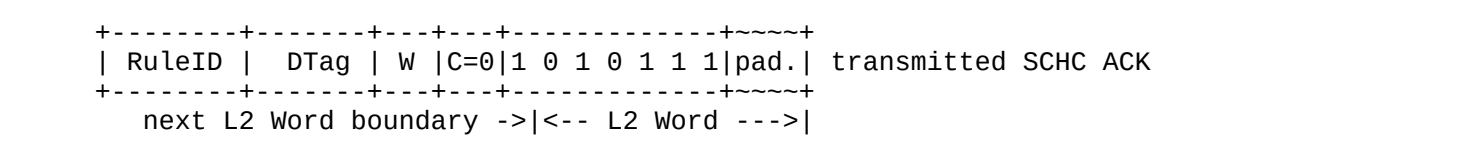


Figure 18: Example of a SCHC ACK Message, Missing Tiles

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

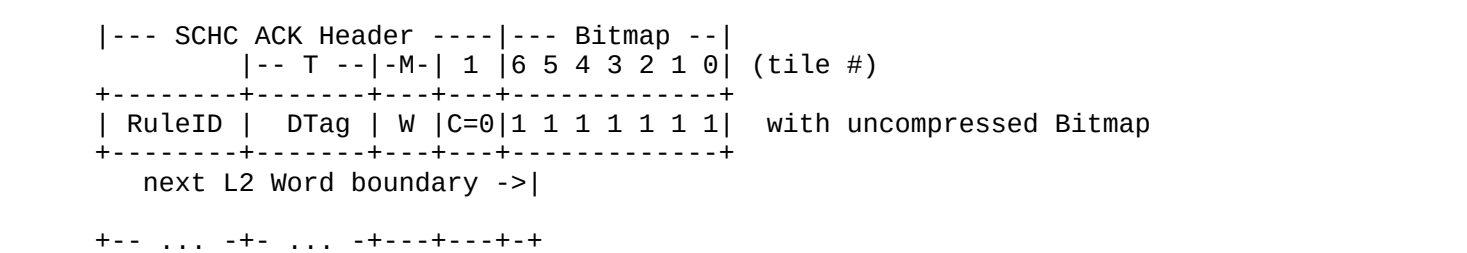
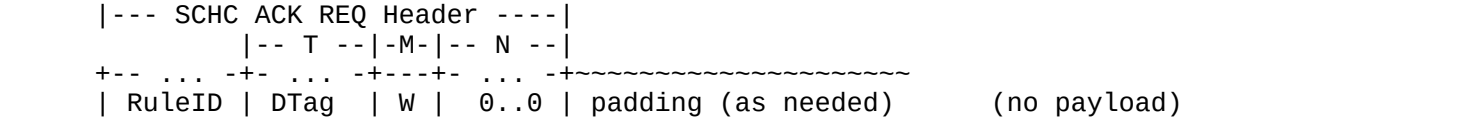


Figure 19: Example of a SCHC ACK Message, No Missing Tile

8.3.3. SCHC ACK REQ Format

The SCHC ACK REQ is used by a sender to request a SCHC ACK from the receiver. Its format is shown in Figure 20. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all zero.



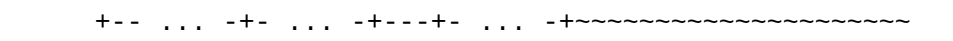


Figure 20: SCHC ACK REQ Format

8.3.4. SCHC Sender-Abort Format

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

The SCHC Sender-Abort format is shown in Figure 21. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all ones.

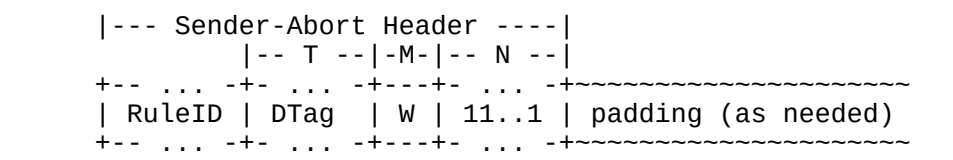


Figure 21: SCHC Sender-Abort Format

If the W field is present:

* the fragment sender MUST set it to all ones. Other values are RESERVED.

* the fragment receiver MUST check its value. If the value is different from all ones, the message MUST be ignored.

The SCHC Sender-Abort MUST NOT be acknowledged.

8.3.5. SCHC Receiver-Abort Format

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

The SCHC Receiver-Abort format is shown in Figure 22. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

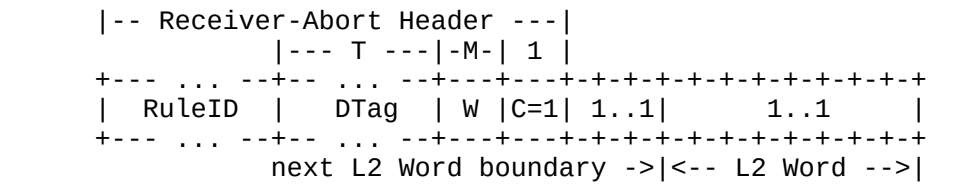


Figure 22: SCHC Receiver-Abort Format

If the W field is present:

* the fragment receiver MUST set it to all ones. Other values are RESERVED.

* if the value is different from all ones, the fragment sender MUST ignore the message.

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

* if the Header does not end at an L2 Word boundary, append bits set to 1 as needed to reach the next L2 Word boundary.

* append exactly one more L2 Word with bits all set to ones.

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

The SCHC Receiver-Abort MUST NOT be acknowledged.

8.4. SCHC F/R Modes

This specification includes several SCHC F/R modes that:

* allow for a range of reliability options, such as optional SCHC Fragment retransmission.

* support various LPWAN characteristics, such as links with variable MTU or unidirectional links.

More modes may be defined in the future.

Appendix B provides examples of fragmentation sessions based on the modes described hereafter.

Appendix C provides examples of Finite State Machines implementing the SCHC F/R modes described hereafter.

8.4.1. No-ACK Mode

The No-ACK mode has been designed under the assumption that data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly. This mode supports L2 technologies that have a variable MTU.

In No-ACK mode, there is no communication from the fragment receiver to the fragment sender. The sender transmits all the SCHC Fragments without expecting any acknowledgement. Therefore, No-ACK does not require bidirectional links: unidirectional links are just fine.

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

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

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

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

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

Each Profile, for each RuleID value, MUST define:

* the size of the DTag field,

* the size and algorithm for the RCS field, and

* the expiration time of the Inactivity Timer.

Each Profile, for each RuleID value, MAY define

* a value of N different from the recommended one, and

* the meaning of values sent in the FCN field, for values different

from the All-1 value.

For each active pair of RuleID and DTag values, the receiver MUST maintain an Inactivity Timer. If the receiver is under-resourced to do this, it MUST silently drop the related messages.

8.4.1.1. Sender Behavior

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

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

The sender MAY transmit a SCHC Sender-Abort.

Figure 39 shows an example of a corresponding state machine.

8.4.1.2. Receiver Behavior

Upon receiving each Regular SCHC Fragment:

* the receiver MUST reset the Inactivity Timer.

* the receiver assembles the payloads of the SCHC Fragments.

On receiving an All-1 SCHC Fragment:

* the receiver MUST append the All-1 SCHC Fragment Payload and the padding bits to the previously received SCHC Fragment Payloads for this SCHC Packet.

* the receiver MUST perform the integrity check.

* if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet.

* the reassembly operation concludes.

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

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

Figure 40 shows an example of a corresponding state machine.

8.4.2. ACK-Always Mode

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

* Data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly,

* The L2 MTU value does not change while the fragments of a SCHC

Packet are being transmitted, and

* There is a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-Always mode on quasi-bidirectional links.

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

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

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

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

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

Each Profile, for each RuleID value, MUST define:

* the value of N,

* the value of WINDOW_SIZE, which MUST be strictly less than 2^N,

* the size and algorithm for the RCS field,

* the value of T,

* the value of MAX_ACK_REQUESTS,

* the expiration time of the Retransmission Timer, and

* the expiration time of the Inactivity Timer.

For each active pair of RuleID and DTag values, the sender MUST maintain:

* one Attempts counter

* one Retransmission Timer

For each active pair of RuleID and DTag values, the receiver MUST maintain

* one Inactivity Timer, and

* one Attempts counter.

8.4.2.1. Sender Behavior

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

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

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

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

* the Fragment MUST be of the Regular type specified in Section 8.3.1.1.

* the FCN field MUST contain the tile index.

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

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

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

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

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

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

* Then, upon receiving a SCHC ACK:

  - if the SCHC ACK indicates that some tiles are missing at the receiver, then the sender MUST transmit all the tiles that have been reported missing, it MUST increment Attempts, it MUST reset the Retransmission Timer, and MUST await the next SCHC ACK.

  - if the current window is not the last one and the SCHC ACK indicates that all tiles were correctly received, the sender MUST stop the Retransmission Timer, it MUST advance to the next fragmentation window, and it MUST start a blind transmission phase as described above.

  - if the current window is the last one and the SCHC ACK indicates that more tiles were received than the sender sent, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

  - if the current window is the last one and the SCHC ACK indicates that all tiles were correctly received, yet the integrity check was a failure, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

  - if the current window is the last one and the SCHC ACK indicates that integrity checking was successful, the sender exits successfully.

* on Retransmission Timer expiration:

- if Attempts is strictly less that MAX_ACK_REQUESTS, the fragment sender MUST send a SCHC ACK REQ and MUST increment the Attempts counter.

- otherwise, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

At any time:

* on receiving a SCHC Receiver-Abort, the fragment sender MAY exit with an error condition.

* on receiving a SCHC ACK that bears a W value different from the W value that it currently uses, the fragment sender MUST silently discard and ignore that SCHC ACK.

Figure 41 shows an example of a corresponding state machine.

8.4.2.2. Receiver Behavior

On receiving a SCHC Fragment with a RuleID and DTag pair not being processed at that time:

* the receiver SHOULD check if the DTag value has not recently been used for that RuleID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

* the receiver MUST start a process to assemble a new SCHC Packet with that RuleID and DTag value pair.

* the receiver MUST start an Inactivity Timer for that RuleID and DTag pair. It MUST initialize an Attempts counter to 0 for that RuleID and DTag pair. It MUST initialize a window counter to 0. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver-Abort.

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

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

The receiver MUST first initialize an empty Bitmap for the first window then enter an "acceptance phase", in which:

* on receiving a SCHC Fragment or a SCHC ACK REQ, either one having the W bit different from the local W bit, the receiver MUST silently ignore and discard that message.

* on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

* on receiving a SCHC Fragment with the W bit equal to the local W bit, the receiver MUST assemble the received tile based on the window counter and on the FCN field in the SCHC Fragment, and it MUST update the Bitmap.

- if the SCHC Fragment received is an All-0 SCHC Fragment, the current window is determined to be a not-last window, the receiver MUST send a SCHC ACK for this window and it MUST enter the "retransmission phase" for this window.

- if the SCHC Fragment received is an All-1 SCHC Fragment, the current window is determined to be the last window, the padding bits of the All-1 SCHC Fragment MUST be assembled after the received tile, the receiver MUST perform the integrity check and it MUST send a SCHC ACK for this window. Then:

  o If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for this window.

  o If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for this window.

In the "retransmission phase":

* if the window is a not-last window:

  - on receiving a SCHC Fragment that is not All-0 or All-1 and that has a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap, and it MUST enter the "acceptance phase" for that new window.

  - on receiving a SCHC ACK REQ with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST send a SCHC ACK for that new window, and it MUST enter the "acceptance phase" for that new window.

  - on receiving a SCHC All-0 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap, it MUST send a SCHC ACK for that new window, and it MUST stay in the "retransmission phase" for that new window.

  - on receiving a SCHC All-1 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap; it MUST assemble the tile received, including the padding bits; it MUST update the Bitmap and perform the integrity check; it MUST send a SCHC ACK for the new window, which is determined to be the last window. Then:

    o If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for that new window.

    o If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for that new window.

  - on receiving a SCHC Fragment with a W bit equal to the local W bit:

    o if the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST silently ignore it and discard it.

    o otherwise, the receiver MUST assemble the tile received and update the Bitmap. If the Bitmap becomes fully populated

with 1's or if the SCHC Fragment is an All-0, the receiver MUST send a SCHC ACK for this window.

- on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

* if the window is the last window:

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

  - on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

  - on receiving a SCHC Fragment with a W bit equal to the local W bit:

    o if the SCHC Fragment received is an All-0 SCHC Fragment, the receiver MUST silently ignore it and discard it.

    o otherwise, the receiver MUST update the Bitmap, and it MUST assemble the tile received. If the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST assemble the padding bits of the All-1 SCHC Fragment after the received tile, it MUST perform the integrity check and:

      + if the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST send a SCHC ACK and it enters the "clean-up phase".

      + if the integrity check indicates that the full SCHC Packet has not been correctly reassembled:

        * if the SCHC Fragment received was an All-1 SCHC Fragment, the receiver MUST send a SCHC ACK for this window.

In the "clean-up phase":

* On receiving an All-1 SCHC Fragment or a SCHC ACK REQ, either one having the W bit equal to the local W bit, the receiver MUST send a SCHC ACK.

* Any other SCHC Fragment received MUST be silently ignored and discarded.

At any time, on sending a SCHC ACK, the receiver MUST increment the Attempts counter.

At any time, on incrementing its window counter, the receiver MUST reset the Attempts counter.

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

Figure 42 shows an example of a corresponding state machine.

8.4.3. ACK-on-Error Mode

The ACK-on-Error mode supports L2 technologies that have variable MTU and out-of-order delivery. It requires an L2 that provides a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-on-Error mode on quasi-bidirectional

links.

In ACK-on-Error mode, windows are used.

All tiles except the last one and the penultimate one MUST be of equal size, hereafter called "regular". The size of the last tile MUST be smaller than or equal to the regular tile size. Regarding the penultimate tile, a Profile MUST pick one of the following two options:

* The penultimate tile size MUST be the regular tile size, or

* the penultimate tile size MUST be either the regular tile size or the regular tile size minus one L2 Word.

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

See Figure 23 for an example.

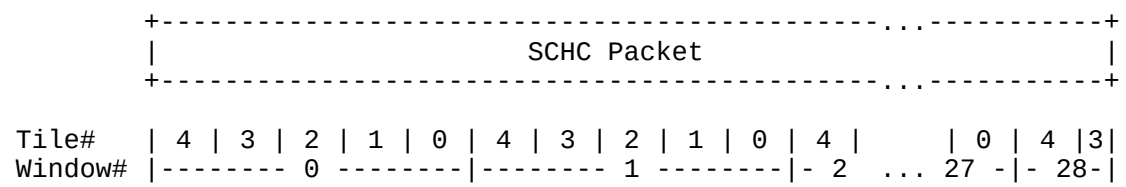


Figure 23: SCHC Packet Fragmented in Tiles, ACK-on-Error Mode

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

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

* integrity checking shows that the fragmented SCHC Packet has been correctly reassembled at the receive end, and this information has been conveyed back to the sender, or

* too many retransmission attempts were made, or

* the receiver determines that the transmission of this fragmented SCHC Packet has been inactive for too long.

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

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

Each Profile, for each RuleID value, MUST define:

* the tile size (a tile does not need to be multiple of an L2 Word, but it MUST be at least the size of an L2 Word),

* the value of M,

* the value of N,

* the value of WINDOW_SIZE, which MUST be strictly less than 2^N,

* the size and algorithm for the RCS field,

* the value of T,

* the value of MAX_ACK_REQUESTS,

* the expiration time of the Retransmission Timer,

* the expiration time of the Inactivity Timer,

* if the last tile is carried in a Regular SCHC Fragment or an All-1 SCHC Fragment (see Section 8.4.3.1), and

* if the penultimate tile MAY be one L2 Word smaller than the regular tile size. In this case, the regular tile size MUST be at least twice the L2 Word size.

For each active pair of RuleID and DTag values, the sender MUST maintain:

* one Attempts counter, and

* one Retransmission Timer.

For each active pair of RuleID and DTag values, the receiver MUST maintain:

* one Inactivity Timer, and

* one Attempts counter.

8.4.3.1. Sender Behavior

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

* the fragment sender MUST select a RuleID and DTag value pair for this SCHC Packet. A Rule MUST NOT be selected if the values of M and WINDOW_SIZE for that Rule are such that the SCHC Packet cannot be fragmented in (2^M) * WINDOW_SIZE tiles or less.

* the fragment sender MUST initialize the Attempts counter to 0 for that RuleID and DTag value pair.

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

* the selected tiles MUST be contiguous in the original SCHC Packet, and

* they MUST be placed in the SCHC Fragment Payload adjacent to one another, in the order they appear in the SCHC Packet, from the start of the SCHC Packet toward its end.

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

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

A Profile MUST define if the last tile of a SCHC Packet is sent:

* in a Regular SCHC Fragment, alone or as part of a multi-tiles Payload,

* alone in an All-1 SCHC Fragment, or

* with any of the above two methods.

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

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

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

In doing the two items above, the sender MUST ascertain that the receiver will not receive the last tile through both a Regular SCHC Fragment and an All-1 SCHC Fragment.

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

* an All-1 SCHC Fragment, or

* a SCHC ACK REQ.

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

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

* it MUST increment the Attempts counter, and

* it MUST reset the Retransmission Timer.

On Retransmission Timer expiration:

* if the Attempts counter is strictly less than MAX_ACK_REQUESTS, the fragment sender MUST send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window,

* otherwise, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

On receiving a SCHC ACK:

* if the W field in the SCHC ACK corresponds to the last window of the SCHC Packet:

  - if the C bit is set, the sender MAY exit successfully.

  - otherwise:

    o if the Profile mandates that the last tile be sent in an All-1 SCHC Fragment:

      + if the SCHC ACK shows no missing tile at the receiver,

the sender:

* MUST send a SCHC Sender-Abort, and

* MAY exit with an error condition.

+ otherwise:

* the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

* if the last of these SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MUST in addition send after it a SCHC ACK REQ with the W field corresponding to the last window.

* in doing the two items above, the sender MUST ascertain that the receiver will not receive the last tile through both a Regular SCHC Fragment and an All-1 SCHC Fragment.

o otherwise:

+ if the SCHC ACK shows no missing tile at the receiver, the sender MUST send the All-1 SCHC Fragment

+ otherwise:

* the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

* the fragment sender MUST then send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window.

* otherwise, the fragment sender:

- MUST send SCHC Fragment messages containing the tiles that are reported missing in the SCHC ACK.

- then, it MAY send a SCHC ACK REQ with the W field corresponding to the last window.

See Figure 43 for one among several possible examples of a Finite State Machine implementing a sender behavior obeying this specification.

8.4.3.2. Receiver Behavior

On receiving a SCHC Fragment with a RuleID and DTag pair not being processed at that time:

* the receiver SHOULD check if the DTag value has not recently been used for that RuleID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

* the receiver MUST start a process to assemble a new SCHC Packet with that RuleID and DTag value pair. The receiver MUST start an Inactivity Timer for that RuleID and DTag value pair. It MUST initialize an Attempts counter to 0 for that RuleID and DTag value

pair. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver-Abort.

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

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

* if the FCN is All-1, if a Payload is present, the full SCHC Fragment Payload MUST be assembled including the padding bits. This is because the size of the last tile is not known by the receiver; therefore, padding bits are indistinguishable from the tile data bits, at this stage. They will be removed by the SCHC C/D sublayer. If the size of the SCHC Fragment Payload exceeds or equals the size of one regular tile plus the size of an L2 Word, this SHOULD raise an error flag.

* otherwise, tiles MUST be assembled based on the a priori known tile size.

  - If allowed by the Profile, the end of the payload MAY contain the last tile, which may be shorter. Padding bits are indistinguishable from the tile data bits, at this stage.

  - The payload may contain the penultimate tile that, if allowed by the Profile, MAY be exactly one L2 Word shorter than the regular tile size.

  - Otherwise, padding bits MUST be discarded. This is possible because:

    o the size of the tiles is known a priori,

    o tiles are larger than an L2 Word, and

    o padding bits are always strictly less than an L2 Word.

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

* if the receiver knows of any windows with missing tiles for the packet being reassembled, it MUST return a SCHC ACK for the lowest-numbered such window:

* otherwise:

  - if it has received at least one tile, it MUST return a SCHC ACK for the highest-numbered window it currently has tiles for,

  - otherwise, it MUST return a SCHC ACK for window numbered 0.

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

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

After receiving an All-1 SCHC Fragment, a receiver MUST check the integrity of the reassembled SCHC Packet at least every time it

prepares for sending a SCHC ACK for the last window.

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

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

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

Reassembly of the SCHC Packet concludes when:

* a Sender-Abort has been received, or

* the Inactivity Timer has expired, or

* the Attempts counter has exceeded MAX_ACK_REQUESTS, or

* at least an All-1 SCHC Fragment has been received and integrity checking of the reassembled SCHC Packet is successful.

See Figure 44 for one among several possible examples of a Finite State Machine implementing a receiver behavior obeying this specification. The example provided is meant to match the sender Finite State Machine of Figure 43.

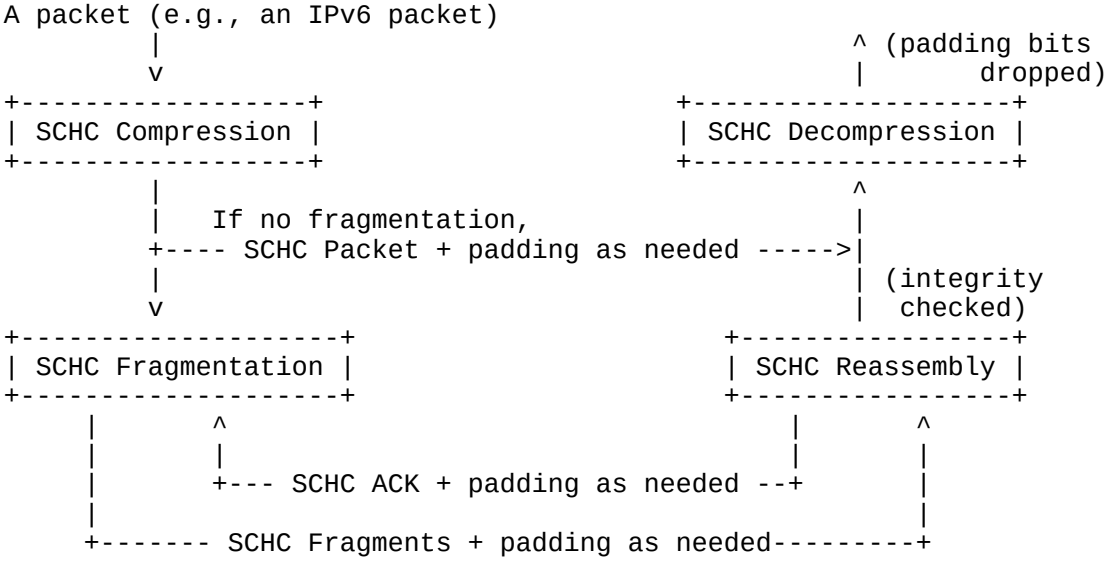
9. Padding Management

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

If the L2 constrains the payload to align to coarser boundaries (for example, bytes), the SCHC messages MUST be padded. When padding occurs, the number of appended bits MUST be strictly less than the L2 Word size.

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

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



Sender Receiver

Figure 24: SCHC Operations, Including Padding as Needed

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

A Profile MUST define the value of the padding bits if the L2 Word is wider than a single bit. The RECOMMENDED value is 0.

10. SCHC Compression for IPv6 and UDP Headers

This section lists the IPv6 and UDP header fields and describes how they can be compressed. An example of a set of Rules for UDP/IPv6 header compression is provided in Appendix A.

10.1. IPv6 Version Field

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

10.2. IPv6 Traffic Class Field

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

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

* 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".

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

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

10.3. Flow Label Field

If the flow label is not set, i.e., its value is zero, the Field Descriptor in the Rule SHOULD contain a TV set to zero, an "equal" MO, and a "not-sent" CDA.

If the flow label is set to a pseudorandom value according to [RFC6437], in the Rule, TV is not set to any particular value, MO is set to "ignore", and CDA is set to "value-sent".

If the flow label is set according to some prior agreement, i.e., by a flow state establishment method as allowed by [RFC6437], the Field Descriptor in the Rule SHOULD contain a TV with this agreed-upon value, an "equal" MO, and a "not-sent" CDA.

10.4. Payload Length Field

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

"compute-*".

10.5. Next Header Field

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

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

10.6. Hop Limit Field

The field behavior for this field is different for Uplink 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 on Internet routing and can change more frequently. The DI can be used to distinguish both directions:

* in an Up Field Descriptor, elide the field: the TV is set to the known constant value, the MO is set to "equal" and the CDA is set to "not-sent".

* in a Dw Field Descriptor, the Hop Limit is elided for transmission and forced to 1 at the receiver, by setting TV to 1, MO to "ignore" and CDA to "not-sent". This prevents any further forwarding.

10.7. IPv6 Addresses Fields

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

10.7.1. IPv6 Source and Destination Prefixes

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

If the Rule is intended to compress packets with different prefix values, match-mapping SHOULD be used. The different prefixes are listed in the TV, the MO is set to "match-mapping" and the CDA is set to "mapping-sent". See Figure 26.

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

10.7.2. IPv6 Source and Destination IID

If the Dev or App IID are based on an L2 address, then the IID can be reconstructed with information coming from the L2 header. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "DevIID" or "AppIID". On LPWAN technologies where the frames carry a single identifier (corresponding to the Dev), AppIID cannot be used.

As described in [RFC8065], it may be undesirable to build the Dev IPv6 IID out of the Dev address. Another static value is used

instead. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDA is set to "not-sent".

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

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

Finally, the IID can be sent in its entirety on the L2. 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 Extension Headers

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

10.9. UDP Source and Destination Ports

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

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

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

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

Otherwise, the port numbers are sent over the L2. 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 parser MUST NOT label this field unless the UDP Length value matches the Payload Length value from the IPv6 header. The TV is not set, the MO is set to "ignore", and the CDA is set to "compute-*".

10.11. UDP Checksum Field

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

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

6LoWPAN Header Compression [RFC6282] also specifies that a UDP checksum can be elided by the compressor and recomputed by the decompressor when an upper layer guarantees the integrity of the UDP payload and pseudo-header. A specific example of this is when a

message integrity check protects the compressed message between the compressor that elides the UDP checksum and the decompressor that computes it, with a strength that is identical or better to the UDP checksum.

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

In particular, when SCHC fragmentation is used, a fragmentation RCS of 2 bytes or more provides equal or better protection than the UDP checksum; in that case, if the compressor is collocated with the fragmentation point and the decompressor is collocated with the packet reassembly point, and if the SCHC Packet is fragmented even when it would fit unfragmented in the L2 MTU, then the compressor MAY verify and then elide the UDP checksum. Whether and when the UDP Checksum is elided is to be specified in the Profile.

Since the compression happens before the fragmentation, implementers 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 IANA actions.

12. Security Considerations

As explained in Section 5, SCHC is expected to be implemented on top of LPWAN technologies, which are expected to implement security measures.

In this section, we analyze the potential security threats that could be introduced into an LPWAN by adding the SCHC functionalities.

12.1. Security Considerations for SCHC Compression/Decompression

12.1.1. Forged SCHC Packet

Let’s assume that an attacker is able to send a forged SCHC Packet to a SCHC decompressor.

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

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

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

As a general safety measure, a SCHC decompressor should never reconstruct a packet larger than MAX_PACKET_SIZE (defined in a Profile, with 1500 bytes as generic default).

12.1.2. Compressed Packet Size as a Side Channel to Guess a Secret Token

Some packet compression methods are known to be susceptible to attacks, such as BREACH and CRIME. The attack involves injecting arbitrary data into the packet and observing the resulting compressed packet size. The observed size potentially reflects correlation between the arbitrary data and some content that was meant to remain secret, such as a security token, thereby allowing the attacker to get at the secret.

By contrast, SCHC compression takes place header field by header field, with the SCHC Packet being a mere concatenation of the compression residues of each of the individual field. Any correlation between header fields does not result in a change in the SCHC Packet size compressed under the same Rule.

If SCHC C/D is used to compress packets that include a secret information field, such as a token, the Rule set should be designed so that the size of the compression residue for the field to remain secret is the same irrespective of the value of the secret information. This is achieved by, e.g., sending this field in extenso with the "ignore" MO and the "value-sent" CDA. This recommendation is disputable if it is ascertained that the Rule set itself will remain secret.

12.1.3. Decompressed Packet Different from the Original Packet

As explained in Section 7.2, using FPs with value 0 in Field Descriptors in a Rule may result in header fields appearing in the decompressed packet in an order different from that in the original packet. Likewise, as stated in Section 7.4.3, using an "ignore" MO together with a "not-sent" CDA will result in the header field taking the TV value, which is likely to be different from the original value.

Depending on the protocol, the order of header fields in the packet may or may not be functionally significant.

Furthermore, if the packet is protected by a checksum or a similar integrity protection mechanism, and if the checksum is transmitted instead of being recomputed as part of the decompression, these situations may result in the packet being considered corrupt and dropped.

12.2. Security Considerations for SCHC Fragmentation/Reassembly

12.2.1. Buffer Reservation Attack

Let’s assume that an attacker is able to send a forged SCHC Fragment to a SCHC reassembler.

A node can perform a buffer reservation attack: the receiver will reserve buffer space for the SCHC Packet. If the implementation has only one buffer, other incoming fragmented SCHC Packets will be dropped while the reassembly buffer is occupied during the reassembly timeout. Once that timeout expires, the attacker can repeat the same procedure, and iterate, thus, creating a denial-of-service attack. An implementation may have multiple reassembly buffers. The cost to mount this attack is linear with the number of buffers at the target node. Better, the cost for an attacker can be increased if individual fragments of multiple SCHC Packets can be stored in the

reassembly buffer. The finer grained the reassembly buffer (down to the smallest tile size), the higher the cost of the attack. If buffer overload does occur, a smart receiver could selectively discard SCHC Packets being reassembled based on the sender behavior, which may help identify which SCHC Fragments have been sent by the attacker. Another mild countermeasure is for the target to abort the fragmentation/reassembly session as early as it detects a non-identical SCHC Fragment duplicate, anticipating for an eventual corrupt SCHC Packet, so as to save the sender the hassle of sending the rest of the fragments for this SCHC Packet.

12.2.2. Corrupt Fragment Attack

Let’s assume that an attacker is able to send a forged SCHC Fragment to a SCHC reassembler. The malicious node is additionally assumed to be able to hear an incoming communication destined to the target node.

It can then send a forged SCHC Fragment that looks like it belongs to a SCHC Packet already being reassembled at the target node. This can cause the SCHC Packet to be considered corrupt and to be dropped by the receiver. The amplification happens here by a single spoofed SCHC Fragment rendering a full sequence of legitimate SCHC Fragments useless. If the target uses ACK-Always or ACK-on-Error mode, such a malicious node can also interfere with the acknowledgement and repetition algorithm of SCHC F/R. A single spoofed ACK, with all Bitmap bits set to 0, will trigger the repetition of WINDOW_SIZE tiles. This protocol loop amplification depletes the energy source of the target node and consumes the channel bandwidth. Similarly, a spoofed ACK REQ will trigger the sending of a SCHC ACK, which may be much larger than the ACK REQ if WINDOW_SIZE is large. These consequences should be borne in mind when defining profiles for SCHC over specific LPWAN technologies.

12.2.3. Fragmentation as a Way to Bypass Network Inspection

Fragmentation is known for potentially allowing one to force through a Network Inspection device (e.g., firewall) packets that would be rejected if unfragmented. This involves sending overlapping fragments to rewrite fields whose initial value led the Network Inspection device to allow the flow to go through.

SCHC F/R is expected to be used over one LPWAN link, where no Network Inspection device is expected to sit. As described in Section 5.2, even if the SCHC F/R on the Network Infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW.

12.2.4. Privacy Issues Associated with SCHC Header Fields

SCHC F/R allocates a DTag value to fragments belonging to the same SCHC Packet. Concerns were raised that, if DTag is a wide counter that is incremented in a predictable fashion for each new fragmented SCHC Packet, it might lead to a privacy issue, such as enabling tracking of a device across LPWANs.

However, SCHC F/R is expected to be used over exactly one LPWAN link. As described in Section 5.2, even if the SCHC F/R on the Network Infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW. Therefore, assuming the tunnel provides confidentiality, neither the DTag field nor any other SCHC-introduced field is visible over the Internet.

13. References

13.1. Normative References

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

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

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

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

[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>.

13.2. Informative References

[ETHERNET] IEEE, "IEEE Standard for Ethernet", DOI 10.1109/IEEESTD.2012.6419735, IEEE Standard 802.3-2012, December 2012, <https://ieeexplore.ieee.org/document/6419735>.

[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>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.

[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>.

[RFC8065] Thaler, D., "Privacy Considerations for IPv6 Adaptation-Layer Mechanisms", RFC 8065, DOI 10.17487/RFC8065, February 2017, <https://www.rfc-editor.org/info/rfc8065>.

Appendix A. Compression Examples

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

The mechanisms defined in this document can be applied to a Dev that embeds some applications running over CoAP. In this example, three flows are considered. The first flow is for the device management

based on CoAP using Link Local IPv6 addresses and UDP ports 123 and 124 for Dev and App, respectively. The second flow is a CoAP server for measurements done by the Dev (using ports 5683) and Global IPv6 Address prefixes alpha::IID/64 to beta::1/64. The last flow is for legacy applications using different ports numbers, the destination IPv6 address prefix is gamma::1/64.

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

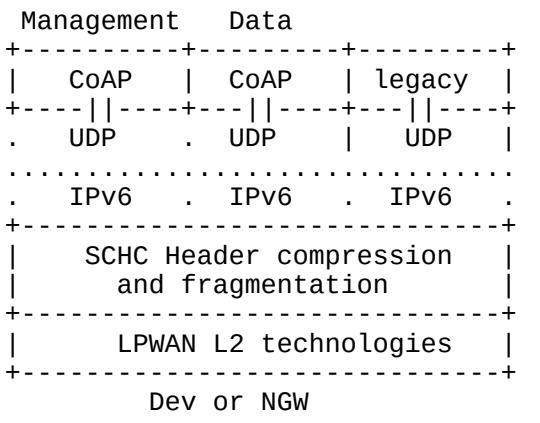


Figure 25: Simplified Protocol Stack for LP-WAN

Rule 0
Special RuleID used to tag an uncompressed UDP/IPV6 packet.

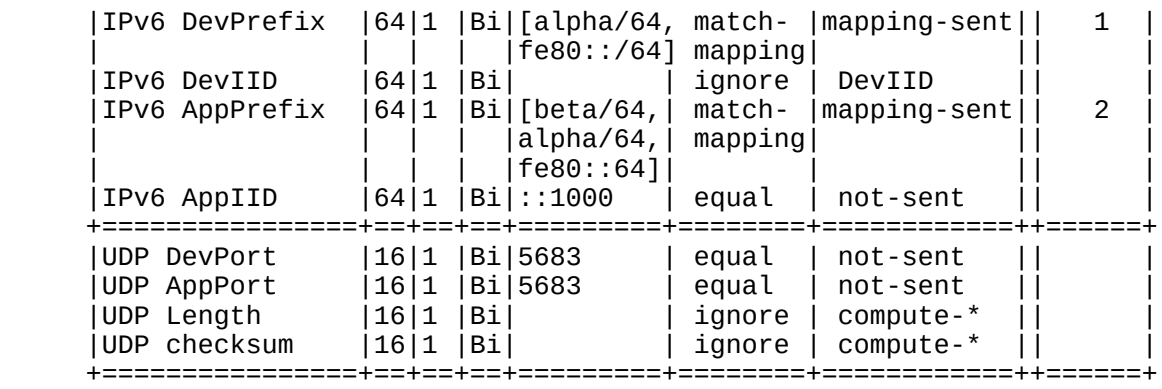
Rule 1

FID	FL	FP	DI	TV	MO	CDA	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 Diffserv	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent	
IPv6 DevPrefix	64	1	Bi	FE80::/64	equal	not-sent	
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	FE80::/64	equal	not-sent	
IPv6 AppIID	64	1	Bi	::1	equal	not-sent	
UDP DevPort	16	1	Bi	123	equal	not-sent	
UDP AppPort	16	1	Bi	124	equal	not-sent	
UDP Length	16	1	Bi		ignore	compute-*	
UDP checksum	16	1	Bi		ignore	compute-*	

Figure 26: Context Rules - Rule 0 and Rule 1

Rule 2

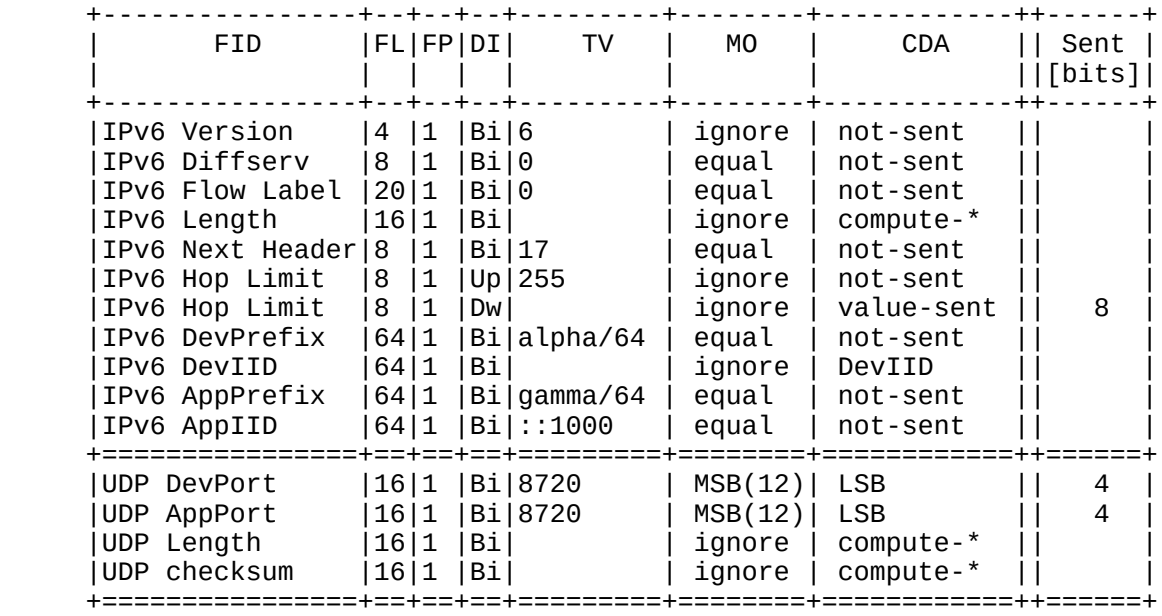
FID	FL	FP	DI	TV	MO	CDA	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 Diffserv	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent	



IPv6 DevPrefix	64	1	Bi	[alpha/64, fe80::/64]	match-mapping	mapping-sent		1
IPv6 DevIID	64	1	Bi		ignore	DevIID		
IPv6 AppPrefix	64	1	Bi	[beta/64, alpha/64, fe80::64]	match-mapping	mapping-sent		2
IPv6 AppIID	64	1	Bi	::1000	equal	not-sent		
UDP DevPort	16	1	Bi	5683	equal	not-sent		
UDP AppPort	16	1	Bi	5683	equal	not-sent		
UDP Length	16	1	Bi		ignore	compute-*		
UDP checksum	16	1	Bi		ignore	compute-*		

Figure 27: Context Rules - Rule 2

Rule 3



FID	FL	FP	DI	TV	MO	CDA	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 Diffserv	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Up	255	ignore	not-sent	
IPv6 Hop Limit	8	1	Dw		ignore	value-sent	8
IPv6 DevPrefix	64	1	Bi	alpha/64	equal	not-sent	
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	gamma/64	equal	not-sent	
IPv6 AppIID	64	1	Bi	::1000	equal	not-sent	
UDP DevPort	16	1	Bi	8720	MSB(12)	LSB	4
UDP AppPort	16	1	Bi	8720	MSB(12)	LSB	4
UDP Length	16	1	Bi		ignore	compute-*	
UDP checksum	16	1	Bi		ignore	compute-*	

Figure 28: Context Rules - Rule 3

Figures 26 to 28 describe an example of a Rule set.

In this example, 0 was chosen as the special RuleID that tags packets that cannot be compressed with any compression Rule.

All the fields described in Rules 1-3 are present in the IPv6 and UDP headers. The DevIID value is inferred from the L2 header.

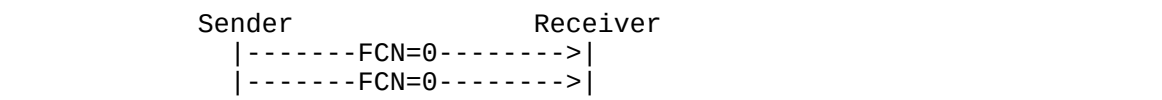
Rules 2-3 use global addresses. The way the Dev learns the prefix is not in the scope of the document.

Rule 3 compresses each port number to 4 bits.

Appendix B. Fragmentation Examples

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

Figure 29 illustrates the transmission in No-ACK mode of a SCHC Packet that needs 11 SCHC Fragments. FCN is 1 bit wide.



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

Figure 29: No-ACK Mode, 11 SCHC Fragments

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

Figure 30 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7 and no lost SCHC Fragment.

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

Figure 30: ACK-on-Error Mode, 11 Tiles, One Tile per SCHC Fragment, No Lost SCHC Fragment

Figure 31 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7, and three lost SCHC Fragments.

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

Figure 31: ACK-on-Error Mode, 11 Tiles, One Tile per SCHC Fragment, Lost SCHC Fragments

Figure 32 shows an example of a transmission in ACK-on-Error mode of a SCHC Packet fragmented in 73 tiles, with N=5, WINDOW_SIZE=28, M=2, and three lost SCHC Fragments.

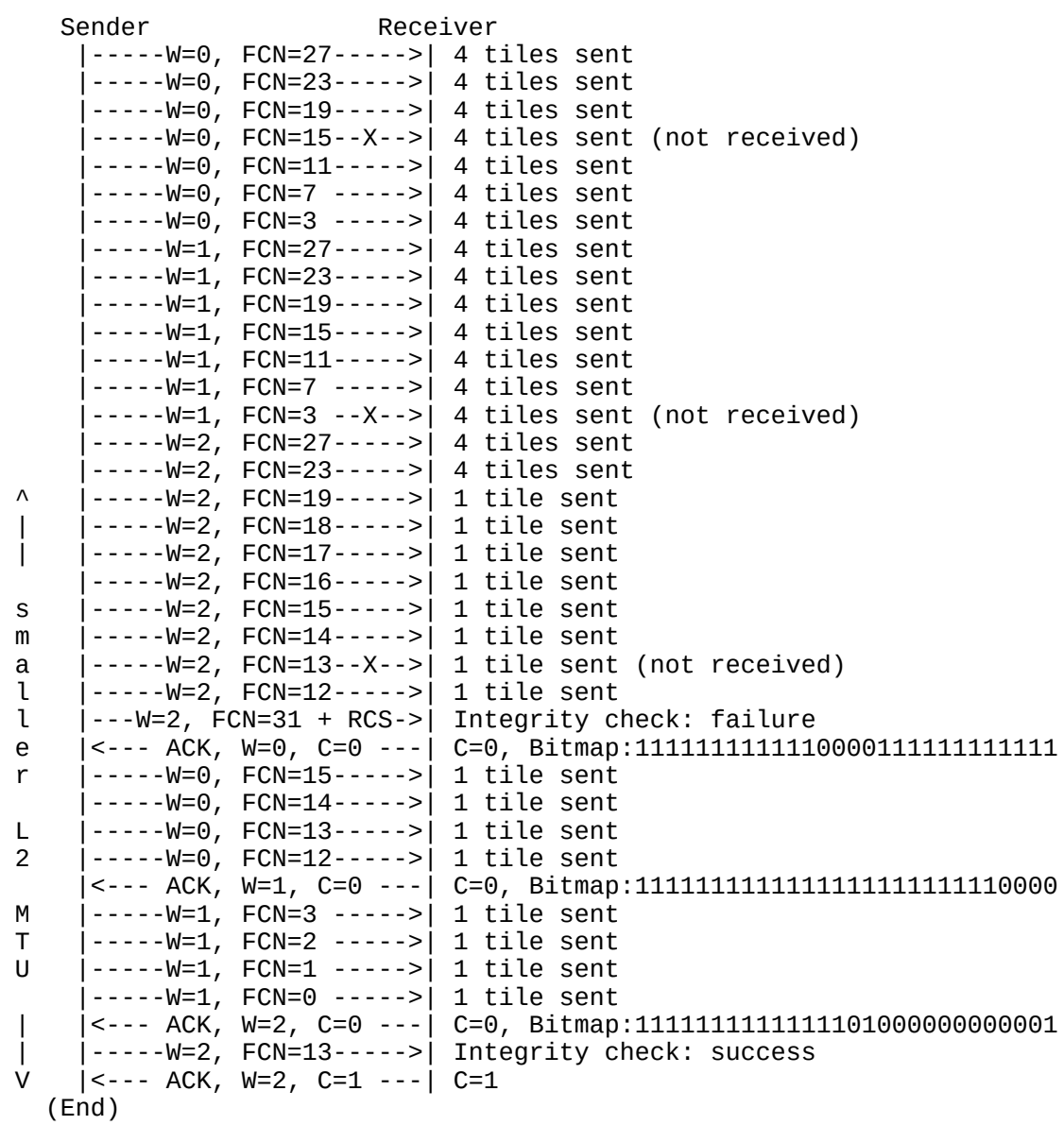
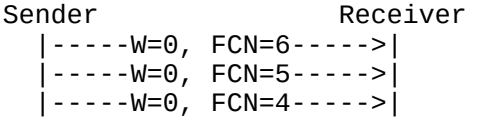


Figure 32: ACK-on-Error Mode, Variable MTU

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

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

Figure 33 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, with N=3, WINDOW_SIZE=7, and no loss.



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

Figure 33: ACK-Always Mode, 11 Tiles, One Tile per SCHC Fragment, No Loss

Figure 34 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7 and three lost SCHC Fragments.

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

Figure 34: ACK-Always Mode, 11 Tiles, One Tile per SCHC Fragment, Three Lost SCHC Fragments

Figure 35 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in six tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7, three lost SCHC Fragments, and only one retry needed to recover each lost SCHC Fragment.

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

Figure 35: ACK-Always Mode, Six Tiles, One Tile per SCHC Fragment, Three Lost SCHC Fragments

Figure 36 illustrates the transmission in ACK-Always mode of a SCHC

Packet fragmented in six tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7, three lost SCHC Fragments, and the second SCHC ACK lost.

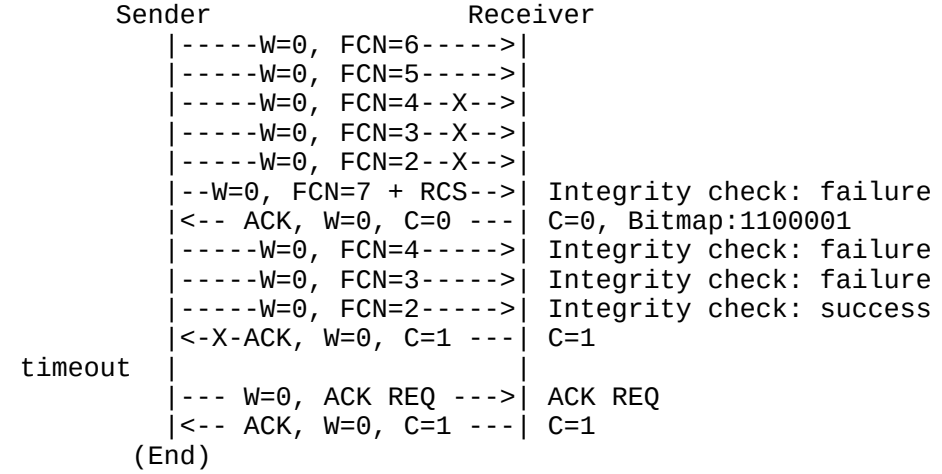


Figure 36: ACK-Always Mode, Six Tiles, One Tile per SCHC Fragment, SCHC ACK Loss

Figure 37 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in six tiles, with N=3, WINDOW_SIZE=7, with three lost SCHC Fragments, and one retransmitted SCHC Fragment lost again.

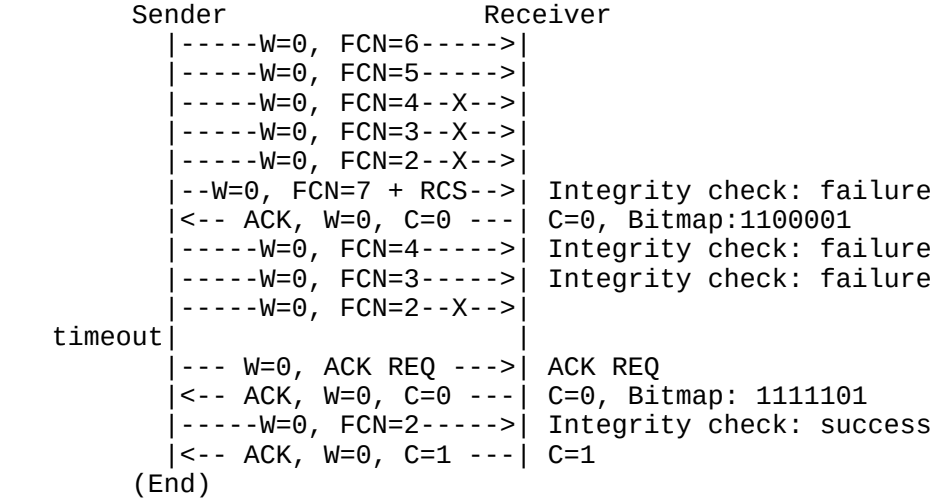
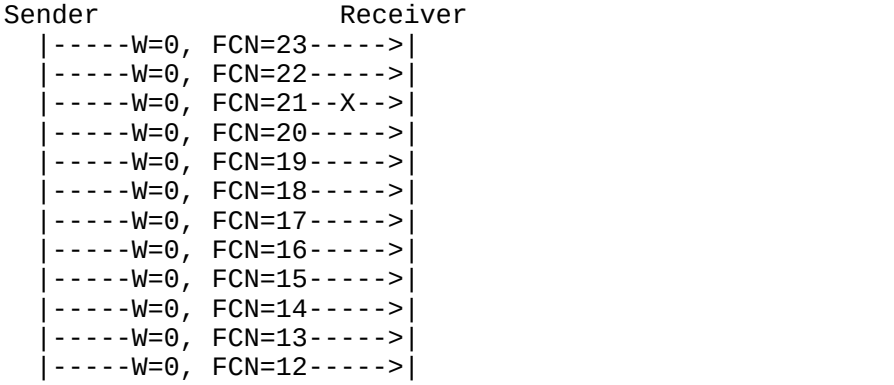


Figure 37: ACK-Always Mode, Six Tiles, Retransmitted SCHC Fragment Lost Again

Figure 38 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 28 tiles, with one tile per SCHC Fragment, N=5, WINDOW_SIZE=24, and two lost SCHC Fragments.



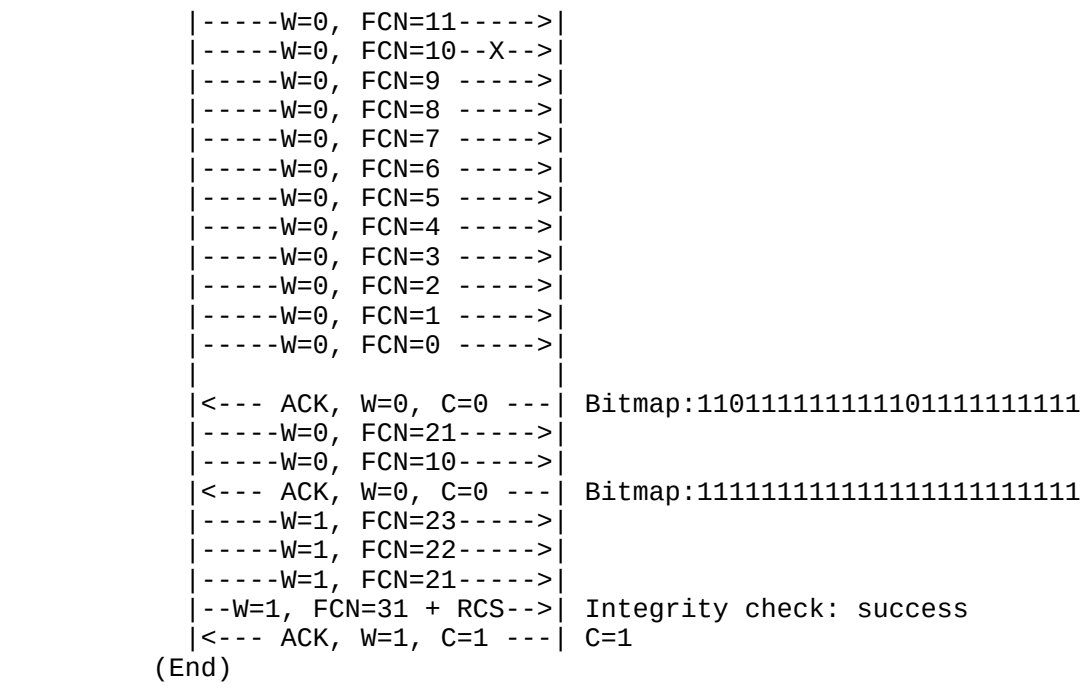


Figure 38: ACK-Always Mode, 28 Tiles, One Tile per SCHC Fragment, Lost SCHC Fragments

Appendix C. Fragmentation State Machines

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

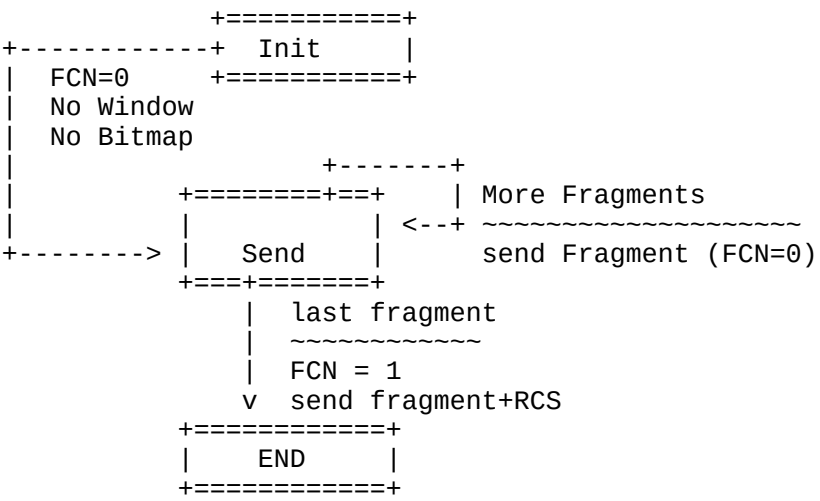


Figure 39: Sender State Machine for the No-ACK Mode

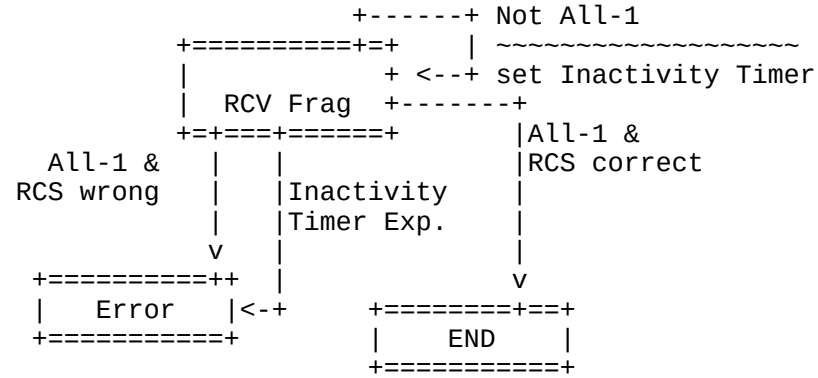


Figure 40: Receiver State Machine for the No-ACK Mode

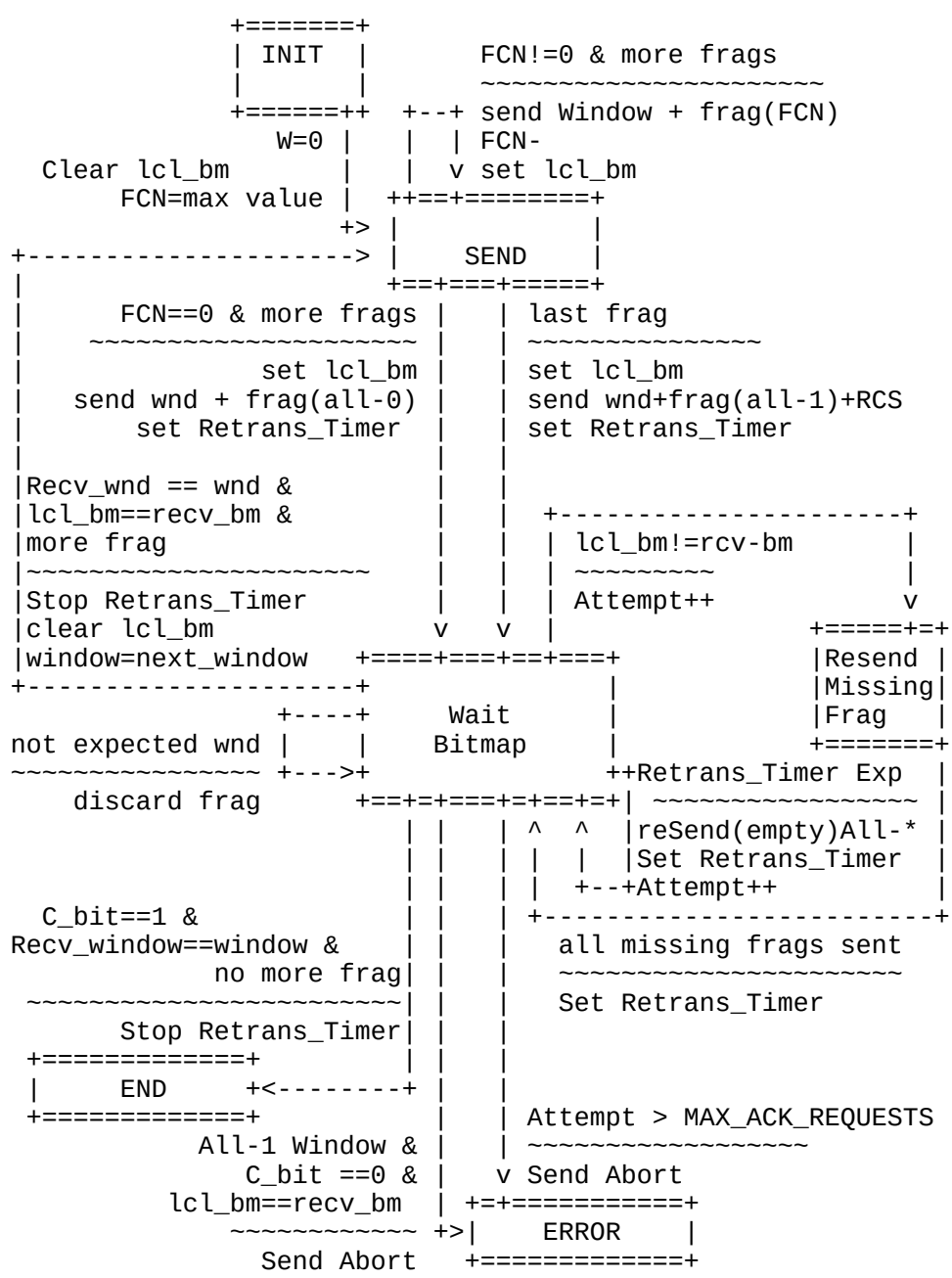
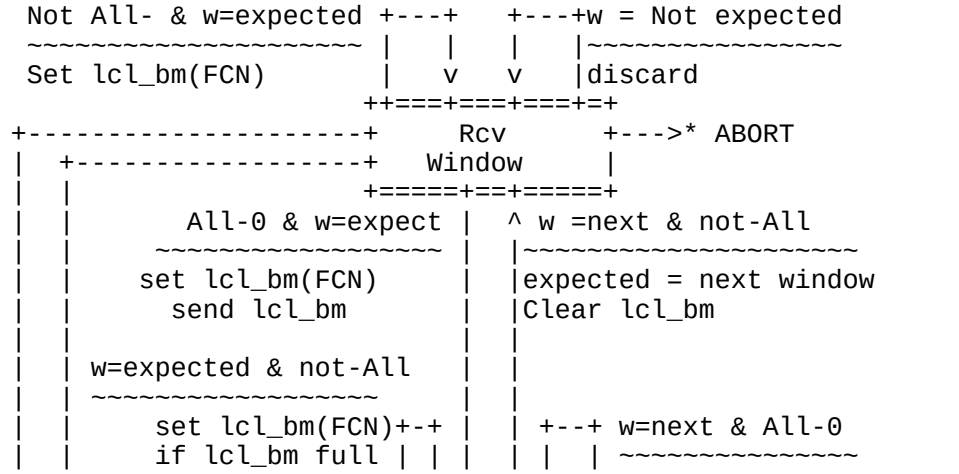
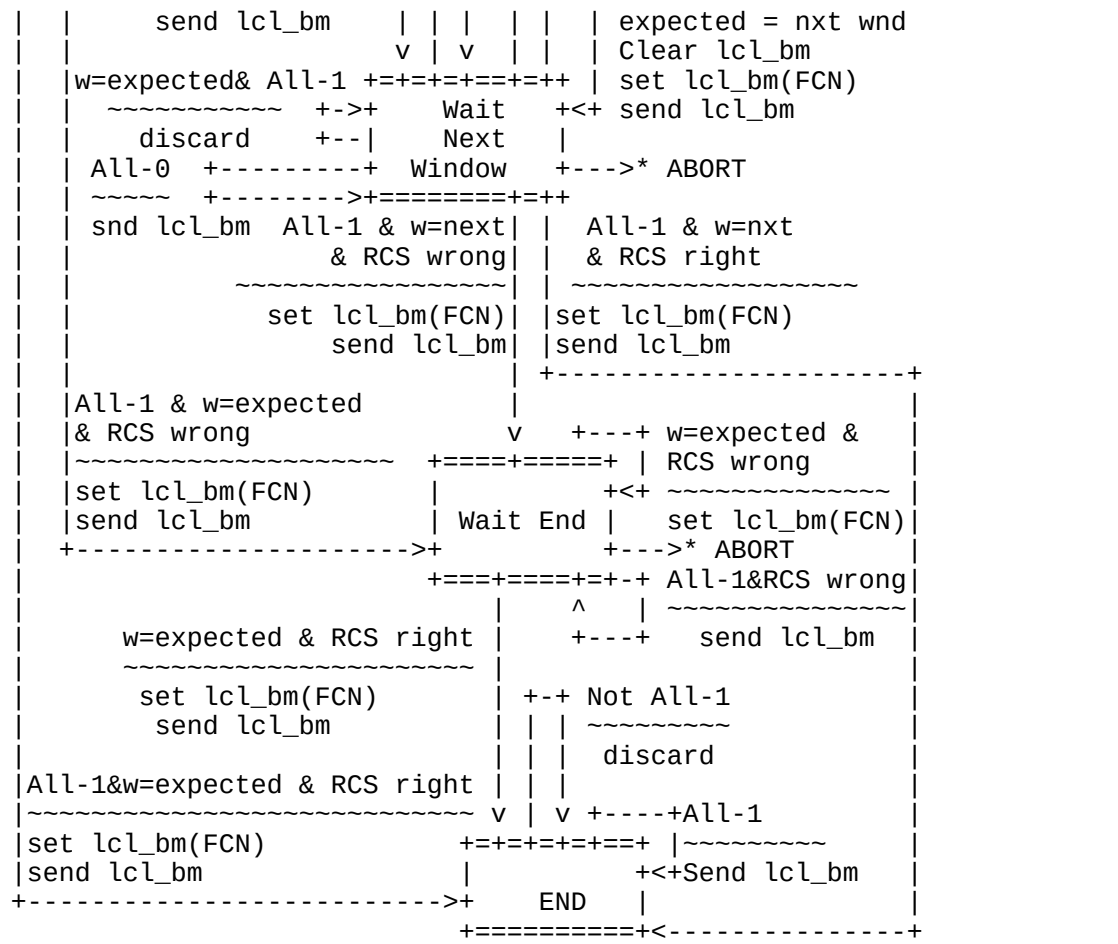


Figure 41: Sender State Machine for the ACK-Always Mode

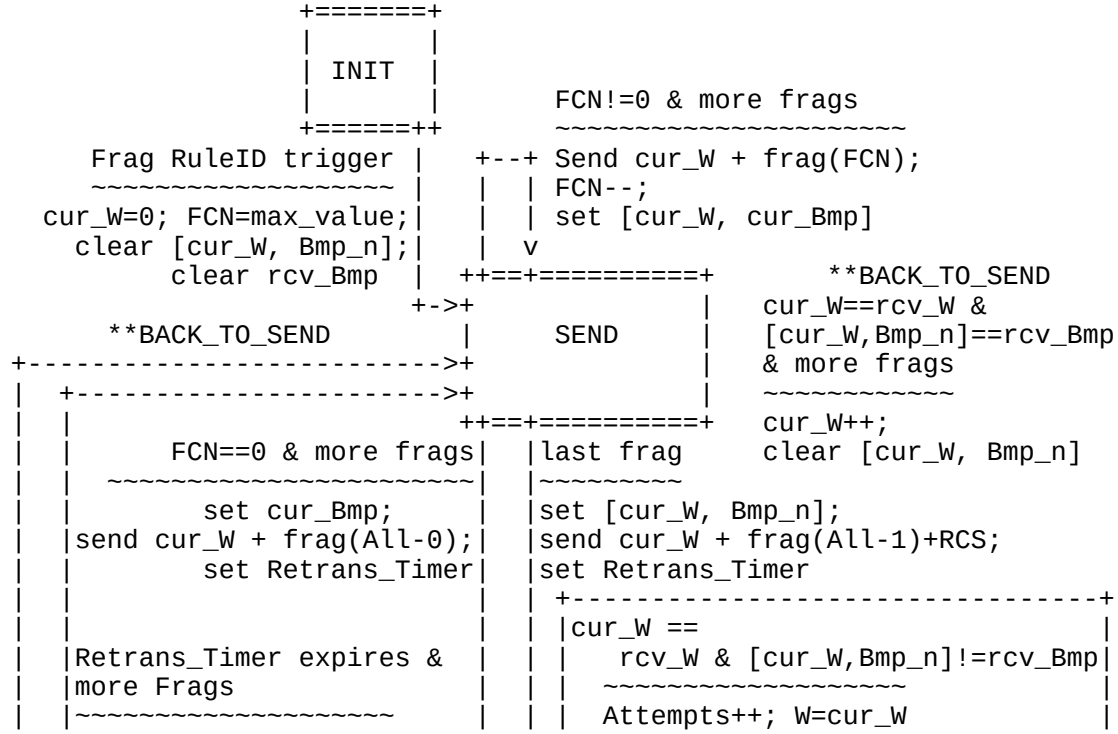




---->* ABORT

In any state
on receiving a SCHC ACK REQ
Send a SCHC ACK for the current window

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



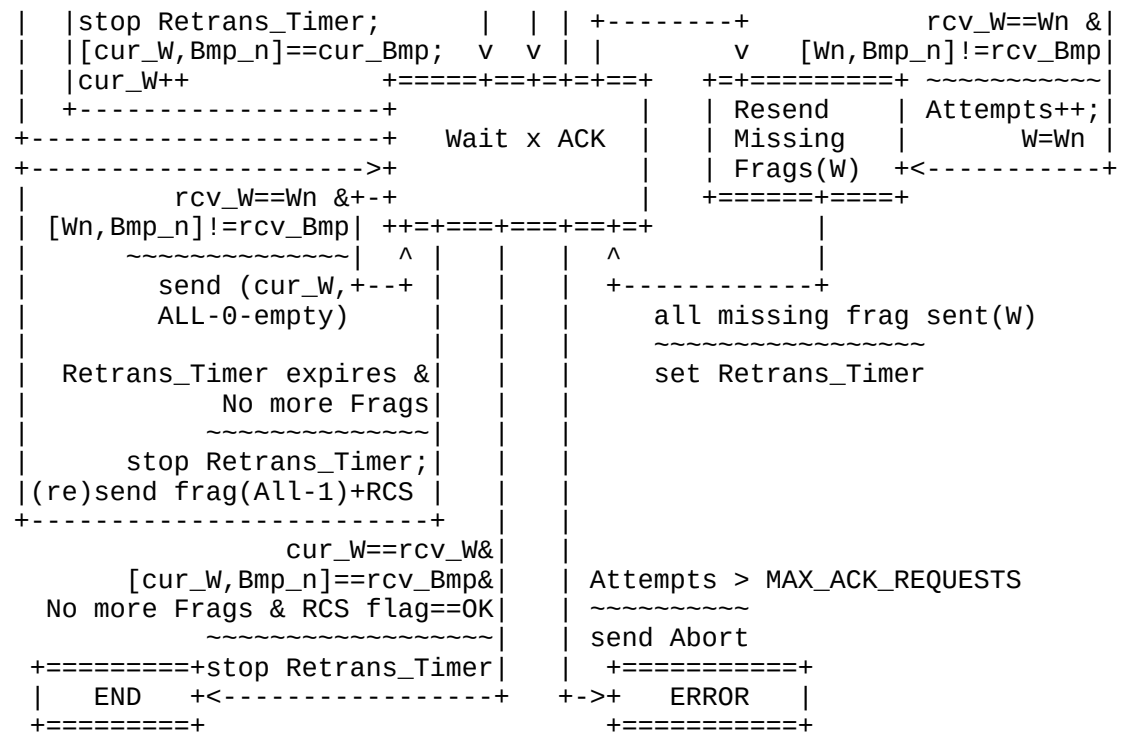


Figure 43: Sender State Machine for the ACK-on-Error Mode

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

Figure 44: Receiver State Machine for the ACK-on-Error Mode

Appendix D. SCHC Parameters

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

* Most common uses cases, deployment scenarios.

* Mapping of the SCHC architectural elements onto the LPWAN architecture.

* Assessment of LPWAN integrity checking.

* Various potential channel conditions for the technology and the corresponding recommended use of SCHC C/D and SCHC F/R.

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

* RuleID numbering scheme, fixed-size or variable-size RuleIDs, number of Rules, the way the RuleID is transmitted.

* maximum packet size that should ever be reconstructed by SCHC decompression (MAX_PACKET_SIZE). See Section 12.

* Padding: size of the L2 Word (for most LPWAN technologies, this would be a byte; for some technologies, a bit).

* Decision to use SCHC fragmentation mechanism or not. If yes, the document must describe:

  - reliability mode(s) used, in which cases (e.g., based on link

channel condition).

- RuleID values assigned to each mode in use.

- presence and number of bits for DTag (T) for each RuleID value, lifetime of DTag at the receiver.

- support for interleaved packet transmission, to what extent.

- WINDOW_SIZE, for modes that use windows.

- number of bits for W (M) for each RuleID value, for modes that use windows.

- number of bits for FCN (N) for each RuleID value, meaning of the FCN values.

- what makes an All-0 SCHC Fragment and a SCHC ACK REQ distinguishable (see Section 8.3.1.1).

- what makes an All-1 SCHC Fragment and a SCHC Sender-Abort distinguishable (see Section 8.3.1.2).

- for RuleIDs that use ACK-on-Error mode: when the last tile of a SCHC Packet is to be sent in a Regular SCHC Fragment, alone in an All-1 SCHC Fragment or with any of these two methods.

- for RuleIDs that use ACK-on-Error mode: if the penultimate tile of a SCHC Packet is of the regular size only or if it can also be one L2 Word shorter.

- for RuleIDs that use ACK-on-Error mode: times at which the sender must listen for SCHC ACKs.

- size of RCS and algorithm for its computation, for each RuleID, if different from the default CRC32. Byte fill-up with zeroes or other mechanism, to be specified. Support for UDP checksum elision.

- Retransmission Timer duration for each RuleID value, if applicable to the SCHC F/R mode.

- Inactivity Timer duration for each RuleID value, if applicable to the SCHC F/R mode.

- MAX_ACK_REQUESTS value for each RuleID value, if applicable to the SCHC F/R mode.

* if L2 Word is wider than a bit and SCHC fragmentation is used, value of the padding bits (0 or 1).

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

* In some LPWAN technologies, as part of energy-saving techniques, Downlink transmission is only possible immediately after an Uplink transmission. In order to avoid potentially high delay in the Downlink transmission of a fragmented SCHC Packet, the SCHC Fragment receiver may perform an Uplink transmission as soon as possible after reception of a SCHC Fragment that is not the last one. Such Uplink transmission may be triggered by the L2 (e.g., an L2 ACK sent in response to a SCHC Fragment encapsulated in a L2 PDU that requires an L2 ACK) or it may be triggered from an upper layer. See Appendix F.

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

  - The way the Contexts are provisioned.

  - The way the Rules are generated.

Appendix E. Supporting Multiple Window Sizes for Fragmentation

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

Appendix F. ACK-Always and ACK-on-Error on Quasi-Bidirectional Links

The ACK-Always and ACK-on-Error modes of SCHC F/R are bidirectional protocols: they require a feedback path from the reassembler to the fragmenter.

Some LPWAN technologies provide quasi-bidirectional connectivity, whereby a Downlink transmission from the Network Infrastructure can only take place right after an Uplink transmission by the Dev.

When using SCHC F/R to send fragmented SCHC Packets Downlink over these quasi-bidirectional links, the following situation may arise: if an Uplink SCHC ACK is lost, the SCHC ACK REQ message by the sender could be stuck indefinitely in the Downlink queue at the Network Infrastructure, waiting for a transmission opportunity.

There are many ways by which this deadlock can be avoided. The Dev application might be sending recurring Uplink messages such as keep-alive, or the Dev application stack might be sending other recurring Uplink messages as part of its operation. However, these are out of the control of this generic SCHC specification.

In order to cope with quasi-bidirectional links, a SCHC-over-foo specification may want to amend the SCHC F/R specification to add a timer-based retransmission of the SCHC ACK. Below is an example of the suggested behavior for ACK-Always mode. Because it is an example, [RFC2119] language is deliberately not used here.

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 UplinkACK Timer) 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 UplinkACK 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 UplinkACK timer for the SCHC ACK is stopped. However, if the UplinkACK timer expires, the SCHC ACK is resent and the UplinkACK timer is reinitialized and restarted.

The default initial value for the UplinkACK Timer, as well as the

maximum number of retries for a specific SCHC ACK, denoted MAX_ACK_REQUESTS, is to be defined in a Profile. The initial value of the UplinkACK 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 finds an opportunity for that transmission. One exception to this recommendation is the special case of the All-1 SCHC Fragment transmission.

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

Acknowledgements

Thanks to (in alphabetical order) Sergio Aguilar Romero, David Black, Carsten Bormann, Deborah Brungard, Brian Carpenter, Philippe Clavier, Alissa Cooper, Roman Danyliw, Daniel Ducuara Beltran, Diego Dujovne, Eduardo Ingles Sanchez, Rahul Jadhav, Benjamin Kaduk, Arunprabhu Kandasamy, Suresh Krishnan, Mirja Kuehlewind, Barry Leiba, Sergio Lopez Bernal, Antoni Markovski, Alexey Melnikov, Georgios Papadopoulos, Alexander Pelov, Charles Perkins, Edgar Ramos, Alvaro Retana, Adam Roach, Shoichi Sakane, Joseph Salowey, Pascal Thubert, and Eric Vyncke for useful design considerations, reviews and comments.

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

Authors' Addresses

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

Email: ana@ackl.io

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

Email: Laurent.Toutain@imt-atlantique.fr

Carles Gomez

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

Email: carlesgo@entel.upc.edu

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

Email: dominique.barthel@orange.com

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
31670 Labege
France

Email: JuanCarlos.Zuniga@sigfox.com