Transmission of SCHC-compressed packets over IEEE 802.15.4 networks
draft-gomez-6lo-schc-15dot4-02

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

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

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

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

This Internet-Draft will expire on 2 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
license-info) in effect on the date of publication of this document.
Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction .................................................. 2
2. Terminology ................................................... 3
   2.1. Requirements language .................................. 4
   2.2. Background on SCHC ...................................... 4
3. Architecture .................................................... 4
   3.1. Network topologies ...................................... 4
   3.2. Protocol stack .......................................... 4
4. Frame Format .................................................... 5
   4.1. SCHC Dispatch ........................................... 6
   4.2. SCHC Header .............................................. 6
   4.3. Padding .................................................. 6
5. SCHC compression for IPv6, UDP, and CoAP headers ................. 6
   5.1. SCHC compression for IPv6 and UDP headers ............. 6
      5.1.1. Compression of IPv6 addresses ...................... 7
   5.2. SCHC compression for CoAP headers ...................... 7
   5.3. Header compression examples ............................ 8
6. Fragmentation and reassembly .................................... 8
7. IANA Considerations ............................................ 8
8. Security Considerations ......................................... 8
9. Acknowledgments ............................................... 8
10. References .................................................... 8
   10.1. Normative References ................................. 8
   10.2. Informative References ............................... 10
Authors’ Addresses ................................................. 10

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 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 RFC 8724 for use over IEEE 802.15.4.

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 SCHC

The reader is expected to be familiar with the terms and concepts defined in the specification of SCHC (RFC 8724).

3. Architecture

3.1. 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.

3.2. 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).
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.

4. Frame Format

This document 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. The 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.

Figure 2: Encapsulated, SCHC-compressed packet. Padding bits are added if needed.
4.1. 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)

4.2. 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.

4.3. 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.

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.

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 (TBD) 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) for each endpoint it communicates with. 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.

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 CDA s 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.

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) for each endpoint it communicates with. Therefore, in such cases, direction information will be specific for each pair of endpoints.
5.3. Header compression examples

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

6. Fragmentation and reassembly

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

7. IANA Considerations

This document requests the allocation of the Dispatch Type Field bit pattern 01000100 (Page 0) 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, and Guangpeng Li 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’Universitat i Recerca del Departament d’Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

10. References

10.1. Normative References


10.2. Informative References

[I-D.toutain-6lo-6lo-and-schc]


Authors’ Addresses

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

Ana Minaburo
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: ana@acklio.io
SCHC Compound ACK
draft-ietf-lpwan-schc-compound-ack-04

Abstract

The present document describes an extension to the SCHC (Static Context Header Compression and fragmentation) protocol [RFC8724]. 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 LWPAN technologies defined in [RFC8376], being Sigfox, LoRaWAN, NB-IoT and IEEE 802.15.4w.

Status of This Memo

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

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

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

This Internet-Draft will expire on 22 September 2022.
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 LWPAN defined 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.

The present document describes an extension to 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 SCHC ACK message format so that it can contain several bitmaps, each bitmap being identified by its corresponding window number.

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,
* has the same SCHC ACK format defined in [RFC8724] when only one window with losses is reported,
* might not cover all windows with fragment losses of a SCHC Packet,
* and is distinguishable from the SCHC Receiver-Abort.

2. Terminology

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 SCHC ACK message that can contain several bitmaps, each bitmap being identified by its corresponding window number.

The SCHC Compound ACK groups the window number (W) with its corresponding bitmap. Windows do not need to be contiguous. However, the window numbers and corresponding bitmaps included in the SCHC Compound ACK message MUST be ordered from the lowest-numbered to the highest-numbered window. Hence, if the 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-between the Rule ID and the C bit. This also avoids confusing any '0s' padding bits following the first bitmap with W number zero.
3.1. SCHC Compound ACK Message Format

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

| SCHC ACK Header -- | RuleID | W | C=b'1 | b'0-pad(opt) |
+-------------------+--------+-----+--------+--------------+
| RuleID            | W      | C   | Bitmap | b'0-pad(opt) |
+--------+--------+-----+--------+--------------+

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.

|--SCHC ACK Header--| W = w1 |...|  W = wi  |
+-------------------+ ------ +...+ ------- + ------ +------------+
|RuleID|W=b'w1|C=b’0| Bitmap |...|  W=b’wi | Bitmap |b’0-pad(opt) |
+------+------+-----+ ------ +...+ ------- + ------ +------------+

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

Figure 2: SCHC Compound ACK message format

The SCHC Compound ACK MUST NOT use the Compressed Bitmap format for intermediate windows/bitmaps (i.e., bitmaps that are not the last one), 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. The optional usage of this Compressed Bitmap for the last bitmap MUST be specified by the SCHC technology-specific profile.

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.

Each different SCHC LPWAN technology profile MUST specify how the SCHC Compound ACK is different from the Receiver-Abort message as per [RFC8724], e.g., the Receiver-Abort message is padded with 1s with an extra byte appended at the end, while the SCHC Compound ACK is 0-padded.
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.

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. Note that even though the protocol allows for such flexibility, the actual decision criteria is not specified in this document.

The following sections describe the differences between the baseline SCHC specification and the present SCHC protocol extension specification.

3.2.1. Sender Behaviour

OLD TEXT ([RFC8724], section 8.4.3.1) - On receiving a SCHC ACK:

* (...)  
* the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.  
* if the last of these SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MUST in addition send after it a SCHC ACK REQ with the W field corresponding to the last window.

NEW TEXT - On receiving a SCHC Compound ACK:

* (...)  
* the fragment sender MUST resend 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 reported missing 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, continue the transmission of the remaining fragments to be transmitted, or repeat the All-1 fragment to confirm that all fragments have been correctly received.
3.2.2. Receiver Behaviour

OLD TEXT ([RFC8724], section 8.4.3.2) - On receiving a SCHC ACK REQ or an All-1 SCHC Fragment:

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

NEW TEXT: On receiving an All-0 SCHC Fragment:

* if the receiver knows of any windows with missing tiles for the packet being reassembled (and if network conditions are known to be conducive), it MAY return a SCHC Compound ACK for the missing fragments, starting from the lowest-numbered window.

NEW TEXT: 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 fragments, starting from the lowest-numbered window.

3.3. SCHC Compound ACK Examples

Figure 3 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.
3.4. SCHC Compound ACK YANG Data Model

The present document also extends the SCHC YANG data model defined in [I-D.ietf-lpwan-schc-yang-data-model] 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.

3.4.1. SCHC YANG Data Model Extension
This module extends the ietf-schc module to include the Compound ACK behavior for ACK-on-Error as defined in RFC YYYY. It introduces a new leaf for ACK-on-Error defining the format of the SCHC Compound ACK, adding the possibility to send several bitmaps in a single SCHC ACK message.
"Initial version for RFC YYYY ";
reference
"RFC YYY: 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.";
}

identity bitmap-compound-ack {
  base bitmap-format-base-type;
  description
  "Compound ACK.";
}

typedef bitmap-format-type {
  type identityref {
    base bitmap-format-base-type;
  }
  description
  "type used in rules";
}

augment "/schc:schc/schc:rule/schc:nature/schc:fragmentation/schc:mode/schc:
ack-on-error" {
  leaf bitmap-format {
    when "derived-from(../schc:fragmentation-mode, 'schc:fragmentation-mod
e-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(../schc:fragmentation-mode, 'schc:fragmentation-mod
e-ack-on-error')";
    type boolean;
    default true;
    description
    "when true ultimate bitmap in the SCHC ACK message can
be compressed";
  }
}

description
"added to SCHC rules";
}

Figure 5: SCHC YANG Data Model - Compound ACK extension

3.4.2. SCHC YANG Tree Extension

   +=-rw bitmap-format?   schc-compound-ack:bitmap-format-type

Figure 6: SCHC YANG Tree - Compound ACK extension

4. SCHC Compound ACK Parameters

This section lists the parameters related to the SCHC Compound ACK usage that need to be defined in the Profile, in addition to the ones listed 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.

* Differentiation between SCHC Receiver-Abort and SCHC Compound ACK message, e.g., Receiver-Abort message padded with 1s with an extra byte appended at the end, while the SCHC Compound ACK is 0-padded.

5. Security considerations

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

6. 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.

7. Normative References

[I-D.ietf-lpwan-schc-yang-data-model]


Authors’ Addresses

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

Carles Gomez
Universitat Politècnica de Catalunya
C/Estève Terradas, 7
08860 Castelldefels
Spain
Email: carlesgo@entel.upc.edu
SCHC over NB-IoT
draft-ietf-lpwan-schc-over-nbiot-08

Abstract

The Static Context Header Compression and Fragmentation (SCHC) specification describes header compression and fragmentation functionalities for LPWAN (Low Power Wide Area Networks) technologies. The Narrowband Internet of Things (NB-IoT) architecture may adapt SCHC to improve its capacities.

This document describes the use of SCHC over the NB-IoT wireless access and provides use-cases for efficient parameterization.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 20, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components
1. Introduction

The Static Context Header Compression (SCHC) [RFC8724] defines a header compression scheme, and fragmentation functionality, suitable for the Low Power Wide Area Networks (LPWAN) networks described in [RFC8376].

In a Narrowband Internet of Things (NB-IoT) network, header compression efficiently brings Internet connectivity to the Device - User Equipment (Dev-UE). This document describes the SCHC parameters used to support the static context header compression and fragmentation over the NB-IoT wireless access. This document assumes functionality for NB-IoT of 3GPP release 15 (3GPR15). Otherwise, the text explicitly mentions other versions’ functionality.
2. Terminology

This document will follow the terms defined in [RFC8724], in [RFC8376], and the [TGPP23720].

* CIoT. Cellular IoT.
* NGW-C-SGN. Network Gateway - CIoT Serving Gateway Node.
* Dev-UE. Device - User Equipment.
* RGW-eNB. Radio Gateway - Node B. Base Station that controls the UE.
* EPC. Evolved Packet Connectivity. Core network of 3GPP LTE systems.
* NGW-MME. Network Gateway - Mobility Management Entity. An entity in charge of handling mobility of the Dev-UE.
* 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-SGW. Network Gateway - Serving Gateway. It routes and forwards the user data packets through the access network.
* HSS. Home Subscriber Server. It is a database that performs mobility management.
* NGW-PGW. Network Gateway - Packet Data Node Gateway. An interface between the internal with the external network.
* PDU. Protocol Data Unit. A data packet including headers that are transmitted between entities through a protocol.
* SDU. Service Data Unit. A data packet (PDU) from higher layer protocols used by lower layer protocols as a payload of their own PDUs.
* 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.
3. Architecture

The Narrowband Internet of Things (NB-IoT) architecture has a complex structure. It relies on different NGWs from different providers and can send data via different paths, each path with different characteristics in terms of bandwidths, acknowledgments, and layer two 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 Node Gateway (NGW-PGW) may get a limited bandwidth transmission from 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. OMA and OneM2M define the northbound APIS [TGPP33203]. In this case, the path is small for data transmission. The main functions of the NGW-SCEF are:

- Connectivity path and Device Monitoring.
4. Data Transmission in the 3GPP Architecture

NB-IoT networks deal with end-to-end user data and in-band signalling between the nodes and functions to configure, control, and monitor the system functions and behaviors. The signalling data 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 maximum 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. However, the recommended 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, which implies that standard specifying SCHC support would not be backward compatible. A terminal or a network supporting a version of the standard without SCHC or without an implementation capability (in case of not being standardized as mandatory capability) cannot utilize SCHC with this approach.

SCHC could be deployed differently depending on where the header compression and the fragmentation are applied. The SCHC functionalities can be used over the radio transmission only, between the Dev-UE and the RGW-eNB. Alternatively, the packets transmitted over the path can use SCHC. Else, when the transmissions go over the NGW-MME or NGW-SCEF, the NGW-CSGN uses SCHC entity. For these two cases, the functions need to be standardized by 3GPP.

Another possibility is to apply SCHC functionalities to the end-to-end connection or at least up to the operator network edge. 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. The radio network transmits the packets as non-IP traffic using IP tunnelling or SCEF services. Since this option does not necessarily require 3GPP standardization, it is possible to also benefit legacy devices with SCHC by using the non-IP transmission features of the operator network.
4.1. Use of SCHC over the Radio link

Deploying SCHC only over the radio link 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 Modulation and Coding Schemes (MCS) selected. The transmissions of Transport Block (TB) 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 [TGPP36321] defines the Transport Blocks characteristics. The Radio link stack shown in Figure 2 comprises the Packet Data Convergence Protocol (PDCP) [TGPP36323], Radio Link Protocol (RLC) [TGPP36322], Medium Access Control protocol (MAC) [TGPP36321], and the Physical Layer [TGPP36201]. The Appendix gives more details of these protocols.

4.1.1. SCHC Entities Placing

The current architecture provides support for header compression in the PDCP layer using RoHC [RFC5795]. Therefore SCHC header compression entities can be deployed similarly without the need for significant changes in the 3GPP specifications.

In this scenario, the RLC layer takes care of fragmentation unless for the Transparent Mode. When packets exceed the Transport Block size at transmission, SCHC fragmentation is unnecessary and should not be used to avoid the additional protocol overhead. The RLC Transparent Mode is not commonly used while sending IP packets in the Radio link. However, given the case in the future, SCHC fragmentation may be used. In that case, a SCHC tile would match the minimum transport block size minus the PDCP and MAC headers.
4.2. Use of SCHC over the No-Access Stratum (NAS)

The NGW-MME conveys mainly control signaling between the Dev-UE and the cellular network [TGPP24301]. 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 No-Access Stratum (DoNAS) or Control Plane CIoT EPS optimization. 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 gives additional details of DoNAS.

4.2.1. SCHC Entities Placing

In this scenario, SCHC may reside in the Non-Access Stratum (NAS) protocol layer. The same principles as for Radio link transmissions apply here as well. Because the NAS protocol already uses RoHC it can adapt SCHC for header compression too. The main difference compared to the radio link 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.

![SCHC entities placement diagram]

*PDCP is bypassed until AS security is activated TGPP36300.

Figure 3: SCHC entities placement in the 3GPP CIOT radio protocol architecture for DoNAS transmissions
4.2.2. Parameters for Static Context Header Compression and Fragmentation (SCHC) for the Section 4.1 and Section 4.2.

These scenarios MUST use SCHC header compression capability to improve the transmission of IPv6 packets. The 3GPP Architecture currently provides Header Compression using the [RFC5795] but the use of SCHC for IoT application MUST be considered to improve the devices connectivity.

* SCHC Context initialization 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 operation [TGPP36323].

* SCHC Rules The network operator in these scenarios defines the number of rules in a context. The operator must be aware of the type of IP traffic that the device will carry out. Implying that the operator might use provision sets of rules compatible with the use case of the device. For devices acting as gateways of other devices, several rules may match the diversity of devices and protocols used by the devices associated with the gateway. Meanwhile, simpler devices (for example, an electricity meter) may have a predetermined set of fixed protocols and parameters. Additionally, the deployment of IPv6 addresses may force different 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. A 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. The packets handled by 3GPP networks are byte-aligned, and therefore the minimum payload possible (including padding) is 8 bits. Therefore in order to use the smallest TB, the maximum SCHC header size 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 capillarity gateway) might require additional bits to handle the variety and multiple parameters of higher-layer protocols deployed. In that sense, the operator may want to have flexibility on the number and type of rules supported by each device independently, and consequently, these scenarios require a configurable value. The configuration may be part of the operation profile agreed together 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 together with the operators and seems quite a reasonable maximum limit even for a device acting as a NAT. More bits could be configured, 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 Section 4.1 and Section 4.2 scenarios, the SCHC fragmentation functions are disabled. The RLC layer of NB-IoT can segment packets in suitable units that fit the selected transport blocks for transmissions of the physical layer. The blocks 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 than allowing the Radio link protocols to take care of the fragmentation natively.

* Fragmentation in RLC Transparent Mode If RLC operates in Transparent Mode, there could be a case to activate a fragmentation function together with a light reliability function such as the ACK-Always mode. In practice, it is uncommon to transmit radio link data using this configuration. It mainly targets signaling transmissions. In those cases, the MAC layer mechanisms ensure reliability, such as repetitions or automatic retransmissions, and additional reliability might only generate protocol overhead.

SCHC may reduce radio network protocols overhead in future operations, support reliable transmissions, and 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 Section 4.3.2.
4.3. End-to-End Compression

The Non-IP Data Delivery (NIDD) services of 3GPP enable the transmission of SCHC packets compressed by the application layer. The packets can be delivered using IP-tunnels to the 3GPP network or NGW-SCEF functions (i.e., 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 under the L2.

4.3.1. SCHC Entities Placing

In the two scenarios using End-to-End compression, SCHC entities are located almost on top of the stack. The NB-IoT connectivity services implement SCHC in the Dev, 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.

```
+---------+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX+--------+
| SCHC    | XXX          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        |   +--------+
+---------+       XXXXXXXXXXXXXXXXXXXXXXXXXXX       +---+  PHY   |
| PHY     +-----+ XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  +---+  PHY |
+---------+                                            +--------+
          |                                             AS |
          UE
```

Figure 4: SCHC entities placed when using Non-IP Delivery (NIDD) 3GPP Services
4.3.2. Parameters for Static Context Header Compression and Fragmentation (SCHC)

These scenarios may use SCHC header compression capability to improve the transmission of IPv6 packets. The use of SCHC for IoT application MUST be considered to improve the devices connectivity.

* 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. Even when the transmission content is not visible for the 3GPP network, the same limitations as for Section 4.1 and Section 4.2 transmissions apply in these scenarios in terms of aiming to use the minimum number of transmissions and minimize the protocol overhead.

* Rule ID Similar to the case of Section 4.1 and Section 4.2, the RuleID size can be dynamically set before the context delivery. For example, negotiated between the applications when choosing a profile according to the type of traffic and application deployed. The same considerations related to the transport block size and performance mentioned for the Section 4.1 and Section 4.2 must be followed when choosing a size value for the RuleID field.

* SCHC MAX_PACKET_SIZE In these scenarios, the maximum recommended MTU size that applies 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 depending on the billing. Packets with very low QoS may get lost before they arrive in the 3GPP radio network transmission, for example, in between the links of a capillarity 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 additional 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 is even desirable to keep track of all
the SCHC packets delivered. In that case, the fragmentation
function could be active for all packets transmitted by the
applications. SCHC ACK-on-Error fragmentation may be active 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 4.2.2 defines
the RuleID size.

SCHC parametrization considers that NB-IoT 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 of 2 bits. The transfer block size has a
wide range of values. Two configurations may be used for the
fragmentation parameters.

* For Transfer Blocks smaller or equal to 300 bits using a 8 bits-
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 300 bits using a 16 bits-
Header_size configuration, with the size of the header fields as
follows:
  - RulesID from 1 to 8 or 10 bits,
  - DTag 1 or 2 bits,
  - FCN 3 bits,
  - W 2 or 3 bits.
The IoT devices communicate with small data transfer and have a battery life of 10 years. These devices use the Power Save Mode and the Idle Mode DRX, which govern how often the device wakes up, stays up, and is reachable. Table 10.5.163a in {3GPP-TS_24.088} 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. To adapt SCHC to the NB-IoT activities, the Inactivity Timer and the Retransmission Timer be set based on these limits.

5. Padding

NB-IoT and 3GPP wireless access, in general, assumes byte-aligned payload. Therefore the L2 word for NB-IoT MUST be considered 8 bits, and the padding treatment should use this value accordingly.

6. Security considerations

This document does not add any security considerations and follows the [RFC8724] and the 3GPP access security document specified in [TGPP33203].

7. Appendix

7.1. NB-IoT User Plane protocol architecture

7.1.1. Packet Data Convergence Protocol (PDCP)

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 (Robust Header Compression)
* 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
7.1.2. Radio Link Protocol (RLC)

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 octets 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.
7.1.3. Medium Access Control (MAC)

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 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.

![Diagram of User Plane packet encapsulation for two transport blocks]

Figure 5: Example of User Plane packet encapsulation for two transport blocks
7.2. NB-IoT Data over NAS (DoNAS)

The Access Stratum (AS) protocol stack used by DoNAS is somehow particular. Since the security associations are not established yet in the radio network, to reduce the protocol overhead, 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 Hybrid Automatic Repeat reQuest (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 16000 Bytes. NAS packets are encapsulated within an RRC (Radio Resource Control) TGPP36331 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.
Figure 6: DoNAS transmission sequence from an Uplink initiated access
8. Normative References


Internet-Draft              LPWAN SCHC NB-IoT                   May 2022


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
Abstract

The Generic Framework for Static Context Header Compression and Fragmentation (SCHC) specification describes two mechanisms: i) an application header compression scheme, and ii) a frame fragmentation and loss recovery functionality. SCHC offers a great level of flexibility that can be tailored for different Low Power Wide Area Network (LPWAN) technologies.

The present document provides the optimal parameters and modes of operation when SCHC is implemented over a Sigfox LPWAN. This set of parameters are also known as a "SCHC over Sigfox profile."

Status of This Memo

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

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

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

This Internet-Draft will expire on 25 August 2022.

1. Introduction ......................................................... 3
2. Terminology ......................................................... 3
3. SCHC over Sigfox .................................................. 3
   3.1. Network Architecture ........................................ 3
   3.2. Uplink ........................................................ 5
   3.3. Downlink ....................................................... 6
   3.4. SCHC-ACK on Downlink ......................................... 7
   3.5. SCHC Rules ..................................................... 7
   3.6. Fragmentation .................................................. 7
       3.6.1. Uplink Fragmentation ...................................... 8
       3.6.2. Downlink Fragmentation .................................... 11
   3.7. SCHC-over-Sigfox F/R Message Formats ....................... 12
       3.7.1. Uplink ACK-on-Error Mode: Single-byte SCHC Header ... 13
       3.7.2. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 2 ...
       3.8. SCHC-Sender Abort ........................................... 16
   3.9. SCHC-Receiver Abort .......................................... 18
   3.10. Padding ....................................................... 19
4. Fragmentation Sequence Examples .................................. 20
   4.1. Uplink No-ACK Examples ....................................... 20
   4.2. Uplink ACK-on-Error Examples: Single-byte SCHC Header ... 21
   4.3. SCHC Abort Examples ......................................... 27
5. Security considerations ............................................ 28
6. Acknowledgements .................................................. 28
7. References ........................................................ 29
   7.1. Normative References ......................................... 29
   7.2. Informative References ....................................... 29
Authors’ Addresses .................................................... 29
1. Introduction

The Generic Framework for Static Context Header Compression and Fragmentation (SCHC) specification [RFC8724] describes two mechanisms: i) a frame fragmentation and loss recovery functionality, and ii) an application header compression scheme. Either can be used on top of all the four LWPAN technologies defined in [RFC8376]. These LWPANs 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 LWPAN 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 LWPAN technology.

This document describes the recommended parameters, settings, and modes of operation to be used when SCHC is implemented over a Sigfox LWPAN. This set of parameters are also known as a "SCHC over Sigfox profile" or simply "SCHC/Sigfox."

2. Terminology

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

3. SCHC over Sigfox

The Generic SCHC Framework described in [RFC8724] takes advantage of the predictability of data flows existing in LWPAN 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). 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 (RG) is a Sigfox Base Station and the Network Gateway (NGW) is the Sigfox cloud-based Network.
In the case of the global Sigfox Network, RGs (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 Sigfox base stations in the world, providing hence a global single star network topology.

The Device sends application flows that are compressed and/or fragmented by a SCHC Compressor/Decompressor (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 forward 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 Dev 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 space diversity, as potentially an uplink message will be received by several base stations.

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)
* 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.

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.

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

Figure 2: SCHC Message in Sigfox

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).

3.3. Downlink

Downlink transmissions are Device-driven and can only take place following an uplink communication that so indicates. Hence, a Device explicitly indicates its intention to receive a downlink message using a downlink request flag when sending the preceding uplink message to the network. 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 UL 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].
3.4. 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 bi-directional 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 UL ACK-on-Error fragmentation mode, a DL opportunity MUST be indicated by the last fragment of every window (i.e. FCN = All-0, or FCN = All-1). 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 in case 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 DL transmission opportunity, then no DL transmission takes place at that opportunity and after a timeout the UL transmissions continue. Intermediate SCHC fragments with FCN different from All-0 or All-1 MUST NOT use the DL request flag to request a SCHC-ACK.

3.5. SCHC Rules

The RuleID MUST be included in the SCHC header. The total number of rules to be used affects directly the Rule ID 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.

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.

3.6. 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 receive end. Since only UL messages/fragments that have passed the Sigfox CRC-check are delivered to the Network 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 3.6.1.5 is RECOMMENDED.

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

3.6.1. 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. Data packets are self-contained (aka "message in a bottle") with all the required information for the network to process them accordingly. Hence, 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.

3.6.1.1. Uplink No-ACK Mode

No-ACK is RECOMMENDED to 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.
Since there are no multiple windows in the No-ACK mode, the W bit is not present. However it is **RECOMMENDED** to 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.

The **RECOMMENDED** Fragmentation Header size is 8 bits, and it is composed as follows:

* RuleID size: 4 bits
* DTag size (T): 0 bits
* Fragment Compressed Number (FCN) size (N): 4 bits
* As per [RFC8724], in the No-ACK mode the W (window) field is not present.
* RCS size: 0 bits (Not used)

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

ACK-on-Error with single-byte header is **RECOMMENDED** for medium to large size packets that need to be sent reliably. ACK-on-Error is optimal for 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.

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

* Rule ID size: 3 bits
* DTag size (T): 0 bits
* Window index (W) size (M): 2 bits
* Fragment Compressed Number (FCN) size (N): 3 bits
* MAX_ACK_REQUESTS: 5
3.6.1.3. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1

ACK-on-Error with two-byte header is RECOMMENDED for very large size packets that need to be sent reliably. ACK-on-Error is optimal for 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 order to allow transmission of large packets/files up to 480 bytes long, the SCHC uplink Fragmentation Header size is RECOMMENDED to be 16 bits in size and composed as follows:

* Rule ID size is: 6 bits
* DTag size (T) is: 0 bits
* Window index (W) size (M): 2 bits
* Fragment Compressed Number (FCN) size (N): 4 bits.
* MAX_ACK_REQUESTS: 5
* WINDOW_SIZE: 12 (with a maximum value of FCN=0b1011)
* Tile size: 10 bytes
* Retransmission Timer: Application-dependent
* Inactivity Timer: Application-dependent
* RCS size: 4 bits

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

In order to allow transmission of very large packets/files up to 2250 bytes long, the SCHC uplink Fragmentation Header size is RECOMMENDED to be 16 bits in size and composed as follows:
* Rule ID size is: 8 bits
* DTag size (T) is: 0 bits
* Window index (W) size (M): 3 bits
* Fragment Compressed Number (FCN) size (N): 5 bits.
* MAX_ACK_REQUESTS: 5
* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)
* Tile size: 10 bytes
* Retransmission Timer: Application-dependent
* Inactivity Timer: Application-dependent
* RCS size: 5 bits

3.6.1.5. All-1 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 comprised of less than WINDOW_SIZE fragments), an All-1 fragment may follow a value of FCN higher than 1 (0b01). In this case, the receiver could not derive from the FCN values alone whether there are any missing fragments right before the All-1 fragment or not.

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.

3.6.2. 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 DL communications.

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 is RECOMMENDED.

The SCHC downlink Fragmentation Header size is RECOMMENDED to be 8 bits in size and is composed as follows:

* RuleID size: 3 bits
* DTag size (T): 0 bits
* Window index (W) size (M) is: 0 bits
* Fragment Compressed Number (FCN) size (N): 5 bits
* MAX_ACK_REQUESTS: 5
* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)
* Tile size: 7 bytes
* Retransmission Timer: Application-dependent
* Inactivity Timer: Application-dependent
* RCS size: 0 bits (Not used)

3.7. 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.
3.7.1. Uplink ACK-on-Error Mode: Single-byte SCHC Header

3.7.1.1. 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.

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

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

The use of SCHC ACK REQ is NOT RECOMMENDED, instead the All-1 SCHC Fragment SHOULD 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.

3.7.1.2. All-1 SCHC Fragment

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

```
|---  SCHC Fragment Header ---|
+ --------------------------- + ------------ +
| RuleID |   W    | FCN=ALL-1 |    Payload   |
+ ------ + ------ + --------- + ------------ +
| 3 bits | 2 bits | 3 bits   | 8 to 88 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 Rule ID, M, and N values). The All-1 MUST have the last tile of the SCHC Packet, which MUST be of at least 1 byte. The SCHC Sender-Abort message header size is of 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.
3.7.1.3. SCHC ACK Format

Figure 5 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 5: 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 6. The window numbered 00, if present in the SCHC Compound ACK, MUST be placed between the Rule ID and the C bit to avoid confusion with padding bits. As padding is needed for the SCHC Compound ACK, padding bits MUST be 0 to make subsequent window numbers and bitmaps distinguishable.

```
|---- SCHC ACK Header ----| - W = x - |...| --- W = x + i ---|
+ ----------------------- + ------ +...+ ------- + ------ + ------- +
| RuleID | W=b'x | C=b'0 | Bitmap |...| W=b’x+i | Bitmap | b’0-pad |
+ ------ + ------ + ----- + ------ +...+ ------- + ------ + ------- +
| 3 bits | 2 bits | 1 bit | 7 bits |   | 2 bits  | 7 bits |
```

On top are noted the window number of the corresponding bitmap. Losses are found in windows x,...,x+i.

Figure 6: SCHC Compound ACK message format

The following figures show examples of the SCHC Compound ACK message format, when used on SCHC over Sigfox.

```
|---- SCHC ACK Header ----| - W=00 - |----- W=01 ------|
+ ----------------------- + ------ +------ + ------ + ------ +
| RuleID | W=b’00 | C=b’0 | Bitmap | W=b’01 | Bitmap | b’0-pad |
+ ------ + ------ + ----- + ------ + ------ + ------ +------ +
| 3 bits | 2 bits | 1 bit | 7 bits | 2 bits | 7 bits | 42 bits |
```

Losses are found in windows 00 and 01.

Figure 7: SCHC Compound ACK example 1
Losses are found in windows 01 and 11.

Figure 8: SCHC Compound ACK example 2

Losses are found in windows 00 and 10.

Figure 9: SCHC Compound ACK example 3

Figure 10 shows the SCHC Compound ACK message format when losses are found in all windows. The window numbers and its corresponding bitmaps are ordered from window numbered 00 to 11, notifying all four possible windows.

--- W=b’10 --| W=b’11 --|

... W=b’10|Bitmap|W=b’11|Bitmap|b’0-pad|

Losses are found in windows 00, 01, 10 and 11.

Figure 10: SCHC Compound ACK example 4
|- SCHC ACK Header -| W=b’00 |-- W=b’01 ---| W=b’10 --| 
+-------------------+------+------+------+------+------+-------+
| RuleID| W=b'00| C=b'0| Bitmap| W=b'01| Bitmap| W=b'10| Bitmap| b'0-pad| 
+-------------------+------+------+------+------+------+-------+
| 3 bits| 2 bits| 1 bit| 7 bits| 2 bits| 7 bits| 2 bits| 7 bits| 33 bits|

Losses are found in windows 00, 01 and 10.

Figure 11: SCHC Compound ACK example 5

3.7.1.4. SCHC Sender-Abort Message format

|---- Sender-Abort Header ----|
+-----------------------------+
| RuleID | W | FCN=ALL-1 |
+ ------ + ------ + --------- +
| 3 bits | 2 bits | 3 bits |

Figure 12: SCHC Sender-Abort message format

3.7.1.5. SCHC Receiver-Abort Message format

|---- Receiver-Abort Header ----|
+-----------------------------+
| RuleID | W=b’11 | C=b’1 | b’1-pad |
+ ------ + ------ + ----- + ------- +
| 3 bits | 2 bits | 1 bit | 58 bits |

Figure 13: SCHC Receiver-Abort message format

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

3.7.2.1. Regular SCHC Fragment

Figure 14 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 14: Regular SCHC Fragment format for all fragments except the last one
The use of SCHC ACK is NOT RECOMMENDED, instead the All-1 SCHC Fragment SHOULD 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).

3.7.2.2. All-1 SCHC Fragment

Figure 15 shows an example of the All-1 message. The All-1 message MUST contain the last tile of the SCHC Packet.

As per [RFC8724] the All-1 must be distinguishable from the a SCHC Sender-Abort message (with same Rule ID, 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 of 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.

3.7.2.3. SCHC ACK Format

Figure 16 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.

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 17. The SCHC Compound ACK can report up to 3 windows with losses. The window number (W) and its corresponding bitmap MUST be ordered from the lowest-numbered
window number to the highest-numbered window. If window numbered 000 is present in the SCHC Compound ACK, the window number 000 MUST be placed between the Rule ID and C bit to avoid confusion with padding bits.

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.

![Diagram of SCHC ACK Header](image)

On top are noted the window number of the corresponding bitmap. Losses are found in windows x,...,x+i.

Figure 17: SCHC Compound ACK message format

3.7.2.4. SCHC Sender-Abort Messages

![Diagram of SCHC Sender-Abort Header](image)

Figure 18: SCHC Sender-Abort message format

3.7.2.5. SCHC Receiver-Abort Message

![Diagram of SCHC Receiver-Abort Header](image)

Figure 19: SCHC Receiver-Abort message format

3.8. 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/Sigfox, a SCHC-Sender Abort
MUST be sent if the number of repeated All-1s (i.e., with the same
bitmap) sent in sequence is greater than or equal to
MAX_ACK_REQUESTS.

* The MAX_ACK_REQUEST counter MUST be reset when a SCHC ACK is
successfully received.

3.9. 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 SCHC/Sigfox 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.

* A SCHC-Receiver Abort can be triggered when the number of ACK
attempts is not strictly less than MAX_ACK_REQUESTS. In the case
of SCHC/Sigfox, a SCHC-Receiver Abort MUST be sent if the number
of repeated SCHC ACKs sent in a row (i.e., synchronized with the
ACK REQ case, and with identical bitmaps) is greater than or equal
to MAX_ACK_REQUESTS.

* 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.

3.10. Padding

The Sigfox payload fields have different characteristics in uplink
and downlink.

Uplink frames 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.

4. 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.

4.1. 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.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------FCN=6 (0110), Seq=1--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=5 (0101), Seq=2--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=4 (0100), Seq=3--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=3 (0011), Seq=4--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=2 (0010), Seq=5--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=1 (0001), Seq=6--------&gt;</td>
<td></td>
</tr>
<tr>
<td>----------FCN=15 (1111), Seq=7--------&gt;</td>
<td>All fragments received</td>
</tr>
</tbody>
</table>

(End)

Figure 20: UL 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 a FCN equal to 15).

Case losses on any fragment except the first.
4.2. 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 DL ACK can be sent opportunistically.

Case No losses

The downlink flag must be enabled in the sender UL message to allow a DL message from the receiver. The DL Enable in the figures shows where the sender should enable the downlink, and wait for an ACK.

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-------> | Missing Fragment - Unable to reassemble
(End)

Figure 21: UL No-ACK Losses (scenario 1)

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-----> |
|-----W=0, FCN=0, Seq=7-----> | DL Enable
(no ACK)

Figure 22: UL 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 correspondingbitmap 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.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>------W=0, FCN=6, Seq=1------</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=5, Seq=2--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=4, Seq=3------</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=3, Seq=4------</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=2, Seq=5--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=1, Seq=6------</td>
<td></td>
</tr>
<tr>
<td>DL Enable</td>
<td>----W=0, FCN=0, Seq=7------&gt;</td>
</tr>
<tr>
<td>d FCN=2, Seq=5</td>
<td>Missing Fragments W=0 =&gt; FCN=5, Seq=2 and FCN=2, Seq=5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;----- ACK, W=0, C=0 ------</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=5, Seq=8------</td>
<td></td>
</tr>
<tr>
<td>------W=0, FCN=2, Seq=9------</td>
<td></td>
</tr>
<tr>
<td>------W=1, FCN=6, Seq=10-----&gt;</td>
<td></td>
</tr>
<tr>
<td>------W=1, FCN=5, Seq=11------</td>
<td></td>
</tr>
<tr>
<td>------W=1, FCN=4, Seq=12------</td>
<td></td>
</tr>
<tr>
<td>DL Enable</td>
<td>------W=1, FCN=7, Seq=13----&gt;</td>
</tr>
<tr>
<td></td>
<td>All fragments received</td>
</tr>
<tr>
<td></td>
<td>&lt;&lt;----- ACK, W=1, C=1 ------</td>
</tr>
</tbody>
</table>

(End)

Figure 23: UL 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.
In the following diagram, besides the All-0 there are other fragment losses in the first window (W=0).

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-----W=0, FCN=6, Seq=1-------&gt;</td>
</tr>
</tbody>
</table>
| |-----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---->
| DL Enable |
| |-----W=1, FCN=7, Seq=11------> Missing Fragment W=0, FCN=0, Seq=7 |
| |<-------- ACK, W=0, C=0 ------| Bitmap:1111110 |
| |-----W=0, FCN=0, Seq=13------> All fragments received |
| DL Enable |
| |-----W=1, FCN=7, Seq=14------> |
| |<-------- ACK, W=1, C=1 ------| C=1 |
| (End) |

Figure 24: UL ACK-on-Error All-0 Lost on First Window
Figure 25: UL 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) should always finish with an All-1, as it serves as an ACK Request message to confirm the correct reception of the retransmitted fragments.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
</table>
| W=0, FCN=6 (110), Seq=1----> | W=0, FCN=5 (101), Seq=2--X-->
| W=0, FCN=5 (101), Seq=2----> | W=0, FCN=4 (100), Seq=3---->
| W=0, FCN=4 (100), Seq=3----> | W=0, FCN=3 (011), Seq=4--X-->
| W=0, FCN=3 (011), Seq=4--X---> | W=0, FCN=2 (010), Seq=5----->
| W=0, FCN=2 (010), Seq=5------> | W=0, FCN=1 (001), Seq=6------>
| W=0, FCN=1 (001), Seq=6-------> | W=0, FCN=0 (000), Seq=7--X-->|
| DL enable (no ACK) | DL enable |
| W=1, FCN=6 (110), Seq=8--X--> | W=1, FCN=7 (111), Seq=11----> |
| W=1, FCN=5 (101), Seq=9-----| W=1, FCN=4 (011), Seq=10--X-->
| W=1, FCN=4 (011), Seq=10--X--> | Missing Fragment W=0 => FCN= 5, 3
| DL enable and 0, W=1 => FCN= 6 and 4 | Bitmap W=0:1010110, Bitmap W=1:010 001
| ---- Compound ACK, W=0,1, C=0 ---- | |
| W=0, FCN=5 (101), Seq=13-----> | W=0, FCN=3 (011), Seq=14---->
| W=0, FCN=3 (011), Seq=14-----> | W=0, FCN=0 (000), Seq=15---->
| W=0, FCN=0 (000), Seq=15-------> | W=1, FCN=6 (110), Seq=16---->
| W=1, FCN=6 (110), Seq=16------> | W=1, FCN=4 (011), Seq=17---->
| W=1, FCN=4 (011), Seq=17-------> | All fragments received
| DL enable | DL enable |
| W=1, FCN=7 (111), Seq=18------> | W=1, FCN=7 (111), Seq=18------>
| <-------- ACK, W=1, C=1 -------- | C=1
| (End) | (End) |

Figure 26: UL ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (1)

Similar case as above, but with less fragments in the second window (W=1)
Figure 27: UL 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.
Figure 28: UL ACK-on-Error ACK Lost

Case SCHC Compound ACK at the end

In this example, SCHC Fragment losses are found in both windows 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.
4.3. 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----->|                   |
|-----W=0, FCN=0, Seq=7----->|                   |

DL Enable (no ACK)
|-----W=1, FCN=7, Seq=8----->|                   |
|-----W=1, FCN=6, Seq=9----->|                   |
|-----W=1, FCN=5, Seq=10-----|                   |
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=11-----| All fragments received
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=14-----| RESEND ACK (1)
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=15-----| RESEND ACK (2)
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=16-----| RESEND ACK (3)
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=17-----| RESEND ACK (4)
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=18-----| RESEND ACK (5)
<------- ACK, W=1, C=1 ---X--| C=1

DL Enable
|-----W=1, FCN=7, Seq=19-----| exit with error condition
|-----Sender-Abort, Seq=19----->| exit with error condition

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 DL enable.
5. 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 with the ID claimed in the message.

Application data can be encrypted at the application level 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 confidentiality keys are pre-provisioned. A confidentiality key is only provisioned if confidentiality is to be used.

The radio protocol has protections against reply attacks, and the cloud-based core network provides firewalling protection against undesired incoming communications.

6. 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 Clement Mannequin, Rafael Vidal, Julien Boite, Renaud Marty, and Antonis Platis for their useful comments and implementation design considerations.

7. References

7.1. Normative References

[I-D.ietf-lpwan-schc-compound-ack]


7.2. Informative References

[sigfox-callbacks]

[sigfox-spec]

Authors’ Addresses

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

Data Model for Static Context Header Compression (SCHC)
draft-ietf-lpwan-schc-yang-data-model-11

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 20 November 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.
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 a non-formal 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,...)
This document defines a YANG module to represent both compression and fragmentation rules, which leads to common representation for values for all the rules elements.

2. SCHC rules

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 a non formal 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,...)

* ...

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

SCHC compression is generic, the main mechanism does not refer to a specific protocol. Any header field is abstracted through an ID, a position, a direction, and a value that can be a numerical value or a string. [RFC8724] and [RFC8824] specify fields for IPv6, UDP, CoAP and OSCORE.

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

The YANG model allows to select the compression or the fragmentation using the feature command.
feature compression {
  description
    "SCHC compression capabilities are taken into account";
}

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

Figure 1: Feature for compression and fragmentation.

2.1. Compression Rules

[RFC8724] proposes a non formal 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 2: Compression Decompression Context

2.2. Identifier generation

Identifier used in the SCHC YANG Data Model are from the identityref statement to ensure to be globally unique and be easily augmented 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 3) shows how an identityref is created for RCS algorithms used during SCHC fragmentation.

```yang
// -- 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."
}

identity rcs-RFC8724 {
  base rcs-algorithm-base-type;
  description
    "CRC 32 defined as default RCS in RFC8724. RCS is 4 byte-long";
}

typedef rcs-algorithm-type {
  type identityref {
    base rcs-algorithm-base-type;
  }
  description
    "type used in rules.";
}

Figure 3: Principle to define a type based on identityref.

2.3.  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 model, each field MUST be identified through a global YANG identityref. A YANG field ID for the protocol always derives 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 7. The example
Figure 4 gives the first field ID definitions. A type is defined for
IPv6 protocol, and each field is based on it. Note that the DiffServ
bits derives from the Traffic Class identity.

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";
}

identity fid-ipv6-version {
    base fid-ipv6-base-type;
    description
        "IPv6 version field from RFC8200";
}

identity fid-ipv6-trafficclass {
    base fid-ipv6-base-type;
    description
        "IPv6 Traffic Class field from RFC8200";
}

identity fid-ipv6-trafficclass-ds {
    base fid-ipv6-trafficclass;
    description
        "IPv6 Traffic Class field from RFC8200,
         DiffServ field from RFC3168";
}

Figure 4: Definition of identityref for field IDs

The type associated to this identity is fid-type (cf. Figure 5)
typedef fid-type {
    type identityref {
        base fid-base-type;
    } description
        "Field ID generic type.";
}

Figure 5: Type definition for field IDs

2.4. 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.

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 in RFC 8824 (cf. 5.3).";
}

identity fl-token-length {
    base fl-base-type;
    description
        "Residue length in Byte is sent as defined for CoAP in RFC 8824 (cf. 4.5).";
}

Figure 6: Definition of identityref for Field Length

The field length function can be defined as an identityref as shown in Figure 6.

Therefore, the type for field length is a union between an integer giving in bits the size of the length and the identityref (cf. Figure 7).
typedef fl-type {
    type union {
        type int64; /* positive integer, expressing length in bits */
        type identityref { /* function */
            base fl-base-type;
        }
    }
    description
    "Field length either a positive integer expressing the size in bits or a function defined through an identityref."
}

Figure 7: Type definition for field Length

2.5. Field position

Field position is a positive integer which gives the position of a field, 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 an uint8.

2.6. Direction Indicator

The Direction Indicator (di) is used to tell if a field appears in both direction (Bi) or only uplink (Up) or Downlink (Dw).
identity di-base-type {
    description
    "Used to extend direction indicators.";
}

identity di-bidirectional {
    base di-base-type;
    description
    "Direction Indication of bidirectionality in RFC 8724 (cf. 7.1).";
}

identity di-up {
    base di-base-type;
    description
    "Direction Indication of uplink defined in RFC 8724 (cf. 7.1).";
}

identity di-down {
    base di-base-type;
    description
    "Direction Indication of downlink defined in RFC 8724 (cf. 7.1).";
}

Figure 8: Definition of identityref for direction indicators

Figure 8 gives the identityref for Direction Indicators. The naming convention is "di" followed by the Direction Indicator name.

The type is "di-type" (cf. Figure 9).

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.";
}

Figure 9: Type definition for direction indicators
2.7. Target Value

The Target Value is a list of binary sequences of any length, aligned to the left. Figure 10 shows the definition of a single element of a Target Value. 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. The index allows to specify several values:

* For Equal and LSB, 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.

```yaml
grouping tv-struct {
  description "Defines the target value element. Always a binary type, strings must be converted to binary. field-id allows the conversion to the appropriate type.";
  leaf value {
    type binary;
    description "Target Value";
  }
  leaf index {
    type uint16;
    description "Index gives the position in the matching-list. If only one element is present, index is 0. Otherwise, indicia is the order in the matching list, starting at 0.";
  }
}
```

Figure 10: Definition of target value

2.8. 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 as listed in Figure 11.
identity mo-base-type {
    description
    "Used to extend Matching Operators with SID values";
}

identity mo-equal {
    base mo-base-type;
    description
    "Equal MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-ignore {
    base mo-base-type;
    description
    "Ignore MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-msb {
    base mo-base-type;
    description
    "MSB MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-match-mapping {
    base mo-base-type;
    description
    "match-mapping MO as defined in RFC 8724 (cf. 7.3)";
}

Figure 11: Definition of identityref for Matching Operator

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

The type is "mo-type" (cf. Figure 12)

typedef mo-type {
    type identityref {
        base mo-base-type;
    }
    description
    "Matching Operator (MO) to compare fields values with
target values";
}

Figure 12: Type definition for Matching Operator
2.8.1. Matching Operator arguments

They are viewed as a list, built with a tv-struct (see chapter Section 2.7).

2.9. Compression Decompression Actions

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

Figure 14 shows some CDA definition, the full definition is in Section 7.

```yaml
identity cda-base-type {
  description
    "Compression Decompression Actions.";
}

identity cda-not-sent {
  base cda-base-type;
  description
    "not-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-value-sent {
  base cda-base-type;
  description
    "value-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-lsb {
  base cda-base-type;
  description
    "LSB CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-mapping-sent {
  base cda-base-type;
  description
    "mapping-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-compute {
  base cda-base-type;
  description
    "compute-* CDA as defined in RFC 8724 (cf. 7.4)";
}

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

typedef cda-type {
    type identityref {
        base cda-base-type;
    }
    description
    "Compression Decompression Action to compression or decompress a field."
}
Figure 15 shows the definition for identifiers from these three modes.

```yamls
identity fragmentation-mode-base-type {
  description
    "fragmentation mode."
}
```

```yamls
identity fragmentation-mode-no-ack {
  base fragmentation-mode-base-type;
  description
    "No-ACK of RFC8724."
}
```

```yamls
identity fragmentation-mode-ack-always {
  base fragmentation-mode-base-type;
  description
    "ACK-Always of RFC8724."
}
```

```yamls
identity fragmentation-mode-ack-on-error {
  base fragmentation-mode-base-type;
  description
    "ACK-on-Error of RFC8724."
}
```

```yamls
typedef fragmentation-mode-type {
  type identityref {
    base fragmentation-mode-base-type;
  }
  description
    "type used in rules"
}
```

Figure 15: Definition of fragmentation mode identifier

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

2.10.2. Fragmentation Header

A data fragment header, starting with the rule ID can be sent on the fragmentation direction. The SCHC header may be composed of (cf. Figure 16):

* 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 --|
+-- ... --|DTag | W | FCN | Fragment Payload | padding (as needed) |
+-- ... --|-----|----|------|------------------|

Figure 16: Data fragment header from RFC8724

2.10.3. 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 identity of the RCS algorithm
is shown in Figure 17.

identity rcs-algorithm-base-type {
  description
  "Identify which algorithm is used to compute RCS.
  The algorithm also defines the size of the RCS field.";
}

identity rcs-RFC8724 {
  base rcs-algorithm-base-type;
  description
  "CRC 32 defined as default RCS in RFC8724. RCS is 4 byte-long"
}

typedef rcs-algorithm-type {
  type identityref {
    base rcs-algorithm-base-type;
  }
  description
  "type used in rules.";
}

Figure 17: type definition for RCS

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 defined in Figure 18 allows to define the behavior:

* all1-data-no: the last fragment contains no data, just the RCS
* all1-data-yes: the last fragment includes a single tile and the RCS
* all1-data-sender-choice: the last fragment may or may not contain a single tile. The receiver can detect if a tile is present.

```plaintext
definition all1-data-base-type {
  description
    "Type to define when to send an Acknowledgment message.";
}
definition all1-data-no {
  base all1-data-base-type;
  description
    "All1 contains no tiles.";
}
definition all1-data-yes {
  base all1-data-base-type;
  description
    "All1 MUST contain a tile.";
}
definition all1-data-sender-choice {
  base all1-data-base-type;
  description
    "Fragmentation process chooses to send tiles or not in all1.";
}
typedef all1-data-type {
  type identityref {
    base all1-data-base-type;
  }
  description
    "Type used in rules.";
}
```

Figure 18: type definition for RCS

The naming convention is "all1-data" followed by the behavior identifier.
2.10.4. Acknowledgment behavior

The acknowledgment fragment header goes in the opposite direction of data. The header is composed of (see Figure 19):

* 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 19: 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 following identifiers (cf. Figure 20) define the acknowledgment behavior.
identity ack-behavior-base-type {
  description
    "Define when to send an Acknowledgment.";
}

identity ack-behavior-after-All0 {
  base ack-behavior-base-type;
  description
    "Fragmentation expects Ack after sending All0 fragment.";
}

identity ack-behavior-after-All1 {
  base ack-behavior-base-type;
  description
    "Fragmentation expects Ack after sending All1 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
    "Type used in rules.";
}

Figure 20: bitmap generation behavior

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

2.10.5. Fragmentation Parameters

The state machine requires some common values to handle fragmentation:

* retransmission-timer expresses, in seconds, the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specified, value must be higher or equal to 1.
* inactivity-timer expresses, in seconds, 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.

* max-ack-requests expresses the number of attempts before aborting (cf. section 8.2.2.4. of [RFC8724]).

* maximum-packet-size reexpresses, in bytes, the larger packet size that can be reassembled.

They are defined as unsigned integers, see Section 7.

2.10.6. Layer 2 parameters

The data model includes two parameters needed for fragmentation:

* l2-word-size: [RFC8724] base fragmentation 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 a uncompressed datagram. By default, the value is set to 1280 bytes.

They are defined as unsigned integer, see Section 7.

3. 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 types 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 in extenso 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.
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.";
}

// SCHC table for a specific device.

container schc {
  list rule {
    key "rule-id-value rule-id-length";
    uses rule-id-type;
    choice nature {
      case fragmentation {
        if-feature "fragmentation";
        uses fragmentation-content;
      }
      case compression {
        if-feature "compression";
        uses compression-content;
      }
      case no-compression {
        description
        "RFC8724 requires a rule for uncompressed headers.";
      }
    }
    description
    "Set of rules compression, no compression or fragmentation rules identified by their rule-id.";
  }
  description
  "Set of rules compression, no compression or fragmentation rules identified by their rule-id.";
}
"a SCHC set of rules is composed of a list of rules which are used for compression, no-compression or fragmentation."

Figure 21: Definition of a SCHC Context

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.

3.1. Compression rule

A compression rule is composed of entries describing its processing (cf. Figure 22). An entry contains all the information defined in Figure 2 with the types defined above.

The compression rule described Figure 2 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 2. Their type reflects the identifier types defined in Section 2.1

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

```yml
grouping compression-rule-entry {
  description
    "These entries define 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. The content is a YANG identifier.
- Field Length: either a positive integer of a function defined as a YANG id.
- Field Position: a positive (and possibly equal to 0) integer.
- Direction Indicator: a YANG identifier giving the direction.
- Target value: a value against which the header Field is compared.
Internet-Draft           LPWAN SCHC YANG module                 May 2022

- Matching Operator: a YANG id giving the operation, parameters may be associated to that operator.
- Comp./Decomp. Action: A YANG id giving the compression or decompression action, parameters may be associated to that action.

leaf field-id {
  type schc:fid-type;
  mandatory true;
  description
  "Field ID, identify a field in the header with a YANG referenceid.";
}
leaf field-length {
  type schc:fl-type;
  mandatory true;
  description
  "Field Length, expressed in number of bits or through a function defined as a YANG referenceid.";
}
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, a YANG referenceid to say if the packet is bidirectional, up or down";
}
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.

Minaburo & Toutain      Expires 20 November 2022               [Page 22]
For use as a matching list for the mo-match-mapping matching operator, positions should take consecutive values starting from 1.

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";
}
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;
  mandatory true;
  description
  "CDA: Compression Decompression Action."
}
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."
}
}

grouping compression-content {

list entry {
    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."
    }
}

Figure 22: Definition of a compression entry

3.2. Fragmentation rule

A Fragmentation rule is composed of entries describing the protocol behavior. Some of them are numerical entries, others are identifiers defined in Section 2.10.

The definition of a Fragmentation rule is divided into three subparts (cf. Figure 24):

* parameters such as the fragmentation-mode, the l2-word-size and the direction. Since Fragmentation rules are always defined for a specific direction, the value must be either di-up or di-down (di-bidirectional is not allowed).

* parameters defining the Fragmentation header format (dtag-size, w-size, fcn-size and rcs-algorithm).

* Protocol parameters for timers (inactivity-timer, retransmission-timer). [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. Figure 23 shows the two parameters defined for timers:

    - the duration of a tick is computed through this formula 2^tick-duration/10^6. When tick-duration is set to 0, the unit is the micro-second. The default value of 20 leads to a unit of about 1.05 seconds. A value of 32 leads to a tick duration of about 1.19 hours.

    - 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] recommandation.
Protocol behavior (maximum-packet-size, max-interleaved-frames, max-ack-requests). If these parameters are specific to a single fragmentation mode, they are grouped in a structure dedicated to that Fragmentation mode. If some parameters can be found in several modes, typically ACK-Always and ACK-on-Error, they are defined in a common part and a when statement indicates which modes are allowed.

grouping timer-duration {
  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;
    description
    "timer duration = ticks-numbers * 2^ticks-duration / 10^6";
  }
  description
  "used by inactivity and retransmission timer. Allows a precision from micro-second to year by sending the tick-duration value. For instance:

  tick-duration / smallest value  highest value

  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.
  ";
}

Figure 23: Timer duration values
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;
        mandatory true;
        description
            "which fragmentation mode is used (noAck, AckAlways,
            AckonError)";
    }
    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
            "Should be up or down, bidirectionnal is forbidden.";
    }
    // 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(../../../fragmentation-mode,
            'fragmentation-mode-ack-on-error')
            or
            derived-from(../../../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-RFC8724";
  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 at most max-interleaved-frames must be active at any time.";
}
container inactivity-timer {
  uses timer-duration;
  description
    "Duration is seconds of the inactivity timer, 0 indicates that the timer is disabled.";
}
container retransmission-timer {
  uses timer-duration;
  when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')"
    or
  derived-from(../fragmentation-mode,
leaf max-ack-requests {
  when "derived-from(../fragmentation-mode,
       'fragmentation-mode-ack-on-error')"
    or
    derived-from(../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(../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-All1 {
      when "derived-from(../fragmentation-mode,
        'fragmentation-mode-ack-on-error')"
      type schc:all1-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(../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.";
Figure 24: Fragmentation Parameters

3.3. YANG Tree

module: ietf-schc
+--rw schc
  +--rw rule* [rule-id-value rule-id-length]
    +--rw rule-id-value           uint32
    +--rw rule-id-length          uint8
    +--rw (nature)?
      +--rw:(fragmentation) {fragmentation}?
        +--rw fragmentation-mode      schc:fragmentation-mode-type
        +--rw 12-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)?
          +--rw:(no-ack)
          +--rw:(ack-always)
          +--rw:(ack-on-error)
            +--rw tile-size?            uint8
            +--rw tile-in-All1?          schc:all1-data-type
            +--rw ack-behavior?          schc:ack-behavior-type
          +--rw:(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 value?               binary
                +--rw index                 uint16
              +--rw matching-operator      schc:mo-type
4. IANA Considerations

This document has no request to IANA.

5. Security considerations

This document does not have any more Security consideration than the ones already raised in [RFC8724] and [RFC8824].

6. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Alexander Pelov for their careful reading and valuable inputs. A special thanks for Carl Moberg for his patience and wise advices when building the model.

7. YANG Module

```yang
<code begins> file ietf-schc@2022-02-15.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:p-wan@ietf.org>  
    Editor:   Laurent Toutain  
    <mailto:laurent.toutain@imt-atlantique.fr>  
    Editor:   Ana Minaburo  
    <mailto:ana@ackl.io>"

  description 
    "Copyright (c) 2021 IETF Trust and the persons identified as
```

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 an abstract way through a table.

<table>
<thead>
<tr>
<th>(FID)</th>
<th>Rule 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Field 1</td>
</tr>
<tr>
<td></td>
<td>Field 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>Field N</td>
</tr>
</tbody>
</table>

This module proposes a global data model that can be used for rule exchanges or modification. It proposes 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 2022-02-15 {
  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";
}

identity fid-ipv6-version {
  base fid-ipv6-base-type;
  description
    "IPv6 version field from RFC8200";
}

identity fid-ipv6-trafficclass {
  base fid-ipv6-base-type;
  description
    "IPv6 Traffic Class field from RFC8200";
}

identity fid-ipv6-trafficclass-ds {
  base fid-ipv6-trafficclass;
description
 "IPv6 Traffic Class field from RFC8200,
  DiffServ field from RFC3168";
}

identity fid-ipv6-trafficclass-ecn {
  base fid-ipv6-trafficclass;
  description
   "IPv6 Traffic Class field from RFC8200,
    ECN field from RFC3168";
}

identity fid-ipv6-flowlabel {
  base fid-ipv6-base-type;
  description
   "IPv6 Flow Label field from RFC8200";
}

identity fid-ipv6-payloadlength {
  base fid-ipv6-base-type;
  description
   "IPv6 Payload Length field from RFC8200";
}

identity fid-ipv6-nextheader {
  base fid-ipv6-base-type;
  description
   "IPv6 Next Header field from RFC8200";
}

identity fid-ipv6-hoplimit {
  base fid-ipv6-base-type;
  description
   "IPv6 Next Header field from RFC8200";
}

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 if it is
    respectively an uplink or a downlink message."
}

identity fid-ipv6-deviid {
  base fid-ipv6-base-type;
  description
   "corresponds to either the source address or the destination
address prefix of RFC 8200. Depending if it is respectively an uplink or a downlink message."
}

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 if it is respectively a downlink or an uplink message.";
}

identity fid-ipv6-appiid {
    base fid-ipv6-base-type;
    description
        "corresponds to either the source address or the destination address prefix of RFC 8200. Depending if it is respectively a downlink or an uplink message.";
}

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

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

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

identity fid-udp-length {
    base fid-udp-base-type;
    description
        "UDP length from RFC 768";
}

identity fid-udp-checksum {
    base fid-udp-base-type;
}
description
   "UDP length from RFC 768";
}

identity fid-coap-base-type {
    base fid-base-type;
    description
       "Field ID base type for UDP headers described in RFC 7252";
}

identity fid-coap-version {
    base fid-coap-base-type;
    description
       "CoAP version from RFC 7252";
}

identity fid-coap-type {
    base fid-coap-base-type;
    description
       "CoAP type from RFC 7252";
}

identity fid-coap-tkl {
    base fid-coap-base-type;
    description
       "CoAP token length from RFC 7252";
}

identity fid-coap-code {
    base fid-coap-base-type;
    description
       "CoAP code from RFC 7252";
}

identity fid-coap-code-class {
    base fid-coap-code;
    description
       "CoAP code class from RFC 7252";
}

identity fid-coap-code-detail {
    base fid-coap-code;
    description
       "CoAP code detail from RFC 7252";
}

identity fid-coap-mid {
    base fid-coap-base-type;
description
   "CoAP message ID from RFC 7252";
}

identity fid-coap-token {
   base fid-coap-base-type;
   description
   "CoAP token from RFC 7252";
}

identity fid-coap-option-if-match {
   base fid-coap-base-type;
   description
   "CoAP option If-Match from RFC 7252";
}

identity fid-coap-option-uri-host {
   base fid-coap-base-type;
   description
   "CoAP option URI-Host from RFC 7252";
}

identity fid-coap-option-etag {
   base fid-coap-base-type;
   description
   "CoAP option Etag from RFC 7252";
}

identity fid-coap-option-if-none-match {
   base fid-coap-base-type;
   description
   "CoAP option if-none-match from RFC 7252";
}

identity fid-coap-option-observe {
   base fid-coap-base-type;
   description
   "CoAP option Observe from RFC 7641";
}

identity fid-coap-option-uri-port {
   base fid-coap-base-type;
   description
   "CoAP option Uri-Port from RFC 7252";
}

identity fid-coap-option-location-path {
   base fid-coap-base-type;
description
  "CoAP option Location-Path from RFC 7252";
}

identity fid-coap-option-uri-path {
  base fid-coap-base-type;
  description
    "CoAP option Uri-Path from RFC 7252";
}

identity fid-coap-option-content-format {
  base fid-coap-base-type;
  description
    "CoAP option Content Format from RFC 7252";
}

identity fid-coap-option-max-age {
  base fid-coap-base-type;
  description
    "CoAP option Max-Age from RFC 7252";
}

identity fid-coap-option-uri-query {
  base fid-coap-base-type;
  description
    "CoAP option Uri-Query from RFC 7252";
}

identity fid-coap-option-accept {
  base fid-coap-base-type;
  description
    "CoAP option Accept from RFC 7252";
}

identity fid-coap-option-location-query {
  base fid-coap-base-type;
  description
    "CoAP option Location-Query from RFC 7252";
}

identity fid-coap-option-block2 {
  base fid-coap-base-type;
  description
    "CoAP option Block2 from RFC 7959";
}

identity fid-coap-option-block1 {
  base fid-coap-base-type;
}
description
   "CoAP option Block1 from RFC 7959";
}

identity fid-coap-option-size2 {
  base fid-coap-base-type;
  description
    "CoAP option size2 from RFC 7959";
}

identity fid-coap-option-proxy-uri {
  base fid-coap-base-type;
  description
    "CoAP option Proxy-Uri from RFC 7252";
}

identity fid-coap-option-proxy-scheme {
  base fid-coap-base-type;
  description
    "CoAP option Proxy-scheme from RFC 7252";
}

identity fid-coap-option-size1 {
  base fid-coap-base-type;
  description
    "CoAP option Size1 from RFC 7252";
}

identity fid-coap-option-no-response {
  base fid-coap-base-type;
  description
    "CoAP option No response from RFC 7967";
}

identity fid-coap-option-oscore-flags {
  base fid-coap-base-type;
  description
    "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

identity fid-coap-option-oscore-piv {
  base fid-coap-base-type;
  description
    "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

identity fid-coap-option-oscore-kid {
  base fid-coap-base-type;
}
description
    "CoAP option oscore flags (see RFC 8824, section 6.4);"
}

identity fid-coap-option-oscore-kidctx {
    base fid-coap-base-type;
    description
        "CoAP option oscore flags (see RFC 8824, 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 in RFC 8824 (cf. 5.3).";
}

identity fl-token-length {
    base fl-base-type;
    description
        "Residue length in Byte is sent as defined
         for CoAP in RFC 8824 (cf. 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 in
         RFC 8724 (cf. 7.1).";
}
identity di-up {
    base di-base-type;
    description
        "Direction Indication of uplink defined in
         RFC 8724 (cf. 7.1).";
}

identity di-down {
    base di-base-type;
    description
        "Direction Indication of downlink defined in
         RFC 8724 (cf. 7.1).";
}

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

identity mo-base-type {
    description
        "Used to extend Matching Operators with SID values";
}

identity mo-equal {
    base mo-base-type;
    description
        "Equal MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-ignore {
    base mo-base-type;
    description
        "Ignore MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-msb {
    base mo-base-type;
    description
        "MSB MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-match-mapping {
    base mo-base-type;
    description
        "match-mapping MO as defined in RFC 8724 (cf. 7.3)";
}
// CDA type definition
//------------------------------

identity cda-base-type {
    description
        "Compression Decompression Actions."
}

identity cda-not-sent {
    base cda-base-type;
    description
        "not-sent CDA as defined in RFC 8724 (cf. 7.4)."
}

identity cda-value-sent {
    base cda-base-type;
    description
        "value-sent CDA as defined in RFC 8724 (cf. 7.4)."
}

identity cda-lsb {
    base cda-base-type;
    description
        "LSB CDA as defined in RFC 8724 (cf. 7.4)."
}

identity cda-mapping-sent {
    base cda-base-type;
    description
        "mapping-sent CDA as defined in RFC 8724 (cf. 7.4)."
}

identity cda-compute {
    base cda-base-type;
    description
        "compute-* CDA as defined in RFC 8724 (cf. 7.4)"
}

identity cda-deviid {
    base cda-base-type;
    description
        "devid CDA as defined in RFC 8724 (cf. 7.4)"
}

identity cda-appliid {
    base cda-base-type;
    description
        "appliid CDA as defined in RFC 8724 (cf. 7.4)";
typedef fid-type {
    type identityref {
        base fid-base-type;
    }
    description
        "Field ID generic type.";
}

typedef fl-type {
    type union {
        type int64; /* positive integer, expressing length in bits */
        type identityref { /* function */
            base fl-base-type;
        }
    }
    description
        "Field length either a positive integer expressing the size in bits or a function defined through an identityref.";
}

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.";
}

typedef mo-type {
    type identityref {
        base mo-base-type;
    }
    description
        "Matching Operator (MO) to compare fields values with target values";
}

typedef cda-type {
    type identityref {
        base cda-base-type;
    }
    description
"Compression Decompression Action to compression or decompress a field."
}

// -- FRAGMENTATION TYPE
// -- fragmentation modes

identity fragmentation-mode-base-type {
    description
    "fragmentation mode."
}

identity fragmentation-mode-no-ack {
    base fragmentation-mode-base-type;
    description
    "No-ACK of RFC8724."
}

identity fragmentation-mode-ack-always {
    base fragmentation-mode-base-type;
    description
    "ACK-Always of RFC8724."
}

identity fragmentation-mode-ack-on-error {
    base fragmentation-mode-base-type;
    description
    "ACK-on-Error of RFC8724."
}

typedef fragmentation-mode-type {
    type identityref {
        base fragmentation-mode-base-type;
    }
    description
    "type used in rules"
}

// -- Ack behavior

identity ack-behavior-base-type {
    description
    "Define when to send an Acknowledgment ."
}

identity ack-behavior-after-All0 {
    base ack-behavior-base-type;
    description
"Fragmentation expects Ack after sending All0 fragment."
}

identity ack-behavior-after-All1 {
    base ack-behavior-base-type;
    description
        "Fragmentation expects Ack after sending All1 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
        "Type used in rules."
}

// -- All1 with data types

identity all1-data-base-type {
    description
        "Type to define when to send an Acknowledgment message."
}

identity all1-data-no {
    base all1-data-base-type;
    description
        "All1 contains no tiles."
}

identity all1-data-yes {
    base all1-data-base-type;
    description
        "All1 MUST contain a tile."
}

identity all1-data-sender-choice {
    base all1-data-base-type;
    description
        "Fragmentation process chooses to send tiles or not in all1."
}
typedef all1-data-type {
    type identityref {
        base all1-data-base-type;
    }
    description
        "Type used 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.";
}

identity rcs-RFC8724 {
    base rcs-algorithm-base-type;
    description
        "CRC 32 defined as default RCS in RFC8724. RCS is 4 byte-long";
}

typedef rcs-algorithm-type {
    type identityref {
        base rcs-algorithm-base-type;
    }
    description
        "type used in rules.";
}

// --------- TIMER DURATION -------------------

grouping timer-duration {
    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;
        description
            "timer duration = ticks-numbers * 2^ticks-duration / 10^6";
    }
    description
        "used by inactivity and retransmission timer. Allows a precision from micro-second to year by sending the
tick-duration value.
For instance:

\[
\frac{\text{tick-duration}}{\text{smallest value}} = \text{highest value}
\]

\[
\begin{array}{cccccccc}
20: & 00y & 000d & 00h & 00m & 01s.048575 & \rightarrow & 00y & 000d 19h 05m 18s.428159 \\
21: & 00y & 000d & 00h & 00m & 02s.097151 & \rightarrow & 00y & 001d 14h 10m 36s.856319 \\
22: & 00y & 000d & 00h & 00m & 04s.194303 & \rightarrow & 00y & 003d 04h 21m 13s.712639 \\
23: & 00y & 000d & 00h & 00m & 08s.388607 & \rightarrow & 00y & 006d 08h 42m 27s.425279 \\
24: & 00y & 000d & 00h & 00m & 16s.777215 & \rightarrow & 00y & 012d 17h 24m 54s.850559 \\
25: & 00y & 000d & 00h & 00m & 33s.554431 & \rightarrow & 00y & 025d 10h 49m 49s.701119
\end{array}
\]

Note that the smallest value is also the incrementation step, so the timer precision.

";"

// -------- RULE ENTRY DEFINITION ------------

grouping tv-struct {
  description
  "Defines the target value element. Always a binary type, strings must be converted to binary. field-id allows the conversion to the appropriate type.";
  leaf value {
    type binary;
    description
    "Target Value";
  }
  leaf index {
    type uint16;
    description
    "Index gives the position in the matching-list. If only one element is present, index is 0. Otherwise, indicia is the order in the matching list, starting at 0.";
  }
}

grouping compression-rule-entry {
  description
  "These entries defines a compression entry (i.e. a line) as defined in RFC 8724.

<table>
<thead>
<tr>
<th>Field 1</th>
<th>FL</th>
<th>FP</th>
<th>DI</th>
<th>Target Value</th>
<th>Matching Operator</th>
<th>Comp/Decomp Act</th>
</tr>
</thead>
</table>

An entry in a compression rule is composed of 7 elements:
- Field ID: The header field to be compressed. The content is a YANG identifier.
- Field Length: either a positive integer of a function defined as a YANG id.
- Field Position: a positive (and possibly equal to 0) integer.
- Direction Indicator: a YANG identifier giving the direction.
- Target value: a value against which the header Field is compared.
- Matching Operator: a YANG id giving the operation, parameters may be associated to that operator.
- Comp./Decomp. Action: A YANG id giving the compression or decompression action, parameters may be associated to that action.

";
leave field-id {
  type schc:fid-type;
  mandatory true;
  description
    "Field ID, identify a field in the header with a YANG referenceid.";
}
leave field-length {
  type schc:fl-type;
  mandatory true;
  description
    "Field Length, expressed in number of bits or through a function defined as a YANG referenceid.";
}
leave 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.";
}
leave direction-indicator {
  type schc:di-type;
  mandatory true;
  description
"Direction Indicator, a YANG referenceid to say if the packet is bidirectional, up or down";
}
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, positions should take consecutive values starting from 1.";
}
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";
}
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;
  mandatory true;
  description
  "CDA: Compression Decompression Action.";
}
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."
}
}

grouping compression-content {
  list entry {
    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."
}

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;
    mandatory true;
    description
    "which fragmentation mode is used (noAck, AckAlways, AckonError)"
  }
  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
"Should be up or down, bidirectionnal is forbidden.";
}

// 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(../fragmentation-mode,
    'fragmentation-mode-ack-on-error')
  or
  derived-from(../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-RFC8724";
  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 at most
  max-interleaved-frames must be active at any time.";
}
container inactivity-timer {
  uses timer-duration;
  description
  "Duration is seconds of the inactivity timer, 0 indicates
  that the timer is disabled.";
}
container retransmission-timer {
  uses timer-duration;
  when "derived-from(../fragmentation-mode,
    'fragmentation-mode-ack-on-error')
  or
  derived-from(../fragmentation-mode,
    'fragmentation-mode-ack-always')
  ";
  description
  "Duration in seconds of the retransmission timer.";
}
leaf max-ack-requests {
  when "derived-from(../fragmentation-mode,
    'fragmentation-mode-ack-on-error')
  or
  derived-from(../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(../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-All1 {
  when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')"
  type schc:all1-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(../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."
}

// 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."
}
// SCHC table for a specific device.

container schc {
    list rule {
        key "rule-id-value rule-id-length";
        uses rule-id-type;
        choice nature {
            case fragmentation {
                if-feature "fragmentation";
                uses fragmentation-content;
            }
            case compression {
                if-feature "compression";
                uses compression-content;
            }
            case no-compression {
                description
                    "RFC8724 requires a rule for uncompressed headers.";
            }
            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.";
        }
    }
}

Figure 26

8. Normative References


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