lpwan Working Group
Internet-Draft
Updates: 8724 (if approved)
Intended status: Standards Track
Expires: 9 September 2023

S. Aguilar
C. Gomez
Universitat Politecnica de Catalunya

8 March 2023

SCHC Streaming Mode
draft-aguilar-lpwan-schc-streaming-00

Abstract

This documents presents an update of SCHC [RFC8724] by providing a new F/R mode called SCHC Streaming mode.

Status of This Memo

This Internet-Draft will expire on 9 September 2023.

Copyright Notice

Table of Contents

1. Introduction

SCHC [RFC8724] provides Fragmentation/Reassembly (F/R) modes, i.e., No-ACK, ACK-Always and ACK-on-Error. These modes allow for SCHC Packets larger than the Maximum Transmission Unit (MTU) of the underlying Layer 2 (L2) to be transferred between the sender and receiver with a range of reliability options, including SCHC Fragment retransmissions, over delay tolerant networks. The available F/R modes allow transmitting non-fragmented SCHC Packets concurrently with fragmented SCHC Fragments, and SCHC Packet interleaving. However, SCHC does not provide an optimal F/R mode for a continuous transmission of un-fragmented SCHC Packets, i.e, streaming of SCHC Packets smaller than, or of the same size as, the L2 MTU.

The streaming of SCHC Packets can be used to send, e.g., sensor measurements or the location coordinates of an asset tracker, which are sent every number of minutes and are optimized to fit in only one SCHC Fragment, with or without SCHC Compression. These SCHC Packets may not require fragmentation but require reliability, as some fragment losses may be incurred due to intermittent connectivity (e.g., vehicles going into tunnels, no coverage areas) or opportunistic coverage (e.g., coverage is available for certain time windows, of duration and frequency that might not be deterministic). With current SCHC F/R modes, each sensor measurement or location information can be sent as a compressed or un-compressed SCHC Packet, with different reliability options, however, each SCHC Packet will require a SCHC ACK, even if it is of only one SCHC Fragment in size. In networks, e.g., LPWANs [RFC8376], the downlink traffic or network capacity may be limited. [I.D.Compound ACK] provides an optimization in the ACK traffic by grouping the feedback of several windows of tiles in the same ACK message, providing flexibility on when the receiver sends feedback.

The present document extends [RFC8724] with a new F/R mode called SCHC Streaming. This F/R mode optimized the overhead of current F/R modes for a contiuos streaming of compressed or un-compressed SCHC Packets which require one SCHC Fragment to be transferred. The SCHC Streaming mode provides different configuration option on when the receiver can provide feedback, therefore adapting to the specifics of each network, e.g., the amount of ACK traffic that can be supported, application delay tolerance, L2 MTU size and the maximum number of window bitmaps that can be carried in a SCHC Compound ACK.

2. Terminology

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

It is assumed that the reader is familiar with the terms and mechanisms defined in [RFC8376] and in [RFC8724], specially Section 8.

3. SCHC Streaming

The SCHC Streaming mode supports L2 technologies that have variable MTU and out-of-order delivery (to some extent). It requires an L2 that provides a feedback path from the reassembler to the fragmenter.

SCHC Streaming mode uses windows, with all tiles, except for the last one, of equal size (regular size). The last tile MAY be smaller or equal to a regular tile.

A SCHC Fragment carries one or several contiguous tiles, which may span multiple windows from the same DTag value. A SCHC Compound ACK reports on the reception of one window of tiles or several windows of tiles, each one identified by its window number and corresponding to the same DTag value.

Each Profile, for each RuleID value, MUST define:

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

* the value of M,

* the value of N,

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

* the size and algorithm for the RCS field,

* the value of T,

* the value of MAX_ACK_REQUESTS,

* the expiration time of the Retransmission Timer,

* the expiration time of the Inactivity Timer,

* when the SCHC Compound ACKs are sent.

For each active RuleID value, the sender MUST maintain:

* one Attempts counter, and

* one Retransmission Timer.

For each active RuleID value, the receiver MUST maintain:

* one Inactivity Timer, and

* one Attempts counter.

3.1. Transfer Cycles

In SCHC Streaming mode the flow of tiles is continuous and it is divided into cycles. There are two cycles, the Window Cycle and the DTag Cycle (see Figure 1). To uniquely identify each tile, a combination of DTag, Window Number and FCN is used in each DTag Cycle.

```
              +--------------------------------------------...-----------....---
-...
              |                SCHC Streaming flow of SCHC Packets
              +--------------------------------------------...-----------...----
-...

Tile#         | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 | ... | 0 | 4 | ... | 0
 |...
Window#       |-------- 0 --------|-------- 1 --------|- 0  ...  1 -|- 0  ...  1
-|...
DTag#         |------------------- 0 ---------------|----- 1 -----|----- 0 ----
-|...

Window Cycle# |--------------------0----------------|------1------|------0-----
-|...
DTag Cycle#   |------------------------------0-------------------|------1-----
-|...
              ^                                       ^             ^
              |                                       |             |
             Window Cycle 0 start          Window Cycle 0 ends      |
              ^                                                     |
              |                                                     |
             DTag Cycle 0 start                          DTag Cycle 0 ends
```

Figure 1: SCHC Packets streaming carried in Tiles and Windows using the SCHC Streaming Mode. M = 1 bit (Window Cycle of 2 Windows) and DTag = 1 bit (DTag Cycle of 2 Window Cycles, i.e., 4 Windows)

The sender will begin the first DTag and Window Cycle by sending tiles using DTag = 0 and Window Number = 0 (the tile index, i.e., the FCN, MUST be decremented by 1 from WINDOW_SIZE - 1 downward). After each window of tiles, the Window Number is increased. Current Window Cycle ends once the Window Number reaches its maximum value and the last fragment of this window is sent. Next Window Cycle will begin by increasing the DTag value by one, and resetting the Window Number and FCN values. The number of Window Cycles without repeating the same DTag, Window Number and FCN value depends on the size of the DTag field, which determines the DTag Cycle. After the DTag reaches its maximum value, and therefore the end of the DTag Cycle, it MUST be reset. To manage the receiver feedback, the Receiver MUST send at least one SCHC Compound ACK per DTag Cycle, i.e., before the DTag is reset, indicating tiles losses in any of previous Window Cycles corresponding to this DTag Cycle. Only one Window Cycle MUST be reported per SCHC Compound ACK. The SCHC Compound ACK MUST be sent before the start of a new DTag Cycle. SCHC Fragments MAY be delivered out-of-order in each DTag Cycle, but all tiles MUST be received before advancing to the next DTag Cycle.

The SCHC All-1 message is used to finalize current SCHC Streaming session in case it is needed.

3.2. ACK Behaviour

A SCHC Compound ACK MAY be sent after the All-0 SCHC Fragment message and MUST be sent after the All-1 SCHC Fragment message. This allows the receiver to provide feedback after any window of tiles. The Profile MUST specify when the sender should listen for a SCHC Compound ACK, specially in networks which require the sender to enable reception of incoming SCHC ACKs. The sender MAY listen after each complete window of tiles (the All-0 message in each window), after the All-0 of the last window of each Window Cycle or after the All-0 of the last window of each DTag Cycle.

The receiver can send SCHC Compound ACKs:

* at the end of each Window Cycle, in the last window (with the maximum window number), an All-0 message indicates the end of current window, and as it is the last window of current Window Cycle, it indicates the end of current Window Cycle. The receiver MAY send a SCHC Compound ACK. Note that after this Window Cycle ends, the receiver MAY request fragments of previous DTag values (before the DTag Cycle ends).

* A success SCHC ACK MUST be sent by the receiver at the end of each DTag Cycle, to acknowledge all SCHC Fragment before continuing to next DTag Cycle. Note that after a new DTag Cycle begins, it is not possible to recover SCHC Fragment from previous DTag Cycles, as the combination of DTag, Window Number and FCN is repeated.

4. SCHC Streaming mode examples

This section provides examples of the SCHC Streaming mode. The configuration used in these examples is as follows:

* RuleID: Same RuleID in all SCHC Fragments.

* M: 2 bits (with values 00,01,10,11)

* N: 3 bits (with values from 6 to 0 plus the All-1)

* DTag: 1 bits

* WINDOW_SIZE: 7 tiles

In Figure 2, a SCHC Streaming transmission example is shown. In this transmission, the first 3 windows have fragment losses. The fourth window has no fragment losses. The receiver sends a SCHC Compound ACK reporting on the fragment losses of the first 3 windows, after receiving the All-0 message that signal the end of current Window

Cycle, i.e., the All-0 message of the fourth window. The sender resends the missing fragments and continues to next Window Cycle by increasing the DTag value.

Next Window Cycle present fragment losses that are recovered at the end of the cycle, as the receiver sends a SCHC Compound ACK message after receiving the All-0 message. The sender resends the missing fragment, and as it is the end of the DTag Cycle, a success ACK is sent by the receiver to continue the transmission in the next DTag Cycle.

```
  Sender                             Receiver
   |-----DTag= 0, W=0, FCN=6 ----->|
   |-----DTag= 0, W=0, FCN=5 ----->|
   |-----DTag= 0, W=0, FCN=4 ----->|
   |-----DTag= 0, W=0, FCN=3 ----->|
   |-----DTag= 0, W=0, FCN=2 --X   |
   |-----DTag= 0, W=0, FCN=1 ----->|
   |-----DTag= 0, W=0, FCN=0 ----->| Bitmap: 1111011
(no ACK)
   |-----DTag= 0, W=1, FCN=6 ----->|
   |-----DTag= 0, W=1, FCN=5 ----->|
   |-----DTag= 0, W=1, FCN=4 ----->|
   |-----DTag= 0, W=1, FCN=3 ----->|
   |-----DTag= 0, W=1, FCN=2 ----->|
   |-----DTag= 0, W=1, FCN=1 --X   |
   |-----DTag= 0, W=1, FCN=0 ----->| Bitmap: 1111101
(no ACK)
   |-----DTag= 0, W=2, FCN=6 ----->|
   |-----DTag= 0, W=2, FCN=5 --X   |
   |-----DTag= 0, W=2, FCN=4 ----->|
   |-----DTag= 0, W=2, FCN=3 ----->|
   |-----DTag= 0, W=2, FCN=2 ----->|
   |-----DTag= 0, W=2, FCN=1 ----->|
   |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
(no ACK)
   |-----DTag= 0, W=3, FCN=6 ----->|
   |-----DTag= 0, W=3, FCN=5 ----->|
   |-----DTag= 0, W=3, FCN=4 ----->|
   |-----DTag= 0, W=3, FCN=3 ----->|
   |-----DTag= 0, W=3, FCN=2 ----->|
   |-----DTag= 0, W=3, FCN=1 ----->|
   |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1111111
   |<--- DTag= 0, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:1111
101, W=2 - Bitmap:1011111]
   |-----DTag= 0, W=0, FCN=2 ----->|
   |-----DTag= 0, W=1, FCN=1 ----->|
   |-----DTag= 0, W=2, FCN=5 ----->|
(next Window Cycle)
```

```
   |-----DTag= 1, W=0, FCN=6 ----->|
   |-----DTag= 1, W=0, FCN=5 ----->|
   |-----DTag= 1, W=0, FCN=4 ----->|
   |-----DTag= 1, W=0, FCN=3 ----->|
   |-----DTag= 1, W=0, FCN=2 --X   |
   |-----DTag= 1, W=0, FCN=1 ----->|
   |-----DTag= 1, W=0, FCN=0 ----->| Bitmap: 1111011
(no ACK)
   |-----DTag= 1, W=1, FCN=6 ----->|
   |-----DTag= 1, W=1, FCN=5 ----->|
   |-----DTag= 1, W=1, FCN=4 ----->|
   |-----DTag= 1, W=1, FCN=3 ----->|
   |-----DTag= 1, W=1, FCN=2 ----->|
   |-----DTag= 1, W=1, FCN=1 --X   |
   |-----DTag= 1, W=1, FCN=0 ----->| Bitmap: 1111101
(no ACK)
   |-----DTag= 1, W=2, FCN=6 ----->|
   |-----DTag= 1, W=2, FCN=5 --X   |
   |-----DTag= 1, W=2, FCN=4 ----->|
   |-----DTag= 1, W=2, FCN=3 ----->|
   |-----DTag= 1, W=2, FCN=2 ----->|
   |-----DTag= 1, W=2, FCN=1 ----->|
   |-----DTag= 1, W=2, FCN=0 ----->| Bitmap: 1011111
(no ACK)
   |-----DTag= 1, W=3, FCN=6 ----->|
   |-----DTag= 1, W=3, FCN=5 ----->|
   |-----DTag= 1, W=3, FCN=4 ----->|
   |-----DTag= 1, W=3, FCN=3 ----->|
   |-----DTag= 1, W=3, FCN=2 ----->|
   |-----DTag= 1, W=3, FCN=1 ----->|
   |-----DTag= 1, W=3, FCN=0 ----->| Bitmap: 1011111
   |<--- DTag= 1, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:1111
101, W=2 - Bitmap:1011111]
   |-----DTag= 1, W=0, FCN=2 ----->|
   |-----DTag= 1, W=1, FCN=1 ----->|
   |-----DTag= 1, W=2, FCN=5 ----->|
   |<--- DTag= 1, ACK, W=3, C=1 ---| C=1 [success ACK is needed before moving to n
ext DTag cycle]
(next Window and DTag Cycle)
```

Figure 2: SCHC Streaming mode sequence example 1

Figure 3 shows another example of SCHC Streaming mode where a SCHC Compound ACK is sent at the ending of the DTag Cycle, recovering SCHC Fragment losses of previous windows of the DTag Cycle. As both Window Cycles present SCHC Fragment losses, two SCHC Compound ACKs are sent by the receiver at the end of the DTag Cycle.

```
    Sender                             Receiver
     |-----DTag= 0, W=0, FCN=6 ----->|
     |-----DTag= 0, W=0, FCN=5 ----->|
     |-----DTag= 0, W=0, FCN=4 ----->|
     |-----DTag= 0, W=0, FCN=3 ----->|
     |-----DTag= 0, W=0, FCN=2 --X   |
     |-----DTag= 0, W=0, FCN=1 ----->|
     |-----DTag= 0, W=0, FCN=0 ----->| Bitmap: 1111011
(no ACK)
     |-----DTag= 0, W=1, FCN=6 ----->|
     |-----DTag= 0, W=1, FCN=5 ----->|
     |-----DTag= 0, W=1, FCN=4 ----->|
     |-----DTag= 0, W=1, FCN=3 ----->|
     |-----DTag= 0, W=1, FCN=2 ----->|
     |-----DTag= 0, W=1, FCN=1 --X   |
     |-----DTag= 0, W=1, FCN=0 ----->| Bitmap: 1111101
(no ACK)
     |-----DTag= 0, W=2, FCN=6 ----->|
     |-----DTag= 0, W=2, FCN=5 --X   |
     |-----DTag= 0, W=2, FCN=4 ----->|
     |-----DTag= 0, W=2, FCN=3 ----->|
     |-----DTag= 0, W=2, FCN=2 ----->|
     |-----DTag= 0, W=2, FCN=1 ----->|
     |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
(no ACK)
     |-----DTag= 0, W=3, FCN=6 ----->|
     |-----DTag= 0, W=3, FCN=5 ----->|
     |-----DTag= 0, W=3, FCN=4 ----->|
     |-----DTag= 0, W=3, FCN=3 ----->|
     |-----DTag= 0, W=3, FCN=2 ----->|
     |-----DTag= 0, W=3, FCN=1 ----->|
     |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
(no ACK)

(next Window Cycle)
     |-----DTag= 1, W=0, FCN=6 ----->|
     |-----DTag= 1, W=0, FCN=5 ----->|
     |-----DTag= 1, W=0, FCN=4 ----->|
     |-----DTag= 1, W=0, FCN=3 ----->|
     |-----DTag= 1, W=0, FCN=2 --X   |
     |-----DTag= 1, W=0, FCN=1 ----->|
     |-----DTag= 1, W=0, FCN=0 ----->| Bitmap: 1111011
(no ACK)
     |-----DTag= 1, W=1, FCN=6 ----->|
     |-----DTag= 1, W=1, FCN=5 ----->|
     |-----DTag= 1, W=1, FCN=4 ----->|
     |-----DTag= 1, W=1, FCN=3 ----->|
     |-----DTag= 1, W=1, FCN=2 ----->|
```

```
     |-----DTag= 1, W=1, FCN=1 --X      |
     |-----DTag= 1, W=1, FCN=0 ----->| Bitmap: 1111101
(no ACK)
     |-----DTag= 1, W=2, FCN=6 ----->|
     |-----DTag= 1, W=2, FCN=5 --X      |
     |-----DTag= 1, W=2, FCN=4 ----->|
     |-----DTag= 1, W=2, FCN=3 ----->|
     |-----DTag= 1, W=2, FCN=2 ----->|
     |-----DTag= 1, W=2, FCN=1 ----->|
     |-----DTag= 1, W=2, FCN=0 ----->| Bitmap: 1011111
(no ACK)
     |-----DTag= 1, W=3, FCN=6 ----->|
     |-----DTag= 1, W=3, FCN=5 ----->|
     |-----DTag= 1, W=3, FCN=4 ----->|
     |-----DTag= 1, W=3, FCN=3 ----->|
     |-----DTag= 1, W=3, FCN=2 ----->|
     |-----DTag= 1, W=3, FCN=1 ----->|
     |-----DTag= 1, W=3, FCN=0 ----->| Bitmap: 1111111

     |<--- DTag= 0, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:11
11101, W=2 - Bitmap:1011111]
     |-----DTag= 0, W=0, FCN=2 ----->|
     |-----DTag= 0, W=1, FCN=1 ----->|
     |-----DTag= 0, W=2, FCN=5 ----->|

     |<--- DTag= 1, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:11
11101, W=2 - Bitmap:1011111]
     |-----DTag= 1, W=0, FCN=2 ----->|
     |-----DTag= 1, W=1, FCN=1 ----->|
     |-----DTag= 1, W=2, FCN=5 ----->|

     |<--- DTag= 1, ACK, W=3, C=1 ---| C=1 success ACK is needed before moving to
next DTag cycle
(next Window and DTag Cycle)
```

Figure 3: SCHC Streaming mode sequence example 2

Figure 4 presents a SCHC Streaming transmission that is closed by the sender using an All-1 message. After the All-1 message, the receiver sends a SCHC Compound ACKs for missing fragments. The sender resends missing fragments and waits for a success SCHC ACK indicating that all SCHC Fragments were correctly received and that current SCHC Streaming transmission can be closed.

```
      Sender                               Receiver
       |-----DTag= 0, W=0, FCN=6 ----->|
       |-----DTag= 0, W=0, FCN=5 ----->|
       |-----DTag= 0, W=0, FCN=4 ----->|
       |-----DTag= 0, W=0, FCN=3 ----->|
       |-----DTag= 0, W=0, FCN=2 --X   |
       |-----DTag= 0, W=0, FCN=1 ----->|
       |-----DTag= 0, W=0, FCN=0 ----->| Bitmap: 1111011
   (no ACK)
       |-----DTag= 0, W=1, FCN=6 ----->|
       |-----DTag= 0, W=1, FCN=5 ----->|
       |-----DTag= 0, W=1, FCN=4 ----->|
       |-----DTag= 0, W=1, FCN=3 ----->|
       |-----DTag= 0, W=1, FCN=2 ----->|
       |-----DTag= 0, W=1, FCN=1 --X   |
       |-----DTag= 0, W=1, FCN=0 ----->| Bitmap: 1111101
   (no ACK)
       |-----DTag= 0, W=2, FCN=6 ----->|
       |-----DTag= 0, W=2, FCN=5 --X   |
       |-----DTag= 0, W=2, FCN=4 ----->|
       |-----DTag= 0, W=2, FCN=3 ----->|
       |-----DTag= 0, W=2, FCN=2 ----->|
       |-----DTag= 0, W=2, FCN=1 ----->|
       |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)
       |-----DTag= 0, W=3, FCN=6 ----->|
       |-----DTag= 0, W=3, FCN=5 ----->|
       |-----DTag= 0, W=3, FCN=4 ----->|
       |-----DTag= 0, W=3, FCN=3 ----->|
       |-----DTag= 0, W=3, FCN=2 ----->|
       |-----DTag= 0, W=3, FCN=1 ----->|
       |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)

   (next Window Cycle)
       |-----DTag= 1, W=0, FCN=6 ----->|
       |-----DTag= 1, W=0, FCN=5 ----->|
       |-----DTag= 1, W=0, FCN=4 ----->|
       |-----DTag= 1, W=0, FCN=3 ----->|
       |-----DTag= 1, W=0, FCN=2 --X   |
       |-----DTag= 1, W=0, FCN=1 ----->|
       |-----DTag= 1, W=0, FCN=0 ----->| Bitmap: 1111011
   (no ACK)
       |-----DTag= 1, W=1, FCN=6 ----->|
       |-----DTag= 1, W=1, FCN=5 ----->|
       |-----DTag= 1, W=1, FCN=4 ----->|
       |-----DTag= 1, W=1, FCN=3 ----->|
       |-----DTag= 1, W=1, FCN=2 ----->|
```

```
       |-----DTag= 1, W=1, FCN=1 --X    |
       |-----DTag= 1, W=1, FCN=0 ----->| Bitmap: 1111101
   (no ACK)
       |-----DTag= 1, W=2, FCN=6 ----->|
       |-----DTag= 1, W=2, FCN=5 --X    |
       |-----DTag= 1, W=2, FCN=4 ----->|
       |-----DTag= 1, W=2, FCN=3 ----->|
       |-----DTag= 1, W=2, FCN=2 ----->|
       |-----DTag= 1, W=2, FCN=1 ----->|
       |-----DTag= 1, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)
       |-----DTag= 1, W=3, FCN=6 ----->|
       |--DTag= 1, W=3, FCN=7, RCS --->| All-1, Bitmap: 1011111

       |<--- DTag= 0, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:
1111101, W=2 - Bitmap:1011111]
       |-----DTag= 0, W=0, FCN=2 ----->|
       |-----DTag= 0, W=1, FCN=1 ----->|
       |-----DTag= 0, W=2, FCN=5 ----->|

       |<--- DTag= 1, Compound ACK ----| [C=0, W=0 - Bitmap:1111011, W=1 - Bitmap:
1111101, W=2 - Bitmap:1011111]
       |-----DTag= 1, W=0, FCN=2 ----->|
       |-----DTag= 1, W=1, FCN=1 ----->|
       |-----DTag= 1, W=2, FCN=5 ----->|

       |<--- DTag= 1, ACK, W=3, C=1 ---| C=1
```

Figure 4: SCHC Streaming mode sequence example 3 - Closed by sender

Figure 5 shows a SCHC Streaming example where the receiver aborts current transmission.

```
     Sender                          Receiver
      |-----DTag= 0, W=0, FCN=6 ----->|
      |-----DTag= 0, W=0, FCN=5 ----->|
      |-----DTag= 0, W=0, FCN=4 ----->|
      |-----DTag= 0, W=0, FCN=3 ----->|
      |-----DTag= 0, W=0, FCN=2 --X    |
      |-----DTag= 0, W=0, FCN=1 ----->|
      |-----DTag= 0, W=0, FCN=0 ----->| Bitmap: 1111011
    (no ACK)
      |-----DTag= 0, W=1, FCN=6 ----->|
      |-----DTag= 0, W=1, FCN=5 ----->|
      |-----DTag= 0, W=1, FCN=4 ----->|
      |-----DTag= 0, W=1, FCN=3 ----->|
      |-----DTag= 0, W=1, FCN=2 ----->|
      |-----DTag= 0, W=1, FCN=1 --X    |
      |-----DTag= 0, W=1, FCN=0 ----->| Bitmap: 1111101
```

```
   (no ACK)
       |-----DTag= 0, W=2, FCN=6 ----->|
       |-----DTag= 0, W=2, FCN=5 --X   |
       |-----DTag= 0, W=2, FCN=4 ----->|
       |-----DTag= 0, W=2, FCN=3 ----->|
       |-----DTag= 0, W=2, FCN=2 ----->|
       |-----DTag= 0, W=2, FCN=1 ----->|
       |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)
       |-----DTag= 0, W=3, FCN=6 ----->|
       |-----DTag= 0, W=3, FCN=5 ----->|
       |-----DTag= 0, W=3, FCN=4 ----->|
       |-----DTag= 0, W=3, FCN=3 ----->|
       |-----DTag= 0, W=3, FCN=2 ----->|
       |-----DTag= 0, W=3, FCN=1 ----->|
       |-----DTag= 0, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)

   (next Window Cycle)
       |-----DTag= 1, W=0, FCN=6 ----->|
       |-----DTag= 1, W=0, FCN=5 ----->|
       |-----DTag= 1, W=0, FCN=4 ----->|
       |-----DTag= 1, W=0, FCN=3 ----->|
       |-----DTag= 1, W=0, FCN=2 --X   |
       |-----DTag= 1, W=0, FCN=1 ----->|
       |-----DTag= 1, W=0, FCN=0 ----->| Bitmap: 1111011
   (no ACK)
       |-----DTag= 1, W=1, FCN=6 ----->|
       |-----DTag= 1, W=1, FCN=5 ----->|
       |-----DTag= 1, W=1, FCN=4 ----->|
       |-----DTag= 1, W=1, FCN=3 ----->|
       |-----DTag= 1, W=1, FCN=2 ----->|
       |-----DTag= 1, W=1, FCN=1 --X   |
       |-----DTag= 1, W=1, FCN=0 ----->| Bitmap: 1111101
   (no ACK)
       |-----DTag= 1, W=2, FCN=6 ----->|
       |-----DTag= 1, W=2, FCN=5 --X   |
       |-----DTag= 1, W=2, FCN=4 ----->|
       |-----DTag= 1, W=2, FCN=3 ----->|
       |-----DTag= 1, W=2, FCN=2 ----->|
       |-----DTag= 1, W=2, FCN=1 ----->|
       |-----DTag= 1, W=2, FCN=0 ----->| Bitmap: 1011111
   (no ACK)
       |<--------- RECV ABORT ---------|
```

Figure 5: SCHC Streaming mode sequence example 4 - Aborted by receiver

5. SCHC Streaming mode YANG Data Model

The present document also extends the SCHC YANG data model defined in [RFC9363] by including a new identity in the fragmentation mode type.

5.1. SCHC YANG Data Model Extension

TBD

5.2. SCHC YANG Tree Extension

TBD

6. Security considerations

TBD

7. IANA Considerations

This document has no IANA actions.

8. 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).

9. References

9.1. Normative References

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

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

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

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

9.2. Informative References

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

Authors' Addresses

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

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

LPWAN Working Group
Internet-Draft
Updates: 8824 (if approved)
Intended status: Standards Track
Expires: 14 September 2023

M. Tiloca
RISE AB
L. Toutain
I. Martinez
IMT Atlantique

13 March 2023

Clarifications and Updates on using Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)

draft-tiloca-lpwan-8824-update-00

Abstract

This document clarifies, updates and extends the method specified in RFC 8824 for compressing Constrained Application Protocol (CoAP) headers using the Static Context Header Compression and fragmentation (SCHC) framework. In particular, it considers recently defined CoAP options and specifies how CoAP headers are compressed in the presence of intermediaries. Therefore, this document updates RFC 8824.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the IPv6 over Low Power Wide-Area Networks Working Group mailing list (lp-wan@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/lp-wan/.

Source for this draft and an issue tracker can be found at https://github.com/git@gitlab.com:crimson84/draft-tiloca-lpwan-8824-update.

Status of This Memo

This Internet-Draft will expire on 14 September 2023.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web-transfer protocol intended for applications based on the REST (Representational State Transfer) paradigm, and designed to be affordable also for resource-constrained devices.

In order to enable the use of CoAP in LPWANs (Low-Power Wide-Area Networks) as well as to improve performance, [RFC8824] defines how to use the Static Context Header Compression and fragmentation (SCHC) framework [RFC8724] for compressing CoAP headers.

This document clarifies, updates and extends the SCHC compression of CoAP headers defined in [RFC8824] at the application level, by: providing specific clarifications; updating specific details of the compression processing, based on recent developments related to the security protocol OSCORE [RFC8613] for end-to-end protection of CoAP messages; and extending the compression processing to take into account additional CoAP options and the presence of CoAP proxies.

In particular, this document updates [RFC8824] as follows.

* It clarifies the SCHC compression for the CoAP options Size1, Size2, Proxy-URI and Proxy-Scheme (see Section 2.1).

* It defines the SCHC compression for the CoAP option Hop-Limit (see Section 2.2).

* It defines the SCHC compression for the recently defined CoAP options Echo (see Section 2.3), Request-Tag (see Section 2.4), EDHOC (see Section 2.5), as well as Q-Block1 and Q-Block2 (see Section 3.1).

* It updates the SCHC compression processing for the CoAP option OSCORE (see Section 3.2), in the light of recent developments related to the security protocol OSCORE as defined in [I-D.ietf-core-oscore-key-update] and [I-D.ietf-core-oscore-groupcomm].

* It clarifies how the SCHC compression handles the CoAP payload marker (see Section 4).

* It defines the SCHC compression of CoAP headers in the presence of CoAP proxies (see Section 5).

This document does not alter the core approach, design choices and features of the SCHC compression applied to CoAP headers.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts related to the SCHC framework [RFC8724], the web-transfer protocol CoAP [RFC7252], the security protocol OSCORE [RFC8613] and the use of SCHC for CoAP [RFC8824].

2. CoAP Options

This section updates and extends Section 5 of [RFC8824], as to how SCHC compresses some specific CoAP options. In particular, Section 2.1 updates Section 5.4 of [RFC8824].

2.1. CoAP Option Size1, Size2, Proxy-URI, and Proxy-Scheme Fields

The SCHC Rule description MAY define sending some field values by describing an empty TV, with the MO set to "ignore" and the CDA set to "value-sent". A Rule MAY also use a "match-mapping" MO when there are different options for the same FID. Otherwise, the Rule sets the TV to the value, the MO to "equal", and the CDA to "not-sent".

2.2. CoAP Option Hop-Limit Field

The Hop-Limit field is an option defined in [RFC8768] that can be used to detect forwarding loops through a chain of CoAP proxies. The first proxy in the chain that understands the option includes it in a received request with a proper value set, before forwarding the request. Any following proxy that understands the option decrements the option value and forwards the request if the new value is different than zero, or returns a 5.08 (Hop Limit Reached) error response otherwise.

When a packet uses the Hop-Limit option, SCHC compression MUST send its content in the Compression Residue. The SCHC Rule describes an empty TV with the MO set to "ignore" and the CDA set to "value-sent".

2.3. CoAP Option Echo Field

The Echo field is an option defined in [RFC9175] that a server can include in a response as a challenge to the client, and that the client echoes back to the server in one or more requests. This enables the server to verify the freshness of a request and to cryptographically verify the aliveness of the client. Also, it forces the client to demonstrate reachability at its claimed network address.

When a packet uses the Echo option, SCHC compression MUST send its content in the Compression Residue. The SCHC Rule describes an empty TV with the MO set to "ignore" and the CDA set to "value-sent".

2.4. CoAP Option Request-Tag Field

The Request-Tag field is an option defined in [RFC9175] that the client can set in request messages of block-wise operations, with value an ephemeral short-lived identifier of the specific block-wise operation in question. This allows the server to match message fragments belonging to the same request operation and, if the server supports it, to reliably process simultaneous block-wise request operations on a single resource. If requests are integrity protected, this also protects against interchange of fragments between different block-wise request operations.

When a packet uses the Request-Tag option, SCHC compression MUST send its content in the Compression Residue. The SCHC Rule describes an empty TV with the MO set to "ignore" and the CDA set to "value-sent".

2.5. CoAP Option EDHOC Field

The EDHOC field is an option defined in [I-D.ietf-core-oscore-edhoc] that a client can include in a request, in order to perform an optimized, shortened execution of the authenticated key establishment protocol EDHOC [I-D.ietf-lake-edhoc]. Such a request conveys both the final EDHOC message and actual application data, where the latter is protected with OSCORE [RFC8613] using a Security Context derived from the result of the current EDHOC execution.

The option occurs at most once and is always empty. The SCHC Rule MUST describe an empty TV, with the MO set to "equal" and the CDA set to "not-sent".

3. SCHC Compression of CoAP Extensions

This section updates and extends Section 6 of [RFC8824], as to how SCHC compresses some specific CoAP options providing protocol extensions. In particular, Section 3.1 updates Section 6.1 of [RFC8824], while Section 3.2 updates Section 6.4 of [RFC8824].

3.1. Block

When a packet uses a Block1 or Block2 option [RFC7959] or a Q-Block1 or Q-Block2 option [RFC9177], SCHC compression MUST send its content in the Compression Residue. The SCHC Rule describes an empty TV with the MO set to "ignore" and the CDA set to "value-sent". The Block1, Block2, Q-Block1 and Q-Block2 options allow fragmentation at the CoAP level that is compatible with SCHC fragmentation. Both fragmentation mechanisms are complementary, and the node may use them for the same packet as needed.

3.2. OSCORE

The security protocol OSCORE [RFC8613] provides end-to-end protection for CoAP messages. Group OSCORE [I-D.ietf-core-oscore-groupcomm] builds on OSCORE and defines end-to-end protection of CoAP messages in group communication [I-D.ietf-core-groupcomm-bis]. This section describes how SCHC Rules can be applied to compress messages protected with OSCORE or Group OSCORE.

Figure 1 shows the OSCORE option value encoding, which was originally defined in Section 6.1 of [RFC8613] and has been extended in [I-D.ietf-core-oscore-key-update][I-D.ietf-core-oscore-groupcomm]. The first byte of the OSCORE option value specifies the content of the OSCORE option using flags, as follows.

* As defined in Section 4.1 of [I-D.ietf-core-oscore-key-update], the eight least significant bit, when set, indicates that the OSCORE option includes a second byte of flags. The seventh least significant bit is currently unassigned.

* As defined in Section 5 of [I-D.ietf-core-oscore-groupcomm], the sixth least significant bit, when set, indicates that the message including the OSCORE option is protected with the group mode of Group OSCORE (see Section 8 of [I-D.ietf-core-oscore-groupcomm]). When not set, the bit indicates that the message is protected either with OSCORE, or with the pairwise mode of Group OSCORE (see Section 9 of [I-D.ietf-core-oscore-groupcomm]), while the specific OSCORE Security Context used to protect the message determines which of the two cases applies.

* As defined in Section 6.1 of [RFC8613], bit h, when set, indicates the presence of the kid context field in the option. Also, bit k, when set, indicates the presence of a kid field. Finally, the three least significant bits form the field n, which indicates the length of the piv (Partial Initialization Vector) field in bytes. When n = 0, no piv is present.

Assuming the presence of a single flag byte, this is followed by the piv field, the kid context field, and the kid field, in that order. Also, if present, the kid context field’s length (in bytes) is encoded in the first byte, denoted by "s".

```
 0 1 2 3 4 5 6 7 <--------- n bytes ------------->
+-+-+-+-+-+-+-+-+---------------------------------+
|0 0 0|h|k|  n  |       Partial IV (if any)       |
+-+-+-+-+-+-+-+-+---------------------------------+
|               |                                 |
|<--   CoAP  -->|<------- CoAP OSCORE_piv ------> |
   OSCORE_flags

 <-- 1 byte --> <------ s bytes ----->
+--------------+----------------------+-----------------------+
|  s (if any)  | kid context (if any) | kid (if any)      ... |
+--------------+----------------------+-----------------------+
|                                     |                       |
|<-------- CoAP OSCORE_kidctx ------->|<-- CoAP OSCORE_kid -->|
```

Figure 1: OSCORE Option

Figure 2 shows the OSCORE option value encoding, with the second byte of flags also present. As defined in Section 4.1 of [I-D.ietf-core-oscore-key-update], the least significant bit d of this byte, when set, indicates that two additional fields are included in the option, following the kid context field (if any).

These two fields, namely x and nonce, are used when running the key update protocol KUDOS defined in [I-D.ietf-core-oscore-key-update], with x specifying the length of the nonce field in bytes as well as the specific behavior to adopt during the KUDOS execution. In particular, the figure provides the breakdown of the x field, where its three least significant bits form the sub-field m, which specifies the size of nonce in bytes, minus 1.

```
     0 1 2 3 4 5 6 7  8   9   10  11  12  13  14  15 <----- n bytes ----->
    +-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
    |1|0|0|h|k|  n  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | d | Partial IV (if any) |
    +-+-+-+-+-+-+-+-+---+---+---+---+---+---+---+---+---------------------+
    |                                               |                     |
    |<-------------------- CoAP -------------------->|<- CoAP OSCORE_piv ->|
                       OSCORE_flags

     <- 1 byte -> <----------- s bytes ------------> <------ 1 byte ----->
    +------------+----------------------------------+---------------------+
    | s (if any) |       kid context (if any)       |     x (if any)      |
    +------------+----------------------------------+---------------------+
    |                                               |                     |
    |<------------- CoAP OSCORE_kidctx ------------>|<-- CoAP OSCORE_x -->|
                                                    |                     |
                                                    |   0 1 2 3 4 5 6 7   |
                                                    |  +-+-+-+-+-+-+-+-+  |
                                                    |  |0|0|b|p|   m   |  |
                                                    |  +-+-+-+-+-+-+-+-+  |

     <----- m + 1 bytes ----->
    +-------------------------+-----------------------+
    |      nonce (if any)     |    kid (if any) ...   |
    +-------------------------+-----------------------+
    |                         |                       |
    |<-- CoAP OSCORE_nonce -->|<-- CoAP OSCORE_kid -->|
```

Figure 2: OSCORE Option during a KUDOS execution

To better perform OSCORE SCHC compression, the Rule description needs to identify the OSCORE option and the fields it contains. Conceptually, it discerns up to six distinct pieces of information within the OSCORE option: the flag bits, the piv, the kid context, the x byte, the nonce, and the kid. The SCHC Rule splits the OSCORE option into six Field Descriptors in order to compress them:

* CoAP OSCORE_flags

* CoAP OSCORE_piv

* CoAP OSCORE_kidctx

* CoAP OSCORE_x

* CoAP OSCORE_nonce

* CoAP OSCORE_kid

Figure 1 shows the OSCORE option format with the four fields OSCORE_flags, OSCORE_piv, OSCORE_kidctx and OSCORE_kid superimposed on it. Also, Figure 2 shows the OSCORE option format with all the six fields superimposed on it, with reference to a message exchanged during an execution of the KUDOS key update protocol.

In both cases, the CoAP OSCORE_kidctx field directly includes the size octet, s. In the latter case, the following applies.

* For the x field, if both endpoints know the value, then the SCHC Rule will describe a TV to this value, with the MO set to "equal" and the CDA set to "not-sent". This models the case where the two endpoints run KUDOS with a pre-agreed size of the nonce field, as well as with a pre-agreed combination of its modes of operations, as per the bits b and p of the m sub-field.

  Otherwise, if the value is changing over time, the SCHC Rule will set the MO to "ignore" and the CDA to "value-sent". The Rule may also use a "match-mapping" MO to compress this field, in case the two endpoints pre-agree on a set of alternative ways to run KUDOS, with respect to the size of the nonce field and the combination of the KUDOS modes of operation to use.

* For the nonce field, the SCHC Rule describes an empty TV with the MO set to "ignore" and the CDA set to "value-sent".

  In addition, for the value of the nonce field, SCHC MUST NOT send it as variable-length data in the Compression Residue, to avoid ambiguity with the length of the nonce field encoded in the x field. Therefore, SCHC MUST use the m sub-field of the x field to define the size of the Compression Residue. SCHC designates a specific function, "osc.x.m", that the Rule MUST use to complete the Field Descriptor. During the decompression, this function returns the length of the nonce field in bytes, as the value of the three least significant bits of the m sub-field of the x field, plus 1.

4. Compression of the CoAP Payload Marker

As originally intended in [RFC8824], the following applies with respect to the 0xFF payload marker. A SCHC compression rule for CoAP includes all the expected CoAP options, therefore the payload marker does not have to be specified.

4.1. Without End-to-End Security

If the CoAP message to compress with SCHC is not going to be protected with OSCORE and includes a payload, then the 0xFF payload marker MUST NOT be included in the compressed message, which is composed of the Compression RuleID, the Compression Residue (if any), and the CoAP payload.

After having decompressed an incoming message, the recipient endpoint MUST prepend the 0xFF payload marker to the CoAP payload, if any was present after the consumed Compression Residue.

4.2. With End-to-End Security

If the CoAP message has to be protected with OSCORE, the same rationale described in Section 4.1 applies to both the Inner SCHC Compression and the Outer SCHC Compression defined in Section 7.2 of [RFC8824]. That is:

* After the Inner SCHC Compression of a CoAP message including a payload, the payload marker MUST NOT be included in the input to the AEAD Encryption, which is composed of the Inner Compression RuleID, the Inner Compression Residue (if any), and the CoAP payload.

* The Outer SCHC Compression takes as input the OSCORE-protected message, which always includes a payload (i.e., the OSCORE Ciphertext) preceded by the payload marker.

* After the Outer SCHC Compression, the payload marker MUST NOT be included in the final compressed message, which is composed of the Outer Compression RuleID, the Outer Compression Residue (if any), and the OSCORE Ciphertext.

After having completed the Outer SCHC Decompression of an incoming message, the recipient endpoint MUST prepend the 0xFF payload marker to the OSCORE Ciphertext.

After having completed the Inner SCHC Decompression of an incoming message, the recipient endpoint MUST prepend the 0xFF payload marker to the CoAP payload, if any was present after the consumed Compression Residue.

5. CoAP Header Compression with Proxies

Building on [RFC8824], this section clarifies how SCHC Compression/Decompression is performed when CoAP proxies are deployed. The following refers to the origin client and origin server as application endpoints.

5.1. Without End-to-End Security

In case OSCORE is not used end-to-end between client and server, the SCHC processing occurs hop-by-hop, by relying on SCHC Rules that are consistently shared between two adjacent hops.

In particular, SCHC is used as defined below.

* The sender application endpoint compresses the CoAP message, by using the SCHC Rules that it shares with the next hop towards the recipient application endpoint. The resulting, compressed message is sent to the next hop towards the recipient application endpoint.

* Each proxy decompresses the incoming compressed message, by using the SCHC Rules that it shares with the (previous hop towards the) sender application endpoint.

  Then, the proxy compresses the CoAP message to be forwarded, by using the SCHC Rules that it shares with the (next hop towards the) recipient application endpoint.

  The resulting, compressed message is sent to the (next hop towards the) recipient application endpoint.

* The recipient application endpoint decompresses the incoming compressed message, by using the SCHC Rules that it shares with the previous hop towards the sender application endpoint.

5.2. With End-to-End Security

In case OSCORE is used end-to-end between client and server (see Section 7.2 of [RFC8824]), the following applies.

The SCHC processing occurs end-to-end as to the Inner SCHC Compression/Decompression, by relying on Inner SCHC Rules that are consistently shared between the two application endpoints acting as OSCORE endpoints and sharing the used OSCORE Security Context.

Instead, the SCHC processing occurs hop-by-hop as to the Outer SCHC Compression/Decompression, by relying on Outer SCHC Rules that are consistently shared between two adjacent hops.

In particular, SCHC is used as defined below.

* The sender application endpoint performs the Inner SCHC Compression on the original CoAP message, by using the Inner SCHC Rules that it shares with the recipient application endpoint.

  Following the AEAD Encryption of the compressed input obtained from the previous step, the sender application endpoint performs the Outer SCHC Compression on the resulting OSCORE-protected message, by using the Outer SCHC Rules that it shares with the next hop towards the recipient application endpoint.

  The resulting, compressed message is sent to the next hop towards the recipient application endpoint.

* Each proxy performs the Outer SCHC Decompression on the incoming compressed message, by using the SCHC Rules that it shares with the (previous hop towards the) sender application endpoint.

  Then, the proxy performs the Outer SCHC Compression of the OSCORE-protected message to be forwarded, by using the SCHC Rules that it shares with the (next hop towards the) recipient application endpoint.

  The resulting, compressed message is sent to the (next hop towards the) recipient application endpoint.

* The recipient application endpoint performs the Outer SCHC Decompression on the incoming compressed message, by using the Outer SCHC Rules that it shares with the previous hop towards the sender application endpoint.

  Then, the recipient application endpoint performs the AEAD Decryption of the OSCORE-protected message obtained from the previous step.

  Finally, the recipient application endpoint performs the Inner SCHC Decompression on the compressed input obtained from the previous step, by using the Inner SCHC rules that it shares with the sender application endpoint. The result is the original CoAP message produced by the sender application endpoint.

6. Examples of CoAP Header Compression with Proxies

TBD

7. Security Considerations

The security considerations discussed in [RFC8724] and [RFC8824] continue to apply. When SCHC is used in the presence of CoAP proxies, the security considerations discussed in Section 11.2 of [RFC7252] continue to apply. When SCHC is used with OSCORE, the security considerations discussed in [RFC8613] continue to apply.

The security considerations in [RFC8824] specifically discuss how the use of SCHC for CoAP when OSCORE is also used may result in (more frequently) triggering key-renewal operations for the two endpoints. This can be due to an earlier exhaustion of the OSCORE Sender Sequence Number space, or to the installation of new compression Rules on one of the endpoints.

In either case, the two endpoints can run the key update protocol KUDOS defined in [I-D.ietf-core-oscore-key-update], as the recommended method to update their shared OSCORE Security Context.

8. IANA Considerations

This document has no actions for IANA.

9. References

9.1. Normative References

[I-D.ietf-core-oscore-edhoc]
Palombini, F., Tiloca, M., Höglund, R., Hristozov, S., and G. Selander, "Profiling EDHOC for CoAP and OSCORE", Work in Progress, Internet-Draft, draft-ietf-core-oscore-edhoc-06, 23 November 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-edhoc-06>.

[I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., Mattsson, J. P., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", Work in Progress, Internet-Draft, draft-ietf-core-oscore-groupcomm-17, 20 December 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-groupcomm-17>.

[I-D.ietf-core-oscore-key-update]
Höglund, R. and M. Tiloca, "Key Update for OSCORE (KUDOS)", Work in Progress, Internet-Draft, draft-ietf-core-oscore-key-update-03, 24 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-oscore-key-update-03>.

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

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

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <https://www.rfc-editor.org/info/rfc7959>.

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

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/info/rfc8613>.

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

[RFC8768] Boucadair, M., Reddy.K, T., and J. Shallow, "Constrained Application Protocol (CoAP) Hop-Limit Option", RFC 8768, DOI 10.17487/RFC8768, March 2020, <https://www.rfc-editor.org/info/rfc8768>.

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

[RFC9175] Amsüss, C., Preuß Mattsson, J., and G. Selander, "Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing", RFC 9175, DOI 10.17487/RFC9175, February 2022, <https://www.rfc-editor.org/info/rfc9175>.

[RFC9177] Boucadair, M. and J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission", RFC 9177, DOI 10.17487/RFC9177, March 2022, <https://www.rfc-editor.org/info/rfc9177>.

9.2. Informative References

[I-D.ietf-core-groupcomm-bis] Dijk, E., Wang, C., and M. Tiloca, "Group Communication for the Constrained Application Protocol (CoAP)", Work in Progress, Internet-Draft, draft-ietf-core-groupcomm-bis-08, 11 January 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-core-groupcomm-bis-08>.

[I-D.ietf-lake-edhoc] Selander, G., Mattsson, J. P., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", Work in Progress, Internet-Draft, draft-ietf-lake-edhoc-19, 3 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lake-edhoc-19>.

Appendix A. YANG data model

TBD

Acknowledgments

The authors sincerely thank Göran Selander for his comments and feedback.

The work on this document has been partly supported by the H2020 projects SIFIS-Home (Grant agreement 952652) and ARCADIAN-IoT (Grant agreement 101020259).

Authors' Addresses

Marco Tiloca
RISE AB
Isafjordsgatan 22
SE-16440 Kista
Sweden

Email: marco.tiloca@ri.se

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

Ivan Martinez
IMT Atlantique
CS 17607, 2 rue de la Chataigneraie
35576 Cesson-Sevigne Cedex
France
Email: ivanmarinomartinez@gmail.com

lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: 24 August 2023 I. Martinez
Institut MINES TELECOM; IMT Atlantique
20 February 2023

SCHC Rule Access Control
draft-toutain-lpwan-access-control-01

Abstract

The framework for SCHC defines an abstract view of the rules, formalized with through a YANG Data Model. In its original description rules are static and share by 2 entities. The use of YANG authorizes rules to be uploaded or modified in a SCHC instance and leads to some possible attacks, if the changes are not controlled. This document summarizes some possible attacks and define augmentation to the existing Data Mode, to restrict the changes in the rule.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Figure Figure 1 focuses on the management part of the SCHC architecture.

```
  ......................................................
  .   ................................                .
  v   ^                               v               ^
 (--------)     +----------+         +--------+    +-------+-------+
 ( Set of )<--->|coreconf  |<=======|Access  |<===| other end     |<===
 ( Rules  )     |request   |        |Control|    | authentication|
 (--------)     |processing|        +--------+    +---------------+
                +----------+
```

Figure 1: Overview of management architecture.

When a management request arrives on a SCHC instance, the identity of the requester must be checked:

* this can be implicit, for instance a LPWAN device receives it from the SCHC core instance. Authentication is done at Layer 2.

* this can be a L2 address. In a LoRaWAN network, the DevEUI allows the SCHC core instance to identify the device.

* IP addresses may also be used as well as cryptographic keys.

The identification of the requester allows to retrieve the associated Set of Rules. This rules are enriched with access control information that will be defined in this document. If the Set of Rules do not contains any access control information, the management is not allowed to modify the Rules content.

2. Attack scenario

A LWM2M device, under control of an attacker, sends some management messages to modify the SCHC rules in core in order to direct the traffic to another application. This can be either to participate to a DDoS attack or to send sensible information to another application.

SCHC rules are defined for a specific traffic. An attacker changes en element (for instance, the dev UDP port number) and therefore no rule matches the traffic, the link may be saturated by no-compressed messages.

3. YANG Access Control

YANG language allows to specify read only or read write nodes. NACM [RFC8341] extends this by allowing users or group od users to perform specific actions.

This granularity do not fit this the rule model. For instance, the goal is not to allow all the field-id leaves to be modified. The objective is to allow a specific rule entry to be changed and therefore some of the leaves to be modified. For instance an entry with field-id containing Uri-path may have his target-value modified, as in the same rule, the entry regarding the app-prefix should not be changed.

The SCHC access control augments the YANG module defined in [I-D.ietf-lpwan-schc-yang-data-model] to allow a remote entity to manipulate the rules. Several levels are defined.

* in the set of rules, it authorizes or not a new rule to be added .

* in a compression rule, it allows to add or remove field descriptions.

* in a compression rule, it allows to modify some elements of the rule, such as the target-value, the matching-operator or/and the comp-decomp-action and associated values.

* in a fragmentation rule, it allows to modify some parameters.

4. YANG Data Model

The YANG DM proposed in Appendix A extends the SCHC YANG Data Model introduced in [I-D.ietf-lpwan-schc-yang-data-model]. It adds read-only leaves containing the access rights. If these leaves are not presents, the information cannot be modified.

4.1. leaf ac-modify-set-of-rules

This leaf controls modifications applied to a set of rules. They are specified with the rule-access-right enumeration:

* no-change (0): rules cannot be modified in the Set of Rules. This is the equivalent of having no access control elements in the set of rules.

* modify-existing-element (1): an existing rule may be modified.

* add-remove-element (2): a rule can be added or deleted from the Set of Rules or an existing rule can be modified.

4.2. leaf ac-modify-compression-rule

This leaf allows to modify a compression element. To be active, leaf ac-modify-set-of-rules MUST be set to modify-existing-element or add-remove-element. This leaf uses the same enumeration as add-remove-element:

* no-change (0): The rule cannot be modified.

* modify-existing-element (1): an existing Field Description may be modified.

* add-remove-element (2): a Field Description can be added or deleted from the Rule or an existing rule can be modified.

4.3. leaf ac-modify-field

This leaf allows to modify a Field Description in a compression rule. To be active, leaves ac-modify-set-of-rules and ac-modify-compression-rule MUST be set to modify-existing-element or add-remove-element and ac-modifiy-compression-rule and leaf

5. Normative References

[I-D.ietf-lpwan-schc-yang-data-model]
Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", Work in Progress,

Internet-Draft, draft-ietf-lpwan-schc-yang-data-model-21, 9 October 2022, <https://www.ietf.org/archive/id/draft-ietf-lpwan-schc-yang-data-model-21.txt>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

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

Appendix A. YANG Data Model

```
<CODE BEGINS> file "ietf-schc-access-control@2023-02-14.yang"
module ietf-schc-access-control {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schc-access-control";
  prefix schc-ac;

  import ietf-schc {
      prefix schc;
  }

  organization
    "IETF IPv6 over Low Power Wide-Area Networks (lpwan) working group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/lpwan/about/>
     WG List:  <mailto:lp-wan@ietf.org>
     Editor:   Juan-Carlos Zuniga
       <mailto:juancarlos.zuniga@sigfox.com>";
  description
     "
     Copyright (c) 2021 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

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

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

```
        for full legal notices.

        The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
        NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
        'MAY', and 'OPTIONAL' in this document are to be interpreted as
        described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.

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

        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 Ack and add the possibility to send several bitmaps in a single
        answer.";

     revision 2023-02-14 {
       description
         "Initial version for RFC YYYY ";
       reference
         "RFC YYYY: Compound Ack";
     }

     typedef rule-access-right {
       type enumeration {
         enum no-changes {
           value 0;
           description
             "No change are allowed.";
         }
         enum modify-existing-element {
           value 1;
           description
             "can modify content inside an element.";
         }
         enum add-remove-element {
           value 2;
           description
             "Allows to add or remove or modify an element.";
         }
       }
     }

     typedef field-access-right {
       type enumeration {
         enum no-change {
           value 0;
           description
```

```
              "Reserved slot number.";
          }
          enum change-tv {
            value 1;
            description
              "Reserved slot number.";
          }
          enum change-mo-cda-tv {
            value 2;
            description
              "Reserved slot number.";
          }
        }

      }

      augment "/schc:schc/schc:rule" {
        leaf ac-modify-set-of-rules {
              config false;
              type rule-access-right;
            }
      }

      augment "/schc:schc/schc:rule/schc:nature/schc:compression" {
        leaf ac-modify-compression-rule {
              config false;
              type rule-access-right;
            }
      }

      augment "/schc:schc/schc:rule/schc:nature/schc:compression/schc:entry" {
        leaf ac-modify-field {
              config false;
              type field-access-right;
            }
      }

      augment "/schc:schc/schc:rule/schc:nature/schc:fragmentation" {
        leaf ac-modify-timers {
              config false;
              type boolean;
            }
      }


    }
    <CODE ENDS>
```

Appendix B. Security Considerations

TBD

Appendix C. IANA Considerations

TBD

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

Ivan Martinez
Institut MINES TELECOM; IMT Atlantique
2 rue de la Chataigneraie
CS 17607
35576 Cesson-Sevigne Cedex
France
Email: ivan-marino.martinez-bolivar@imt-atlantique.fr

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: 27 August 2023
23 February 2023

A. Minaburo
Acklio
L. Toutain
Institut MINES TELECOM; IMT Atlantique

SCHC Rule Access Control
draft-toutain-lpwan-sid-allocation-02

Abstract

blabla

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

RFC9363 defines a YANG Data Model for SCHC rules. [I-D.ietf-core-sid] specifies the process for SID allocation and management. This document discuss of the SID allocation for RFC9363.

2. SCHC YANG Data Model

The version @2023-01-18 of the SCHC YANG Data Model published in the RFC 9363 contains 136 SIDs (92 for identities, 2 for features and 42 for data). [I-D.ietf-core-sid] indicates that the SID range for the YANG Data Model specified in RFC is between 1000 and 59 000 and that the maximum request pool SHOULD NOT exceed 1000. The draft also gives some pre allocated values.

Since SIDs will be used either to represent unique identity contained in the data model and also leaves (data) forming this data model, it could be wise to distinguish between identifiers and data.

Data structures are delta encoded and included as a CBOR element, the size depends on the value. Deltas between -24 and +23 are encoded on a single byte. Deltas between -256 and +255 use 2 bytes and larger values corresponding to the RFC SID range will be encoded into 3 bytes. To optimize the CORECONF representation delta should be smaller as possible for the more frequent leaves.

On the other hand identities are included in the CORECONF representation and for the RFC SID range the size is constant and equal to 3 bytes.

2.1. Example

CORECONF

```
{5095: {1: [{4:
        [{1: 5015,
          5: 5018,
          6: 5068,
          7: 4,
          8: 1,
          9: 5083,
          13: [{1: 0, 2: h'06'}]},
         {1: 5015,
          5: 5018,
          6: 2000003,
          7: 8,
          8: 1,
          9: 5083,
          13: [{1: 0, 2: h'00'}]}]
    }]}
}
```

RESTCONF

```
{"ietf-schc:schc": {"rule": [{"entry":
             [{"comp-decomp-action": "ietf-schc:cda-not-sent",
               "direction-indicator": "ietf-schc:di-bidirectional",
               "field-id": "ietf-schc:fid-ipv6-version",
               "field-length": 4,
               "field-position": 1,
               "matching-operator": "ietf-schc:mo-equal",
               "target-value": [{"index": 0, "value": "Bg=="}]},
              {"comp-decomp-action": "ietf-schc:cda-not-sent",
               "direction-indicator": "ietf-schc:di-bidirectional",
               "field-id": "ietf-schc-oam:fid-icmpv6-type",
               "field-length": 8,
               "field-position": 1,
               "matching-operator": "ietf-schc:mo-equal",
               "target-value": [{"index": 0, "value": "gA=="}]} ]
         }]}
    }
```

Figure 1

The example in Figure 1 gives a CORECONF structure transposed the CBOR diagnostic notation and its equivalent in RESTCONF with JSON. For readability and compactness, this example is edited and do not encode a full rule as defined in RFC9363.

The default SID numbering produced by pyang is used, starting from 5000 for SCHC Data Model defined in RFC9363 and 2000000 for an experimental module for OAM.

We can see the delta encoding. The first SID 5095 represents "ietf-schc:schc". "/ietf-schc:schc/rule" which is coded with a +1 since SID 5096 has been assigned. "/ietf-schc:schc/rule/entry" is coded with a delta of 4. Then a list of Field Description follows. +1 represents the leaf "ietf-schc:schc/rule/entry/comp-decomp-action" and the value assigned to that key contains the SID of "ietf-schc:cda-not-sent" identity.

Note that the second element contains a "field-id" belonging to the "ietf-schc-oam" module and the associate SID is 2000003.

3. Recommendation for SID values

The SCHC YANG Data Model defined in RFC 9363 will probably be augmented, to include for instance access control data. To keep a compact representation, delta values must be kept as small as possible. The LPWAN working group should not use the automatic SID numbering and provide a more optimal allocation scheme for augmentation of the SCHC YANG Data Model.

A first recommendation is to avoid merging data and identity in order to limit the delta encoding. The distance between these two sections can be 255 SID to allow deltas on 2 bytes.

The second recommendation is to leave some unused SID around SCHC rules to allow augmentation.

4. SID for data

We propose to use a range of 300 values for the YANG Data Model defined in RFC9263, which introduce room for future augmentation of the Data Model, such as [I-D.toutain-lpwan-access-control] or [I-D.ietf-lpwan-schc-compound-ack]. This will break the automatic allocation process done by pyang and based on the nature of the SID and the alphabetical order.

It is also worth noting that in the current SID allocation based on alphabetical order places rule-id-value and rule-id-length, rule-nature from the 33 to 35 position. CBOR encoding will be on two bytes for each of the values. Since these three values are present in all the rules, a smaller value will optimize the CORECONF representation.

The allocation algorithm is the following:

* leaves between containers and list a maximal distance of 23 SIDS. Positive and negative deltas will be encoded on 1 byte.

* fill this gap with the more common values defined in the container or the list

* keep unused values for future augmentations.

* a guard of 255 after the last list will be kept unused before allocating identities. This range allow a delta encoded on 2 bytes.

The LPWAN group will receive an range of SID values (we suppose starting at 5000). The SIDs will be allocated following the previous algorithm.

Other RFCs modifying the SCHC YANG Data Model will include a YANG module. The lpwan WG will decide of the SID allocation and produce a SID file with the mapping.

5. SID allocation

We propose the following allocation scheme for RFC9363:

```
   5000    - 5022 : RESERVED FOR /ietf-schc:schc

   5023    module ietf-schc
   5024    data /ietf-schc:schc

   5025    - 5046 : RESERVED FOR /ietf-schc:schc AND /ietf-schc:schc/rule

   5047    data /ietf-schc:schc/rule
   5048    data /ietf-schc:schc/rule/rule-id-length
   5049    data /ietf-schc:schc/rule/rule-id-value
   5050    data /ietf-schc:schc/rule/rule-nature

   5051    - 5069 : RESERVED FOR /ietf-schc:schc/rule AND /ietf-schc:schc/rule/en
try

   5070    data /ietf-schc:schc/rule/entry
   5071    data /ietf-schc:schc/rule/entry/comp-decomp-action
   5072    data /ietf-schc:schc/rule/entry/comp-decomp-action-value
   5073    data /ietf-schc:schc/rule/entry/comp-decomp-action-value/index
   5074    data /ietf-schc:schc/rule/entry/comp-decomp-action-value/value
   5075    data /ietf-schc:schc/rule/entry/direction-indicator
   5076    data /ietf-schc:schc/rule/entry/field-id
   5077    data /ietf-schc:schc/rule/entry/field-length
   5078    data /ietf-schc:schc/rule/entry/field-position
   5079    data /ietf-schc:schc/rule/entry/matching-operator
```

```
5080    data /ietf-schc:schc/rule/entry/matching-operator-value
5081    data /ietf-schc:schc/rule/entry/matching-operator-value/index
5082    data /ietf-schc:schc/rule/entry/matching-operator-value/value
5083    data /ietf-schc:schc/rule/entry/target-value
5084    data /ietf-schc:schc/rule/entry/target-value/index
5085    data /ietf-schc:schc/rule/entry/target-value/value

5086    - 5094 : RESERVED

5094    data /ietf-schc:schc/rule/ack-behavior
5095    data /ietf-schc:schc/rule/direction
5096    data /ietf-schc:schc/rule/dtag-size
5097    data /ietf-schc:schc/rule/fcn-size
5098    data /ietf-schc:schc/rule/fragmentation-mode
5099    data /ietf-schc:schc/rule/inactivity-timer
5100    data /ietf-schc:schc/rule/inactivity-timer/ticks-duration
5101    data /ietf-schc:schc/rule/inactivity-timer/ticks-numbers
5102    data /ietf-schc:schc/rule/l2-word-size
5103    data /ietf-schc:schc/rule/max-ack-requests
5104    data /ietf-schc:schc/rule/max-interleaved-frames
5105    data /ietf-schc:schc/rule/maximum-packet-size
5106    data /ietf-schc:schc/rule/rcs-algorithm
5107    data /ietf-schc:schc/rule/retransmission-timer
5108    data /ietf-schc:schc/rule/retransmission-timer/ticks-duration
5109    data /ietf-schc:schc/rule/retransmission-timer/ticks-numbers

5110    - 5115 : RESERVED FOR TIMER

5116    data /ietf-schc:schc/rule/tile-in-all-1
5117    data /ietf-schc:schc/rule/tile-size
5118    data /ietf-schc:schc/rule/w-size
5119    data /ietf-schc:schc/rule/window-size

5120    - 5299 : RESERVED FOR 2 BYTES DELTAS

5300    identity ack-behavior-after-all-0
5301    identity ack-behavior-after-all-1
5302    identity ack-behavior-base-type
5303    identity ack-behavior-by-layer2
5304    identity all-1-data-base-type
5305    identity all-1-data-no
5306    identity all-1-data-sender-choice
5307    identity all-1-data-yes
5308    identity cda-appiid
5309    identity cda-base-type
5310    identity cda-compute
5311    identity cda-deviid
5312    identity cda-lsb
```

```
5313    identity cda-mapping-sent
5314    identity cda-not-sent
5315    identity cda-value-sent
5316    identity di-base-type
5317    identity di-bidirectional
5318    identity di-down
5319    identity di-up
5320    identity fid-base-type
5321    identity fid-coap-base-type
5322    identity fid-coap-code
5323    identity fid-coap-code-class
5324    identity fid-coap-code-detail
5325    identity fid-coap-mid
5326    identity fid-coap-option
5327    identity fid-coap-option-accept
5328    identity fid-coap-option-block1
5329    identity fid-coap-option-block2
5330    identity fid-coap-option-content-format
5331    identity fid-coap-option-etag
5332    identity fid-coap-option-if-match
5333    identity fid-coap-option-if-none-match
5334    identity fid-coap-option-location-path
5335    identity fid-coap-option-location-query
5336    identity fid-coap-option-max-age
5337    identity fid-coap-option-no-response
5338    identity fid-coap-option-observe
5339    identity fid-coap-option-oscore-flags
5340    identity fid-coap-option-oscore-kid
5341    identity fid-coap-option-oscore-kidctx
5342    identity fid-coap-option-oscore-piv
5343    identity fid-coap-option-proxy-scheme
5344    identity fid-coap-option-proxy-uri
5345    identity fid-coap-option-size1
5346    identity fid-coap-option-size2
5347    identity fid-coap-option-uri-host
5348    identity fid-coap-option-uri-path
5349    identity fid-coap-option-uri-port
5350    identity fid-coap-option-uri-query
5351    identity fid-coap-tkl
5352    identity fid-coap-token
5353    identity fid-coap-type
5354    identity fid-coap-version
5355    identity fid-ipv6-appiid
5356    identity fid-ipv6-appprefix
5357    identity fid-ipv6-base-type
5358    identity fid-ipv6-deviid
5359    identity fid-ipv6-devprefix
5360    identity fid-ipv6-flowlabel
```

```
5361    identity fid-ipv6-hoplimit
5362    identity fid-ipv6-nextheader
5363    identity fid-ipv6-payload-length
5364    identity fid-ipv6-trafficclass
5365    identity fid-ipv6-trafficclass-ds
5366    identity fid-ipv6-trafficclass-ecn
5367    identity fid-ipv6-version
5368    identity fid-oscore-base-type
5369    identity fid-udp-app-port
5370    identity fid-udp-base-type
5371    identity fid-udp-checksum
5372    identity fid-udp-dev-port
5373    identity fid-udp-length
5374    identity fl-base-type
5375    identity fl-token-length
5376    identity fl-variable
5377    identity fragmentation-mode-ack-always
5378    identity fragmentation-mode-ack-on-error
5379    identity fragmentation-mode-base-type
5380    identity fragmentation-mode-no-ack
5381    identity mo-base-type
5382    identity mo-equal
5383    identity mo-ignore
5384    identity mo-match-mapping
5385    identity mo-msb
5386    identity nature-base-type
5387    identity nature-compression
5388    identity nature-fragmentation
5389    identity nature-no-compression
5390    identity rcs-algorithm-base-type
5391    identity rcs-crc32
5392    feature compression
5393    feature fragmentation

5394    - 5500 : RESERVED FOR IDENTITY
```

For instance [I-D.toutain-lpwan-access-control] augments the model with "ac-modify-set-of-rules" at the top level, "ac-modify-compression-rule" for each compression rule, "ac-modify-field" in each Field Description of a compression rule and finally "ac-modify-timers" in fragmentation rules. Delta representation will be on 1 byte.

The following SIDs could be assigned:

* 5022: ac-modify-set-of-rules

* 5051: ac-modify-compression-rule

* 5069: ac-modify-field

* 5068: ac-modify-timers

[I-D.ietf-lpwan-schc-compound-ack] augments the model for fragmentation, with 3 identity and two leaves. identities can get a SID 5394 to 5396 and the two SIDs for the leaves can be 5120 and 5122. There delta representations will be coded on 2 bytes.

6. Normative References

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-19, 26 July 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-core-sid-19>.

[I-D.ietf-lpwan-schc-compound-ack]
Zúñiga, J. C., Gomez, C., Aguilar, S., Toutain, L., Cespedes, S., and D. S. W. L. Torre, "SCHC Compound ACK", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-compound-ack-12, 21 February 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-schc-compound-ack-12>.

[I-D.ietf-lpwan-schc-yang-data-model]
Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-yang-data-model-21, 9 October 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-schc-yang-data-model-21>.

[I-D.toutain-lpwan-access-control]
Minaburo, A., Toutain, L., and I. Martinez, "SCHC Rule Access Control", Work in Progress, Internet-Draft, draft-toutain-lpwan-access-control-01, 20 February 2023, <https://datatracker.ietf.org/doc/html/draft-toutain-lpwan-access-control-01>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

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

Appendix A. Security Considerations

TBD

Appendix B. IANA Considerations

TBD

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