

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2021

A. Minaburo
Acklio
L. Toutain
Institut MINES TELECOM; IMT Atlantique
R. Andreasen
Universidad de Buenos Aires
March 08, 2021

LPWAN Static Context Header Compression (SCHC) for CoAP
draft-ietf-lpwan-coap-static-context-hc-19

Abstract

This draft defines how to compress the Constrained Application Protocol (CoAP) using the Static Context Header Compression (SCHC). SCHC is a header compression mechanism adapted for Constrained Devices. SCHC uses a static description of the header to reduce the header's redundancy and size. While RFC 8724 describes the SCHC compression and fragmentation framework, and its application for IPv6/UDP headers, this document applies 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 applying SCHC to flexible headers and how to leverage the asymmetry for more efficient compression 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 September 9, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Terminology	4
2.	SCHC Applicability to CoAP	4
3.	CoAP Headers compressed with SCHC	7
3.1.	Differences between CoAP and UDP/IP Compression	8
4.	Compression of CoAP header fields	9
4.1.	CoAP version field	9
4.2.	CoAP type field	9
4.3.	CoAP code field	9
4.4.	CoAP Message ID field	10
4.5.	CoAP Token fields	10
5.	CoAP options	10
5.1.	CoAP Content and Accept options.	11
5.2.	CoAP option Max-Age, Uri-Host, and Uri-Port fields	11
5.3.	CoAP option Uri-Path and Uri-Query fields	11
5.3.1.	Variable number of Path or Query elements	13
5.4.	CoAP option Size1, Size2, Proxy-URI and Proxy-Scheme fields	13
5.5.	CoAP option ETag, If-Match, If-None-Match, Location-Path, and Location-Query fields	13
6.	SCHC compression of CoAP extension RFCs	13
6.1.	Block	13
6.2.	Observe	13
6.3.	No-Response	14
6.4.	OSCORE	14
7.	Examples of CoAP header compression	15
7.1.	Mandatory header with CON message	15
7.2.	OSCORE Compression	16
7.3.	Example OSCORE Compression	20
8.	IANA Considerations	31
9.	Security considerations	31

10. Acknowledgements	32
11. Normative References	32
Authors' Addresses	33

1. Introduction

CoAP [RFC7252] is a command/response protocol designed for micro-controllers with a small RAM and ROM and optimized for REST-based (Representative state transfer) services. Although the Constrained Devices leads the CoAP design, a CoAP header's size is still too large for LPWAN (Low Power Wide Area Networks). SCHC header compression over CoAP header is required to increase performance or use CoAP over LPWAN technologies.

The [RFC8724] defines SCHC, a header compression mechanism for the LPWAN network based on a static context. Section 5 of the [RFC8724] explains where compression and decompression occur in the architecture. The SCHC compression scheme assumes as a prerequisite that both end-points know the static context before transmission. The way the context is configured, provisioned, or exchanged is out of this document's scope.

CoAP is an application protocol, so CoAP compression requires installing common Rules between the two SCHC instances. SCHC compression may apply at two different levels: at IP and UDP in the LPWAN network and another at the application level for CoAP. These two compressions may be independent. Both follow the same principle described in [RFC8724]. As different entities manage the CoAP compression at different levels, the SCHC Rules driving the compression/decompression are also different. The [RFC8724] describes how to use SCHC for IP and UDP headers. This document specifies how to apply SCHC compression to CoAP headers.

SCHC compresses and decompresses headers based on common contexts between Devices. SCHC context includes multiple Rules. Each Rule can match the header fields to specific values or ranges of values. If a Rule matches, the matched header fields are replaced by the RuleID and the Compression Residue that contains the residual bits of the compression. Thus, different Rules may correspond to different protocol headers in the packet that a Device expects to send or receive.

A Rule describes the packets' entire header 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). The direction indicator is used for compression to give the best TV to

the FID when these values differ in the transmission direction. So a field may be described several times.

A Matching Operator (MO) is associated with each header field description. The Rule is selected if all the MOs fit the TVs for all fields of the incoming header. A Rule cannot be selected if the message contains an unknown field to the SCHC compressor.

In that case, a Compression/Decompression Action (CDA) associated with each field gives the method to compress and decompress each field. Compression mainly results in one of 4 actions:

- o send the field value (value-sent),
- o send nothing (not-sent),
- o send some least significant bits of the field (LSB) or,
- o send an index (mapping-sent).

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

SCHC is a general mechanism applied to different protocols, the exact Rules to be used depending on the protocol and the Application. 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. SCHC Applicability to CoAP

SCHC Compression for CoAP header MAY be done in conjunction with the lower layers (IPv6/UDP) or independently. The SCHC adaptation layers, described in Section 5 of [RFC8724], 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, the Device and the NGW perform SCHC C/D (Static Context Header Compression Compressor/

Decompressor). The Application communicating with the Device does not implement SCHC C/D.

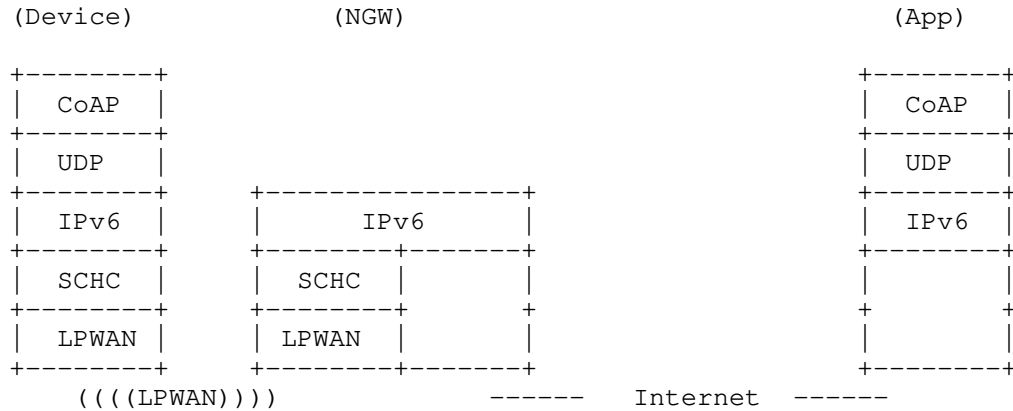


Figure 1: Compression/Decompression at the LPWAN boundary.

Figure 1 shows the use of SCHC header compression above layer 2 in the Device and the NGW. The SCHC layer receives non-encrypted packets and can apply compression Rules to all the headers in the stack. On the other end, the NGW receives the SCHC packet and reconstructs the headers using the Rule and the Compression Residue. After the decompression, the NGW forwards the IPv6 packet toward the destination. The same process applies in the other direction when a non-encrypted packet arrives at the NGW. Thanks to the IP forwarding based on the IPv6 prefix, the NGW identifies the Device and compresses headers using the Device's Rules.

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, the Compression Residue, and CoAP payload are encrypted using a mechanism such as DTLS. Only the other end (App) can decipher the information. If needed, layers below use SCHC to compress the header as defined in [RFC8724] (represented in dotted lines).

This use case needs an end-to-end context initialization between the Device and the Application. The context initialization is out of the scope of this document.

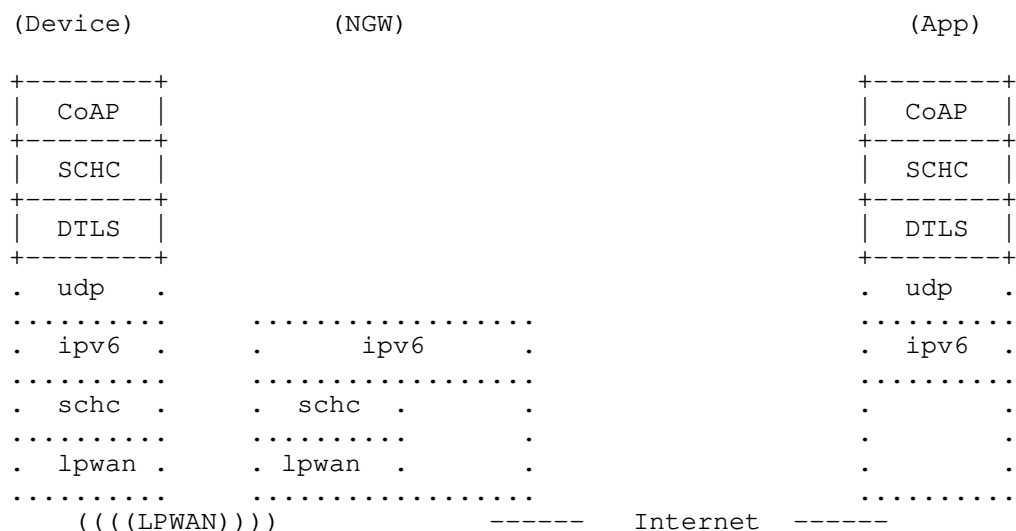


Figure 2: Standalone CoAP end-to-end Compression/Decompression

The third example, Figure 3, shows the use of Object Security for Constrained RESTful Environments (OSCORE) [RFC8613]. In this case, SCHC needs two Rules to compress the CoAP header. A first Rule focused on the inner header. The result of this first compression is encrypted using the OSCORE mechanism. Then a second Rule compresses the outer header, including the OSCORE Options.

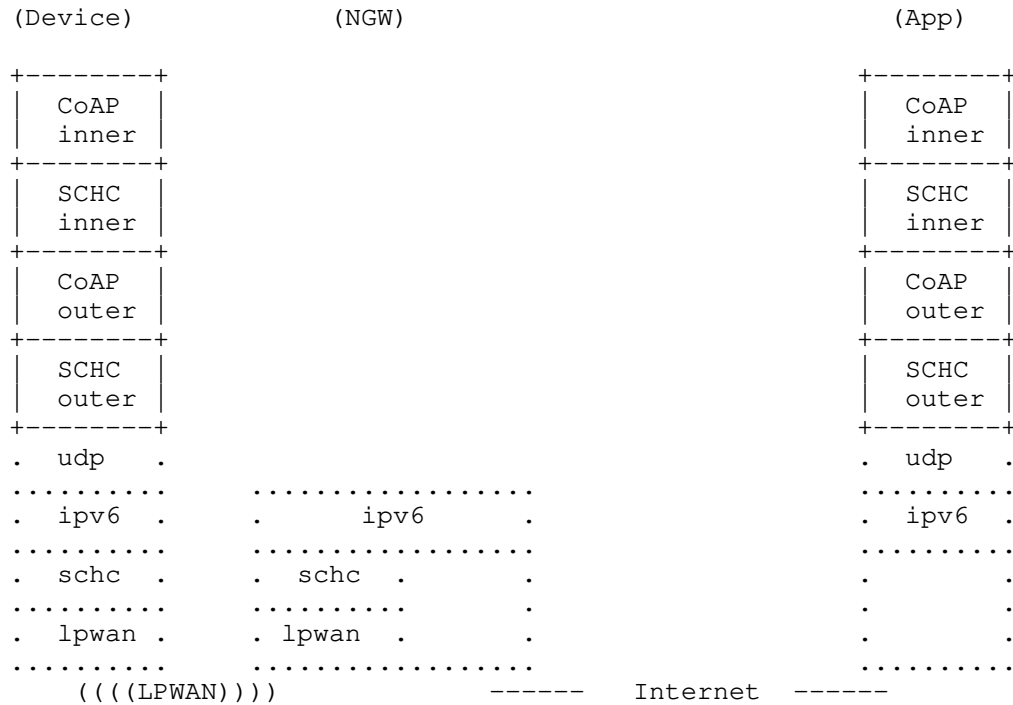


Figure 3: OSCORE compression/decompression.

In the case of several SCHC instances, as shown in Figure 2 and Figure 3, the Rules may come from different provisioning domains.

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 like the one for IP and UDP explained in the [RFC8724]. For CoAP, the 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 header compression 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. Where 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 in the following aspects:

- o 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 might not be present in the response. A request might contain an Accept option, and the response might include a Content-Format option. In comparison, IPv6 and UDP return path swap the value of some fields in the header. However, all the directions have the same fields (e.g., source and destination address fields).

The [RFC8724] defines the use of a direction indicator (DI) in the Field Descriptor, which allows a single Rule to process a message header differently depending on the direction.

- o 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 "match-mapping" MO to limit the range of expected values in a particular direction and reduce the Compression Residue's size. 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 has the same behavior, the 0.0X code format value in the request, and the Y.ZZ code format in the response.
- o In SCHC, the Rule defines the different header fields' length, so SCHC does not need to send it. In IPv6 and UDP headers, the fields have a fixed size, known by definition. On the other hand, some CoAP header fields have variable lengths, and the Rule description specifies it. For example, in a URI-path or URI-query, the Token size may vary from 0 to 8 bytes, and the CoAP options use the Type-Length-Value encoding format.

When doing SCHC compression of a variable-length field, Section 7.5.2 from [RFC8724] offers the possibility to define a function for the Field length in the Field Description to know the length before compression. If the field length is unknown, the

Rule will set it as a variable, and SCHC will send the compressed field's length in the Compression Residue.

- o A field can appear several times in the CoAP headers. It is found typically 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, thereby removing the matching operation's ambiguity.
- o Field lengths defined in the CoAP protocol can be too large regarding LPWAN traffic constraints. For instance, 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, SCHC can apply the Most Significant Bits (MSB) MO 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 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, or if a new version of CoAP is defined, new Rules will be needed to avoid ambiguities between versions.

4.2. CoAP type field

The CoAP protocol [RFC7252] has four types of messages: two requests (CON, NON), one response (ACK), and one empty message (RST).

The SCHC compression SHOULD elide this field if, for instance, a client is sending only NON or only CON messages. For the RST message, SCHC may use a dedicated Rule. For other usages, SCHC can use a "match-mapping" MO.

4.3. CoAP code field

The code field is an IANA registry [RFC7252], and it indicates the Request Method used in CoAP. 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, SCHC may split the code values into two fields description, the request codes with the 0 class and the

response values. SCHC will use the direction indicator to identify the correct value in the packet.

If the Device only implements a CoAP client, SCHC compression may reduce the request code to the set of requests the client can process.

For known values, SCHC can use a "match-mapping" MO. If SCHC cannot compress the code field, it will send the values in the Compression Residue.

4.4. CoAP Message ID field

SCHC can compress the Message ID field with the "MSB" MO and the "LSB" CDA. See section 7.4 of [RFC8724].

4.5. CoAP Token fields

CoAP defines the Token using two CoAP fields, Token Length in the mandatory header and Token Value directly following the mandatory CoAP header.

SCHC processes the Token length as any header field. If the value does not change, the size can be stored in the TV and elided during the transmission. Otherwise, SCHC will send the token length in the Compression Residue.

For the Token Value, SCHC MUST NOT send it as a variable-length in the Compression Residue to avoid ambiguity with Token Length. Therefore, SCHC MUST use the Token length value to define the size of the Compression Residue. SCHC designates a specific function "tkl" that the Rule MUST use to complete the field description. During the decompression, this function returns the value contained in the Token Length field.

5. CoAP options

CoAP defines options placed after the basic header in Option Numbers order; see [RFC7252]. Each Option instance in a message uses the format Delta-Type (D-T), Length (L), Value (V). The SCHC Rule 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 well-known size, the Rule may keep the length value. 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 requests and responses do not include the same options. So Compression Rules may reflect this asymmetry 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.

If CoAP introduces a new option, the SCHC Rules MAY be updated, and the new Field ID description MUST be assigned to allow its compression. Otherwise, if no Rule describes this new option, the SCHC compression is not achieved, and SCHC sends the CoAP header without compression.

5.1. CoAP Content and Accept options.

If the client expects a single value, it can be stored in the TV and elided during the transmission. Otherwise, if the client expects several possible values, a "match-mapping" SHOULD be used to limit the Compression Residue's size. If not, SCHC has to send the option value in the Compression Residue (fixed or variable length).

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

SCHC compresses these three fields in the same way. When the value of these options is known, SCHC can elide these fields. If the option uses well-known values, SCHC can use a "match-mapping" MO. Otherwise, SCHC will use "value-sent" MO, and the Compression Residue will send these options' values.

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

The Uri-Path and Uri-Query fields are repeatable options; this means that in the CoAP header, they may appear several times with different values. SCHC Rule description uses the Field Position (FP) to distinguish the different instances in the path.

To compress repeatable field values, SCHC may use a "match-mapping" MO to reduce the size of variable Paths or Queries. In these cases, to optimize the compression, several elements can be regrouped into a single entry. The Numbering of elements does not change, and the first matching element sets the MO comparison.

Field	FL	FP	DI	Target Value	Matching Operator	CDA
Uri-Path		1	up	["/a/b", "/c/d"]	match-mapping	mapping-sent
Uri-Path	var	3	up		ignore	value-sent

Figure 4: complex path example

In Figure 4, SCHC can use a single bit in the Compression Residue to code one of the two paths. If regrouping were not allowed, 2 bits in the Compression Residue would be needed. SCHC sends the third path element as a variable size in the Compression Residue.

The length of URI-Path and URI-Query may be known when the rule is defined. In any case, SCHC MUST set the field length to variable. The unit to indicate the Compression Residue size is in Byte.

SCHC compression can use the MSB MO to a Uri-Path or Uri-Query element. However, attention to the length is important because the MSB value is in bits, and the size MUST always be a multiple of 8 bits.

The length sent at the beginning of a variable-length Compression Residue indicates the LSB's size in bytes.

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

Field	FL	FP	DI	Target Value	Match Opera.	CDA
Uri-Path		1	up	"c"	equal	not-sent
Uri-Path	var	2	up		ignore	value-sent
Uri-Query	var	1	up	"k=\\"	MSB(24)	LSB

Figure 5: CORECONF URI compression

Figure 5 shows the Rule description for a URI-Path and a URI-Query. SCHC compresses the first part of the URI-Path with a "not-sent" CDA. SCHC will send the second element of the URI-Path with the length (i.e., 0x2 X 6) followed by the query option (i.e., 0x05 eth0").

5.3.1. Variable number of Path or Query elements

SCHC fixed the number of Uri-Path or Uri-Query elements in a Rule at the Rule creation time. If the number varies, SCHC SHOULD create several Rules to cover all the possibilities. Another one is to define the length of Uri-Path to variable and sends a Compression Residue with a length of 0 to indicate that this Uri-Path is empty. However, this adds 4 bits to the variable Compression Residue size. See section 7.5.2 [RFC8724].

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

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

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

A Rule entry cannot store these fields' values. The Rule description MUST always send these values in the Compression Residue.

6. SCHC compression of CoAP extension RFCs

6.1. Block

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

6.2. Observe

The [RFC7641] defines the Observe option. The SCHC Rule description will not define the TV, but MO to "ignore," and the CDA 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 the Observe option MAY use an RST message to inform a server that the client does not require the Observe response, a specific

SCHC Rule SHOULD exist to allow the message's compression with the RST type.

6.3. No-Response

The [RFC7967] defines a No-Response option limiting the responses made by a server to a request. Different behaviors exist while using this option to limit the responses made by a server to a request. If both ends know the value, then the SCHC Rule will describe a TV to this value, with a MO set to "equal" and CDA set to "not-sent."

Otherwise, if the value is changing over time, the SCHC Rule will set the MO to "ignore" and CDA to "value-sent." The Rule may also use a "match-mapping" to compress this option.

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.

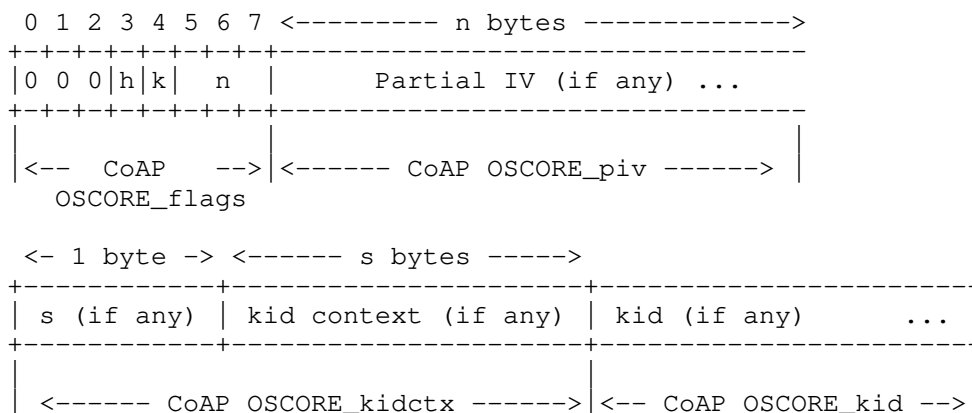


Figure 6: OSCORE Option

The Figure 6 shows the OSCORE Option Value encoding defined in Section 6.1 of [RFC8613], where 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 kid context field's length is encoded in the first byte denoting by 's' the length of the kid context in bytes.

To better perform OSCORE SCHC compression, the Rule description needs to identify the OSCORE Option and the fields it contains. Conceptually, it discerns up to 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:

- o CoAP OSCORE_flags,
- o CoAP OSCORE_piv,
- o CoAP OSCORE_kidctx,
- o CoAP OSCORE_kid.

Figure 6 shows the OSCORE Option format with those four fields superimposed on it. Note that the CoAP OSCORE_kidctx field directly includes the size octet s.

7. Examples of CoAP header compression

7.1. Mandatory header with CON message

In this first scenario, the SCHC Compressor at the Network Gateway side receives a POST message from an Internet client, which is immediately acknowledged by the Device. Figure 7 describes the SCHC Rule descriptions for this scenario.

RuleID 1

Field	FL	FP	DI	Target Value	Match Opera.	CDA	Sent [bits]
CoAP version	2	1	bi	01	equal	not-sent	
CoAP Type	2	1	dw	CON	equal	not-sent	
CoAP Type	2	1	up	[ACK, RST]	match-mapping	matching-sent	T
CoAP TKL	4	1	bi	0	equal	not-sent	
CoAP Code	8	1	bi	[0.00, ... 5.05]	match-mapping	matching-sent	CC CCC
CoAP MID	16	1	bi	0000	MSB(7)	LSB	M-ID
CoAP Uri-Path	var	1	dw	path	equal 1	not-sent	

Figure 7: CoAP Context to compress header without Token

In this example, SCHC compression elides the version and the Token Length fields. The 26 method and response codes defined in [RFC7252] has been shrunk to 5 bits using a "match-mapping" MO. The Uri-Path contains a single element indicated in the TV and elided with the CDA "not-sent."

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 client located in an Application Server sending a request to a server located in the Device may not be compressed through this Rule since the MID might not start with 7 bits equal to 0. A CoAP proxy placed before the SCHC C/D can rewrite the message ID to fit the value and match the Rule.

7.2. OSCORE Compression

OSCORE aims to solve the problem of end-to-end encryption for CoAP messages. Therefore, the goal is to hide as much as possible the message 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 that is not necessary for proxy operation. However, it is part of the message that can be encrypted

until it reaches its end destination. The Outer Message acts as a shell matching the regular CoAP message format and includes all Options and information needed for proxy operation and caching. Figure 8 illustrates this analysis.

The CoAP protocol arranges the options into one of 3 classes; each granted a specific type of protection by the protocol:

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

These classes point out that the Outer option contains the OSCORE Option and that the message is OSCORE protected; this option carries the information necessary to retrieve the Security Context. The endpoint will use this Security Context to decrypt the message correctly.

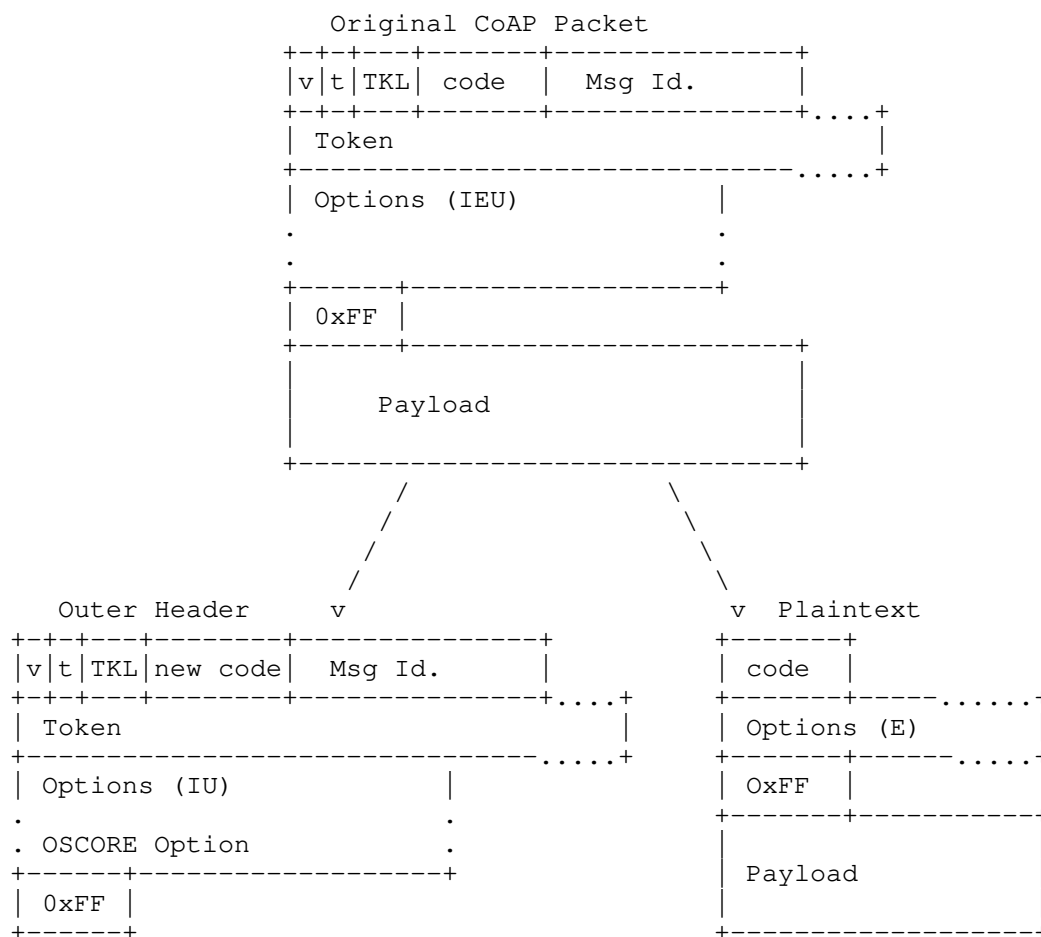


Figure 8: A CoAP packet is split into an OSCORE outer and plaintext

Figure 8 shows the packet format for the OSCORE Outer header and Plaintext.

In the Outer Header, the original header code is hidden and replaced by a default dummy value. As seen in Sections 4.1.3.5 and 4.2 of [RFC8613], the message code is replaced by POST for requests and Changed for responses when CoAP is not using the Observe option. If CoAP uses Observe, the OSCORE message code is replaced by FETCH for requests and Content for responses.

The first byte of the Plaintext contains the original packet code, followed by the message code, the class E options, and, if present, the original message Payload preceded by its payload marker.

An AEAD algorithm now encrypts the Plaintext. This integrity protects the Security Context parameters and, eventually, any class I options from the Outer Header. The resulting Ciphertext becomes the new payload of the OSCORE message, as illustrated in Figure 9.

As defined in [RFC5116], this Ciphertext is the encrypted Plaintext's concatenation of the authentication tag. Note that Inner Compression only affects the Plaintext before encryption. Thus only the first variable-length of the Ciphertext can be reduced. The authentication tag is fixed in length and is considered part of the cost of protection.

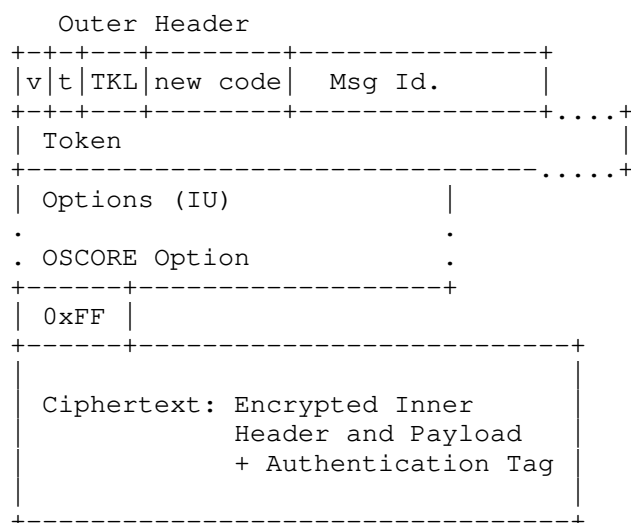


Figure 9: OSCORE message

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

The OSCORE message 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 corresponding end-point can only decrypt the Inner part of the message, this end-point will also have to implement Inner SCHC Compression/Decompression.

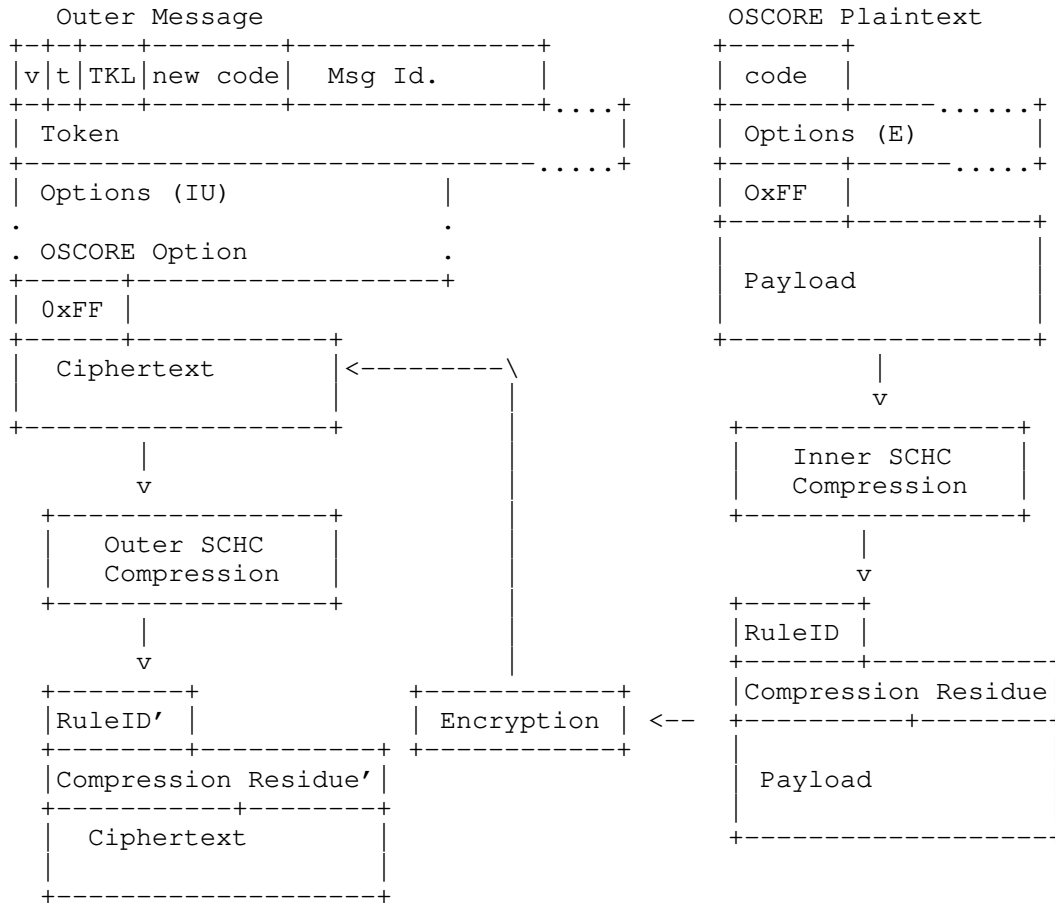


Figure 10: OSCORE Compression Diagram

7.3. Example OSCORE Compression

This section gives an example with a GET Request and its consequent Content Response from a Device-based CoAP client to a cloud-based CoAP server. The example also describes a possible set of Rules for the Inner and Outer SCHC Compression. A dump of the results and a

contrast between SCHC + OSCORE performance with SCHC + COAP performance is also listed. This example gives an approximation of the cost of security with SCHC-OSCORE.

Our first 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 already present in a regular CoAP message. The methods described in Section 4 apply to these fields. As an example, see Figure 13.

RuleID 0

Field	FL	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP Code	8	1	up	1	equal	not-sent	
CoAP Code	8	1	dw	[69, 132]	match-mapping	mapping-sent	c
CoAP Uri-Path		1	up	temperature	equal	not-sent	

Figure 13: Inner SCHC Rules

Figure 14 shows the Plaintext obtained for the example GET request. The packet follows the process of Inner Compression and Encryption until the payload. The outer OSCORE Message adds the result of the Inner process.

In this case, the original message has no payload, and its resulting Plaintext compressed up to only 1 byte (size of the RuleID). The AEAD algorithm preserves this length in its first output and yields a

fixed-size tag. SCHC cannot compress the tag, and the OSCORE message must include it without compression. The use of integrity protection translates into an overhead in total message length, limiting the amount of compression that can be achieved and plays into the cost of adding security to the exchange.

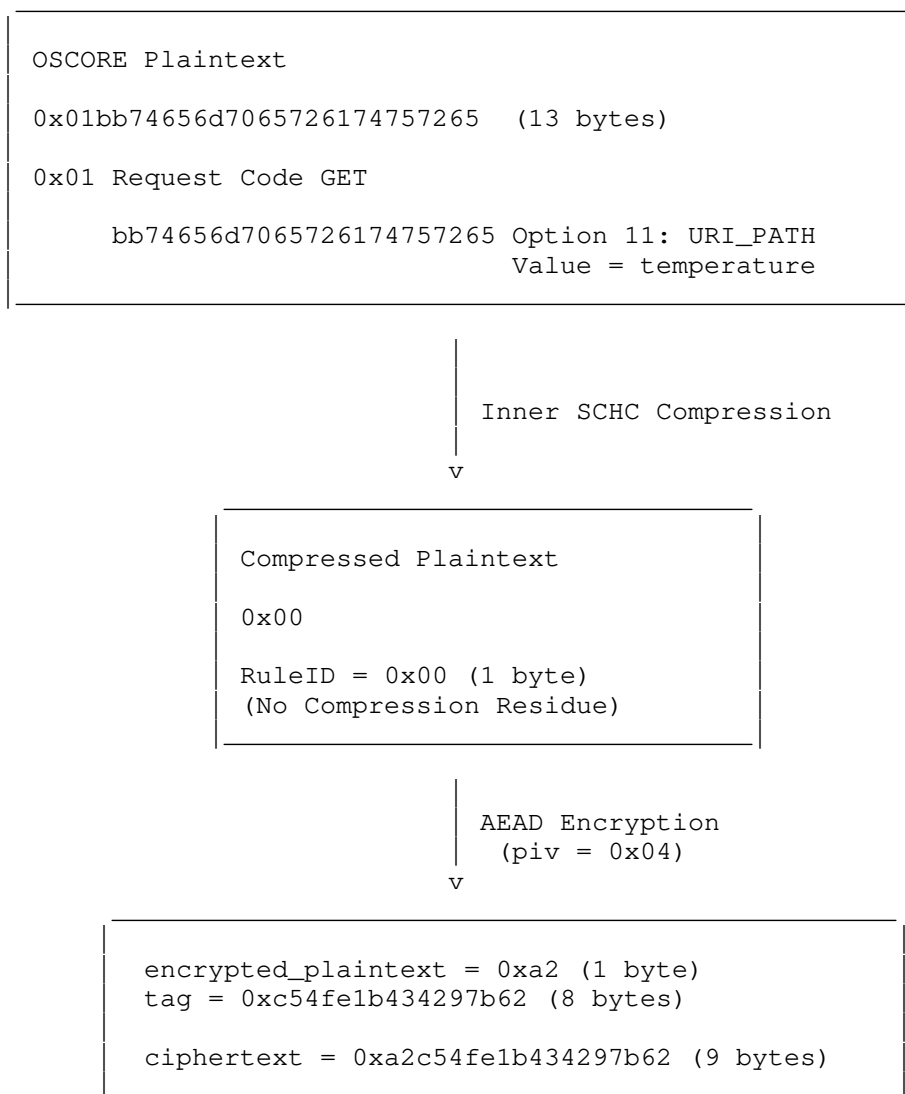


Figure 14: Plaintext compression and encryption for GET Request

Figure 15 shows the process for the example CONTENT Response. The Compression 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 if SCHC cannot compress the tag. The overhead for the tag bytes limits the SCHC's performance but brings security to the transmission.

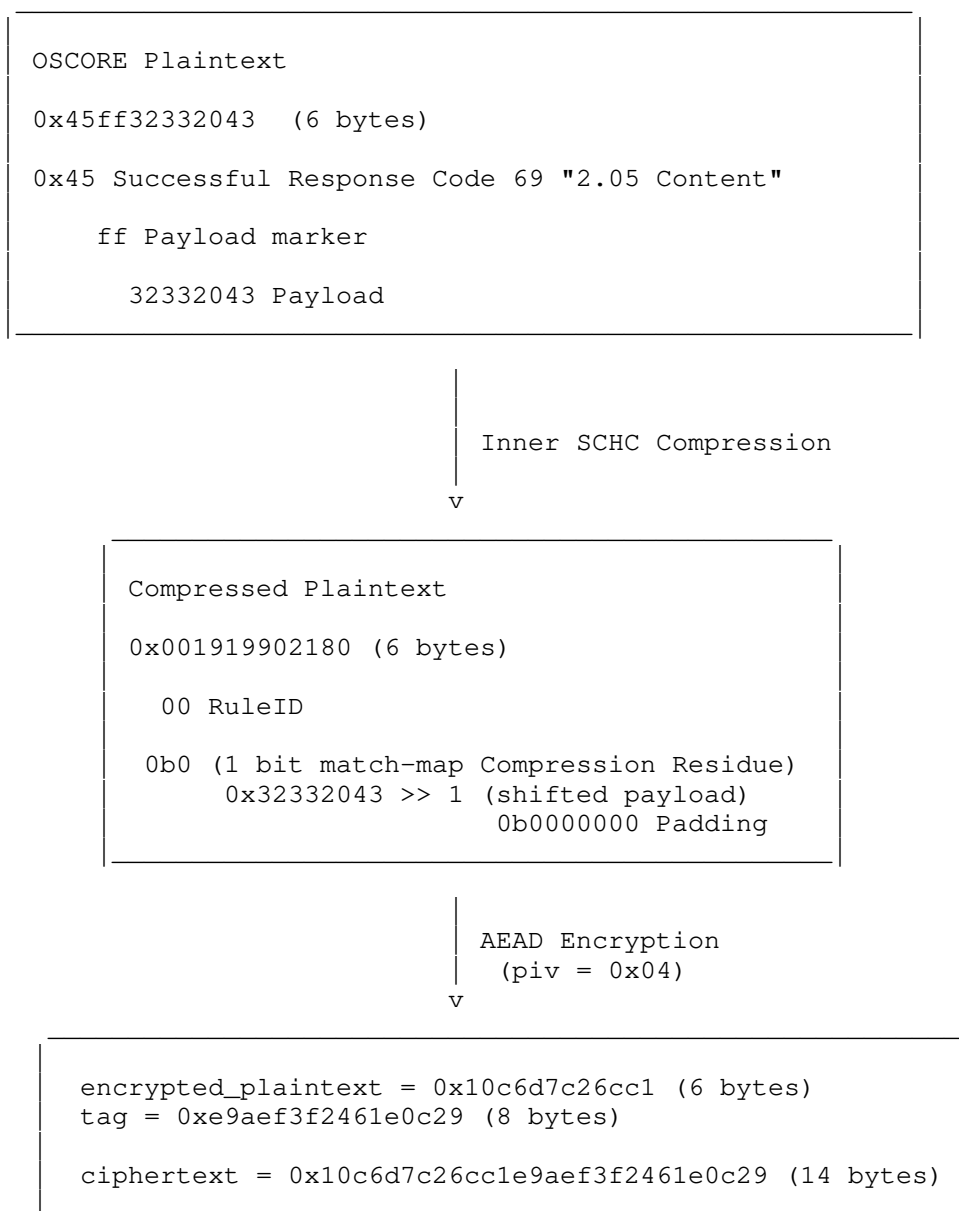


Figure 15: Plaintext compression and encryption for CONTENT Response

The Outer SCHC Rules (Figure 18) must process the OSCORE Options fields. Figure 16 and Figure 17 shows a dump of the OSCORE Messages

generated from the example messages. They include 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:

=====

0x6144000182d008ff10c6d7c26ccl9aef3f2461e0c29
(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:

0x10c6d7c26ccl9aef3f2461e0c29 (14 bytes)

Figure 17: Protected and Inner SCHC Compressed CONTENT Response

For the flag bits, some SCHC compression methods are useful, depending on the Application. The most straightforward alternative is to provide a fixed value for the flags, combining MO "equal" and CDA "not-sent." This SCHC definition saves most bits but could prevent flexibility. Otherwise, SCHC could use a "match-mapping" MO to choose from several configurations for the exchange. If not, the SCHC description may use an "MSB" MO to mask off the three hard-coded most significant bits.

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

The piv field lends itself to having some bits masked off with "MSB" MO and "LSB" CDA. This SCHC description could be useful in applications where the message frequency is low such as LPWAN technologies. Note that compressing the sequence numbers may reduce the maximum number of sequence numbers that can be used in an exchange. Once the sequence number exceeds the maximum value, the OSCORE keys need to be re-established.

The size *s* included in the kid context field MAY be masked off with "LSB" CDA. The rest of the field could have additional bits masked off or have the whole field 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	FL	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP version	2	1	bi	01	equal	not-sent	
CoAP Type	2	1	up	0	equal	not-sent	
CoAP Type	2	1	dw	2	equal	not-sent	
CoAP TKL	4	1	bi	1	equal	not-sent	
CoAP Code	8	1	up	2	equal	not-sent	
CoAP Code	8	1	dw	68	equal	not-sent	
CoAP MID	16	1	bi	0000	MSB(12)	LSB	MMMM
CoAP Token	tkl	1	bi	0x80	MSB(5)	LSB	TTT
CoAP OSCORE_flags	8	1	up	0x09	equal	not-sent	
CoAP OSCORE_piv	var	1	up	0x00	MSB(4)	LSB	PPPP
CoAP OSCORE_kid	var	1	up	0x636c69656e70	MSB(52)	LSB	KKKK
CoAP OSCORE_kidctx	var	1	bi	b''	equal	not-sent	
CoAP OSCORE_flags	8	1	dw	b''	equal	not-sent	
CoAP OSCORE_piv	var	1	dw	b''	equal	not-sent	
CoAP OSCORE_kid	var	1	dw	b''	equal	not-sent	

Figure 18: Outer SCHC Rules

The Outer Rule of Figure 18 is applied to the example GET Request and CONTENT Response. Figure 19 and Figure 20 show the resulting messages.

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

In contrast, comparing these results with what would be obtained by SCHC compressing the original CoAP messages without protecting them with OSCORE is done by compressing the CoAP messages according to the SCHC Rules in Figure 21.

RuleID 1

Field	FL	FP	DI	Target Value	MO	CDA	Sent [bits]
CoAP version	2	1	bi	01	equal	not-sent	
CoAP Type	2	1	up	0	equal	not-sent	
CoAP Type	2	1	dw	2	equal	not-sent	
CoAP TKL	4	1	bi	1	equal	not-sent	
CoAP Code	8	1	up	2	equal	not-sent	
CoAP Code	8	1	dw	[69,132]	match-mapping	mapping-sent	
CoAP MID	16	1	bi	0000	MSB(12)	LSB	C MMMM
CoAP Token	tkl	1	bi	0x80	MSB(5)	LSB	TTT
CoAP Uri-Path		1	up	temperature	equal	not-sent	

Figure 21: SCHC-CoAP Rules (No OSCORE)

Figure 21 Rule yields the SCHC compression results in Figure 22 for 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.

8. IANA Considerations

This document has no request to IANA.

9. Security considerations

The use of SCHC header compression for CoAP header fields only affects the representation of the header information. SCHC header compression itself does not increase or decrease the overall level of security of the communication. When the connection does not use a security protocol (such as OSCORE, DTLS, etc.), it is necessary to use a layer-two security mechanism to protect the SCHC messages.

If LPWAN is the layer-two technology, the SCHC security considerations of [RFC8724] continue to apply. When using another layer-two protocol, use of a cryptographic integrity-protection mechanisms to protect the SCHC headers is REQUIRED. Such cryptographic integrity protection is necessary in order to continue to provide the properties that [RFC8724] relies upon.

When SCHC is used with OSCORE, the security considerations of [RFC8613] continue to apply.

When SCHC is used with the OSCORE outer headers, the Initialization Vector (IV) size in the Compression Residue must be carefully selected. There is a tradeoff between compression efficiency (with a longer "MSB" MO prefix) and the frequency at which the Device must renew its key material (in order to prevent the IV from expanding to

an uncompressable value). The key renewal operation itself requires several message exchanges and requires energy-intensive computation, but the optimal tradeoff will depend on the specifics of the device and expected usage patterns.

If an attacker can introduce a corrupted SCHC-compressed packet onto a link, DoS attacks are possible by causing excessive resource consumption at the decompressor. However, an attacker able to inject packets at the link layer is also capable of other, potentially more damaging, attacks.

SCHC compression emits variable-length Compression Residues for some CoAP fields. In the compressed header representation, the length field that is sent is not the length of the original header field but rather the length of the Compression Residue that is being transmitted. If a corrupted packet arrives at the decompressor with a longer or shorter length than the original compressed representation possessed, the SCHC decompression procedures will detect an error and drop the packet.

SCHC header compression rules MUST remain tightly coupled between compressor and decompressor. If the compression rules get out of sync, a Compression Residue might be decompressed differently at the receiver than the initial message submitted to compression procedures. Accordingly, any time the context Rules are updated on an OSCORE endpoint, that endpoint MUST trigger OSCORE key re-establishment. Similar procedures may be appropriate to signal Rule updates when other message-protection mechanisms are in use.

10. Acknowledgements

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

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>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/info/rfc5116>>.

- [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: Informational
Expires: August 26, 2021

JC. Zuniga
SIGFOX
C. Gomez
S. Aguilar
Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
S. Cespedes
D. Wistuba
NIC Labs, Universidad de Chile
February 22, 2021

SCHC over Sigfox LPWAN
draft-ietf-lpwan-schc-over-sigfox-05

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 August 26, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. SCHC: Generic Framework for Static Context Header Compression and Fragmentation	3
4. SCHC over Sigfox	3
4.1. Network Architecture	3
4.2. Uplink	5
4.3. Downlink	6
4.4. SCHC-ACK on Downlink	7
4.5. SCHC Rules	7
4.6. Fragmentation	7
4.6.1. Uplink Fragmentation	8
4.6.2. Downlink Fragmentation	11
4.7. Padding	12
5. Fragmentation Sequence Examples	12
5.1. Uplink No-ACK Examples	12
5.2. Uplink ACK-on-Error Examples: Single-byte SCHC Header	13
5.3. SCHC Abort Examples	19
6. Security considerations	21
7. Acknowledgements	21
8. References	21
8.1. Normative References	21
8.2. Informative References	22
Authors' Addresses	22

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.

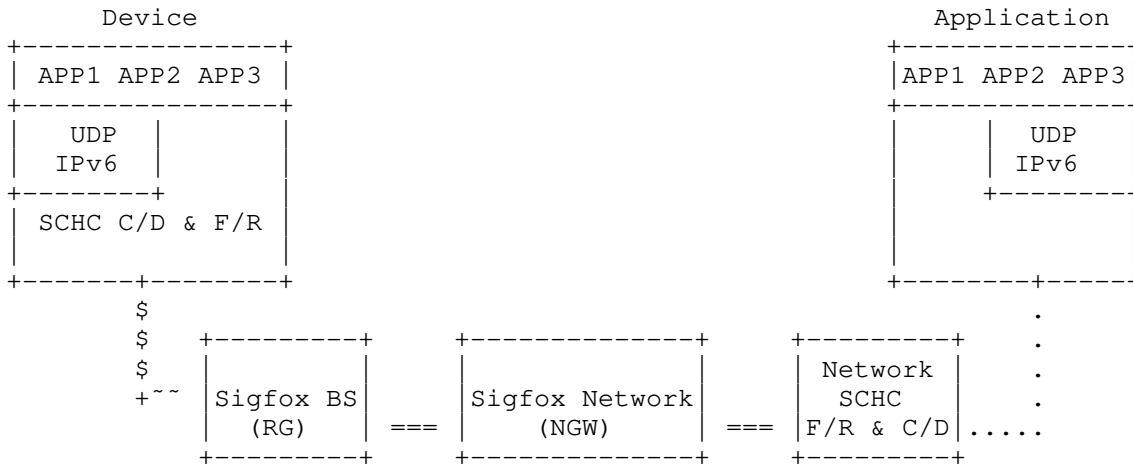


Figure 1: Network Architecture

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:

- o Device ID
- o Message Sequence Number
- o Message Payload
- o Message Timestamp
- o Device Geolocation (optional)
- o RSSI (optional)
- o Device Temperature (optional)
- o 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.

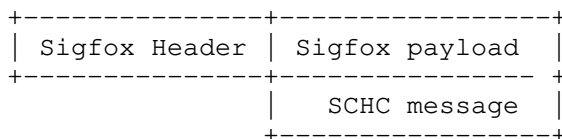


Figure 2: SCHC Message in Sigfox

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. SCHC-ACK on Downlink

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

For the UL ACK-on-Error fragmentation mode, a DL opportunity MUST be indicated by the last fragment of every window (i.e. FCN = All-0, or FCN = All-1). The Device sends the fragments in sequence and, after transmitting the FCN = All-0 or FCN = All-1, it opens up a reception opportunity. The Network SCHC can then decide to respond at that opportunity (or wait for a further one) with a SCHC-ACK indicating in case there are missing fragments from the current or previous windows. If there is no SCHC-ACK to be sent, or if the network decides to wait for a further DL transmission opportunity, then no DL transmission takes place at that opportunity and after a timeout the UL transmissions continue. Intermediate SCHC fragments with FCN different from All-0 or All-1 MUST NOT use the DL request flag to request a SCHC-ACK.

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:

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

- o Rule ID size: 3 bits
- o DTag size (T): 0 bits
- o Window index (W) size (M): 2 bits
- o Fragment Compressed Number (FCN) size (N): 3 bits
- o MAX_ACK_REQUESTS: 5
- o WINDOW_SIZE: 7 (with a maximum value of FCN=0b110)

- o Tile size: 11 bytes
- o Retransmission Timer: Application-dependent
- o Inactivity Timer: Application-dependent
- o 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:

- o Rule ID size is: 8 bits
- o DTag size (T) is: 0 bits
- o Window index (W) size (M): 3 bits
- o Fragment Compressed Number (FCN) size (N): 5 bits.
- o MAX_ACK_REQUESTS: 5
- o WINDOW_SIZE: 31 (with a maximum value of FCN=0b111110)
- o Tile size: 10 bytes
- o Retransmission Timer: Application-dependent
- o Inactivity Timer: Application-dependent
- o 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:

- o RuleID size: 3 bits
- o DTag size (T): 0 bits

- o Window index (W) size (M) is: 0 bits
- o Fragment Compressed Number (FCN) size (N): 5 bits
- o MAX_ACK_REQUESTS: 5
- o WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)
- o Tile size: 7 bytes
- o Retransmission Timer: Application-dependent
- o Inactivity Timer: Application-dependent
- o 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. Fragmentation Sequence Examples

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

5.1. Uplink No-ACK Examples

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

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

```

Sender                                     Receiver
|-----FCN=6 (0110), Seq=1----->|
|-----FCN=5 (0101), Seq=2----->|
|-----FCN=4 (0100), Seq=3----->|
|-----FCN=3 (0011), Seq=4----->|
|-----FCN=2 (0010), Seq=5----->|
|-----FCN=1 (0001), Seq=6----->|
|-----FCN=15 (1111), Seq=7----->| All fragments received
(End)

```

Figure 3: UL No-ACK No-Losses

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

Case losses on any fragment except the first.

```

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

```

Figure 4: UL No-ACK Losses (scenario 1)

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

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

Case No losses

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

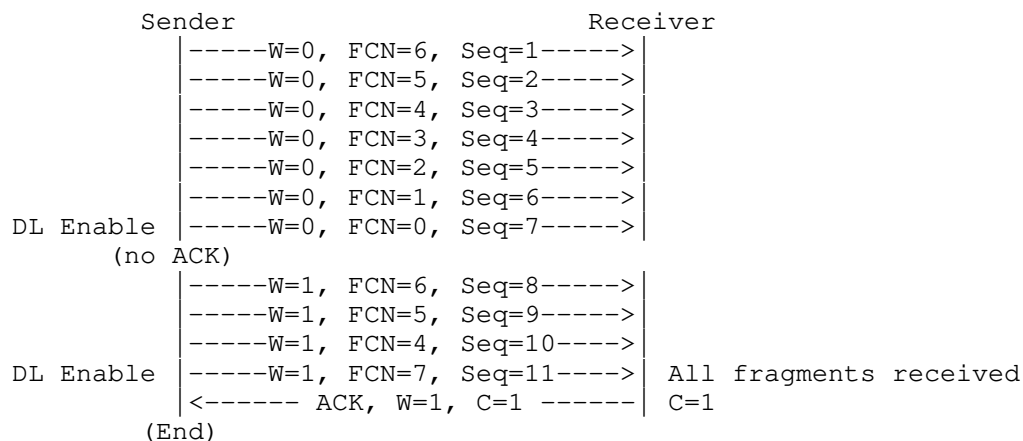


Figure 5: UL ACK-on-Error No-Losses

Case Fragments lost in first window

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

After the missing fragments from the first window (W=0) are resent, the sender without opening a reception window, continues transmitting the following window. Finally, the All-1 fragment is sent, the downlink is enabled and the ACK received with a C=1.

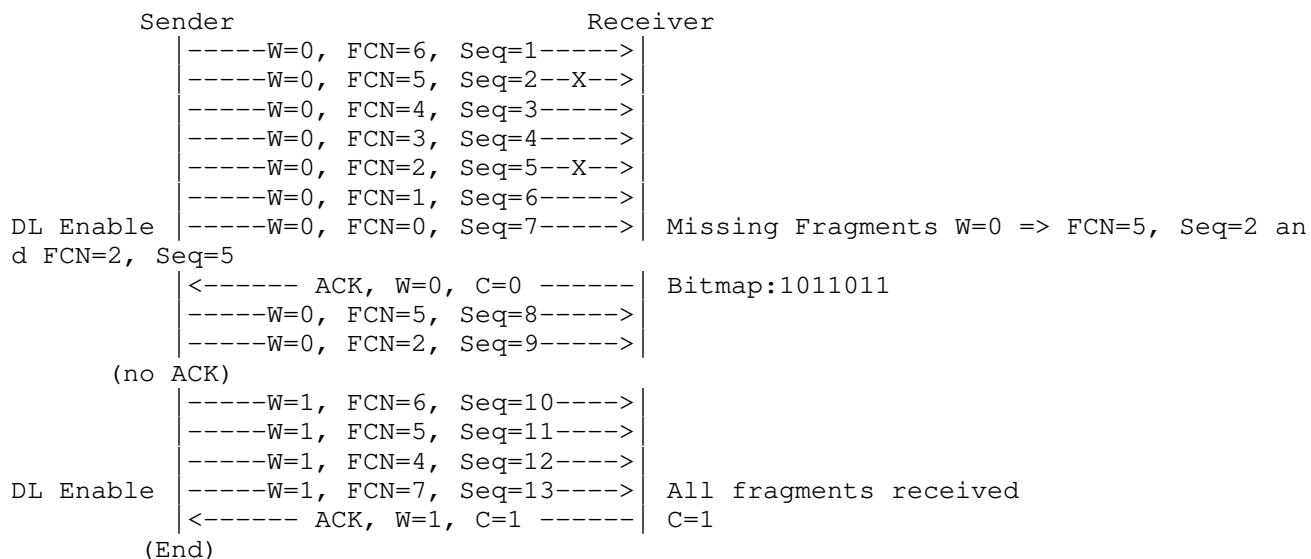


Figure 6: UL ACK-on-Error Losses on First Window

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

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

The sender resends the missing All-0 messages (with any other missing fragment from window 0). Note that this behaviour can take place in any intermediate window if the All-0 message is lost.

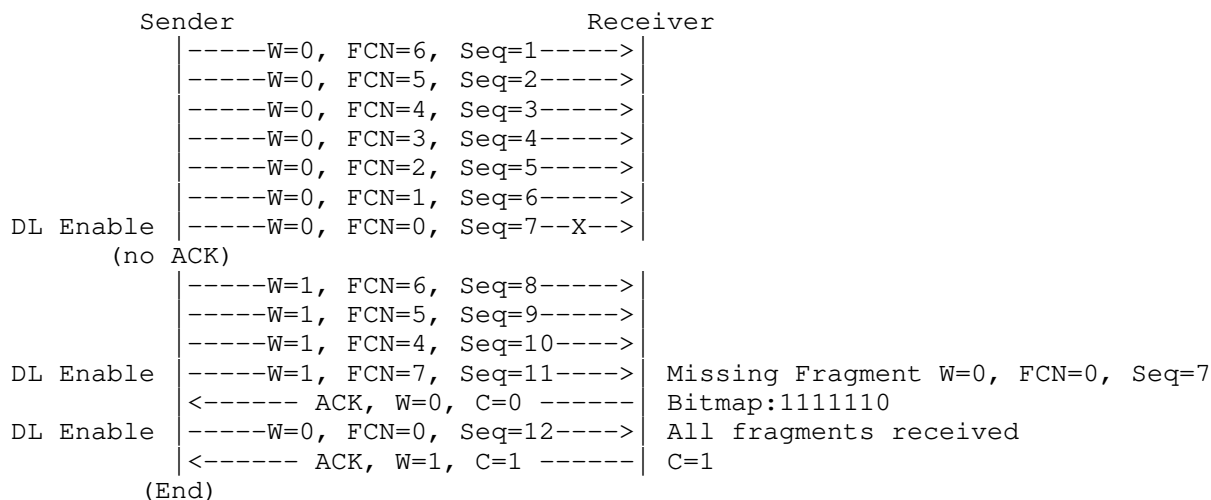


Figure 7: UL ACK-on-Error All-0 Lost on First Window

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

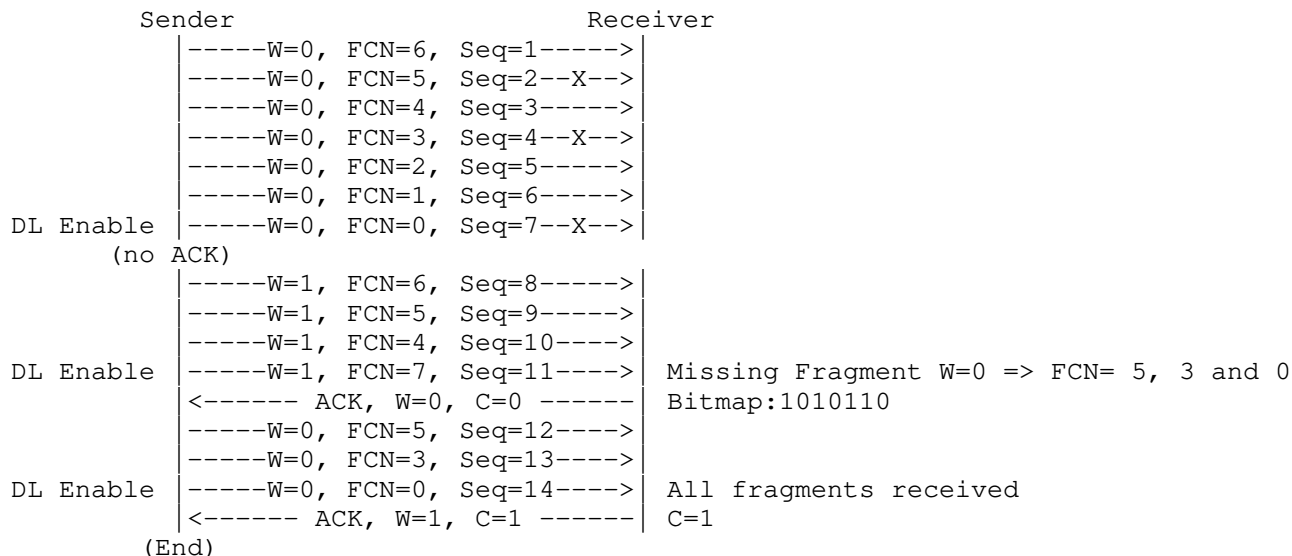


Figure 8: UL ACK-on-Error All-0 and other Fragments Lost on First Window

In the following case, there are losses in both the first (W=0) and second (W=1) window. The retransmission cycles (after the All-1 is

sent, not in intermediate windows) should always finish with an All-0 (if this message was lost) or with an All-1. This is required for the sender to open a reception window so the receiver can send an ACK. Else, there is no way for the Receiver to send an ACK, if All-1 message is lost, then an ACK timeout happen and an ACK is resent.

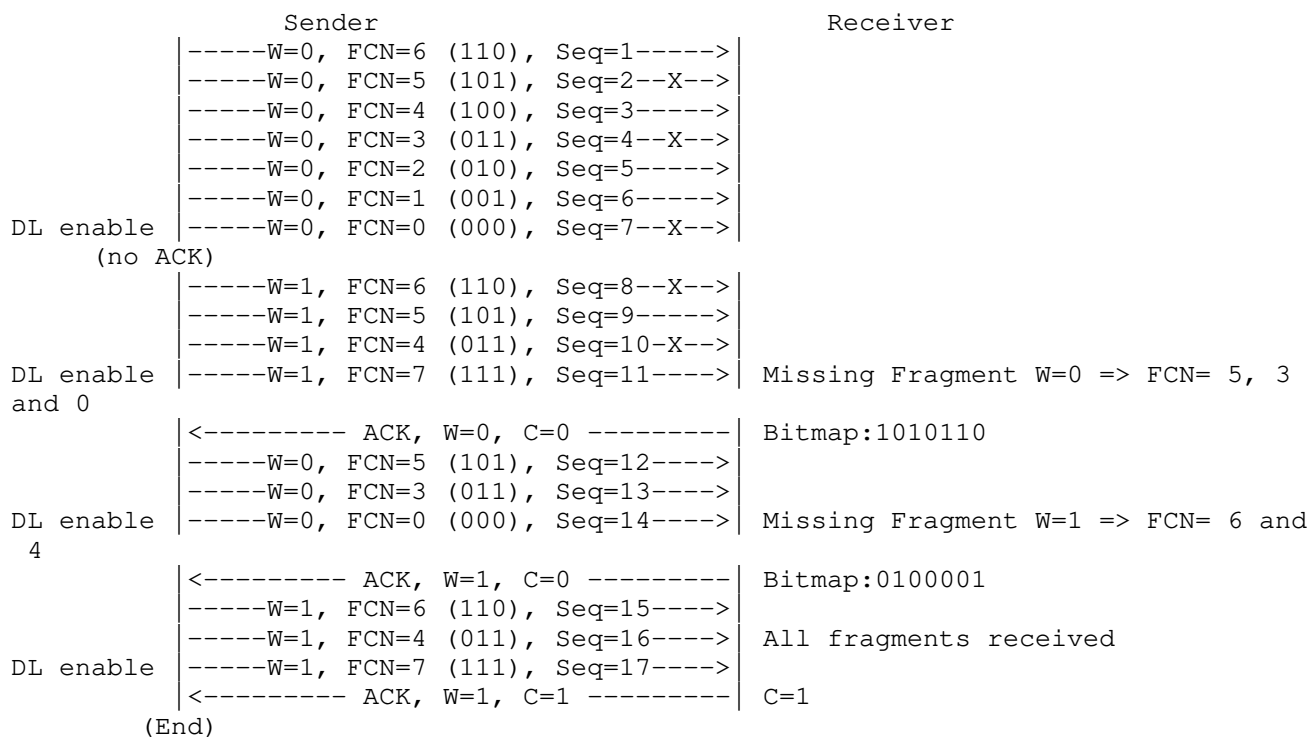


Figure 9: UL ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (1)

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

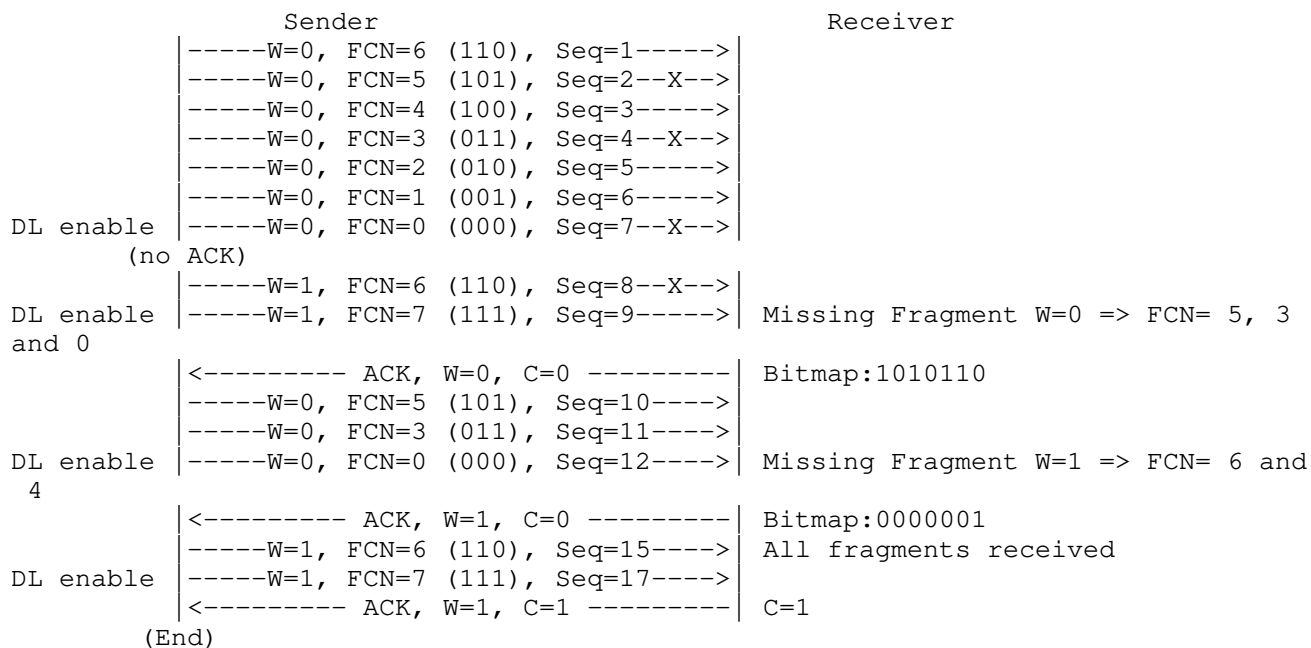


Figure 10: UL ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (2)

Case ACK is lost

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

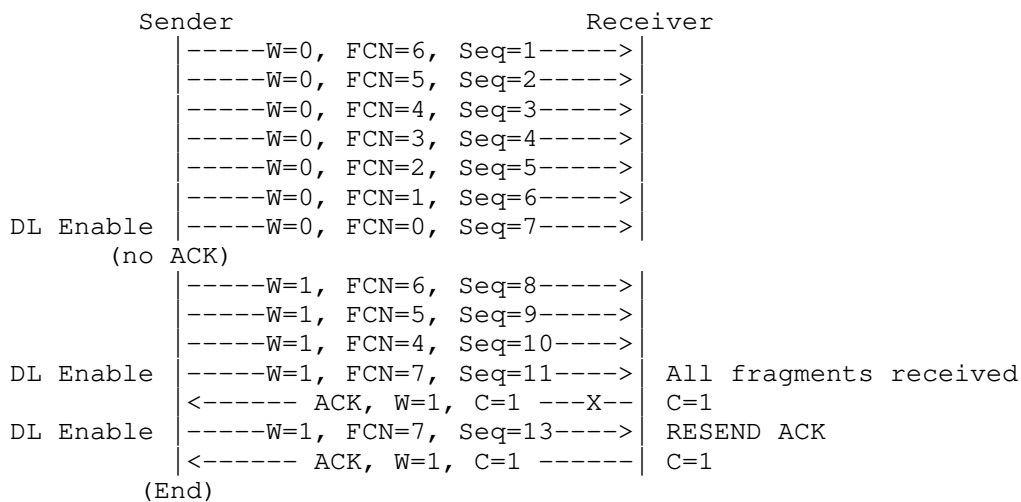


Figure 11: UL ACK-on-Error ACK Lost

The number of times an ACK will be requested is determined by the MAX_ACK_REQUESTS.

5.3. SCHC Abort Examples

Case SCHC Sender-Abort

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

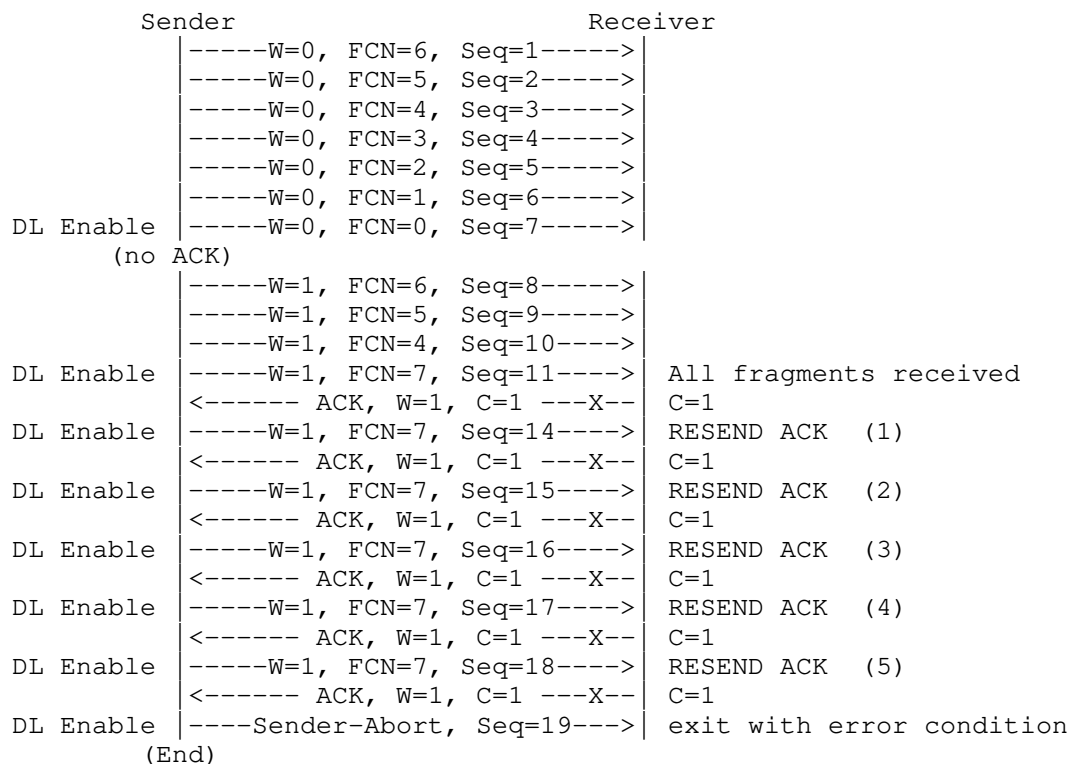


Figure 12: UL ACK-on-Error Sender-Abort

Case Receiver-Abort

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

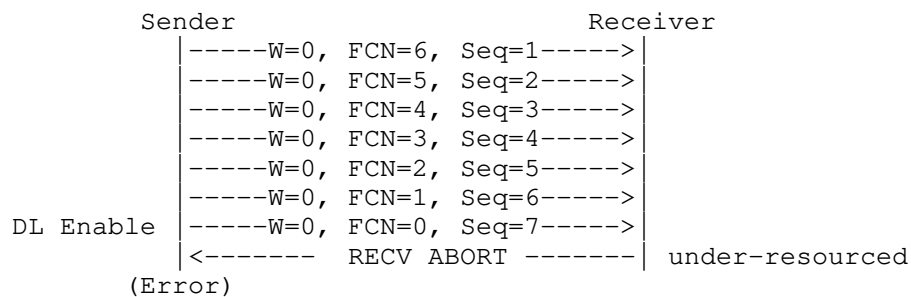


Figure 13: UL ACK-on-Error Receiver-Abort

6. Security considerations

The radio protocol authenticates and ensures the integrity of each message. This is achieved by using a unique device ID and an AES-128 based message authentication code, ensuring that the message has been generated and sent by the device 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.

7. Acknowledgements

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

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

The authors would like to thank Clement Mannequin, Rafael Vidal and Antonis Platis for their useful comments and implementation design considerations.

8. References

8.1. 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>>.

8.2. 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

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

Email: sergio.aguilar.romero@upc.edu

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

Email: Laurent.Toutain@imt-atlantique.fr

Sandra Céspedes
NIC Labs, Universidad de Chile
Av. Almte. Blanco Encalada 1975
Santiago
Chile

Email: scéspedes@niclabs.cl

Diego Wistuba
NIC Labs, Universidad de Chile
Av. Almte. Blanco Encalada 1975
Santiago
Chile

Email: wistuba@niclabs.cl

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 6, 2021

A. Minaburo
Acklio
L. Toutain
Institut MINES TELECOM; IMT Atlantique
February 02, 2021

Data Model for Static Context Header Compression (SCHC)
draft-ietf-lpwan-schc-yang-data-model-04

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 August 6, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	SCHC rules	2
2.1.	Compression Rules	3
2.2.	Field Identifier	3
2.3.	Field length	5
2.4.	Field position	6
2.5.	Direction Indicator	6
2.6.	Target Value	7
2.7.	Matching Operator	8
2.7.1.	Matching Operator arguments	9
2.8.	Compression Decompression Actions	10
2.8.1.	Compression Decompression Action arguments	12
3.	Rule definition	12
3.1.	Compression rule	14
3.1.1.	Compression context representation.	14
3.1.2.	Rule definition	15
3.2.	Fragmentation rule	16
4.	IANA Considerations	24
5.	Security considerations	24
6.	Acknowledgements	24
7.	YANG Module	24
8.	Normative References	41
	Authors' Addresses	42

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 non formal representation of the rules used either for compression/decompression (or C/D) or fragmentation/reassembly (or F/R). The goal of this document is to formalize the description of the rules to offer:

- o the same definition on both ends, even if the internal representation is different.
- o an update the other end to set up some specific values (e.g. IPv6 prefix, Destination address,...)
- o ...

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 protocol. Any header field is abstracted through an ID, a position, a direction and a value that can be a numerical value or a string. [RFC8724] and [I-D.ietf-lpwan-coap-static-context-hc] specifies fields for IPv6, UDP, CoAP and OSCORE. [I-D.barthel-lpwan-oam-schc] describes ICMPv6 header compression.

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 non formal representation of the compression rule. A compression context for a device is composed of a set of rules. Each rule contains information to describe a specific field in the header to be compressed.

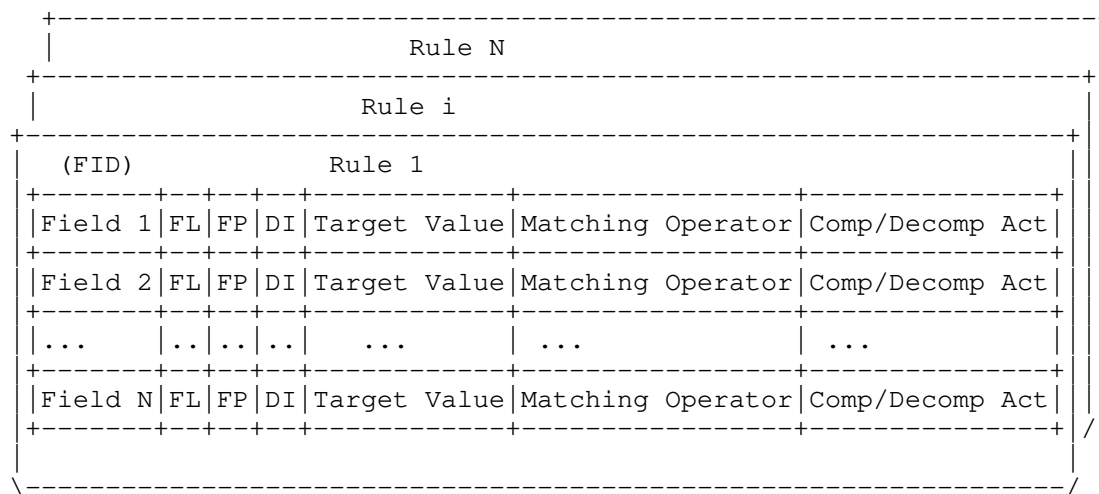


Figure 1: Compression Decompression Context

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 examples, identification is done 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 in 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 some examples 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 bits  
    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 device.";
    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 Compression Decompression Action

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

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

Figure 12: Type definition for Compression 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 identified 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 identifier.
- Field Length : either a positive integer of a function defined as a YANG id.
- Field Position: a positive (and possibly equal to 0) integer.
- Direction Indicator: a YANG identifier giving the direction.
- Target value: a value against which the header Field is compared.
- Matching Operator: a YANG id giving the operation, parameters may be associated to that operator.
- Comp./Decomp. Action: A YANG id giving the compression or decompression action, parameters may be associated to that action.

```
leaf field-id {
  description "Field ID, identify a field in the header with a YANG identifier.";
  mandatory true;
  type schc:field-id-type;
}
leaf field-length {
  description "Field Length in bit or through a function defined as a YANG 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 work for a specific direction, 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 dtag-size {
    type uint8;
    description "size in bit of the DTag field";
  }

  leaf wsize {
    type uint8;
    description "size in bit of the window field";
  }
  leaf fc-size {
    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:

- o maximum-window-size contains the maximum FCN value that can be used.
- o retransmission-timer gives in seconds the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specified, value must be higher or equal to 1.
- o 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.
- o max-ack-requests gives the number of attempts before aborting (cf. section 8.2.2.4. of [RFC8724]).
- o 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"; // Check 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 strictly 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:

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

- o 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, AckOnError)";
        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 fcnsz            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.

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

This module 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-nexthead {
    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 destination
        address prefix of RFC 8200. Depending if it is respectively
        a uplink or a downlink message.";
}

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

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

identity fid-ipv6-appiid {
    base field-id-base-type;
    description "correspond either to the source address or the destination
        address prefix of RFC 768. Depending if it is respectively
        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 bits
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 identifier.
- Field Length : either a positive integer or a function defined as a YANG referenceid.
- 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 referenceid.";
      mandatory true;
      type schc:field-id-type;
    }
    leaf field-length {
      description "Field Length in bit or through a function defined as a YANG 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 packet 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 forbidden.";
        mandatory true;
    }
    leaf dtag-size {
        type uint8;
        description "size in bit of the DTag field";
    }
}

```

```
    }
    leaf wsize {
        type uint8;
        description "size in bit of the window field";
    }
    leaf fcnsz {
        type uint8;
        description "size in bit of the FCN field";
        mandatory true;
    }
    leaf RCS-algorithm {
        type RCS-algorithm-type;
        default schc:RFC8724-RCS;
        description "Algorith 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-02 (work in progress), November 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-18 (work in progress), January 2021.
- [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

LPWAN Working Group
Internet-Draft
Intended status: Informational
Expires: October 31, 2021

A. Pelov
Acklio
P. Thubert
Cisco Systems
A. Minaburo
Acklio
April 29, 2021

LPWAN Static Context Header Compression (SCHC) Architecture
draft-pelov-lpwan-architecture-02

Abstract

This document defines the LPWAN SCHC architecture.

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 October 31, 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. LPWAN Technologies and Profiles	2
3. The Static Context Header Compression	3
4. SCHC Endpoints	3
5. SCHC Instances	4
6. SCHC Data Model	5
7. Security Considerations	7
8. IANA Consideration	7
9. Acknowledgements	7
10. References	7
10.1. Normative References	7
10.2. Informative References	8
10.3. URIs	9
Authors' Addresses	9

1. Introduction

The IETF LPWAN WG defined the necessary operations to enable IPv6 over selected Low-Power Wide Area Networking (LPWAN) radio technologies. [rfc8376] presents an overview of those technologies.

The Static Context Header Compression (SCHC) [rfc8724] technology is the core product of the IETF LPWAN working group. [rfc8724] defines a generic framework for header compression and fragmentation, based on a static context that is pre-installed on the SCHC endpoints.

This document details the constitutive elements of a SCHC-based solution, and how the solution can be deployed. It provides a general architecture for a SCHC deployment, positioning the required specifications, describing the possible deployment types, and indicating models whereby the rules can be distributed and installed to enable reliable and scalable operations.

2. LPWAN Technologies and Profiles

Because LPWAN technologies [rfc8376] have strict yet distinct constraints, e.g., in terms of maximum frame size, throughput, and/or directionality, a SCHC instance must be profiled to adapt to the specific necessities of the technology to which it is applied.

Appendix D. "SCHC Parameters" of [rfc8724] lists the information that an LPWAN technology-specific document must provide to profile SCHC for that technology.

As an example, [rfc9011] provides the SCHC profile for LoRaWAN networks.

3. The Static Context Header Compression

SCHC [rfc8724] specifies an extreme compression capability based on a state that must match on the compressor and decompressor side. This state comprises a set of Compression/Decompression (C/D) rules.

The SCHC Parser analyzes incoming packets and creates a list of fields that it matches against the compression rules. The rule that matches best is used to compress the packet, and the rule identifier (RuleID) is transmitted together with the compression residue to the decompressor. Based on the RuleID and the residue, the decompressor can rebuild the original packet and forward it in its uncompressed form over the Internet.

[rfc8724] also provides a Fragmentation/Reassembly (F/R) capability to cope with the maximum frame size of a Link, which is extremely constrained in the case of an LPWAN network.

If a SCHC-compressed packet is too large to be sent in a single Link-Layer PDU, the SCHC fragmentation can be applied on the compressed packet. The process of SCHC fragmentation is similar to that of compression; the fragmentation rules that are programmed for this device are checked to find the most appropriate one, regarding the SCHC packet size, the link error rate, and the reliability level required by the application.

The nature of a ruleID allows to determine if it is a compression or fragmentation rule.

4. SCHC Endpoints

Section 3 of [rfc8724] depicts a typical network architecture for an LPWAN network, simplified from that shown in [rfc8376] and reproduced in Figure 1.

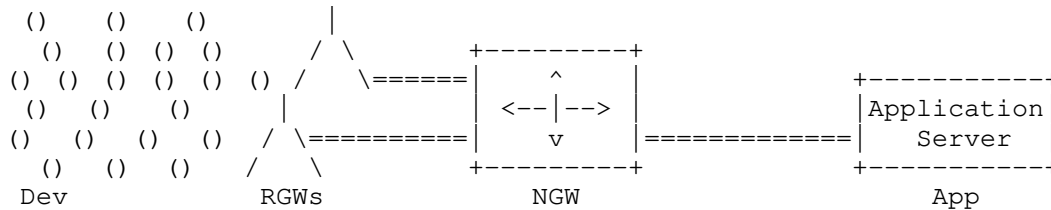


Figure 1: Typical LPWAN Network Architecture

Typically, an LPWAN network topology is star-oriented, which means that all packets between the same source-destination pair follow the same path from/to a central point. In that model, highly constrained

Devices (Dev) exchange information with LPWAN Application Servers (Apps) through a central Network Gateway (NGW), which can be powered and is typically a lot less constrained than the Devices. Because devices embed built-in applications, the traffic flows to be compressed are known in advance and the location of the C/D and F/R functions (e.g., at the Dev and NGW), and the associated rules, can be pre provisioned in the network .

Then again, SCHC is very generic and its applicability is not limited to star-oriented deployments and/or to use cases where applications are very static and the state can be provisioned in advance. [I-D.thubert-intarea-schc-over-ppp] describes an alternate deployment where the C/D and/or F/R operations are performed between peers of equal capabilities over a PPP [rfc2516] connection. SCHC over PPP illustrates that with SCHC, the protocols that are compressed can be discovered dynamically and the rules can be fetched on-demand by both parties from the same Uniform Resource Name (URN) [rfc8141], ensuring that the peers use the exact same set of rules.

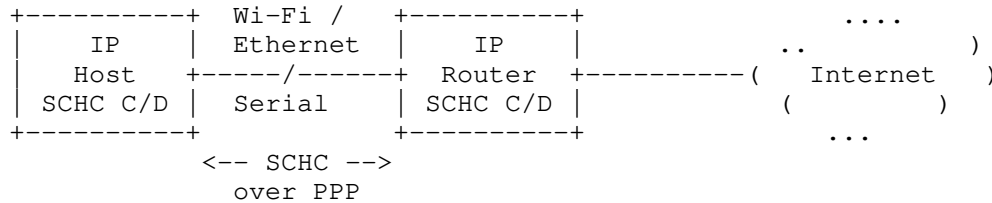


Figure 2: PPP-based SCHC Deployment

5. SCHC Instances

The rule database contains a set of rules that are specific per device. There is thus a SCHC instance per pair of endpoints. [rfc8724] states that a SCHC instance obtains the rules to process C/D and F/R before the session starts, and that rules cannot be modified during the session.

[rfc8724] was defined to compress IPv6 [rfc8200] and UDP; but SCHC really is a generic compression and fragmentation technology. As such, SCHC is agnostic to which protocol it compresses and at which layer it is operated. The C/D peers may be hosted by different entities for different layers, and the F/R operation may also be performed between different parties, or different sub-layers in the same stack, and/or managed by different organizations.

If a protocol or a layer requires additional capabilities, it is always possible to document more specifically how to use SCHC in that context, or to specify additional behaviours. For instance,

[I-D.ietf-lpwan-coap-static-context-hc] extends the compression to CoAP [RFC7252] and OSCORE [RFC8613].

As represented figure Figure 3, the fragmentation and the compression of the IP and UDP headers may be operated by a network SCHC instance whereas the end-to-end compression of the application payload happens between the device and the application. The compression of the application payload may be split in two instances to deal with the encrypted portion of the application PDU.

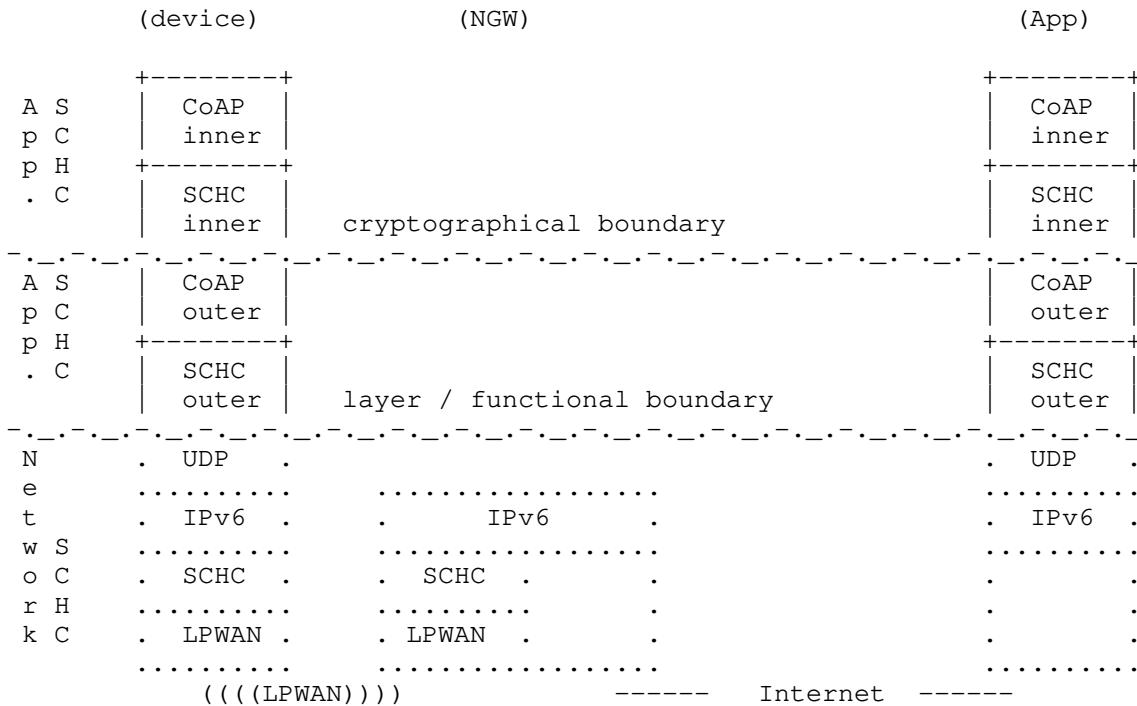


Figure 3: Different SCHC instances in a global system

This document defines a generic architecture for SCHC that can be used at any of these levels. The goal of the architectural document is to orchestrate the different protocols and data model defined by the LPWAN working group to design an operational and interoperable framework for allowing IP application over constrained networks.

6. SCHC Data Model

A SCHC instance, summarized in the Figure 4, implies C/D and/or F/R present in both end and that both ends are provisionned with the same set of rules.



Figure 4: Summarized SCHC elements

To be able to provision end-points from different vendors, a common rule representation is needed that expresses the SCHC rules in an interoperable fashion. To that effect, [I-D.ietf-lpwan-schc-yang-data-model] defines a rule representation using the YANG [1] formalism.

[I-D.ietf-lpwan-schc-yang-data-model] defines an YANG data model to represent the rules. This enables the use of several protocols for rule management, such as NETCONF[RFC6241], RESTCONF[RFC8040], and CORECONF[I-D.ietf-core-comi]. NETCONF uses SSH, RESTCONF uses HTTPS, and CORECONF uses CoAP(s) as their respective transport layer protocols. The data is represented in XML under NETCONF, in JSON[RFC8259] under RESTCONF and in CBOR[RFC8949] under CORECONF.

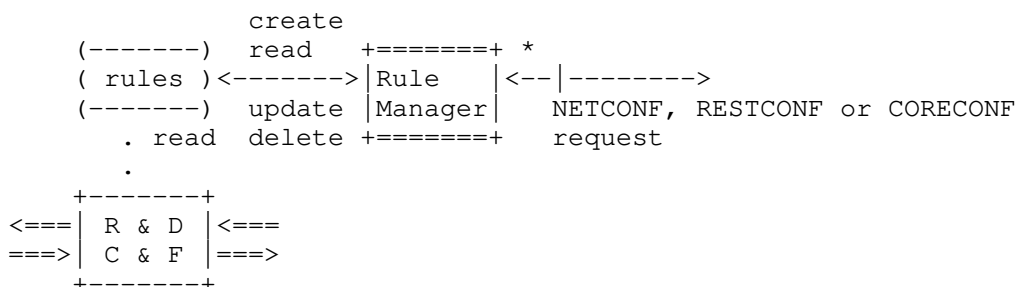


Figure 5: Summarized SCHC elements

The Rule Manager (RM) is in charge of handling data derived from the YANG Data Model and apply changes to the rules database Figure 5.

The RM is a application using the Internet to exchange information, therefore:

- o for the network-level SCHC, the communication does not require routing. Each of the end-points having an RM and both RMs can be

viewed on the same link, therefore wellknown Link Local addresses can be used to identify the device and the core RM. L2 security MAY be deemed as sufficient, if it provides the necessary level of protection.

- o for application-level SCHC, routing is involved and global IP addresses SHOULD be used. End-to-end encryption is RECOMMENDED.

Management messages can also be carried in the negotiation protocol as proposed in [I-D.thubert-intarea-schc-over-ppp]. The RM traffic may be itself compressed by SCHC, especially if CORECONF is used, [I-D.ietf-lpwan-coap-static-context-hc] can be used.

7. Security Considerations

SCHC is sensitive to the rules that could be abused to form arbitrary long messages or as a form of attack against the C/D and/or F/R functions, say to generate a buffer overflow and either modify the device or crash it. It is thus critical to ensure that the rules are distributed in a fashion that is protected against tempering, e.g., encrypted and signed.

8. IANA Consideration

This document has no request to IANA

9. Acknowledgements

The authors would like to thank (in alphabetic order):

10. References

10.1. Normative References

- [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>>.
- [rfc9011] Gimenez, O., Ed. and I. Petrov, Ed., "Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN", RFC 9011, DOI 10.17487/RFC9011, April 2021, <<https://www.rfc-editor.org/info/rfc9011>>.

10.2. Informative References

- [I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", draft-ietf-core-comi-11 (work in progress), January 2021.
- [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-19 (work in progress), March 2021.
- [I-D.ietf-lpwan-schc-yang-data-model]
Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", draft-ietf-lpwan-schc-yang-data-model-04 (work in progress), February 2021.
- [I-D.thubert-intarea-schc-over-ppp]
Thubert, P., "SCHC over PPP", draft-thubert-intarea-schc-over-ppp-03 (work in progress), April 2021.
- [rfc2516] Mamakos, L., Lidl, K., Evarts, J., Carrel, D., Simone, D., and R. Wheeler, "A Method for Transmitting PPP Over Ethernet (PPPoE)", RFC 2516, DOI 10.17487/RFC2516, February 1999, <<https://www.rfc-editor.org/info/rfc2516>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [rfc8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <<https://www.rfc-editor.org/info/rfc8141>>.

- [rfc8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <<https://www.rfc-editor.org/info/rfc8200>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [rfc8376] Farrell, S., Ed., "Low-Power Wide Area Network (LPWAN) Overview", RFC 8376, DOI 10.17487/RFC8376, May 2018, <<https://www.rfc-editor.org/info/rfc8376>>.
- [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>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

10.3. URIs

- [1] RFC7950

Authors' Addresses

Alexander Pelov
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France

Email: a@ackl.io

Pascal Thubert
Cisco Systems
45 Allee des Ormes - BP1200
06254 Mougins - Sophia Antipolis
France

Email: pthubert@cisco.com

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

Email: ana@ackl.io

LPWAN
Internet-Draft
Updates: 5172 (if approved)
Intended status: Standards Track
Expires: 23 October 2021

P. Thubert, Ed.
Cisco Systems
21 April 2021

SCHC over PPP
draft-thubert-intarea-schc-over-ppp-03

Abstract

This document extends RFC 5172 to signal the use of SCHC as the compression method between a pair of nodes over PPP. Combined with RFC 2516, this enables the use of SCHC over Ethernet and Wi-Fi.

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 23 October 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. BCP 14	3
3. Extending RFC 5172	3
4. Profiling SCHC for high speed links	4
4.1. Mapping the SCHC Architecture	4
4.2. SCHC Parameters	5
4.2.1. Resulting Packet Format	6
4.3. Security Considerations	8
5. IANA Considerations	8
6. Acknowledgments	9
7. Normative References	9
8. Informative References	9
Author's Address	10

1. Introduction

The Point-to-Point Protocol (PPP) [RFC5172] provides a standard method of encapsulating network-layer protocol information over serial (point-to-point and bus) links. "A Method for Transmitting PPP Over Ethernet (PPPoE)" [RFC2516] transports PPP over Ethernet between a pair of nodes. It is compatible with a translating bridge to Wi-Fi, and therefore enables PPP over Wi-Fi as well.

PPP also proposes an extensible Link Control Protocol and a family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols. "IP Version 6 over PPP" [RFC5072] specifies the IPv6 Control Protocol (IPV6CP), which is an NCP for a PPP link, and allows for the negotiation of desirable parameters for an IPv6 interface over PPP. "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol" [RFC5172] defines the IPv6 datagram compression option that can be negotiated by a node on the link through the IPV6CP.

PPP is not commonly used in Low-Power Wide Area Networks (LPWAN) but the extreme compression techniques that are defined for use in LPWAN may be applicable to more traditional links where PPP applies.

The "Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6" [SCHC] is a new technology that can provide an extreme compression performance but requires a same state to be provisioned on both ends before it can be operated.

The "SCHC Architecture" [I-D.pelov-lpwan-architecture] enables a peer to peer SCHC operation in addition to the classical device to network LPWAN paradigm, e.g., over a PPP connection. To enable SCHC over PPP and therefore Ethernet and Wi-Fi, this specification extends [RFC5172] to signal SCHC as an additional compression method for use over PPP.

An example use case for SCHC over PPP over Ethernet (SCHCoPPPoE) is to apply SCHC to periodic flows and maintain them at a protocol-independent size and rate. The constant size may be too small for a particular flow or protocol. The SCHC fragmentation can then be used to transport a protocol data unit (PDU) as N compressed SCHC fragments, in which case the effective PDU rate is the TSN frame rate divided by N.

This can be useful to streamline the frames and simplifies the scheduling of Deterministic Networking [DetNet] and Operational Technology (OT) control flows over IEEE Std 802.1 Time-Sensitive Networking (TSN) [IEEE802.1TSNTG] or one of the RAW Technologies [RAW Technologies].

2. BCP 14

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

3. Extending RFC 5172

With this specification, a PPP session defines a virtual link where a SCHC context is established with a particular set of Rules, which is indicated at the set up of the PPP session as follows:

[RFC5172] defines an IPV6CP option called the IPv6-Compression-Protocol Configuration option with a type of 2. The option contains an IPv6-Compression-Protocol field value that indicates a compression protocol and an optional data field as shown in Figure 1:

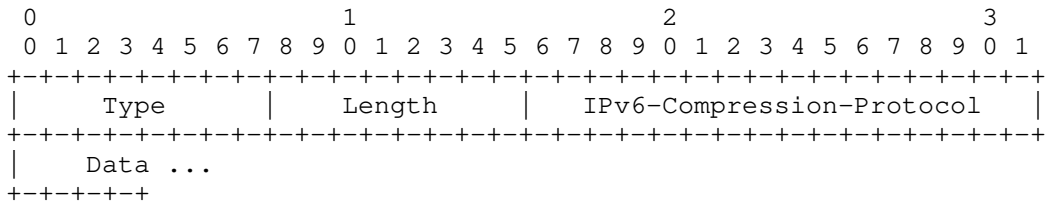


Figure 1: The IPv6-Compression-Protocol Configuration Option

This specification indicates a new IPv6-Compression-Protocol field value for [SCHC] (see Section 5), and enables to transport a Uniform Resource Identifier (URI) [RFC3986] of the set of rules in the optional data. The default format for the set of rules is YANG using the "Data Model for SCHC" [SCHC_DATA_MODEL] encoded in JSON as specified in [RFC7951]. The size of the URL is computed based on the Length of the option as Length-4. If the encoding is asymmetrical, the initiator of the session is considered downstream, playing the role of the device in an LPWAN network.

4. Profiling SCHC for high speed links

Appendix D of [SCHC] specifies the profile information that technology specifications such as this must provide. The following section address this requirement.

4.1. Mapping the SCHC Architecture

This specification leverages SCHC between an end point that is an IP Host and possibly a serial DTE (Data Terminal Equipment), and another that is an IP Node (either another IP Host or a Router) and possibly a serial DCE (Data Control Equipment), or a more modern physical or emulated endpoint, e.g., Ethernet devices that exchange IP packets over PPPoE.

Both endpoints MUST support the function of SCHC Compressor/Decompressor (C/D) as shown in Figure 2.

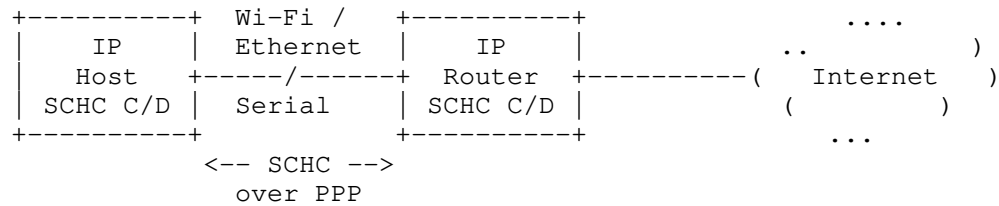


Figure 2: Typical Deployment

The SCHC Fragmenter/Reassembler (F/R) is generally not needed, because the maximum transmission unit (MTU) is expected to be large enough and SCHC only reduces the frame size vs. native IP. But it may be used to obtain a small protocol-independant frame size for the compressed packets, possibly way smaller than MTU.

A context may be generated for a particular upper layer application, such as a control loop using an industrial automation protocol, to protect the particular flow with a DetNet service. The context can be asymmetric, e.g., when connecting a primary and a secondary endpoints, a client and a server, or a programmable logic controller with a sensor or an actuator.

4.2. SCHC Parameters

Compared to typical LPWANs, most serial links and emulations such as PPPoE are very fast and most of the constraints can be alleviated. For this reason, the SCHC profile for PPP is defined as follows:

RuleID numbering scheme: The RuleID for a compression rule is expressed as 2 bytes. The first (leftmost) 2 bits of that RuleID MUST be set to 0 This leaves 14 bits to index the rule. A SCHC compressed packet is always in the form:

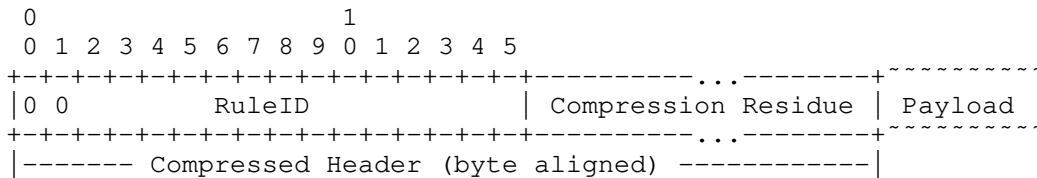


Figure 3: SCHC Compressed Packet

This specification only supports the No-ACK Mode of SCHC fragmentation as specified in section 8.4.1 of [SCHC]. The SCHC Fragment Header is 2 bytes long.

The RuleID for a fragmentation rule is expressed as 4 bits. The bits MUST all set to 1 for a fragmentation rule in No-ACK Mode. The DTag field is 11 bits long (T=11) and the FCN field is one bit (N=1), which is set to 1 on the last fragment as illustrated in Appendix B of [SCHC] and to 0 otherwise. There is no W field (M=0).

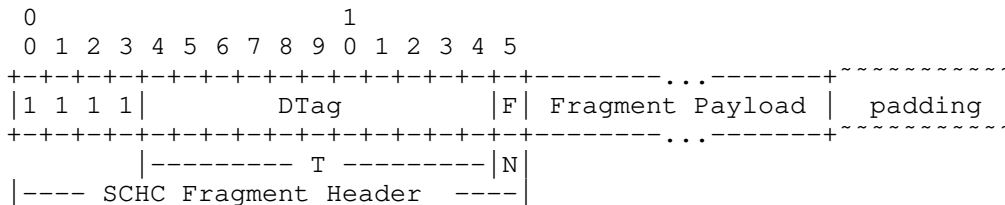


Figure 4: SCHC Fragment

The No-ACK mode has been designed under the assumption that data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly and a DetNet PREOF function might be needed to reorder the fragments.

Maximum packet size: `MAX_PACKET_SIZE` is aligned to the PPP Link MTU.

Padding: The Compression Residue MUST be aligned to the L2 word. For Ethernet, the L2 word is one byte, so padding is needed up to the next byte boundary. If a compression rule produces a residue that is not byte aligned, then it is implicitly terminated with a statement that indicates padding till the next byte boundary. The padding bit is 0.

4.2.1. Resulting Packet Format

In the case of PPPoE, the sequence of compression and encapsulation is as follows:

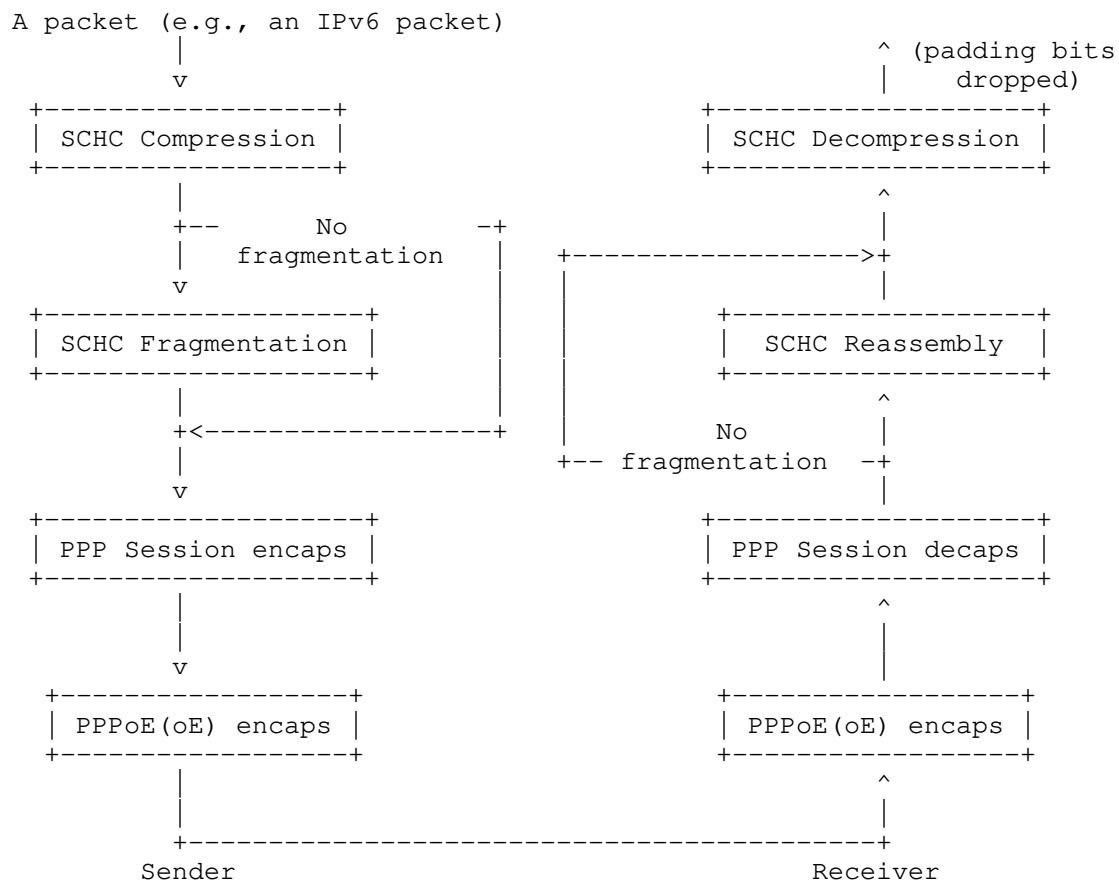


Figure 5: Stack Operation (no fragment)

In the case of PPPoE, a frame that transports an IPv6 packet compressed with SCHC with no fragmentation shows as follows:

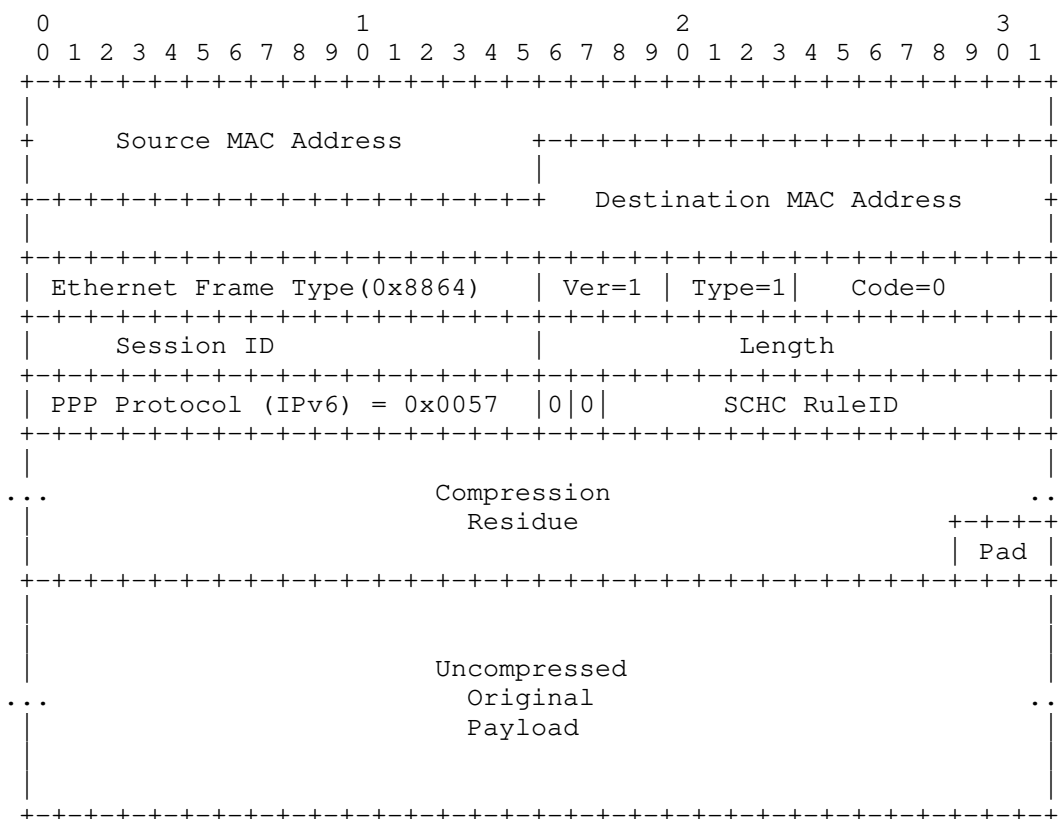


Figure 6: SCHC over PPP over Ethernet Format

4.3. Security Considerations

This draft enables to use the SCHC compression and fragmentation over PPP and therefore Ethernet and Wi-Fi with PPPoE. It inherits the possible threats against SCHC listed in the "Security considerations" section of [SCHC].

5. IANA Considerations

This document requests the allocation of a new value in the registry "IPv6-Compression-Protocol Types" for "SCHC". A suggested value is proposed in Table 1:

Value	Description	Reference
4	Static Context Header Compression (SCHC)	This document

Table 1: IP Header Compression Configuration Option Suboption Types

6. Acknowledgments

7. 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>>.
- [RFC2516] Mamakos, L., Lidl, K., Evarts, J., Carrel, D., Simone, D., and R. Wheeler, "A Method for Transmitting PPP Over Ethernet (PPPoE)", RFC 2516, DOI 10.17487/RFC2516, February 1999, <<https://www.rfc-editor.org/info/rfc2516>>.
- [RFC5072] Varada, S., Ed., Haskins, D., and E. Allen, "IP Version 6 over PPP", RFC 5072, DOI 10.17487/RFC5072, September 2007, <<https://www.rfc-editor.org/info/rfc5072>>.
- [RFC5172] Varada, S., Ed., "Negotiation for IPv6 Datagram Compression Using IPv6 Control Protocol", RFC 5172, DOI 10.17487/RFC5172, March 2008, <<https://www.rfc-editor.org/info/rfc5172>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [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>>.
- [SCHC] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zúñiga, "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>>.

8. Informative References

[I-D.pelov-lpwan-architecture]

Pelov, A., Thubert, P., and A. Minaburo, "Static Context Header Compression (SCHC) Architecture", Work in Progress, Internet-Draft, draft-pelov-lpwan-architecture-00, 19 January 2021, <<https://tools.ietf.org/html/draft-pelov-lpwan-architecture-00>>.

[SCHC_DATA_MODEL]

Minaburo, A. and L. Toutain, "Data Model for Static Context Header Compression (SCHC)", Work in Progress, Internet-Draft, draft-ietf-lpwan-schc-yang-data-model-03, 10 July 2020, <<https://tools.ietf.org/html/draft-ietf-lpwan-schc-yang-data-model-03>>.

[RAW Technologies]

Thubert, P., Cavalcanti, D., Vilajosana, X., Schmitt, C., and J. Farkas, "Reliable and Available Wireless Technologies", Work in Progress, Internet-Draft, draft-thubert-raw-technologies-05, 18 May 2020, <<https://tools.ietf.org/html/draft-thubert-raw-technologies-05>>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <<https://www.rfc-editor.org/info/rfc7951>>.

[DetNet] Finn, N., Thubert, P., Varga, B., and J. Farkas, "Deterministic Networking Architecture", RFC 8655, DOI 10.17487/RFC8655, October 2019, <<https://www.rfc-editor.org/info/rfc8655>>.

[IEEE802.1TSNIG]

IEEE, "Time-Sensitive Networking (TSN) Task Group", <<https://1.ieee802.org/tsn/>>.

Author's Address

Pascal Thubert (editor)
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
06254 Mougins - Sophia Antipolis
France

Phone: +33 497 23 26 34
Email: pthubert@cisco.com