lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: January 4, 2021 Institut MINES TELECOM; IMT Atlantique
R. Andreasen
Universidad de Buenos Aires
July 03, 2020

LPWAN Static Context Header Compression (SCHC) for CoAP
draft-ietf-lpwan-coap-static-context-hc-15

Abstract

This draft defines the way Static Context Header Compression (SCHC) header compression can be applied to the Constrained Application Protocol (CoAP). SCHC is a header compression mechanism adapted for constrained devices. SCHC uses a static description of the header to reduce the redundancy and the size of the information in the header. While [rfc8724] describes the SCHC compression and fragmentation framework, and its application for IPv6/UDP headers, this document applies the use of SCHC for CoAP headers. The CoAP header structure differs from IPv6 and UDP since CoAP uses a flexible header with a variable number of options, themselves of variable length. The CoAP protocol messages format is asymmetric: the request messages have a header format different from the one in the response messages. This specification gives guidance on how to apply SCHC to flexible headers and how to leverage the asymmetry for more efficient compression Rules.

Table of Contents

1. Introduction

CoAP [rfc7252] is designed to easily interop with HTTP (Hypertext Transfer Protocol) and is optimized for REST-based (Representational state transfer) services. Although CoAP was designed for constrained devices, the size of a CoAP header still is too large for the constraints of LPWAN (Low Power Wide Area Networks) and some compression is needed to reduce the header size.

The [rfc8724] defines SCHC, a header compression mechanism for LPWAN network based on a static context. Section 5 of the [rfc8724] explains the architecture where compression and decompression are done. The context is known by both ends before transmission. The way the context is configured, provisioned or exchanged is out of the scope of this document.

CoAP is an End-to-End protocol at the application level, so CoAP compression requires to install common Rules between two hosts and IP Routing may be needed to allow End-to-End communication. Therefore, SCHC compression may apply at two different levels, one to compress IP and UDP as described in [rfc8724] in the LPWAN network and another at the application level. These two compressions may be independent. The Compression Rules can be set up by two independent entities and are out of the scope of this document. In both cases, SCHC mechanism remains the same.

SCHC compresses and decompresses headers based on shared contexts between devices. Each context consists of multiple Rules. Each Rule can match header fields and specific values or ranges of values. If a Rule matches, the matched header fields are substituted by the RuleID and optionally some residual bits. Thus, different Rules may correspond to different types of packets that a device expects to send or receive.

A Rule describes the complete header of the packet with an ordered list of fields descriptions, see section 7 of [rfc8724], thereby each description contains the field ID (FID), its length (FL) and its position (FP), a direction indicator (DI) (upstream, downstream and bidirectional) and some associated Target Values (TV).

A Matching Operator (MO) is associated to each header field description. The Rule is selected if all the MOs fit the TVs for all fields of the incoming header.

In that case, a Compression/Decompression Action (CDA) associated to each field defines how the compressed and the decompressed values are computed. Compression mainly results in one of 4 actions:

- send the field value,
- send nothing,
- send some least significant bits of the field or
- send an index.

After applying the compression there may be some bits to be sent, these values are called Compression Residues.

SCHC is a general mechanism that can be applied to different protocols, the exact Rules to be used depend on the protocol and the application. The section 10 of the [rfc8724] describes the compression scheme for IPv6 and UDP headers. This document targets the CoAP header compression using SCHC.

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.

2. Applying SCHC to CoAP headers

The SCHC Compression Rules can be applied to CoAP headers. SCHC Compression of the CoAP header MAY be done in conjunction with the lower layers (IPv6/UDP) or independently. The SCHC adaptation layers as described in Section 5 of [rfc8724] and may be used as shown in Figure 1, Figure 2 and Figure 3.

In the first example Figure 1, a Rule compresses the complete header stack from IPv6 to CoAP. In this case, SCHC C/D (Static Context Header Compression Compressor/Decompressor) is performed at the Sender and at the Receiver. The host communicating with the device do not implement SCHC C/D.

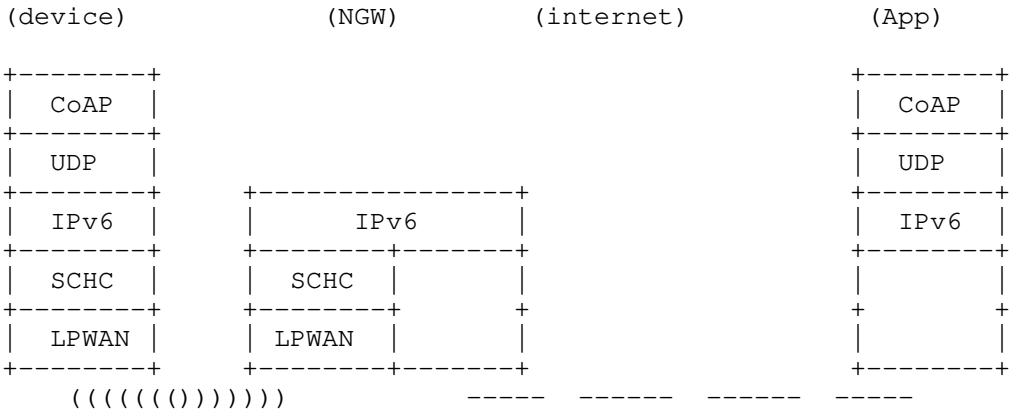


Figure 1: Compression/decompression at the LPWAN bondary

In the second example, Figure 2, the SCHC compression is applied in the CoAP layer, compressing the CoAP header independently of the other layers. The RuleID and the Compression Residue are encrypted using a mechanism such as DTLS. Only the other end can decipher the information. If needed, layers below use SCHC to compress the header as defined in [rfc8724] document. This use case realizes an End-to-End context initialization between the sender and the receiver and is out-of-scope of this document.

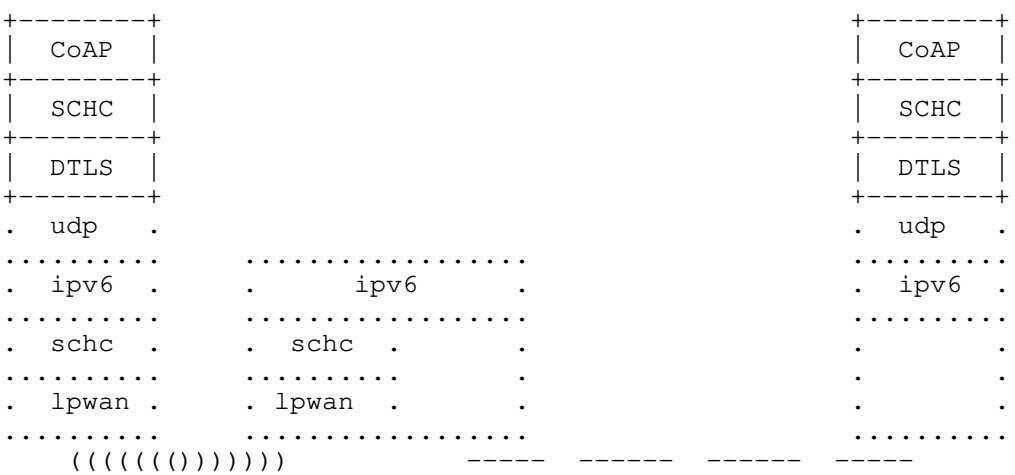


Figure 2: Standalone CoAP ene-to-end compression/decompression

In the third example, Figure 3 the Object Security for Constrained RESTful Environments (OSCORE) [rfc8613] is used. In this case, two rulesets are used to compress the CoAP message. A first ruleset focused on the inner header and is applied end to end by both ends. A second ruleset compresses the outer header and the layers below and is done between the Sender and the Receiver.

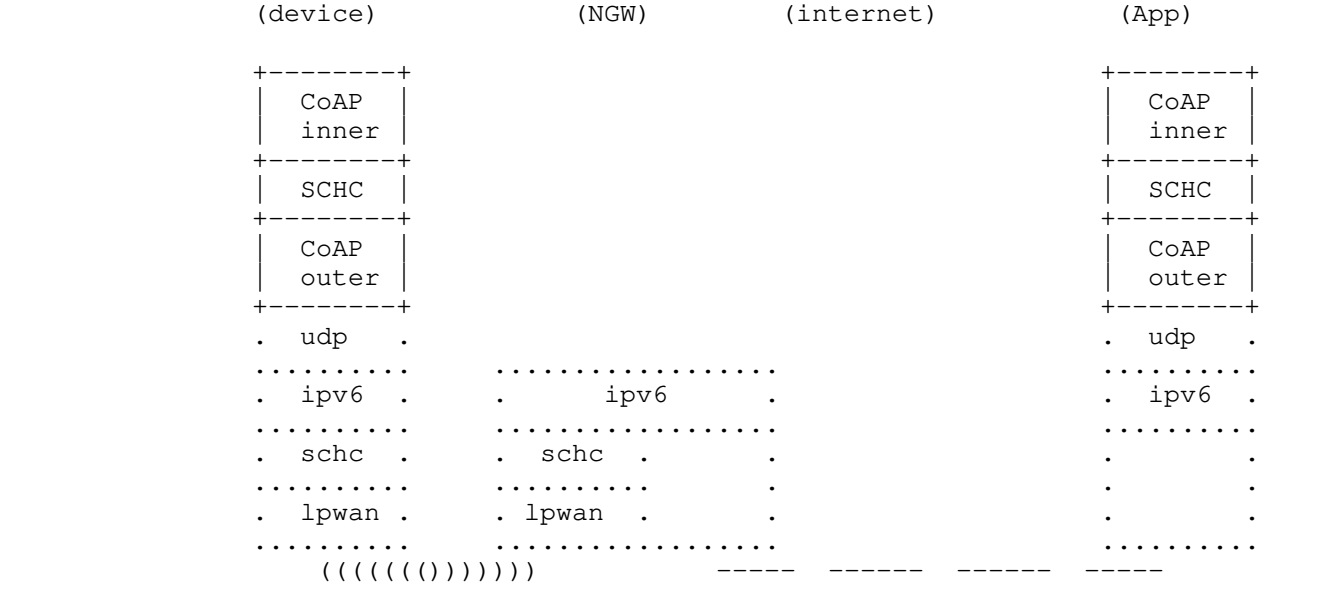


Figure 3: OSCORE compression/decompression.

In case of 2 rule-sets, as shown in Figure 2 and Figure 3, they may come from different provisioning domains, and that they do not include the cryptography part that is done in between the two SCHC activities. This document focuses on CoAP compression represented in the dashed boxes in the previous figures.

3. CoAP Headers compressed with SCHC

The use of SCHC over the CoAP header uses the same description and compression/decompression techniques as the one for IP and UDP explained in the [rfc8724]. For CoAP, SCHC Rules description uses the direction information to optimize the compression by reducing the number of Rules needed to compress headers. The field description MAY define both request/response headers and target values in the same Rule, using the DI (direction indicator) to make the difference. As for other protocols, when the compressor does not find a correct Rule to compress the header, the packet MUST be sent uncompressed using the RuleID dedicated to this purpose, and the Compression

Residue is the complete header of the packet. See section 6 of [rfc8724].

3.1. Differences between CoAP and UDP/IP Compression

CoAP compression differs from IPv6 and UDP compression on the following aspects:

- The CoAP protocol is asymmetric, the headers are different for a request or a response. For example, the URI-path option is mandatory in the request, and it is not present in the response, a request may contain an Accept option, and the response may include a Content option. In comparison, IPv6 and UDP returning path swap the value of some fields in the header.
  But all the directions have the same fields (e.g., source and destination addresses fields).

  The [rfc8724] defines the use of a Direction Indicator (DI) in the Field Description, which allows a single Rule to process message headers differently depending on the direction.

- Even when a field is "symmetric" (i.e., found in both directions), the values carried in each direction are different.
  The compression may use a matching list in the TV to limit the range of expected values in a particular direction and therefore reduces the size of the Compression Residue. Through the Direction Indicator (DI), a field description in the Rules splits the possible field value into two parts, one for each direction. For instance, if a client sends only CON requests, the type can be elided by compression, and the answer may use one single bit to carry either the ACK or RST type. The field Code have as well the same behavior, the 0.0X code format value in the request and Y.ZZ code format in the response.

- Headers in IPv6 and UDP have a fixed size. The size is not sent as part of the Compression Residue, but is defined in the Rule. Some CoAP header fields have variable lengths, so the length is also specified in the Field Description. For example, the Token size may vary from 0 to 8 bytes. And the CoAP options have a variable length since they use the Type-Length-Value encoding format, as URI-path or URI-query.

  Section 7.5.2 from [rfc8724] offers the possibility to define a function for the Field length in the Field Description to have knowledge of the length before compression. When doing SCHC compression of a variable-length field,
  if the field size is not known, the Field Length in the Rule is set as variable and the size is sent with the Compression Residue.

- A field can appear several time in the CoAP headers. This is typical for elements of a URI (path or queries). The SCHC specification [rfc8724] allows a Field ID to appear several times in the Rule, and uses the Field Position (FP) to identify the correct instance, and thereby removing the ambiguity of the matching operation.

- Field sizes defined in the CoAP protocol can be too large regarding LPWAN traffic constraints. This is particularly true for the Message ID field and the Token field. SCHC uses different Matching operators (MO) to perform the compression, see section 7.4 of [rfc8724]. In this case the Most Significant Bits (MSB) MO can be applied to reduce the information carried on LPWANs.

4. Compression of CoAP header fields

This section discusses the compression of the different CoAP header fields. The CoAP compression with SCHC follows the Section 7.1 of [rfc8724].

4.1. CoAP version field

CoAP version is bidirectional and MUST be elided during the SCHC compression, since it always contains the same value. In the future, if new versions of CoAP are defined, new Rules will be needed to avoid ambiguities between versions.

4.2. CoAP type field

The CoAP Protocol [rfc7252] has four type of messages: two request (CON, NON); one response (ACK) and one empty message (RST).

The field SHOULD be elided if for instance a client is sending only NON or only CON messages. For the RST message a dedicated Rule may be needed. For other usages a mapping list can be used.

4.3. CoAP code field

The code field indicates the Request Method used in CoAP, a IANA registry [rfc7252]. The compression of the CoAP code field follows the same principle as that of the CoAP type field. If the device plays a specific role, the set of code values can be split in two parts, the request codes with the 0 class and the response values.

If the device only implements a CoAP client, the request code can be reduced to the set of requests the client is able to process.

A mapping list can be used for known values. For other values the field cannot be compressed an the value needs to be sent in the Compression Residue.

4.4. CoAP Message ID field

The Message ID field can be compressed with the MSB(x) MO and the Least Significant Bits (LSB) CDA, see section 7.4 of [rfc8724].

4.5. CoAP Token fields

Token is defined through two CoAP fields, Token Length in the mandatory header and Token Value directly following the mandatory CoAP header.

Token Length is processed as any protocol field. If the value does not change, the size can be stored in the TV and elided during the transmission. Otherwise, it will have to be sent in the Compression Residue.

Token Value MUST NOT be sent as a variable length residue to avoid ambiguity with Token Length. Therefore, Token Length value MUST be used to define the size of the Compression Residue. A specific function designated as "TKL" MUST be used in the Rule. During the decompression, this function returns the value contained in the Token Length field.

5. CoAP options

CoAP defines options that are placed after the based header in Option Numbers order, see [rfc7252]. Each Option instance in a message uses the format Delta-Type (D-T), Length (L), Value (V). When applying SCHC compression to the Option, the D-T, L, and V format serves to make the Rule description of the Option. The SCHC compression builds the description of the Option by using in the Field ID the Option Number built from D-T; in TV, the Option Value; and the Option Length uses section 7.4 of RFC8724. When the Option Length has a wellknown size it can be stored in the Rule. Therefore, SCHC compression does not send it. Otherwise, SCHC Compression carries the length of the Compression Residue in addition to the Compression Residue value.

CoAP request and response do not include the same options. So Compression Rules may reflect these assymetry by tagging the direction indicator.

Note that length coding differs between CoAP options and SCHC variable size Compression Residue.

The following sections present how SCHC compresses some specific CoAP Options.

5.1. CoAP Content and Accept options.

If a single value is expected by the client, it can be stored in the TV and elided during the transmission. Otherwise, if several possible values are expected by the client, a matching-list SHOULD be used to limit the size of the Compression Residue. Otherwise, the value has to be sent as a Compression Residue (fixed or variable length).

5.2. CoAP option Max-Age, Uri-Host and Uri-Port fields

If the duration is known by both ends, the value can be elided.

A matching list can be used if some well-known values are defined.

Otherwise these options can be sent as a Compression Residue (fixed or variable length).

5.3. CoAP option Uri-Path and Uri-Query fields

Uri-Path and Uri-Query elements are a repeatable options, the Field Position (FP) gives the position in the path.

A Mapping list can be used to reduce the size of variable Paths or Queries. In that case, to optimize the compression, several elements can be regrouped into a single entry. Numbering of elements do not change, MO comparison is set with the first element of the matching.

Field	FL	FP	DI	Target Value	Match Opera.	CDA
URI-Path		1	up	["/a/b", "/c/d"]	equal	not-sent
URI-Path	var	3	up		ignore	value-sent

Figure 4: complex path example

In Figure 4 a single bit residue can be used to code one of the 2 paths. If regrouping were not allowed, a 2 bits residue would be needed. The third path element is sent as a variable size residue.

5.3.1. Variable length Uri-Path and Uri-Query

When the length is not known at the Rule creation, the Field Length MUST be set to variable, and the unit is set to bytes.

The MSB MO can be applied to a Uri-Path or Uri-Query element. Since MSB value is given in bit, the size MUST always be a multiple of 8 bits.

The length sent at the beginning of a variable length residue indicates the size of the LSB in bytes.

For instance for a CORECONF path /c/X6?k="eth0" the Rule can be set to:

Field	FL	FP	DI	Target Value	Match Opera.	CDA
URI-Path	8	1	up	"c"	equal	not-sent
URI-Path	var	2	up		ignore	value-sent
URI-Query	var	1	up	"k="	MSB(16)	LSB

Figure 5: CORECONF URI compression

Figure 5 shows the parsing and the compression of the URI, where c is not sent. The second element is sent with the length (i.e. 0x2 X 6) followed by the query option (i.e. 0x05 "eth0").

5.3.2. Variable number of path or query elements

The number of Uri-path or Uri-Query elements in a Rule is fixed at the Rule creation time. If the number varies, several Rules SHOULD be created to cover all the possibilities. Another possibility is to define the length of Uri-Path to variable and send a Compression Residue with a length of 0 to indicate that this Uri-Path is empty. This adds 4 bits to the variable Residue size. See section 7.5.2 [rfc8724]

5.4. CoAP option Size1, Size2, Proxy-URI and Proxy-Scheme fields

If the field value has to be sent, TV is not set, MO is set to "ignore" and CDA is set to "value-sent". A mapping MAY also be used.

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

5.5. CoAP option ETag, If-Match, If-None-Match, Location-Path and Location-Query fields

These fields values cannot be stored in a Rule entry. They MUST always be sent with the Compression Residues.

6. SCHC compression of CoAP extension RFCs

6.1. Block

Block [rfc7959] allows a fragmentation at the CoAP level. SCHC also includes a fragmentation protocol. They can be both used. If a block option is used, its content MUST be sent as a Compression Residue.

6.2. Observe

The [rfc7641] defines the Observe option. The TV is not set, MO is set to "ignore" and the CDA is set to "value-sent". SCHC does not limit the maximum size for this option (3 bytes). To reduce the transmission size, either the device implementation MAY limit the delta between two consecutive values, or a proxy can modify the increment.

Since an RST message may be sent to inform a server that the client does not require Observe response, a Rule MUST allow the transmission of this message.

6.3. No-Response

The [rfc7967] defines a No-Response option limiting the responses made by a server to a request. If the value is known by both ends, then TV is set to this value, MO is set to "equal" and CDA is set to "not-sent".

Otherwise, if the value is changing over time, TV is not set, MO is set to "ignore" and CDA to "value-sent". A matching list can also be used to reduce the size.

6.4. OSCORE

OSCORE [rfc8613] defines end-to-end protection for CoAP messages. This section describes how SCHC Rules can be applied to compress OSCORE-protected messages.

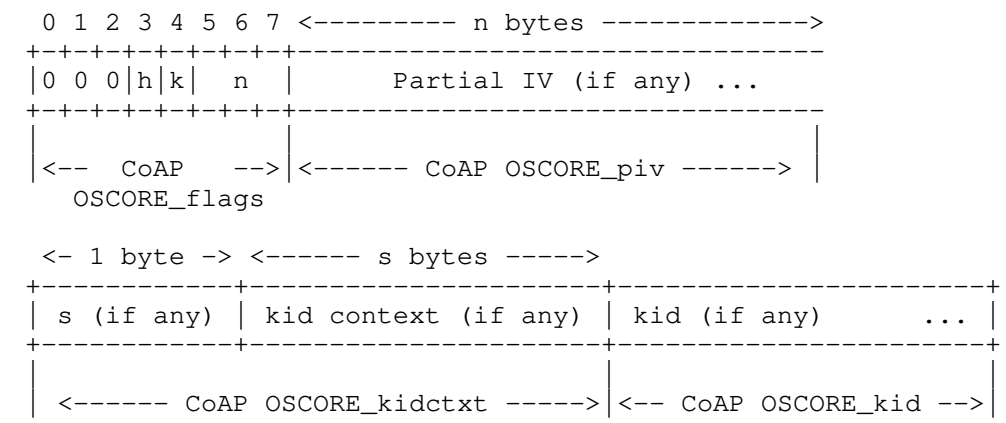


Figure 6: OSCORE Option

The encoding of the OSCORE Option Value defined in Section 6.1 of [rfc8613] is repeated in Figure 6.

The first byte specifies the content of the OSCORE options using flags. The three most significant bits of this byte are reserved and always set to 0. Bit h, when set, indicates the presence of the kid context field in the option. Bit k, when set, indicates the presence of a kid field. The three least significant bits n indicate the length of the piv (Partial Initialization Vector) field in bytes. When n = 0, no piv is present.

The flag byte is followed by the piv field, kid context field, and kid field in this order, and if present, the length of the kid context field is encoded in the first byte denoting by s the length of the kid context in bytes.

This specification recommends identifying the OSCORE Option and the fields it contains Conceptually, it discerns up to 4 distinct pieces of information within the OSCORE option: the flag bits, the piv, the kid context, and the kid. The SCHC Rule splits into four field descriptions the OSCORE option to compress them:

- CoAP OSCORE_flags,
- CoAP OSCORE_piv,
- CoAP OSCORE_kidctxt,
- CoAP OSCORE_kid.

The OSCORE Option shows superimposed these four fields using the format Figure 6, the CoAP OSCORE_kidctxt field includes the size bits s.

7. Examples of CoAP header compression

7.1. Mandatory header with CON message

In this first scenario, the LPWAN Compressor at the Network Gateway side receives from an Internet client a POST message, which is immediately acknowledged by the Device. For this simple scenario, the Rules are described Figure 7.

RuleID 1

Field	FL	FP	DI	Target Value	Match Opera.	CDA	Sent [bits]
CoAP version			bi	01	equal	not-sent	
CoAP Type			dw	CON	equal	not-sent	
CoAP Type			up	[ACK, RST]	match-map	matching-sent	T
CoAP TKL			bi	0	equal	not-sent	
CoAP Code			bi	[0.00, ... 5.05]	match-map	matching-sent	CC CCC
CoAP MID			bi	0000	MSB(7)	LSB	M-ID
CoAP Uri-Path			dw	path	equal 1	not-sent	

Figure 7: CoAP Context to compress header without token

The version and Token Length fields are elided. The 26 method and response codes defined in [rfc7252] has been shrunk to 5 bits using a matching list. Uri-Path contains a single element indicated in the matching operator.

SCHC Compression reduces the header sending only the Type, a mapped code and the least significant bits of Message ID (9 bits in the example above).

Note that a request sent by a client located in an Application Server to a server located in the device, may not be compressed through this Rule since the MID will not start with 7 bits equal to 0. A CoAP proxy, before the core SCHC C/D can rewrite the message ID to a value matched by the Rule.

7.2. OSCORE Compression

OSCORE aims to solve the problem of end-to-end encryption for CoAP messages. The goal, therefore, is to hide as much of the message as possible while still enabling proxy operation.

Conceptually this is achieved by splitting the CoAP message into an Inner Plaintext and Outer OSCORE Message. The Inner Plaintext contains sensitive information which is not necessary for proxy operation. This, in turn, is the part of the message which can be encrypted until it reaches its end destination. The Outer Message acts as a shell matching the format of a regular CoAP message, and includes all Options and information needed for proxy operation and caching. This decomposition is illustrated in Figure 8.

CoAP options are sorted into one of 3 classes, each granted a specific type of protection by the protocol:

- Class E: Encrypted options moved to the Inner Plaintext,
- Class I: Integrity-protected options included in the AAD for the encryption of the Plaintext but otherwise left untouched in the Outer Message,
- Class U: Unprotected options left untouched in the Outer Message.

Additionally, the OSCORE Option is added as an Outer option, signalling that the message is OSCORE protected. This option carries the information necessary to retrieve the Security Context with which the message was encrypted so that it may be correctly decrypted at the other end-point.

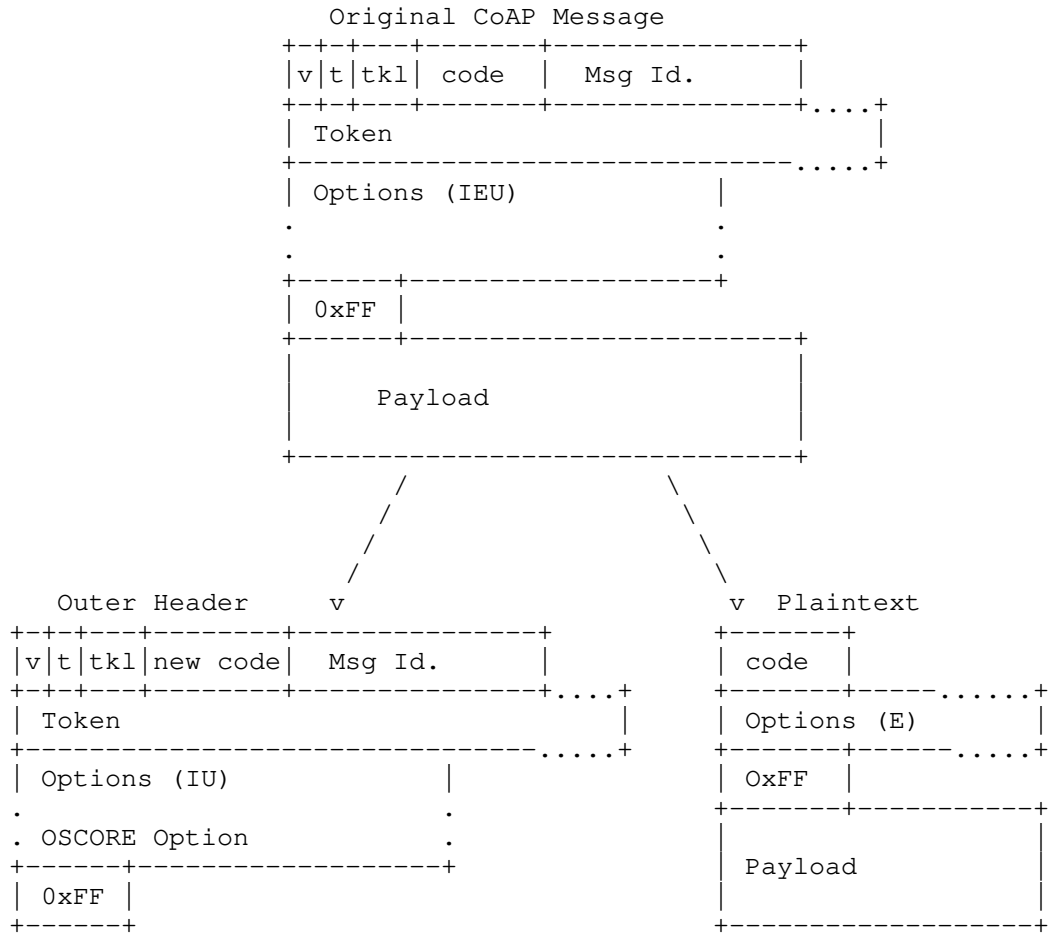


Figure 8: A CoAP message is split into an OSCORE outer and plaintext

Figure 8 shows the message format for the OSCORE Message and Plaintext.

In the Outer Header, the original message code is hidden and replaced by a default dummy value. As seen in Sections 4.1.3.5 and 4.2 of the [rfc8613], the message code is replaced by POST for requests and Changed for responses when Observe is not used. If Observe is used, the message code is replaced by FETCH for requests and Content for responses.

The original message code is put into the first byte of the Plaintext. Following the message code, the class E options comes and

if present the original message Payload is preceded by its payload marker.

The Plaintext is now encrypted by an AEAD algorithm which integrity protects Security Context parameters and eventually any class I options from the Outer Header. Currently no CoAP options are marked class I. The resulting Ciphertext becomes the new Payload of the OSCORE message, as illustrated in Figure 9.

This Ciphertext is, as defined in RFC 5116, the concatenation of the encrypted Plaintext and its authentication tag. Note that Inner Compression only affects the Plaintext before encryption, thus we can only aim to reduce this first, variable length component of the Ciphertext. The authentication tag is fixed in length and considered part of the cost of protection.

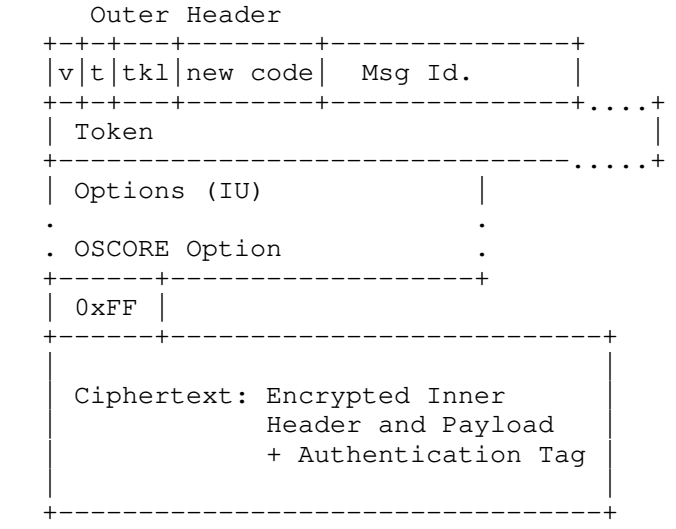


Figure 9: OSCORE message

The SCHC Compression scheme consists of compressing both the Plaintext before encryption and the resulting OSCORE message after encryption, see Figure 10.

This translates into a segmented process where SCHC compression is applied independently in 2 stages, each with its corresponding set of Rules, with the Inner SCHC Rules and the Outer SCHC Rules. This way compression is applied to all fields of the original CoAP message.

Note that since the Inner part of the message can only be decrypted by the corresponding end-point, this end-point will also have to implement Inner SCHC Compression/Decompression.

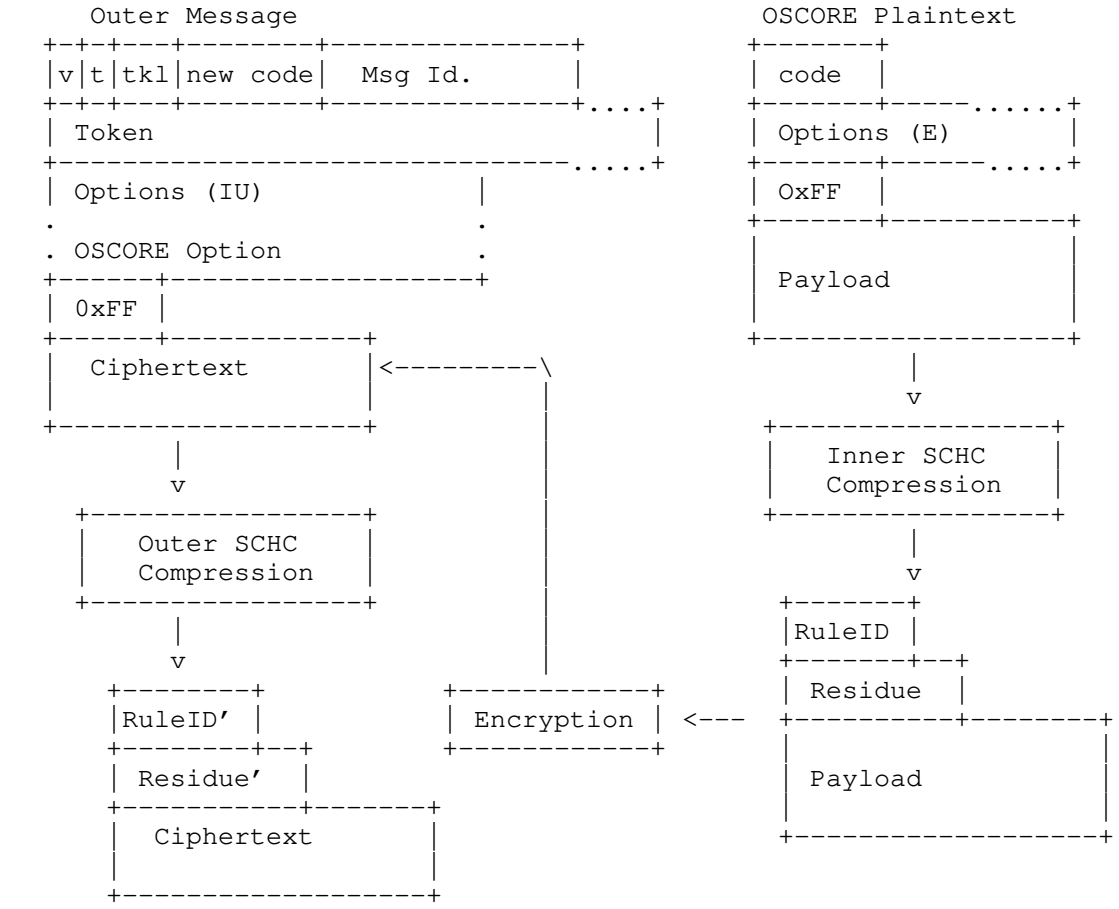


Figure 10: OSCORE Compression Diagram

7.3. Example OSCORE Compression

An example is given with a GET Request and its consequent Content Response from a device-based CoAP client to a cloud-based CoAP server. A possible set of Rules for the Inner and Outer SCHC Compression is shown. A dump of the results and a contrast between SCHC + OSCORE performance with SCHC + COAP performance is also listed. This gives an approximation to the cost of security with SCHC-OSCORE.

Our first example CoAP message is the GET Request in Figure 11

```
Original message:
=================
0x4101000182bb74656d7065726174757265

Header:
0x4101
01   Ver
  00   CON
    0001   tkl
        00000001   Request Code 1 "GET"

0x0001 = mid
0x82 = token

Options:
0xbb74656d7065726174757265
Option 11: URI_PATH
Value = temperature

Original msg length:   17 bytes.
```

Figure 11: CoAP GET Request

Its corresponding response is the CONTENT Response in Figure 12.

```
Original message:
=================
0x6145000182ff32332043

Header:
0x6145
01   Ver
  10   ACK
    0001   tkl
        01000101   Successful Response Code 69 "2.05 Content"

0x0001 = mid
0x82 = token

0xFF  Payload marker
Payload:
0x32332043

Original msg length:   10
```

Figure 12: CoAP CONTENT Response

The SCHC Rules for the Inner Compression include all fields that are already present in a regular CoAP message. The methods described in Section 4 applies to these fields. As an example, see Figure 13.

RuleID 0

Field	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP Code		up	1	equal	not-sent	
CoAP Code		dw	[69,132]	match-map	match-sent	c
CoAP Uri-Path		up	temperature	equal	not-sent	
COAP Option-End		dw	0xFF	equal	not-sent	

Figure 13: Inner SCHC Rules

Figure 14 shows the Plaintext obtained for our example GET Request and follows the process of Inner Compression and Encryption until we end up with the Payload to be added in the outer OSCORE Message.

In this case the original message has no payload and its resulting Plaintext can be compressed up to only 1 byte (size of the RuleID). The AEAD algorithm preserves this length in its first output, but also yields a fixed-size tag which cannot be compressed and has to be included in the OSCORE message. This translates into an overhead in total message length, which limits the amount of compression that can be achieved and plays into the cost of adding security to the exchange.

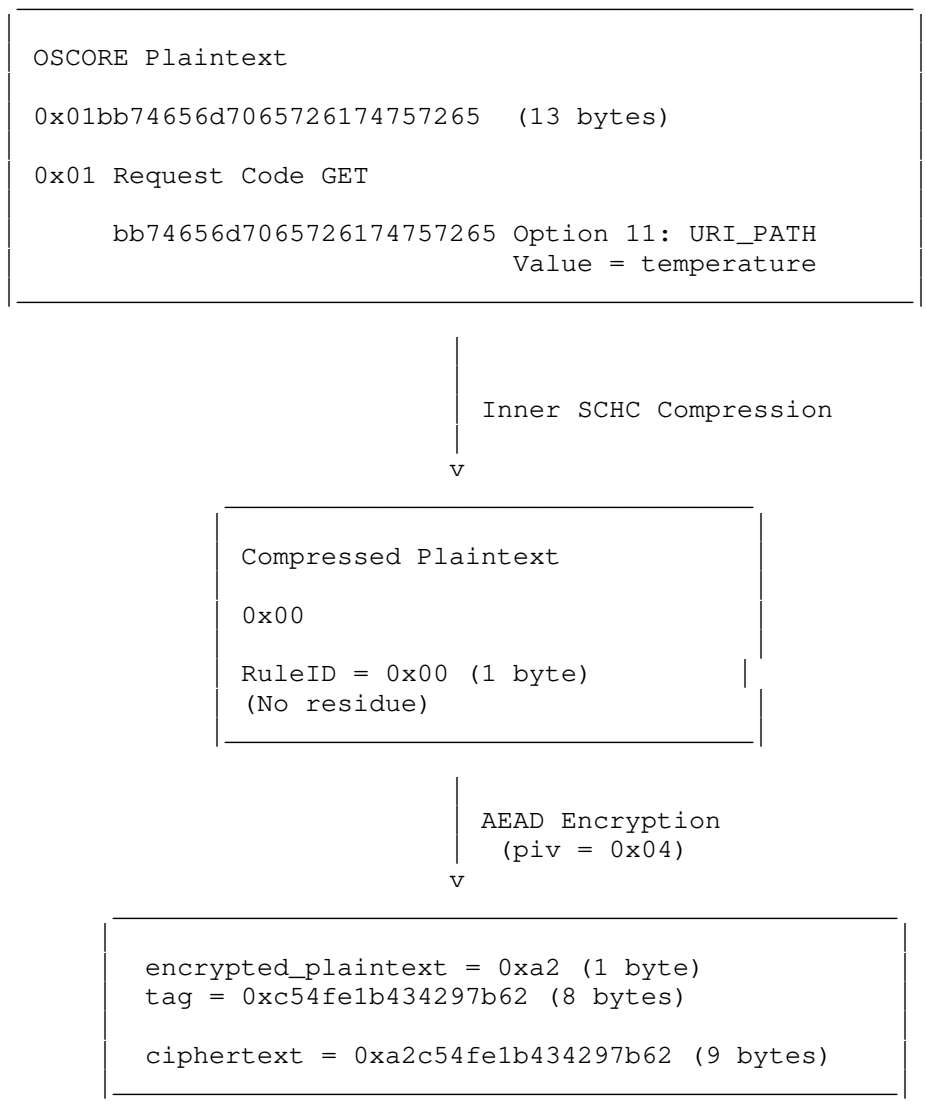


Figure 14: Plaintext compression and encryption for GET Request

In Figure 15 the process is repeated for the example CONTENT Response. The residue is 1 bit long. Note that since SCHC adds padding after the payload, this misalignment causes the hexadecimal code from the payload to differ from the original, even though it has not been compressed.

On top of this, the overhead from the tag bytes is incurred as before.

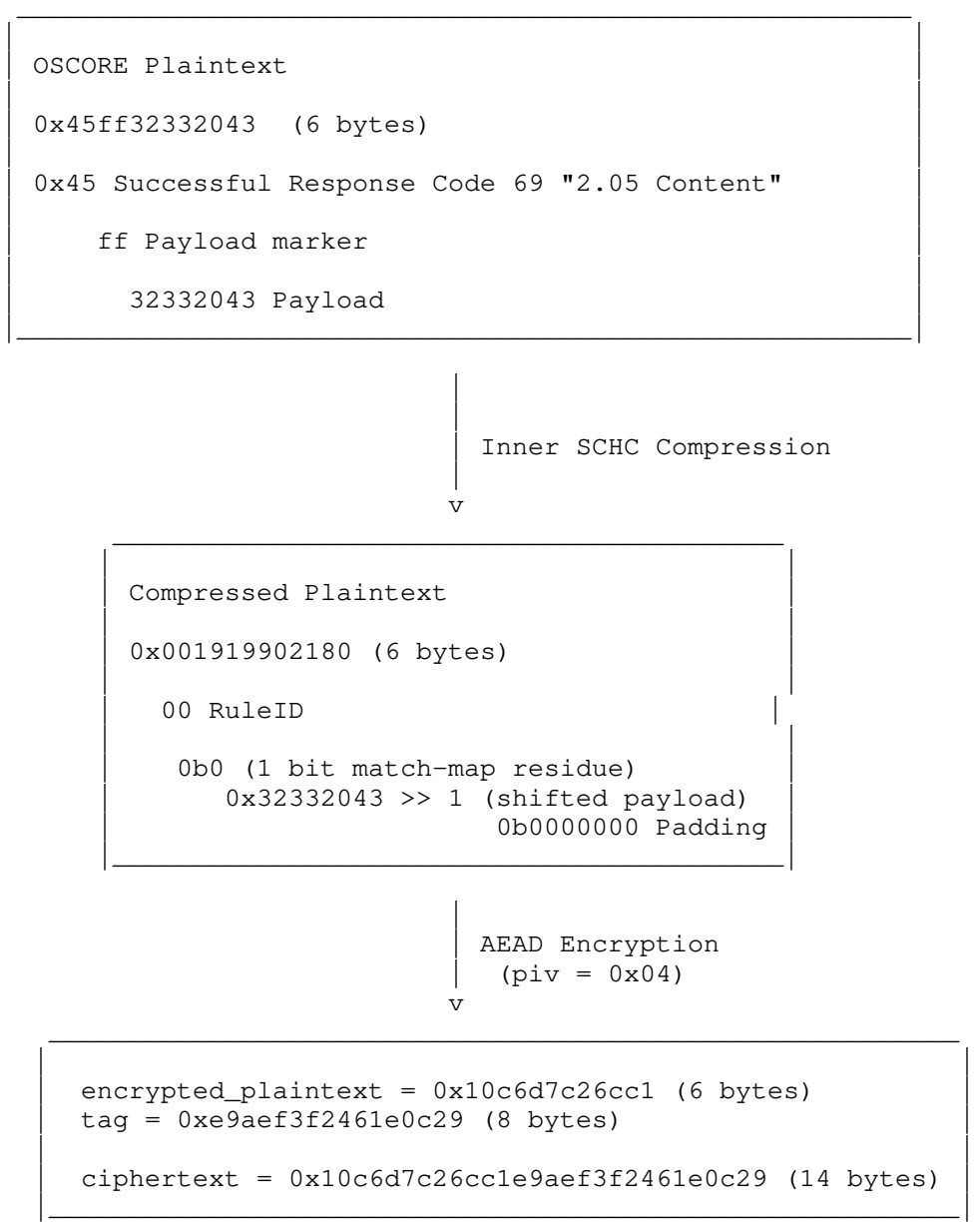


Figure 15: Plaintext compression and encryption for CONTENT Response

The Outer SCHC Rules (Figure 18) must process the OSCORE Options fields. In Figure 16 and Figure 17 we show a dump of the OSCORE Messages generated from our example messages once they have been provided with the Inner Compressed Ciphertext in the payload. These are the messages that have to be compressed by the Outer SCHC Compression.

```
Protected message:
==================
0x4102000182d8080904636c69656e74ffa2c54fe1b434297b62
(25 bytes)

Header:
0x4102
01   Ver
  00   CON
    0001   tkl
        00000010   Request Code 2 "POST"

0x0001 = mid
0x82 = token

Options:
0xd8080904636c69656e74 (10 bytes)
Option 21: OBJECT_SECURITY
Value = 0x0904636c69656e74
          09 = 000 0 1 001 Flag byte
                   h k  n
            04 piv
              636c69656e74 kid


0xFF  Payload marker
Payload:
0xa2c54fe1b434297b62 (9 bytes)
```

Figure 16: Protected and Inner SCHC Compressed GET Request

```
Protected message:
==================
0x6144000182d008ff10c6d7c26cc1e9aef3f2461e0c29
(22 bytes)

Header:
0x6144
01   Ver
  10   ACK
    0001   tkl
        01000100   Successful Response Code 68 "2.04 Changed"

0x0001 = mid
0x82 = token

Options:
0xd008 (2 bytes)
Option 21: OBJECT_SECURITY
Value = b''

0xFF  Payload marker
Payload:
0x10c6d7c26cc1e9aef3f2461e0c29 (14 bytes)
```

Figure 17: Protected and Inner SCHC Compressed CONTENT Response

For the flag bits, a number of compression methods has been shown to be useful depending on the application. The simplest alternative is to provide a fixed value for the flags, combining MO equal and CDA not- sent. This saves most bits but could prevent flexibility. Otherwise, match-mapping could be used to choose from an interested number of configurations to the exchange. Otherwise, MSB could be used to mask off the 3 hard-coded most significant bits.

Note that fixing a flag bit will limit the choice of CoAP Options that can be used in the exchange, since their values are dependent on certain options.

The piv field lends itself to having a number of bits masked off with MO MSB and CDA LSB. This could be useful in applications where the message frequency is low such as that found in LPWAN technologies. Note that compressing the sequence numbers effectively reduces the maximum amount of sequence numbers that can be used in an exchange. Once this amount is exceeded, the OSCORE keys need to be re-established.

The size s included in the kid context field MAY be masked off with CDA MSB. The rest of the field could have additional bits masked

off, or have the whole field be fixed with MO equal and CDA not-sent. The same holds for the kid field.

Figure 18 shows a possible set of Outer Rules to compress the Outer Header.

RuleID 0

Field	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP version		bi	01	equal	not-sent	
CoAP Type		up	0	equal	not-sent	
CoAP Type		dw	2	equal	not-sent	
CoAP TKL		bi	1	equal	not-sent	
CoAP Code		up	2	equal	not-sent	
CoAP Code		dw	68	equal	not-sent	
CoAP MID		bi	0000	MSB(12)	LSB	MMMM
CoAP Token		bi	0x80	MSB(5)	LSB	TTT
CoAP OSCORE_flags		up	0x09	equal	not-sent	
CoAP OSCORE_piv		up	0x00	MSB(4)	LSB	PPPP
COAP OSCORE_kid		up	0x636c69656e70	MSB(52)	LSB	KKKK
COAP OSCORE_kidctxt		bi	b''	equal	not-sent	
CoAP OSCORE_flags		dw	b''	equal	not-sent	
CoAP OSCORE_piv		dw	b''	equal	not-sent	
CoAP OSCORE_kid		dw	b''	equal	not-sent	
COAP Option-End		dw	0xFF	equal	not-sent	

Figure 18: Outer SCHC Rules

These Outer Rules are applied to the example GET Request and CONTENT Response. The resulting messages are shown in Figure 19 and Figure 20.

```
Compressed message:
==================
0x001489458a9fc3686852f6c4 (12 bytes)
0x00 RuleID
    1489 Compression Residue
        458a9fc3686852f6c4 Padded payload

Compression Residue:
0b 0001 010 0100 0100 (15 bits -> 2 bytes with padding)
    mid tkn piv  kid

Payload
0xa2c54fe1b434297b62 (9 bytes)

Compressed message length: 12 bytes
```

Figure 19: SCHC-OSCORE Compressed GET Request

```
Compressed message:
==================
0x0014218daf84d983d35de7e48c3c1852 (16 bytes)
0x00 RuleID
    14 Compression Residue
      218daf84d983d35de7e48c3c1852 Padded payload
Compression Residue:
0b0001 010 (7 bits -> 1 byte with padding)
  mid  tkn

Payload
0x10c6d7c26cc1e9aef3f2461e0c29 (14 bytes)

Compressed msg length: 16 bytes
```

Figure 20: SCHC-OSCORE Compressed CONTENT Response

For contrast, we compare these results with what would be obtained by SCHC compressing the original CoAP messages without protecting them with OSCORE. To do this, we compress the CoAP messages according to the SCHC Rules in Figure 21.

RuleID 1

Field	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP version		bi	01	equal	not-sent	
CoAP Type		up	0	equal	not-sent	
CoAP Type		dw	2	equal	not-sent	
CoAP TKL		bi	1	equal	not-sent	
CoAP Code		up	2	equal	not-sent	
CoAP Code		dw	[69,132]	match-map	map-sent	C
CoAP MID		bi	0000	MSB(12)	LSB	MMMM
CoAP Token		bi	0x80	MSB(5)	LSB	TTT
CoAP Uri-Path		up	temperature	equal	not-sent	
COAP Option-End		dw	0xFF	equal	not-sent	

Figure 21: SCHC-CoAP Rules (No OSCORE)

This yields the results in Figure 22 for the Request, and Figure 23 for the Response.

Compressed message:
==========
0x0114
0x01 = RuleID

Compression Residue:
0b00010100 (1 byte)

Compressed msg length: 2

Figure 22: CoAP GET Compressed without OSCORE

```
Compressed message:
==================
0x010a32332043
0x01 = RuleID

Compression Residue:
0b00001010 (1 byte)

Payload
0x32332043

Compressed msg length: 6
```

Figure 23: CoAP CONTENT Compressed without OSCORE

As can be seen, the difference between applying SCHC + OSCORE as compared to regular SCHC + COAP is about 10 bytes of cost.

8. IANA Considerations

This document has no request to IANA.

9. Security considerations

When applied to LPWAN, the Security Considerations of SCHC header compression [rfc8724] are valid for SCHC CoAP header compression. When CoAP uses OSCORE, the security considerations defined in [rfc8613] does not change when SCHC header compression is applied.

The definition of SCHC over CoAP header fields permits the compression of header information only. The SCHC header compression itself does not increase or reduce the level of security in the communication. When the connection does not use any security protocol as OSCORE, DTLS, or other, it is highly necessary to use a layer two security.

DoS attacks are possible if an intruder can introduce a compressed SCHC corrupted packet onto the link and cause a compression efficiency reduction. However, an intruder having the ability to add corrupted packets at the link layer raises additional security issues than those related to the use of header compression.

SCHC compression returns variable-length Residues for some CoAP fields. In the compressed header, the length sent is not the original header field length but the length of the Residue. So if a corrupted packet comes to the decompressor with a longer or shorter

length than the one in the original header, SCHC decompression will detect an error and drops the packet.

OSCORE compression is also based on the same compression method described in [rfc8724]. The size of the Initialisation Vector (IV) residue must be considered carefully. A residue size obtained with LSB CDA over the IV has an impact on the compression efficiency and the frequency the device will renew its key. This operation requires several exchanges and is energy-consuming.

SCHC header and compression Rules MUST remain tightly coupled. Otherwise, an encrypted residue may be decompressed differently by the receiver. To avoid this situation, if the Rule is modified in one location, the OSCORE keys MUST be re-established.

10. Acknowledgements

The authors would like to thank (in alphabetic order): Christian Amsuss, Dominique Barthel, Carsten Bormann, Theresa Enghardt, Thomas Fossati, Klaus Hartke, Francesca Palombini, Alexander Pelov and Goran Selander.

11. 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>.

[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>.

[rfc7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <https://www.rfc-editor.org/info/rfc7641>.

[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>.

[rfc7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <https://www.rfc-editor.org/info/rfc7967>.

[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>.

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

Ricardo Andreasen
Universidad de Buenos Aires
Av. Paseo Colon 850
C1063ACV Ciudad Autonoma de Buenos Aires
Argentina

Email: randreasen@fi.uba.ar

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 29, 2021

O. Gimenez, Ed.
Semtech
I. Petrov, Ed.
Acklio

January 25, 2021

Static Context Header Compression (SCHC) over LoRaWAN draft-ietf-lpwan-schc-over-lorawan-14

Abstract

The Static Context Header Compression (SCHC) specification describes generic header compression and fragmentation techniques for Low Power Wide Area Networks (LPWAN) technologies. SCHC is a generic mechanism designed for great flexibility so that it can be adapted for any of the LPWAN technologies.

This document specifies a profile of RFC8724 to use SCHC in LoRaWAN(R) networks, and provides elements such as efficient parameterization and modes of operation.

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 July 29, 2021.

Copyright Notice

Table of Contents

1. Introduction

SCHC specification [RFC8724] describes generic header compression and fragmentation techniques that can be used on all Low Power Wide Area Networks (LPWAN) technologies defined in [RFC8376]. Even though those technologies share a great number of common features like star-oriented topologies, network architecture, devices with mostly quite predictable communications, etc; they do have some slight differences with respect to payload sizes, reactiveness, etc.

SCHC provides a generic framework that enables those devices to communicate on IP networks. However, for efficient performance, some parameters and modes of operation need to be set appropriately for each of the LPWAN technologies.

This document describes the parameters and modes of operation when SCHC is used over LoRaWAN networks. LoRaWAN protocol is specified by the LoRa Alliance(R) in [lora-alliance-spec]

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.

This section defines the terminology and acronyms used in this document. For all other definitions, please look up the SCHC specification [RFC8724].

- DevEUI: Device Extended Unique Identifier, an IEEE EUI-64 identifier used to identify the device during the procedure while joining the network (Join Procedure). It is assigned by the manufacturer or the device owner and provisioned on the Network Gateway.

- DevAddr: a 32-bit non-unique identifier assigned to a device either:

  * Statically: by the device manufacturer in _Activation by Personalization_ mode.

  * Dynamically: after a Join Procedure by the Network Gateway in _Over The Air Activation_ mode.

- Downlink: LoRaWAN term for a frame transmitted by the network and received by the device.

- EUI: Extended Unique Identifier
- LoRaWAN: LoRaWAN is a wireless technology based on Industrial, Scientific, and Medical (ISM) radio bands that is used for long-range, low-power, low-data-rate applications developed by the LoRa Alliance, a membership consortium: https://www.lora-alliance.org [1].
- FRMPayload: Application data in a LoRaWAN frame.
- MSB: Most Significant Byte
- OUI: Organisation Unique Identifier. IEEE assigned prefix for EUI.
- RCS: Reassembly Check Sequence. Used to verify the integrity of the fragmentation-reassembly process.
- RX: Device's reception window.
- RX1/RX2: LoRaWAN class A devices open two RX windows following an uplink, called RX1 and RX2.
- SCHC gateway: The LoRaWAN Application Server that manages translation between IPv6 network and the Network Gateway (LoRaWAN Network Server).
- Tile: Piece of a fragmented packet as described in [RFC8724] section 8.2.2.1
- Uplink: LoRaWAN term for a frame transmitted by the device and received by the network.

3. Static Context Header Compression Overview

This section contains a short overview of SCHC. For a detailed description, refer to the full specification [RFC8724].

It defines:

1. Compression mechanisms to avoid transporting information known by both sender and receiver over the air. Known information is part of the "context". This component is called SCHC Compressor/Decompressor (SCHC C/D).
2. Fragmentation mechanisms to allow SCHC Packet transportation on small, and potentially variable, MTU. This component is called SCHC Fragmentation/Reassembly (SCHC F/R).

Context exchange or pre-provisioning is out of scope of this document.

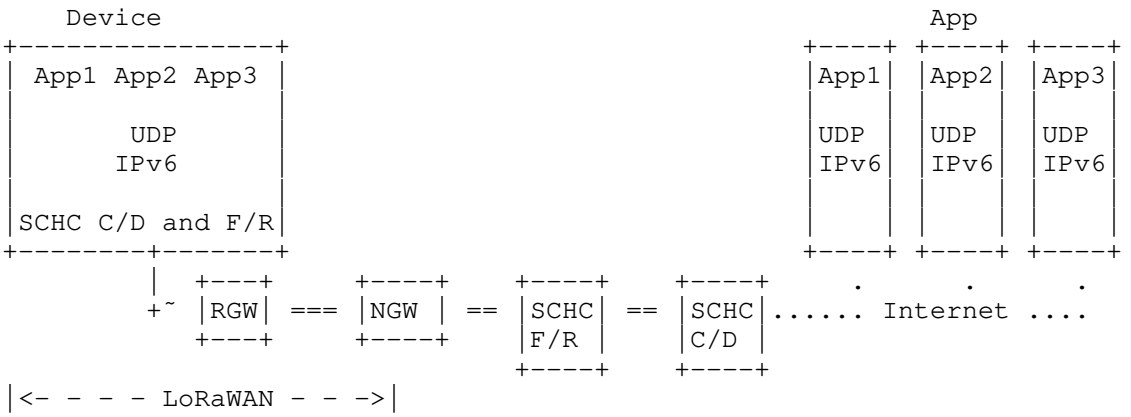


Figure 1: Architecture

Figure 1 represents the architecture for compression/decompression, it is based on [RFC8376] terminology. The device is sending applications flows using IPv6 or IPv6/UDP protocols. These flows might be compressed by a Static Context Header Compression Compressor/Decompressor (SCHC C/D) to reduce headers size and fragmented by the SCHC Fragmentation/Reassembly (SCHC F/R). The resulting information is sent on a layer two (L2) frame to an LPWAN Radio Gateway (RGW) that forwards the frame to a Network Gateway (NGW). The NGW sends the data to a SCHC F/R for reassembly, if required, then to SCHC C/D for decompression. The SCHC C/D shares the same rules with the device. The SCHC C/D and F/R can be located on the Network Gateway (NGW) or in another place as long as a communication is established between the NGW and the SCHC F/R, then SCHC F/R and C/D. The SCHC C/D and F/R in the device and the SCHC gateway MUST share the same set of rules. After decompression, the packet can be sent on the Internet to one or several LPWAN Application Servers (App).

The SCHC C/D and F/R process is bidirectional, so the same principles can be applied to the other direction.

In a LoRaWAN network, the RGW is called a Gateway, the NGW is Network Server, and the SCHC C/D and F/R are an Application Server. It can be provided by the Network Gateway or any third party software. Figure 1 can be mapped in LoRaWAN terminology to:

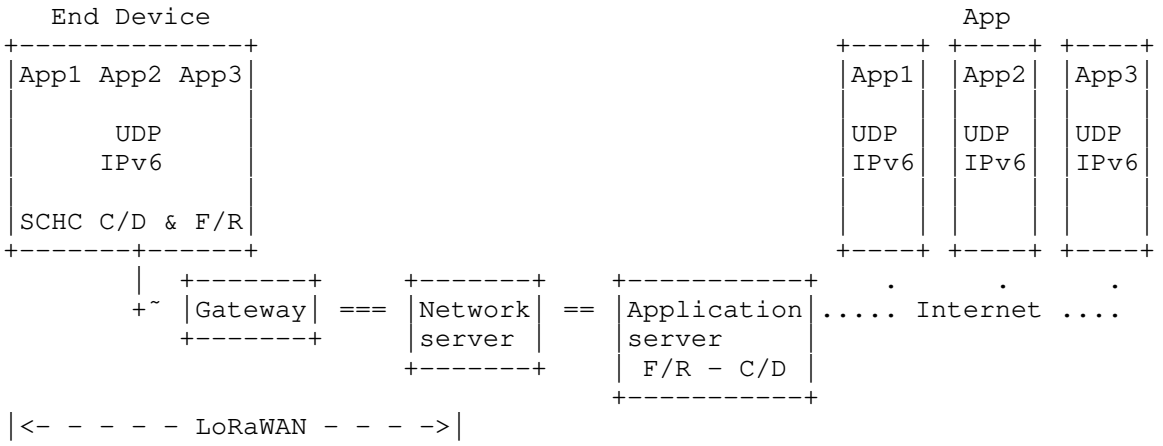


Figure 2: SCHC Architecture mapped to LoRaWAN

4. LoRaWAN Architecture

An overview of LoRaWAN [lora-alliance-spec] protocol and architecture is described in [RFC8376]. The mapping between the LPWAN architecture entities as described in [RFC8724] and the ones in [lora-alliance-spec] is as follows:

o Devices are LoRaWAN End Devices (e.g. sensors, actuators, etc.). There can be a very high density of devices per radio gateway (LoRaWAN gateway). This entity maps to the LoRaWAN end-device.

o The Radio Gateway (RGW), which is the endpoint of the constrained link. This entity maps to the LoRaWAN Gateway.

o The Network Gateway (NGW) is the interconnection node between the Radio Gateway and the SCHC gateway (LoRaWAN Application server). This entity maps to the LoRaWAN Network Server.

o SCHC C/D and F/R are handled by LoRaWAN Application Server; ie the LoRaWAN application server will do the SCHC C/D and F/R.

o The LPWAN-AAA Server is the LoRaWAN Join Server. Its role is to manage and deliver security keys in a secure way, so that the devices root key is never exposed.

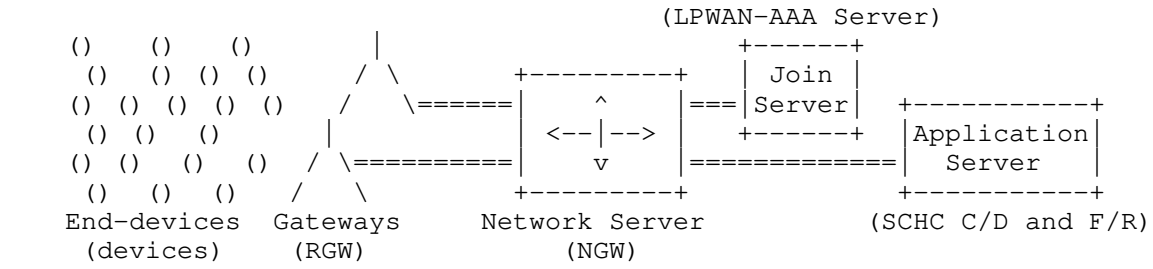


Figure 3: LPWAN Architecture

Note: Figure 3 terms are from LoRaWAN, with [RFC8376] terminology in brackets.

SCHC Compressor/Decompressor (SCHC C/D) and SCHC Fragmentation/ Reassembly (SCHC F/R) are performed on the LoRaWAN end-device and the Application Server (called SCHC gateway). While the point-to-point link between the device and the Application Server constitutes a single IP hop, the ultimate end-point of the IP communication may be an Internet node beyond the Application Server. In other words, the LoRaWAN Application Server (SCHC gateway) acts as the first hop IP router for the device. The Application Server and Network Server may be co-located, which effectively turns the Network/Application Server into the first hop IP router.

4.1. Device classes (A, B, C) and interactions

The LoRaWAN MAC layer supports 3 classes of devices named A, B and C. All devices implement the Class A, some devices may implement Class B or Class C. Class B and Class C are mutually exclusive.

- Class A: The Class A is the simplest class of devices. The device is allowed to transmit at any time, randomly selecting a communication channel. The Network Gateway may reply with a downlink in one of the 2 receive windows immediately following the uplinks. Therefore, the Network Gateway cannot initiate a downlink, it has to wait for the next uplink from the device to get a downlink opportunity. The Class A is the lowest power consumption class.

- Class B: Class B devices implement all the functionalities of Class A devices, but also schedule periodic listen windows. Therefore, opposed to the Class A devices, Class B devices can receive downlinks that are initiated by the Network Gateway and not following an uplink. There is a trade-off between the periodicity of those scheduled Class B listen windows and the power consumption of the device: if the periodicity is high

downlinks from the NGW will be sent faster, but the device wakes up more often: it will have higher power consumption.

- Class C: Class C devices implement all the functionalities of Class A devices, but keep their receiver open whenever they are not transmitting. Class C devices can receive downlinks at any time at the expense of a higher power consumption. Battery-powered devices can only operate in Class C for a limited amount of time (for example for a firmware upgrade over-the-air). Most of the Class C devices are grid powered (for example Smart Plugs).

4.2. Device addressing

LoRaWAN end-devices use a 32-bit network address (devAddr) to communicate with the Network Gateway over-the-air, this address might not be unique in a LoRaWAN network. Devices using the same devAddr are distinguished by the Network Gateway based on the cryptographic signature appended to every LoRaWAN frame.

To communicate with the SCHC gateway, the Network Gateway MUST identify the devices by a unique 64-bit device identifier called the DevEUI.

The DevEUI is assigned to the device during the manufacturing process by the device's manufacturer. It is built like an Ethernet MAC address by concatenating the manufacturer's IEEE OUI field with a vendor unique number. e.g.: 24-bit OUI is concatenated with a 40-bit serial number. The Network Gateway translates the devAddr into a DevEUI in the uplink direction and reciprocally on the downlink direction.

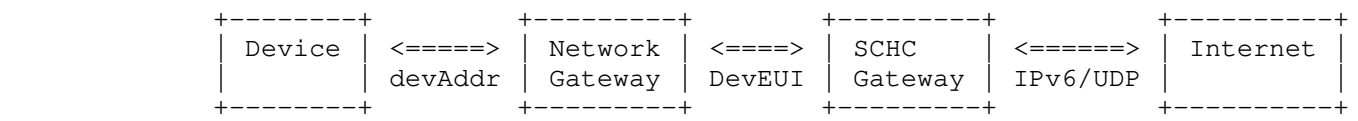


Figure 4: LoRaWAN addresses

4.3. General Frame Types

LoRaWAN implements the possibility to send confirmed or unconfirmed frames:

- Confirmed frame: The sender asks the receiver to acknowledge the frame.

- Unconfirmed frame: The sender does not ask the receiver to acknowledge the frame.

As SCHC defines its own acknowledgment mechanisms, SCHC does not require the use of LoRaWAN Confirmed frames (MType=0b100 as per [lora-alliance-spec])

4.4. LoRaWAN MAC Frames

In addition to regular data frames, LoRaWAN implements JoinRequest and JoinAccept frame types, which are used by a device to join a network:

- JoinRequest: This frame is used by a device to join a network. It contains the device's unique identifier DevEUI and a random nonce that will be used for session key derivation.

- JoinAccept: To on-board a device, the Network Gateway responds to the JoinRequest issued by a device with a JoinAccept frame. That frame is encrypted with the device's AppKey and contains (amongst other fields) the network's major settings and a random nonce used to derive the session keys.

- Data: MAC and application data. Application data are protected with AES-128 encryption. MAC related data are AES-128 encrypted with another key.

4.5. LoRaWAN FPort

The LoRaWAN MAC layer features a frame port field in all frames. This field (FPort) is 8 bits long and the values from 1 to 223 can be used. It allows LoRaWAN networks and applications to identify data.

4.6. LoRaWAN empty frame

A LoRaWAN empty frame is a LoRaWAN frame without FPort (cf Section 5.1) and FRMPayload.

4.7. Unicast and multicast technology

LoRaWAN technology supports unicast downlinks, but also multicast: a packet sent over LoRaWAN radio link can be received by several devices. It is useful to address many devices with same content, either a large binary file (firmware upgrade), or same command (e.g: lighting control). As IPv6 is also a multicast technology this feature can be used to address a group of devices.

Note 1: IPv6 multicast addresses must be defined as per [RFC4291]. LoRaWAN multicast group definition in a Network Gateway and the relation between those groups and IPv6 groupID are out of scope of this document.

Note 2: LoRa Alliance defined [lora-alliance-remote-multicast-set] as the RECOMMENDED way to setup multicast groups on devices and create a synchronized reception window.

5. SCHC-over-LoRaWAN

5.1. LoRaWAN FPort and RuleID

The FPort field is part of the SCHC Message, as shown in Figure 5. The SCHC C/D and the SCHC F/R SHALL concatenate the FPort field with the LoRaWAN payload to recompose the SCHC Message.

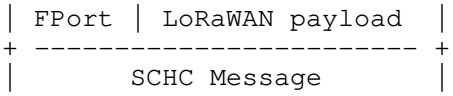


Figure 5: SCHC Message in LoRaWAN

Note: SCHC Message is any datagram sent by SCHC C/D or F/R layers.

A fragmented datagram with application payload transferred from device to Network Gateway, is called an uplink fragmented datagram. It uses an FPort for data uplink and its associated SCHC control downlinks, named FPortUp in this document. The other way, a fragmented datagram with application payload transferred from Network Gateway to device, is called downlink fragmented datagram. It uses another FPort for data downlink and its associated SCHC control uplinks, named FPortDown in this document.

All RuleID can use arbitrary values inside the FPort range allowed by LoRaWAN specification and MUST be shared by the device and SCHC gateway prior to the communication with the selected rule. The uplink and downlink fragmentation FPorts MUST be different.

5.2. Rule ID management

RuleID MUST be 8 bits, encoded in the LoRaWAN FPort as described in Section 5.1. LoRaWAN supports up to 223 application FPorts in the range [1;223] as defined in section 4.3.2 of [lora-alliance-spec], it implies that RuleID MSB SHOULD be inside this range. An application can send non SCHC traffic by using FPort values different from the ones used for SCHC.

In order to improve interoperability, RECOMMENDED fragmentation RuleID values are:

- RuleID = 20 (8-bit) for uplink fragmentation, named FPortUp.
- RuleID = 21 (8-bit) for downlink fragmentation, named FPortDown.
- RuleID = 22 (8-bit) for which SCHC compression was not possible (i.e., no matching compression Rule was found), as described in [RFC8724] section 6.

FPortUp value MUST be different from FPortDown. The remaining RuleIDs are available for compression. RuleIDs are shared between uplink and downlink sessions. A RuleID not in the set(s) of FPortUp or FPortDown means that the fragmentation is not used, thus, on reception, the SCHC Message MUST be sent to the SCHC C/D layer.

The only uplink frames using the FPortDown port are the fragmentation SCHC control messages of a downlink fragmented datagram (for example, SCHC ACKs). Similarly, the only downlink frames using the FPortUp port are the fragmentation SCHC control messages of an uplink fragmented datagram.

An application can have multiple fragmented datagrams between a device and one or several SCHC gateways. A set of FPort values is REQUIRED for each SCHC gateway instance the device is required to communicate with. The application can use additional uplinks or downlink fragmented parameters but SHALL implement at least the parameters defined in this document.

The mechanism for context distribution across devices and gateways is outside the scope of this document.

5.3. Interface IDentifier (IID) computation

In order to mitigate the risks described in [RFC8064] and [RFC8065], implementation MUST implement the following algorithm and SHOULD use it.

1. key = LoRaWAN AppSKey
2. cmac = aes128_cmac(key, DevEUI)
3. IID = cmac[0..7]

aes128_cmac algorithm is described in [RFC4493]. It has been chosen as it is already used by devices for LoRaWAN protocol.

As AppSKey is renewed each time a device joins or rejoins a LoRaWAN network, the IID will change over time; this mitigates privacy, location tracking and correlation over time risks. Join periodicity is defined at the application level.

Address scan risk is mitigated thanks to AES-128, which provides enough entropy bits of the IID.

Using this algorithm will also ensure that there is no correlation between the hardware identifier (IEEE-64 DevEUI) and the IID, so an attacker cannot use manufacturer OUI to target devices.

Example with:

- DevEUI: 0x1122334455667788

- appSKey: 0x00AABBCCDDEEFF00AABBCCDDEEFFAABB

1. key: 0x00AABBCCDDEEFF00AABBCCDDEEFFAABB
2. cmac: 0xBA59F4B196C6C3432D9383C145AD412A
3. IID: 0xBA59F4B196C6C343

Figure 6: Example of IID computation.

There is a small probability of IID collision in a LoRaWAN network. If this occurs, the IID can be changed by rekeying the device at L2 level (ie: trigger a LoRaWAN join). The way the device is rekeyed is out of scope of this document and left to the implementation.

Note: Implementation also using another IID source MUST ensure that the same IID is shared between the device and the SCHC gateway in the compression and decompression of the IPv6 address of the device.

5.4. Padding

All padding bits MUST be 0.

5.5. Decompression

SCHC C/D MUST concatenate FPort and LoRaWAN payload to retrieve the SCHC Packet as per Section 5.1.

RuleIDs matching FPortUp and FPortDown are reserved for SCHC Fragmentation.

5.6. Fragmentation

The L2 Word Size used by LoRaWAN is 1 byte (8 bits). The SCHC fragmentation over LoRaWAN uses the ACK-on-Error mode for uplink fragmentation and Ack-Always mode for downlink fragmentation. A LoRaWAN device cannot support simultaneous interleaved fragmented datagrams in the same direction (uplink or downlink).

The fragmentation parameters are different for uplink and downlink fragmented datagrams and are successively described in the next sections.

5.6.1. DTag

[RFC8724] section 8.2.4 describes the possibility to interleave several fragmented SCHC datagrams for the same RuleID. This is not used in SCHC over LoRaWAN profile. A device cannot interleave several fragmented SCHC datagrams on the same FPort. This field is not used and its size is 0.

Note: The device can still have several parallel fragmented datagrams with more than one SCHC gateway thanks to distinct sets of FPorts, cf Section 5.2.

5.6.2. Uplink fragmentation: From device to SCHC gateway

In this case, the device is the fragment transmitter, and the SCHC gateway the fragment receiver. A single fragmentation rule is defined. SCHC F/R MUST concatenate FPort and LoRaWAN payload to retrieve the SCHC Packet, as per Section 5.1.

- SCHC fragmentation reliability mode: "ACK-on-Error".
- SCHC header size is two bytes (the FPort byte + 1 additional byte).
- RuleID: 8 bits stored in LoRaWAN FPort. cf Section 5.2
- DTag: Size T=0 bit, not used. cf Section 5.6.1
- Window index: 4 windows are used, encoded on M = 2 bits
- FCN: The FCN field is encoded on N = 6 bits, so WINDOW_SIZE = 63 tiles are allowed in a window.
- Last tile: it can be carried in a Regular SCHC Fragment, alone in an All-1 SCHC Fragment or with any of these two methods. Implementation must ensure that:

  * The sender MUST ascertain that the receiver will not receive the last tile through both a Regular SCHC Fragment and an All-1 SCHC Fragment during the same session.

  * If the last tile is in All-1 SCHC message: current L2 MTU MUST be big enough to fit the All-1 header and the last tile.

- Penultimate tile MUST be equal to the regular size.

- RCS: Use recommended calculation algorithm in [RFC8724] (S.8.2.3. Integrity Checking).

- Tile: size is 10 bytes.

- Retransmission timer: Set by the implementation depending on the application requirements. The default RECOMMENDED duration of this timer is 12 hours; this value is mainly driven by application requirements and MAY be changed by the application.

- Inactivity timer: The SCHC gateway implements an "inactivity timer". The default RECOMMENDED duration of this timer is 12 hours; this value is mainly driven by application requirements and MAY be changed by the application.

- MAX_ACK_REQUESTS: 8. With this set of parameters, the SCHC fragment header is 16 bits, including FPort; payload overhead will be 8 bits as FPort is already a part of LoRaWAN payload. MTU is: _4 windows * 63 tiles * 10 bytes per tile = 2520 bytes_

In addition to the per-rule context parameters specified in [RFC8724], for uplink rules, an additional context parameter is added: whether or not to ack after each window.
For battery powered devices, it is RECOMMENDED to use the ACK mechanism at the end of each window instead of waiting until the end of all windows:

- The SCHC receiver SHOULD send a SCHC ACK after every window even if there is no missing tile.

- The SCHC sender SHOULD wait for the SCHC ACK from the SCHC receiver before sending tiles from the next window. If the SCHC ACK is not received, it SHOULD send a SCHC ACK REQ up to MAX_ACK_REQUESTS times, as described previously.

This will avoid useless uplinks if the device has lost network coverage.

For non-battery powered devices, the SCHC receiver MAY also choose to send a SCHC ACK only at the end of all windows. This will reduce downlink load on the LoRaWAN network, by reducing the number of downlinks.

SCHC implementations MUST be compatible with both behaviors, and this selection is part of the rule context.

5.6.2.1. Regular fragments

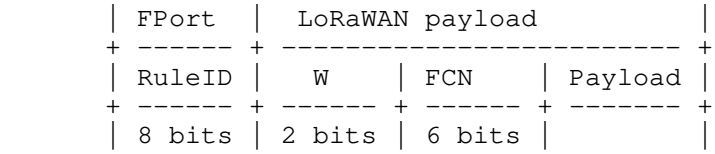


Figure 7: All fragments except the last one. SCHC header size is 16 bits, including LoRaWAN FPort.

5.6.2.2. Last fragment (All-1)

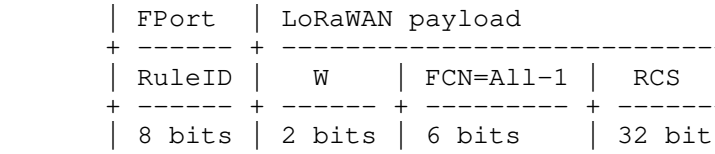


Figure 8: All-1 SCHC Message: the last fragment without last tile.

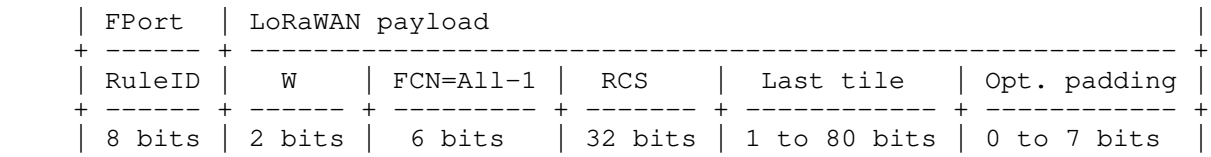


Figure 9: All-1 SCHC Message: the last fragment with last tile.

5.6.2.3. SCHC ACK

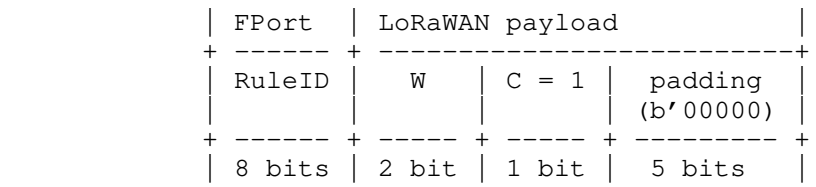


Figure 10: SCHC ACK format, correct RCS check.

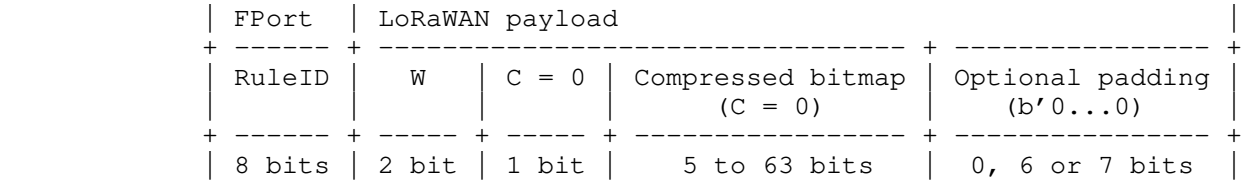


Figure 11: SCHC ACK format, failed RCS check.

Note: Because of the bitmap compression mechanism and L2 byte alignment, only the following discrete values are possible for the compressed bitmap size: 5, 13, 21, 29, 37, 45, 53, 61, 62 and 63. Bitmaps of 63 bits will require 6 bits of padding.

5.6.2.4. Receiver-Abort

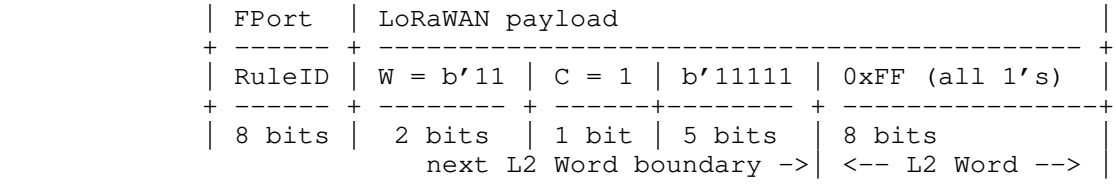


Figure 12: Receiver-Abort format.

5.6.2.5. SCHC acknowledge request

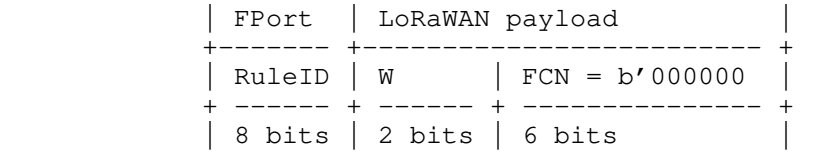


Figure 13: SCHC ACK REQ format.

5.6.3. Downlink fragmentation: From SCHC gateway to device

In this case, the device is the fragmentation receiver, and the SCHC gateway the fragmentation transmitter. The following fields are common to all devices. SCHC F/R MUST concatenate FPort and LoRaWAN payload to retrieve the SCHC Packet as described in Section 5.1.

- SCHC fragmentation reliability mode:
  - Unicast downlinks: ACK-Always.
  - Multicast downlinks: No-ACK, reliability has to be ensured by the upper layer. This feature is OPTIONAL and may not be implemented by SCHC gateway.
- RuleID: 8 bits stored in LoRaWAN FPort. cf Section 5.2
- DTag: Size T=0 bit, not used. cf Section 5.6.1
- FCN: The FCN field is encoded on N=1 bit, so WINDOW_SIZE = 1 tile.
- RCS: Use recommended calculation algorithm in [RFC8724] (S.8.2.3. Integrity Checking).
- Inactivity timer: The default RECOMMENDED duration of this timer is 12 hours; this value is mainly driven by application requirements and MAY be changed by the application.

The following parameters apply to ACK-Always (Unicast) only:

- Retransmission timer: See Section 5.6.3.5.
- MAX_ACK_REQUESTS: 8.
- Window index (unicast only): encoded on M=1 bit, as per [RFC8724].

As only 1 tile is used, its size can change for each downlink, and will be the currently available MTU.

Class A devices can only receive during an RX slot, following the transmission of an uplink. Therefore the SCHC gateway cannot initiate communication (e.g., start a new SCHC session). In order to create a downlink opportunity it is RECOMMENDED for Class A devices to send an uplink every 24 hours when no SCHC session is started, this is application specific and can be disabled. The RECOMMENDED uplink is a LoRaWAN empty frame as defined Section 4.6. As this uplink is to open an RX window, any LoRaWAN uplink frame from the device MAY reset this counter.

Note: The Fpending bit included in LoRaWAN protocol SHOULD NOT be used for SCHC-over-LoRaWAN protocol. It might be set by the Network Gateway for other purposes but not SCHC needs.

5.6.3.1. Regular fragments

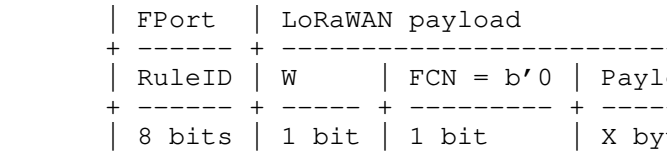


Figure 14: All fragments but the last one. Header size 10 bits, including LoRaWAN FPort.

5.6.3.2. Last fragment (All-1)

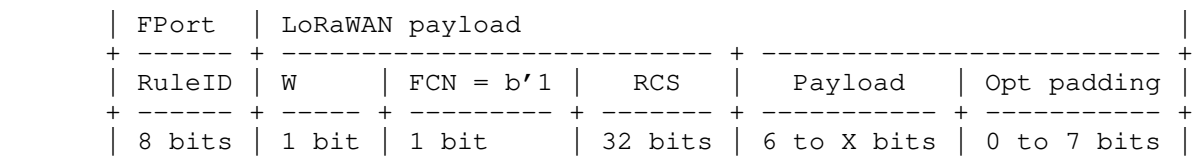


Figure 15: All-1 SCHC Message: the last fragment.

5.6.3.3. SCHC ACK

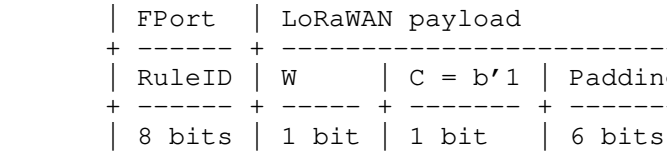


Figure 16: SCHC ACK format, RCS is correct.

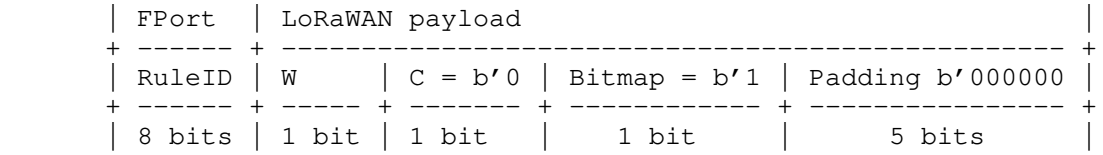


Figure 17: SCHC ACK format, RCS is incorrect.

5.6.3.4. Receiver-Abort

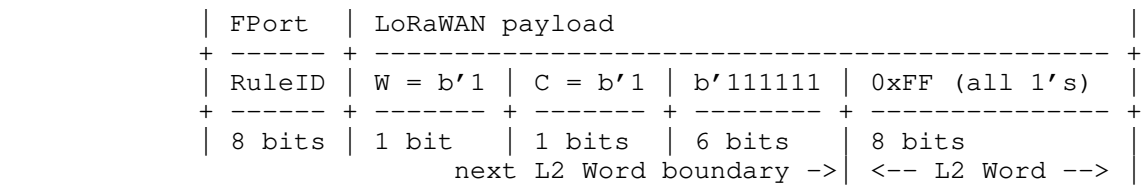


Figure 18: Receiver-Abort packet (following an All-1 SCHC Fragment with incorrect RCS).

5.6.3.5. Downlink retransmission timer

Class A and Class B or Class C devices do not manage retransmissions and timers the same way.

5.6.3.5.1. Class A devices

Class A devices can only receive in an RX slot following the transmission of an uplink.

The SCHC gateway implements an inactivity timer with a RECOMMENDED duration of 36 hours. For devices with very low transmission rates (example 1 packet a day in normal operation), that duration may be extended: it is application specific.

RETRANSMISSION_TIMER is application specific and its RECOMMENDED value is INACTIVITY_TIMER/(MAX_ACK_REQUESTS + 1).

SCHC All-0 (FCN=0)

All fragments but the last have an FCN=0 (because window size is 1). Following an All-0 SCHC Fragment, the device MUST transmit the SCHC ACK message. It MUST transmit up to MAX_ACK_REQUESTS SCHC ACK messages before aborting. In order to progress the fragmented datagram, the SCHC layer should immediately queue for transmission those SCHC ACK if no SCHC downlink have been received during RX1 and RX2 window. LoRaWAN layer will respect the applicable local spectrum regulation.

Note: The ACK bitmap is 1 bit long and is always 1.

SCHC All-1 (FCN=1)

SCHC All-1 is the last fragment of a datagram, the corresponding SCHC ACK message might be lost; therefore the SCHC gateway MUST request a retransmission of this ACK when the retransmission timer expires. To open a downlink opportunity the device MUST transmit an uplink every RETRANSMISSION_TIMER/(MAX_ACK_REQUESTS * SCHC_ACK_REQ_DN_OPPORTUNITY). The format of this uplink is application specific. It is RECOMMENDED for a device to send an empty frame (see Section 4.6) but it is application specific and will be used by the NGW to transmit a potential SCHC ACK REQ. SCHC_ACK_REQ_DN_OPPORTUNITY is application specific and its recommended value is 2. It MUST be greater than 1. This allows to open a downlink opportunity to any downlink with higher priority than the SCHC ACK REQ message.

Note: The device MUST keep this SCHC ACK message in memory until it receives a downlink SCHC Fragmentation Message (with FPort == FPortDown) that is not a SCHC ACK REQ: it indicates that the SCHC gateway has received the SCHC ACK message.

5.6.3.6. Class B or Class C devices

Class B devices can receive in scheduled RX slots or in RX slots following the transmission of an uplink. Class C devices are almost in constant reception.

RECOMMENDED retransmission timer value:

- Class B: 3 times the ping slot periodicity.
- Class C: 30 seconds.

The RECOMMENDED inactivity timer value is 12 hours for both Class B and Class C devices.

5.7. SCHC Fragment Format

5.7.1. All-0 SCHC fragment

Uplink fragmentation (Ack-On-Error):

All-0 is distinguishable from a SCHC ACK REQ as [RFC8724] states _This condition is also met if the SCHC Fragment Header is a multiple of L2 Words_; this condition met: SCHC header is 2 bytes.

Downlink fragmentation (Ack-always):

As per [RFC8724] the SCHC All-1 MUST contain the last tile, implementation must ensure that SCHC All-0 message Payload will be at least the size of an L2 Word.

5.7.2. All-1 SCHC fragment

All-1 is distinguishable from a SCHC Sender-Abort as [RFC8724] states _This condition is met if the RCS is present and is at least the size of an L2 Word_; this condition met: RCS is 4 bytes.

5.7.3. Delay after each LoRaWAN frame to respect local regulation

This profile does not define a delay to be added after each LoRaWAN frame, local regulation compliance is expected to be enforced by LoRaWAN stack.

6. Security Considerations

This document is only providing parameters that are expected to be best suited for LoRaWAN networks for [RFC8724]. IID security is discussed in Section 5.3. As such, this document does not contribute to any new security issues beyond those already identified in [RFC8724]. Moreover, SCHC data (LoRaWAN payload) are protected at the LoRaWAN level by an AES-128 encryption with a session key shared by the device and the SCHC gateway. These session keys are renewed at each LoRaWAN session (ie: each join or rejoin to the LoRaWAN network)

7. IANA Considerations

This document has no IANA actions.

Acknowledgements

Thanks to all those listed in the Contributors section for the excellent text, insightful discussions, reviews and suggestions, and also to (in alphabetical order) Dominique Barthel, Arunprabhu Kandasamy, Rodrigo Munoz, Alexander Pelov, Pascal Thubert, Laurent Toutain for useful design considerations, reviews and comments.

Contributors

Contributors ordered by family name.

Vincent Audebert
EDF R&D
Email: vincent.audebert@edf.fr

Julien Catalano
Kerlink
Email: j.catalano@kerlink.fr

Michael Coracin
Semtech
Email: mcoracin@semtech.com

Marc Le Gourrierec
Sagemcom
Email: marc.legourrierec@sagemcom.com

Nicolas Sornin
Semtech
Email: nsornin@semtech.com

Alper Yegin
Actility
Email: alper.yegin@actility.com

10. References

10.1. Normative References

[lora-alliance-spec]
Alliance, L., "LoRaWAN Specification Version V1.0.4", <https://lora-alliance.org/resource_hub/lorawan-104-specification-package/>.

[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>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[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>.

10.2. Informative References

[lora-alliance-remote-multicast-set] Alliance, L., "LoRaWAN Remote Multicast Setup Specification Version 1.0.0", <https://lora-alliance.org/sites/default/files/2018-09/remote_multicast_setup_v1.0.0.pdf>.

[RFC8064] Gont, F., Cooper, A., Thaler, D., and W. Liu, "Recommendation on Stable IPv6 Interface Identifiers", RFC 8064, DOI 10.17487/RFC8064, February 2017, <https://www.rfc-editor.org/info/rfc8064>.

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

[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>.

10.3. URIs

[1] https://www.lora-alliance.org

Appendix A. Examples

In following examples "applicative data" refers to the IPv6 payload sent by the application to the SCHC layer.

A.1. Uplink - Compression example - No fragmentation

This example represents an applicative data going through SCHC over LoRaWAN, no fragmentation required

An applicative data of 78 bytes is passed to SCHC compression layer. Rule 1 is used by SCHC C/D layer, allowing to compress it to 40 bytes and 5 bits: 1 byte RuleID, 21 bits residue + 37 bytes payload.

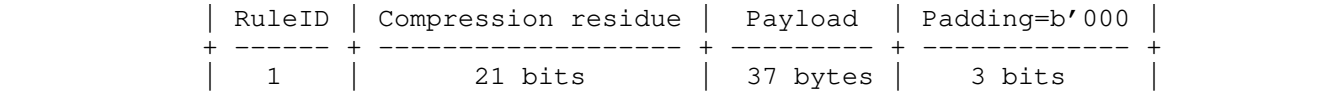


Figure 19: Uplink example: SCHC Message

The current LoRaWAN MTU is 51 bytes, although 2 bytes FOpts are used by LoRaWAN protocol: 49 bytes are available for SCHC payload; no need for fragmentation. The payload will be transmitted through FPort = 1.

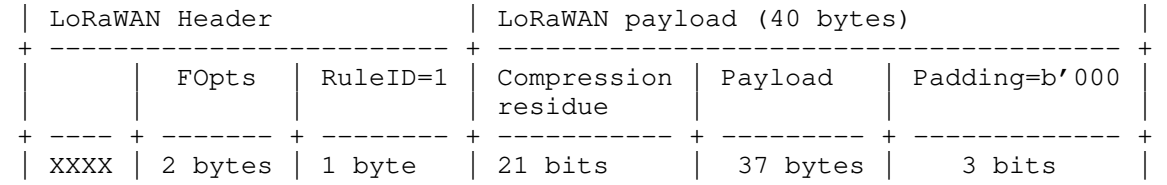


Figure 20: Uplink example: LoRaWAN packet

A.2. Uplink - Compression and fragmentation example

This example represents an applicative data going through SCHC, with fragmentation.

An applicative data of 300 bytes is passed to SCHC compression layer. Rule 1 is used by SCHC C/D layer, allowing to compress it to 282 bytes and 5 bits: 1 byte RuleID, 21 bits residue + 279 bytes payload.

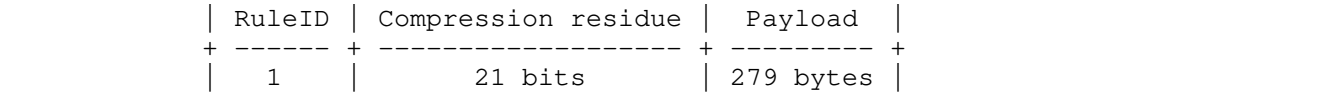


Figure 21: Uplink example: SCHC Message

The current LoRaWAN MTU is 11 bytes, 0 bytes FOpts are used by LoRaWAN protocol: 11 bytes are available for SCHC payload + 1 byte FPort field. SCHC header is 2 bytes (including FPort) so 1 tile is sent in first fragment.

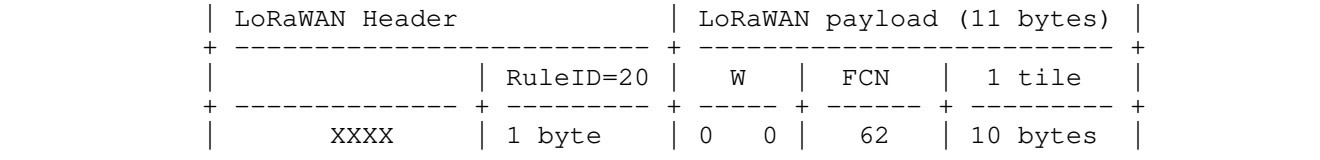


Figure 22: Uplink example: LoRaWAN packet 1

Content of the tile is:

RuleID	Compression residue	Payload
1	21 bits	6 bytes + 3 bits

Figure 23: Uplink example: LoRaWAN packet 1 - Tile content

Next transmission MTU is 11 bytes, although 2 bytes FOpts are used by LoRaWAN protocol: 9 bytes are available for SCHC payload + 1 byte FPort field, a tile does not fit inside so LoRaWAN stack will send only FOpts.

Next transmission MTU is 242 bytes, 4 bytes FOpts. 23 tiles are transmitted:

LoRaWAN Header				LoRaWAN payload (231 bytes)	
	FOpts	RuleID=20	W	FCN	23 tiles
XXXX	4 bytes	1 byte	0 0	61	230 bytes

Figure 24: Uplink example: LoRaWAN packet 2

Next transmission MTU is 242 bytes, no FOpts. All 5 remaining tiles are transmitted, the last tile is only 2 bytes + 5 bits. Padding is added for the remaining 3 bits.

LoRaWAN Header	LoRaWAN payload (44 bytes)				
	RuleID=20	W	FCN	5 tiles	Padding=b'000
XXXX	1 byte	0 0	38	42 bytes+5 bits	3 bits

Figure 25: Uplink example: LoRaWAN packet 3

Then All-1 message can be transmitted:

LoRaWAN Header		LoRaWAN payload (44 bytes)		
	RuleID=20	W	FCN	RCS
XXXX	1 byte	0 0	63	4 bytes

Figure 26: Uplink example: LoRaWAN packet 4 - All-1 SCHC message

All packets have been received by the SCHC gateway, computed RCS is correct so the following ACK is sent to the device by the SCHC receiver:

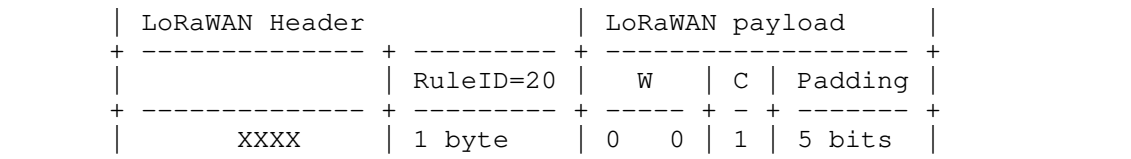


Figure 27: Uplink example: LoRaWAN packet 5 - SCHC ACK

A.3. Downlink

An applicative data of 155 bytes is passed to SCHC compression layer. Rule 1 is used by SCHC C/D layer, allowing to compress it to 130 bytes and 5 bits: 1 byte RuleID, 21 bits residue + 127 bytes payload.

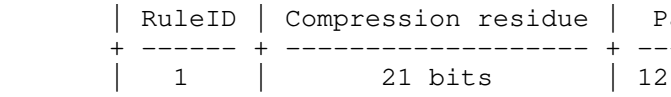


Figure 28: Downlink example: SCHC Message

The current LoRaWAN MTU is 51 bytes, no FOpts are used by LoRaWAN protocol: 51 bytes are available for SCHC payload + FPort field => it has to be fragmented.

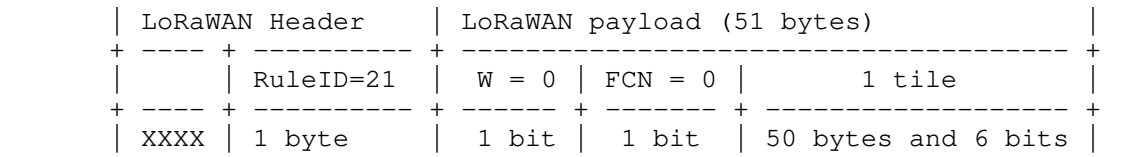


Figure 29: Downlink example: LoRaWAN packet 1 - SCHC Fragment 1

Content of the tile is:

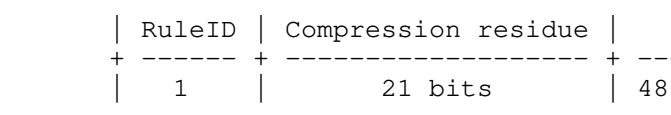


Figure 30: Downlink example: LoRaWAN packet 1: Tile content

The receiver answers with a SCHC ACK:

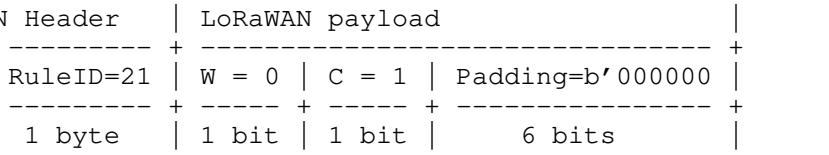


Figure 31: Downlink example: LoRaWAN packet 2 - SCHC ACK

The second downlink is sent, two FOpts:

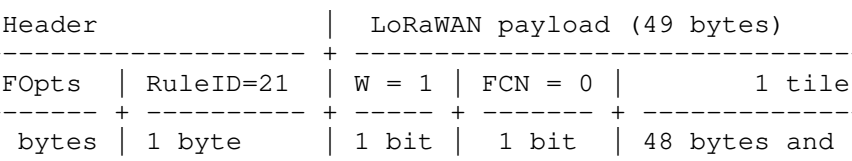


Figure 32: Downlink example: LoRaWAN packet 3 - SCHC Fragment 2

The receiver answers with an SCHC ACK:

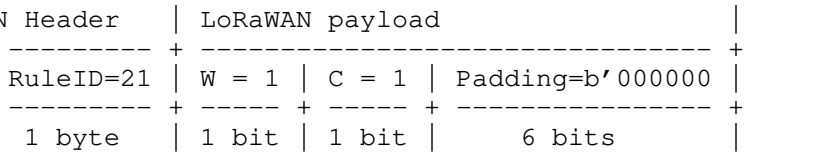


Figure 33: Downlink example: LoRaWAN packet 4 - SCHC ACK

The last downlink is sent, no FOpts:

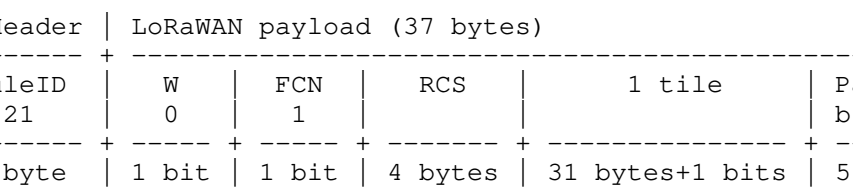


Figure 34: Downlink example: LoRaWAN packet 5 - All-1 SCHC message

The receiver answers to the sender with an SCHC ACK:

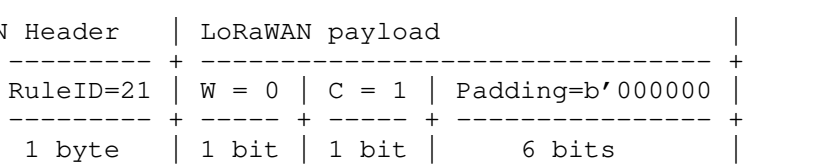


Figure 35: Downlink example: LoRaWAN packet 6 - SCHC ACK

Authors' Addresses

Olivier Gimenez (editor)
Semtech
14 Chemin des Clos
Meylan
France

Email: ogimenez@semtech.com

Ivaylo Petrov (editor)
Acklio
1137A Avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France

Email: ivaylo@ackl.io

lpwan Working Group
Internet-Draft
Intended status: Informational
Expires: July 23, 2021

E. Ramos
Ericsson
A. Minaburo
Acklio

January 19, 2021

SCHC over NB-IoT
draft-ietf-lpwan-schc-over-nbiot-04

Abstract

The Static Context Header Compression (SCHC) specification describes a header compression and fragmentation functionalities for LPWAN (Low Power Wide Area Networks) technologies. SCHC was designed to be adapted over any of the LPWAN technologies.

This document describes the use of SCHC over the NB-IoT wireless access, and provides elements for an 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 http://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 July 23, 2021.

Copyright Notice

Table of Contents

1. Introduction

The Static Context Header Compression (SCHC) {RFC8724} defines a header compression scheme and fragmentation functionality, both

specially tailored for Low Power Wide Area Networks (LPWAN) networks defined in [RFC8376].

Header compression is needed to efficiently bring Internet connectivity to the node within an NB-IoT network. SCHC uses a static context to performs header compression with specific parameters that need to be adapted into the NB-IoT wireless access. This document assumes functionality for NB-IoT of 3GPP release 15 otherwise other versions functionality is explicitly mentioned in the text.

This document describes the use of SCHC and its parameterizing over the NB-IoT wireless access.

2. Terminology

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

- CIoT. Cellular IoT
- C-SGN. CIoT Serving Gateway Node
- UE. User Equipment
- eNB. Node B. Base Station that controls the UE
- EPC. Evolved Packet Connectivity. Core network of 3GPP LTE systems.
- EUTRAN. Evolved Universal Terrestrial Radio Access Network. Radio network from LTE based systems.
- MME. Mobility Management Entity. Handle mobility of the UE
- NB-IoT. Narrow Band IoT. Referring to 3GPP LPWAN technology based in LTE architecture but with additional optimization for IoT and using a Narrow Band spectrum frequency.
- SGW. Serving Gateway. Routes and forwards the user data packets through the access network
- HSS. Home Subscriber Server. It is a database that performs mobility management
- PGW. Packet Data Node Gateway. An interface between the internal with the external network

- o PDU. Protocol Data Unit. Data packets including headers that are transmitted between entities through a protocol.

- o SDU. Service Data Unit. Data packets (PDUs) from higher layers protocols used by lower layer protocols as a payload of their own PDUs that has not yet been encapsulated.

- o IWK-SCEF. InterWorking Service Capabilities Exposure Function. Used in roaming scenarios and serves for interconnection with the SCEF of the Home PLMN and is located in the Visited PLMN

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

3. Architecture

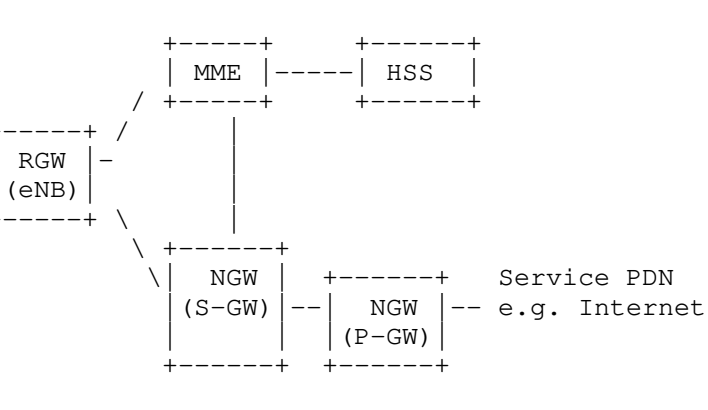


Figure 1: 3GPP network architecture

The architecture for 3GPP LTE network has been reused for NB-IoT with some optimizations and simplifications known as Cellular IoT (CIoT). Considering the typical use cases for CIoT devices here are described some of the additions to the LTE architecture specific for CIoT. C-SGN(CIoT Serving Gateway Node) is a deployment option co-locating EPS entities in the control plane and user plane paths (for example, MME + SGW + P-GW) and the external interfaces of the entities supported. The C-SGN also supports at least some of the following CIoT EPS Optimizations:

- o Control Plane CIoT EPS Optimization for small data transmission.

- o User Plane CIoT EPS Optimization for small data transmission.

- Necessary security procedures for efficient small data transmission.

- SMS without combined attach for NB-IoT only UEs.

- Paging optimizations for coverage enhancements.

- Support for non-IP data transmission via SGi tunneling and/or SCEF.

- Support for Attach without PDN (Packet Data Network) connectivity.

Another node introduced in the CIOT architecture is the SCEF (Service Capability Exposure Function) that provide means to securely expose service and network capabilities to entities external to the network operator. The northbound APIS are defined by OMA and OneM2M. The main functions of a SCEF are:

- Non-IP Data Delivery (NIDD) established through the SCEF.

- Monitoring and exposure of event related to UE reachability, loss of connectivity, location reporting, roaming status, communication failure and change of IMEI-IMSI association.

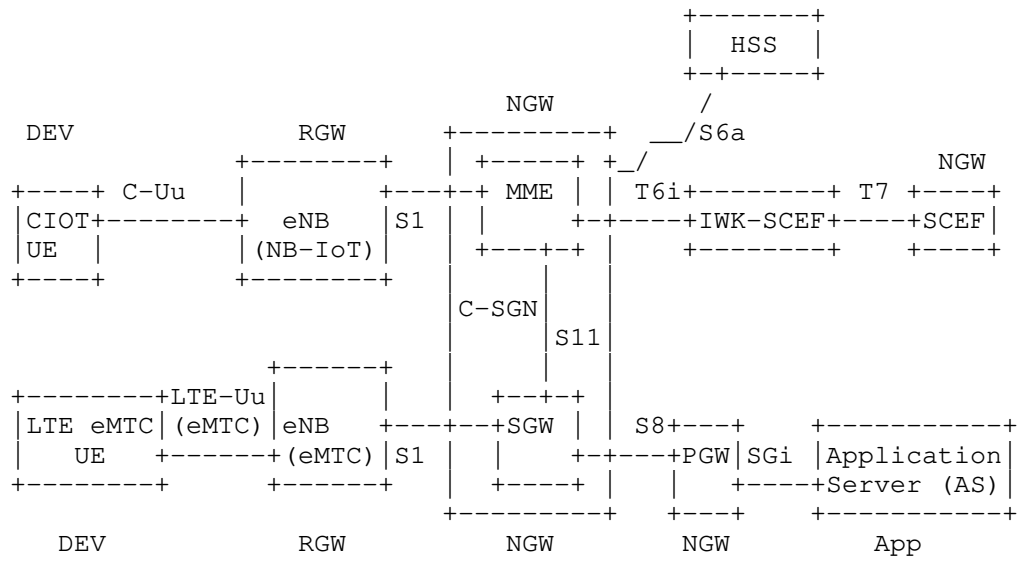


Figure 2: 3GPP optimized CIOT network architecture

4. Data Transmission

3GPP networks deal not only with data transmitted end-to-end but also with in-band signaling that is used between the nodes and functions to configure, control and monitor the system functions and behaviors. The control data is handled using a Control Plane which has a specific set of protocols, handling processes and entities. In contrast, the end-to-end or user data utilize a User Plane with characteristics of its own separated from the Control Plane. The handling and setup of the Control Plane and User Plane spans over the whole 3GPP network and it has particular implications in the radio network (i.e., EUTRAN) and in the packet core (ex., EPC).

For the CIOT cases, additionally to transmissions of data over User Plane, 3GPP has specified optimizations for small data transmissions, allowing to transport user data (IP, Non-IP) within signaling on the access network (Data transmission over Control Plane or Data Over NAS).

The maximum recommended MTU size is 1358 Bytes. The radio network protocols limit the packet sizes to be transmitted over the air including radio protocol overhead to 1600 Octets. But the value is reduced further to avoid fragmentation in the backbone of the network due to the payload encryption size (multiple of 16) and handling of the additional core transport overhead.

NB-IoT and in general the cellular technologies interfaces and functions are standardized by 3GPP. Therefore the introduction of SCHC entities to UE, eNB and C-SGN does need to be specified in the NB-IoT standard. This implies that standard specified SCHC support would not be backwards compatible. A terminal or a network supporting a version of the standard without support of SCHC or without capability implementation (in case of not being standardized as mandatory capability) is not able to utilize the compression services with this approach.

SCHC could be deployed differently depending on where the header compression and the fragmentation are applied. The SCHC functionalities could be applied to the packets about to be transmitted over the air, or to the whole end-to-end link. To accomplish the first, it is required to place SCHC compression and decompression entities in the eNB and in the UE for transmissions over the User Plane. Additionally, to handle the case of the transmissions over Control Plane or Data Over NAS, the network SCHC entity has to be placed in the C-SGN as well. For these two cases, the functions are 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. In that case, the SCHC entities would be placed in the application layer of the terminal in one end, and either in the application servers or in a broker function in the edge of the operator network in the other end. For the radio network, the packets are transmitted as non-IP traffic, which can be currently served utilizing IP tunneling or SCEF services. Since this option does not necessarily require 3GPP standardization, it is possible to also benefit legacy devices with SCHC by utilizing the non-IP transmission features of the operator network.

Accordingly, there are four different scenarios where SCHC can be used in the NB-IoT architecture. IP header compression on the data transmission over User Plane, IP header compression on the optimized transmissions over Control Plane (i.e.,DoNAS), non-IP transmissions of SCHC packets by IP tunneling, and non-IP transmissions of SCHC packets by SCEF forwarding. The following sections describe each of them in more detail. The first two scenarios refer to transmissions using the 3GPP IP transmission capabilities and the last two refers to transmission using the Non-IP capabilities.

5. IP based Data Transmission

5.1. SCHC over User Plane transmissions

Deploying SCHC only over the radio link would require to place it as part of the User Plane data transmission. The User Plane utilizes the protocol stack of the Access Stratum (AS) for data transfer. AS (Access Stratum) is the functional layer responsible for transporting data over wireless connection and managing radio resources. The user plane AS has support for features such as reliability, segmentation and concatenation. The transmissions of the AS make use of link adaptation, meaning that the transport format utilized for the transmissions are optimized according to the radio conditions, the number of bits to transmit and the power and interference constrains. That means that the number of bits transmitted over the air depends of the Modulation and Coding Schemes (MCS) selected. The transmissions in the physical layer happens at network synchronized intervals of times called TTI (Transmission Time Interval). The transmission of a Transport Block (TB) is completed during, at least, one TTI. Each Transport Block has a different MCS and number of bits available to transmit. The Transport Blocks characteristics are defined by the MAC technical specification TGPP36321. The Access Stratum for User Plane is comprised by Packet Data Convergence Protocol (PDCP) TGPP36323, Radio Link Protocol (RLC) TGPP36322, Medium Access Control protocol (MAC) TGPP36321 and the Physical Layer TGPP36201. More details of this protocols are given in the Appendix.

5.1.1. SCHC Entities Placing

The current architecture provides support for header compression in PDCP utilizing RoHC [RFC5795]. Therefore SCHC entities can be deployed in similar fashion without need for major changes in the 3GPP specifications.

In this scenario, RLC takes care of the handling of fragmentation (if transparent mode is not configured) when packets exceeds the transport block size at the time of transmission. Therefore SCHC fragmentation is not needed and should not be used to avoid additional protocol overhead. It is not common to configure RLC in Transparent Mode for IP based user plane data. But 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.

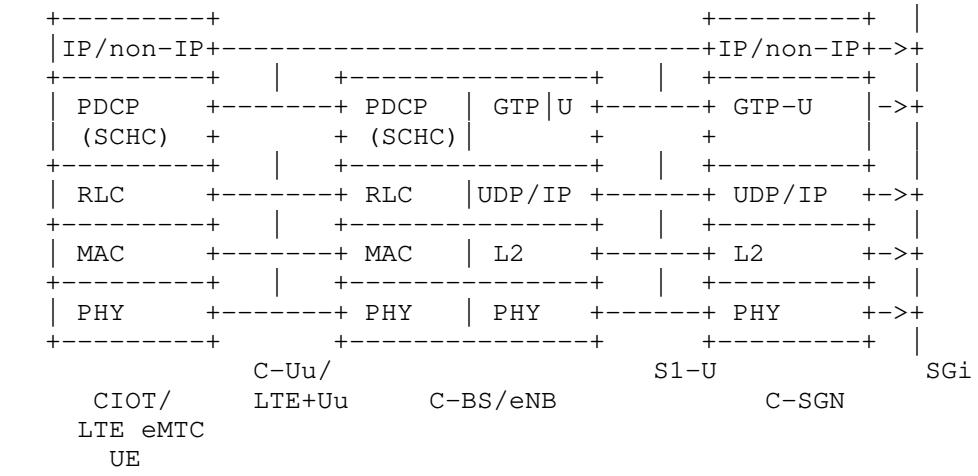


Figure 3: SCHC entities placement in the 3GPP CIOT radio protocol architecture for data over user plane

5.2. Data Over Control Plane

The Non-Access Stratum (NAS), conveys mainly control signaling between the UE and the cellular network TGPP24301. NAS is transported on top of the Access Stratum (AS) already mentioned in the previous section.

NAS has been adapted to provide support for user plane data transmissions to reduce the overhead when transmitting infrequent small quantities of data. This is known as Data over NAS (DoNAS) or

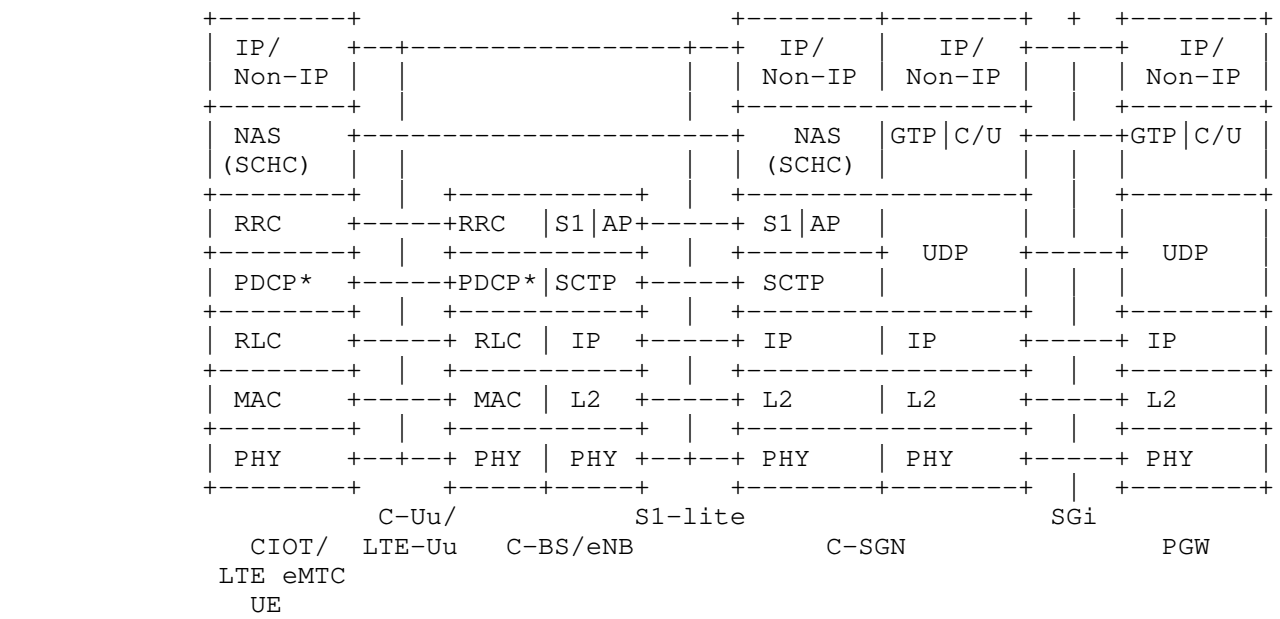
Control Plane CIoT EPS optimization. In DoNAS the UE makes use of the pre-established NAS security and piggyback uplink small data into the initial NAS uplink message, and uses an additional NAS message to receive downlink small data response.

The data encryption from the network side is performed by the C-SGN in a NAS PDU. Depending on the data type signaled indication (IP or non-IP data), the network allocates an IP address or just establish a direct forwarding path. DoNAS (Data over NAS) is regulated under rate control upon previous agreement, meaning that a maximum number of bits per unit of time is agreed 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 to do a short transmission and receive 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 and the network. The support for mobility of DoNAS is present but produces additional overhead. Additional details of DoNAS are given in the Appendix.

5.2.1. SCHC Entities Placing

In this scenario SCHC can be applied in the NAS protocol layer instead of PDCP. The same principles than for user plane transmissions applies here as well. The main difference is the physical placing of the SCHC entities in the network side as the C-SGN (placed in the core network) is the terminating node for NAS instead of the eNB.



*PDCP is bypassed until AS security is activated TGPP36300.

Figure 4

5.3. Parameters for Static Context Header Compression (SCHC)

5.3.1. SCHC Context initialization

RRC (Radio Resource Control) protocol is the main tool used to configure the operation parameters of the AS transmissions for 3GPP technologies. RoHC operation is configured with this protocol and it is to expect that SCHC will be configured and the static context distributed in similar fashion for these scenarios.

5.3.2. SCHC Rules

The number of rules in a context are defined by the network operator in these scenarios. For this, the operator must be aware of the type of IP traffic that the device will carry out. This means that the operator might provision sets of rules compatible with the use case of the device. For devices acting as gateways of other devices several rules that match the diversity of devices and protocols used by the devices associated to the gateway. Meanwhile than simpler devices (for example an electricity meter) may have a predetermined set of protocols and parameters fixed. Additionally, the deployment

of IPV4 addresses in addition to IPV6 may force to provision separate rules to deal with each of the cases.

5.3.3. Rule ID

For these transmission scenarios in NB-IoT, a reasonable assumption of 9 bytes of radio protocol overhead can be expected. PDCP 5 bytes due to header and integrity protection, and 4 bytes of RLC and MAC. The minimum physical Transport Block (TB) that can withhold this overhead value according to 3GPP Release 15 specifications are: 88, 104, 120 and 144 bits. If it is wished to optimize the number of transmissions of a very small application packet so that in some cases can be transmitted using only one physical layer transmission, then the SCHC overhead should not exceed the available number of bits of the smallest utile 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 utilize the smallest TB the maximum SCHC is 8 bits. This must include the Compression Residue in addition to the Rule ID. In the other hand, it is possible that more complex NB-IoT devices (such as a capillarity gateway) might require additional bits to handle the variety and multiple parameters the 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 a configurable value is preferred for these scenarios. The configuration may be set as part of the operation profile agreed together with the context distribution. The Rule Id field size may range for example from 2 bits resulting in 4 rules to a 8 bits value that would yield up to 256 rules which 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 take in account the byte-alignment of the expected Compression Residue too. In the minimum TB size case, 2 bits size of Rule Id leave only 6 bits available for Compression Residue.

5.3.4. SCHC MAX_PACKET_SIZE

The Access Stratum can handle the fragmentation of SCHC packets if needed including reliability. Hence the packet size is limited by the MTU possible to be handled by the AS radio protocols that corresponds to 1600 bytes for 3GPP Release 15.

5.3.5. Fragmentation

For these scenarios the SCHC fragmentation functions are recommend to be 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 selection of the blocks is done according to the input of the link adaptation 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 depends of the network load, interference and number of bits to be transmitted and QoS. Even if setting a value that allows the construction of data units following SCHC tiles principle, the protocol overhead may be greater or equal than allowing the AS radio protocols to take care of the fragmentation natively.

5.3.5.1. Fragmentation in Transparent Mode

If RLC is configured to operate 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 very rare to transmit user plane data using this configuration and it is mainly targeting control plane transmissions. In those cases the reliability is normally ensured by MAC based mechanisms, such as repetitions or automatic retransmissions, and additional reliability might only generate protocol overhead.

In future operations, it could be devised the utilization of SCHC to reduce radio network protocols overhead and support the reliability of the transmissions, and targeting small data with the fewer possible transmissions. This could be realized by using fixed or limited set of transport blocks compatible with the tiling SCHC fragmentation handling.

6. Non-IP based Data Transmission

The Non-IP Data Delivery (NIDD) services of 3GPP enable the possibility of transmitting SCHC packets compressed by the application layer. The packets can be delivered by means of IP-tunnels to the 3GPP network or using SCEF functions (i.e., API calls). In both cases the packet IP is not understood by the 3GPP network since it is already compressed and the network does not has information of the context used for compression. Therefore the network will treat the packet as a Non-IP traffic and deliver it to the UE without any other stack element, directly under the L2.

6.1. SCHC Entities Placing

In the two scenarios using NIDD, SCHC entities are located almost in top of the stack. In the terminal, it may be implemented by a application utilizing the NB-IoT connectivity services. In the network side, the SCHC entities are located in the Application Server (AS). The IP tunneling scenario requires that the Application Server

sends the compressed packet over an IP connection that is terminated by the 3GPP core network. If instead the SCEF services are used, then it is possible to utilize a API call to transfer the SCHC packets between the core network and the AS, also an IP tunnel could be established by the AS, if negotiated with the SCEF.

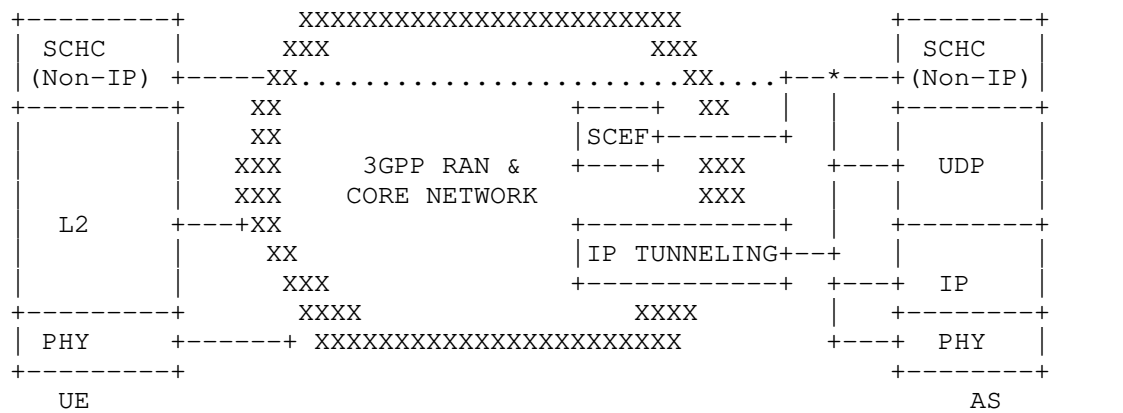


Figure 5: SCHC entities placed when using Non-IP Delivery (NIDD) 3GPP Sevices

6.2. Parameters for Static Context Header Compression

6.2.1. SCHC Context initialization

The static context is handled in the application layer level, consequently the contexts are required to be distributed according to the applications own capabilities, perhaps utilizing IP data transmissions up to context initialization. Also the same IP tunneling or SCEF services used later for the SCHC packets transport may be used by the applications in both ends to deliver the static contexts to be used.

6.2.2. SCHC Rules

Even when the transmissions content are not visible for the 3GPP network, the same limitations than for IP based data transmissions applies in these scenarios in terms of aiming to use the minimum number of transmission and minimize the protocol overhead.

6.2.3. Rule ID

Similarly to the case of IP transmissions, the Rule ID size can be dynamically set prior the context delivery. For example negotiated between the applications when choosing a profile according to the type of traffic and type of application deployed. Same considerations related to the transport block size and performance mentioned for the IP type of traffic has to be follow when choosing a size value for the Rule ID field.

6.2.4. 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), and not only the radio as the IP transmissions case.

6.3. Fragmentation

In principle the fragmentation function should be activated for packets greater than 1358 Bytes. Since the 3GPP reliability functions take great deal care of it, for simple point to point connections may be enough a NO-ACK mode. Nevertheless additional considerations for more complex cases are mentioned in the next subsection to be taken in account.

6.3.1. Fragmentation modes

Depending of the QoS that has been assigned to the packets, it is possible that packets are lost before they arrive to 3GPP radio network transmission, for example in between the links of a capillarity gateway, or due to buffer overflow handling in a backhaul connection. In consequence, it is possible to secure additional reliability on the packets transmitted with a small trade-off on additional transmissions to signal the packets arrival indication end-to-end if no transport protocol takes care of retransmission. To achieve this, the packets fragmentation is activated with the ACK-on-Error mode enabled. In some cases, it 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 MAX_PACKET_SIZE == 1 Byte) and the ACK-on-Error mode. In the NAS stratum, the use of only fragmentation when a non-IP packet is transmitted is possible if this packet is considered as a SCHC packet and is identifyed using the RuleID for non-compressing packets as {RFC8724} allows it, depending on the application an ACK-onError mode may be used.

6.3.2. Fragmentation Parameters

The Fragmentation Rule ID is given when choosing the profile according to the fragmentation mode, 1 bit can be used to recognize each mode.

To adapt SCHC to the NB-IoT constraints, two configuration are proposed to fill the best the transfer block (TB). The Header size needs to be multiple of 4 and the Tiles can keep a fix value of 4 or 8 bits to avoid the need of padding.

- 8 bits-Header_size configuration, with the size of the header fields as follow: Rule ID 3 bits, DTag 1 bit, FCN 3 bits, W 1 bits. This configuration may ne used with TB less than 300 bits.

- 16 bits-Header_size configuration, with the size of the header fields as follow: Rules ID 8 - 10 bits, DTag 1 or 2 bits, FCN 3 bits, W 2 or 3 bits. This configuration may be used with TB above 300 bits.

The IoT devices communicates with small data transfer and have a battery life of 10 years. To minise the power consumption these devices use the Power Save Mode and the Idle Mode DRX which govern how often the device wakes up, stays up and are reachable. The 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 Innactivity Timer may be above 1h or 10h and the Retransmission Timer may be below than 1h or 10h.

7. 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 treatment of padding should use this value accordingly.

8. Security considerations

3GPP access security is specified in (TGPP33203).

9. 3GPP References

- TGPP23720 3GPP, "TR 23.720 v13.0.0 - Study on architecture enhancements for Cellular Internet of Things", 2016.

- TGPP33203 3GPP, "TS 33.203 v13.1.0 - 3G security; Access security for IP-based services", 2016.

- TGPP36321 3GPP, "TS 36.321 v13.2.0 - Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification", 2016

- TGPP36323 3GPP, "TS 36.323 v13.2.0 - Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification", 2016.

- TGPP36331 3GPP, "TS 36.331 v13.2.0 - Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification", 2016.

- TGPP36300 3GPP, "TS 36.300 v15.1.0 - Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN); Overall description; Stage 2", 2018

- TGPP24301 3GPP "TS 24.301 v15.2.0 - Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS); Stage 3", 2018

- TGPP24088 3GPP, "TS 24.088 v12.9.0 - Mobile radio interface Layer 3 specification;Core network protocols; Stage 3", 2015.

10. Appendix

10.1. NB-IoT User Plane protocol architecture

10.1.1. Packet Data Convergence Protocol (PDCP)

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

- Header compression and decompression by means of 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

10.1.2. 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 that are 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 set of entities, one in each direction (downlink and uplink) must be configured and they are 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). In this mode RLC do not segment or concatenate SDUs from higher layers and do not include any header to the payload. When acting as a transmitter, RLC receives SDUs from upper layers and transmit directly to its flow RLC receiver via lower layers. Similarly, an TM RLC receiver would only deliver without additional processing the packets to higher layers upon reception.

- Unacknowledged Mode (UM). This mode provides support for segmentation and concatenation of payload. The size of the RLC packet depends of the indication given at a particular transmission opportunity by the lower layer (MAC) and are octets aligned. The packet delivery to the receiver do not include support for reliability and the lost of a segment from a packet means a whole packet loss. Also in 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 discard, reordering of out of sequence and loss detection.

- Acknowledged Mode (AM). Additional to the same functions supported from UM, this mode also adds a moving windows based reliability service on top of the lower layer services. It also provides support for re-segmentation and it requires bidirectional communication to exchange acknowledgment reports called RLC Status Report and trigger retransmissions is needed. Protocol error detection is also supported by this mode. The mode uses depends of 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 map to a UM configuration.

10.1.3. 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 the efficiency of data transmission. MAC also provides error correction and reliability support by means of 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 additional to the higher layers data.

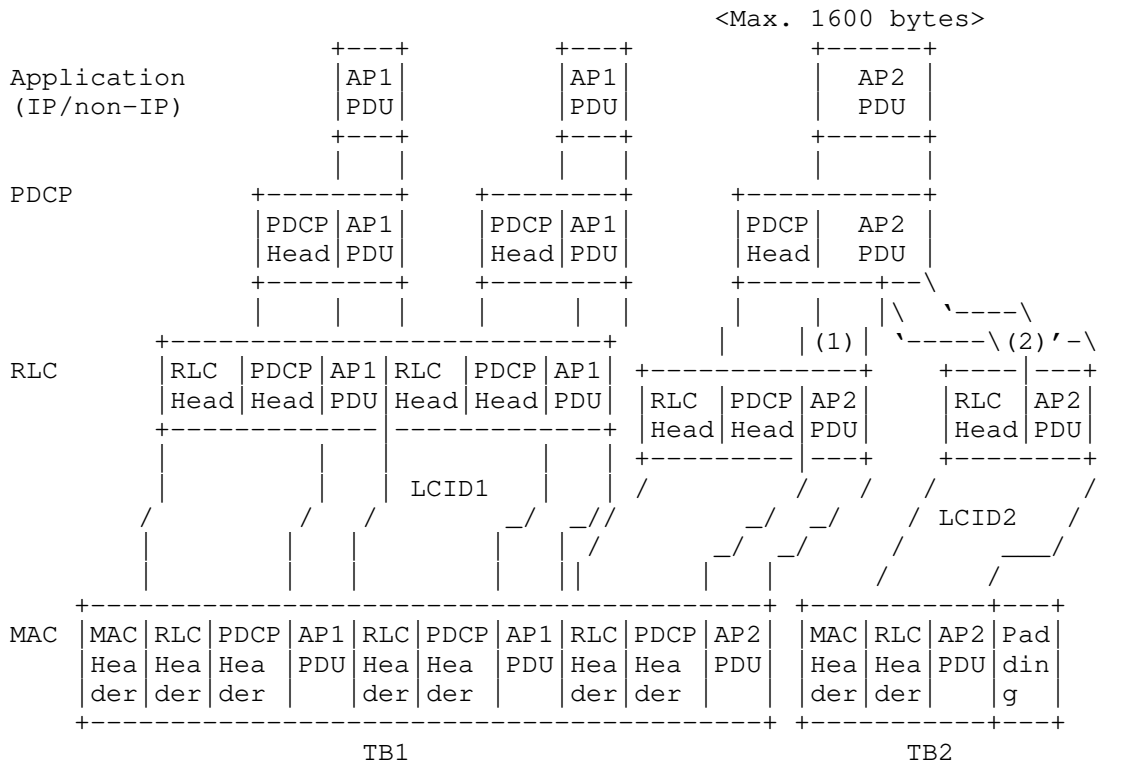


Figure 6: Example of User Plane packet encapsulation for two transport blocks

10.2. NB-IoT Data over NAS (DoNAS)

The AS protocol stack used by DoNAS is somehow special. 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) is configured by default in AM mode, but depending of the features supported by the network and the terminal it may be configured in other modes by the network operator. For example, the transparent mode does not add any header or does not process the payload in any way reducing the overhead, but the MTU would be limited by the transport block used to transmit the data which is couple of thousand of bits maximum. If UM (only Release 15 compatible terminals) is used, the RLC mechanisms of reliability is 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 for the AM case because the lack of status reporting but with the same support for segmentation up to 16000 Bytes. NAS packet are encapsulated within a RRC (Radio Resource Control) TGPP36331 message.

Depending of the data type indication signaled (IP or non-IP data), the network allocates an IP address or just establish 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 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 to do a short transmission and receive an acknowledgment or short feedback from the network. Depending of 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 and the network. The support for mobility of DoNAS is present but produces additional overhead.

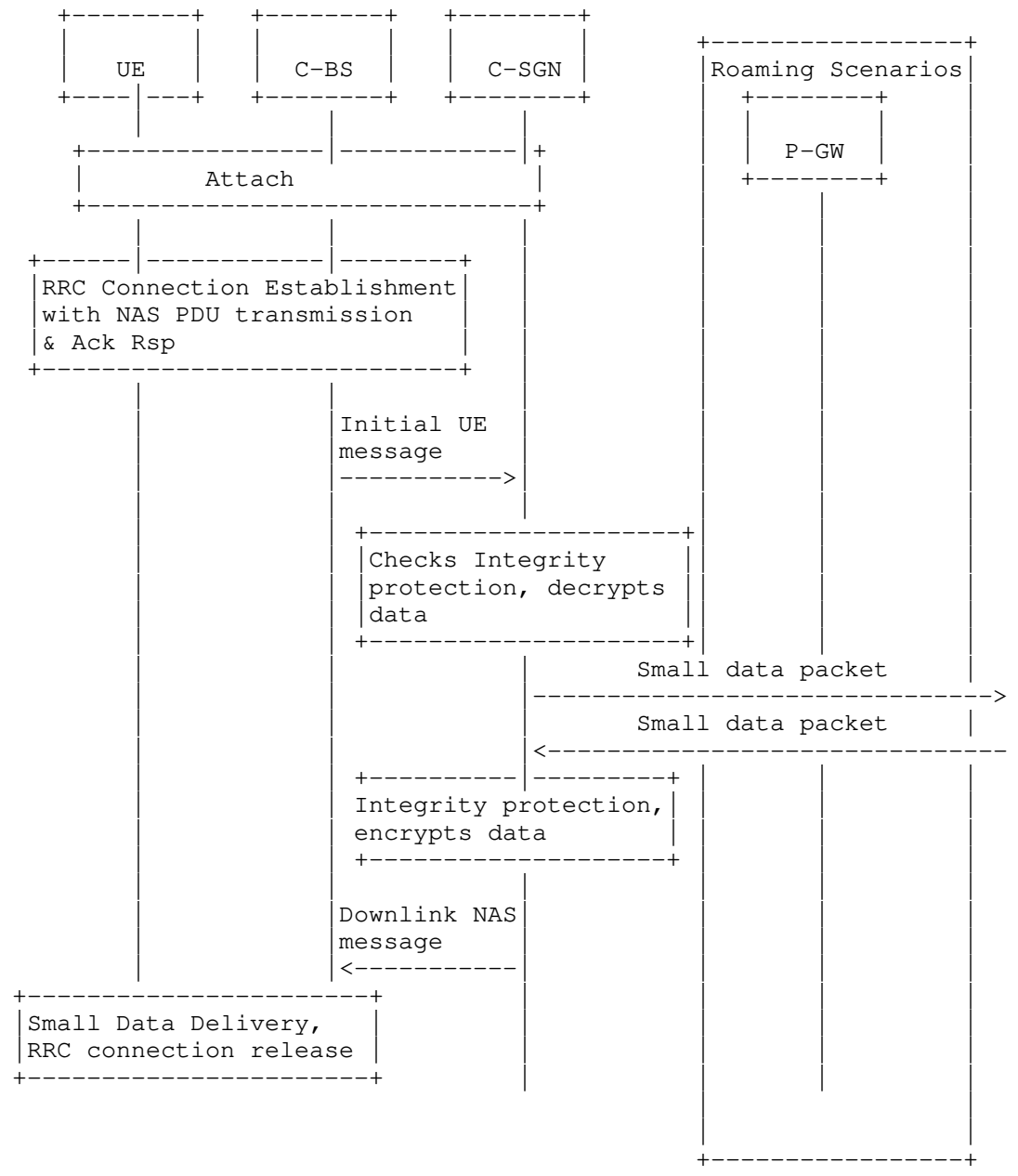


Figure 7: DoNAS transmission sequence from an Uplink initiated access

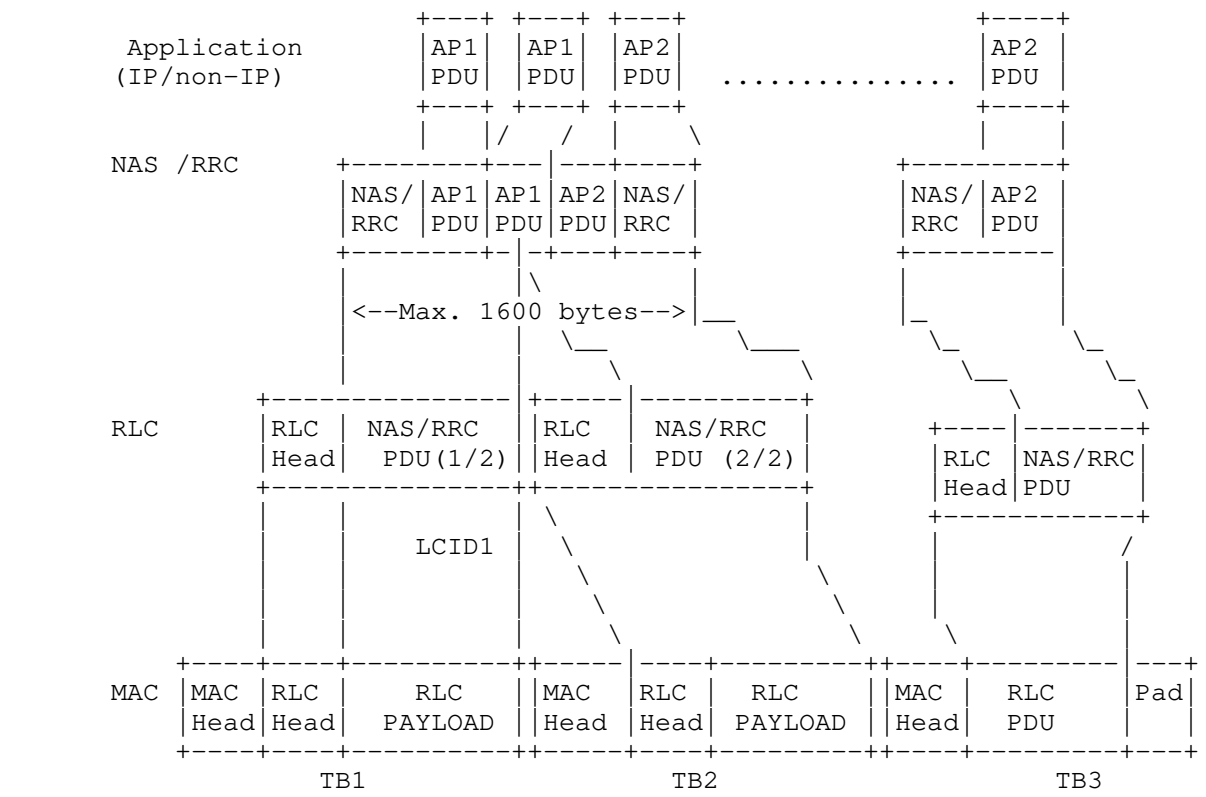


Figure 8: Example of User Plane packet encapsulation for Data over NAS

11. Normative References

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

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

[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>.

Authors' Addresses

Edgar Ramos
Ericsson
Hirsalantie 11
02420 Jorvas, Kirkkonummi
Finland

Email: edgar.ramos@ericsson.com

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

Email: ana@ackl.io

lpwan Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2021

JC. Zuniga
SIGFOX
C. Gomez
Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
October 31, 2020

SCHC over Sigfox LPWAN
draft-ietf-lpwan-schc-over-sigfox-04

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 May 4, 2021.

Table of Contents

1. Introduction

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

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

2. Terminology

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

3. SCHC: Generic Framework for Static Context Header Compression and Fragmentation

The Generic Framework for Static Context Header Compression and Fragmentation (SCHC) described in [RFC8724] takes advantage of the predictability of data flows existing in LPWAN applications to avoid context synchronization.

Contexts must 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.

4. SCHC over Sigfox

4.1. 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.

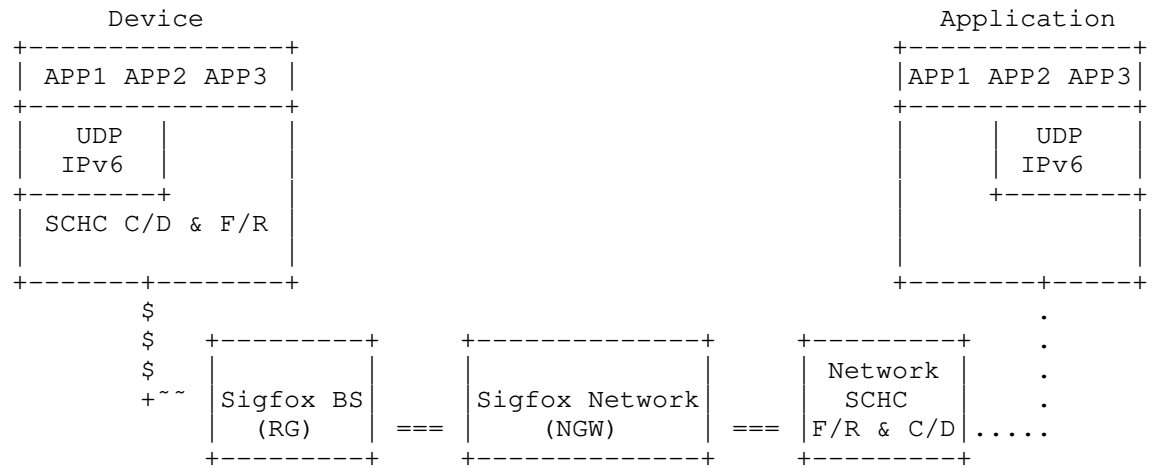


Figure 1: Network Architecture

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 share 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).

4.2. Uplink

Uplink Sigfox transmissions occur in repetitions over different times and frequencies. Besides these 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 Core Network, multiple input copies can be combined at the NGW and hence provide 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.

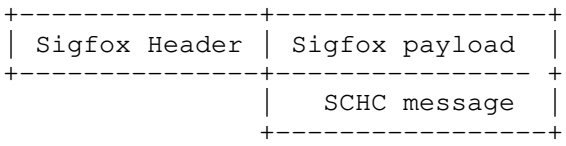


Figure 2: SCHC Message in Sigfox

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).

4.3. Downlink

Downlink transmissions are Device-driven and can only take place following an uplink communication that so indicates. Hence, a Device willing to receive a downlink message indicates explicitly its intention to the network in the preceding uplink message with a downlink request flag, and then it opens a fixed window for downlink reception after completing the uplink transmission. 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 it 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 receiving 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 back to the Network through the Sigfox protocol and reported by the Sigfox Network. This acknowledgement can be retrieved through callbacks by the customer.

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].

4.4. 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. 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.

4.5. 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.

4.6. 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 data frame. 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 CRC-check are delivered to the Network SCHC C/D + F/R, and each one has an associated Sigfox Message Sequence Number (see Section 4.2), integrity can be guaranteed when no consecutive messages are missing from the sequence and all FCN bitmaps are complete. In order to support multiple flows/RuleIDs (potentially interleaved), the implementation of a central message sequence counter at the Network SCHC C/D + F/R is required. With this functionality and in order to save protocol overhead, the use of a dedicated Reassembly Check Sequence (RCS) is NOT RECOMMENDED.

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

4.6.1. Uplink Fragmentation

Sigfox uplink transmissions are completely asynchronous and can take place in any random frequency of the allowed uplink bandwidth allocation. Hence, devices can go to deep sleep mode, and then wake up and transmit whenever there is a need to send any information to the network. In that way, there is no need to perform any network attachment, synchronization, or other procedure before transmitting a data packet. All data packets are self-contained (aka "message in a bottle") with all the required information for the network to process them accordingly.

Since uplink transmissions occur asynchronously, an 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.

4.6.1.1. 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 splitted 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)

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

ACK-on-Error with single-byte header is RECOMMENDED for medium-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
- WINDOW_SIZE: 7 (with a maximum value of FCN=0b110)

- Tile size: 11 bytes
- Retransmission Timer: Application-dependent
- Inactivity Timer: Application-dependent
- RCS size: 0 bits (Not used)

The correspondent SCHC ACK in the downlink is 13 bits long, so padding is needed to complete the required 64 bits of Sigfox payload.

4.6.1.3. Uplink ACK-on-Error Mode: Two-byte SCHC Header

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 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: 0 bits (Not used)

The correspondent SCHC ACK in the downlink is 43 bits long, so padding is needed to complete the required 64 bits of Sigfox payload.

4.6.1.4. All-1 behaviour + Sigfox Sequence Number

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.

However, since a Message Sequence Number is provided by the Sigfox protocol together with the Sigfox Payload, the receiver can detect if there are missing fragments before the All-1 and hence construct the corresponding SCHC ACK Bitmap accordingly.

4.6.2. 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)

4.7. 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 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 would be necessary.

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 ERDF and the Spanish Government through project TEC2016-79988-P.

The authors would like to thank Diego Wistuba, Sergio Aguilar, Clement Mannequin, Sandra Cespedes and Rafel Vidal for their useful comments and implementation design considerations.

7. References

7.1. Normative 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>.

[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>.

7.2. Informative References

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

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

Authors' Addresses

Juan Carlos Zuniga
SIGFOX
Montreal QC
Canada

Email: JuanCarlos.Zuniga@sigfox.com
URI: http://www.sigfox.com/

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

Email: carlesgo@entel.upc.edu

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

Email: Laurent.Toutain@imt-atlantique.fr

lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: January 11, 2021 Institut MINES TELECOM; IMT Atlantique
July 10, 2020

Data Model for Static Context Header Compression (SCHC)
draft-ietf-lpwan-schc-yang-data-model-03

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 January 11, 2021.

Table of Contents

1. Introduction

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 this constrained network. Draft [RFC8724] provides an abstract 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 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 do no refers to a specific fields. A field is abstracted through an ID, a position, a direction and a value that can be a numerical value or a string. [RFC8724] and [I-D.ietf-lpwan-coap-static-context-hc] specifies 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].

2.1. Compression Rules

[RFC8724] proposes an abstract 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.

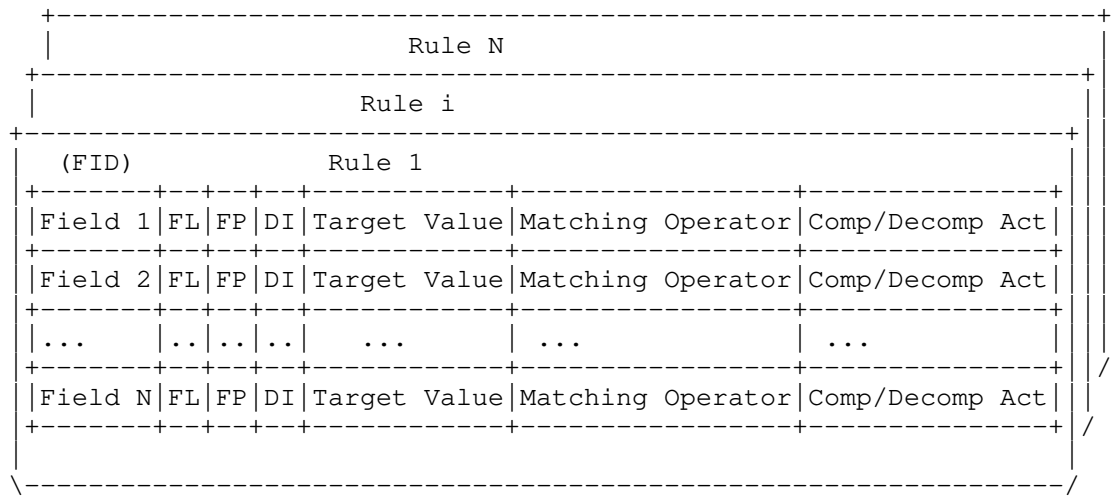


Figure 1: Compression Decompression Context

2.2. 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 one and apply compression. The link between the list given by the parsed fields and the rules is done through a field ID. [RFC8724] do not state how

the field ID value can be constructed. In the examples, it was given through a string indexed by the protocol name (e.g. IPv6.version, CoAP.version,...).

Using the YANG model, each field MUST be identified through a global YANG identityref. A YANG field ID derives from the field-id-base-type. Figure 2 gives some field ID definitions. Note that some field IDs can be splitted is smaller pieces. This is the case for "fid-ipv6-trafficclass-ds" and "fid-ipv6-trafficclass-ecn" which are a subset of "fid-ipv6-trafficclass-ds".

```
   identity field-id-base-type  {
           description "Field ID with SID";
    }

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

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

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

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

  ...
```

Figure 2: Definition of identityref for field IDs

Figure 2 gives an example of field ID identityref definitions. The base identity is field-id-base-type, and field id are derived for it. The naming convention is "fid" followed by the protocol name and the field name.

The yang model in annex (see Section 7) gives the full definition of the field ID for [RFC8724], [I-D.ietf-lpwan-coap-static-context-hc], and [I-D.barthel-lpwan-oam-schc].

The type associated to this identity is field-id-type (cf. Figure 3)

```
typedef field-id-type {
    description "Field ID generic type.";
    type identityref {
        base field-id-base-type;
    }
}
```

Figure 3: Type definition for field IDs

2.3. 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 in byte and [I-D.ietf-lpwan-coap-static-context-hc] defines the "tkl" function for managing the CoAP Token length field.

```
identity field-length-base-type {
      description "used to extend field length functions";
}

identity fl-variable {
         base field-length-base-type;
         description "residue length in Byte is sent";
}

identity fl-token-length {
         base field-length-base-type;
         description "residue length in Byte is sent";
}
```

Figure 4: Definition of identityref for field ILength

As for field ID, field length function can be defined as a identityref as shown in Figure 4.

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 5).

```
    typedef field-length-type {
        description "Field length either a positive integer giving the size in bi
ts
        or a function defined through an identityref.";
        type union {
            type int64; /* positive length in bits */
            type identityref { /* function */
                base field-length-base-type;
            }
        }
    }
```

Figure 5: Type definition for field Length

The naming convention is fl followed by the function name as defined in SCHC specifications.

2.4. Field position

Field position is a positive integer which gives the position of a field, the default value is 1, but if the field is repeated several times, the value is higher. value 0 indicates that the position is not important and is not taken into account during the rule selection process.

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

2.5. 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 direction-indicator-base-type {
           description "used to extend field length functions";
     }

     identity di-bidirectional {
              base direction-indicator-base-type;
              description "Direction Indication of bi directionality";
     }

     identity di-up {
              base direction-indicator-base-type;
              description "Direction Indication of upstream";
     }

     identity di-down {
              base direction-indicator-base-type;
              description "Direction Indication of downstream";
     }
```

Figure 6: Definition of identityref for direction indicators

Figure 6 gives the identityref for Direction Indicators.

The type is "direction-indicator-type" (cf. Figure 7).

```
 typedef direction-indicator-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 devi
ce.";
     type identityref {
         base direction-indicator-base-type;
     }
 }
```

Figure 7: Type definition for direction indicators

2.6. Target Value

Target Value may be either a string or binary sequence. For match-mapping, several of these values can be contained in a Target Value field. In the data model, this is generalized by adding a position, which orders the list of values. By default the position is set to 0.

The leaf "value" is not mandatory to represent a non existing value in a TV.

```
    grouping target-values-struct {
        description "defines the target value element. Can be either an arbitrary
        binary or ascii element. All target values are considered as a matching l
ists.
        Position is used to order values, by default position 0 is used when cont
aining
        a single element.";

        leaf value {
            type union {
                type binary;
                type string;
            }
        }
        leaf position {
            description "If only one element position is 0, otherwise position is
 the
            matching list.";
            type uint16;
        }
    }
```

Figure 8: Definition of target value

Figure 8 gives the definition of a single element of a Target Value. In the rule, this will be used as a list, with position as a key. The highest position value is used to compute the size of the index sent in residue.

2.7. 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.

```
     identity matching-operator-base-type {
        description "used to extend Matching Operators with SID values";
     }

     identity mo-equal {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-ignore {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-msb {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-matching {
        base matching-operator-base-type;
        description "RFC 8724";
     }
```

Figure 9: Definition of identityref for Matching Operator

the type is "matching-operator-type" (cf. Figure 10)

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

Figure 10: Type definition for Matching Operator

2.7.1. Matching Operator arguments

Some Matching Operator such as MSB can take some values. Even if currently LSB is the only MO takes only one argument, in the future some MO may require several arguments. They are viewed as a list of target-values-type.

2.8. Compression Decompression Actions

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

```
        identity compression-decompression-action-base-type;

       identity cda-not-sent {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-value-sent {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-lsb {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-mapping-sent {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-compute-length {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-compute-checksum {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

       identity cda-deviid {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }

      identity cda-appiid {
           base compression-decompression-action-base-type;
               description "RFC 8724";
       }
```

Figure 11: Definition of identityref for Compresion Decompression Action

The type is "comp-decomp-action-type" (cf. Figure 12)

```
   typedef comp-decomp-action-type {
        description "Compression Decompression Action to compression or decompres
s a field.";
        type identityref {
            base compression-decompression-action-base-type;
        }
    }
```

Figure 12: Type definition for Compresion Decompression Action

2.8.1. Compression Decompression Action arguments

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

3. Rule definition

A rule is either a C/D or an F/R rule. A rule is identified by the rule ID value and its associated length. The YANG grouping rule-id-type defines the structure used to represent a rule ID. Length of 0 is allowed to represent an implicit rule.

```
// Define rule ID. Rule ID is composed of a RuleID value and a Rule ID Length

    grouping rule-id-type {
        leaf rule-id {
            type uint32;
            description "rule ID value, this value must be unique combined with t
he length";
        }
        leaf rule-length {
            type uint8 {
                range 0..32;
            }
            description "rule ID length in bits, value 0 is for implicit rules";
        }
    }

// SCHC table for a specific device.

    container schc {
        leaf version{
            type uint64;
            mandatory false;
            description "used as an indication for versioning";
        }
        list rule {
            key "rule-id rule-length";
            uses rule-id-type;
            choice nature {
                case fragmentation {
                    uses fragmentation-content;
                }
                case compression {
                    uses compression-content;
                }
            }
        }
    }
```

Figure 13: Definition of a SCHC Context

To access to a specific rule, rule-id and its specific length is used as a key. The rule is either a compression or a fragmentation rule.

Each context can be identify though a version id.

3.1. Compression rule

A compression rule is composed of entries describing its processing (cf. Figure 14). An entry contains all the information defined in Figure 1 with the types defined above.

3.1.1. Compression context representation.

The compression rule described Figure 1 is associated to a rule ID. The compression rule entry is defined in Figure 14. Each column in the table is either represented by a leaf or a list. Note that Matching Operators and Compression Decompression actions can have arguments. They are viewed a ordered list of strings and numbers as in target values.

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

     +-------+--+--+--+------------+-----------------+---------------+
     |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 iden
tifer.
     - Field Length : either a positive integer of a function defined as a YAN
G 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.
     ";

     leaf field-id {
         description "Field ID, identify a field in the header with a YANG ide
ntityref.";
         mandatory true;
         type schc:field-id-type;
     }
     leaf field-length {
         description "Field Length in bit or through a function defined as a Y
ANG identityref";
         mandatory true;
         type schc:field-length-type;
     }
     leaf field-position {
         description "field position in the header is a integer. If the field
is not repeated
         in the header the value is 1, and incremented for each repetition of
the field. Position
```

```
            0 means that the position is not important and order may change when
decompressed";
            mandatory true;
            type uint8;
        }
        leaf direction-indicator {
            description "Direction Indicator, a YANG identityref to say if the pa
cket is bidirectionnal,
            up or down";
            mandatory true;
            type schc:direction-indicator-type;
        }
        list target-values {
            description "a list of value to compare with the header field value.
If target value
            is a singleton, position must be 0. For matching-list, should be cons
ecutive position
            values starting from 1.";
            key position;
            uses target-values-struct;
        }
        leaf matching-operator {
            mandatory true;
            type schc:matching-operator-type;
        }
        list matching-operator-value {
            key position;
            uses target-values-struct;
        }
        leaf comp-decomp-action {
            mandatory true;
            type schc:comp-decomp-action-type;
        }
        list comp-decomp-action-value {
            key position;
            uses target-values-struct;
        }
    }
```

Figure 14: Definition of a compression entry

3.1.2. Rule definition

A compression rule is a list of entries.

```
    grouping compression-content {
        description "define a compression rule composed of a list of entries.";
        list entry {
            key "field-id field-position direction-indicator";
            uses compression-rule-entry;
        }
    }
```

Figure 15: Definition of a compression rule

To identify a specific entry Field ID, position and direction are needed.

3.2. Fragmentation rule

Parameters for fragmentation are defined in Annex D of [RFC8724].

Figure 16 gives the first elements found in this structure. It starts with a direction. Since fragmentation rules are unidirectional, they contain a mandatory direction. The type is the same as the one used in compression entries, but the use of bidirectionnal is forbidden.

The next elements describe size of SCHC fragmentation header fields. Only the FCN size is mandatory and value must be higher or equal to 1.

```
    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 direction {
            type schc:direction-indicator-type;
            description "should be up or down, bi directionnal is forbidden.";
            mandatory true;
        }
        leaf dtagsize {
            type uint8;
            description "size in bit of the DTag field";

        }
        leaf wsize {
            type uint8;
            description "size in bit of the window field";
        }
        leaf fcnsize {
            type uint8 {
                range 1..max;
            }
            description "size in bit of the FCN field";
            mandatory true;
        }
...
```

Figure 16: Definition of a fragmentation parameters, SCHC header

RCS algorithm is defined (Figure 17), by default with the CRC computation proposed in [RFC8724]. The algorithms are identified through an identityref specified in the SCHC Data Model and with the type RCS-algorithm-type (Figure 18).

```
...
        leaf RCS-algorithm {
            type RCS-algorithm-type;
            default schc:RFC8724-RCS;
            description "Algoritm used for RCS";
        }
...
```

Figure 17: Definition of a fragmentation parameters, RCS algorithm

```
identity RCS-algorithm-base-type {
    description "identify which algorithm is used to compute RSC.
    The algorithm defines also the size if the RSC field.";
}

identity RFC8724-RCS {
    description "CRC 32 defined as default RCS in RFC8724.";
    base RCS-algorithm-base-type;
}

typedef RCS-algorithm-type {
    type identityref {
        base RCS-algorithm-base-type;
    }
}
```

Figure 18: Definition of identityref for RCS Algorithm

Figure 19 gives the parameters used by the state machine to handle fragmentation:

- maximum-window-size contains the maximum FCN value that can be used.
- retransmission-timer gives in seconds the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specifed, value must be higher or equal to 1.
- inactivity-timer gives in seconds the duration before aborting (cf. section 8.2.2.4. of [RFC8724]), value of 0 explicitly indicates that this timer is disabled.
- max-ack-requests gives the number of attempts before aborting (cf. section 8.2.2.4. of [RFC8724]).
- maximum-packet-size gives in bytes the larger packet size that can be reassembled.

```
...
        leaf maximum-window-size {
            type uint16;
            description "by default 2^wsize - 2";
        }

        leaf retransmission-timer {
            type uint64 {
                range 1..max;
            }
            description "duration in seconds of the retransmission timer"; // Che
ck the units
        }

        leaf inactivity-timer {
            type uint64;
            description "duration is seconds of the inactivity timer, 0 indicates
 the timer is disabled"; // check units
        }

        leaf max-ack-requests {
            type uint8 {
                range 1..max;
            }
            description "the maximum number of retries for a specific SCHC ACK.";
        }

        leaf maximum-packet-size {
            type uint16;
            default 1280;
            description "When decompression is done, packet size must not strictl
y exceed this limit in Bytes";
        }
...
```

Figure 19: Definition of a fragmentation state machine parameters

Figure 20 gives information related to a specific compression mode: fragmentation-mode MUST be set with a specific behavior. Identityref are given Figure 21.

For Ack on Error some specific information may be provided:

- tile-size gives in bits the size of the tile; If set to 0 a single tile is inserted inside a fragment.

- tile-in All1 indicates if All1 contains only the RCS (all1-data-no) or may contain a single tile (all1-data-yes). Since the reassembly process may detect this behavior, the choice can be left to the fragmentation process. In that case identityref all1-

data-sender-choice as to be specified. All possible values are given Figure 21.

- ack-behavior tells when the fragmentation process may send acknowledgments. When ack-behavior-after-All0 is specified, the ack may be sent after the reception of All-0 fragment. When ack-behavior-after-All1 is specified, the ack may be sent after the reception of All-1 fragment at the end of the fragmentation process. ack-behavior-always do not impose a limitation at the SCHC level. The constraint may come from the LPWAN technology. All possible values are given Figure 21.

```
...
        leaf fragmentation-mode {
            type schc:fragmentation-mode-type;
            description "which fragmentation mode is used (noAck, AckAlways, Acko
nError)";
            mandatory true;
        }

        choice mode {
            case no-ack;
            case ack-always;
            case ack-on-error {
                leaf tile-size {
                    type uint8;
                    description "size in bit of tiles, if not specified or set to
 0: tile fills the fragment.";
                }
                leaf tile-in-All1 {
                    type schc:all1-data-type;
                    description "When true, sender and receiver except a tile in
All-1 frag";
                }
                leaf ack-behavior {
                    type schc:ack-behavior-type;
                    description "Sender behavior to acknowledge, after All-0, All
-1 or when the
                    LPWAN allows it (Always)";
                }
            }
       }
...
```

Figure 20: Definition of a fragmentation specific information

```
// -- FRAGMENTATION TYPE

// -- fragmentation modes

    identity fragmentation-mode-base-type {
        description "fragmentation mode";
```

```
    }

    identity fragmentation-mode-no-ack {
        description "No Ack of RFC 8724.";
        base fragmentation-mode-base-type;
    }

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

    typedef fragmentation-mode-type {
        type identityref {
            base fragmentation-mode-base-type;
        }
    }

// -- Ack behavior

    identity ack-behavior-base-type {
        description "define when to send an Acknowledgment message";
    }

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

    identity ack-behavior-after-All1 {
        description "fragmentation expects Ack after sending All1 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-always {
        description "fragmentation expects Ack after sending every fragment.";
        base ack-behavior-base-type;
    }

    typedef ack-behavior-type {
        type identityref {
            base ack-behavior-base-type;
        }
    }
```

```
// -- All1 with data types

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

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

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

    identity all1-data-sender-choice {
        description "Fragmentation process choose to send tiles or not in all1.";
        base all1-data-base-type;
    }

    typedef all1-data-type {
        type identityref {
            base all1-data-base-type;
        }
    }
```

Figure 21: Specific types for Ack On Error mode

YANG Tree

```
module: schc
  +--rw schc
     +--rw version?   uint64
     +--rw rule* [rule-id rule-length]
        +--rw rule-id                        uint32
        +--rw rule-length                    uint8
        +--rw (nature)?
           +--:(fragmentation)
           |  +--rw direction                schc:direction-indicator-type
           |  +--rw dtagsize?                uint8
           |  +--rw wsize?                   uint8
           |  +--rw fcnsize                  uint8
           |  +--rw RCS-algorithm?           RCS-algorithm-type
           |  +--rw maximum-window-size?     uint16
           |  +--rw retransmission-timer?    uint64
           |  +--rw inactivity-timer?        uint64
           |  +--rw max-ack-requests?        uint8
           |  +--rw maximum-packet-size?     uint16
           |  +--rw fragmentation-mode       schc:fragmentation-mode-type
           |  +--rw (mode)?
           |     +--:(no-ack)
           |     +--:(ack-always)
           |     +--:(ack-on-error)
           |        +--rw tile-size?         uint8
           |        +--rw tile-in-All1?      schc:all1-data-type
           |        +--rw ack-behavior?      schc:ack-behavior-type
           +--:(compression)
              +--rw entry* [field-id field-position direction-indicator]
                 +--rw field-id                    schc:field-id-type
                 +--rw field-length                schc:field-length-type
                 +--rw field-position              uint8
                 +--rw direction-indicator         schc:direction-indicator-type
                 +--rw target-values* [position]
                 |  +--rw value?      union
                 |  +--rw position    uint16
                 +--rw matching-operator           schc:matching-operator-type
                 +--rw matching-operator-value* [position]
                 |  +--rw value?      union
                 |  +--rw position    uint16
                 +--rw comp-decomp-action          schc:comp-decomp-action-type
                 +--rw comp-decomp-action-value* [position]
                    +--rw value?      union
                    +--rw position    uint16
```

Figure 22

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 on [RFC8724]

6. Acknowledgements

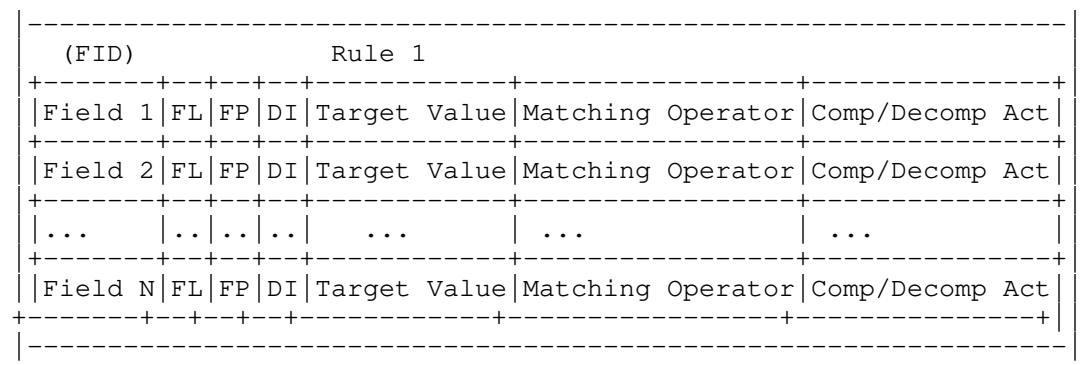
The authors would like to thank Dominique Barthel, Carsten Bormann, Alexander Pelov.

7. YANG Module

```
<code begins> file schc@2020-02-28.yang
module schc{
    yang-version "1";
    namespace "urn:ietf:lpwan:schc:rules-description";
    prefix "schc";

    description
    "Generic Data model for Static Context Header Compression Rule for SCHC,
    based on draft-ietf-lpwan-ipv6-static-context-hc-18. Include compression
    rules and fragmentation rules.

    This module is a YANG model for SCHC rules (RFc 8724).
    RFC 8724 describes a rule in a abstract way through a 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 describes some operations in fields.
    This data model applies both to compression and fragmentation.";
```

```
    revision 2020-06-15 {
        description "clean up and add descriptions, merge schc-id to this file";
    }

    revision 2020-02-28 {
        description "Add Fragmentation parameters";
    }

    revision 2020-01-23 {
        description "Modified TV with binary and union";
    }

    revision 2020-01-07 {
        description "First version of the YANG model";
    }

// -------------------------
//  Field ID type definition
//-------------------------

// generic value TV definition

    identity field-id-base-type  {
           description "Field ID with SID";
    }

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

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

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

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

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

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

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

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

    identity fid-ipv6-devprefix {
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 8200. Depending if it is respec
tively
                             a uplink or an downklink message.";
    }

    identity fid-ipv6-deviid {
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 8200. Depending if it is respec
tively
                             a uplink or an downklink message.";
    }

    identity fid-ipv6-appprefix {
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 768. Depending if it is respect
ively
                             a downlink or an uplink message.";
    }

    identity fid-ipv6-appiid {
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 768. Depending if it is respect
ively
                             a downlink or an uplink message.";
    }
```

```
    identity fid-udp-dev-port {
        base field-id-base-type;
        description "UDP length from RFC 768";
    }

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

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

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

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

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

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

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

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

    identity fid-coap-code-detail {
        base field-id-base-type;
        description "CoAP code detail from RFC 7252";
```

```
    }

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

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

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

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

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

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

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

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

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

    identity fid-coap-option-uri-path {
```

```
        base field-id-base-type;
        description "CoAP option Uri-Path from RFC 7252";
    }

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

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

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

    identity fid-coap-option-accept {
        base field-id-base-type;
        description "CoAP option Max-Age from RFC 7252";
    }

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

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

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

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

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

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

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

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

    identity fid-coap-option-oscore-flags {
        base field-id-base-type;
        description "CoAP option oscore flags (see draft schc coap, section 6.4)"
;
    }

    identity fid-coap-option-oscore-piv {
        base field-id-base-type;
        description "CoAP option oscore flags (see draft schc coap, section 6.4)"
;
    }

    identity fid-coap-option-oscore-kid {
        base field-id-base-type;
        description "CoAP option oscore flags (see draft schc coap, section 6.4)"
;
    }

    identity fid-coap-option-oscore-kidctx {
        base field-id-base-type;
        description "CoAP option oscore flags (see draft schc coap, section 6.4)"
;
    }

    identity fid-icmpv6-type {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }

    identity fid-icmpv6-code {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }

    identity fid-icmpv6-checksum {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
```

```
    }

    identity fid-icmpv6-identifier {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }

    identity fid-icmpv6-sequence {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }


/// !!!!!!! See future CoAP extentions

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

  identity field-length-base-type {
        description "used to extend field length functions";
  }

  identity fl-variable {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }

  identity fl-token-length {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }

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

  identity direction-indicator-base-type {
        description "used to extend field length functions";
  }

  identity di-bidirectional {
           base direction-indicator-base-type;
           description "Direction Indication of bi directionality";
  }

  identity di-up {
           base direction-indicator-base-type;
```

```
            description "Direction Indication of upstream";
    }

    identity di-down {
            base direction-indicator-base-type;
            description "Direction Indication of downstream";
    }

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

     identity matching-operator-base-type {
        description "used to extend Matching Operators with SID values";
     }

     identity mo-equal {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-ignore {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-msb {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-matching {
        base matching-operator-base-type;
        description "RFC 8724";
     }

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

      identity compression-decompression-action-base-type;

      identity cda-not-sent {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-value-sent {
```

```
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-lsb {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-mapping-sent {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-compute-length {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-compute-checksum {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-deviid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

   identity cda-appiid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

// -- type definition

    typedef field-id-type {
        description "Field ID generic type.";
        type identityref {
            base field-id-base-type;
        }
    }

    typedef field-length-type {
        description "Field length either a positive integer giving the size in bi
ts
        or a function defined through an identityref.";
        type union {
            type int64; /* positive length in bits */
```

```
            type identityref { /* function */
                base field-length-base-type;
            }
        }
    }

    typedef direction-indicator-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 devi
ce.";
        type identityref {
            base direction-indicator-base-type;
        }
    }

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

    typedef comp-decomp-action-type {
        description "Compression Decompression Action to compression or decompres
s a field.";
        type identityref {
            base compression-decompression-action-base-type;
        }
    }

// -- FRAGMENTATION TYPE

// -- fragmentation modes

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

    identity fragmentation-mode-no-ack {
        description "No Ack of RFC 8724.";
        base fragmentation-mode-base-type;
    }

    identity fragmentation-mode-ack-always {
        description "Ack Always of RFC8724.";
        base fragmentation-mode-base-type;
    }
    identity fragmentation-mode-ack-on-error {
        description "Ack on Error of RFC8724.";
        base fragmentation-mode-base-type;
```

```
    }

    typedef fragmentation-mode-type {
        type identityref {
            base fragmentation-mode-base-type;
        }
    }

// -- Ack behavior

    identity ack-behavior-base-type {
        description "define when to send an Acknowledgment message";
    }

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

    identity ack-behavior-after-All1 {
        description "fragmentation expects Ack after sending All1 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-always {
        description "fragmentation expects Ack after sending every fragment.";
        base ack-behavior-base-type;
    }

    typedef ack-behavior-type {
        type identityref {
            base ack-behavior-base-type;
        }
    }

// -- All1 with data types

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

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

    identity all1-data-yes {
        description "All1 MUST contain a tile";
```

```
        base all1-data-base-type;
    }

    identity all1-data-sender-choice {
        description "Fragmentation process choose to send tiles or not in all1.";
        base all1-data-base-type;
    }

    typedef all1-data-type {
        type identityref {
            base all1-data-base-type;
        }
    }

// -- RCS algorithm types

    identity RCS-algorithm-base-type {
        description "identify which algorithm is used to compute RSC.
        The algorithm defines also the size if the RSC field.";
    }

    identity RFC8724-RCS {
        description "CRC 32 defined as default RCS in RFC8724.";
        base RCS-algorithm-base-type;
    }

    typedef RCS-algorithm-type {
        type identityref {
            base RCS-algorithm-base-type;
        }
    }



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

    grouping target-values-struct {
        description "defines the target value element. Can be either an arbitrary
        binary or ascii element. All target values are considered as a matching l
ists.
        Position is used to order values, by default position 0 is used when cont
aining
        a single element.";

        leaf value {
            type union {
                type binary;
                type string;
            }
        }
```

```
        leaf position {
            description "If only one element position is 0, otherwise position is
 the
            matching list.";
            type uint16;
        }
    }

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

        +-------+--+--+--+------------+-----------------+---------------+
        |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 iden
tifer.
        - Field Length : either a positive integer of a function defined as a YAN
F 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, paramters may be
        associated to that operator.
        - Comp./Decomp. Action: A YANG id giving the compression or decompression
        action, paramters may be associated to that action.
        ";

        leaf field-id {
            description "Field ID, identify a field in the header with a YANG ref
enceid.";
            mandatory true;
            type schc:field-id-type;
        }
        leaf field-length {
            description "Field Length in bit or through a function defined as a Y
ANG referenceid";
            mandatory true;
            type schc:field-length-type;
        }
        leaf field-position {
            description "field position in the header is a integer. If the field
is not repeated
            in the header the value is 1, and incremented for each repetition of
the field. Position
            0 means that the position is not important and order may change when
decompressed";
            mandatory true;
            type uint8;
        }
        leaf direction-indicator {
            description "Direction Indicator, a YANG referenceid to say if the pa
cket is bidirectionnal,
            up or down";
            mandatory true;
```

```
            type schc:direction-indicator-type;
        }
        list target-values {
            description "a list of value to compare with the header field value.
If target value
            is a singleton, position must be 0. For matching-list, should be cons
ecutive position
            values starting from 1.";
            key position;
            uses target-values-struct;
        }
        leaf matching-operator {
            mandatory true;
            type schc:matching-operator-type;
        }
        list matching-operator-value {
            key position;
            uses target-values-struct;
        }
        leaf comp-decomp-action {
            mandatory true;
            type schc:comp-decomp-action-type;
        }
        list comp-decomp-action-value {
            key position;
            uses target-values-struct;
        }
    }

    grouping compression-content {
        description "define a compression rule composed of a list of entries.";
        list entry {
            key "field-id field-position direction-indicator";
            uses compression-rule-entry;
        }
    }

    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 direction {
            type schc:direction-indicator-type;
            description "should be up or down, bi directionnal is forbiden.";
            mandatory true;
        }
        leaf dtagsize {
            type uint8;
            description "size in bit of the DTag field";
```

```
        }
        leaf wsize {
            type uint8;
            description "size in bit of the window field";
        }
        leaf fcnsize {
            type uint8;
            description "size in bit of the FCN field";
            mandatory true;
        }
        leaf RCS-algorithm {
            type RCS-algorithm-type;
            default schc:RFC8724-RCS;
            description "Algoritm used for RCS";
        }
        leaf maximum-window-size {
            type uint16;
            description "by default 2^wsize - 1";
        }

        leaf retransmission-timer {
            type uint64 {
                range 1..max;
            }
            description "duration in seconds of the retransmission timer"; // Che
ck the units
        }

        leaf inactivity-timer {
            type uint64;
            description "duration is seconds of the inactivity timer, 0 indicates
 the timer is disabled"; // check units
        }

        leaf max-ack-requests {
            type uint8 {
                range 1..max;
            }
            description "the maximum number of retries for a specific SCHC ACK.";
        }

        leaf maximum-packet-size {
            type uint16;
            default 1280;
            description "When decompression is done, packet size must not strictl
y exceed this limit in Bytes";
        }

        leaf fragmentation-mode {
            type schc:fragmentation-mode-type;
            description "which fragmentation mode is used (noAck, AckAlways, Acko
nError)";
```

```
            mandatory true;
        }

        choice mode {
            case no-ack;
            case ack-always;
            case ack-on-error {
                leaf tile-size {
                    type uint8;
                    description "size in bit of tiles, if not specified or set to
 0: tile fills the fragment.";
                }
                leaf tile-in-All1 {
                    type schc:all1-data-type;
                    description "When true, sender and receiver except a tile in
All-1 frag";
                }
                leaf ack-behavior {
                    type schc:ack-behavior-type;
                    description "Sender behavior to acknowledge, after All-0, All
-1 or when the
                    LPWAN allows it (Always)";
                }
            }
       }
    }



// Define rule ID. Rule ID is composed of a RuleID value and a Rule ID Length

    grouping rule-id-type {
        leaf rule-id {
            type uint32;
            description "rule ID value, this value must be unique combined with t
he length";
        }
        leaf rule-length {
            type uint8 {
                range 0..32;
            }
            description "rule ID length in bits, value 0 is for implicit rules";
        }
    }

// SCHC table for a specific device.

    container schc {
        leaf version{
            type uint64;
            mandatory false;
            description "used as an indication for versioning";
```

```
        }
        list rule {
            key "rule-id rule-length";
            uses rule-id-type;
            choice nature {
                case fragmentation {
                    uses fragmentation-content;
                }
                case compression {
                    uses compression-content;
                }
            }
        }
    }


}
```

<code ends>

Figure 23

8. Normative References

[I-D.barthel-lpwan-oam-schc]
Barthel, D., Toutain, L., Kandasamy, A., Dujovne, D., and J. Zuniga, "OAM for LPWAN using Static Context Header Compression (SCHC)", draft-barthel-lpwan-oam-schc-01 (work in progress), March 2020.

[I-D.ietf-lpwan-coap-static-context-hc]
Minaburo, A., Toutain, L., and R. Andreasen, "LPWAN Static Context Header Compression (SCHC) for CoAP", draft-ietf-lpwan-coap-static-context-hc-15 (work in progress), July 2020.

[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>.

[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>.

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