6lo Working Group
Internet-Draft
Intended status: Standards Track
Expires: 12 May 2023

C. Gomez
UPC
A. Minaburo
Acklio

November 2022

Transmission of SCHC-compressed packets over IEEE 802.15.4 networks
draft-gomez-6lo-schc-15dot4-05

Abstract

A framework called Static Context Header Compression and fragmentation (SCHC) has been designed with the primary goal of supporting IPv6 over Low Power Wide Area Network (LPWAN) technologies [RFC8724]. One of the SCHC components is a header compression mechanism. If used properly, SCHC header compression allows a greater compression ratio than that achievable with traditional 6LoWPAN header compression [RFC6282]. For this reason, it may make sense to use SCHC header compression in some 6LoWPAN environments, including IEEE 802.15.4 networks. This document specifies how a SCHC-compressed packet can be carried over IEEE 802.15.4 networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

RFC 6282 is the main specification for IPv6 over Low power Wireless Personal Area Network (6LoWPAN) IPv6 header compression [RFC6282]. This RFC was designed assuming IEEE 802.15.4 as the layer below the 6LoWPAN adaptation layer, and it has also been reused (with proper adaptations) for IPv6 header compression over many other technologies relatively similar to IEEE 802.15.4 in terms of characteristics such as physical layer bit rate, layer 2 maximum payload size, etc. Examples of such technologies comprise BLE, DECT-ULE, ITU G.9959, MS/TP, NFC, and PLC. RFC 6282 provides additional functionality, such as a mechanism for UDP header compression.

In the best cases, RFC 6282 allows to compress a 40-byte IPv6 header down to a 2-byte compressed header (for link-local interactions) or a 3-byte compressed header (when global IPv6 addresses are used). On the other hand, an RFC 6282 compressed UDP header has a typical size of 4 bytes. Therefore, in advantageous conditions, a 48-byte uncompressed IPv6/UDP header may be compressed down to a 6-byte format (when using link-local addresses) or a 7-byte format (for global interactions) by using RFC 6282.

Recently, a framework called Static Context Header Compression (SCHC) has been designed with the primary goal of supporting IPv6 over Low Power Wide Area Network (LPWAN) technologies [RFC8724]. SCHC comprises header compression and fragmentation functionality tailored to the extraordinary constraints of LPWAN technologies, which are more severe than those exhibited by IEEE 802.15.4 or other relatively similar technologies. SCHC header compression allows a greater compression ratio than that of RFC 6282. If used properly, SCHC allows to compress an IPv6/UDP header down to e.g. a single byte. In addition, SCHC can be used to compress Constrained Application Protocol (CoAP) headers as well [RFC7252][RFC8824], which further increases the achievable performance improvement of using SCHC header compression, since there is no 6LoWPAN header compression mechanism defined for CoAP. Therefore, it may make sense to use SCHC header compression in some 6LoWPAN environments [I-D.toutain-6lo-6lo-and-schc], including IEEE 802.15.4 networks, considering its greater efficiency.

If SCHC header compression is added to the panoply of header compression mechanisms used in 6LoWPAN environments, then there is a need to signal when a packet header has been compressed by using SCHC. To this end, the present document specifies a 6LoWPAN Dispatch Type for SCHC header compression [RFC4944].

This document specifies how a SCHC-compressed packet can be carried over IEEE 802.15.4 networks. Note that, as per this document, and while SCHC defines fragmentation mechanisms as well, 6LoWPAN/6Lo fragmentation is used when necessary to transport SCHC-compressed packets over IEEE 802.15.4 networks [RFC4944][RFC8930][RFC8931].

TO-DO: indicate here any specific updates of previous RFCs due to this specification.

2. Terminology

2.1. Requirements language

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 BCP14 [RFC2119], [RFC8174], when, and only when, they appear in all capitals, as shown here.

2.2. Background on previous specifications

The reader is expected to be familiar with the terms and concepts defined in specifications of RPL [RFC6550] and companion documents [RFC6553][RFC6554][RFC9008], 6LoWPAN Routing Header [RFC8138], and SCHC [RFC8724].

RFC 8724 defines the Rule concept, whereby a Rule may be used to support header compression or fragmentation functionality. In the present document, Rules are only used for header compression.

3. Architecture

3.1. Protocol stack

The traditional 6LoWPAN-based protocol stack for constrained devices (Figure 1, left) places the 6LoWPAN adaptation layer between IPv6 and an underlying technology such as IEEE 802.15.4. Suitable upper layer protocols include CoAP [RFC7252] and UDP. (Note that, while CoAP has also been specified over TCP, and TCP may play a significant role in IoT environments [RFC9006], 6LoWPAN header compression has not been defined for TCP.)

6LoWPAN can be envisioned as a set of two main sublayers, where the upper one provides header compression, while the lower one offers fragmentation.

This document defines an alternative approach for packet header compression over IEEE 802.15.4, which leads to a modified protocol stack (Figure 1, right). Fragmentation functionality remains the one defined by 6LoWPAN [RFC4944] and 6Lo [RFC8930][RFC8931].

```
        +------------+         +------------+
        | CoAP, other|         | CoAP, other|
        +------------+         +------------+
        | UDP, other |         | UDP, other |
        +------------+         +------------+
        |    IPv6    |         |    IPv6    |
        +------------+         +------------+
        | 6LoWPAN HC |         |  SCHC HC   |  <-- NEW
        +------------+         +------------+
        |6LoWPAN Frag|         |6LoWPAN Frag|
        +------------+         +------------+
        |  802.15.4  |         |  802.15.4  |
        +------------+         +------------+
```

Figure 1: Traditional 6LoWPAN-based protocol stack over IEEE 802.15.4 (left) and alternative protocol stack using SCHC for header compression (right). HC and Frag stand for Header Compression and Fragmentation, respectively.

SCHC header compression may be applied to the headers of different protocols or sets of protocols. Some examples include: i) IPv6 packet headers, ii) joint IPv6 and UDP packet headers, iii) joint IPv6, UDP and CoAP packet headers, etc.

3.2. Network topologies

IEEE 802.15.4 supports two main network topologies: the star topology, and the peer-to-peer (i.e., mesh) topology.

SCHC has been designed for LPWAN technologies, which are typically based on a star topology where constrained devices (e.g., sensors) communicate with a less constrained, central network gateway [RFC 8376]. However, as stated in [draft-ietf-lpwan-architecture], SCHC is generic and it can also be used in networking environments beyond the ones originally considered for SCHC.

SCHC compression is applicable to both star topology and mesh topology IEEE 802.15.4 networks.

3.3. Multihop communication

6LoWPAN defines two approaches for multihop communication: Route-Over and Mesh-Under [RFC6606]. In Route-Over, routing is performed at the IP layer. In Mesh-Under, routing functionality is located at the adaptation layer, below IP. This section describes how SCHC-compressed packets are transmitted over a multihop IEEE 802.15.4 network, for both Route-Over and Mesh-Under.

3.3.1. Straightforward Route-Over approach

SCHC header compression MAY be used in a Route-Over network in a straightforward approach, whereby all network nodes MUST store all the Rules in use by any nodes in the network. In this case, 6LoWPAN routers are able to decompress (if needed) received packet headers and compress packet headers before being forwarded.

The frame format to be used to carry a SCHC-compressed packet in the straightforward Route-Over approach is described in Section 4.1.

3.3.2. Tunneled, RPL-based Route-Over approach

In a Route-Over network that uses the IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6550], the RPL non-storing mode [RFC6550, RFC 6554] and [RFC8138] MAY be exploited in order to efficiently transmit SCHC-compressed packets. In this approach, packets sent by a 6LN are tunneled to the root, and packets intended for 6LNs are tunneled from the root. Traffic between two 6LNs traverses an Upward tunnel to the root and a Downward tunnel from the root.

In this approach, a network node MUST store the Rules defined for its communication with other endpoints. A 6LR is thus relieved to store Rules used by pairs of endpoints that do not include the 6LR itself. A 6LBR MUST store all the Rules used by all nodes in the network.

RFC 9008 describes how the communication between a 6LN and another endpoint (another 6LN or the root of the same RPL domain, or an external node, e.g., on the Internet) is performed. In RPL non-storing mode, for Downward traffic, the root adds a source-routing header. The root also performs IPv6-in-IPv6 encapsulation, except when the root itself is the packet source. The IPv6-in-IPv6 encapsulation terminates at the 6LN (if it is a RAL) or at the last 6LR (if the 6LN is a RUL). For Upward traffic, IPv6-in-IPv6 encapsulation is performed by the first 6LR when the 6LN is a RUL that sends a packet to an external node or to another 6LN in the same RPL domain, but not to the root. When the 6LN is a RAL that sends packets to the same destinations, IPv6-in-IPv6 encapsulation may be performed (by the RAL). The destination in the outer header of the IPv6-in-IPv6 encapsulation for Upward traffic is the root.

This document updates RFC 9008 by specifying that, in the tunneled, RPL-based Route-Over approach, when a 6LN transmits an IPv6 packet whose header is compressed by means of SCHC instead of 6LoWPAN header compression (RFC 6282), the SCHC-compressed packet MUST be tunneled by means of IPv6-in-IPv6 encapsulation up to the root. This applies regardless of the inner, SCHC-compressed packet destination.

(TO-DO: address the case when the 6LN is a RUL.)

For the sake of efficiency, RFC 8138 MUST be used to compress IPv6-in-IPv6 headers, the RPL Option (RFC 6553) and the source routing header (RPL Routing Header type 3, RFC 6554).

The frame format to be used to carry a SCHC-compressed packet in the tunneled, RPL-based Route-Over approach is described in Section 4.3.

3.3.3. Mesh-Under approach

When SCHC header compression is used in a Mesh-Under network, Mesh-Under operates as described in RFC 4944. The frame format to be used to carry a SCHC-compressed packet in the Mesh-Under approach is described in Section 4.3.

For header compression in a Mesh-Under network, a network node MUST store the Rules defined for its communication with other endpoints.

In this case, a RuleID MAY be reused across disjoint pairs of endpoints, to identify different Rules used by such disjoint pairs of endpoints, at the expense of increased RuleID management and device configuration complexity.

4. Frame Format

This section defines the frame format to be used when a SCHC-compressed packet is carried over IEEE 802.15.4. Such format is carried as IEEE 802.15.4 frame payload.

4.1. Single-hop or straightforward Route-Over frame format

This subsection defines the frame format for carrying SCHC-compressed packets over IEEE 802.15.4 for single-hop communication or when the straightforward Route-Over approach (see 3.3.1) is used. This format comprises a SCHC Dispatch Type, a SCHC Packet (i.e. a SCHC-compressed packet (RFC 8724), and Padding bits, if any). Figure 2 illustrates the described frame format.

```
<---------- IEEE 802.15.4 frame payload ---------->

                <----- SCHC Packet ----->
+---------------+-------------+---------+ - - - - +
| SCHC Dispatch | SCHC Header | Payload | Padding |
+---------------+-------------+---------+ - - - - +
```

Figure 2: Encapsulated, SCHC-compressed packet, for single-hop or straightforward Route-Over transmission. Padding bits are added if needed.

4.1.1. SCHC Dispatch

Adding SCHC header compression to the panoply of header compression mechanisms used in 6LoWPAN/6Lo environments creates the need to signal when a packet header has been compressed by using SCHC. To this end, the present document specifies the SCHC Dispatch. The SCHC Dispatch indicates that the next field in the frame format is a SCHC-compressed header (SCHC Header in Figure 2, see 4.2)).

This document defines the SCHC Dispatch as a 6LoWPAN Dispatch Type for SCHC header compression [RFC4944]. With the aim to minimize overhead, the present document allocates a 1-byte pattern in Page 0 [RFC8025] for the SCHC Dispatch Type:

SCHC Dispatch Type bit pattern: 01000100 (Page 0) (Note: to be confirmed by IANA))

4.1.2. SCHC Header

SCHC Header (Figure 2) corresponds to a packet header that has been compressed by using SCHC. As defined in [RFC8724], the SCHC Header comprises a RuleID, and a compression residue. As per the present specification, a RuleID size between 1 and 16 bits is RECOMMENDED. In order to decide the RuleID size to be used in a network, the trade-off between (compressed) header overhead and the number of Rules needs to be carefully assessed.

4.1.3. Padding

If SCHC header compression leads to a SCHC Packet size of a non-integer number of bytes, padding bits of value equal to zero MUST be appended to the SCHC Packet as appropriate to align to an octet boundary.

4.2. Tunneled, RPL-based Route-Over frame format

This subsection defines the frame formats for carrying SCHC-compressed packets over IEEE 802.15.4 in the tunneled, RPL-based Route-Over approach (see 3.3.2). Such formats are based on RFC 8138; however, instead of RFC 6282 header compression, this specification uses SCHC header compression. Accordingly, this specification updates RFC 8138 by stating that a 6LoRH header MUST always be placed before the LOWPAN_IPHC as defined in RFC 6282 [RFC6282] or the SCHC Dispatch, followed by the SCHC-compressed packet, as defined in the present specification.

Since 6LoRH uses Dispatch Types in Page 1, the present specification also defines a SCHC Dispatch Type in Page 1, with the same bit pattern as the one in Page 0: 01000100 (to be confirmed by IANA).

In the tunneled, RPL-based Route-Over frame formats, the SCHC-compressed header is preceded by the SCHC Dispatch (in this case, in Page 1).

The frame format for Downward transmission is shown in Figure 3:

```
<----------------- IEEE 802.15.4 frame payload ---------------------->

                                                 <- SCHC pkt ->
+-- ... -+-- ... --+- ... -+--- ... --+---- ... -+-----+-------+ - - +
|11110001|SRH-6LoRH| RPI-  | IP-in-IP | 01000100 |SCHC |payload| pad |
|Page 1  |         | 6LoRH |  6LoRH   |SCHCDsptch| hdr |       |     |
+-- ... -+-- ... --+- ... -+--- ... --+---- ... -+-----+-------+ - - +
                                        (Page 1)

                                      <----- This specification ----->
```

Figure 3: Downward frame format for SCHC-compressed packets in the tunneled, RPL-based Route-Over approach.

The frame format for Upward transmission is shown in Figure 4 (note that it does not include the source routing header that is present in the Downward frame format):

```
<------------- IEEE 802.15.4 frame payload ---------------->

                                       <- SCHC pkt ->
+-- ... -+- ... -+--- ... --+---- ... -+-----+-------+ - - +
|11110001| RPI-  | IP-in-IP | 01000100 |SCHC |payload| pad |
|Page 1  | 6LoRH |  6LoRH   |SCHCDsptch| hdr |       |     |
+-- ... -+- ... -+--- ... --+---- ... -+-----+-------+ - - +
                              (Page 1)

                            <----- This specification ----->
```

Figure 4: Upward frame format for SCHC-compressed packets in the tunneled, RPL- based Route-Over approach.

4.3. Mesh-Under frame format

This subsection describes the frame format for carrying SCHC-compressed packets over IEEE 802.15.4 in the Mesh-Under approach (see 3.3.3).

TO-DO: show the formats, based on RFC 4944, but with SCHC-compressed headers.

5. SCHC compression for IPv6, UDP, and CoAP headers

SCHC header compression may be applied to the headers of different protocols or sets of protocols. Some examples include: i) IPv6 packet headers, ii) joint IPv6 and UDP packet headers, iii) joint IPv6, UDP and CoAP packet headers, etc.

Each Rule defines the set of protocols whose headers are compressed. For example, in a given deployment, RuleIDs 1 to 3 may be defined for IPv6 header compression only, RuleIDs 4 to 7 may be used for IPv6/UDP header compression, and RuleIDs 8 to 15 may be used for IPv6/UDP/CoAP header compression.

This section describes how IPv6, UDP, and CoAP header fields are compressed.

5.1. SCHC compression for IPv6 and UDP headers

IPv6 and UDP header fields MUST be compressed as per Section 10 of RFC 8724.

IPv6 addresses are split into two 64-bit-long fields; one for the prefix and one for the Interface Identifier (IID).

To allow for a single Rule being used for both directions, RFC 8724 identifies IPv6 addresses and UDP ports by their role (Dev or App) and not by their position in the header (source or destination). This optimization can be used as is in some IEEE 802.15.4 networks (e.g., an IEEE 802.15.4 star topology where the peripheral devices (Devs) send/receive packets to/from a network-side entity (App)).

However, in some types of 6LoWPAN environments (e.g., when a sender and its destination are both peer nodes in a mesh topology network), additional functionality is needed to allow use of the Dev and App roles for C/D. In this case, each SCHC C/D entity needs to know its role (Dev or App) in addition to the Rule(s), and corresponding

RuleIDs, for each endpoint it communicates with before such communication occurs [I-D.ietf-lpwan-architecture]. In such cases, the terms Uplink and Downlink that have been defined in RFC 8724 need to be understood in the context of each specific pair of endpoints.

5.1.1. Compression of IPv6 addresses

Compression of IPv6 source and destination prefixes MUST be performed as per Section 10.7.1 of RFC 8724. Additional guidance is given in the present section.

Compression of IPv6 source and destination IIDs MUST be performed as per Section 10.7.2 of RFC 8724. One particular consideration when SCHC C/D is used in IEEE 802.15.4 networks is that, in contrast with some LPWAN technologies, IEEE 802.15.4 data frame headers include both source and destination fields. If the Dev or App IID are based on an L2 address, in some cases the IID can be reconstructed with information coming from the L2 header. Therefore, in those cases, DevIID and AppIID CDAs can be used.

5.2. SCHC compression for CoAP headers

CoAP header fields MUST be compressed as per Sections 4 to 6 of RFC 8824. Additional guidance is given in this section.

For CoAP header compression/decompression, the SCHC Rules description uses direction information in order to reduce the number of Rules needed to compress headers.

As stated in 5.1, in some types of 6LoWPAN environments (e.g., when a sender and its destination are both peer nodes in a mesh topology network), each SCHC C/D entity needs to know its role (Dev or App), in addition to the Rule(s), and corresponding RuleIDs, for each endpoint it communicates with before such communication occurs [I-D.ietf-lpwan-architecture]. Therefore, in such cases, direction information will be specific to each pair of endpoints.

6. Fragmentation and reassembly

After applying SCHC header compression to a packet intended for transmission, if the size of the resulting SCHC Packet (Section 4) exceeds the IEEE 802.15.4 frame payload space available, such SCHC Packet MUST be fragmented, carried and reassembled by means of the fragmentation and reassembly functionality defined by 6LoWPAN [RFC4944] or 6Lo [RFC8930][RFC8931].

In a Route-Over multihop network, the 6LoWPAN fragment forwarding technique called Virtual Reassembly Buffer (VRB) [RFC8930] SHOULD be used. However, VRB might not be the best approach for a particular network, e.g., if at least one of the caveats described in Section 6 of RFC 8930 is unacceptable or cannot be addressed.

7. IANA Considerations

This document requests the allocation of the Dispatch Type Field bit pattern 01000100 (in Pages 0 and 1) as SCHC Dispatch Type.

8. Security Considerations

This document does not define SCHC header compression functionality beyond the one defined in RFC 8724. Therefore, the security considerations in section 12.1 of RFC 8724 and in section 9 of RFC 8824 apply.

As a safety measure, a SCHC decompressor implementing the present specification MUST NOT reconstruct a packet larger than 1500 bytes [RFC8724].

IEEE 802.15.4 networks support link-layer security mechanisms such as encryption and authentication. As in RFC 8824, the use of a cryptographic integrity-protection mechanism to protect the SCHC headers is REQUIRED.

9. Acknowledgments

Ana Minaburo and Laurent Toutain suggested for the first time the use of SCHC in environments where 6LoWPAN has traditionally been used. Laurent Toutain, Pascal Thubert, Dominique Barthel, Guangpeng Li, and Carsten Bormann made comments that helped shape this document.

Carles Gomez has been funded in part by the Spanish Government through project PID2019-106808RA-I00, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

10. References

10.1. Normative References

[I-D.ietf-lpwan-architecture]
Pelov, A., Thubert, P., and A. Minaburo, "LPWAN Static Context Header Compression (SCHC) Architecture", Work in Progress, Internet-Draft, draft-ietf-lpwan-architecture-02, 30 June 2022, <https://www.ietf.org/archive/id/draft-ietf-lpwan-architecture-02.txt>.

[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>.

[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>.

[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>.

[RFC6550] Winter, T., Ed., Thubert, P., Ed., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, JP., and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks", RFC 6550, DOI 10.17487/RFC6550, March 2012, <https://www.rfc-editor.org/info/rfc6550>.

[RFC6553] Hui, J. and JP. Vasseur, "The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams", RFC 6553, DOI 10.17487/RFC6553, March 2012, <https://www.rfc-editor.org/info/rfc6553>.

[RFC6554] Hui, J., Vasseur, JP., Culler, D., and V. Manral, "An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL)", RFC 6554, DOI 10.17487/RFC6554, March 2012, <https://www.rfc-editor.org/info/rfc6554>.

[RFC6606] Kim, E., Kaspar, D., Gomez, C., and C. Bormann, "Problem Statement and Requirements for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing", RFC 6606, DOI 10.17487/RFC6606, May 2012, <https://www.rfc-editor.org/info/rfc6606>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8025] Thubert, P., Ed. and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Paging Dispatch", RFC 8025, DOI 10.17487/RFC8025, November 2016, <https://www.rfc-editor.org/info/rfc8025>.

[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>.

[RFC8138] Thubert, P., Ed., Bormann, C., Toutain, L., and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header", RFC 8138, DOI 10.17487/RFC8138, April 2017, <https://www.rfc-editor.org/info/rfc8138>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

[RFC8930] Watteyne, T., Ed., Thubert, P., Ed., and C. Bormann, "On Forwarding 6LoWPAN Fragments over a Multi-Hop IPv6 Network", RFC 8930, DOI 10.17487/RFC8930, November 2020, <https://www.rfc-editor.org/info/rfc8930>.

[RFC8931] Thubert, P., Ed., "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Selective Fragment Recovery", RFC 8931, DOI 10.17487/RFC8931, November 2020, <https://www.rfc-editor.org/info/rfc8931>.

[RFC9008] Robles, M.I., Richardson, M., and P. Thubert, "Using RPI Option Type, Routing Header for Source Routes, and IPv6-in-IPv6 Encapsulation in the RPL Data Plane", RFC 9008, DOI 10.17487/RFC9008, April 2021, <https://www.rfc-editor.org/info/rfc9008>.

10.2. Informative References

[I-D.toutain-6lo-6lo-and-schc]
Minaburo, A. and L. Toutain, "Comparison of 6lo and SCHC", Work in Progress, Internet-Draft, draft-toutain-6lo-6lo-and-schc-00, 4 November 2019, <https://www.ietf.org/archive/id/draft-toutain-6lo-6lo-and-schc-00.txt>.

[RFC9006] Gomez, C., Crowcroft, J., and M. Scharf, "TCP Usage Guidance in the Internet of Things (IoT)", RFC 9006, DOI 10.17487/RFC9006, March 2021, <https://www.rfc-editor.org/info/rfc9006>.

Appendix A. Header compression examples

TO-DO: provide examples for IPv6-only, IPv6/UDP and IPv6/UDP/CoAP.

Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: carles.gomez@upc.edu

Ana Minaburo
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: ana@ackl.io

lpwan Working Group JC. Zuniga
Internet-Draft Cisco
Updates: 8724, 9363 (if approved) C. Gomez
Intended status: Standards Track S. Aguilar
Expires: 7 October 2023 Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
S. Cespedes
Concordia University
D. Wistuba
NIC Labs, Universidad de Chile
5 April 2023

SCHC Compound ACK
draft-ietf-lpwan-schc-compound-ack-17

Abstract

The present document updates the SCHC (Static Context Header Compression and fragmentation) protocol RFC8724 and the corresponding Yang Module RFC9363. It defines a SCHC Compound ACK message format and procedure, which are intended to reduce the number of response transmissions (i.e., SCHC ACKs) in the ACK-on-Error mode, by accumulating bitmaps of several windows in a single SCHC message (i.e., the SCHC Compound ACK).

Both message format and procedure are generic, so they can be used, for instance, by any of the four Low Power Wide Area Networks (LPWANs) technologies defined in RFC8376, being Sigfox, LoRaWAN, NB-IoT and IEEE 802.15.4w.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Generic Framework for Static Context Header Compression and Fragmentation (SCHC) specification [RFC8724] describes two mechanisms: i) a protocol header compression scheme, and ii) a frame fragmentation and loss recovery functionality. Either can be used on top of radio technologies such as the four Low Power Wide Area Networks (LPWANs) listed in [RFC8376], being Sigfox, LoRaWAN, NB-IoT and IEEE 802.15.4w. These LPWANs have similar characteristics such as star-oriented topologies, network architecture, connected devices with built-in applications, etc.

SCHC offers a great level of flexibility to accommodate all these LPWAN technologies. Even though there are a great number of similarities between them, some differences exist with respect to the transmission characteristics, payload sizes, etc. Hence, there are optimal parameters and modes of operation that can be used when SCHC is used on top of a specific LPWAN technology.

In ACK-on-Error mode in [RFC8724] the SCHC Packet is fragmented into pieces called tiles, with all tiles of the same size except for the last one, which can be smaller. Successive tiles are grouped in windows of fixed size. A SCHC Fragment carries one or several contiguous tiles, which may span multiple windows. When sending all tiles from all windows, the last tile is sent in an All-1 SCHC Fragment. The SCHC receiver, after receiving the All-1 SCHC Fragment will send a SCHC ACK reporting on the reception of exactly one window of tiles. In case of SCHC Fragment losses, a bitmap is added to the failure SCHC ACK, where each bit in the bitmap corresponds to a tile in the window. If SCHC Fragment losses span multiple windows, the SCHC receiver will send one failure SCHC ACK per window with losses.

The present document updates the SCHC protocol for frame fragmentation and loss recovery. It defines a SCHC Compound ACK format and procedure, which is intended to reduce the number of response transmissions (i.e., SCHC ACKs) in the ACK-on-Error mode of SCHC. The SCHC Compound ACK extends the failure SCHC ACK message format so that it can contain several bitmaps, each bitmap being identified by its corresponding window number. The SCHC Compound ACK is backwards compatible with the SCHC ACK as defined in [RFC8724], and introduces flexibility, as the receiver has the capability to respond to the All-0 SCHC Fragment, providing more downlink opportunities, and therefore adjusting to the delay requirements of the application.

2. Terminology

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.

It is assumed that the reader is familiar with the terms and mechanisms defined in [RFC8376] and in [RFC8724].

3. SCHC Compound ACK

The SCHC Compound ACK is a failure SCHC ACK message that can contain several bitmaps, each bitmap being identified by its corresponding window number. In [RFC8724], the failure SCHC ACK message only contain one bitmap corresponding to one window. The SCHC Compound ACK extends this format allowing more windows to be acknowledged in a single ACK, reducing the total number of failure SCHC ACK messages, specially when fragment losses are present in intermediate windows.

The SCHC Compound ACK MAY be used in fragmentation modes that use windows and that allow reporting the bitmaps of multiple windows at the same time, and MUST NOT be used otherwise.

The SCHC Compound ACK:

* provides feedback only for windows with fragment losses,

* has a variable size that depends on the number of windows with fragment losses being reported in the single Compound SCHC ACK,

* includes the window number (i.e., W) of each bitmap,

* might not cover all windows with fragment losses of a SCHC Packet,

* and is distinguishable from the SCHC Receiver-Abort.

3.1. SCHC Compound ACK Message Format

Figure 1 shows the success SCHC ACK format, i.e., when all fragments have been correctly received (C=1), as defined in [RFC8724].

```
|-- SCHC ACK Header --|
         |--T-|---M--| 1 |
+--------+----+------+---+~~~~~~~~~~~~~~~~~~
| RuleID |DTag|   W  |C=1| padding as needed
+--------+----+------+---+~~~~~~~~~~~~~~~~~~
```

Figure 1: SCHC Success ACK message format, as defined in RFC8724

In case SCHC Fragment losses are found in any of the windows of the SCHC Packet, the SCHC Compound ACK MAY be used. The SCHC Compound ACK message format is shown in Figure 2 and Figure 3.

```
|--- SCHC ACK Header --|- W=w1 -|...|---- W=wi -----|
       |--T-|---M--|-1-|        |...|---M--|        |---M--|
+------+----+------+---+--------+...+------+--------+------+~~~~~+
|RuleID|DTag| W=w1 |C=0| Bitmap |...| W=wi | Bitmap |00..00| pad |
+------+----+------+---+--------+...+------+--------+------+~~~~~+
                         next L2 Word boundary ->|<-- L2 Word ->|

   Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 2: SCHC Compound ACK message format

The SCHC Compound ACK groups the window number (W) with its corresponding bitmap. Window numbers do not need to be contiguous. However, the window numbers and its corresponding bitmaps included in the SCHC Compound ACK message MUST be ordered from the lowest-numbered to the highest-numbered window. Hence, if the bitmap of window number zero is present in the SCHC Compound ACK message, it MUST always be the first one in order and its W number MUST be placed in the SCHC ACK Header.

If M or more padding bits would be needed after the last bitmap in the message to fill the last L2 Word, M bits at 0 MUST be appended after the last bitmap, and then padding is applied as needed (see Figure 2). Since window number 0, if present in the message, is placed as w1, the M bits set to zero can't be confused with window number 0, and therefore they signal the end of the SCHC Compound ACK message.

Figure 3 shows the case when the required padding bits are strictly less than M bits. In this case, the layer-2 MTU (Maximum Transmission Unit) does not leave room for any extra window value, let alone any bitmap, thereby signaling the end of the SCHC Compound ACK message.

```
|--- SCHC ACK Header --|- W=w1 -|...|---- W=wi -----|
       |--T-|---M--|-1-|        |...|---M--|        |---M--|
+------+----+------+---+--------+...+------+--------+~~~+
|RuleID|DTag| W=w1 |C=0| Bitmap |...| W=wi | Bitmap |pad|
+------+----+------+---+--------+...+------+--------+~~~+
                                next L2 Word boundary ->|
   Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 3: SCHC Compound ACK message format with less than M padding bits

The SCHC Compound ACK MUST NOT use the Compressed Bitmap format for intermediate windows/bitmaps (i.e., bitmaps that are not the last one of the SCHC Compound ACK message), and therefore intermediate bitmaps fields MUST be of size WINDOW_SIZE. Hence, the SCHC Compound ACK MAY use a Compressed Bitmap format only for the last bitmap in the message. The optional usage of this Compressed Bitmap for the last bitmap MUST be specified by the SCHC technology-specific profile.

The case where the last bitmap is effectively compressed corresponds to Figure 3, with the last bitmap ending, by construction, on an L2 Word boundary, therefore resulting in no padding at all.

Figure 4 illustrates a bitmap compression example of a SCHC Compound ACK, where the bitmap of the last window (wi) indicates that the first tile has not been correctly received. Because the compression algorithm resulted in effective compression, no padding is needed.

```
|--- SCHC ACK Header --|- W=w1 -|...|-------- W=wi -------|
       |--T-|---M--|-1-|        |...|---M--|
+------+----+------+---+--------+...+------+--------------+
|RuleID|DTag| W=w1 |C=0| Bitmap |...| W=wi |0 1 1 1 1 1 1 |
+------+----+------+---+--------+...+------+--------------+
                        next L2 Word boundary ->|

                SCHC Compound ACK with uncompressed Bitmap

|--- SCHC ACK Header --|- W=w1 -|...|-- W=wi --|
       |--T-|---M--|-1-|        |...|---M--|
+------+----+------+---+--------+...+------+---+
|RuleID|DTag| W=w1 |C=0| Bitmap |...| W=wi |0 1|
+------+----+------+---+--------+...+------+---+
                        next L2 Word boundary ->|

       Transmitted SCHC Compound ACK with compressed Bitmap

  Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 4: SCHC Compound ACK message format with compressed bitmap

Figure 5 illustrates another bitmap compression example of a SCHC Compound ACK, where the bitmap of the last window (wi) indicates that the second and the fourth tile have not been correctly received. In this example, the compression algorithm does not result in effective compression of the last bitmap. Besides, because more than M bits of padding would be needed to fill the last L2 Word, M bits at 0 are appended to the message before padding is applied.

```
   |--- SCHC ACK Header --|-W=w1-|...|-------- W=wi -------|
          |--T-|---M--|-1-|      |...|---M--|
   +------+----+------+---+------+...+------+--------------+
   |RuleID|DTag| W=w1 |C=0|Bitmap|...| W=wi |1 0 1 0 1 1 1 |
   +------+----+------+---+------+...+------+--------------+
                        next L2 Word boundary ->|
                     SCHC Compound ACK with uncompressed Bitmap

   |--- SCHC ACK Header --|-W=w1-|...|-------- W=wi -------|
          |--T-|---M--|-1-|      |...|---M--|              |---M--|
   +------+----+------+---+------+...+------+--------------+------+~~~+
   |RuleID|DTag| W=w1 |C=0|Bitmap|...| W=wi |1 0 1 0 1 1 1 |00..00|pad|
   +------+----+------+---+------+...+------+--------------+------+~~~+
                        next L2 Word boundary ->|<------ L2 Word ------>|
                      Transmitted SCHC Compound ACK

        Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 5: SCHC Compound ACK message format with compressed bitmap

If a SCHC sender gets a SCHC Compound ACK with invalid W's, such as duplicate W values or W values not sent yet, it MUST discard the whole SCHC Compound ACK message.

Note: because it has a C bit reset to 0, the SCHC Compound ACK is distinguishable from the Receiver-Abort message [RFC8724], which has a C bit set to 1.

3.2. SCHC Compound ACK Behaviour

The SCHC ACK-on-Error behaviour is described in section 8.4.3 of [RFC8724]. The present document slightly modifies this behaviour, since in the baseline SCHC specification a SCHC ACK reports only one bitmap for the reception of exactly one window of tiles. The present SCHC Compound ACK specification extends the SCHC ACK message format so that it can contain several bitmaps, each bitmap being identified by its corresponding window number.

The SCHC ACK format, as presented in [RFC8724], can be considered a special SCHC Compound ACK case, in which it reports only the tiles of one window. Therefore, the SCHC Compound ACK is backwards compatible with the SCHC ACK format presented in [RFC8724]. The receiver can suspect if the sender does not support the SCHC Compound ACK, if the sender does not resend any tiles from windows that are not the first one in the SCHC Compound ACK and more ACKs are needed. In that case, the receiver can send SCHC Compound ACKs with only one window of tiles.

Also, some flexibility is introduced with respect to [RFC8724], in that the receiver has the capability to respond to the All-0 with a SCHC Compound ACK or not, depending on certain parameters, like network conditions, sender buffer/chache size, supported application delay. Note that even though the protocol allows for such flexibility, the actual decision criteria is not specified in this document. The application MUST set expiration timer values according to when the feedback is expected to be received, e.g., after the All-0 or after the All-1.

The following Section 8.4.3 (and its subsections) replaces the complete sections 8.4.3 (and its subsections) of RFC 8724.

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 Compound ACK reports on the reception of one window of tiles or several windows of tiles, each one identified by its window number.

See Figure 23 for an example.

```
         +-----------------------------------------------...-----------+
         |                       SCHC Packet                           |
         +-----------------------------------------------...-----------+

   Tile#    | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 |     | 0 | 4 |3|
   Window#  |-------- 0 --------|-------- 1 --------|- 2  ... 27 -|- 28-|


   SCHC Fragment msg   |-----------|
```

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 Compound ACKs to the fragment sender about windows for which tiles are missing. No SCHC Compound 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.

* Usage or not of the SCHC Compound ACK message.

* Usage or not of the compressed bitmap format in the last window of the SCHC Compound ACK message.

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 Compound 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 Compound 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 Compound ACK:

* if one of the W field in the SCHC Compound 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 Compound 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 of all the windows that are reported missing in the SCHC Compound ACK.

            * if the last of these SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MAY either send in addition a SCHC ACK REQ with the W field corresponding to the last window, or repeat the All-1 SCHC Fragment to ask the receiver confirmation that all tiles have been correctly received.

            * 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 Compound 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 Compound 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 Compound 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 All-0 SCHC Fragment:

* if the receiver knows of any windows with missing tiles for the packet being reassembled (and depending on certain parameters, like network conditions, sender buffer/chache size, supported application delay, among others), it MAY return a SCHC Compound ACK for the missing tiles, starting from the lowest-numbered window.

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 Compound ACK for the missing tiles, starting from the lowest-numbered window.

* otherwise:

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

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

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

Upon sending a SCHC Compound 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 Compound 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.

4. SCHC Compound ACK Example

Figure 7 shows an example transmission of a SCHC Packet in ACK-on-Error mode using the SCHC Compound ACK. In the example, the SCHC Packet is fragmented in 14 tiles, with N=3, WINDOW_SIZE=7, M=2 and two lost SCHC fragments. Only 1 compound SCHC ACK is generated.

```
         Sender               Receiver
           |-----W=0, FCN=6 ----->|
           |-----W=0, FCN=5 ----->|
           |-----W=0, FCN=4 ----->|
           |-----W=0, FCN=3 ----->|
           |-----W=0, FCN=2 --X   |
           |-----W=0, FCN=1 ----->|
           |-----W=0, FCN=0 ----->| Bitmap: 1111011
       (no ACK)
           |-----W=1, FCN=6 ----->|
           |-----W=1, FCN=5 ----->|
           |-----W=1, FCN=4 ----->|
           |-----W=1, FCN=3 ----->|
           |-----W=1, FCN=2 ----->|
           |-----W=1, FCN=1 --X   |
           |-- W=1, FCN=7 + RCS ->| Integrity check: failure
           |<--- Compound ACK ----| [C=0, W=0 - Bitmap:1111011,
           |-----W=0, FCN=2 ----->|        W=1 - Bitmap:1111101]
           |-----W=1, FCN=1 ----->| Integrity check: success
           |<--- ACK, W=1, C=1 ---| C=1
         (End)
```

Figure 7: SCHC Compound ACK message sequence example

```
 |--- SCHC ACK Header --|- W=00 --|----- W=01 -----|
        |--T-|---M--|-1-|         |---M--|         |---M--|
+------+----+------+---+---------+------+---------+------+-----+
|RuleID|DTag| W=00 |C=0| 1111011 | W=01 | 1111101 |  00  | pad |
+------+----+------+---+---------+------+---------+------+-----+
                        next L2 Word boundary ->|<-- L2 Word ->|
```

Figure 8: SCHC Compound ACK message format example: Losses are found in windows 00 and 01

5. SCHC Compound ACK YANG Data Model

The present document also extends the SCHC YANG data model defined in [RFC9363] by including a new leaf in the Ack-on-Error fragmentation mode to describe both the option to use the SCHC Compound ACK, as well as its bitmap format.

5.1. SCHC YANG Data Model Extension

```
<CODE BEGINS> file "ietf-lpwan-schc-compound-ack@2023-03-16.yang"
module ietf-lpwan-schc-compound-ack {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:"
          + "ietf-lpwan-schc-compound-ack";
  prefix schc-compound-ack;

  import ietf-schc {
    prefix schc;
  }

  organization
    "IETF IPv6 over Low Power Wide-Area Networks (lpwan)
     working group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/lpwan/about/>
     WG List:  <mailto:lp-wan@ietf.org>
     Editor:   Laurent Toutain
       <mailto:laurent.toutain@imt-atlantique.fr>
     Editor:   Juan Carlos Zuniga
       <mailto:j.c.zuniga@ieee.org>
     Editor:   Sergio Aguilar
       <mailto:sergio.aguilar.romero@upc.edu>";
  description
    "Copyright (c) 2023 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.
     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject to
     the license terms contained in, the Revised BSD License set
```

```
         forth in Section 4.c of the IETF Trust's Legal Provisions
         Relating to IETF Documents
         (https://trustee.ietf.org/license-info).
         This version of this YANG module is part of RFC 9363
         (https://www.rfc-editor.org/info/rfc9363); see the RFC itself
         for full legal notices.
         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 (RFC 2119) (RFC 8174) when, and only when,
         they appear in all capitals, as shown here.
         ***************************************************************
         Generic data model for the Static Context Header Compression
         Rule for SCHC, based on RFCs 8724 and 8824.  Including
         compression, no-compression, and fragmentation Rules.";

      revision 2023-03-16 {
        description
          "Initial version for RFC YYYY ";
        reference
          "RFC YYYY: SCHC Compound ACK";
      }

      identity bitmap-format-base-type {
        description
          "Define how the bitmap is formed in ACK messages.";
      }

      identity bitmap-RFC8724 {
        base bitmap-format-base-type;
        description
          "Bitmap by default as defined in RFC8724.";
        reference
          "RFC 8724    SCHC: Generic Framework for Static
                       Context Header Compression and Fragmentation";
      }

      identity bitmap-compound-ack {
        base bitmap-format-base-type;
        description
          "Compound ACK allows several bitmaps in a ACK message.";
      }

      typedef bitmap-format-type {
        type identityref {
          base bitmap-format-base-type;
        }
        description
```

```
          "Type of bitmap used in rules.";
      }

      augment "/schc:schc/schc:rule/schc:nature/"
            + "schc:fragmentation/schc:mode/schc:ack-on-error" {
        leaf bitmap-format {
          when "derived-from-or-self(../schc:fragmentation-mode,
                                'schc:fragmentation-mode-ack-on-error')";
          type schc-compound-ack:bitmap-format-type;
          default "schc-compound-ack:bitmap-RFC8724";
          description
            "How the bitmaps are included in the SCHC ACK message.";
        }
        leaf last-bitmap-compression {
          when "derived-from-or-self(../schc:fragmentation-mode,
                                'schc:fragmentation-mode-ack-on-error')";
          type boolean;
          default "true";
          description
            "When true the ultimate bitmap in the SCHC ACK message
             can be compressed. Default behavior from RFC8724";
          reference
            "RFC 8724   SCHC: Generic Framework for Static
                        Context Header Compression and
                        Fragmentation";
        }
        description
          "Augment the SCHC rules to manage Compound Ack.";
      }
    }
    <CODE ENDS>
```

Figure 9: SCHC YANG Data Model - Compound ACK extension

5.2. SCHC YANG Tree Extension

```
   module: ietf-lpwan-schc-compound-ack
   augment /schc:schc/schc:rule/schc:nature/schc:fragmentation/
         schc:mode/schc:ack-on-error:
   +--rw bitmap-format?             schc-compound-ack:bitmap-format-type
   +--rw last-bitmap-compression?   boolean
```

Figure 10: Tree Diagram - Compound ACK extension

6. SCHC Compound ACK Parameters

This section lists the parameters related to the SCHC Compound ACK usage that need to be defined in the Profile. This list MUST be appended to the list of SCHC parameters under "Decision to use SCHC fragmentation mechanism or not. If yes, the document must describe:" in Annex D of [RFC8724].

* Usage or not of the SCHC Compound ACK message.

* Usage or not of the compressed bitmap format in the last window of the SCHC Compound ACK message.

7. Security considerations

The current document specifies a message format extension for SCHC. Hence, the same Security Considerations defined in [RFC8724] and in [RFC9363] apply.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:
/schc:schc/schc:rule/schc:nature/schc:fragmentation/schc:mode/schc:ack-on-error: All the data nodes may be modified. The Rule contains sensitive information, such as the SCHC F/R mode configuration and usage and configuration of the SCHC Compound ACK. An attacker may try to modify other devices' Rules by changing the F/R mode or the usage of the SCHC Compound ACK and may block communication or create extra ACKs. Therefore, a device must be allowed to modify only its own rules on the remote SCHC instance. The identity of the requester must be validated. This can be done through certificates or access lists. Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:
/schc:schc/schc:rule/schc:nature/schc:fragmentation/schc:mode/schc:ack-on-error: By reading this module, an attacker may learn the F/R mode used by the device and how the device manage the bitmap creation and also learn the buffer sizes and when the device will request an ACK.

8. IANA Considerations

This document registers one URI and one YANG data model.

8.1. URI Registration

IANA registered the following URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-lpwan-schc-compound-ack

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

8.2. YANG Module Name Registration

IANA has registered the following YANG data model in the "YANG Module Names" registry [RFC6020].

name: ietf-lpwan-schc-compound-ack

namespace: urn:ietf:params:xml:ns:yang:ietf-lpwan-schc-compound-ack

prefix: schc-compound-ack

reference: RFC

9. Acknowledgements

Carles Gomez has been funded in part by the Spanish Government through the TEC2016-79988-P grant, and the PID2019-106808RA-I00 grant (funded by MCIN / AEI / 10.13039/501100011033), and by Secretaria d’Universitats i Recerca del Departament d’Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

Sergio Aguilar has been funded by the ERDF and the Spanish Government through project TEC2016-79988-P and project PID2019-106808RA-I00, AEI/FEDER, EU (funded by MCIN / AEI / 10.13039/501100011033).

Sandra Cespedes has been funded in part by the ANID Chile Project FONDECYT Regular 1201893 and Basal Project FB0008.

Diego Wistuba has been funded by the ANID Chile Project FONDECYT Regular 1201893.

The authors would like to thank Rafael Vidal, Julien Boite, Renaud Marty, Antonis Platis, Dominique Barthel and Pascal Thubert for their very useful comments, reviews and implementation design considerations.

10. References

10.1. 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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC9363] Minaburo, A. and L. Toutain, "A YANG Data Model for Static Context Header Compression (SCHC)", RFC 9363, DOI 10.17487/RFC9363, March 2023, <https://www.rfc-editor.org/info/rfc9363>.

10.2. Informative References

[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>.

Authors' Addresses

Juan Carlos Zuniga
Cisco
Montreal QC
Canada
Email: juzuniga@cisco.com

Carles Gomez
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: carles.gomez@upc.edu

Sergio Aguilar
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: sergio.aguilar.romero@upc.edu

Laurent Toutain
IMT-Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France
Email: Laurent.Toutain@imt-atlantique.fr

Sandra Cespedes
Concordia University
1455 De Maisonneuve Blvd. W.
Montreal QC, H3G 1M8
Canada
Email: sandra.cespedes@concordia.ca

Diego Wistuba
NIC Labs, Universidad de Chile
Av. Almte. Blanco Encalada 1975
Santiago
Chile
Email: wistuba@niclabs.cl

lpwan Working Group E. Ramos
Internet-Draft Ericsson
Intended status: Standards Track A. Minaburo
Expires: 18 June 2023 Acklio
15 December 2022

Static Context Header Compression over Narrowband Internet of Things
draft-ietf-lpwan-schc-over-nbiot-15

Abstract

This document describes Static Context Header Compression and Fragmentation (SCHC) specifications, RFC 8724 and RFC 8824, combination with the 3rd Generation Partnership Project (3GPP) and the Narrowband Internet of Things (NB-IoT).

This document has two parts. One normative to specify the use of SCHC over NB-IoT. And one informational, which recommends some values if 3GPP wanted to use SCHC inside their architectures.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines the scenarios where the Static Context Header Compression and fragmentation (SCHC) [RFC8724] and [RFC8824] are suitable for 3rd Generation Partnership Project (3GPP) and Narrowband Internet of Things (NB-IoT) protocol stacks.

In the 3GPP and the NB-IoT networks, header compression efficiently brings Internet connectivity to the Device-User Equipment (Dev-UE), the radio (RGW-eNB) and network (NGW-MME) gateways, and the Application Server. This document describes the SCHC parameters supporting static context header compression and fragmentation over the NB-IoT architecture.

This document assumes functionality for NB-IoT of 3GPP release 15 [_3GPPR15]. Otherwise, the text explicitly mentions other versions' functionality.

This document has two parts, a standard end-to-end scenario describing how any application must use SCHC over the 3GPP public service. And informational scenarios about how 3GPP could use SCHC in their protocol stack network.

2. Conventions and Definitions

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. Terminology

This document will follow the terms defined in [RFC8724], in [RFC8376], and the [TR23720].

* Capillary Gateway. A capillary gateway facilitates seamless integration because it has wide area connectivity through cellular and provides wide area access as a proxy to other devices using LAN technologies (BT, Wi-Fi, Zigbee, or others.)

* CIoT EPS. Cellular IoT Evolved Packet System. It is a functionality to improve the support of small data transfers.

* Dev-UE. Device - User Equipment.

* DoNAS. Data over Non-Access Stratum.

* EPC. Evolved Packet Connectivity. Core network of 3GPP LTE systems.

* EUTRAN. Evolved Universal Terrestrial Radio Access Network. Radio access network of LTE-based systems.

* HARQ. Hybrid Automatic Repeat Request.

* HSS. Home Subscriber Server. It is a database that contains users' subscription data, including data needed for mobility management.

* IP address. IPv6 or IPv4 address used.

* IWK-SCEF. InterWorking Service Capabilities Exposure Function. It is used in roaming scenarios, it is located in the Visited PLMN and serves for interconnection with the SCEF of the Home PLMN.

* L2. Layer-2 in the 3GPP architectures it includes MAC, RLC and PDCP layers see Appendix A.

* LCID. Logical Channel ID. Is the logical channel instance of the corresponding MAC SDU.

* MAC. Medium Access Control protocol, part of L2.

* NAS. Non-Access Stratum.

* NB-IoT. Narrowband IoT. A 3GPP LPWAN technology based on the LTE architecture but with additional optimization for IoT and using a Narrowband spectrum frequency.

* NGW-CSGN. Network Gateway - CIoT Serving Gateway Node.

* NGW-CSGW. Network Gateway - Cellular Serving Gateway. It routes and forwards the user data packets through the access network.

* NGW-MME. Network Gateway - Mobility Management Entity. An entity in charge of handling mobility of the Dev-UE.

* NGW-PGW. Network Gateway - Packet Data Network Gateway. An interface between the internal with the external network.

* NGW-SCEF. Network Gateway - Service Capability Exposure Function. EPC node for exposure of 3GPP network service capabilities to 3rd party applications.

* NIDD. Non-IP Data Delivery.

* PDCP. Packet Data Convergence Protocol part of L2.

* PLMN. Public Land Mobile Network. Combination of wireless communication services offered by a specific operator.

* PDU. Protocol Data Unit. A data packet including headers that are transmitted between entities through a protocol.

* RLC. Radio Link Protocol part of L2.

* RGW-eNB. Radio Gateway - evolved Node B. Base Station that controls the UE.

* SDU. Service Data Unit. A data packet (PDU) from higher layer protocols used by lower layer protocols as a payload of their own PDUs.

4. NB-IoT Architecture

The Narrowband Internet of Things (NB-IoT) architecture has a complex structure. It relies on different NGWs from different providers. It can send data via different paths, each with different characteristics in terms of bandwidth, acknowledgments, and layer-2 reliability and segmentation.

Figure 1 shows this architecture, where the Network Gateway Cellular Internet of Things Serving Gateway Node (NGW-CSGN) optimizes co-locating entities in different paths. For example, a Dev-UE using the path formed by the Network Gateway Mobility Management Entity (NGW-MME), the NGW-CSGW, and Network Gateway Packet Data Network Gateway (NGW-PGW) may get a limited bandwidth transmission from a few bytes/s to one thousand bytes/s only.

Another node introduced in the NB-IoT architecture is the Network Gateway Service Capability Exposure Function (NGW-SCEF), which securely exposes service and network capabilities to entities external to the network operator. The Open Mobile Alliance (OMA) [OMA0116] and the One Machine to Machine (OneM2M) [TR-0024] define the northbound APIs. [TS23222] defines architecture for the common API framework for 3GPP northbound APIs and [TS33122] defines security aspects for common API framework for 3GPP northbound APIs. In this case, the path is small for data transmission. The main functions of the NGW-SCEF are Connectivity path and Device Monitoring.

```
     +---+              +---------+    +------+
     |Dev| \            | +-----+ | ---| HSS  |
     |-UE|  \           | | NGW | |    +------+
     +---+  |           | |-MME |\__
             \          / +-----+ | \
     +---+    \+-----+ /|   |     | +------+
     |Dev| ----| RGW |- |   |     | | NGW- |
     |-UE|     |-eNB |  |   |     | | SCEF |---------+
     +---+    /+-----+ \|   |     | +------+         |
             /          \ +------+|                  |
            /           |\| NGW- || +-----+   +-----------+
     +---+ /            | | CSGW |--| NGW-|---|Application|
     |Dev|              | |      || | PGW |   |   Server  |
     |-UE|              | +------+| +-----+   +-----------+
     +---+              |         |
                        |NGW-CSGN |
                        +---------+
```

Figure 1: 3GPP network architecture

5. Data Transmission in the 3GPP Architecture

NB-IoT networks deal with end-to-end user data and in-band signaling between the nodes and functions to configure, control, and monitor the system functions and behaviors. The signaling uses a different path with specific protocols, handling processes, and entities but can transport end-to-end user data for IoT services. In contrast, the end-to-end application only transports end-to-end data.

The recommended 3GPP MTU size is 1358 bytes. The radio network protocols limit the packet sizes over the air, including radio protocol overhead, to 1600 bytes, see Section 5.2.3. However, the recommended 3GPP MTU is smaller to avoid fragmentation in the network backbone due to the payload encryption size (multiple of 16) and the additional core transport overhead handling.

3GPP standardizes NB-IoT and, in general, the cellular technologies interfaces and functions. Therefore, the introduction of SCHC entities to Dev-UE, RGW-eNB, and NGW-CSGN needs to be specified in the NB-IoT standard.

This document identifies the use cases of SCHC over the NB-IoT architecture.

First, the radio transmission where, see Section 5.2.1, the Dev-UE and the RGW-eNB can use the SCHC functionalities.

Second, the packets transmitted over the control path can also use SCHC when the transmission goes over the NGW-MME or NGW-SCEF. See Section 5.2.2.

These two use cases are also valid for any 3GPP architecture and not only for NB-IoT. And as the 3GPP internal network is involved, they have been put in the informational part of this section.

And third, over the SCHC over Non-IP Data Delivery (NIDD) connection or at least up to the operator network edge, see Section 5.1.1. In this case, SCHC functionalities are available in the application layer of the Dev-UE and the Application Servers or a broker function at the edge of the operator network. NGW-PGW or NGW-SCEF transmit the packets which are non-IP traffic, using IP tunneling or API calls. It is also possible to benefit legacy devices with SCHC by using the non-IP transmission features of the operator network.

A non-IP transmission refers to other layer-2 transport different from NB-IoT.

5.1. Normative Part.

This scenarios does not modify the 3GPP architecture or any of its components, it only use it as a layer-2 transmission.

5.1.1. SCHC over Non-IP Data Delivery (NIDD)

This section specifies the use of SCHC over Non-IP Data Delivery (NIDD) services of 3GPP. The NIDD services of 3GPP enable the transmission of SCHC packets compressed by the application layer. The packets can be delivered between the NGW-PGW and the Application Server or between the NGW-SCEF and the Application Server, using IP-tunnels or API calls. In both cases, as compression occurs before transmission, the network will not understand the packet, and the network does not have context information of this compression. Therefore, the network will treat the packet as Non-IP traffic and deliver it to the other side without any other protocol stack element, directly over the layer-2.

5.1.1.1. SCHC Entities Placing over NIDD

In the two scenarios using NIDD compression, SCHC entities are located almost on top of the stack. The NB-IoT connectivity services implement SCHC in the Dev-UE, an in the Application Server. The IP tunneling scenario requires that the Application Server send the compressed packet over an IP connection terminated by the 3GPP core network. If the transmission uses the NGW-SCEF services, it is possible to utilize an API call to transfer the SCHC packets between the core network and the Application Server. Also, an IP tunnel could be established by the Application Server if negotiated with the NGW-SCEF.

```
+---------+       XXXXXXXXXXXXXXXXXXXXXXXX             +--------+
| SCHC    |      XXX                    XXX            | SCHC   |
|(Non-IP) +-----XX........................XX....+--*---+(Non-IP)|
+---------+    XX                  +----+  XX   |  |   +--------+
|         |    XX                  |SCEF+-------+  |   |        |
|         |   XXX     3GPP RAN &   +----+  XXX     +---+  UDP   |
|         |   XXX    CORE NETWORK          XXX     |   |        |
|   L2    +---+XX                  +------------+  |   +--------+
|         |     XX                 |IP TUNNELING+--+   |        |
|         |      XXX               +------------+  +---+  IP    |
+---------+       XXXX                 XXXX        |   +--------+
| PHY     +------+ XXXXXXXXXXXXXXXXXXXXXXX         +---+  PHY   |
+---------+                                            +--------+
  Dev-UE                                              Application
                                                         Server
```

Figure 2: End-to End Compression. SCHC entities placed when using Non-IP Delivery (NIDD) 3GPP Services

5.1.1.2. Parameters for Static Context Header Compression and Fragmentation (SCHC)

These scenarios MAY use SCHC header compression capability to improve the transmission of IPv6 packets.

* SCHC Context initialization.

The application layer handles the static context; consequently, the context distribution MUST be according to the application's capabilities, perhaps utilizing IP data transmissions up to context initialization. Also, the static contexts delivery may use the same IP tunneling or NGW-SCEF services used later for the SCHC packets transport.

* SCHC Rules.

For devices acting as a capillary gateway, several rules match the diversity of devices and protocols used by the devices associated with the gateway. Meanwhile, simpler devices may have predetermined protocols and fixed parameters.

* Rule ID.

This scenario can dynamically set the RuleID size before the context delivery. For example, negotiate between the applications when choosing a profile according to the type of traffic and application deployed. Transmission optimization may require only one physical layer transmission. SCHC overhead SHOULD NOT exceed the available number of effective bits of the smallest physical TB available to optimize the transmission. The packets handled by 3GPP networks are byte-aligned. Thus, to use the smallest TB, the maximum SCHC header size is 12 bits. On the other hand, more complex NB-IoT devices (such as a capillary gateway) might require additional bits to handle the variety and multiple parameters of higher-layer protocols deployed. The configuration may be part of the agreed operation profile and content distribution. The RuleID field size may range from 2 bits, resulting in 4 rules to an 8-bit value that would yield up to 256 rules that can be used with the operators and seems quite a reasonable maximum limit even for a device acting as a NAT. An application may use a larger RuleID, but it should consider the byte alignment of the expected Compression Residue. In the minimum TB size case, 2 bits of RuleID leave only 6 bits available for Compression Residue.

* SCHC MAX_PACKET_SIZE.

In these scenarios, the maximum RECOMMENDED MTU size is 1358 bytes since the SCHC packets (and fragments) are traversing the whole 3GPP network infrastructure (core and radio), not only the radio as the IP transmissions case.

* Fragmentation.

Packets larger than 1358 bytes need the SCHC fragmentation function. Since the 3GPP uses reliability functions, the No-ACK fragmentation mode MAY be enough in point-to-point connections. Nevertheless, additional considerations are described below for more complex cases.

* Fragmentation modes.

A global service assigns a QoS to the packets e.g. depending on the billing. Packets with very low QoS may get lost before arriving in the 3GPP radio network transmission, for example, in between the links of a capillary gateway or due to buffer overflow handling in a backhaul connection. The use of SCHC fragmentation with the ACK-on-Error mode is RECOMMENDED to secure additional reliability on the packets transmitted with a small trade-off on further transmissions to signal the end-to-end arrival of the packets if no transport protocol takes care of retransmission.
Also, the ACK-on-Error mode could be desirable to keep track of all the SCHC packets delivered. In that case, the fragmentation function could be activated for all packets transmitted by the applications. SCHC ACK-on-Error fragmentation MAY be activated in transmitting non-IP packets on the NGW-MME. A non-IP packet will use SCHC reserved RuleID for non-compressing packets as [RFC8724] allows it.

* Fragmentation Parameters.

SCHC profile will have specific Rules for the fragmentation modes. The rule will identify, which fragmentation mode is in use, and section Section 5.2.3 defines the RuleID size.

SCHC parametrization considers that NBIoT aligns the bit and uses padding and the size of the Transfer Block. SCHC will try to reduce padding to optimize the compression of the information. The Header size needs to be multiple of 4, and the Tiles MAY keep a fixed value of 4 or 8 bits to avoid padding except for transfer block equals 16 bits where Tiles may be 2 bits. The transfer block size has a wide range of values. Two configurations are RECOMMENDED for the fragmentation parameters.

* For Transfer Blocks smaller or equal to 304 bits using an 8-bit Header_size configuration, with the size of the header fields as follows:

  - RuleID from 1 - 3 bits,

  - DTag 1 bit,

  - FCN 3 bits,

  - W 1 bits.

* For Transfer Blocks bigger than 304 bits using a 16-bit Header_size configuration, with the size of the header fields as follows:

  - RulesID from 8 - 10 bits,

  - DTag 1 or 2 bits,

  - FCN 3 bits,

  - W 2 or 3 bits.

* WINDOW_SIZE of 2^N-1 is RECOMMENDED.

* RCS will follow the default size defined in section 8.2.3 of the [RFC8724], with a length equal to the L2 Word.

* MAX_ACK_REQ is RECOMMENDED to be 2, but applications MAY change this value based on transmission conditions.

The IoT devices communicate with small data transfer and use the Power Save Mode and the Idle Mode DRX, which govern how often the device wakes up, stays up, and is reachable. The use of the different modes allows the battery to last ten years.
Table 10.5.163a in [TS24008] specifies a range for the radio timers as N to 3N in increments of one where the units of N can be 1 hour or 10 hours. The Inactivity Timer and the Retransmission Timer be set based on these limits.

5.2. Informational Part.

These scenarios shows how 3GPP could use SCHC for their transmissions.

5.2.1. Use of SCHC over the Radio link

Deploying SCHC over the radio link only would require placing it as part of the protocol stack for data transfer between the Dev-UE and the RGW-eNB. This stack is the functional layer responsible for transporting data over the wireless connection and managing radio resources. There is support for features such as reliability, segmentation, and concatenation. The transmissions use link adaptation, meaning that the system will optimize the transport format used according to the radio conditions, the number of bits to transmit, and the power and interference constraints. That means that the number of bits transmitted over the air depends on the selected Modulation and Coding Schemes (MCS). Transport Block (TB) transmissions happen in the physical layer at network-synchronized intervals called Transmission Time Interval (TTI). Each Transport Block has a different MCS and number of bits available to transmit. The MAC layer [TR36321] defines the Transport Blocks' characteristics. The Radio link stack shown in Figure 3 comprises the Packet Data Convergence Protocol (PDCP) [TS36323], Radio Link Protocol (RLC) [TS36322], Medium Access Control protocol (MAC) [TR36321], and the Physical Layer [TS36201]. The Appendix A gives more details about these protocols.

```
  +---------+                             +---------+  |
  |IP/non-IP+-----------------------------+IP/non-IP+->+
  +---------+   |   +---------------+   |  +---------+  |
  | PDCP    +-------+ PDCP   | GTP|U +------+ GTP-U    |->+
  | (SCHC)  +       + (SCHC)|        +      +          |  |
  +---------+   |   +---------------+   |  +---------+  |
  | RLC     +-------+ RLC    |UDP/IP +------+ UDP/IP  +->+
  +---------+   |   +---------------+   |  +---------+  |
  | MAC     +-------+ MAC    | L2     +------+ L2      +->+
  +---------+   |   +---------------+   |  +---------+  |
  | PHY     +-------+ PHY    | PHY    +------+ PHY     +->+
  +---------+       +---------------+      +---------+  |
            C-Uu/                     S1-U              SGi
    Dev-UE              RGW-eNB            NGW-CSGN
           Radio Link
```

Figure 3: SCHC over the Radio link

5.2.1.1. SCHC Entities Placing over the Radio Link

The 3GPP architecture supports Robust Header Compression (ROHC) [RFC5795] in the PDCP layer. Therefore, the architecture can deploy SCHC header compression entities similarly without the need for significant changes in the 3GPP specifications.

The RLC layer has three functional modes Transparent Mode (TM), Unacknowledged Mode (UM), and Acknowledged Mode (AM). The mode of operation controls the functionalities of the RLC layer. TM only applies to signaling packets, while AM or UM carry signaling and data packets.

The RLC layer takes care of fragmentation unless for the Transparent Mode. In AM or UM modes, the SCHC fragmentation is unnecessary and SHOULD NOT be used. While sending IP packets, the Radio link does not commonly use the RLC Transparent Mode. However, if other protocol overhead optimizations are targeted for NB-IoT traffic, SCHC fragmentation may be used for TM transmission mode in the future.

5.2.2. Use of SCHC over the Non-Access Stratum (NAS)

This section consists of IETF suggestions to the 3GPP. The NGW-MME conveys mainly signaling between the Dev-UE and the cellular network [TR24301]. The network transports this traffic on top of the radio link.

This kind of flow supports data transmissions to reduce the overhead when transmitting infrequent small quantities of data. This transmission is known as Data over Non-Access Stratum (DoNAS) or Control Plane Cellular Internet of Things (CIoT) evolved packet system (EPS) optimizations. In DoNAS, the Dev-UE uses the pre-established security and can piggyback small uplink data into the initial uplink message and uses an additional message to receive a downlink small data response.

The NGW-MME performs the data encryption from the network side in a DoNAS PDU. Depending on the data type signaled indication (IP or non-IP data), the network allocates an IP address or establishes a direct forwarding path. DoNAS is regulated under rate control upon previous agreement, meaning that a maximum number of bits per unit of time is agreed upon per device subscription beforehand and configured in the device.

The system will use DoNAS when a terminal in a power-saving state requires a short transmission and receives an acknowledgment or short feedback from the network. Depending on the size of buffered data to transmit, the Dev-UE might deploy the connected mode transmissions instead, limiting and controlling the DoNAS transmissions to predefined thresholds and a good resource optimization balance for the terminal and the network. The support for mobility of DoNAS is present but produces additional overhead. The Appendix B gives additional details of DoNAS.

5.2.2.1. SCHC Entities Placing over DoNAS

SCHC resides in this scenario’s Non-Access Stratum (NAS) protocol layer. The same principles as for the section Section 5.2.1 apply here as well. Because the NAS protocol already uses ROHC [RFC5795], it can also adapt SCHC for header compression. The main difference compared to the radio link, section Section 5.2.1, is the physical placing of the SCHC entities. On the network side, the NGW-MME resides in the core network and is the terminating node for NAS instead of the RGW-eNB.

```
+--------+                       +--------+--------+  +  +--------+
| IP/    +--+-----------------+--+  IP/   |   IP/  +-----+   IP/  |
| Non-IP |  |                 |  | Non-IP | Non-IP |  |  | Non-IP |
+--------+  |                 |  +-----------------+  |  +--------+
| NAS    +-----------------------+   NAS  |GTP-C/U +-----+GTP-C/U |
|(SCHC)  |  |                 |  | (SCHC) |        |  |  |        |
+--------+  |  +-----------+  |  +-----------------+  |  +--------+
| RRC    +-----+RRC  |S1|AP+-----+ S1|AP  |        |  |  |        |
+--------+  |  +-----------+  |  +--------+  UDP   +-----+  UDP   |
| PDCP*  +-----+PDCP*|SCTP +-----+ SCTP   |        |  |  |        |
+--------+  |  +-----------+  |  +-----------------+  |  +--------+
| RLC    +-----+ RLC | IP  +-----+ IP     | IP     +-----+ IP     |
+--------+  |  +-----------+  |  +-----------------+  |  +--------+
| MAC    +-----+ MAC | L2  +-----+ L2     | L2     +-----+ L2     |
+--------+  |  +-----------+  |  +-----------------+  |  +--------+
| PHY    +--+--+ PHY | PHY +--+--+ PHY    | PHY    +-----+ PHY    |
+--------+     +-----+-----+     +--------+--------+  |  +--------+
           C-Uu/             S1                    SGi
 Dev-UE           RGW-eNB              NGW-MME              NGW-PGW
```

*PDCP is bypassed until AS security is activated TGPP36300.

Figure 4: SCHC entities placement in the 3GPP CIOT radio protocol architecture for DoNAS transmissions

5.2.3. Parameters for Static Context Header Compression and Fragmentation (SCHC) for the Radio link and DONAS use-cases.

If 3GPP incorporates SCHC, it is recommended that these scenarios use SCHC header compression [RFC8724] capability to optimize the data transmission.

* SCHC Context initialization.

The RRC (Radio Resource Control) protocol is the main tool used to configure the parameters of the Radio link. It will configure SCHC and the static context distribution as it has made for ROHC [RFC5795] operation [TS36323].

* SCHC Rules.

The network operator in these scenarios defines the number of rules. For this, the network operator must know the IP traffic the device will carry. The operator might supply rules compatible with the device’s use case. For devices acting as a capillary gateway, several rules match the diversity of devices and protocols used by the devices associated with the gateway. Meanwhile, simpler devices may have predetermined protocols and fixed parameters. The use of IPv6 and IPv4 may force to get more rules to deal with each case.

* RuleID.

There is a reasonable assumption of 9 bytes of radio protocol overhead for these transmission scenarios in NB-IoT, where PDCP uses 5 bytes due to header and integrity protection, and RLC and MAC use 4 bytes. The minimum physical Transport Blocks (TB) that can withhold this overhead value according to 3GPP Release 15 specifications are 88, 104, 120, and 144 bits. As for Section 5.1.1.2, these scenarios must optimize the physical layer where the smallest TB is 12 bits. These 12 bits must include the Compression Residue in addition to the RuleID. On the other hand, more complex NB-IoT devices (such as a capillary gateway) might require additional bits to handle the variety and multiple parameters of higher-layer protocols deployed. In that sense, the operator may want flexibility on the number and type of rules independently supported by each device; consequently, these scenarios require a configurable value. The configuration may be part of the agreed operation profile with the content distribution. The RuleID field size may range from 2 bits, resulting in 4 rules to an 8-bit value that would yield up to 256 rules that can be used with the operators and seems quite a reasonable maximum limit even for a device acting as a NAT. An application may use a larger RuleID, but it should consider the byte alignment of the expected Compression Residue. In the minimum TB size case, 2 bits of RuleID leave only 6 bits available for Compression Residue.

* SCHC MAX_PACKET_SIZE.

The Radio Link can handle the fragmentation of SCHC packets if needed, including reliability. Hence, the packet size is limited by the MTU handled by the radio protocols, which corresponds to 1600 bytes for 3GPP Release 15.

* Fragmentation.

For the Radio link Section 5.2.1 and DoNAS' Section 5.2.2 scenarios, the SCHC fragmentation functions are disabled. The RLC layer of NB-IoT can segment packets into suitable units that fit the selected transport blocks for transmissions of the physical layer. The block selection is made according to the link adaptation input function in the MAC layer and the quantity of data in the buffer. The link adaptation layer may produce different results at each Time Transmission Interval (TTI), resulting in varying physical transport blocks that depend on the network load, interference, number of bits transmitted, and QoS. Even if setting a value that allows the construction of data units following the SCHC tiles principle, the protocol overhead may be greater or equal to allowing the Radio link protocols to take care of the fragmentation intrinsically.

* Fragmentation in RLC Transparent Mode.

The RLC Transparent Mode mostly applies to control signaling transmissions. When RLC operates in Transparent Mode, the MAC layer mechanisms ensure reliability and generate overhead. This additional reliability implies sending repetitions or automatic retransmissions.

The ACK-Always fragmentation mode of SCHC may reduce this overhead in future operations when data transmissions may use this mode. ACK-Always mode may transmit compressed data with fewer possible transmissions by using fixed or limited transport blocks compatible with the tiling SCHC fragmentation handling. For SCHC fragmentation parameters see Section 5.1.1.2.

6. Padding

NB-IoT and 3GPP wireless access, in general, assumes byte-aligned payload. Therefore, the layer 2 word for NB-IoT MUST be considered 8 bits, and the padding treatment should use this value accordingly.

7. IANA considerations

This document has no IANA actions.

8. Security considerations

This document does not add any security considerations and follows the [RFC8724] and the 3GPP access security document specified in [TS33122].

9. References

9.1. 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>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

9.2. Informative References

[OMA0116] OMA, "Common definitions for RESTful Network APIs", 2018, <https://www.openmobilealliance.org/release/REST_NetAPI_Common/V1_0-20180116-A/OMA-TS-REST_NetAPI_Common-V1_0-20180116-A.pdf>.

[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>.

[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>.

[TR-0024] OneM2M, "3GPP_Interworking", 2020, <https://ftp.onem2m.org/work%20programme/WI-0037/TR-0024-3GPP_Interworking-V4_3_0.DOCX>.

[TR23720] 3GPP, "Study on architecture enhancements for Cellular Internet of Things", 2015, <https://www.3gpp.org/ftp/Specs/archive/23_series/23.720/23720-d00.zip>.

[TR24301] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification", 2019, <https://www.3gpp.org/ftp//Specs/archive/24_series/24.301/24301-f80.zip>.

[TR36321] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification", 2016, <https://www.3gpp.org/ftp/Specs/archive/36_series/36.321/36321-d20.zip>.

[TS23222] 3GPP, "Common API Framework for 3GPP Northbound APIs", 2022, <https://www.3gpp.org/ftp/Specs/archive/23_series/23.222/23222-f60.zip>.

[TS24008] 3GPP, "Mobile radio interface layer 3 specification.", 2018, <https://www.3gpp.org/ftp//Specs/archive/24_series/24.008/24008-f50.zip>.

[TS33122] 3GPP, "Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs", 2018, <https://www.3gpp.org/ftp//Specs/archive/33_series/33.122/33122-f30.zip>.

[TS36201] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); LTE physical layer; General description", 2018, <https://www.3gpp.org/ftp/Specs/archive/36_series/36.201/36201-f10.zip>.

[TS36322] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification", 2018, <https://www.3gpp.org/ftp/Specs/archive/36_series/36.322/36322-f01.zip>.

[TS36323] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification", 2016, <https://www.3gpp.org/ftp/Specs/archive/36_series/36.323/36323-d20.zip>.

[TS36331] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification", 2018, <https://www.3gpp.org/ftp//Specs/archive/36_series/36.331/36331-f51.zip>.

[_3GPPR15] 3GPP, "The Mobile Broadband Standard", 2019, <https://www.3gpp.org/release-15>.

Appendix A. NB-IoT User Plane protocol architecture

A.1. Packet Data Convergence Protocol (PDCP) [TS36323]

Each of the Radio Bearers (RB) is associated with one PDCP entity. Moreover, a PDCP entity is associated with one or two RLC entities depending on the unidirectional or bi-directional characteristics of the RB and RLC mode used. A PDCP entity is associated with either a control plane or a user plane with independent configuration and functions. The maximum supported size for NB-IoT of a PDCP SDU is 1600 octets. The primary services and functions of the PDCP sublayer for NB-IoT for the user plane include:

* Header compression and decompression using ROHC [RFC5795]

* Transfer of user and control data to higher and lower layers

* Duplicate detection of lower layer SDUs when re-establishing connection (when RLC with Acknowledge Mode in use for User Plane only)

* Ciphering and deciphering

* Timer-based SDU discard in uplink

A.2. Radio Link Protocol (RLC) [TS36322]

RLC is a layer-2 protocol that operates between the UE and the base station (eNB). It supports the packet delivery from higher layers to MAC, creating packets transmitted over the air, optimizing the Transport Block utilization. RLC flow of data packets is unidirectional, and it is composed of a transmitter located in the transmission device and a receiver located in the destination device. Therefore, to configure bi-directional flows, two sets of entities, one in each direction (downlink and uplink), must be configured and effectively peered to each other. The peering allows the transmission of control packets (ex., status reports) between entities. RLC can be configured for data transfer in one of the following modes:

* Transparent Mode (TM). RLC does not segment or concatenate SDUs from higher layers in this mode and does not include any header to the payload. RLC receives SDUs from upper layers when acting as a transmitter and transmits directly to its flow RLC receiver via lower layers. Similarly, a TM RLC receiver would only deliver without processing the packets to higher layers upon reception.

* Unacknowledged Mode (UM). This mode provides support for segmentation and concatenation of payload. The RLC packet's size depends on the indication given at a particular transmission

opportunity by the lower layer (MAC) and is octet-aligned. The packet delivery to the receiver does not include reliability support, and the loss of a segment from a packet means a complete packet loss. Also, in the case of lower layer retransmissions, there is no support for re-segmentation in case of change of the radio conditions triggering the selection of a smaller transport block. Additionally, it provides PDU duplication detection and discards, reordering of out-of-sequence, and loss detection.

* Acknowledged Mode (AM). In addition to the same functions supported by UM, this mode also adds a moving windows-based reliability service on top of the lower layer services. It also supports re-segmentation, and it requires bidirectional communication to exchange acknowledgment reports called RLC Status Report and trigger retransmissions. This model also supports protocol error detection. The mode used depends on the operator configuration for the type of data to be transmitted. For example, data transmissions supporting mobility or requiring high reliability would be most likely configured using AM. Meanwhile, streaming and real-time data would be mapped to a UM configuration.

A.3. Medium Access Control (MAC) [TR36321]

MAC provides a mapping between the higher layers abstraction called Logical Channels comprised by the previously described protocols to the Physical layer channels (transport channels). Additionally, MAC may multiplex packets from different Logical Channels and prioritize what to fit into one Transport Block if there is data and space available to maximize data transmission efficiency. MAC also provides error correction and reliability support through Hybrid Automatic Repeat reQuest (HARQ), transport format selection, and scheduling information reporting from the terminal to the network. MAC also adds the necessary padding and piggyback control elements when possible and the higher layers data.

```
                                               <Max. 1600 bytes>
                       +---+         +---+          +------+
   Application         |AP1|         |AP1|          |  AP2 |
   (IP/non-IP)         |PDU|         |PDU|          |  PDU |
                       +---+         +---+          +------+
                       |   |         |  |           |      |
   PDCP           +--------+    +--------      +-----------+
                  |PDCP|AP1|    |PDCP|AP1|     |PDCP|  AP2 |
                  |Head|PDU|    |Head|PDU|     |Head|  PDU |
                  +--------+    +--------+     +--------+--\
                  |    |   |    |     |  |     |    |   |\   `-------\
            +---------------------------+      |    |(1)|  `------\(2)\
   RLC      |RLC |PDCP|AP1|RLC |PDCP|AP1| +-------------+    +----|---+
            |Head|Head|PDU|Head|Head|PDU| |RLC |PDCP|AP2|    |RLC |AP2|
            +-------------|-------------+ |Head|Head|PDU|    |Head|PDU|
            |         |   |         |   | +---------|---+    +--------+
            |         |   | LCID1   |   | /         /   /   /         /
           /         /   /        _/  _//        _/  _/    / LCID2   /
           |        |   |        |   | /       _/  _/     /      ___/
           |        |   |        |   ||       |   |      /      /
       +------------------------------------------+ +-----------+---+
   MAC |MAC|RLC|PDCP|AP1|RLC|PDCP|AP1|RLC|PDCP|AP2| |MAC|RLC|AP2|Pad|
       |Hea|Hea|Hea |PDU|Hea|Hea |PDU|Hea|Hea |PDU| |Hea|Hea|PDU|din|
       |der|der|der |   |der|der |   |der|der |   | |der|der|   |g  |
       +------------------------------------------+ +-----------+---+
                         TB1                               TB2
```

(1) Segment One
(2) Segment Two

Figure 5: Example of User Plane packet encapsulation for two transport blocks

Appendix B. NB-IoT Data over NAS (DoNAS)

The Access Stratum (AS) protocol stack used by DoNAS is specific because the radio network still needs to establish the security associations and reduce the protocol overhead, so the PDCP (Packet Data Convergence Protocol) is bypassed until AS security is activated. RLC (Radio Link Control protocol) uses, by default, the AM mode, but depending on the network's features and the terminal, it may change to other modes by the network operator. For example, the transparent mode does not add any header or process the payload to reduce the overhead, but the MTU would be limited by the transport block used to transmit the data, which is a couple of thousand bits maximum. If UM (only Release 15 compatible terminals) is used, the RLC mechanisms of reliability are disabled, and only the reliability provided by the MAC layer by HARQ is available. In this case, the protocol overhead might be smaller than the AM case because of the lack of status reporting but with the same support for segmentation up to 1600 bytes. NAS packets are encapsulated within an RRC (Radio Resource Control) [TS36331] message.

Depending on the data type indication signaled (IP or non-IP data), the network allocates an IP address or establishes a direct forwarding path. DoNAS is regulated under rate control upon previous agreement, meaning that a maximum number of bits per unit of time is agreed upon per device subscription beforehand and configured in the device. The use of DoNAS is typically expected when a terminal in a power-saving state requires a short transmission and receiving an acknowledgment or short feedback from the network. Depending on the size of buffered data to transmit, the UE might be instructed to deploy the connected mode transmissions instead, limiting and controlling the DoNAS transmissions to predefined thresholds and a good resource optimization balance for the terminal the network. The support for mobility of DoNAS is present but produces additional overhead.

```
       +--------+   +--------+   +--------+
       |        |   |        |   |        |       +-----------------+
       |   UE   |   |  C-BS  |   |  C-SGN |       |Roaming Scenarios|
       +----|---+   +--------+   +--------+       |  +--------+     |
            |            |            |           |  |        |     |
        +----------------|------------|+          |  |  P-GW  |     |
        |        Attach                |          |  +--------+     |
        +------------------------------+          |       |         |
            |            |            |           |       |         |
     +------|------------|--------+   |           |       |         |
     |RRC Connection Establishment|   |           |       |         |
     |with NAS PDU transmission   |   |           |       |         |
     |& Ack Rsp                   |   |           |       |         |
     +----------------------------+   |           |       |         |
            |            |            |           |       |         |
            |            |Initial UE  |           |       |         |
            |            |message     |           |       |         |
            |            |----------->|           |       |         |
            |            |            |           |       |         |
            |            | +---------------------+|       |         |
            |            | |Checks Integrity     ||       |         |
            |            | |protection, decrypts ||       |         |
            |            | |data                 ||       |         |
            |            | +---------------------+|       |         |
            |            |            |       Small data packet     |
            |            |            |------------------------------->
            |            |            |       Small data packet     |
            |            |            |<-------------------------------
            |            | +----------|---------+ |       |         |
            |            | Integrity protection,| |       |         |
            |            | encrypts data        | |       |         |
            |            | +--------------------+ |       |         |
            |            |            |           |       |         |
            |            |Downlink NAS|           |       |         |
            |            |message     |           |       |         |
            |            |<-----------|           |       |         |
    +-----------------------+         |           |       |         |
    |Small Data Delivery,   |         |           |       |         |
    |RRC connection release |         |           |       |         |
    +-----------------------+         |           |       |         |
                                                  |                 |
                                                  |                 |
                                                  +-----------------+
```

Figure 6: DoNAS transmission sequence from an Uplink initiated access

```
                      +---+ +---+ +---+                  +----+
    Application       |AP1| |AP1| |AP2|                  |AP2 |
   (IP/non-IP)        |PDU| |PDU| |PDU|  ............... |PDU |
                      +---+ +---+ +---+                  +----+
                      |   | |   | |   |                  |    |
                      |   | |   | |   |                  |    |
                      |   | |   | |   |                  |    |
                      |   | |   | |   |                  |    |
                      |   |/   /  |    \                 |    |
   NAS /RRC      +--------+---|---+----+            +---------+
                 |NAS/|AP1|AP1|AP2|NAS/|            |NAS/|AP2 |
                 |RRC |PDU|PDU|PDU|RRC |            |RRC |PDU |
                 +--------+-|-+---+----+            +---------|
                 |          |         |            |         |
                 |          |\         |            |         |
                 |<--Max. 1600 bytes-->|__          |_        |
                 |          |  \__        \____       \_       \
                 |          |     \           \         \__     \
                 |          |      \          |           |      \_
            +---------------|+-----|----------+            \       \
   RLC      |RLC | NAS/RRC  ||RLC  | NAS/RRC  |       +----|-------+
            |Head|  PDU(1/2)||Head | PDU (2/2)|       |RLC |NAS/RRC|
            +---------------++---------------+       |Head|PDU    |
            |    |          | \               |       +------------+
            |    |    LCID1 |  \              |       |           /
            |    |          |   \              \      |           |
            |    |          |    \              \     |           |
            |    |          |     \              \     \          |
       +----+----+----------++-----|----+---------++----+---------|---+
   MAC |MAC |RLC |    RLC   ||MAC  |RLC |  RLC    ||MAC |  RLC    |Pad|
       |Head|Head|  PAYLOAD ||Head |Head| PAYLOAD ||Head|  PDU    |   |
       +----+----+----------++-----+----+---------++----+---------+---+
                TB1                   TB2                     TB3
```

Figure 7: Example of User Plane packet encapsulation for Data over NAS

Appendix C. Acknowledgements

The authors would like to thank (in alphabetic order): Carles Gomez, Antti Ratilainen, Tuomas Tirronen, Pascal Thubert, Eric Vyncke.

Authors' Addresses

Edgar Ramos
Ericsson
Hirsalantie 11
FI- 02420 Jorvas, Kirkkonummi
Finland
Email: edgar.ramos@ericsson.com

Ana Minaburo
Acklio
1137A Avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: ana@ackl.io

lpwan Working Group JC. Zuniga
Internet-Draft
Intended status: Standards Track C. Gomez
Expires: 7 August 2023 S. Aguilar
Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
S. Cespedes
Concordia University
D. Wistuba
NIC Labs, Universidad de Chile
J. Boite
Unabiz (Sigfox)
3 February 2023

SCHC over Sigfox LPWAN
draft-ietf-lpwan-schc-over-sigfox-23

Abstract

The Static Context Header Compression and fragmentation (SCHC) specification (RFC8724) describes a generic framework for application header compression and fragmentation modes designed for Low Power Wide Area Network (LPWAN) technologies. The present document defines a profile of SCHC over Sigfox LPWAN, and provides optimal parameter values and modes of operation.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Generic Framework for Static Context Header Compression and Fragmentation (SCHC) specification [RFC8724] can be used in conjunction with any of the four LPWAN technologies described in [RFC8376]. These LPWANs have similar characteristics such as star-oriented topologies, network architecture, connected devices with built-in applications, etc.

SCHC offers a considerable degree of flexibility to accommodate all these LPWAN technologies. Even though there are a great number of similarities between them, some differences exist with respect to the transmission characteristics, payload sizes, etc. Hence, there are optimal parameters and modes of operation that can be used when SCHC is used in conjunction with a specific LPWAN technology.

Sigfox is an LPWAN technology that offers energy-efficient connectivity for devices at a very low cost. Sigfox complete documentation can be found in [sigfox-docs]. Sigfox aims to provide a very wide area network composed of Base Stations that receive short uplink messages (up to 12 bytes in size) sent by devices over the long-range Sigfox radio protocol, as described in [RFC8376]. Base Stations then forward messages to the Sigfox Cloud infrastructure for further processing (e.g., to offer geolocation services) and final delivery to the customer. Base Stations also relay downlink messages (with a fixed 8 bytes size) sent by the Sigfox Cloud to the devices, downlink messages being generated when devices explicitly request for it with a flag in an uplink message. With SCHC functionalities, the Sigfox network offers more reliable communications (including recovery of lost messages) and is able to convey extended-size payloads (allowing for fragmentation/reassembly of messages) [sigfox-spec].

This document describes the parameters, settings, and modes of operation to be used when SCHC is implemented over a Sigfox LPWAN. The set of parameters forms a "SCHC over Sigfox profile". The SCHC over Sigfox Profile is applicable to the Sigfox Radio specification versions up to v1.6/March 2022 [sigfox-spec] (support for future versions would have to be assessed).

2. Terminology

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.

It is assumed that the reader is familiar with the terms and mechanisms defined in [RFC8376] and in [RFC8724]. Also, it is assumed that the reader is familiar with Sigfox terminology [sigfox-spec].

3. SCHC over Sigfox

The Generic SCHC Framework described in [RFC8724] takes advantage of previous knowledge of traffic flows existing in LPWAN applications to avoid context synchronization.

Contexts need to be stored and pre-configured on both ends. This can be done either by using a provisioning protocol, by out-of-band means, or by pre-provisioning them (e.g., at manufacturing time). For example, the context exchange can be done by using NETCONF[RFC6241] with SSH, RESTCONF[RFC8040] with HTTPs, and CORECONF[I-D.ietf-core-comi] with CoAP[RFC7252] as provisioning protocols. The contexts can be encoded in XML under NETCONF, in JSON[RFC8259] under RESTCONF and in CBOR[RFC8949] under CORECONF. The way contexts are configured and stored on both ends is out of the scope of this document.

3.1. Network Architecture

Figure 1 represents the architecture for Compression/Decompression (C/D) and Fragmentation/Reassembly (F/R) based on the terminology defined in [RFC8376], where the Radio Gateway (RGW) is a Sigfox Base Station and the Network Gateway (NGW) is the Sigfox cloud-based Network.

```
   Sigfox Device                                           Application
 +----------------+                                     +--------------+
 | APP1 APP2 APP3 |                                     |APP1 APP2 APP3|
 +----------------+                                     +--------------+
 |   UDP  |       |                                     |     |  UDP   |
 |  IPv6  |       |                                     |     | IPv6   |
 +--------+       |                                     |     +--------+
 | SCHC C/D & F/R |                                     |              |
 |                |                                     |              |
 +-------+--------+                                     +--------+-----+
         $                                                       .
         $   +---------+     +--------------+     +---------+    .
         $   |         |     |   Network    |     | Network |    .
         +~~ |Sigfox BS|     |   Gateway    |     |  SCHC   |    .
             |  (RGW)  | === |    (NGW)     | ... |C/D & F/R|.....
             |         |     | Sigfox Cloud |     |         |   IP-based
             +---------+     +--------------+     +---------+   Network
 ------- Uplink message ------>
                                        <------- Downlink message ------
 Legend:
 $, ~ : Radio link
 = : Internal Sigfox Network
 . : External IP-based Network
```

Figure 1: Network Architecture

In the case of the global Sigfox Network, RGWs (or Base Stations) are distributed over multiple countries wherever the Sigfox LPWAN service is provided. The NGW (or cloud-based Sigfox Core Network) is a single entity that connects to all RGWs (Sigfox Base Stations) in the world, providing hence a global single star network topology.

The Sigfox Device sends application packets that are compressed and/or fragmented by a SCHC C/D + F/R to reduce headers size and/or fragment the packet. The resulting SCHC Message is sent over a layer two (L2) Sigfox frame to the Sigfox Base Stations, which then forwards the SCHC Message to the Network Gateway (NGW). The NGW then delivers the SCHC Message and associated gathered metadata to the Network SCHC C/D + F/R.

The Sigfox Network (NGW) communicates with the Network SCHC C/D + F/R for compression/decompression and/or for fragmentation/reassembly. The Network SCHC C/D + F/R shares the same set of rules as the Device SCHC C/D + F/R. The Network SCHC C/D + F/R can be collocated with the NGW or it could be located in a different place, as long as a tunnel or secured communication is established between the NGW and the SCHC C/D + F/R functions. After decompression and/or reassembly, the packet can be forwarded over the Internet to one (or several) LPWAN Application Server(s) (App).

The SCHC C/D + F/R processes are bidirectional, so the same principles are applicable on both Uplink (UL) and Downlink (DL).

3.2. Uplink

Uplink Sigfox transmissions occur in repetitions over different times and frequencies. Besides time and frequency diversities, the Sigfox network also provides spatial diversity, as potentially an Uplink message will be received by several base stations. The uplink message application payload size can be up to 12 bytes.

Since all messages are self-contained and base stations forward all these messages back to the same Sigfox Network, multiple input copies can be combined at the NGW providing for extra reliability based on the triple diversity (i.e., time, space and frequency).

A detailed description of the Sigfox Radio Protocol can be found in [sigfox-spec].

Messages sent from the Device to the Network are delivered by the Sigfox network (NGW) to the Network SCHC C/D + F/R through a callback/API with the following information:

* Device ID

* Message Sequence Number

* Message Payload

* Message Timestamp

* Device Geolocation (optional)

* Received Signal Strength Indicator (RSSI) (optional)

* Device Temperature (optional)

* Device Battery Voltage (optional)

The Device ID is a globally unique identifier assigned to the Device, which is included in the Sigfox header of every message. The Message Sequence Number is a monotonically increasing number identifying the specific transmission of this Uplink message, and it is also part of the Sigfox header. The Message Payload corresponds to the payload that the Device has sent in the Uplink transmission. Battery Voltage, temperature and RSSI values are sent in the confirmation control message, which is mandatorially sent by the device after the successful reception of a downlink message (see [sigfox-callbacks] Section 5.2).

The Message Timestamp, Device Geolocation, RSSI, Device Temperature and Device Battery Voltage are metadata parameters provided by the Network.

A detailed description of the Sigfox callbacks/APIs can be found in [sigfox-callbacks].

Only messages that have passed the L2 Cyclic Redundancy Check (CRC) at network reception are delivered by the Sigfox Network to the Network SCHC C/D + F/R.

The L2 Word Size used by Sigfox is 1 byte (8 bits).

Figure 2 shows a SCHC Message sent over Sigfox, where the SCHC Message could be a full SCHC Packet (e.g., compressed) or a SCHC Fragment (e.g., a piece of a bigger SCHC Packet).

```
| Sigfox Header | Sigfox payload  |
+---------------+---------------- +
                |   SCHC message  |
```

Figure 2: SCHC Message in Sigfox

3.3. Downlink

Downlink transmissions are Device-driven and can only take place following an Uplink communication that so indicates. Hence, a Sigfox Device explicitly indicates its intention to receive a Downlink message (with a size of 8 bytes) using a Downlink request flag when sending the preceding Uplink message to the network. The Downlink request flag is part of the Sigfox protocol headers. After completing the Uplink transmission, the Device opens a fixed window for Downlink reception. The delay and duration of the reception opportunity window have fixed values. If there is a Downlink message to be sent for this given Device (e.g., either a response to the Uplink message or queued information waiting to be transmitted), the network transmits this message to the Device during the reception window. If no message is received by the Device after the reception opportunity window has elapsed, the Device closes the reception window opportunity and gets back to the normal mode (e.g., continue Uplink transmissions, sleep, stand-by, etc.)

When a Downlink message is sent to a Device, a reception acknowledgement is generated by the Device and sent back to the Network through the Sigfox radio protocol and reported in the Sigfox Network backend.

A detailed description of the Sigfox Radio Protocol can be found in [sigfox-spec] and a detailed description of the Sigfox callbacks/APIs can be found in [sigfox-callbacks]. A Downlink request flag can be included in the information exchange between the Sigfox Network and Network SCHC.

3.3.1. SCHC ACK on Downlink

As explained previously, Downlink transmissions are Device-driven and can only take place following a specific Uplink transmission that indicates and allows a following Downlink opportunity. For this reason, when SCHC bidirectional services are used (e.g., Ack-on-Error fragmentation mode) the SCHC protocol implementation needs to consider the times when a Downlink message (e.g., SCHC ACK) can be sent and/or received.

For the Uplink ACK-on-Error fragmentation mode, a Downlink opportunity MUST be indicated by the last fragment of every window, which is signalled by a specific value of the Fragment Compressed Number (FCN) value, i.e., FCN = All-0, or FCN = All-1. The FCN is the tile index in a specific window. The combination of the FCN and the window number uniquely identifies a SCHC Fragment as explained in [RFC8724]. The Device sends the fragments in sequence and, after transmitting the FCN = All-0 or FCN = All-1, it opens up a reception opportunity. The Network SCHC can then decide to respond at that opportunity (or wait for a further one) with a SCHC ACK indicating that there are missing fragments from the current or previous windows. If there is no SCHC ACK to be sent, or if the network decides to wait for a further Downlink transmission opportunity, then no Downlink transmission takes place at that opportunity and after a timeout the Uplink transmissions continue. Intermediate SCHC fragments with FCN different from All-0 or All-1 MUST NOT use the Downlink request flag to request a SCHC ACK.

3.4. SCHC Rules

The RuleID MUST be included in the SCHC header. The total number of rules to be used affects directly the RuleID field size, and therefore the total size of the fragmentation header. For this reason, it is RECOMMENDED to keep the number of rules that are defined for a specific device to the minimum possible. Large RuleID sizes (and thus larger fragmentation header) is acceptable for devices without significant energy constraints (e.g., a sensor that is powered by the electricity grid).

RuleIDs can be used to differentiate data traffic classes (e.g., QoS, control vs. data, etc.), and data sessions. They can also be used to interleave simultaneous fragmentation sessions between a Device and the Network.

3.5. Fragmentation

The SCHC specification [RFC8724] defines a generic fragmentation functionality that allows sending data packets or files larger than the maximum size of a Sigfox payload. The functionality also defines a mechanism to send reliably multiple messages, by allowing to resend selectively any lost fragments.

The SCHC fragmentation supports several modes of operation. These modes have different advantages and disadvantages depending on the specifics of the underlying LPWAN technology and application Use Case. This section describes how the SCHC fragmentation functionality should optimally be implemented when used over a Sigfox LPWAN for the most typical Use Case applications.

As described in section 8.2.3 of [RFC8724], the integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receiver end. Since only Uplink/Downlink messages/fragments that have passed the Sigfox CRC-check are delivered to the Network/Sigfox Device SCHC C/D + F/R, integrity can be guaranteed when no consecutive messages are missing from the sequence and all FCN bitmaps are complete. With this functionality in mind, and in order to save protocol and processing overhead, the use of a Reassembly Check Sequence (RCS) as described in Section 3.5.1.5 MUST be used.

3.5.1. Uplink Fragmentation

Sigfox Uplink transmissions are completely asynchronous and take place in any random frequency of the allowed Uplink bandwidth allocation. In addition, devices may go to deep sleep mode, and then wake up and transmit whenever there is a need to send information to the network, as there is no need to perform any network attachment, synchronization, or other procedure before transmitting a data packet.

Since Uplink transmissions are asynchronous, a SCHC fragment can be transmitted at any given time by the Device. Sigfox Uplink messages are fixed in size, and as described in [RFC8376] they can carry 0-12 bytes payload. Hence, a single SCHC Tile size per fragmentation mode can be defined so that every Sigfox message always carries one SCHC Tile.

When the ACK-on-Error mode is used for Uplink fragmentation, the SCHC Compound ACK defined in [I-D.ietf-lpwan-schc-compound-ack]) MUST be used in the Downlink responses.

3.5.1.1. SCHC Sender-Abort

As defined in [RFC8724], a SCHC Sender-Abort can be triggered when the number of SCHC ACK REQ attempts is greater than or equal to MAX_ACK_REQUESTS. In the case of SCHC over Sigfox, a SCHC Sender-Abort MUST be sent if the number of repeated All-1s sent in sequence, without a Compound ACK reception inbetween, is greater than or equal to MAX_ACK_REQUESTS.

3.5.1.2. SCHC Receiver-Abort

As defined in [RFC8724], a SCHC Receiver-Abort is triggered when the receiver has no RuleID and DTag pairs available for a new session. In the case of this profile a SCHC Receiver-Abort MUST be sent if, for a single device, all the RuleIDs are being processed by the receiver (i.e., have an active session) at a certain time and a new one is requested, or if the RuleID of the fragment is not valid.

A SCHC Receiver-Abort MUST be triggered when the Inactivity Timer expires.

MAX_ACK_REQUESTS can be increased when facing high error rates.

Although a SCHC Receiver-Abort can be triggered at any point in time, a SCHC Receiver-Abort Downlink message MUST only be sent when there is a Downlink transmission opportunity.

3.5.1.3. Single-byte SCHC Header for Uplink Fragmentation

3.5.1.3.1. Uplink No-ACK Mode: Single-byte SCHC Header

Single-byte SCHC Header No-ACK mode MUST be used for transmitting short, non-critical packets that require fragmentation and do not require full reliability. This mode can be used by Uplink-only devices that do not support Downlink communications, or by bidirectional devices when they send non-critical data. Note that sending non-critical data by using a reliable fragmentation mode (which is only possible for bidirectional devices) may incur unnecessary overhead.

Since there are no multiple windows in the No-ACK mode, the W bit is not present. However, it MUST use the FCN field to indicate the size of the data packet. In this sense, the data packet would need to be split into X fragments and, similarly to the other fragmentation modes, the first transmitted fragment would need to be marked with FCN = X-1. Consecutive fragments MUST be marked with decreasing FCN values, having the last fragment marked with FCN = (All-1). Hence, even though the No-ACK mode does not allow recovering missing fragments, it allows indicating implicitly the size of the expected packet to the Network and hence detect at the receiver side whether all fragments have been received or not. In case the FCN field is not used to indicate the size of the data packet, the Network can detect whether all fragments have been received or not by using the integrity check.

When using the Single-byte SCHC Header for Uplink Fragmentation, the Fragmentation Header MUST be of 8 bit size, and the Fragment header is composed as follows:

* RuleID size: 3 bits

* DTag size (T): 0 bit

* Fragment Compressed Number (FCN) size (N): 5 bits

Other F/R parameters MUST be configured as follows:

* As per [RFC8724], in the No-ACK mode the W (window) field is not present.

* Regular tile size: 11 bytes

* All-1 tile size: 0 to 10 bytes

* Inactivity Timer: Application-dependent. The default value is 12 hours.

* RCS size: 5 bits

The maximum SCHC Packet size is 340 bytes.

Section 3.6.1 presents SCHC Fragment format examples and Section 5.1 provides fragmentation examples, using Single-byte SCHC Header No-ACK mode.

3.5.1.3.2. Uplink ACK-on-Error Mode: Single-byte SCHC Header

ACK-on-Error with single-byte header MUST be used for short to medium size packets that need to be sent reliably. ACK-on-Error is optimal for reliable SCHC Packet transmission over Sigfox transmissions, since it leads to a reduced number of ACKs in the lower capacity Downlink channel. Also, Downlink messages can be sent asynchronously and opportunistically. In contrast, ACK-Always would not minimize the number of ACKs, and No-ACK would not allow reliable transmission.

Allowing transmission of packets/files up to 300 bytes long, the SCHC Uplink Fragmentation Header size is 8 bits in size and is composed as follows:

* RuleID size: 3 bits

* DTag size (T): 0 bit

* Window index (W) size (M): 2 bits

* Fragment Compressed Number (FCN) size (N): 3 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

* WINDOW_SIZE: 7 (i.e., the maximum FCN value is 0b110)

* Regular tile size: 11 bytes

* All-1 tile size: 0 to 10 bytes

* Retransmission Timer: Application-dependent. The default value is 12 hours.

* Inactivity Timer: Application-dependent. The default value is 12 hours.

* RCS size: 3 bits

Section 3.6.2 presents SCHC Fragment format examples and Section 5.2 provides fragmentation examples, using ACK-on-Error with single-byte header.

3.5.1.4. Two-byte SCHC Header for Uplink Fragmentation

ACK-on-Error with two-byte header MUST be used for medium-large size packets that need to be sent reliably. ACK-on-Error is optimal for reliable SCHC Packet transmission over Sigfox, since it leads to a reduced number of ACKs in the lower capacity Downlink channel. Also, Downlink messages can be sent asynchronously and opportunistically. In contrast, ACK-Always would not minimize the number of ACKs, and No-ACK would not allow reliable transmission.

3.5.1.4.1. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1

In order to allow transmission of medium-large packets/files up to 480 bytes long, the SCHC Uplink Fragmentation Header size is 16 bits in size and composed as follows:

* RuleID size is: 6 bits

* DTag size (T) is: 0 bit

* Window index (W) size (M): 2 bits

* Fragment Compressed Number (FCN) size (N): 4 bits.

* RCS size: 4 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

* WINDOW_SIZE: 12 (with a maximum value of FCN=0b1011)

* Regular tile size: 10 bytes

* All-1 tile size: 1 to 10 bytes

* Retransmission Timer: Application-dependent. The default value is 12 hours.

* Inactivity Timer: Application-dependent. The default value is 12 hours.

Note that WINDOW_SIZE is limited to 12. This because, 4 windows (M = 2) with bitmaps of size 12 can be fitted in a single SCHC Compound ACK.

Section 3.6.3 presents SCHC Fragment format examples, using ACK-on-Error with two-byte header Option 1.

3.5.1.4.2. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 2

In order to allow transmission of very large packets/files up to 2400 bytes long, the SCHC Uplink Fragmentation Header size is 16 bits in size and composed as follows:

* RuleID size is: 8 bits

* DTag size (T) is: 0 bit

* Window index (W) size (M): 3 bits

* Fragment Compressed Number (FCN) size (N): 5 bits.

* RCS size: 5 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)

* Regular tile size: 10 bytes

* All-1 tile size: 0 to 9 bytes

* Retransmission Timer: Application-dependent. The default value is 12 hours.

* Inactivity Timer: Application-dependent. The default value is 12 hours.

Section 3.6.4 presents SCHC Fragment format examples, using ACK-on-Error with two-byte header Option 1.

3.5.1.5. All-1 SCHC Fragment and RCS behaviour

For ACK-on-Error, as defined in [RFC8724], it is expected that the last SCHC fragment of the last window will always be delivered with an All-1 FCN. Since this last window may not be full (i.e., it may be composed of fewer than WINDOW_SIZE fragments), an All-1 fragment may follow a value of FCN higher than 1 (0b01). In this case, the receiver cannot determine from the FCN values alone whether there are or not any missing fragments right before the All-1 fragment.

For Rules where the number of fragments in the last window is unknown, an RCS field MUST be used, indicating the number of fragments in the last window, including the All-1. With this RCS value, the receiver can detect if there are missing fragments before the All-1 and hence construct the corresponding SCHC ACK Bitmap accordingly, and send it in response to the All-1.

3.5.2. Downlink Fragmentation

In some LPWAN technologies, as part of energy-saving techniques, Downlink transmission is only possible immediately after an Uplink transmission. This allows the device to go in a very deep sleep mode and preserve battery, without the need to listen to any information from the network. This is the case for Sigfox-enabled devices, which can only listen to Downlink communications after performing an Uplink transmission and requesting a Downlink.

When there are fragments to be transmitted in the Downlink, an Uplink message is required to trigger the Downlink communication. In order to avoid potentially high delay for fragmented datagram transmission in the Downlink, the fragment receiver MAY perform an Uplink transmission as soon as possible after reception of a Downlink fragment that is not the last one. Such Uplink transmission MAY be triggered by sending a SCHC message, such as a SCHC ACK. However, other data messages can equally be used to trigger Downlink communications. The fragment receiver MUST send an Uplink transmission (e.g., empty message) and request a Downlink every 24 hours when no SCHC session is started. The use or not of this Uplink transmission (and the transmission rate, if used) will depend on application specific requirements.

Sigfox Downlink messages are fixed in size, and as described in [RFC8376] they can carry up to 8 bytes payload. Hence, a single SCHC Tile size per mode can be defined so that every Sigfox message always carries one SCHC Tile.

For reliable Downlink fragment transmission, the ACK-Always mode SHOULD be used. Note that ACK-on-Error does not guarantee Uplink feedback (since no SCHC ACK will be sent when no errors occur in a window), and No-ACK would not allow reliable transmission.

The SCHC Downlink Fragmentation Header size is 8 bits in size and is composed as follows:

* RuleID size: 3 bits

* DTag size (T): 0 bit

* Window index (W) size (M) is: 0 bit

* Fragment Compressed Number (FCN) size (N): 5 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)

* Regular tile size: 7 bytes

* All-1 tile size: 0 to 6 bytes

* Retransmission Timer: Application-dependent. The default value is 12 hours.

* Inactivity Timer: Application-dependent. The default value is 12 hours.

* RCS size: 5 bits

3.6. SCHC over Sigfox F/R Message Formats

This section depicts the different formats of SCHC Fragment, SCHC ACK (including the SCHC Compound ACK defined in [I-D.ietf-lpwan-schc-compound-ack]), and SCHC Abort used in SCHC over Sigfox.

3.6.1. Uplink No-ACK Mode: Single-byte SCHC Header

3.6.1.1. Regular SCHC Fragment

Figure 3 shows an example of a regular SCHC fragment for all fragments except the last one. As tiles are of 11 bytes, padding MUST NOT be added. The penultimate tile of a SCHC Packet is of regular size.

```
|- SCHC Fragment Header -|
+------------------------+---------+
|   RuleID    |   FCN    | Payload |
+-------------+----------+---------+
|   3 bits    |  5 bits  | 88 bits |
```

Figure 3: Regular SCHC Fragment format for all fragments except the last one

3.6.1.2. All-1 SCHC Fragment

Figure 4 shows an example of the All-1 message. The All-1 message MAY contain the last tile of the SCHC Packet. Padding MUST NOT be added, as the resulting size is L2-word-multiple.

The All-1 messages Fragment Header includes a 5-bit RCS, and 3 bits are added as padding to complete two bytes. The payload size of the All-1 message ranges from 0 to 80 bits.

```
|--------  SCHC Fragment Header -------|
+--------------------------------------+---------------+
| RuleID | FCN=ALL-1 |   RCS  |  b'000 |    Payload    |
+--------+-----------+--------+--------+---------------+
| 3 bits |  5 bits   | 5 bits | 3 bits | 0 to 80 bits  |
```

Figure 4: All-1 SCHC Message format with last tile

As per [RFC8724] the All-1 must be distinguishable from a SCHC Sender-Abort message (with same RuleID, and N values). The All-1 MAY have the last tile of the SCHC Packet. The SCHC Sender-Abort message header size is 1 byte, with no padding bits.

For the All-1 message to be distinguishable from the Sender-Abort message, the Sender-Abort message MUST be of 1 byte (only header with no padding). This way, the minimum size of the All-1 is 2 bytes, and the Sender-Abort message is 1 byte.

3.6.1.3. SCHC Sender-Abort Message format

```
          Sender-Abort
     |------ Header ------|
     +--------------------+
     | RuleID | FCN=ALL-1 |
     +--------+-----------+
     | 3 bits |  5 bits   |
```

Figure 5: SCHC Sender-Abort message format

3.6.2. Uplink ACK-on-Error Mode: Single-byte SCHC Header

3.6.2.1. Regular SCHC Fragment

Figure 6 shows an example of a regular SCHC fragment for all fragments except the last one. As tiles are of 11 bytes, padding MUST NOT be added.

```
|-- SCHC Fragment Header --|
+--------------------------+---------+
| RuleID |   W    |  FCN   | Payload |
+--------+--------+--------+---------+
| 3 bits | 2 bits | 3 bits | 88 bits |
```

Figure 6: Regular SCHC Fragment format for all fragments except the last one

The SCHC ACK REQ MUST NOT be used, instead the All-1 SCHC Fragment MUST be used to request a SCHC ACK from the receiver (Network SCHC). As per [RFC8724], the All-0 message is distinguishable from the SCHC ACK REQ (All-1 message). The penultimate tile of a SCHC Packet is of regular size.

3.6.2.2. All-1 SCHC Fragment

Figure 7 shows an example of the All-1 message. The All-1 message MAY contain the last tile of the SCHC Packet. Padding MUST NOT be added, as the resulting size is L2-word-multiple.

```
|-------------  SCHC Fragment Header -----------|
+-----------------------------------------------+--------------+
| RuleID |   W    | FCN=ALL-1 |  RCS   |b'00000 |   Payload    |
+--------+--------+-----------+--------+--------+--------------+
| 3 bits | 2 bits |  3 bits   | 3 bits | 5 bits | 0 to 80 bits |
```

Figure 7: All-1 SCHC Message format with last tile

As per [RFC8724] the All-1 must be distinguishable from a SCHC Sender-Abort message (with same RuleID, M, and N values). The All-1 MAY have the last tile of the SCHC Packet. The SCHC Sender-Abort message header size is 1 byte, with no padding bits.

For the All-1 message to be distinguishable from the Sender-Abort message, the Sender-Abort message MUST be of 1 byte (only header with no padding). This way, the minimum size of the All-1 is 2 bytes, and the Sender-Abort message is 1 byte.

3.6.2.3. SCHC ACK Format

Figure 8 shows the SCHC ACK format when all fragments have been correctly received (C=1). Padding MUST be added to complete the 64-bit Sigfox Downlink frame payload size.

```
|---- SCHC ACK Header ----|
+-------------------------+---------+
| RuleID |    W    | C=b'1 | b'0-pad |
+--------+---------+-------+---------+
| 3 bits | 2 bits | 1 bit | 58 bits |
```

Figure 8: SCHC Success ACK message format

In case SCHC fragment losses are found in any of the windows of the SCHC Packet (C=0), the SCHC Compound ACK defined in [I-D.ietf-lpwan-schc-compound-ack] MUST be used. The SCHC Compound ACK message format is shown in Figure 9.

```
|--- SCHC ACK Header ---|- W=w1 -|...|----- W=wi ------|
+------+--------+-------+--------+...+--------+--------+------+-------+
|RuleID| W=b'w1 | C=b'0 | Bitmap |...| W=b'wi | Bitmap | b'00 |b'0-pad|
+------+--------+-------+--------+...+--------+--------+------+-------+
|3 bits| 2 bits | 1 bit | 7 bits |...| 2 bits | 7 bits |2 bits|

   Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 9: SCHC Compound ACK message format

3.6.2.4. SCHC Sender-Abort Message format

```
|---- Sender-Abort Header ----|
+-----------------------------+
| RuleID | W=b'11 | FCN=ALL-1 |
+--------+--------+-----------+
| 3 bits | 2 bits |  3 bits   |
```

Figure 10: SCHC Sender-Abort message format

3.6.2.5. SCHC Receiver-Abort Message format

```
      |- Receiver-Abort Header -|
      +---------------------------------+-----------------+---------+
      | RuleID | W=b'11 | C=b'1 |  b'11 |  0xFF (all 1's) | b'0-pad |
      +--------+--------+-------+-------+-----------------+---------+
      | 3 bits | 2 bits | 1 bit | 2 bit |  8 bit          | 48 bits |
               next L2 Word boundary ->| <-- L2 Word --> |
```

Figure 11: SCHC Receiver-Abort message format

3.6.3. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1

3.6.3.1. Regular SCHC Fragment

Figure 12 shows an example of a regular SCHC fragment for all fragments except the last one. The penultimate tile of a SCHC Packet is of the regular size.

```
           |------- SCHC Fragment Header ------|
           +-----------------------------------+---------+
           | RuleID |    W   |  FCN   | b'0000 | Payload |
           +--------+--------+--------+--------+---------+
           | 6 bits | 2 bits | 4 bits | 4 bits | 80 bits |
```

Figure 12: Regular SCHC Fragment format for all fragments except the last one

The SCHC ACK REQ MUST NOT be used, instead the All-1 SCHC Fragment MUST be used to request a SCHC ACK from the receiver (Network SCHC). As per [RFC8724], the All-0 message is distinguishable from the SCHC ACK REQ (All-1 message).

3.6.3.2. All-1 SCHC Fragment

Figure 13 shows an example of the All-1 message. The All-1 message MUST contain the last tile of the SCHC Packet.

The All-1 message Fragment Header contains an RCS of 4 bits to complete the two-byte size. The size of the last tile ranges from 8 to 80 bits.

```
           |--------- SCHC Fragment Header -------|
           +--------------------------------------+--------------+
           | RuleID |    W   | FCN=ALL-1 |  RCS   |    Payload   |
           +--------+--------+-----------+--------+--------------+
           | 6 bits | 2 bits |  4 bits   | 4 bits | 8 to 80 bits |
```

Figure 13: All-1 SCHC message format with last tile

As per [RFC8724] the All-1 must be distinguishable from the SCHC Sender-Abort message (with same RuleID, M and N values). The All-1 MUST have the last tile of the SCHC Packet, that MUST be of at least 1 byte. The SCHC Sender-Abort message header size is 2 byte, with no padding bits.

For the All-1 message to be distinguishable from the Sender-Abort message, the Sender-Abort message MUST be of 2 byte (only header with no padding). This way, the minimum size of the All-1 is 3 bytes, and the Sender-Abort message is 2 bytes.

3.6.3.3. SCHC ACK Format

Figure 14 shows the SCHC ACK format when all fragments have been correctly received (C=1). Padding MUST be added to complete the 64-bit Sigfox Downlink frame payload size.

```
|---- SCHC ACK Header ----|
+-------------------------+---------+
| RuleID |    W    | C=b'1 | b'0-pad |
+--------+---------+-------+---------+
| 6 bits | 2 bits  | 1 bit | 55 bits |
```

Figure 14: SCHC Success ACK message format

The SCHC Compound ACK message MUST be used in case SCHC fragment losses are found in any window of the SCHC Packet (C=0). The SCHC Compound ACK message format is shown in Figure 15. The SCHC Compound ACK can report up to 4 windows with losses. as shown in Figure 16.

When sent in the Downlink, the SCHC Compound ACK MUST be 0 padded (Padding bits must be 0) to complement the 64 bits required by the Sigfox payload.

```
|--- SCHC ACK Header ---|- W=w1 -|...|---- W=wi -----|
+--------+------+-------+--------+...+------+--------+------+-------+
| RuleID |W=b'w1| C=b'0 | Bitmap |...|W=b'wi| Bitmap | b'00 |b'0-pad|
+--------+------+-------+--------+...+------+--------+------+-------+
| 6 bits |2 bits| 1 bit | 12 bits|...|2 bits| 12 bits|2 bits|

   Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 15: SCHC Compound ACK message format

```
       |- SCHC ACK Header -|- W=0 -|      |- W=1 -|...
       +------+------+-----+-------+------+-------+...
       |RuleID|W=b'00|C=b'0|Bitmap |W=b'01|Bitmap |...
       +------+------+-----+-------+------+-------+...
       |6 bits|2 bits|1 bit|12 bits|2 bits|12 bits|...

                   ...       |- W=2 -|      |- W=3 -|
                   ...+------+-------+------+-------+---+
                   ...|W=b'10|Bitmap |W=b'11|Bitmap |b'0|
                   ...+------+-------+------+-------+---+
                   ...|2 bits|12 bits|2 bits|12 bits|
```

Losses are found in windows W = w1,...,wi; where w1<w2<...<wi

Figure 16: SCHC Compound ACK message format example with losses in all windows

3.6.3.4. SCHC Sender-Abort Message Format

```
           |---- Sender-Abort Header ----|
           +-----------------------------+
           | RuleID |   W    | FCN=ALL-1 |
           +--------+--------+-----------+
           | 6 bits | 2 bits |  4 bits   |
```

Figure 17: SCHC Sender-Abort message format

3.6.3.5. SCHC Receiver-Abort Message Format

```
      |- Receiver-Abort Header -|
      +---------------------------------+-----------------+---------+
      | RuleID | W=b'11 | C=b'1 |  0x7F |  0xFF (all 1's) | b'0-pad |
      +--------+--------+-------+-------+-----------------+---------+
      | 6 bits | 2 bits | 1 bit | 7 bit |  8 bit          | 40 bits |
               next L2 Word boundary ->| <-- L2 Word --> |
```

Figure 18: SCHC Receiver-Abort message format

3.6.4. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 2

3.6.4.1. Regular SCHC Fragment

Figure 19 shows an example of a regular SCHC fragment for all fragments except the last one. The penultimate tile of a SCHC Packet is of the regular size.

```
|-- SCHC Fragment Header --|
+--------------------------+---------+
| RuleID |   W    | FCN    | Payload |
+--------+--------+--------+---------+
| 8 bits | 3 bits | 5 bits | 80 bits |
```

Figure 19: Regular SCHC Fragment format for all fragments except the last one

The SCHC ACK REQ MUST NOT be used, instead the All-1 SCHC Fragment MUST be used to request a SCHC ACK from the receiver (Network SCHC). As per [RFC8724], the All-0 message is distinguishable from the SCHC ACK REQ (All-1 message).

3.6.4.2. All-1 SCHC Fragment

Figure 20 shows an example of the All-1 message. The All-1 message MAY contain the last tile of the SCHC Packet.

The All-1 message Fragment Header contains an RCS of 5 bits, and 3 padding bits to complete a 3-byte Fragment Header. The size of the last tile, if present, ranges from 8 to 72 bits.

```
|-------------- SCHC Fragment Header -----------|
+-----------------------------------------------+---------------+
| RuleID |    W   | FCN=ALL-1 |  RCS   | b'000  |    Payload    |
+--------+--------+-----------+--------+--------+---------------+
| 8 bits | 3 bits |  5 bits   | 5 bits | 3 bits | 8 to 72 bits  |
```

Figure 20: All-1 SCHC message format with last tile

As per [RFC8724] the All-1 must be distinguishable from the SCHC Sender-Abort message (with same RuleID, M and N values). The SCHC Sender-Abort message header size is 2 byte, with no padding bits.

For the All-1 message to be distinguishable from the Sender-Abort message, the Sender-Abort message MUST be of 2 byte (only header with no padding). This way, the minimum size of the All-1 is 3 bytes, and the Sender-Abort message is 2 bytes.

3.6.4.3. SCHC ACK Format

Figure 21 shows the SCHC ACK format when all fragments have been correctly received (C=1). Padding MUST be added to complete the 64-bit Sigfox Downlink frame payload size.

```
                |---- SCHC ACK Header ----|
                +-------------------------+---------+
                | RuleID |    W   | C=b'1 | b'0-pad |
                +--------+--------+-------+---------+
                | 8 bits | 3 bits | 1 bit | 52 bits |
```

Figure 21: SCHC Success ACK message format

The SCHC Compound ACK message MUST be used in case SCHC fragment losses are found in any window of the SCHC Packet (C=0). The SCHC Compound ACK message format is shown in Figure 22. The SCHC Compound ACK can report up to 3 windows with losses.

When sent in the Downlink, the SCHC Compound ACK MUST be 0 padded (Padding bits must be 0) to complement the 64 bits required by the Sigfox payload.

```
   |-- SCHC ACK Header --|- W=w1 -|...|---- W=wi -----|
   +------+------+-------+--------+...+------+--------+------+-------+
   |RuleID|W=b'w1| C=b'0 | Bitmap |...|W=b'wi| Bitmap | 000  |b'0-pad|
   +------+------+-------+--------+...+------+--------+------+-------+
   |8 bits|3 bits| 1 bit | 31 bits|...|3 bits| 31 bits|3 bits|

        Losses are found in windows W = w1,...,wi; where w1<w2<...<wi
```

Figure 22: SCHC Compound ACK message format

3.6.4.4. SCHC Sender-Abort Message Format

```
               |---- Sender-Abort Header ----|
               +-----------------------------+
               | RuleID |   W    | FCN=ALL-1 |
               +--------+--------+-----------+
               | 8 bits | 3 bits |  5 bits   |
```

Figure 23: SCHC Sender-Abort message format

3.6.4.5. SCHC Receiver-Abort Message Format

```
     |-- Receiver-Abort Header -|
     +---------------------------------+-----------------+---------+
     | RuleID | W=b'111 | C=b'1 | b'1111 |  0xFF (all 1's) | b'0-pad |
     +--------+---------+-------+--------+-----------------+---------+
     | 8 bits |  3 bits | 1 bit | 4 bit |  8 bit          | 40 bits |
                next L2 Word boundary ->| <-- L2 Word --> |
```

Figure 24: SCHC Receiver-Abort message format

3.6.5. Downlink ACK-Always Mode: Single-byte SCHC Header

3.6.5.1. Regular SCHC Fragment

Figure 25 shows an example of a regular SCHC fragment for all fragments except the last one. The penultimate tile of a SCHC Packet is of the regular size.

```
     SCHC Fragment
  |--    Header   --|
  +-----------------+---------+
  | RuleID |  FCN   | Payload |
  +--------+--------+---------+
  | 3 bits | 5 bits | 56 bits |
```

Figure 25: Regular SCHC Fragment format for all fragments except the last one

The SCHC ACK MUST NOT be used, instead the All-1 SCHC Fragment MUST be used to request a SCHC ACK from the receiver. As per [RFC8724], the All-0 message is distinguishable from the SCHC ACK REQ (All-1 message).

3.6.5.2. All-1 SCHC Fragment

Figure 26 shows an example of the All-1 message. The All-1 message MAY contain the last tile of the SCHC Packet.

The All-1 message Fragment Header contains an RCS of 5 bits, and 3 padding bits to complete a 2-byte Fragment Header. The size of the last tile, if present, ranges from 8 to 48 bits.

```
|--------- SCHC Fragment Header -------|
+--------------------------------------+---------------+
| RuleID | FCN=ALL-1 |  RCS   | b'000  |    Payload    |
+--------+-----------+--------+--------+---------------+
| 3 bits |  5 bits   | 5 bits | 3 bits | 0 to 48 bits  |
```

Figure 26: All-1 SCHC message format with last tile

As per [RFC8724] the All-1 must be distinguishable from the SCHC Sender-Abort message (with same RuleID and N values). The SCHC Sender-Abort message header size is 1 byte, with no padding bits.

For the All-1 message to be distinguishable from the Sender-Abort message, the Sender-Abort message MUST be of 1 byte (only header with no padding). This way, the minimum size of the All-1 is 2 bytes, and the Sender-Abort message is 1 bytes.

3.6.5.3. SCHC ACK Format

Figure 27 shows the SCHC ACK format when all fragments have been correctly received (C=1). Padding MUST be added to complete 2 bytes.

```
          SCHC ACK
       |--   Header   --|
       +-----------------+---------+
       | RuleID | C=b'1 | b'0-pad |
       +--------+-------+---------+
       | 3 bits | 1 bit |  4 bits |
```

Figure 27: SCHC Success ACK message format

The SCHC ACK message format is shown in Figure 28.

```
 |---- SCHC ACK Header ----|
 +--------+-------+--------+---------+
 | RuleID | C=b'0 | Bitmap | b'0-pad |
 +--------+-------+--------+---------+
 | 3 bits | 1 bit | 31 bits|  5 bits |
```

Figure 28: SCHC Compound ACK message format

3.6.5.4. SCHC Sender-Abort Message Format

```
        Sender-Abort
     |----   Header   ----|
     +--------------------+
     | RuleID | FCN=ALL-1 |
     +--------+-----------+
     | 3 bits |  5 bits   |
```

Figure 29: SCHC Sender-Abort message format

3.6.5.5. SCHC Receiver-Abort Message Format

```
     Receiver-Abort
   |---  Header  ---|
   +----------------+--------+-----------------+
   | RuleID | C=b'1 | b'1111 |  0xFF (all 1's) |
   +--------+-------+--------+-----------------+
   | 3 bits | 1 bit | 4 bit  |  8 bit          |
```

Figure 30: SCHC Receiver-Abort message format

3.7. Padding

The Sigfox payload fields have different characteristics in Uplink and Downlink.

Uplink messages can contain a payload size from 0 to 12 bytes. The Sigfox radio protocol allows sending zero bits, one single bit of information for binary applications (e.g., status), or an integer number of bytes. Therefore, for 2 or more bits of payload it is required to add padding to the next integer number of bytes. The reason for this flexibility is to optimize transmission time and hence save battery consumption at the device.

Downlink frames on the other hand have a fixed length. The payload length MUST be 64 bits (i.e., 8 bytes). Hence, if less information bits are to be transmitted, padding MUST be used with bits equal to 0. The receiver MUST remove the added padding bits before the SCHC reassembly process.

4. Fragmentation Rules Examples

This section provides an example of RuleIDs configuration for interoperability between the F/R modes presented in this document. Note that the RuleID space for Uplink F/R is different than the one for Downlink F/R, therefore this section is divided in two subsections: Rules for Uplink fragmentation and Rules for Downlink fragmentation.

For Uplink F/R, multiple header length were described in Section 3.5. All of them are part of the SCHC over Sigfox Profile, and offer not only low protocol overhead for small payloads (single byte header) but also extensibility to transport larger payloads with more overhead (2 bytes header, option 1 and 2). The usage of the RuleID space for each header length is an implementation choice, but we provide an example of it in the following section. This illustrates implementation choices made in order to 1) identify the different header length, and 2) finally parse the RuleID field to identify the RuleID value and execute the associated treatment.

4.1. Uplink Fragmentation Rules Examples

The RuleID field for Uplink F/R modes have different sizes depending on the header length. In order to identify the header length and then the value of the RuleID, the RuleID field is interpreted as follows:

* The RuleID field is the first one to be parsed in the SCHC header, starting from the leftmost bits.

* For Single-byte SCHC Header F/R modes, it is expected a RuleID field of 3 bits:

  - if the first 3 leftmost bits have a value different than 0b'111, then it signals a Single-byte SCHC Header F/R mode,

  - if their value is 0b'111, then it signals a Two-byte SCHC Header F/R mode.

* For Single-byte SCHC Header F/R modes:

  - there are 7 RuleIDs available (with values from 0b'000-0b'110), the RuleID with value 0b'111 is reserved to indicate a Two-byte SCHC Header.

  - This set of rules is called "standard rules", and it is used to implement Single-byte SCHC header modes.

  - Each RuleID is associated with a set of properties defining if Uplink F/R is used and which Uplink F/R mode is used. As an example, the RuleID 0b'000 is mapped onto Uplink No-ACK Mode: Single-byte SCHC Header, and the RuleIDs 0b'001 and 0b'002 are mapped onto Uplink ACK-on-Error Mode: Single-byte SCHC Header (2 RuleIDs to allow for SCHC Packet interleaving).

* For Two-byte SCHC Header F/R modes at least 6 bits for the RuleID field are expected:

  - the 3 first leftmost bits are always 0b'111,

    o if the following 3 bits have a different value than 0b'111, then it signals the Two-byte SCHC Header Option 1,

    o if the following 3 bits are 0b'111, then it signals the Two-byte SCHC Header Option 2.

  - For the Two-byte SCHC Header Option 1, there are 7 RuleIDs available (0b'111000-0b'111110), 0b'111111 being reserved to indicate the Two-byte SCHC Header Option 2. This set of rules is called "extended rules", and it is used to implement the Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1.

- For the Two-byte SCHC Header Option 2, there are 2 additional bits to parse as the RuleID, so 4 RuleIDs available (0b'11111100-0b'11111111). This set of rules is used to cover specific cases that previous RuleIDs do not cover. As an example, RuleID 0b'00111111 is used to transport uncompressed IPv6 packets using the Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 2.

4.2. Downlink Fragmentation Rules Example

* For the Downlink ACK-Always Mode: Single-byte SCHC Header, RuleIDs can get values in ranges from 0b'000 to 0b'111.

5. Fragmentation Sequence Examples

In this section, some sequence diagrams depicting messages exchanges for different fragmentation modes and use cases are shown. In the examples, 'Seq' indicates the Sigfox Sequence Number of the frame carrying a fragment.

5.1. Uplink No-ACK Examples

The FCN field indicates the size of the data packet. The first fragment is marked with FCN = X-1, where X is the number of fragments the message is split into. All fragments are marked with decreasing FCN values. Last packet fragment is marked with the FCN = All-1 (1111).

Case No losses - All fragments are sent and received successfully.

```
Sender                     Receiver
  |-------FCN=6,Seq=1-------->|
  |-------FCN=5,Seq=2-------->|
  |-------FCN=4,Seq=3-------->|
  |-------FCN=3,Seq=4-------->|
  |-------FCN=2,Seq=5-------->|
  |-------FCN=1,Seq=6-------->|
  |-------FCN=15,Seq=7------->| All fragments received
(End)
```

Figure 31: Uplink No-ACK No-Losses

When the first SCHC fragment is received, the Receiver can calculate the total number of SCHC fragments that the SCHC Packet is composed of. For example, if the first fragment is numbered with FCN=6, the receiver can expect six more messages/fragments (i.e., with FCN going from 5 downwards, and the last fragment with an FCN equal to 15).

Case losses on any fragment except the first.

```
   Sender                     Receiver
     |-------FCN=6,Seq=1-------->|
     |-------FCN=5,Seq=2----X    |
     |-------FCN=4,Seq=3-------->|
     |-------FCN=3,Seq=4-------->|
     |-------FCN=2,Seq=5-------->|
     |-------FCN=1,Seq=6-------->|
     |-------FCN=15,Seq=7------->| Missing Fragment Unable to reassemble
   (End)
```

Figure 32: Uplink No-ACK Losses (scenario 1)

5.2. Uplink ACK-on-Error Examples: Single-byte SCHC Header

The single-byte SCHC header ACK-on-Error mode allows sending up to 28 fragments and packet sizes up to 300 bytes. The SCHC fragments may be delivered asynchronously and Downlink ACK can be sent opportunistically.

Case No losses

The Downlink flag must be enabled in the sender Uplink message to allow a Downlink message from the receiver. The Downlink Enable in the figures shows where the sender MUST enable the Downlink, and wait for an ACK.

```
           Sender                    Receiver
             |-----W=0,FCN=6,Seq=1----->|
             |-----W=0,FCN=5,Seq=2----->|
             |-----W=0,FCN=4,Seq=3----->|
             |-----W=0,FCN=3,Seq=4----->|
             |-----W=0,FCN=2,Seq=5----->|
             |-----W=0,FCN=1,Seq=6----->|
   DL Enable |-----W=0,FCN=0,Seq=7----->|
         (no ACK)
             |-----W=1,FCN=6,Seq=8----->|
             |-----W=1,FCN=5,Seq=9----->|
             |-----W=1,FCN=4,Seq=10---->|
   DL Enable |-----W=1,FCN=7,Seq=11---->| All fragments received
             |<- Compound ACK,W=1,C=1 --| C=1
           (End)
```

Figure 33: Uplink ACK-on-Error No-Losses

Case Fragment losses in first window

In this case, fragments are lost in the first window (W=0). After the first All-0 message arrives, the Receiver leverages the opportunity and sends a SCHC ACK with the corresponding bitmap and C=0.

After the loss fragments from the first window (W=0) are resent, the sender continues transmitting the fragments of the following window (W=1) without opening a reception opportunity. Finally, the All-1 fragment is sent, the Downlink is enabled, and the SCHC ACK is received with C=1. Note that the SCHC Compound ACK also uses a Sequence Number.

```
        Sender                    Receiver
          |-----W=0,FCN=6,Seq=1----->|
          |-----W=0,FCN=5,Seq=2--X   |
          |-----W=0,FCN=4,Seq=3----->|
          |-----W=0,FCN=3,Seq=4----->|
          |-----W=0,FCN=2,Seq=5--X   |                    __
          |-----W=0,FCN=1,Seq=6----->|                   | W=0
DL Enable |-----W=0,FCN=0,Seq=7----->| Missing Fragments<- FCN=5,Seq=2
          |<- Compound ACK,W=0,C=0 --| Bitmap:1011011    | FCN=2,Seq=5
          |-----W=0,FCN=5,Seq=9----->|                    --
          |-----W=0,FCN=2,Seq=10---->|
          |-----W=1,FCN=6,Seq=11---->|
          |-----W=1,FCN=5,Seq=12---->|
          |-----W=1,FCN=4,Seq=13---->|
DL Enable |-----W=1,FCN=7,Seq=14---->| All fragments received
          |<-Compound ACK,W=1,C=1 ---| C=1
        (End)
```

Figure 34: Uplink ACK-on-Error Losses on First Window

Case Fragment All-0 lost in first window (W=0)

In this example, the All-0 of the first window (W=0) is lost. Therefore, the Receiver waits for the next All-0 message of intermediate windows, or All-1 message of last window to generate the corresponding SCHC ACK, notifying the absence of the All-0 of window 0.

The sender resends the missing All-0 messages (with any other missing fragment from window 0) without opening a reception opportunity.

```
           Sender                          Receiver
             |-----W=0,FCN=6,Seq=1----->|
             |-----W=0,FCN=5,Seq=2----->|
             |-----W=0,FCN=4,Seq=3----->|
             |-----W=0,FCN=3,Seq=4----->|
             |-----W=0,FCN=2,Seq=5----->|
             |-----W=0,FCN=1,Seq=6----->| DL Enable
                     |-----W=0,FCN=0,Seq=7--X    |
         (no ACK)
             |-----W=1,FCN=6,Seq=8----->|
             |-----W=1,FCN=5,Seq=9----->|                   __
             |-----W=1,FCN=4,Seq=10---->|                  |W=0
    DL Enable |-----W=1,FCN=7,Seq=11---->| Missing Fragment<- FCN=0,Seq=7
             |<-Compound ACK,W=0,C=0 ---| Bitmap:1111110    |__
             |-----W=0,FCN=0,Seq=13---->| All fragments received
    DL Enable |-----W=1,FCN=7,Seq=14---->|
             |<-Compound ACK,W=1,C=1 ---| C=1
           (End)
```

Figure 35: Uplink ACK-on-Error All-0 Lost on First Window

In the following diagram, besides the All-0 there are other fragment losses in the first window (W=0).

```
           Sender                          Receiver
             |-----W=0,FCN=6,Seq=1----->|
             |-----W=0,FCN=5,Seq=2--X   |
             |-----W=0,FCN=4,Seq=3----->|
             |-----W=0,FCN=3,Seq=4--X   |
             |-----W=0,FCN=2,Seq=5----->|
             |-----W=0,FCN=1,Seq=6----->|
    DL Enable |-----W=0,FCN=0,Seq=7--X   |
         (no ACK)
             |-----W=1,FCN=6,Seq=8----->|
             |-----W=1,FCN=5,Seq=9----->|                   __
             |-----W=1,FCN=4,Seq=10---->|                  |W=0
    DL Enable |-----W=1,FCN=7,Seq=11---->| Missing Fragment<- FCN=5,Seq=2
             |<--Compound ACK,W=0,C=0 --| Bitmap:1010110    |FCN=3,Seq=4
             |-----W=0,FCN=5,Seq=13---->|                   |FCN=0,Seq=7
             |-----W=0,FCN=3,Seq=14---->|                    --
             |-----W=0,FCN=0,Seq=15---->| All fragments received
    DL Enable |-----W=1,FCN=7,Seq=16---->|
             |<-Compound ACK,W=1,C=1 ---| C=1
           (End)
```

Figure 36: Uplink ACK-on-Error All-0 and other Fragments Lost on First Window

In the next examples, there are fragment losses in both the first (W=0) and second (W=1) windows. The retransmission cycles after the All-1 is sent (i.e., not in intermediate windows) MUST always finish with an All-1, as it serves as an ACK Request message to confirm the correct reception of the retransmitted fragments.

```
          Sender                    Receiver
            |-----W=0,FCN=6,Seq=1----->|
            |-----W=0,FCN=5,Seq=2--X   |
            |-----W=0,FCN=4,Seq=3----->|
            |-----W=0,FCN=3,Seq=4--X   |                    __
            |-----W=0,FCN=2,Seq=5----->|                   |W=0
            |-----W=0,FCN=1,Seq=6----->|                   |FCN=5,Seq=2
  DL enable |-----W=0,FCN=0,Seq=7--X   |                   |FCN=3,Seq=4
       (no ACK)                                            |FCN=0,Seq=7
            |-----W=1,FCN=6,Seq=8--X   |                   |W=1
            |-----W=1,FCN=5,Seq=9----->|                   |FCN=6,Seq=8
            |-----W=1,FCN=4,Seq=10-X   |                   |FCN=4,Seq=10
  DL enable |-----W=1,FCN=7,Seq=11---->| Missing Fragment<-|__
            |<-Compound ACK,W=0,1,C=0--| Bitmap W=0:1010110
            |-----W=0,FCN=5,Seq=13---->|        W=1:0100001
            |-----W=0,FCN=3,Seq=14---->|
            |-----W=0,FCN=0,Seq=15---->|
            |-----W=1,FCN=6,Seq=16---->|
            |-----W=1,FCN=4,Seq=17---->| All fragments received
  DL enable |-----W=1,FCN=7,Seq=18---->|
            |<-Compound ACK,W=1,C=1----| C=1
          (End)
```

Figure 37: Uplink ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (1)

Similar case as above, but with fewer fragments in the second window (W=1)

```
          Sender                    Receiver
            |-----W=0,FCN=6,Seq=1----->|
            |-----W=0,FCN=5,Seq=2--X   |
            |-----W=0,FCN=4,Seq=3----->|
            |-----W=0,FCN=3,Seq=4--X   |
            |-----W=0,FCN=2,Seq=5----->|                    __
            |-----W=0,FCN=1,Seq=6----->|                   |W=0
  DL enable |-----W=0,FCN=0,Seq=7--X   |                   |FCN=5,Seq=2
        (no ACK)                                           |FCN=3,Seq=4
            |-----W=1,FCN=6,Seq=8--X   |                   |FCN=0,Seq=7
  DL enable |-----W=1,FCN=7,Seq=9----->| Missing Fragment--> W=1
            |<-Compound ACK,W=0,1, C=0-| Bitmap W=0:1010110,|FCN=6,Seq=8
            |-----W=0,FCN=5,Seq=11---->|        W=1:0000001 |__
            |-----W=0,FCN=3,Seq=12---->|
            |-----W=0,FCN=0,Seq=13---->|
            |-----W=1,FCN=6,Seq=14---->| All fragments received
  DL enable |-----W=1,FCN=7,Seq=15---->|
            |<-Compound ACK, W=1,C=1---| C=1
          (End)
```

Figure 38: Uplink ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (2)

Case SCHC ACK is lost

SCHC over Sigfox does not implement the SCHC ACK REQ message. Instead, it uses the SCHC All-1 message to request a SCHC ACK, when required.

```
          Sender                     Receiver
             |-----W=0,FCN=6,Seq=1----->|
             |-----W=0,FCN=5,Seq=2----->|
             |-----W=0,FCN=4,Seq=3----->|
             |-----W=0,FCN=3,Seq=4----->|
             |-----W=0,FCN=2,Seq=5----->|
             |-----W=0,FCN=1,Seq=6----->|
   DL Enable |-----W=0,FCN=0,Seq=7----->|
         (no ACK)
             |-----W=1,FCN=6,Seq=8----->|
             |-----W=1,FCN=5,Seq=9----->|
             |-----W=1,FCN=4,Seq=10---->|
   DL Enable |-----W=1,FCN=7,Seq=11---->| All fragments received
             | X--Compound ACK,W=1,C=1 -| C=1
   DL Enable |-----W=1,FCN=7,Seq=13---->| RESEND ACK
             |<-Compound ACK,W=1,C=1 ---| C=1
           (End)
```

Figure 39: Uplink ACK-on-Error ACK Lost

Case SCHC Compound ACK at the end

In this example, SCHC Fragment losses are found in both window 0 and 1. However, the sender does not send a SCHC ACK after the All-0 of window 0. Instead, it sends a SCHC Compound ACK notifying losses of both windows.

```
          Sender                            Receiver
            |-----W=0,FCN=6,Seq=1----->|
            |-----W=0,FCN=5,Seq=2--X   |
            |-----W=0,FCN=4,Seq=3----->|
            |-----W=0,FCN=3,Seq=4--X   |
            |-----W=0,FCN=2,Seq=5----->|
            |-----W=0,FCN=1,Seq=6----->|                     __
  DL enable |-----W=0,FCN=0,Seq=7----->| Waits for          |W=0
         (no ACK)                      next DL opportunity |FCN=5,Seq=2
            |-----W=1,FCN=6,Seq=8--X   |                    |FCN=3,Seq=4
  DL enable |-----W=1,FCN=7,Seq=9----->| Missing Fragment<-- W=1
            |<-Compound ACK,W=0,1, C=0-| Bitmap W=0:1010110 |FCN=6,Seq=8
            |-----W=0,FCN=5,Seq=11---->|        W=1:0000001  --
            |-----W=0,FCN=3,Seq=12---->|
            |-----W=1,FCN=6,Seq=13---->| All fragments received
  DL enable |-----W=1,FCN=7,Seq=14---->|
            |<-Compound ACK, W=1, C=1 -| C=1
          (End)
```

Figure 40: Uplink ACK-on-Error Fragments Lost on First and Second Windows with one Compound ACK

The number of times the same SCHC ACK message will be retransmitted is determined by the MAX_ACK_REQUESTS.

5.3. SCHC Abort Examples

Case SCHC Sender-Abort

The sender may need to send a Sender-Abort to stop the current communication. This may happen, for example, if the All-1 has been sent MAX_ACK_REQUESTS times.

```
        Sender                     Receiver
          |-----W=0,FCN=6,Seq=1----->|
          |-----W=0,FCN=5,Seq=2----->|
          |-----W=0,FCN=4,Seq=3----->|
          |-----W=0,FCN=3,Seq=4----->|
          |-----W=0,FCN=2,Seq=5----->|
          |-----W=0,FCN=1,Seq=6----->|
DL Enable |-----W=0,FCN=0,Seq=7----->|
      (no ACK)
          |-----W=1,FCN=6,Seq=8----->|
          |-----W=1,FCN=5,Seq=9----->|
          |-----W=1,FCN=4,Seq=10---->|
DL Enable |-----W=1,FCN=7,Seq=11---->| All fragments received
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |-----W=1,FCN=7,Seq=13---->| RESEND ACK  (1)
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |-----W=1,FCN=7,Seq=15---->| RESEND ACK  (2)
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |-----W=1,FCN=7,Seq=17---->| RESEND ACK  (3)
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |-----W=1,FCN=7,Seq=18---->| RESEND ACK  (4)
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |-----W=1,FCN=7,Seq=19---->| RESEND ACK  (5)
          | X--Compound ACK,W=1,C=1 -| C=1
DL Enable |----Sender-Abort,Seq=20-->| exit with error condition
        (End)
```

Figure 41: Uplink ACK-on-Error Sender-Abort

Case Receiver-Abort

The receiver may need to send a Receiver-Abort to stop the current communication. This message can only be sent after a Downlink enable.

```
        Sender                      Receiver
          |-----W=0,FCN=6,Seq=1----->|
          |-----W=0,FCN=5,Seq=2----->|
          |-----W=0,FCN=4,Seq=3----->|
          |-----W=0,FCN=3,Seq=4----->|
          |-----W=0,FCN=2,Seq=5----->|
          |-----W=0,FCN=1,Seq=6----->|
DL Enable |-----W=0,FCN=0,Seq=7----->|
          |<------  RECV ABORT ------| under-resourced
       (Error)
```

Figure 42: Uplink ACK-on-Error Receiver-Abort

6. Security considerations

The radio protocol authenticates and ensures the integrity of each message. This is achieved by using a unique device ID and an AES-128 based message authentication code, ensuring that the message has been generated and sent by the device (see [sigfox-spec] Section 3.8) or network (see [sigfox-spec] Section 4.3) with the ID claimed in the message [sigfox-spec].

Application data can be encrypted at the application layer or not, depending on the criticality of the use case. This flexibility allows providing a balance between cost and effort vs. risk. AES-128 in counter mode is used for encryption. Cryptographic keys are independent for each device. These keys are associated with the device ID and separate integrity and encryption keys are pre-provisioned. An encryption key is only provisioned if confidentiality is to be used (see [sigfox-spec] Section 5.3. Note that further documentation is available at Sigfox upon request).

The radio protocol has protections against replay attacks, and the cloud-based core network provides firewalling protection against undesired incoming communications [sigfox-spec].

The previously described security mechanisms do not guarantee an E2E security between the Device SCHC C/D + F/R and the Network SCHC C/D + F/R: potential security threats described in [RFC8724] are applicable to the profile specified in this document.

In some circumstances, sending device location information is privacy-sensitive. The Device Geolocation parameter provided by the Network is optional, therefore it can be omitted to protect this aspect of the device privacy.

7. IANA Considerations

This document has no IANA actions.

8. Acknowledgements

Carles Gomez has been funded in part by the Spanish Government through the Jose Castillejo CAS15/00336 grant, the TEC2016-79988-P grant, and the PID2019-106808RA-I00 grant, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

Sergio Aguilar has been funded by the ERDF and the Spanish Government through project TEC2016-79988-P and project PID2019-106808RA-I00, AEI/FEDER, EU.

Sandra Cespedes has been funded in part by the ANID Chile Project FONDECYT Regular 1201893 and Basal Project FB0008.

Diego Wistuba has been funded by the ANID Chile Project FONDECYT Regular 1201893.

The authors would like to thank Ana Minaburo, Clement Mannequin, Rafael Vidal, Julien Boite, Renaud Marty, and Antonis Platis for their useful comments and implementation design considerations.

9. References

9.1. Normative References

[I-D.ietf-lpwan-schc-compound-ack]
Zuniga, JC., Gomez, C., Aguilar, S., Toutain, L., Cespedes, S., and D. Wistuba, "SCHC Compound ACK", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-compound-ack-10, January 2023, <httpsout-://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-compound-ack-10.txt>.

[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>.

[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>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[sigfox-spec] Sigfox, "Sigfox Radio Specifications", <https://build.sigfox.com/sigfox-device-radio-specifications>.

9.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P. V. D., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-11, 17 January 2021, <https://www.ietf.org/archive/id/draft-ietf-core-comi-11.txt>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[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>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[sigfox-callbacks] Sigfox, "Sigfox Callbacks", <https://support.sigfox.com/docs/callbacks-documentation>.

[sigfox-docs] Sigfox, "Sigfox Documentation", <https://support.sigfox.com/docs>.

Authors' Addresses

Juan Carlos Zuniga
Montreal QC
Canada
Email: j.c.zuniga@ieee.org

Carles Gomez
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: carles.gomez@upc.edu

Sergio Aguilar
Universitat Politecnica de Catalunya
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: sergio.aguilar.romero@upc.edu

Laurent Toutain
IMT-Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France
Email: Laurent.Toutain@imt-atlantique.fr

Sandra Cespedes
Concordia University
1455 De Maisonneuve Blvd. W.
Montreal QC, H3G 1M8
Canada
Email: sandra.cespedes@concordia.ca

Diego Wistuba
NIC Labs, Universidad de Chile
Av. Almte. Blanco Encalada 1975
Santiago
Chile
Email: wistuba@niclabs.cl

Julien Boite
Unabiz (Sigfox)
Labege
France
Email: julien.boite@unabiz.com
URI: https://www.sigfox.com/

lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: 12 April 2023 Institut MINES TELECOM; IMT Atlantique
9 October 2022

Data Model for Static Context Header Compression (SCHC)
draft-ietf-lpwan-schc-yang-data-model-21

Abstract

This document describes a YANG data model for the SCHC (Static Context Header Compression) compression and fragmentation rules.

This document formalizes the description of the rules for better interoperability between SCHC instances either to exchange a set of rules or to modify some rules parameters.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

SCHC is a compression and fragmentation mechanism for constrained networks defined in [RFC8724]. It is based on a static context shared by two entities at the boundary of the constrained network. [RFC8724] provides an informal representation of the rules used either for compression/decompression (or C/D) or fragmentation/reassembly (or F/R). The goal of this document is to formalize the description of the rules to offer:

* the same definition on both ends, even if the internal representation is different;

* an update of the other end to set up some specific values (e.g. IPv6 prefix, destination address,...).

[I-D.ietf-lpwan-architecture] illustrates the exchange of rules using the YANG data model.

This document defines a YANG module [RFC7950] to represent both compression and fragmentation rules, which leads to common representation for values for all the rules elements.

2. Requirements Language

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. Terminology

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

* App: LPWAN Application, as defined by [RFC8376]. An application sending/receiving packets to/from the Dev.

* Bi: Bidirectional. Characterizes a Field Descriptor that applies to headers of packets traveling in either direction (Up and Dw, see this glossary).

* CDA: Compression/Decompression Action. 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.

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

* Dev: Device, as defined by [RFC8376].

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

* DI: Direction Indicator. This field tells which direction of packet travel (Up, Dw or Bi) a Field Description applies to. This allows for asymmetric processing, using the same Rule.

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

* FID: Field Identifier. This identifies the protocol and field a Field Description applies to.

* FL: Field Length is the length of the original packet header field. It is expressed as a number of bits for header fields of fixed lengths or as a type (e.g., variable, token length, ...) for field lengths that are unknown at the time of Rule creation. The length of a header field is defined in the corresponding protocol specification (such as IPv6 or UDP).

* FP: when a Field is expected to appear multiple times in a header, Field Position specifies the occurrence this Field Description applies to (for example, first uri-path option, second uri-path, etc. in a CoAP header), counting from 1. The value 0 is special and means "don’t care", see [RFC8724] Section 7.2.

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

* 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.

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

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

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

* Up: Uplink direction for compression/decompression, from the Dev SCHC C/D to the network SCHC C/D.

4. SCHC rules

SCHC compression is generic, the main mechanism does not refer to a specific protocol. Any header field is abstracted through an Field Identifier (FID), a position (FP), a direction (DI), and a value that can be a numerical value or a string. [RFC8724] and [RFC8824] specify fields for IPv6 [RFC8200], UDP[RFC0768], CoAP [RFC7252] including options defined for no server response [RFC7967] and OSCORE [RFC8613]. For the latter [RFC8824] splits this field into sub-fields.

SCHC fragmentation requires a set of common parameters that are included in a rule. These parameters are defined in [RFC8724].

The YANG data model enables the compression and the fragmentation selection using the feature statement.

4.1. Compression Rules

[RFC8724] proposes an informal representation of the compression rule. A compression context for a device is composed of a set of rules. Each rule contains information to describe a specific field in the header to be compressed.

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

Figure 1: Compression Decompression Context

4.2. Identifier generation

Identifiers used in the SCHC YANG data model are from the identityref statement to ensure global uniqueness and easy augmentation if needed. The principle to define a new type based on a group of identityref is the following:

* define a main identity ending with the keyword base-type.

* derive all the identities used in the Data Model from this base type.

* create a typedef from this base type.

The example (Figure 2) shows how an identityref is created for RCS (Reassembly Check Sequence) algorithms used during SCHC fragmentation.

```
     identity rcs-algorithm-base-type {
       description
         "Identify which algorithm is used to compute RCS.
          The algorithm also defines the size of the RCS field.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity rcs-crc32 {
       base rcs-algorithm-base-type;
       description
         "CRC 32 defined as default RCS in RFC8724. This RCS is
          4 bytes long.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef rcs-algorithm-type {
       type identityref {
         base rcs-algorithm-base-type;
       }
       description
         "Define the type for RCS algorithm in rules.";
     }
```

Figure 2: Principle to define a type based on identityref.

4.3. Convention for Field Identifier

In the process of compression, the headers of the original packet are first parsed to create a list of fields. This list of fields is matched against the rules to find the appropriate rule and apply compression. [RFC8724] does not state how the field ID value is constructed. In examples, identification is done through a string indexed by the protocol name (e.g. IPv6.version, CoAP.version,...).

The current YANG data model includes fields definitions found in [RFC8724], [RFC8824].

Using the YANG data model, each field MUST be identified through a global YANG identityref.
A YANG field ID for the protocol is always derived from the fid-base-type. Then an identity for each protocol is specified using the naming convention fid-<<protocol name>>-base-type. All possible fields for this protocol MUST derive from the protocol identity. The naming convention is "fid-" followed by the protocol name and the field name. If a field has to be divided into sub-fields, the field identity serves as a base.

The full field-id definition is found in Section 6. A type is defined for IPv6 protocol, and each field is based on it. Note that the DiffServ bits derive from the Traffic Class identity.

4.4. Convention for Field length

Field length is either an integer giving the size of a field in bits or a specific function. [RFC8724] defines the "var" function which allows variable length fields (whose length is expressed in bytes) and [RFC8824] defines the "tkl" function for managing the CoAP Token length field.

The naming convention is "fl-" followed by the function name.

The field length function can be defined as an identityref as described in Section 6. Therefore, the type for field length is a union between an integer giving the size of the length in bits and the identityref.

4.5. Convention for Field position

Field position is a positive integer which gives the occurrence times of a specific field from the header start. The default value is 1, and incremented at each repetition. Value 0 indicates that the position is not important and is not considered during the rule selection process.

Field position is a positive integer. The type is uint8.

4.6. Convention for Direction Indicator

The Direction Indicator (di) is used to tell if a field appears in both directions (Bi) or only uplink (Up) or Downlink (Dw). The naming convention is "di" followed by the Direction Indicator name.

The type is "di-type".

4.7. Convention for Target Value

The Target Value is a list of binary sequences of any length, aligned to the left. In the rule, the structure will be used as a list, with index as a key. The highest index value is used to compute the size of the index sent in residue for the match-mapping CDA (Compression Decompression Action). The index can specify several values:

* For Equal and MSB, Target Value contains a single element. Therefore, the index is set to 0.

* For match-mapping, Target Value can contain several elements. Index values MUST start from 0 and MUST be contiguous.

If the header field contains text, the binary sequence uses the same encoding.

4.8. Convention for Matching Operator

Matching Operator (MO) is a function applied between a field value provided by the parsed header and the target value. [RFC8724] defines 4 MO.

The naming convention is "mo-" followed by the MO name.

The type is "mo-type"

4.8.1. Matching Operator arguments

They are viewed as a list, built with a tv-struct (see Section 4.7).

4.9. Convention for Compression Decompression Actions

Compression Decompression Action (CDA) identifies the function to use for compression or decompression. [RFC8724] defines 6 CDA.

The naming convention is "cda-" followed by the CDA name.

4.9.1. Compression Decompression Action arguments

Currently no CDA requires arguments, but in the future some CDA may require one or several arguments. They are viewed as a list, of target-value type.

4.10. Fragmentation rule

Fragmentation is optional in the data model and depends on the presence of the "fragmentation" feature.

Most of the fragmentation parameters are listed in Annex D of [RFC8724].

Since fragmentation rules work for a specific direction, they MUST contain a mandatory direction indicator. The type is the same as the one used in compression entries, but bidirectional MUST NOT be used.

4.10.1. Fragmentation mode

[RFC8724] defines 3 fragmentation modes:

* No Ack: this mode is unidirectional, no acknowledgment is sent back.

* Ack Always: each fragmentation window must be explicitly acknowledged before going to the next.

* Ack on Error: A window is acknowledged only when the receiver detects some missing fragments.

The type is "fragmentation-mode-type". The naming convention is "fragmentation-mode-" followed by the fragmentation mode name.

4.10.2. Fragmentation Header

A data fragment header, starting with the rule ID, can be sent in the fragmentation direction. [RFC8724] indicates that the SCHC header may be composed of (cf. Figure 3):

* a Datagram Tag (Dtag) identifying the datagram being fragmented if the fragmentation applies concurrently on several datagrams. This field is optional and its length is defined by the rule.

* a Window (W) used in Ack-Always and Ack-on-Error modes. In Ack-Always, its size is 1. In Ack-on-Error, it depends on the rule. This field is not needed in No-Ack mode.

* a Fragment Compressed Number (FCN) indicating the fragment/tile position within the window. This field is mandatory on all modes defined in [RFC8724], its size is defined by the rule.

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

Figure 3: Data fragment header from RFC8724

4.10.3. Last fragment format

The last fragment of a datagram is sent with an RCS (Reassembly Check Sequence) field to detect residual transmission error and possible losses in the last window. [RFC8724] defines a single algorithm based on Ethernet CRC computation.

The naming convention is "rcs-" followed by the algorithm name.

For Ack-on-Error mode, the All-1 fragment may just contain the RCS or can include a tile. The parameters define the behavior:

* all-1-data-no: the last fragment contains no data, just the RCS

* all-1-data-yes: the last fragment includes a single tile and the RCS

* all-1-data-sender-choice: the last fragment may or may not contain a single tile. The receiver can detect if a tile is present.

The naming convention is "all-1-data-" followed by the behavior identifier.

4.10.4. Acknowledgment behavior

The acknowledgment fragment header goes in the opposite direction of data. [RFC8724] defines the header, composed of (see Figure 4):

* a Dtag (if present).

* a mandatory window as in the data fragment.

* a C bit giving the status of RCS validation. In case of failure, a bitmap follows, indicating the received tile.

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

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

Figure 4: Acknowledgment fragment header for RFC8724

For Ack-on-Error, SCHC defines when an acknowledgment can be sent. This can be at any time defined by the layer 2, at the end of a window (FCN all-0) or as a response to receiving the last fragment (FCN all-1). The naming convention is "ack-behavior" followed by the algorithm name.

4.10.5. Timer values

The state machine requires some common values to handle fragmentation correctly.

* retransmission-timer gives the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specified, value MUST be strictly positive.

* inactivity-timer gives the duration before aborting a fragmentation session (cf. section 8.2.2.4. of [RFC8724]). The value 0 explicitly indicates that this timer is disabled.

[RFC8724] do not specify any range for these timers. [RFC9011] recommends a duration of 12 hours. In fact, the value range should be between milliseconds for real time systems to several days. To allow a large range of applications, two parameters must be specified:

* the duration of a tick. It is computed by this formula 2^tick-duration/10^6. When tick-duration is set to 0, the unit is the microsecond. The default value of 20 leads to a unit of 1.048575 second. A value of 32 leads to a tick duration of about 1 hour 11 minutes.

* the number of ticks in the predefined unit. With the default tick-duration value of 20, the timers can cover a range between 1.0 sec and 19 hours covering [RFC9011] recommendation.

4.10.6. Fragmentation Parameter

The SCHC fragmentation protocol specifies the number of attempts before aborting through the parameter:

* max-ack-requests (cf. section 8.2.2.4. of [RFC8724]).

4.10.7. Layer 2 parameters

The data model includes two parameters needed for fragmentation:

* l2-word-size: [RFC8724] base fragmentation, in bits, on a layer 2 word which can be of any length. The default value is 8 and correspond to the default value for byte aligned layer 2. A value of 1 will indicate that there is no alignment and no need for padding.

* maximum-packet-size: defines the maximum size of an uncompressed datagram. By default, the value is set to 1280 bytes.

They are defined as unsigned integers, see Section 6.

5. Rule definition

A rule is identified by a unique rule identifier (rule ID) comprising both a Rule ID value and a Rule ID length. The YANG grouping rule-id-type defines the structure used to represent a rule ID. A length of 0 is allowed to represent an implicit rule.

Three natures of rules are defined in [RFC8724]:

* Compression: a compression rule is associated with the rule ID.

* No compression: this identifies the default rule used to send a packet integrally when no compression rule was found (see [RFC8724] section 6).

* Fragmentation: fragmentation parameters are associated with the rule ID. Fragmentation is optional and feature "fragmentation" should be set.

The YANG data model introduces respectively these three identities :

* nature-compression

* nature-no-compression

* nature-fragmentation

The naming convention is "nature-" followed by the nature identifier.

To access a specific rule, the rule ID length and value are used as a key. The rule is either a compression or a fragmentation rule.

5.1. Compression rule

A compression rule is composed of entries describing its processing. An entry contains all the information defined in Figure 1 with the types defined above.

The compression rule described Figure 1 is defined by compression-content. It defines a list of compression-rule-entry, indexed by their field id, position and direction. The compression-rule-entry element represent a line of the table Figure 1. Their type reflects the identifier types defined in Section 4.1

Some checks are performed on the values:

* target value MUST be present for MO different from ignore.

* when MSB MO is specified, the matching-operator-value must be present

5.2. Fragmentation rule

A Fragmentation rule is composed of entries describing the protocol behavior. Some on them are numerical entries, others are identifiers defined in Section 4.10.

5.3. YANG Tree

The YANG data model described in this document conforms to the Network Management Datastore Architecture defined in [RFC8342].

```
module: ietf-schc
  +--rw schc
     +--rw rule* [rule-id-value rule-id-length]
        +--rw rule-id-value                  uint32
        +--rw rule-id-length                 uint8
        +--rw rule-nature                    nature-type
        +--rw (nature)?
           +--:(fragmentation) {fragmentation}?
           |  +--rw fragmentation-mode
           |  |       schc:fragmentation-mode-type
           |  +--rw l2-word-size?            uint8
           |  +--rw direction                schc:di-type
           |  +--rw dtag-size?               uint8
           |  +--rw w-size?                  uint8
           |  +--rw fcn-size                 uint8
           |  +--rw rcs-algorithm?           rcs-algorithm-type
           |  +--rw maximum-packet-size?     uint16
           |  +--rw window-size?             uint16
           |  +--rw max-interleaved-frames?   uint8
           |  +--rw inactivity-timer
           |  |  +--rw ticks-duration?   uint8
           |  |  +--rw ticks-numbers?    uint16
           |  +--rw retransmission-timer
           |  |  +--rw ticks-duration?   uint8
```

```
             |  |  +--rw ticks-numbers?    uint16
             |  +--rw max-ack-requests?         uint8
             |  +--rw (mode)?
             |     +--:(no-ack)
             |     +--:(ack-always)
             |     +--:(ack-on-error)
             |        +--rw tile-size?          uint8
             |        +--rw tile-in-all-1?      schc:all-1-data-type
             |        +--rw ack-behavior?       schc:ack-behavior-type
             +--:(compression) {compression}?
                +--rw entry*
                        [field-id field-position direction-indicator]
                   +--rw field-id                    schc:fid-type
                   +--rw field-length                schc:fl-type
                   +--rw field-position              uint8
                   +--rw direction-indicator         schc:di-type
                   +--rw target-value* [index]
                   |  +--rw index    uint16
                   |  +--rw value?   binary
                   +--rw matching-operator           schc:mo-type
                   +--rw matching-operator-value* [index]
                   |  +--rw index    uint16
                   |  +--rw value?   binary
                   +--rw comp-decomp-action          schc:cda-type
                   +--rw comp-decomp-action-value* [index]
                      +--rw index    uint16
                      +--rw value?   binary
```

Figure 5: Overview of SCHC data model

6. YANG Module

```
<CODE BEGINS> file "ietf-schc@2022-10-09.yang"
module ietf-schc {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schc";
  prefix schc;

  organization
    "IETF IPv6 over Low Power Wide-Area Networks (lpwan) working
     group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/lpwan/about/>
     WG List:  <mailto:lp-wan@ietf.org>
     Editor:   Laurent Toutain
       <mailto:laurent.toutain@imt-atlantique.fr>
     Editor:   Ana Minaburo
       <mailto:ana@ackl.io>";
```

```
     description
       "
        Copyright (c) 2022 IETF Trust and the persons identified as
        authors of the code.  All rights reserved.

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject to
        the license terms contained in, the Revised BSD License set
        forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX
        (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
        for full legal notices.

        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 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.

        ***************************************************************

        Generic Data model for Static Context Header Compression Rule
        for SCHC, based on RFC 8724 and RFC8824. Include compression,
        no compression and fragmentation rules.

        This module is a YANG model for SCHC rules (RFC 8724 and
        RFC8824). RFC 8724 describes compression rules in a abstract
        way through a table.

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

        This module specifies a global data model that can be used for
        rule exchanges or modification. It specifies both the data model
        format and the global identifiers used to describe some
```

```
        operations in fields.
        This data model applies to both compression and fragmentation.";

     revision 2022-10-09 {
       description
         "Initial version from RFC XXXX.";
       reference
         "RFC XXX: Data Model for Static Context Header Compression
          (SCHC)";
     }

     feature compression {
       description
         "SCHC compression capabilities are taken into account.";
     }

     feature fragmentation {
       description
         "SCHC fragmentation capabilities are taken into account.";
     }

     // -------------------------
     //  Field ID type definition
     //--------------------------
     // generic value TV definition

     identity fid-base-type {
       description
         "Field ID base type for all fields.";
     }

     identity fid-ipv6-base-type {
       base fid-base-type;
       description
         "Field ID base type for IPv6 headers described in RFC 8200.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
     }

     identity fid-ipv6-version {
       base fid-ipv6-base-type;
       description
         "IPv6 version field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
     }

     identity fid-ipv6-trafficclass {
```

```
       base fid-ipv6-base-type;
       description
         "IPv6 Traffic Class field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
     }

     identity fid-ipv6-trafficclass-ds {
       base fid-ipv6-trafficclass;
       description
         "IPv6 Traffic Class field: DiffServ field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification,
          RFC 3168 The Addition of Explicit Congestion Notification
                   (ECN) to IP";
     }

     identity fid-ipv6-trafficclass-ecn {
       base fid-ipv6-trafficclass;
       description
         "IPv6 Traffic Class field: ECN field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification,
          RFC 3168 The Addition of Explicit Congestion Notification
                   (ECN) to IP";
     }

     identity fid-ipv6-flowlabel {
       base fid-ipv6-base-type;
       description
         "IPv6 Flow Label field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
     }

     identity fid-ipv6-payload-length {
       base fid-ipv6-base-type;
       description
         "IPv6 Payload Length field.";
       reference
         "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
     }

     identity fid-ipv6-nextheader {
       base fid-ipv6-base-type;
       description
         "IPv6 Next Header field.";
       reference
```

```
        "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
    }

    identity fid-ipv6-hoplimit {
      base fid-ipv6-base-type;
      description
        "IPv6 Next Header field.";
      reference
        "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
    }

    identity fid-ipv6-devprefix {
      base fid-ipv6-base-type;
      description
        "Corresponds to either the source address or the destination
         address prefix of RFC 8200 depending on whether it is an
         uplink or a downlink message.";
      reference
        "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
    }

    identity fid-ipv6-deviid {
      base fid-ipv6-base-type;
      description
        "Corresponds to either the source address or the destination
         address IID of RFC 8200 depending on whether it is an uplink
         or a downlink message.";
      reference
        "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
    }

    identity fid-ipv6-appprefix {
      base fid-ipv6-base-type;
      description
        "Corresponds to either the source address or the destination
         address prefix of RFC 8200 depending on whether it is an
         uplink or a downlink message.";
      reference
        "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
    }

    identity fid-ipv6-appiid {
      base fid-ipv6-base-type;
      description
        "Corresponds to either the source address or the destination
         address IID of RFC 8200 depending on whether it is an uplink
         or a downlink message.";
      reference
```

```
           "RFC 8200 Internet Protocol, Version 6 (IPv6) Specification";
       }

       identity fid-udp-base-type {
         base fid-base-type;
         description
           "Field ID base type for UDP headers described in RFC 768.";
         reference
           "RFC 768 User Datagram Protocol";
       }

       identity fid-udp-dev-port {
         base fid-udp-base-type;
         description
           "UDP source or destination port, if uplink or downlink
            communication, respectively.";
         reference
           "RFC 768 User Datagram Protocol";
       }

       identity fid-udp-app-port {
         base fid-udp-base-type;
         description
           "UDP destination or source port, if uplink or downlink
            communication, respectively.";
         reference
           "RFC 768 User Datagram Protocol";
       }

       identity fid-udp-length {
         base fid-udp-base-type;
         description
           "UDP length.";
         reference
           "RFC 768 User Datagram Protocol";
       }

       identity fid-udp-checksum {
         base fid-udp-base-type;
         description
           "UDP length.";
         reference
           "RFC 768 User Datagram Protocol";
       }

       identity fid-coap-base-type {
         base fid-base-type;
         description
```

```
         "Field ID base type for UDP headers described.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-version {
       base fid-coap-base-type;
       description
         "CoAP version.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-type {
       base fid-coap-base-type;
       description
         "CoAP type.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-tkl {
       base fid-coap-base-type;
       description
         "CoAP token length.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-code {
       base fid-coap-base-type;
       description
         "CoAP code.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-code-class {
       base fid-coap-code;
       description
         "CoAP code class.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-code-detail {
       base fid-coap-code;
       description
```

```
         "CoAP code detail.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-mid {
       base fid-coap-base-type;
       description
         "CoAP message ID.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-token {
       base fid-coap-base-type;
       description
         "CoAP token.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-option-if-match {
       base fid-coap-base-type;
       description
         "CoAP option If-Match.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-option-uri-host {
       base fid-coap-base-type;
       description
         "CoAP option URI-Host.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-option-etag {
       base fid-coap-base-type;
       description
         "CoAP option Etag.";
       reference
         "RFC 7252 The Constrained Application Protocol (CoAP)";
     }

     identity fid-coap-option-if-none-match {
       base fid-coap-base-type;
       description
```

```
          "CoAP option if-none-match.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-observe {
        base fid-coap-base-type;
        description
          "CoAP option Observe.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-uri-port {
        base fid-coap-base-type;
        description
          "CoAP option Uri-Port.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-location-path {
        base fid-coap-base-type;
        description
          "CoAP option Location-Path.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-uri-path {
        base fid-coap-base-type;
        description
          "CoAP option Uri-Path.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-content-format {
        base fid-coap-base-type;
        description
          "CoAP option Content Format.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-max-age {
        base fid-coap-base-type;
        description
```

```
        "CoAP option Max-Age.";
      reference
        "RFC 7252 The Constrained Application Protocol (CoAP)";
    }

    identity fid-coap-option-uri-query {
      base fid-coap-base-type;
      description
        "CoAP option Uri-Query.";
      reference
        "RFC 7252 The Constrained Application Protocol (CoAP)";
    }

    identity fid-coap-option-accept {
      base fid-coap-base-type;
      description
        "CoAP option Accept.";
      reference
        "RFC 7252 The Constrained Application Protocol (CoAP)";
    }

    identity fid-coap-option-location-query {
      base fid-coap-base-type;
      description
        "CoAP option Location-Query.";
      reference
        "RFC 7252 The Constrained Application Protocol (CoAP)";
    }

    identity fid-coap-option-block2 {
      base fid-coap-base-type;
      description
        "CoAP option Block2.";
      reference
        "RFC 7959 Block-Wise Transfers in the Constrained
                  Application Protocol (CoAP)";
    }

    identity fid-coap-option-block1 {
      base fid-coap-base-type;
      description
        "CoAP option Block1.";
      reference
        "RFC 7959 Block-Wise Transfers in the Constrained
                  Application Protocol (CoAP)";
    }

    identity fid-coap-option-size2 {
```

```
        base fid-coap-base-type;
        description
          "CoAP option size2.";
        reference
          "RFC 7959 Block-Wise Transfers in the Constrained
                    Application Protocol (CoAP)";
      }

      identity fid-coap-option-proxy-uri {
        base fid-coap-base-type;
        description
          "CoAP option Proxy-Uri.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-proxy-scheme {
        base fid-coap-base-type;
        description
          "CoAP option Proxy-scheme.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-size1 {
        base fid-coap-base-type;
        description
          "CoAP option Size1.";
        reference
          "RFC 7252 The Constrained Application Protocol (CoAP)";
      }

      identity fid-coap-option-no-response {
        base fid-coap-base-type;
        description
          "CoAP option No response.";
        reference
          "RFC 7967 Constrained Application Protocol (CoAP)
                    Option for No Server Response";
      }

      identity fid-oscore-base-type {
        base fid-coap-type;
        description
          "OSCORE options (RFC8613) split in sub options.";
        reference
          "RFC 8824 Static Context Header Compression (SCHC) for the
                    Constrained Application Protocol (CoAP)";
```

```
     }

     identity fid-coap-option-oscore-flags {
       base fid-oscore-base-type;
       description
         "CoAP option oscore flags.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP) (see
                   section 6.4)";
     }

     identity fid-coap-option-oscore-piv {
       base fid-oscore-base-type;
       description
         "CoAP option oscore flags.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP) (see
                   section 6.4)";
     }

     identity fid-coap-option-oscore-kid {
       base fid-oscore-base-type;
       description
         "CoAP option oscore flags.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP) (see
                   section 6.4)";
     }

     identity fid-coap-option-oscore-kidctx {
       base fid-oscore-base-type;
       description
         "CoAP option oscore flags.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP)(see
                   section 6.4)";
     }

     //----------------------------------
     // Field Length type definition
     //----------------------------------

     identity fl-base-type {
       description
```

```
         "Used to extend field length functions.";
     }

     identity fl-variable {
       base fl-base-type;
       description
         "Residue length in Byte is sent as defined for CoAP.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP) (see
                   section 5.3)";
     }

     identity fl-token-length {
       base fl-base-type;
       description
         "Residue length in Byte is sent as defined for CoAP.";
       reference
         "RFC 8824 Static Context Header Compression (SCHC) for the
                   Constrained Application Protocol (CoAP) (see
                   section 4.5)";
     }

     //---------------------------------
     // Direction Indicator type
     //---------------------------------

     identity di-base-type {
       description
         "Used to extend direction indicators.";
     }

     identity di-bidirectional {
       base di-base-type;
       description
         "Direction Indication of bidirectionality.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.1.)";
     }

     identity di-up {
       base di-base-type;
       description
         "Direction Indication of uplink.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
```

```
                   Header Compression and Fragmentation (see
                   section 7.1).";
     }

     identity di-down {
       base di-base-type;
       description
         "Direction Indication of downlink.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.1).";
     }

     //----------------------------------
     // Matching Operator type definition
     //----------------------------------

     identity mo-base-type {
       description
         "Matching Operator: used in the rule selection process
          to check is a Target Value matches the field's value.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see*
                   section 7.2).";
     }

     identity mo-equal {
       base mo-base-type;
       description
         "equal MO.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.3).";
     }

     identity mo-ignore {
       base mo-base-type;
       description
         "ignore MO.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.3).";
     }
```

```
     identity mo-msb {
       base mo-base-type;
       description
         "MSB MO.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.3).";
     }

     identity mo-match-mapping {
       base mo-base-type;
       description
         "match-mapping MO.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.3).";
     }

     //------------------------------
     // CDA type definition
     //------------------------------

     identity cda-base-type {
       description
         "Compression Decompression Actions. Specify the action to
          be applied to the field's value in a specific rule.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.2).";
     }

     identity cda-not-sent {
       base cda-base-type;
       description
         "not-sent CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-value-sent {
       base cda-base-type;
       description
         "value-sent CDA.";
```

```
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-lsb {
       base cda-base-type;
       description
         "LSB CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-mapping-sent {
       base cda-base-type;
       description
         "mapping-sent CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-compute {
       base cda-base-type;
       description
         "compute-* CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-deviid {
       base cda-base-type;
       description
         "DevIID CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     identity cda-appiid {
       base cda-base-type;
```

```
       description
         "AppIID CDA.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context
                   Header Compression and Fragmentation (see
                   section 7.4).";
     }

     // -- type definition

     typedef fid-type {
       type identityref {
         base fid-base-type;
       }
       description
         "Field ID generic type.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef fl-type {
       type union {
         type uint64 {
           range 1..max;
         }
         type identityref {
           base fl-base-type;
         }
       }
       description
         "Field length either a positive integer expressing the size in
          bits or a function defined through an identityref.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef di-type {
       type identityref {
         base di-base-type;
       }
       description
         "Direction in LPWAN network, up when emitted by the device,
          down when received by the device, bi when emitted or
          received by the device.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
```

```
                  Compression and Fragmentation";
     }

     typedef mo-type {
       type identityref {
         base mo-base-type;
       }
       description
         "Matching Operator (MO) to compare fields values with
          target values.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef cda-type {
       type identityref {
         base cda-base-type;
       }
       description
         "Compression Decompression Action to compression or
          decompress a field.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     // -- FRAGMENTATION TYPE
     // -- fragmentation modes

     identity fragmentation-mode-base-type {
       description
         "Define the fragmentation mode.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity fragmentation-mode-no-ack {
       base fragmentation-mode-base-type;
       description
         "No-ACK mode.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity fragmentation-mode-ack-always {
```

```
       base fragmentation-mode-base-type;
       description
         "ACK-Always mode.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity fragmentation-mode-ack-on-error {
       base fragmentation-mode-base-type;
       description
         "ACK-on-Error mode.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef fragmentation-mode-type {
       type identityref {
         base fragmentation-mode-base-type;
       }
       description
         "Define the type used for fragmentation mode in rules.";
     }

     // -- Ack behavior

     identity ack-behavior-base-type {
       description
         "Define when to send an Acknowledgment .";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity ack-behavior-after-all-0 {
       base ack-behavior-base-type;
       description
         "Fragmentation expects Ack after sending All-0 fragment.";
     }

     identity ack-behavior-after-all-1 {
       base ack-behavior-base-type;
       description
         "Fragmentation expects Ack after sending All-1 fragment.";
     }

     identity ack-behavior-by-layer2 {
```

```
        base ack-behavior-base-type;
        description
          "Layer 2 defines when to send an Ack.";
      }

      typedef ack-behavior-type {
        type identityref {
          base ack-behavior-base-type;
        }
        description
          "Define the type used for Ack behavior in rules.";
      }

      // -- All-1 with data types

      identity all-1-data-base-type {
        description
          "Type to define when to send an Acknowledgment message.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                     Compression and Fragmentation";
      }

      identity all-1-data-no {
        base all-1-data-base-type;
        description
          "All-1 contains no tiles.";
      }

      identity all-1-data-yes {
        base all-1-data-base-type;
        description
          "All-1 MUST contain a tile.";
      }

      identity all-1-data-sender-choice {
        base all-1-data-base-type;
        description
          "Fragmentation process chooses to send tiles or not in All-1.";
      }

      typedef all-1-data-type {
        type identityref {
          base all-1-data-base-type;
        }
        description
          "Define the type used for All-1 format in rules.";
      }
```

```
     // -- RCS algorithm types

     identity rcs-algorithm-base-type {
       description
         "Identify which algorithm is used to compute RCS.
          The algorithm also defines the size of the RCS field.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     identity rcs-crc32 {
       base rcs-algorithm-base-type;
       description
         "CRC 32 defined as default RCS in RFC8724. This RCS is
          4 bytes long.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     typedef rcs-algorithm-type {
       type identityref {
         base rcs-algorithm-base-type;
       }
       description
         "Define the type for RCS algorithm in rules.";
     }

     // --------  RULE ENTRY DEFINITION ------------

     grouping tv-struct {
       description
         "Defines the target value element. If the header field
          contains a text, the binary sequence uses the same encoding.
          field-id allows the conversion to the appropriate type.";
       leaf index {
         type uint16;
         description
           "Index gives the position in the matching-list. If only one
            element is present, index is 0. Otherwise, index is the
            the order in the matching list, starting at 0.";
       }
       leaf value {
         type binary;
         description
           "Target Value content as an untyped binary value.";
       }
```

```
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     grouping compression-rule-entry {
       description
         "These entries defines a compression entry (i.e. a line)
          as defined in RFC 8724.

      +-------+--+--+--+------------+-----------------+---------------+
      |Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|
      +-------+--+--+--+------------+-----------------+---------------+

          An entry in a compression rule is composed of 7 elements:
          - Field ID: The header field to be compressed.
          - Field Length : Either a positive integer of a function.
          - Field Position: A positive (and possibly equal to 0)
            integer.
          - Direction Indicator: An indication in which direction
            compression and decompression process is effective.
          - Target value: A value against which the header Field is
            compared.
          - Matching Operator: The comparison operation and optional
            associate parameters.
          - Comp./Decomp. Action: The compression or decompression
            action, and optional parameters.
         ";
       leaf field-id {
         type schc:fid-type;
         mandatory true;
         description
           "Field ID, identify a field in the header with a YANG
            identity reference.";
       }
       leaf field-length {
         type schc:fl-type;
         mandatory true;
         description
           "Field Length, expressed in number of bits if the length is
            known when the Rule is created or through a specific
            function if the length is variable.";
       }
       leaf field-position {
         type uint8;
         mandatory true;
         description
           "Field position in the header is an integer. Position 1
```

```
             matches the first occurrence of a field in the header,
             while incremented position values match subsequent
             occurrences.
             Position 0 means that this entry matches a field
             irrespective of its position of occurrence in the
             header.
             Be aware that the decompressed header may have
             position-0 fields ordered differently than they
             appeared in the original packet.";
        }
        leaf direction-indicator {
          type schc:di-type;
          mandatory true;
          description
            "Direction Indicator, indicate if this field must be
             considered for rule selection or ignored based on the
             direction (bi directionnal, only uplink, or only
             downlink).";
        }
        list target-value {
          key "index";
          uses tv-struct;
          description
            "A list of value to compare with the header field value.
             If target value is a singleton, position must be 0.
             For use as a matching list for the mo-match-mapping matching
             operator, index should take consecutive values starting
             from 0.";
        }
        leaf matching-operator {
          type schc:mo-type;
          must "../target-value or derived-from-or-self(.,
                                                      'mo-ignore')" {
            error-message
              "mo-equal, mo-msb and mo-match-mapping need target-value";
            description
              "target-value is not required for mo-ignore.";
          }
          must "not (derived-from-or-self(., 'mo-msb')) or
                ../matching-operator-value" {
            error-message "mo-msb requires length value";
          }
          mandatory true;
          description
            "MO: Matching Operator.";
          reference
            "RFC 8724 SCHC: Generic Framework for Static Context Header
                      Compression and Fragmentation (see Section 7.3).";
```

```
       }
       list matching-operator-value {
         key "index";
         uses tv-struct;
         description
           "Matching Operator Arguments, based on TV structure to allow
            several arguments.
            In RFC 8724, only the MSB matching operator needs arguments
            (a single argument, which is the number of most significant
            bits to be matched).";
       }
       leaf comp-decomp-action {
         type schc:cda-type;
         must "../target-value or
                    derived-from-or-self(., 'cda-value-sent') or
                    derived-from-or-self(., 'cda-compute') or
                    derived-from-or-self(., 'cda-appiid') or
                    derived-from-or-self(., 'cda-deviid')" {
           error-message
             "cda-not-sent, cda-lsb, cda-mapping-sent need
              target-value";
           description
             "target-value is not required for some CDA.";
           }
         mandatory true;
         description
           "CDA: Compression Decompression Action.";
         reference
           "RFC 8724 SCHC: Generic Framework for Static Context Header
                     Compression and Fragmentation (see section 7.4)";
       }
       list comp-decomp-action-value {
         key "index";
         uses tv-struct;
         description
           "CDA arguments, based on a TV structure, in order to allow
            for several arguments. The CDAs specified in RFC 8724
            require no argument.";
       }

     }

     // --Rule nature

     identity nature-base-type {
       description
         "A rule, identified by its RuleID, are used for a single
          purpose. RFC 8724 defines 2 natures:
```

```
           compression, no compression and fragmentation.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-compression {
        base nature-base-type;
        description
          "Identify a compression rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-no-compression {
        base nature-base-type;
        description
          "Identify a no compression rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-fragmentation {
        base nature-base-type;
        description
          "Identify a fragmentation rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      typedef nature-type {
        type identityref {
          base nature-base-type;
        }
        description
          "defines the type to indicate the nature of the rule.";
      }

      grouping compression-content {
        list entry {
          must "derived-from-or-self(../rule-nature,
                                      'nature-compression')" {
            error-message "Rule nature must be compression";
          }
          key "field-id field-position direction-indicator";
```

```
         uses compression-rule-entry;
         description
           "A compression rule is a list of rule entries, each
            describing a header field. An entry is identified
            through a field-id, its position in the packet, and
            its direction.";
       }
       description
         "Define a compression rule composed of a list of entries.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     grouping fragmentation-content {
       description
         "This grouping defines the fragmentation parameters for
          all the modes (No-ACK, ACK-Always and ACK-on-Error) specified
          in RFC 8724.";
       leaf fragmentation-mode {
         type schc:fragmentation-mode-type;
         must "derived-from-or-self(../rule-nature,
                                    'nature-fragmentation')" {
           error-message "Rule nature must be fragmentation";
         }
         mandatory true;
         description
           "Which fragmentation mode is used (No-Ack, ACK-Always,
            ACK-on-Error).";
       }
       leaf l2-word-size {
         type uint8;
         default "8";
         description
           "Size, in bits, of the layer 2 word.";
       }
       leaf direction {
         type schc:di-type;
         must "derived-from-or-self(., 'di-up') or
               derived-from-or-self(., 'di-down')" {
           error-message
             "Direction for fragmentation rules are up or down.";
         }
         mandatory true;
         description
           "MUST be up or down, bidirectional MUST NOT be used.";
       }
       // SCHC Frag header format
```

```
       leaf dtag-size {
         type uint8;
         default "0";
         description
           "Size, in bits, of the DTag field (T variable from
            RFC8724).";
       }
       leaf w-size {
         when "derived-from-or-self(../fragmentation-mode,
                                 'fragmentation-mode-ack-on-error')
               or
               derived-from-or-self(../fragmentation-mode,
                                 'fragmentation-mode-ack-always') ";
         type uint8;
         description
           "Size, in bits, of the window field (M variable from
            RFC8724).";
       }
       leaf fcn-size {
         type uint8;
         mandatory true;
         description
           "Size, in bits, of the FCN field (N variable from RFC8724).";
       }
       leaf rcs-algorithm {
         type rcs-algorithm-type;
         default "schc:rcs-crc32";
         description
           "Algorithm used for RCS. The algorithm specifies the RCS
            size.";
       }
       // SCHC fragmentation protocol parameters
       leaf maximum-packet-size {
         type uint16;
         default "1280";
         description
           "When decompression is done, packet size must not
            strictly exceed this limit, expressed in bytes.";
       }
       leaf window-size {
         type uint16;
         description
           "By default, if not specified 2^w-size - 1. Should not exceed
            this value. Possible FCN values are between 0 and
            window-size - 1.";
       }
       leaf max-interleaved-frames {
         type uint8;
```

```
          default "1";
          description
            "Maximum of simultaneously fragmented frames. Maximum value
             is 2^dtag-size. All DTAG values can be used, but more than
             max-interleaved-frames MUST NOT be active at any time";
        }
        container inactivity-timer {
          leaf ticks-duration {
            type uint8;
            default "20";
            description
              "Duration of one tick in micro-seconds:
                  2^ticks-duration/10^6 = 1.048s.";
          }
          leaf ticks-numbers {
            type uint16 {
              range "0..max";
            }
            description
              "Timer duration = ticks-numbers*2^ticks-duration / 10^6.";
          }

          description
            "Duration is seconds of the inactivity timer, 0 indicates
             that the timer is disabled.

             Allows a precision from micro-second to year by sending the
             tick-duration value. For instance:

           tick-duration /  smallest value          highest value
           v
           20: 00y 000d 00h 00m 01s.048575<->00y 000d 19h 05m 18s.428159
           21: 00y 000d 00h 00m 02s.097151<->00y 001d 14h 10m 36s.856319
           22: 00y 000d 00h 00m 04s.194303<->00y 003d 04h 21m 13s.712639
           23: 00y 000d 00h 00m 08s.388607<->00y 006d 08h 42m 27s.425279
           24: 00y 000d 00h 00m 16s.777215<->00y 012d 17h 24m 54s.850559
           25: 00y 000d 00h 00m 33s.554431<->00y 025d 10h 49m 49s.701119

            Note that the smallest value is also the incrementation step,
            so the timer precision.";
        }
        container retransmission-timer {
          leaf ticks-duration {
            type uint8;
            default "20";
            description
              "Duration of one tick in micro-seconds:
                  2^ticks-duration/10^6 = 1.048s.";
```

```
         }
         leaf ticks-numbers {
           type uint16 {
             range "1..max";
           }
           description
             "Timer duration = ticks-numbers*2^ticks-duration / 10^6.";
         }

         when "derived-from-or-self(../fragmentation-mode,
                                    'fragmentation-mode-ack-on-error')
               or
               derived-from-or-self(../fragmentation-mode,
                                    'fragmentation-mode-ack-always') ";
         description
           "Duration in seconds of the retransmission timer.
            See inactivity timer.";
       }
       leaf max-ack-requests {
         when "derived-from-or-self(../fragmentation-mode,
                                    'fragmentation-mode-ack-on-error')
               or
               derived-from-or-self(../fragmentation-mode,
                                    'fragmentation-mode-ack-always') ";
         type uint8 {
           range "1..max";
         }
         description
           "The maximum number of retries for a specific SCHC ACK.";
       }
       choice mode {
         case no-ack;
         case ack-always;
         case ack-on-error {
           leaf tile-size {
             when "derived-from-or-self(../fragmentation-mode,
                                  'fragmentation-mode-ack-on-error')";
             type uint8;
             description
               "Size, in bits, of tiles. If not specified or set to 0,
                tiles fill the fragment.";
           }
           leaf tile-in-all-1 {
             when "derived-from-or-self(../fragmentation-mode,
                                  'fragmentation-mode-ack-on-error')";
             type schc:all-1-data-type;
             description
               "Defines whether the sender and receiver expect a tile in
```

```
                All-1 fragments or not, or if it is left to the sender's
                choice.";
           }
           leaf ack-behavior {
             when "derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-on-error')";
             type schc:ack-behavior-type;
             description
               "Sender behavior to acknowledge, after All-0, All-1 or
                when the LPWAN allows it.";
           }
         }
         description
           "RFC 8724 defines 3 fragmentation modes.";
       }
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     // Define rule ID. Rule ID is composed of a RuleID value and a
     // Rule ID Length

     grouping rule-id-type {
       leaf rule-id-value {
         type uint32;
         description
           "Rule ID value, this value must be unique, considering its
            length.";
       }
       leaf rule-id-length {
         type uint8 {
           range "0..32";
         }
         description
           "Rule ID length, in bits. The value 0 is for implicit
            rules.";
       }
       description
         "A rule ID is composed of a value and a length, expressed in
          bits.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

     // SCHC table for a specific device.
```

```
     container schc {
       list rule {
         key "rule-id-value rule-id-length";
         uses rule-id-type;
         leaf rule-nature {
           type nature-type;
           mandatory true;
           description
             "Specify the rule’s nature.";
         }
         choice nature {
           case fragmentation {
             if-feature "fragmentation";
             uses fragmentation-content;
           }
           case compression {
             if-feature "compression";
             uses compression-content;
           }
           description
             "A rule is for compression, for no-compression or for
              fragmentation.";
         }
         description
           "Set of rules compression, no compression or fragmentation
            rules identified by their rule-id.";
       }
       description
         "A SCHC set of rules is composed of a list of rules which are
          used for compression, no-compression or fragmentation.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }
   }
   <CODE ENDS>
```

Figure 6

7. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

* Openschc is implementing the conversion between the local rule representation and the representation conforming to the data model in JSON and CBOR (following -08 draft).

8. IANA Considerations

This document registers one URI and one YANG modules.

8.1. URI Registration

This document requests IANA to register the following URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-schc

Registrant Contact: The IESG.

XML: N/A; the requested URI is an XML namespace.

8.2. YANG Module Name Registration

This document registers the following one YANG modules in the "YANG Module Names" registry [RFC6020].

name: ietf-schc

namespace: urn:ietf:params:xml:ns:yang:ietf-schc

prefix: schc

reference: RFC XXXX Data Model for Static Context Header Compression (SCHC)

9. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

This data model formalizes the rules elements described in [RFC8724] for compression, and fragmentation. As explained in the architecture document [I-D.ietf-lpwan-architecture], a rule can be read, created, updated or deleted in response to a management request. These actions can be done between two instances of SCHC or between a SCHC instance and a rule repository.

```
                        create
             (-------)  read   +=======+ *
             ( rules )<------->|Rule   |<--|-------->
             (-------)  update |Manager|   NETCONF, RESTCONF,...
                . read  delete +=======+   request
                .
             +-------+
         <===| R & D |<===
         ===>| C & F |===>
             +-------+
```

The rule contains sensitive information such as the application IPv6 address where the device's data will be sent after decompression. A device may try to modify other devices' rules by changing the application address and may block communication or allows traffic eavesdropping. Therefore, a device must be allowed to modify only its own rules on the remote SCHC instance. The identity of the requester must be validated. This can be done through certificates or access lists. By reading a module, an attacker may know the traffic a device can generate and learn about application addresses or REST API.

The full tree is sensitive, since it represents all the elements that can be managed. This module aims to be encapsulated into a YANG module including access controls and identities.

10. Annex A : Example

The informal rules given Figure 7 will represented in XML as shown in Figure 8.

```
/-------------------------\
|Rule 6/3           110   |
|---------------+---+--+--+----------------+-------+----------------\
|IPV6.VER       |  4| 1|BI|               6|EQUAL  |NOT-SENT        |
|IPV6.TC        |  8| 1|BI|               0|EQUAL  |NOT-SENT        |
|IPV6.FL        | 20| 1|BI|               0|IGNORE |NOT-SENT        |
|IPV6.LEN       | 16| 1|BI|                |IGNORE |COMPUTE-LENGTH  |
|IPV6.NXT       |  8| 1|BI|              58|EQUAL  |NOT-SENT        |
|IPV6.HOP_LMT   |  8| 1|BI|             255|IGNORE |NOT-SENT        |
|IPV6.DEV_PREFIX| 64| 1|BI|200104701f2101d2|EQUAL  |NOT-SENT        |
|IPV6.DEV_IID   | 64| 1|BI|0000000000000003|EQUAL  |NOT-SENT        |
|IPV6.APP_PREFIX| 64| 1|BI|                |IGNORE |VALUE-SENT      |
|IPV6.APP_IID   | 64| 1|BI|                |IGNORE |VALUE-SENT      |
\---------------+---+--+--+----------------+-------+----------------/
/-------------------------\
|Rule 12/11     00001100  |
!=========================+=========================================\
!^ Fragmentation mode : NoAck   header dtag 2 Window  0 FCN  3  UP ^!
!^ No Tile size specified                                          ^!
!^ RCS Algorithm: RCS_CRC32                                        ^!
\===================================================================/
/-------------------------\
|Rule 100/8     01100100  |
| NO COMPRESSION RULE     |
\-------------------------/
```

Figure 7: Rules example

```
<?xml version='1.0' encoding='UTF-8'?>
  <schc xmlns="urn:ietf:params:xml:ns:yang:ietf-schc">
  <rule>
    <rule-id-value>6</rule-id-value>
    <rule-id-length>3</rule-id-length>
    <rule-nature>nature-compression</rule-nature>
    <entry>
      <field-id>fid-ipv6-version</field-id>
      <field-length>4</field-length>
      <field-position>1</field-position>
      <direction-indicator>di-bidirectional</direction-indicator>
      <matching-operator>mo-equal</matching-operator>
      <comp-decomp-action>cda-not-sent</comp-decomp-action>
      <target-value>
        <index>0</index>
```

```
          <value>AAY=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-trafficclass</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AA==</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-flowlabel</field-id>
        <field-length>20</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AA==</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-payload-length</field-id>
        <field-length>16</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-compute</comp-decomp-action>
      </entry>
      <entry>
        <field-id>fid-ipv6-nextheader</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>ADo=</value>
        </target-value>
      </entry>
      <entry>
```

```
        <field-id>fid-ipv6-hoplimit</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AP8=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-devprefix</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>IAEEcB8hAdI=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-deviid</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AAAAAAAAAAM=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-appprefix</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-value-sent</comp-decomp-action>
      </entry>
      <entry>
        <field-id>fid-ipv6-appiid</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
```

```
        &lt;matching-operator&gt;mo-ignore&lt;/matching-operator&gt;
        &lt;comp-decomp-action&gt;cda-value-sent&lt;/comp-decomp-action&gt;
      &lt;/entry&gt;
    &lt;/rule&gt;
    &lt;rule&gt;
      &lt;rule-id-value&gt;12&lt;/rule-id-value&gt;
      &lt;rule-id-length&gt;11&lt;/rule-id-length&gt;
      &lt;rule-nature&gt;nature-fragmentation&lt;/rule-nature&gt;
      &lt;direction&gt;di-up&lt;/direction&gt;
      &lt;rcs-algorithm&gt;rcs-crc32&lt;/rcs-algorithm&gt;
      &lt;dtag-size&gt;2&lt;/dtag-size&gt;
      &lt;fcn-size&gt;3&lt;/fcn-size&gt;
      &lt;fragmentation-mode&gt;fragmentation-mode-no-ack&lt;/fragmentation-mode&gt;
    &lt;/rule&gt;
    &lt;rule&gt;
      &lt;rule-id-value&gt;100&lt;/rule-id-value&gt;
      &lt;rule-id-length&gt;8&lt;/rule-id-length&gt;
      &lt;rule-nature&gt;nature-no-compression&lt;/rule-nature&gt;
    &lt;/rule&gt;
   &lt;/schc&gt;
```

Figure 8: XML representation of the rules.

11. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Ivan Martinez, Alexander Pelov for their careful reading and valuable inputs. A special thanks for Joe Clarke, Carl Moberg, Tom Petch, Martin Thomson, and Eric Vyncke for their explanations and wise advices when building the model.

12. References

12.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[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>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

[RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", RFC 8724, DOI 10.17487/RFC8724, April 2020, <https://www.rfc-editor.org/info/rfc8724>.

[RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", RFC 8824, DOI 10.17487/RFC8824, June 2021, <https://www.rfc-editor.org/info/rfc8824>.

12.2. Informative References

[I-D.ietf-lpwan-architecture] Pelov, A., Thubert, P., and A. Minaburo, "LPWAN Static Context Header Compression (SCHC) Architecture", Work in Progress, Internet-Draft, draft-ietf-lpwan-architecture-02, 30 June 2022, <https://www.ietf.org/archive/id/draft-ietf-lpwan-architecture-02.txt>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

[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>.

[RFC9011] Gimenez, O., Ed. and I. Petrov, Ed., "Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN", RFC 9011, DOI 10.17487/RFC9011, April 2021, <https://www.rfc-editor.org/info/rfc9011>.

Authors' Addresses

Ana Minaburo
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: ana@ackl.io

Laurent Toutain
Institut MINES TELECOM; IMT Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France
Email: Laurent.Toutain@imt-atlantique.fr