lpwan Working Group
Internet-Draft
Intended status: Informational
Expires: 29 December 2023

D. Barthel
Orange SA
L. Toutain
IMT Atlantique
A. Kandasamy
Acklio
D. Dujovne
Universidad Diego Portales
JC. Zuniga
Cisco

27 June 2023

OAM for LPWAN using Static Context Header Compression (SCHC)
draft-barthel-lpwan-oam-schc-05

Abstract

This document describes ICMPv6 compression with SCHC and how basic OAM is performed on Low Power Wide Area Networks (LPWANs) by compressing ICMPv6/IPv6 headers and by protecting the LPWAN network and the Device from undesirable ICMPv6 traffic.

With IP protocols now generalizing to constrained networks, users expect to be able to Operate, Administer and Maintain them with the familiar tools and protocols they already use on less constrained networks.

OAM uses specific messages sent into the data plane to measure some parameters of a network. Most of the time, no explicit values are sent is these messages. Network parameters are obtained from the analysis of these specific messages.

This can be used:

* To detect if a host is up or down.

* To measure the RTT and its variation over time.

* To learn the path used by packets to reach a destination.

OAM in LPWAN is a little bit trickier since the bandwidth is limited and extra traffic added by OAM can introduce perturbation on regular transmission.

Three main scenarios are investigated:

* OAM reachability messages coming from internet. In that case, the SCHC core should act as a proxy and handle specifically the OAM traffic.

* OAM messages initiated by LPWAN devices: They can be anticipated by the core SCHC.

* OAM error messages coming from internet. In that case, the SCHC core may forward a compressed version to the device.

The primitive functionalities of OAM are achieved with the ICMPv6 protocol.

ICMPv6 defines messages that inform the source of IPv6 packets of errors during packet delivery. It also defines the Echo Request/ Reply messages that are used for basic network troubleshooting (ping command). ICMPv6 messages are transported on IPv6.

This document also introduces the notion of actions in a SCHC rule, to perform locally some operations.

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 29 December 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

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

extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Use cases 4
4. Detailed behavior 5
4.1. Device does a ping 5
4.1.1. Rule example 6
4.2. Device is ping'ed 7
4.2.1. Rule example 7
4.3. Device is the source of an ICMPv6 error message 8
4.4. Device is the destination of an ICMPv6 error message . . 9
4.4.1. ICMPv6 error message compression. 10
5. YANG identities and tree 13
6. YANG Module 14
7. Security considerations 18
8. IANA Considerations 18
9. References 18
9.1. Normative References 18
9.2. Informative References 19
Authors' Addresses 20

1. Introduction

The primitive functionalities of OAM [RFC6291] are achieved with the ICMPv6 protocol.

ICMPv6 [RFC4443] is a companion protocol to IPv6 [RFC8200].

[RFC4443] defines a generic message format. This format is used for messages to be sent back to the source of an IPv6 packet to inform it about errors during packet delivery.

More specifically, [RFC4443] defines 4 error messages: Destination Unreachable, Packet Too Big, Time Exceeded and Parameter Problem.

[RFC4443] also defines the Echo Request and Echo Reply messages, which provide support for the ping application.

Other ICMPv6 messages are defined in other RFCs, such as an extended format of the same messages [RFC4884] and other messages used by the Neighbor Discovery Protocol [RFC4861].

This document focuses on using Static Context Header Compression (SCHC) to compress [RFC4443] messages that need to be transmitted over the LPWAN network, and on having the LPWAN gateway proxying the Device to save it the unwanted traffic.

LPWANs salient characteristics are described in [RFC8376].

2. Terminology

This draft re-uses the Terminology defined in [RFC8724].

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. Use cases

In the LPWAN architecture, we can distinguish the following cases:

* the Device is the originator of an Echo Request message, and therefore the destination of the Echo Reply message. This message is compressed by the device through SCHC rules specifying ICMPv6 fields.

* the Device is the destination of an Echo Request message, and therefore the purported source of an Echo Reply message. The core SCHC can either send a compressed SCHC message, or proxy the answer to avoid sending data on the constrained link. The proxy answer can be related to the device activity.

* the Device is the (purported) source of an ICMP error message, mainly in response to an incorrect incoming IPv6 message, or in response to a ping request. In this case, as much as possible, the core SCHC C/D should act as a proxy and originate the ICMP Destination Unreachable message, so that the Device and the LPWAN network are protected from this unwanted traffic.

* the Device is the destination of the ICMP message, mainly in response to a packet sent by the Device to the network that generates an error. In this case, we want the ICMP message to reach the Device, and this document describes in Section 4.4.1 what SCHC compression should be applied.

These cases are further described in Section 4.

4. Detailed behavior

4.1. Device does a ping

A Device may send some Echo Request message to check the availability of the network or the host running the Application.

If a ping request is generated by a Device, then SCHC compression applies.

The format of an ICMPv6 Echo Request message is described in Figure 1, with Type=128 and Code=0.

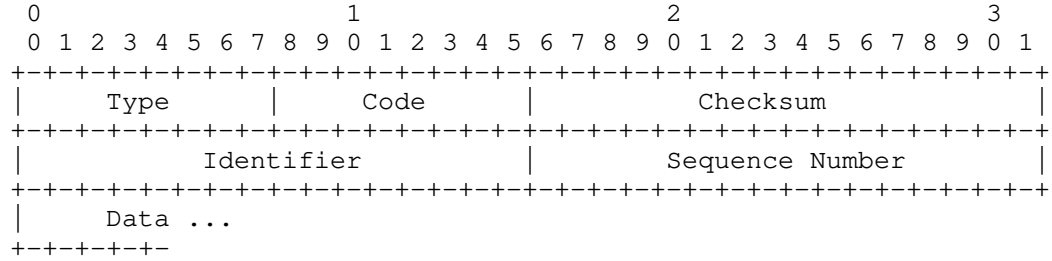


Figure 1: ICMPv6 Echo Request message format

If we assume that one rule will be devoted to compressing Echo Request messages, then Type and Code are known in the rule to be 128 and 0 and can therefore be elided with the not-sent CDA.

Checksum can be reconstructed with the compute-checksum CDA and therefore is not transmitted.

[RFC4443] states that Identifier and Sequence Number are meant to aid in matching Echo Replies to this Echo Request and that they may be zero. Data is zero or more bytes of arbitrary data.

For constrained devices or networks, we recommend that Identifier be zero, Sequence Number be a counter on 3 bits, and Data be zero bytes (absent). Therefore, Identifier is elided with the not-sent CDA, Sequence Number is transmitted on 3 bits with the LSB CDA and no Data is transmitted.

The transmission cost of the Echo Request message is therefore the size of the Rule Id + 3 bits. The rule ID length can be chosen to avoid adding padding.

When the destination receives the Echo Request message, it will respond back with a Echo Reply message. This message bears the same format as the Echo Request message but with Type = 129 (see Figure 1).

[RFC4443] states that the Identifier, Sequence Number and Data fields of the Echo Reply message shall contain the same values as the invoking Echo Request message. Therefore, a rule shall be used similar to that used for compressing the Echo Request message.

4.1.1. Rule example

The following rule gives an example of a SCHC compression. The type can be elided if the direction is taken into account. Identifier is ignored and generated as 0 at decompression. This implies that only one single ping can be launched at any given time on a device. Finally, only the least significant 8 bits of the sequence number are sent on the LPWAN, allowing a serie of 255 consecutive pings.

Field	FL	FP	DI	Value	Matching Operator	CDA	Sent bits
IPv6 Headers description							
ICMPv6 Type	8	1	Up	128	equal	not-sent	
ICMPv6 Type	8	1	Dw	129	equal	not-sent	
ICMPv6 Code	8	1	Bi	0	equal	not-sent	
ICMPv6 Identifier	16	1	Bi	0	ignore	not-sent	
ICMPv6 Sequence	16	1	Bi	0	MSB(24)	LSB	8

Table 1: Example of compression rule for a ping from the device

4.2. Device is ping'ed

If the Device is pinged (i.e., is the destination of an Echo Request message), the device receives the compress message and generate an Echo. In that case, the fields sequence number and identifier cannot be compressed if the source is not aware of the compression scheme.

But the default behavior is to avoid propagating the Echo Request message over the LPWAN.

This is done by proxying the ping request on the core SCHC C/D. This requires to introduce a new processing when the rule is selected. The selection of a compression rule triggers the compression and sends the SCHC packet to the other end. Specifying an Action, change this behavior. In our case, being processed by the compressor, the packet description is processed by a ping proxy. Since the rule is used for the selection, so CDAs are not necessary and set to "not-sent".

The ping-proxy takes a parameter in second, gives the interval during which the device is considered active. During this interval, the proxy-ping echoes ping requests, after this duration, the ping request will be discarded.

The resulting behavior is shown on Figure 2 and described below:

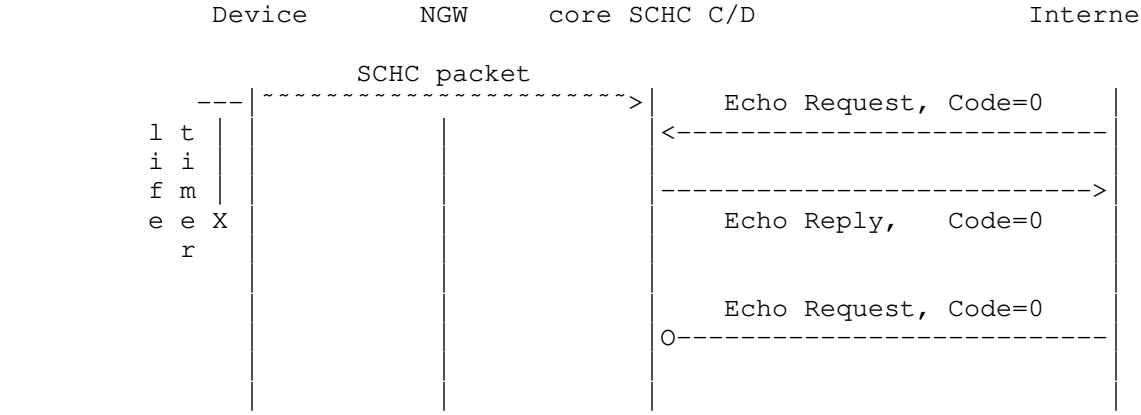


Figure 2: Examples of ICMPv6 Echo Request/Reply

4.2.1. Rule example

The following rule shows an example of a compression rule for pinging a device.

Field	FL	FP	DI	Value	Matching Operator	CDA	Sent bits
Action: proxy-ping(300)							
IPv6 Headers description							
ICMPv6 Type	8	1	Dw	128	equal	not-sent	
ICMPv6 Code	8	1	Bi	0	equal	not-sent	
ICMPv6 Identifier	16	1	Bi	0	ignore	not-sent	
ICMPv6 Sequence	16	1	Bi	0	MSB(24)	LSB	8

Table 2: Example of compression rule for a ping to a device

In this example, type and code are elided, the identifer has to be sent, and the sequence number is limited to one byte.

4.3. Device is the source of an ICMPv6 error message

As stated in [RFC4443], a node should generate an ICMPv6 message in response to an IPv6 packet that is malformed or which cannot be processed due to some incorrect field value.

The general intent of this document is to spare both the Device and the LPWAN network this un-necessary traffic. The incorrect packets should be caught at the core SCHC C/D and the ICMPv6 notification should be sent back from there.

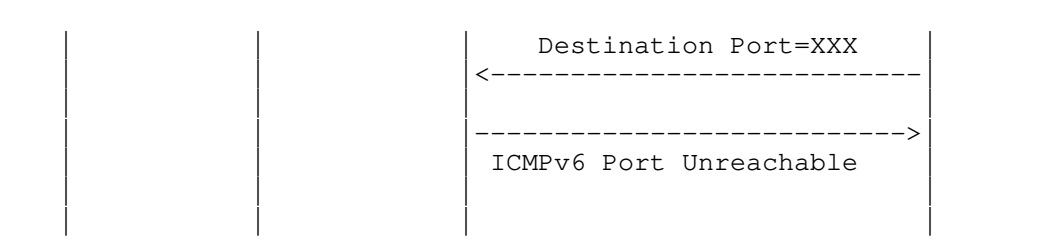


Figure 3: Example of ICMPv6 error message sent back to the Internet

Figure 3 shows an example of an IPv6 packet trying to reach a Device.

Let's assume that no rule matches the incoming packet (i.e. there is no co-compression rule)

Instead of sending the packet over the LPWAN and having this packet rejected by the Device, the core SCHC C/D issues an ICMPv6 error message Destination Unreachable (Type 1) with Code 1 (Port Unreachable) on behalf of the Device.

In that case the SCHC C/D MAY act as a router (i.e. it MUST have a routable IPv6 address to generate an ICMPv6 message). When compressing a packet containing an IPv6 header, no compression rules are found and: * if a rule contains some extension headers, a parameter problem may be generated (type 4), * no rule contains the IPv6 device address found in the incoming packet, a no route to destination ICMPv6 message (type 0, code 3) may be generated, * a device IPv6 address is found, but no port matches, a port unreachable ICMPv6 message (type 0, code 4) may be generated,

4.4. Device is the destination of an ICMPv6 error message

In this situation, we assume that a Device has been configured to send information to a server on the Internet. If this server becomes no longer accessible, an ICMPv6 message will be generated back towards the Device by either an intermediate router or the destination. This information can be useful to the Device, for example for reducing the reporting rate in case of periodic reporting of data. Therefore, we compress the ICMPv6 message using SCHC and forward it to the Device over the LPWAN. We also introduce new MO and CDA that can be used to test the presence and/or compress the returning payload.

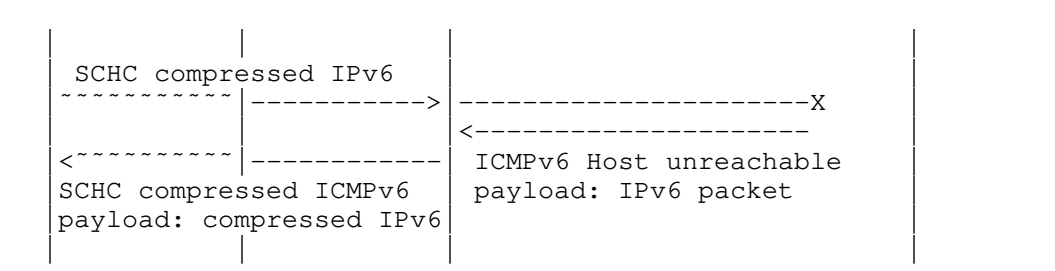


Figure 4: Example of ICMPv6 error message sent back to the Device

Figure 4 illustrates this behavior. The ICMPv6 error message is compressed as described in Section 4.4.1 and forwarded over the LPWAN to the Device.

The SCHC returning message contains the SCHC residue of the ICMPv6 message and MAY contain the compressed original message contained in the ICMP message. The compression can be done by the core SCHC by reversing the direction as if this message was issued by the device.

4.4.1. ICMPv6 error message compression.

The ICMPv6 error messages defined in [RFC4443] contain the fields shown in Figure 5.

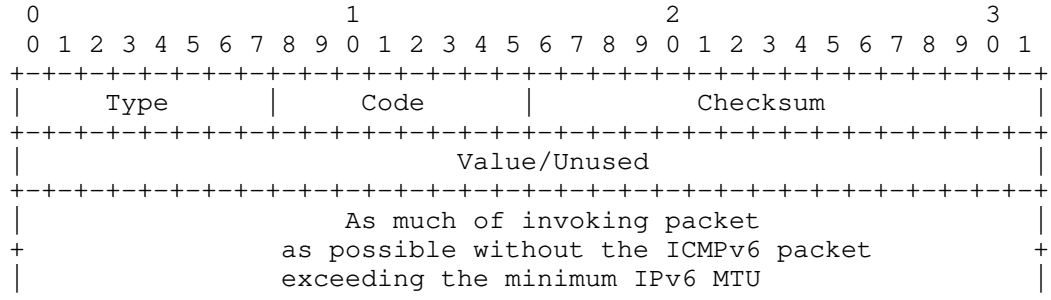


Figure 5: ICMPv6 Error Message format

[RFC4443] states that Type can take the values 1 to 4, and Code can be set to values between 0 and 6. Value is unused for the Destination Unreachable and Time Exceeded messages. It contains the MTU for the Packet Too Big message and a pointer to the byte causing the error for the Parameter Error message. Therefore, Value is never expected to be greater than 1280 in LPWAN networks.

The payload is viewed as a field. An unsued field MUST not appear in the compressoin rules.

The source address of the message SHOULD be ignore, since it can be initiated by any router on the path.

The following generic rule can therefore be used to compress all ICMPv6 error messages as defined today. More specific rules can also be defined to achieve better compression of some error messages.

The Type field can be associated to a matching list [1, 2, 3, 4] and is therefore compressed down to 2 bits. Code can be reduced to 3 bits using the LSB CDA. Value can be sent on 11 bits using the LSB CDA, but if the Device is known to send smaller packets, then the size of this field can be further reduced.

The first rule example Table 3 just sends the ICMP type and code as residue to the device.

Field	FL	FP	DI	Value	Matching Operator	CDA	Sent bits
IPv6 Headers description							
ICMPv6 Type	8	1	Dw	128	equal	not-sent	
ICMPv6 Code	8	1	Dw	[0,1,2,3,4,5,6]	match-mapping	mapping-sent	3
ICMPv6 Payload	var	1	Dw	0	ignore	not-sent	

Table 3: Example of compression rule for a ICMP error to a device

The second rule example Table 4 also only sends the ICMP type and code as residue to the device, but it introduces the new MO "rev-rule-match". This MO will check if a rule matches the payload.

Field	FL	FP	DI	Value	Matching Operator	CDA	Sent bits
				IPv6 Headers description			
ICMPv6 Type	8	1	Dw	128	equal	not-sent	
ICMPv6 Code	8	1	Dw	[0,1,2,3,4,5,6]	match-mapping	mapping-sent	
ICMPv6 Payload	var	1	Dw	0	rev-rule-match	not-sent	

Table 4: Example of compression rule for a ICMP error to a device

By [RFC4443], the rest of the ICMPv6 message must contain as much as possible of the IPv6 offending (invoking) packet that triggered this ICMPv6 error message. This information is used to try and identify the SCHC rule that was used to decompress the offending IPv6 packet. If the rule can be found then the Rule Id is added at the end of the compressed ICMPv6 message. Otherwise the compressed packet ends with the compressed Value field.

The third rule example Table 5 also sends the ICMP type, code and the compresssed payload as residue. It can be noted that this field is identified as "variable" in the rule which will introduce a size before the IPv6 compressed header.

Field	FL	FP	DI	Value	Matching Operator	CDA	Sent bits
IPv6 Headers description							
ICMPv6 Type	8	1	Dw	128	equal	not-sent	
ICMPv6 Code	8	1	Dw	[0,1,2,3,4,5,6]	match-mapping	mapping-sent	
ICMPv6 Payload	var	1	Dw	0	rev-rule-match	rev-compress-sent	(compressed IPv6 header*9) + 4 or +12

Table 5: Example of compression rule for a ICMP error to a device

LT: do we add packet too big, for instance if a fragmentation rule cannot handle a size larger than 1280?

5. YANG identities and tree

Figure 6 shows the augmentation of the Data Model defined in [RFC9363]

This YANG module extends Field ID identities to includes fields contained in ICMPv6 header. Note that the ICMPv6 payload is parsed to the specific field "fid-icmpv6-payload"

It also defines two new Most identities:

* mo-rev-rule-match: The value contained in the Field Value matches a rule. The direction used for matching isthe opposite of the incoming message: UP becomes DOWN and DOWN becomes UP. This MO can be used to test if the Payload contained in the ICMPv6 message matches a rule. This means that the original packet, at the origine of the ICMPv6 message, may have been generated from the SCHC decompression.

* mo-rule-match: The value contained in the Target Value matches a rule. The direction is the one of the incoming message. This MO is not used for ICMPv6 messages, but since it can be used in other situations, it has been included in the Data Model.

The Field Value may be compressed by a rule. The result SHOULD be included in the SCHC message as a variable length residue. It contains the Rule ID used by the compression, the residue, the payload and some padding bits since the variable length init is in bytes.

* cda-rev-compress-sent: The direction used for compression is the opposite of the incoming message: UP becomes DOWN and DOWN becomes UP.

* cda-compress-sent: The direction used for compression is the same as for the incoming message.

module: ietf-schc-oam

```
  augment /schc:schc/schc:rule/schc:nature/schc:compression:
    +--rw proxy-behavior?         schc-oam:proxy-type
    +--rw proxy-behavior-value* [index]
       +--rw index    uint16
       +--rw value?   binary
```

Figure 6: YANG tree

6. YANG Module

```
module ietf-schc-oam {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schc-oam";
  prefix schc-oam;

  import ietf-schc {
      prefix schc;
  }

  organization
    "IETF IPv6 over Low Power Wide-Area Networks (lpwan) working group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/lpwan/about/>
     WG List:  <mailto:p-wan@ietf.org>
     Editor:   Laurent Toutain
       <mailto:laurent.toutain@imt-atlantique.fr>
     Editor:   Ana Minaburo
       <mailto:ana@ackl.io>";
  description
     "
     Copyright (c) 2021 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.
```

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

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

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

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

     This module extends the ietf-schc module to include the compound-ack
     behavior for Ack On Error as defined in RFC YYYY.
     It introduces a new leaf for Ack on Error defining the format of the
     SCHC Ack and add the possibility to send several bitmaps in a single
     answer.";

  revision 2023-06-26 {
    description
      "Initial version for RFC YYYY ";
    reference
      "RFC YYYY: OAM";
  }

  identity fid-icmpv6-base-type {
    base schc:fid-base-type;
    description
      "Field IP base type for ICMPv6 headers described in RFC 4443";
    reference
      "RFC 4443   Internet Control Message Protocol (ICMPv6)
                  for the Internet Protocol Version 6 (IPv6) Specification";
  }

// ICMPv6 Fields

  identity fid-icmpv6-type {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 type field";
  }
```

```
  identity fid-icmpv6-code {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 code field";
  }

  identity fid-icmpv6-checksum {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 checksum field";
  }

    identity fid-icmpv6-mtu {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 MTU (see draft OAM)";
  }

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

  identity fid-icmpv6-identifier {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 identifier field";
  }

  identity fid-icmpv6-sequence {
    base schc:fid-icmpv6-base-type;
    description
      "ICMPv6 sequence number field";
  }


  identity fid-icmpv6-payload {
    base schc:fid-icmpv6-base-type;
    description
      "payload in the ICMPv6 message";
  }

// MO and CDA

  identity mo-rule-match {
    base schc:mo-base-type;
    description
```

```
        "Macthing operator return true, if the TV matches a rule
        keeping UP and DOWN direction." ;
  }

  identity mo-rev-rule-match {
    base schc:mo-base-type;
    description
        "Macthing operator return true, if the TV matches a rule
        reversing UP and DOWN direction." ;
  }


  identity cda-compress-sent {
    base schc:mo-base-type;
    description
        "Send a compressed version of TV keeping UP and
        DOWN direction." ;
  }

  identity  cda-rev-compress-sent {
    base schc:mo-base-type;
    description
        "Send a compressed version of TV reversing UP and
        DOWN direction." ;
  }

// Proxy actions

  identity  proxy-schc-message{
    description
      "Define how the message is proxied after compression";
  }

  identity proxy-none {
    base proxy-schc-message;
    description
      "The message is not proxied and sent to L2,
      default behavior of RFC 8724";
  }

  identity proxy-pingv6 {
    base proxy-schc-message;
    description
      "The message is processed by an ping6 proxy";
  }

  typedef proxy-type {
    type identityref {
```

```
      base proxy-schc-message;
    }
    description
      "type used in rules";
  }

// SCHC rule

  augment "/schc:schc/schc:rule/schc:nature/schc:compression" {
    leaf proxy-behavior {
        type schc-oam:proxy-type;
        default "schc-oam:proxy-none";
        description
              "Entity proxying the SCHC message.";
    }
    list proxy-behavior-value {
        key "index";
        uses schc:tv-struct;
        description
              "Parameters associated to the proxy action.";
    }
    description
      "added to SCHC rules";
  }


}
```

Figure 7: YANG module

7. Security considerations

flood the return path with ICMP error messages.

8. IANA Considerations

TODO

9. References

9.1. Normative References

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

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, DOI 10.17487/RFC4861, September 2007, <https://www.rfc-editor.org/info/rfc4861>.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, DOI 10.17487/RFC4884, April 2007, <https://www.rfc-editor.org/info/rfc4884>.

[RFC6291] Andersson, L., van Helvoort, H., Bonica, R., Romascanu, D., and S. Mansfield, "Guidelines for the Use of the "OAM" Acronym in the IETF", BCP 161, RFC 6291, DOI 10.17487/RFC6291, June 2011, <https://www.rfc-editor.org/info/rfc6291>.

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

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

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

9.2. Informative References

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

Authors' Addresses

Dominique Barthel
Orange SA
28 chemin du Vieux Chene
BP 98
38243 Meylan Cedex
France
Email: dominique.barthel@orange.com

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

Arunprabhu Kandasamy
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne Cedex
France
Email: arun@ackl.io

Diego Dujovne
Universidad Diego Portales
Vergara 432
Santiago
Chile
Email: diego.dujovne@mail.udp.cl

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

lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: September 9, 2021 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:

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

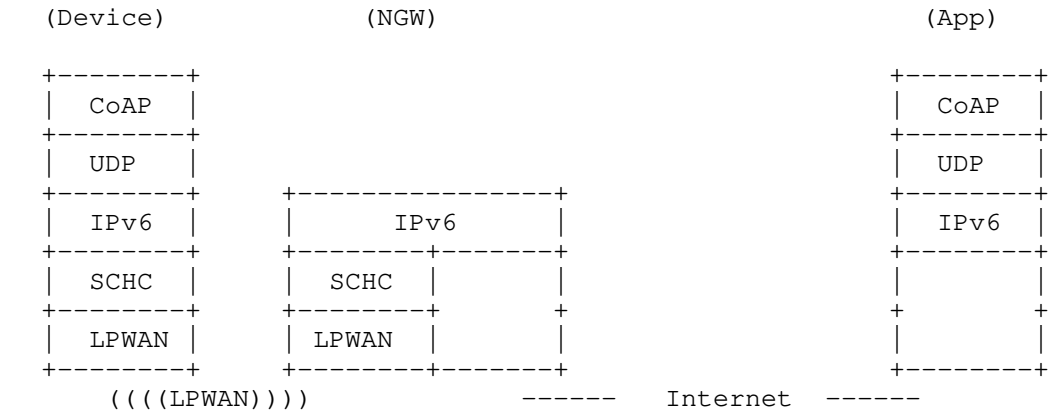


Figure 1: Compression/Decompression at the LPWAN boundary.

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.

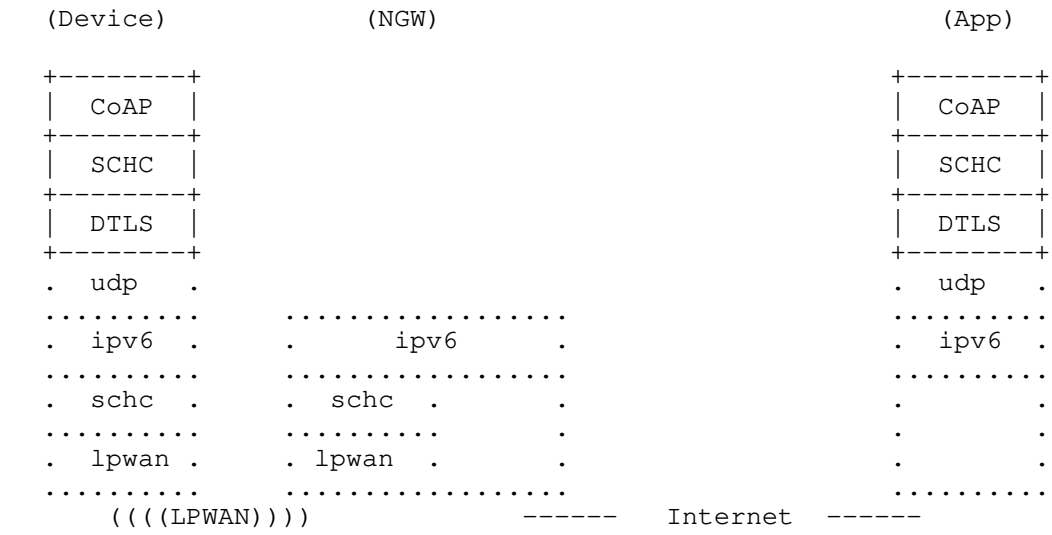


Figure 2: Standalone CoAP end-to-end Compression/Decompression

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.

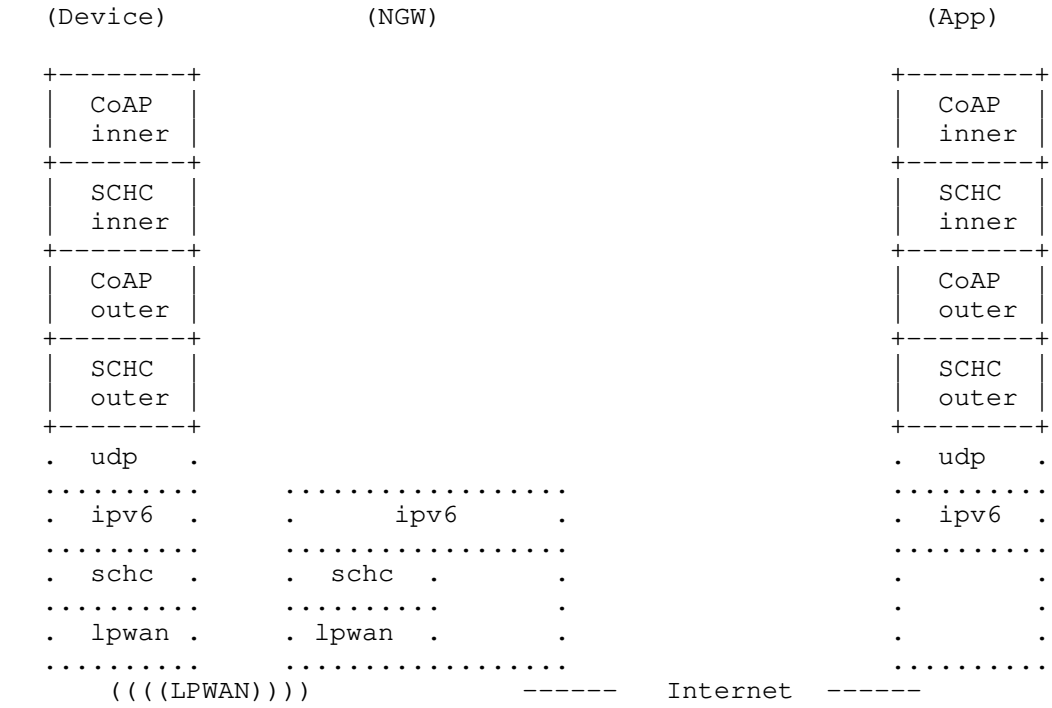


Figure 3: OSCORE compression/decompression.

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:

- 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 returning 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.

- 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.

- 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.

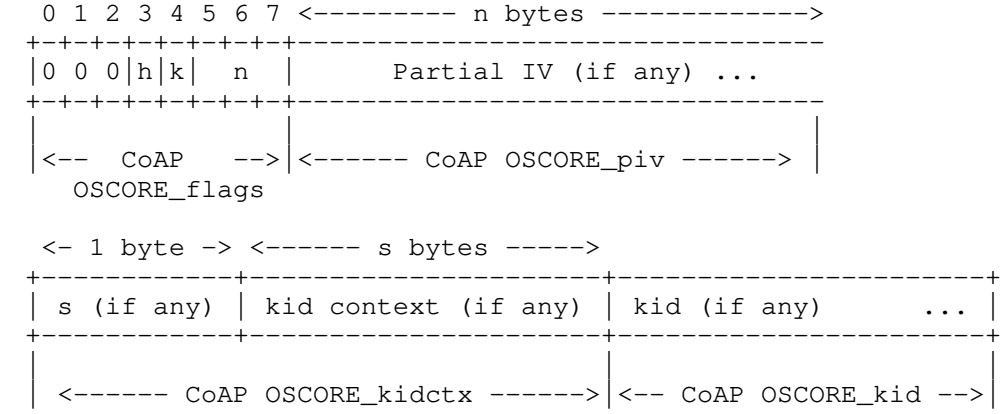


Figure 6: OSCORE Option

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:

- CoAP OSCORE_flags,
- CoAP OSCORE_piv,
- CoAP OSCORE_kidctx,
- 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:

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

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 end-point will use this Security Context to decrypt the message correctly.

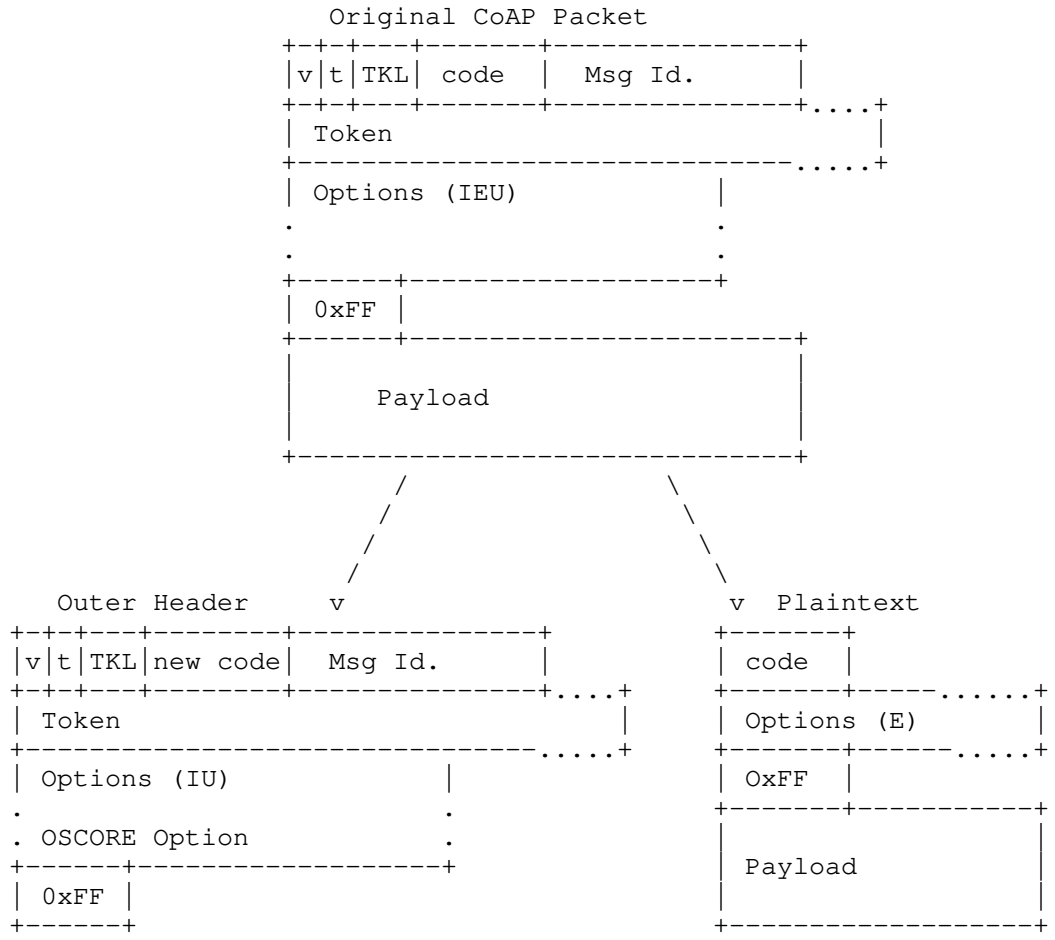


Figure 8: A CoAP packet is split into an OSCORE outer and plaintext

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.

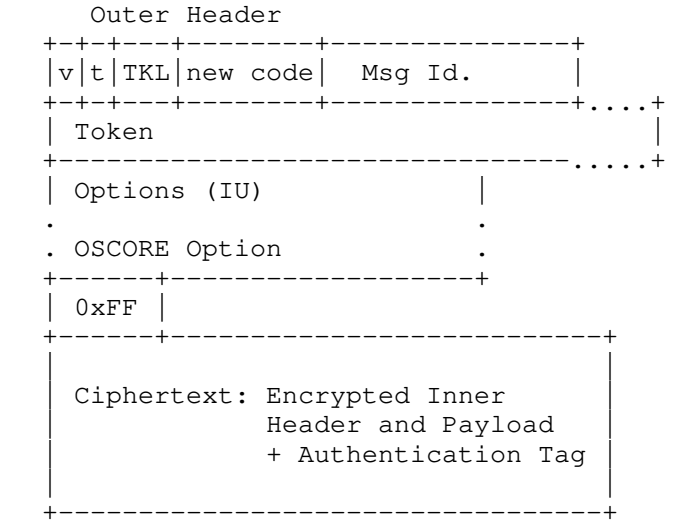


Figure 9: OSCORE message

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.

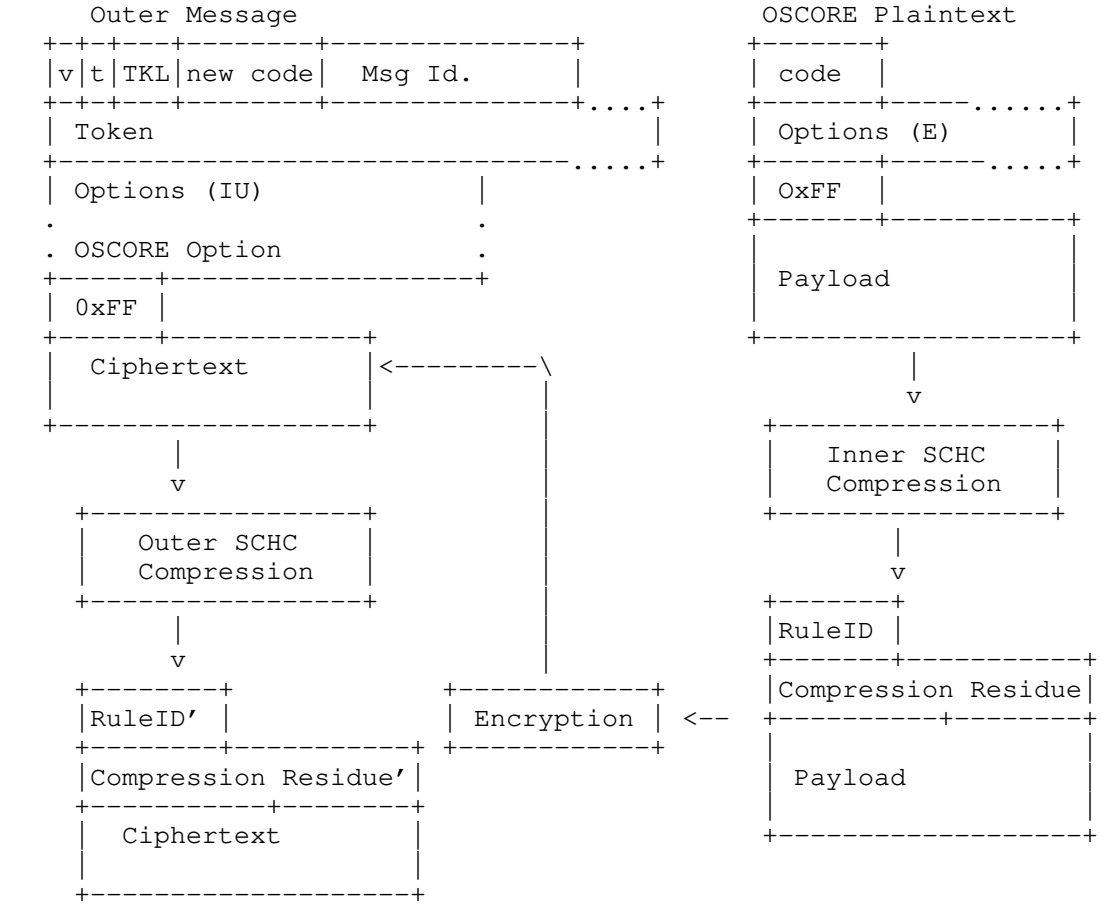


Figure 10: OSCORE Compression Diagram

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.

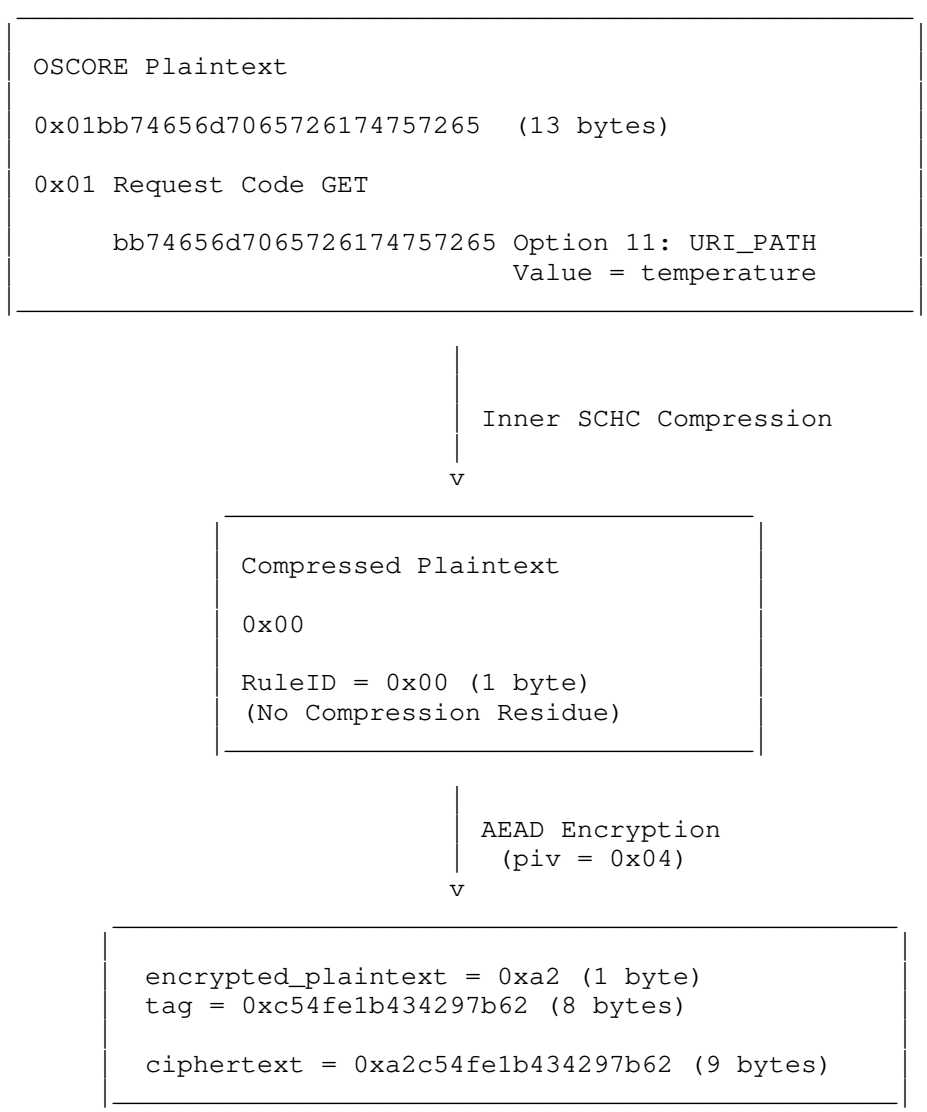


Figure 14: Plaintext compression and encryption for GET Request

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.

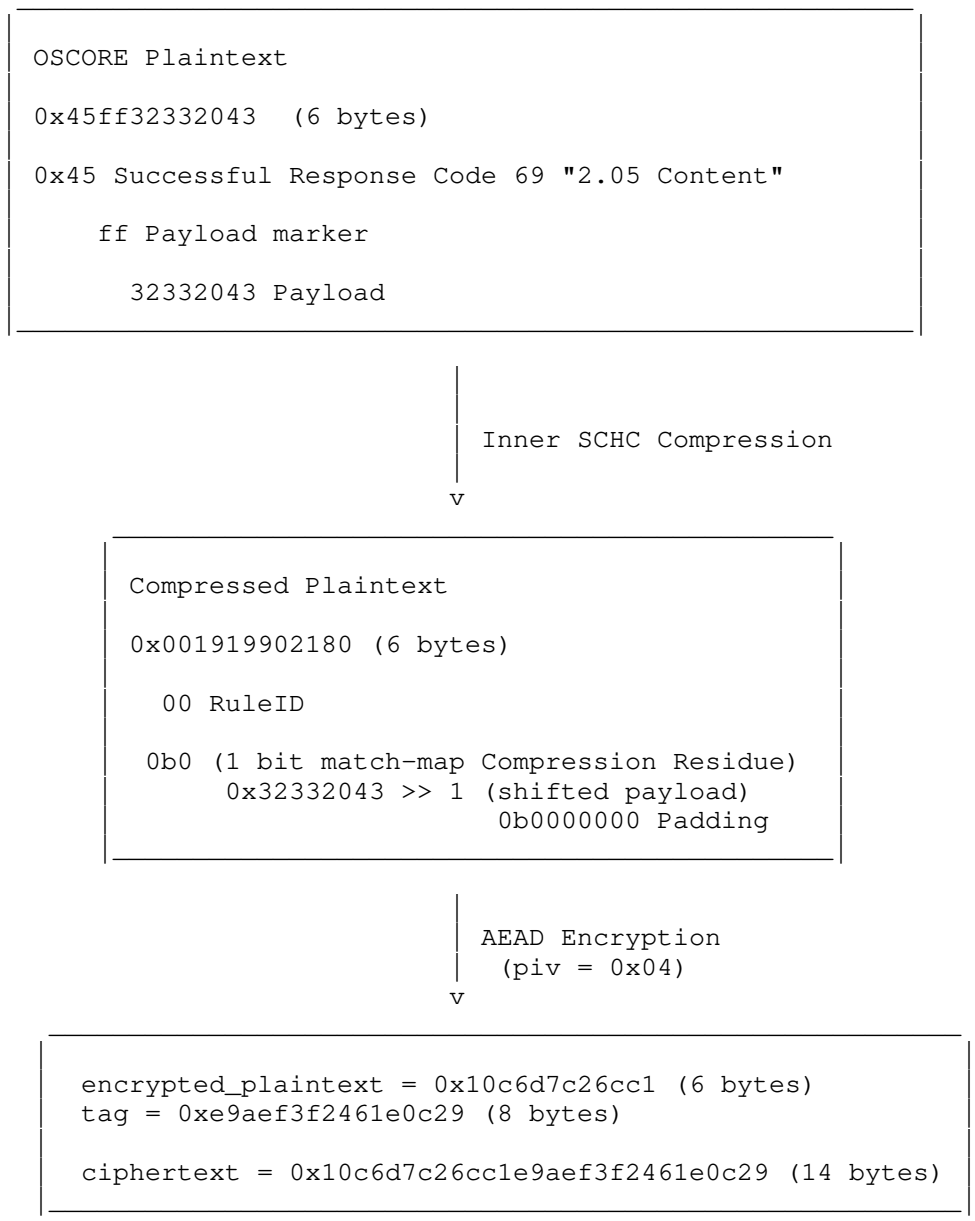


Figure 15: Plaintext compression and encryption for CONTENT Response

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:
==================
0x6144000182d008ff10c6d7c26cc1e9aef3f2461e0c29
(22 bytes)

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

0x0001 = mid
0x82 = token

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

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

Figure 17: Protected and Inner SCHC Compressed CONTENT Response

For the flag bits, 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	C
CoAP MID	16	1	bi	0000	MSB(12)	LSB	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 udpates 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: Standards Track
Expires: May 3, 2020

A. Minaburo
Acklio
L. Toutain
IMT-Atlantique
C. Gomez
Universitat Politecnica de Catalunya
D. Barthel
Orange Labs
JC. Zuniga
SIGFOX

October 31, 2019

Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6
draft-ietf-lpwan-ipv6-static-context-hc-22

Abstract

This document defines the Static Context Header Compression (SCHC) framework, which provides both header compression and fragmentation functionalities. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

SCHC compression is based on a common static context stored both in the LPWAN device and in the network infrastructure side. This document defines a header compression mechanism and its application to compress IPv6/UDP headers.

This document also specifies a fragmentation and reassembly mechanism that is used to support the IPv6 MTU requirement over the LPWAN technologies. Fragmentation is needed for IPv6 datagrams that, after SCHC compression or when such compression was not possible, still exceed the layer-2 maximum payload size.

The SCHC header compression and fragmentation mechanisms are independent of the specific LPWAN technology over which they are used. This document defines generic functionalities and offers flexibility with regard to parameter settings and mechanism choices. This document standardizes the exchange over the LPWAN between two SCHC entities. Settings and choices specific to a technology or a product are expected to be grouped into profiles, which are specified in other documents. Data models for the context and profiles are out of scope.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2020.

Copyright Notice

Copyright (c) 2019 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 4
2. Requirements Notation 5
3. LPWAN Architecture 5
4. Terminology 6
5. SCHC overview 8
5.1. SCHC Packet format 10
5.2. Functional mapping 11
6. Rule ID 12
7. Compression/Decompression 12
7.1. SCHC C/D Rules 13
7.2. Rule ID for SCHC C/D 15
7.3. Packet processing 15
7.4. Matching operators 17
7.5. Compression Decompression Actions (CDA) 18

7.5.1. processing fixed-length fields 19
7.5.2. processing variable-length fields 19
7.5.3. not-sent CDA 20
7.5.4. value-sent CDA 20
7.5.5. mapping-sent CDA 20
7.5.6. LSB CDA 21
7.5.7. DevIID, AppIID CDA 21
7.5.8. Compute-* 21
8. Fragmentation/Reassembly 22
8.1. Overview 22
8.2. SCHC F/R Protocol Elements 22
8.2.1. Messages 22
8.2.2. Tiles, Windows, Bitmaps, Timers, Counters 23
8.2.3. Integrity Checking 25
8.2.4. Header Fields 26
8.3. SCHC F/R Message Formats 28
8.3.1. SCHC Fragment format 28
8.3.2. SCHC ACK format 30
8.3.3. SCHC ACK REQ format 32
8.3.4. SCHC Sender-Abort format 33
8.3.5. SCHC Receiver-Abort format 33
8.4. SCHC F/R modes 34
8.4.1. No-ACK mode 34
8.4.2. ACK-Always mode 36
8.4.3. ACK-on-Error mode 43
9. Padding management 51
10. SCHC Compression for IPv6 and UDP headers 52
10.1. IPv6 version field 52
10.2. IPv6 Traffic class field 52
10.3. Flow label field 52
10.4. Payload Length field 53
10.5. Next Header field 53
10.6. Hop Limit field 53
10.7. IPv6 addresses fields 53
10.7.1. IPv6 source and destination prefixes 54
10.7.2. IPv6 source and destination IID 54
10.8. IPv6 extension headers 54
10.9. UDP source and destination ports 55
10.10. UDP length field 55
10.11. UDP Checksum field 55
11. IANA Considerations 56
12. Security considerations 56
12.1. Security considerations for SCHC Compression/Decompression 56
12.1.1. Forged SCHC Packet 56
12.1.2. Compressed packet size as a side channel to guess a secret token 57
12.1.3. decompressed packet different from the original

packet 58
12.2. Security considerations for SCHC Fragmentation/Reassembly 58
12.2.1. Buffer reservation attack 58
12.2.2. Corrupt Fragment attack 59
12.2.3. Fragmentation as a way to bypass Network Inspection 59
12.2.4. Privacy issues associated with SCHC header fields . 59
13. Acknowledgements 60
14. References 60
14.1. Normative References 60
14.2. Informative References 61
Appendix A. Compression Examples 61
Appendix B. Fragmentation Examples 64
Appendix C. Fragmentation State Machines 71
Appendix D. SCHC Parameters 77
Appendix E. Supporting multiple window sizes for fragmentation . 79
Appendix F. ACK-Always and ACK-on-Error on quasi-bidirectional links 79
Authors' Addresses 81

1. Introduction

This document defines the Static Context Header Compression (SCHC) framework, which provides both header compression and fragmentation functionalities. SCHC has been designed for Low Power Wide Area Networks (LPWAN).

LPWAN technologies impose some strict limitations on traffic. For instance, devices sleep most of the time and may only receive data during short periods of time after transmission, in order to preserve battery. LPWAN technologies are also characterized by a greatly reduced data unit and/or payload size (see [RFC8376]).

Header compression is needed for efficient Internet connectivity to a node within an LPWAN network. The following properties of LPWAN networks can be exploited to get an efficient header compression:

- The network topology is star-oriented, which means that all packets between the same source-destination pair follow the same path. For the needs of this document, the architecture can simply be described as Devices (Dev) exchanging information with LPWAN Application Servers (App) through a Network Gateway (NGW).

- Because devices embed built-in applications, the traffic flows to be compressed are known in advance. Indeed, new applications are less frequently installed in an LPWAN device, than they are in a general-purpose computer or smartphone.

SCHC compression uses a Context (a set of Rules) in which information about header fields is stored. This Context is static: the values of the header fields and the actions to do compression/decompression do not change over time. This avoids the need for complex resynchronization mechanisms. Indeed, a return path may be more restricted/expensive, sometimes completely unavailable [RFC8376]. A compression protocol that relies on feedback is not compatible with the characteristics of such LPWANs.

In most cases, a small Rule identifier is enough to represent the full IPv6/UDP headers. The SCHC header compression mechanism is independent of the specific LPWAN technology over which it is used.

Furthermore, some LPWAN technologies do not provide a fragmentation functionality; to support the IPv6 MTU requirement of 1280 bytes [RFC8200], they require a fragmentation protocol at the adaptation layer below IPv6. Accordingly, this document defines an optional fragmentation/reassembly mechanism for LPWAN technologies to support the IPv6 MTU requirement.

This document defines generic functionality and offers flexibility with regard to parameters settings and mechanism choices. Technology-specific settings are expected to be grouped into Profiles specified in other documents.

2. Requirements Notation

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. LPWAN Architecture

LPWAN network architectures are similar among them, but each LPWAN technology names architecture elements differently. In this document, we use terminology from [RFC8376], which identifies the following entities in a typical LPWAN network (see Figure 1):

o Devices (Dev) are the end-devices or hosts (e.g. sensors, actuators, etc.). There can be a very high density of devices per radio gateway.

o The Radio Gateway (RGW) is the end point of the constrained link.

o The Network Gateway (NGW) is the interconnection node between the Radio Gateway and the Internet.

o Application Server (App) is the end point of the application level protocol on the Internet side.

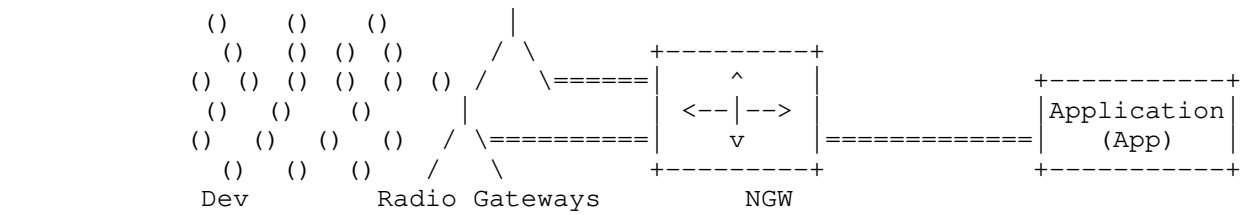


Figure 1: LPWAN Architecture, simplified from that shown in RFC8376

4. Terminology

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

The SCHC acronym is pronounced like "sheek" in English (or "chic" in French). Therefore, this document writes "a SCHC Packet" instead of "an SCHC Packet".

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

- AppIID: Application Interface Identifier. The IID that identifies the application server interface.

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

- CDA: Compression/Decompression Action. Describes the pair of inverse actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field.

- Compression Residue. The bits that remain to be sent (beyond the Rule ID itself) after applying the SCHC compression.

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

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

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

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

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

- Field Description. A tuple containing identifier, value, matching operator and actions to be applied to a field.

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

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

- FP: when a Field is expected to appear multiple times in a header, Field Position specifies the occurrence this Field Description applies to (for example, first uri-path option, second uri-path, etc. in a CoAP header).

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

- L2: Layer two. The immediate lower layer SCHC interfaces with. It is provided by an underlying LPWAN technology. It does not necessarily correspond to the OSI model definition of Layer 2.

- L2 Word: this is the minimum subdivision of payload data that the L2 will carry. In most L2 technologies, the L2 Word is an octet. In bit-oriented radio technologies, the L2 Word might be a single bit. The L2 Word size is assumed to be constant over time for each device.

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

- Padding (P). Extra bits that may be appended by SCHC to a data unit that it passes to the underlying Layer 2 for transmission. SCHC itself operates on bits, not bytes, and does not have any alignment prerequisite. See Section 9.

- Profile: SCHC offers variations in the way it is operated, with a number of parameters listed in Appendix D. A Profile indicates a

particular setting of all these parameters. Both ends of a SCHC communication must be provisioned with the same Profile information and with the same set of Rules before the communication starts, so that there is no ambiguity in how they expect to communicate.

- Rule: A set of Field Descriptions.

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

- SCHC C/D: SCHC Compressor/Decompressor. A mechanism used on both sides, at the Dev and at the network, to achieve Compression/ Decompression of headers.

- SCHC F/R: SCHC Fragmentation / Reassembly. A mechanism used on both sides, at the Dev and at the network, to achieve Fragmentation / Reassembly of SCHC Packets.

- SCHC Packet: A packet (e.g. an IPv6 packet) whose header has been compressed as per the header compression mechanism defined in this document. If the header compression process is unable to actually compress the packet header, the packet with the uncompressed header is still called a SCHC Packet (in this case, a Rule ID is used to indicate that the packet header has not been compressed). See Section 7 for more details.

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

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

Additional terminology for the optional SCHC Fragmentation / Reassembly mechanism (SCHC F/R) is found in Section 8.2.

5. SCHC overview

SCHC can be characterized as an adaptation layer between an upper layer (typically, IPv6) and an underlying layer (typically, an LPWAN technology). SCHC comprises two sublayers (i.e. the Compression sublayer and the Fragmentation sublayer), as shown in Figure 2.

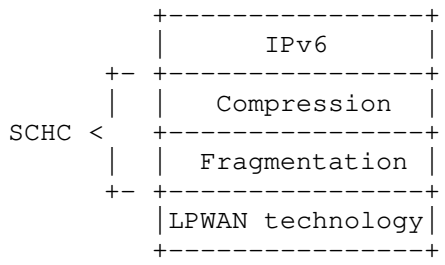
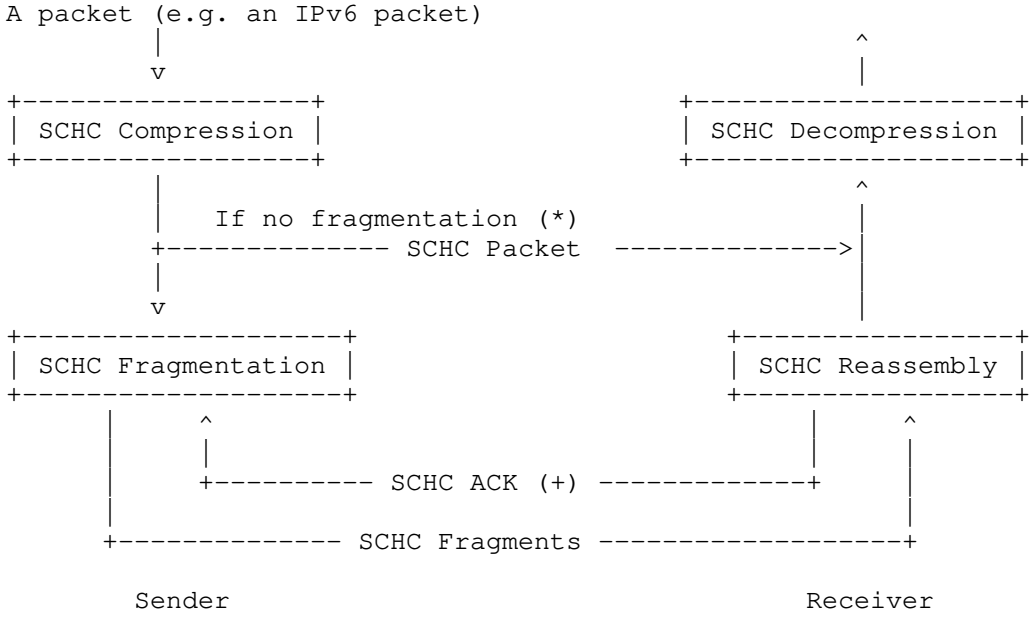


Figure 2: Protocol stack comprising IPv6, SCHC and an LPWAN technology

Before an upper layer packet (e.g. an IPv6 packet) is transmitted to the underlying layer, header compression is first attempted. The resulting packet is called a SCHC Packet, whether or not any compression is performed. If needed by the underlying layer, the optional SCHC Fragmentation MAY be applied to the SCHC Packet. The inverse operations take place at the receiver. This process is illustrated in Figure 3.



*: the decision to not use SCHC Fragmentation is left to each Profile.
+: optional, depends on Fragmentation mode.

Figure 3: SCHC operations at the Sender and the Receiver

5.1. SCHC Packet format

The SCHC Packet is composed of the Compressed Header followed by the payload from the original packet (see Figure 4). The Compressed Header itself is composed of the Rule ID and a Compression Residue, which is the output of compressing the packet header with that Rule (see Section 7). The Compression Residue may be empty. Both the Rule ID and the Compression Residue potentially have a variable size, and are not necessarily a multiple of bytes in size.

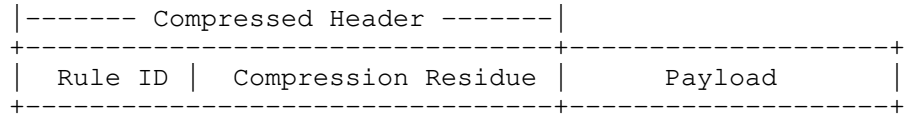


Figure 4: SCHC Packet

5.2. Functional mapping

Figure 5 maps the functional elements of Figure 3 onto the LPWAN architecture elements of Figure 1.

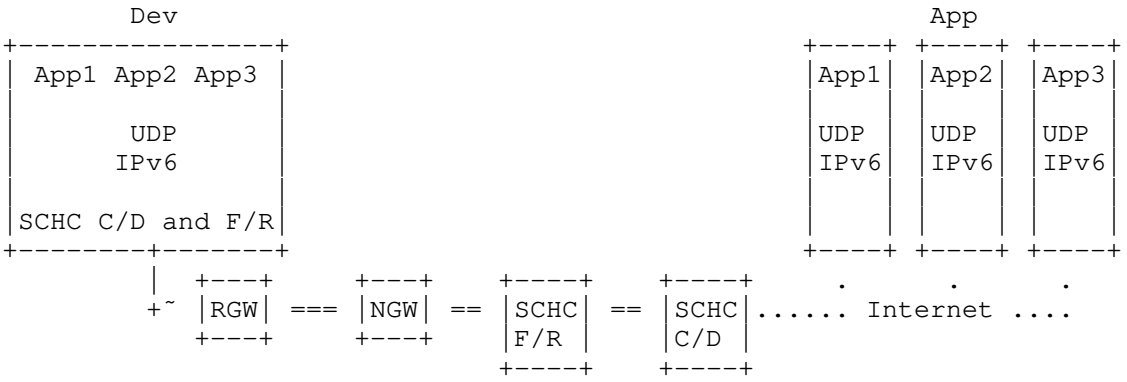


Figure 5: Architecture

SCHC C/D and SCHC F/R are located on both sides of the LPWAN transmission, hereafter called "the Dev side" and "the Network infrastructure side".

The operation in the Uplink direction is as follows. The Device application uses IPv6 or IPv6/UDP protocols. Before sending the packets, the Dev compresses their headers using SCHC C/D and, if the SCHC Packet resulting from the compression needs to be fragmented by SCHC, SCHC F/R is performed (see Section 8). The resulting SCHC Fragments are sent to an LPWAN Radio Gateway (RGW) which forwards them to a Network Gateway (NGW). The NGW sends the data to a SCHC F/R for re-assembly (if needed) and then to the SCHC C/D for decompression. After decompression, the packet can be sent over the Internet to one or several LPWAN Application Servers (App).

The SCHC F/R and C/D on the Network infrastructure side can be part of the NGW, or located in the Internet as long as a tunnel is established between them and the NGW. For some LPWAN technologies, it may be suitable to locate the SCHC F/R functionality nearer the NGW, in order to better deal with time constraints of such technologies.

The SCHC C/Ds on both sides MUST share the same set of Rules. So MUST the SCHC F/Rs on both sides.

The operation in the Downlink direction is similar to that in the Uplink direction, only reversing the order in which the architecture elements are traversed.

6. Rule ID

Rule IDs identify the Rules used for Compression/Decompression or for Fragmentation/Reassembly.

The scope of the Rule ID of a Compression/Decompression Rule is the link between the SCHC C/D in a given Dev and the corresponding SCHC C/D in the Network insfractructure side. The scope of the Rule ID of a Fragmentation/Reassembly Rule is the link between the SCHC F/R in a given Dev and the corresponding SCHC F/R in the Network insfractructure side. If such a link is bidirectional, the scope includes both directions.

Inside their scopes, Rules for Compression/Decompression and Rules for Fragmentation/Reassembly share the same Rule ID space.

The size of the Rule IDs is not specified in this document, as it is implementation-specific and can vary according to the LPWAN technology and the number of Rules, among others. It is defined in Profiles.

The Rule IDs are used:

- For SCHC C/D, to identify the Rule (i.e., the set of Field Descriptions) that is used to compress a packet header.
  - At least one Rule ID MUST be allocated to tagging packets for which SCHC compression was not possible (no matching compression Rule was found).
- In SCHC F/R, to identify the specific mode and settings of F/R for one direction of traffic (Up or Dw).
  - When F/R is used for both communication directions, at least two Rule ID values are needed for F/R, one per direction of traffic. This is because F/R may entail control messages flowing in the reverse direction compared to data traffic.

7. Compression/Decompression

Compression with SCHC is based on using a set of Rules, called the Context, to compress or decompress headers. SCHC avoids Context synchronization traffic, which consumes considerable bandwidth in other header compression mechanisms such as RoHC [RFC5795]. Since

the content of packets is highly predictable in LPWAN networks, static Contexts may be stored beforehand. The Contexts MUST be stored at both ends, and they can be learned by a provisioning protocol or by out of band means, or they can be pre-provisioned. The way the Contexts are provisioned is out of the scope of this document.

7.1. SCHC C/D Rules

The main idea of the SCHC compression scheme is to transmit the Rule ID to the other end instead of sending known field values. This Rule ID identifies a Rule that matches the original packet values. Hence, when a value is known by both ends, it is only necessary to send the corresponding Rule ID over the LPWAN network. The manner by which Rules are generated is out of the scope of this document. The Rules MAY be changed at run-time but the mechanism is out of scope of this document.

The Context is a set of Rules. See Figure 6 for a high level, abstract representation of the Context. The formal specification of the representation of the Rules is outside the scope of this document.

Each Rule itself contains a list of Field Descriptions composed of a Field Identifier (FID), a Field Length (FL), a Field Position (FP), a Direction Indicator (DI), a Target Value (TV), a Matching Operator (MO) and a Compression/Decompression Action (CDA).

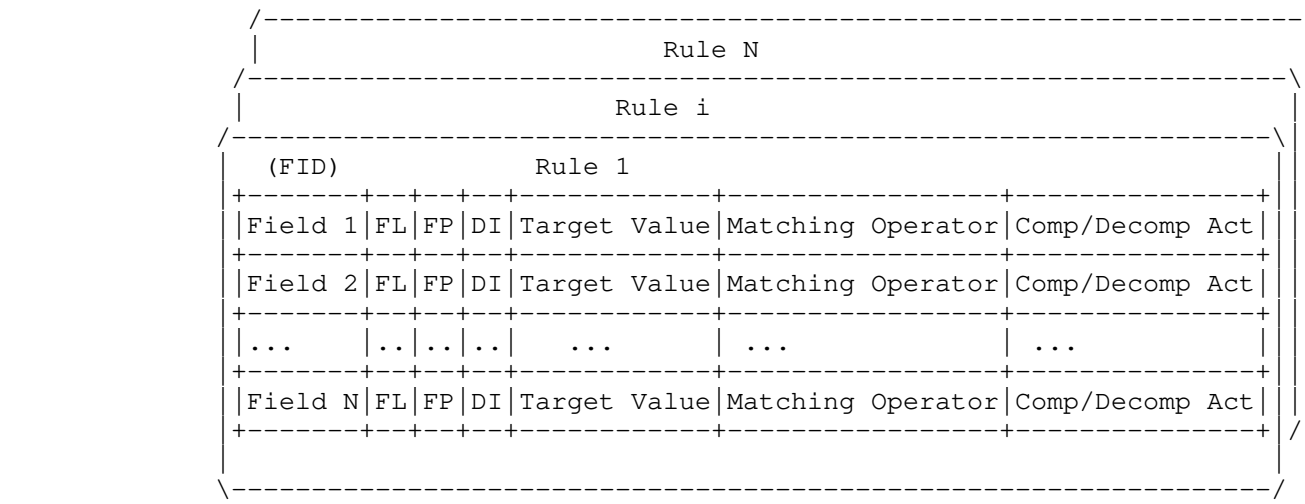


Figure 6: A Compression/Decompression Context

A Rule does not describe how the compressor parses a packet header to find and identify each field (e.g. the IPv6 Source Address, the UDP Destination Port or a CoAP URI path option). It is assumed that there is a protocol parser alongside SCHC that is able to identify all the fields encountered in the headers to be compressed, and to label them with a Field ID. Rules only describe the compression/ decompression behavior for each header field, after it has been identified.

In a Rule, the Field Descriptions are listed in the order in which the fields appear in the packet header. The Field Descriptions describe the header fields with the following entries:

- Field ID (FID) designates a protocol and field (e.g. UDP Destination Port), unambiguously among all protocols that a SCHC compressor processes. In the presence of protocol nesting, the Field ID also identifies the nesting.

- Field Length (FL) represents the length of the field. It can be either a fixed value (in bits) if the length is known when the Rule is created or a type if the length is variable. The length of a header field is defined by its own protocol specification (e.g. IPv6 or UDP). If the length is variable, the type defines the process to compute the length and its unit (bits, bytes...).

- Field Position (FP): most often, a field only occurs once in a packet header. However, some fields may occur multiple times. An example is the uri-path of CoAP. FP indicates which occurrence this Field Description applies to. If FP is not specified in the Field Description, it takes the default value of 1. The value 1 designates the first occurrence. The value 0 is special. It means "don't care", see Section 7.3.

- A Direction Indicator (DI) indicates the packet direction(s) this Field Description applies to. Three values are possible:

  * UPLINK (Up): this Field Description is only applicable to packets sent by the Dev to the App,

  * DOWNLINK (Dw): this Field Description is only applicable to packets sent from the App to the Dev,

  * BIDIRECTIONAL (Bi): this Field Description is applicable to packets traveling both Up and Dw.

- Target Value (TV) is the value used to match against the packet header field. The Target Value can be a scalar value of any type (integer, strings, etc.) or a more complex structure (array, list,

etc.). The types and representations are out of scope for this document.

- Matching Operator (MO) is the operator used to match the Field Value and the Target Value. The Matching Operator may require some parameters. MO is only used during the compression phase. The set of MOs defined in this document can be found in Section 7.4.

- Compression Decompression Action (CDA) describes the compression and decompression processes to be performed after the MO is applied. Some CDAs might use parameter values for their operation. CDAs are used in both the compression and the decompression functions. The set of CDAs defined in this document can be found in Section 7.5.

7.2. Rule ID for SCHC C/D

Rule IDs are sent by the compression function in one side and are received for the decompression function in the other side. In SCHC C/D, the Rule IDs are specific to the Context related to one Dev. Hence, multiple Dev instances, which refer to different header compression Contexts, MAY reuse the same Rule ID for different Rules. On the Network infrastructure side, in order to identify the correct Rule to be applied, the SCHC Decompressor needs to associate the Rule ID with the Dev identifier. Similarly, the SCHC Compressor on the Network infrastructure side first identifies the destination Dev before looking for the appropriate compression Rule (and associated Rule ID) in the Context of that Dev.

7.3. Packet processing

The compression/decompression process follows several phases:

- Compression Rule selection: the general idea is to browse the Rule set to find a Rule that has a matching Field Descriptor (given the DI and FP) for all and only those header fields that appear in the packet being compressed. The detailed algorithm is the following:

  * The first step is to check the Field Identifiers (FID). If any header field of the packet being examined cannot be matched with a Field Description with the correct FID, the Rule MUST be disregarded. If any Field Description in the Rule has a FID that cannot be matched to one of the header fields of the packet being examined, the Rule MUST be disregarded.

  * The next step is to match the Field Descriptions by their direction, using the Direction Indicator (DI). If any field of

the packet header cannot be matched with a Field Description with the correct FID and DI, the Rule MUST be disregarded.

* Then the Field Descriptions are further selected according to Field Position (FP). If any field of the packet header cannot be matched with a Field Description with the correct FID, DI and FP, the Rule MUST be disregarded.

  The value 0 for FP means "don't care", i.e. the comparison of this Field Description's FP with the position of the field of the packet header being compressed returns True, whatever that position. FP=0 can be useful to build compression Rules for protocols headers in which some fields order is irrelevant. An example could be uri-queries in CoAP. Care needs to be exercised when writing Rules containing FP=0 values. Indeed, it may result in decompressed packets having fields ordered differently compared to the original packet.

* Once each header field has been associated with a Field Description with matching FID, DI and FP, each packet field's value is then compared to the corresponding Target Value (TV) stored in the Rule for that specific field, using the matching operator (MO). If every field in the packet header satisfies the corresponding matching operators (MO) of a Rule (i.e. all MO results are True), that Rule is valid for use to compress the header. Otherwise, the Rule MUST be disregarded.

  This specification does not prevent multiple Rules from matching the above steps and therefore being valid for use. Whether multiple valid Rules are allowed or not and what to do in the case of multiple valid Rules are left to the implementation. As long as the same Rule set is installed at both ends, this degree of freedom does not constitute an interoperability issue.

* If no valid compression Rule is found, then the header MUST be sent in its entirety using the Rule ID of the "default" Rule dedicated to this purpose. Sending an uncompressed header is likely to require SCHC F/R.

o Compression: if a valid Rule was found, each field of the header is compressed according to the Compression/Decompression Actions (CDAs) of the Rule. The fields are compressed in the order that the Field Descriptions appear in the Rule. The compression of each field results in a residue, which may be empty. The Compression Residue for the packet header is the concatenation of the non-empty residues for each field of the header, in the order the Field Descriptions appear in the Rule. The order in which the

Field Descriptions appear in the Rule is therefore semantically important.

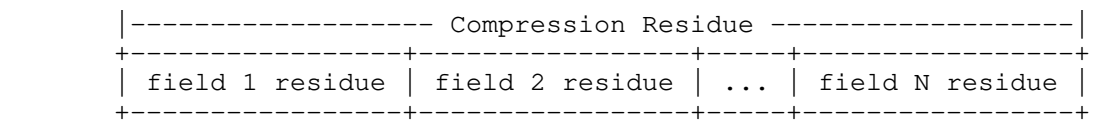


Figure 7: Compression Residue structure

- Sending: The Rule ID is sent to the other end followed by the Compression Residue (which could be empty) or the uncompressed header, and directly followed by the payload (see Figure 4). The way the Rule ID is sent will be specified in the Profile and is out of the scope of the present document. For example, it could be included in an L2 header or sent as part of the L2 payload.

- Decompression: when decompressing, on the Network infrastructure side the SCHC C/D needs to find the correct Rule based on the L2 address of the Dev; in this way, it can use the DevIID and the Rule ID. On the Dev side, only the Rule ID is needed to identify the correct Rule since the Dev typically only holds Rules that apply to itself.

  This Rule describes the compressed header format. From this, the decompressor determines the order of the residues, the fixed-sized or variable-sized nature of each residue (see Section 7.5.2), and the size of the fixed-sized residues.

  From the received compressed header, it can therefore retrieve all the residue values and associate them to the corresponding header fields.

  For each field in the header, the receiver applies the CDA action associated to that field in order to reconstruct the original header field value. The CDA application order can be different from the order in which the fields are listed in the Rule. In particular, Compute-* MUST be applied after the application of the CDAs of all the fields it computes on.

7.4. Matching operators

Matching Operators (MOs) are functions used by both SCHC C/D endpoints. They are not typed and can be applied to integer, string or any other data type. The result of the operation can either be True or False. MOs are defined as follows:

- equal: The match result is True if the field value in the packet matches the TV.

- ignore: No matching is attempted between the field value in the packet and the TV in the Rule. The result is always true.

- MSB(x): A match is obtained if the most significant (leftmost) x bits of the packet header field value are equal to the TV in the Rule. The x parameter of the MSB MO indicates how many bits are involved in the comparison. If the FL is described as variable, the x parameter must be a multiple of the FL unit. For example, x must be multiple of 8 if the unit of the variable length is bytes.

- match-mapping: With match-mapping, the Target Value is a list of values. Each value of the list is identified by an index. Compression is achieved by sending the index instead of the original header field value. This operator matches if the header field value is equal to one of the values in the target list.

7.5. Compression Decompression Actions (CDA)

The Compression Decompression Action (CDA) describes the actions taken during the compression of header fields and the inverse action taken by the decompressor to restore the original value.

Action	Compression	Decompression
not-sent	elided	use TV stored in Rule
value-sent	send	use received value
mapping-sent	send index	retrieve value from TV list
LSB	send LSB	concat. TV and received value
compute-*	elided	recompute at decompressor
DevIID	elided	build IID from L2 Dev addr
AppIID	elided	build IID from L2 App addr

Table 1: Compression and Decompression Actions

Table 1 summarizes the basic actions that can be used to compress and decompress a field. The first column shows the action's name. The second and third columns show the compression and decompression behaviors for each action.

7.5.1. processing fixed-length fields

If the field is identified in the Field Description as being of fixed length, then applying the CDA to compress this field results in a fixed amount of bits. The residue for that field is simply the bits resulting from applying the CDA to the field. This value may be empty (e.g. not-sent CDA), in which case the field residue is absent from the Compression Residue.

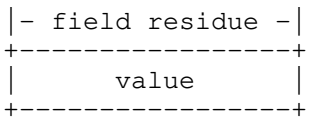


Figure 8: fixed sized field residue structure

7.5.2. processing variable-length fields

If the field is identified in the Field Description as being of variable length, then applying the CDA to compress this field may result in a value of fixed size (e.g. not-sent or mapping-sent) or of variable size (e.g. value-sent or LSB). In the latter case, the residue for that field is the bits that result from applying the CDA to the field, preceded with the size of the value. The most significant bit of the size is stored to the left (leftmost bit of the residue field).

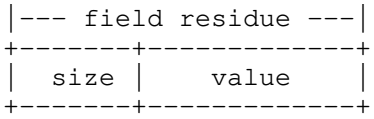


Figure 9: variable sized field residue structure

The size (using the unit defined in the FL) is encoded on 4, 12 or 28 bits as follows:

- If the size is between 0 and 14, it is encoded as a 4 bits unsigned integer.
- Sizes between 15 and 254 are encoded as 0b1111 followed by the 8 bits unsigned integer.
- Larger sizes are encoded as 0xfff followed by the 16 bits unsigned integer.

If the field is identified in the Field Description as being of variable length and this field is not present in the packet header being compressed, size 0 MUST be sent to denote its absence.

7.5.3. not-sent CDA

The not-sent action can be used when the field value is specified in a Rule and therefore known by both the Compressor and the Decompressor. This action SHOULD be used with the "equal" MO. If MO is "ignore", there is a risk to have a decompressed field value different from the original field that was compressed.

The compressor does not send any residue for a field on which not-sent compression is applied.

The decompressor restores the field value with the Target Value stored in the matched Rule identified by the received Rule ID.

7.5.4. value-sent CDA

The value-sent action can be used when the field value is not known by both the Compressor and the Decompressor. The field is sent in its entirety, using the same bit order as in the original packet header.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.5.2.

This action is generally used with the "ignore" MO.

7.5.5. mapping-sent CDA

The mapping-sent action is used to send an index (the index into the Target Value list of values) instead of the original value. This action is used together with the "match-mapping" MO.

On the compressor side, the match-mapping Matching Operator searches the TV for a match with the header field value. The mapping-sent CDA then sends the corresponding index as the field residue. The most significant bit of the index is stored to the left (leftmost bit of the residue field).

On the decompressor side, the CDA uses the received index to restore the field value by looking up the list in the TV.

The number of bits sent is the minimal size for coding all the possible indices.

The first element in the list MUST be represented by index value 0, and successive elements in the list MUST have indices incremented by 1.

7.5.6. LSB CDA

The LSB action is used together with the "MSB(x)" MO to avoid sending the most significant part of the packet field if that part is already known by the receiving end.

The compressor sends the Least Significant Bits as the field residue value. The number of bits sent is the original header field length minus the length specified in the MSB(x) MO. The bits appear in the residue in the same bit order as in the original packet header.

The decompressor concatenates the x most significant bits of Target Value and the received residue value.

If this action is performed on a variable length field, the size of the residue value (using the units defined in FL) MUST be sent as described in Section 7.5.2.

7.5.7. DevIID, AppIID CDA

These actions are used to process respectively the Dev and the App Interface Identifiers (DevIID and AppIID) of the IPv6 addresses. AppIID CDA is less common since most current LPWAN technologies frames contain a single L2 address, which is the Dev’s address.

The IID value MAY be computed from the Device ID present in the L2 header, or from some other stable identifier. The computation is specific to each Profile and MAY depend on the Device ID size.

In the downlink direction (Dw), at the compressor, the DevIID CDA may be used to generate the L2 addresses on the LPWAN, based on the packet’s Destination Address.

7.5.8. Compute-*

Some fields can be elided at the compressor and recomputed locally at the decompressor.

Because the field is uniquely identified by its Field ID (e.g. UDP length), the relevant protocol specification unambiguously defines the algorithm for such computation.

Examples of fields that know how to recompute themselves are UDP length, IPv6 length and UDP checksum.

8. Fragmentation/Reassembly

8.1. Overview

In LPWAN technologies, the L2 MTU typically ranges from tens to hundreds of bytes. Some of these technologies do not have an internal fragmentation/reassembly mechanism.

The optional SCHC Fragmentation/Reassembly (SCHC F/R) functionality enables such LPWAN technologies to comply with the IPv6 MTU requirement of 1280 bytes [RFC8200]. It is OPTIONAL to implement per this specification, but Profiles may specify that it is REQUIRED.

This specification includes several SCHC F/R modes, which allow for a range of reliability options such as optional SCHC Fragment retransmission. More modes may be defined in the future.

The same SCHC F/R mode MUST be used for all SCHC Fragments of a given SCHC Packet. This document does not specify which mode(s) must be implemented and used over a specific LPWAN technology. That information will be given in Profiles.

SCHC allows transmitting non-fragmented SCHC Packet concurrently with fragmented SCHC Packets. In addition, SCHC F/R provides protocol elements that allow transmitting several fragmented SCHC Packets concurrently, i.e. interleaving the transmission of fragments from different fragmented SCHC Packets. A Profile MAY restrict the latter behavior.

The L2 Word size (see Section 4) determines the encoding of some messages. SCHC F/R usually generates SCHC Fragments and SCHC ACKs that are multiples of L2 Words.

8.2. SCHC F/R Protocol Elements

This subsection describes the different elements that are used to enable the SCHC F/R functionality defined in this document. These elements include the SCHC F/R messages, tiles, windows, bitmaps, counters, timers and header fields.

The elements are described here in a generic manner. Their application to each SCHC F/R mode is found in Section 8.4.

8.2.1. Messages

SCHC F/R defines the following messages:

- SCHC Fragment: A message that carries part of a SCHC Packet from the sender to the receiver.

- SCHC ACK: An acknowledgement for fragmentation, by the receiver to the sender. This message is used to indicate whether or not the reception of pieces of, or the whole of the fragmented SCHC Packet, was successful.

- SCHC ACK REQ: A request by the sender for a SCHC ACK from the receiver.

- SCHC Sender-Abort: A message by the sender telling the receiver that it has aborted the transmission of a fragmented SCHC Packet.

- SCHC Receiver-Abort: A message by the receiver to tell the sender to abort the transmission of a fragmented SCHC Packet.

The format of these messages is provided in Section 8.3.

8.2.2. Tiles, Windows, Bitmaps, Timers, Counters

8.2.2.1. Tiles

The SCHC Packet is fragmented into pieces, hereafter called tiles. The tiles MUST be non-empty and pairwise disjoint. Their union MUST be equal to the SCHC Packet.

See Figure 10 for an example.

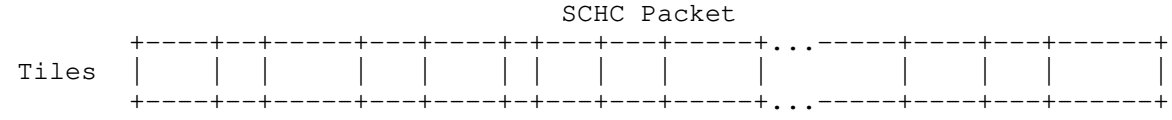


Figure 10: a SCHC Packet fragmented in tiles

Modes (see Section 8.4) MAY place additional constraints on tile sizes.

Each SCHC Fragment message carries at least one tile in its Payload, if the Payload field is present.

8.2.2.2. Windows

Some SCHC F/R modes may handle successive tiles in groups, called windows.

If windows are used

- all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW_SIZE.
- WINDOW_SIZE MUST be specified in a Profile.
- the windows are numbered.
- their numbers MUST increment by 1 from 0 upward, from the start of the SCHC Packet to its end.
- the last window MUST contain WINDOW_SIZE tiles or less.
- tiles are numbered within each window.
- the tile indices MUST decrement by 1 from WINDOW_SIZE - 1 downward, looking from the start of the SCHC Packet toward its end.
- each tile of a SCHC Packet is therefore uniquely identified by a window number and a tile index within this window.

See Figure 11 for an example.

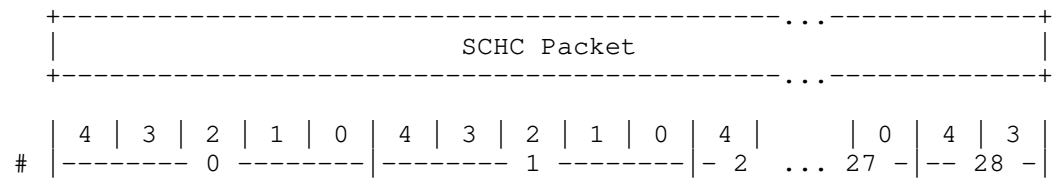


Figure 11: a SCHC Packet fragmented in tiles grouped in 29 windows, with WINDOW_SIZE = 5

Appendix E discusses the benefits of selecting one among multiple window sizes depending on the size of the SCHC Packet to be fragmented.

When windows are used

- Bitmaps (see Section 8.2.2.3) MAY be sent back by the receiver to the sender in a SCHC ACK message.
- A Bitmap corresponds to exactly one Window.

8.2.2.3. Bitmaps

Each bit in the Bitmap for a window corresponds to a tile in the window. Each Bitmap has therefore WINDOW_SIZE bits. The bit at the left-most position corresponds to the tile numbered WINDOW_SIZE - 1. Consecutive bits, going right, correspond to sequentially decreasing tile indices. In Bitmaps for windows that are not the last one of a SCHC Packet, the bit at the right-most position corresponds to the tile numbered 0. In the Bitmap for the last window, the bit at the right-most position corresponds either to the tile numbered 0 or to a tile that is sent/received as "the last one of the SCHC Packet" without explicitly stating its number (see Section 8.3.1.2).

At the receiver

- a bit set to 1 in the Bitmap indicates that a tile associated with that bit position has been correctly received for that window.

- a bit set to 0 in the Bitmap indicates that there has been no tile correctly received, associated with that bit position, for that window. Possible reasons include that the tile was not sent at all, not received, or received with errors.

8.2.2.4. Timers and counters

Some SCHC F/R modes can use the following timers and counters

- Inactivity Timer: a SCHC Fragment receiver uses this timer to abort waiting for a SCHC F/R message.

- Retransmission Timer: a SCHC Fragment sender uses this timer to abort waiting for an expected SCHC ACK.

- Attempts: this counter counts the requests for SCHC ACKs, up to MAX_ACK_REQUESTS.

8.2.3. Integrity Checking

The integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receive end. By default, integrity checking is performed by computing a Reassembly Check Sequence (RCS) based on the SCHC Packet at the sender side and transmitting it to the receiver for comparison with the RCS locally computed after reassembly.

The RCS supports UDP checksum elision by SCHC C/D (see Section 10.11).

The CRC32 polynomial 0xEDB88320 (i.e. the reversed polynomial representation, which is used e.g. in the Ethernet standard [ETHERNET]) is RECOMMENDED as the default algorithm for computing the RCS. Nevertheless, other RCS lengths or other algorithms MAY be required by the Profile.

The RCS MUST be computed on the full SCHC Packet concatenated with the padding bits, if any, of the SCHC Fragment carrying the last tile. The rationale is that the SCHC reassembler has no way of knowing the boundary between the last tile and the padding bits. Indeed, this requires decompressing the SCHC Packet, which is out of the scope of the SCHC reassembler.

Note that the concatenation of the complete SCHC Packet and any padding bits, if present, of the last SCHC Fragment does not generally constitute an integer number of bytes. For implementers to be able to use byte-oriented CRC libraries, it is RECOMMENDED that the concatenation of the complete SCHC Packet and any last fragment padding bits be zero-extended to the next byte boundary and that the RCS be computed on that byte array. A Profile MAY specify another behavior.

8.2.4. Header Fields

The SCHC F/R messages contain the following fields (see the formats in Section 8.3):

- Rule ID: this field is present in all the SCHC F/R messages. It is used to identify
  - that a SCHC F/R message is being carried, as opposed to an unfragmented SCHC Packet,
  - which SCHC F/R mode is used
  - in case this mode uses windows, what the value of WINDOW_SIZE is,
  - what other optional fields are present and what the field sizes are.

  The Rule ID tells apart a non-fragmented SCHC Packet from SCHC Fragments. It will also tell apart SCHC Fragments of fragmented SCHC Packets that use different SCHC F/R modes or different parameters. Interleaved transmission of these is therefore possible.

All SCHC F/R messages pertaining to the same SCHC Packet MUST bear the same Rule ID.

- Datagram Tag (DTag). This field allows differentiating SCHC F/R messages belonging to different SCHC Packets that may be using the same Rule ID simultaneously. Hence, it allows interleaving fragments of a new SCHC Packet with fragments of a previous SCHC Packet under the same Rule ID.

  The size of the DTag field (called T, in bits) is defined by each Profile for each Rule ID. When T is 0, the DTag field does not appear in the SCHC F/R messages and the DTag value is defined as 0.

  When T is 0, there can be no more than one fragmented SCHC Packet in transit for each fragmentation Rule ID.

  If T is not 0, DTag

  * MUST be set to the same value for all the SCHC F/R messages related to the same fragmented SCHC Packet,

  * MUST be set to different values for SCHC F/R messages related to different SCHC Packets that are being fragmented under the same Rule ID, and whose transmission may overlap.

- W: The W field is optional. It is only present if windows are used. Its presence and size (called M, in bits) is defined by each SCHC F/R mode and each Profile for each Rule ID.

  This field carries information pertaining to the window a SCHC F/R message relates to. If present, W MUST carry the same value for all the SCHC F/R messages related to the same window. Depending on the mode and Profile, W may carry the full window number, or just the least significant bit or any other partial representation of the window number.

- Fragment Compressed Number (FCN). The FCN field is present in the SCHC Fragment Header. Its size (called N, in bits) is defined by each Profile for each Rule ID.

  This field conveys information about the progress in the sequence of tiles being transmitted by SCHC Fragment messages. For example, it can contain a partial, efficient representation of a larger-sized tile index. The description of the exact use of the FCN field is left to each SCHC F/R mode. However, two values are reserved for special purposes. They help control the SCHC F/R process:

  * The FCN value with all the bits equal to 1 (called All-1) signals the very last tile of a SCHC Packet. By extension, if windows are used, the last window of a packet is called the All-1 window.

  * If windows are used, the FCN value with all the bits equal to 0 (called All-0) signals the last tile of a window that is not the last one of the SCHC packet. By extension, such a window is called an All-0 window.

- Reassembly Check Sequence (RCS). This field only appears in the All-1 SCHC Fragments. Its size (called U, in bits) is defined by each Profile for each Rule ID.

  See Section 8.2.3 for the RCS default size, default polynomial and details on RCS computation.

- C (integrity Check): C is a 1-bit field. This field is used in the SCHC ACK message to report on the reassembled SCHC Packet integrity check (see Section 8.2.3).

  A value of 1 tells that the integrity check was performed and is successful. A value of 0 tells that the integrity check was not performed, or that is was a failure.

- Compressed Bitmap. The Compressed Bitmap is used together with windows and Bitmaps (see Section 8.2.2.3). Its presence and size is defined for each F/R mode for each Rule ID.

  This field appears in the SCHC ACK message to report on the receiver Bitmap (see Section 8.3.2.1).

8.3. SCHC F/R Message Formats

This section defines the SCHC Fragment formats, the SCHC ACK format, the SCHC ACK REQ format and the SCHC Abort formats.

8.3.1. SCHC Fragment format

A SCHC Fragment conforms to the general format shown in Figure 12. It comprises a SCHC Fragment Header and a SCHC Fragment Payload. The SCHC Fragment Payload carries one or several tile(s).

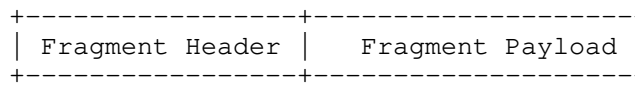


Figure 12: SCHC Fragment general format

8.3.1.1. Regular SCHC Fragment

The Regular SCHC Fragment format is shown in Figure 13. Regular SCHC Fragments are generally used to carry tiles that are not the last one of a SCHC Packet. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

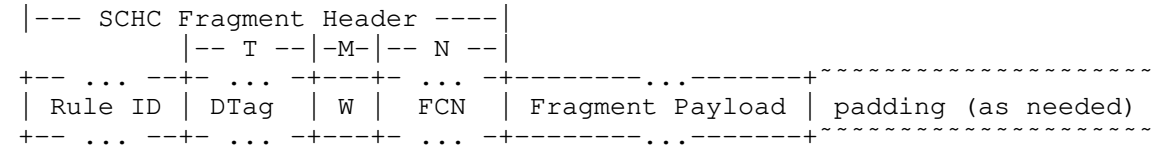


Figure 13: Detailed Header Format for Regular SCHC Fragments

The FCN field MUST NOT contain all bits set to 1.

The Fragment Payload of a SCHC Fragment with FCN equal to 0 (called an All-0 SCHC Fragment) MUST be distinguishable by size from a SCHC ACK REQ message (see Section 8.3.3) that has the same T, M and N values, even in the presence of padding. This condition is met if the Payload is at least the size of an L2 Word. This condition is also met if the SCHC Fragment Header is a multiple of L2 Words.

8.3.1.2. All-1 SCHC Fragment

The All-1 SCHC Fragment format is shown in Figure 14. The sender uses the All-1 SCHC Fragment format for the message that completes the emission of a fragmented SCHC Packet. The DTag field, the W field, the RCS field and the Payload are OPTIONAL, their presence is specified by each mode and Profile. At least one of RCS field or Payload MUST be present. The FCN field is all ones.

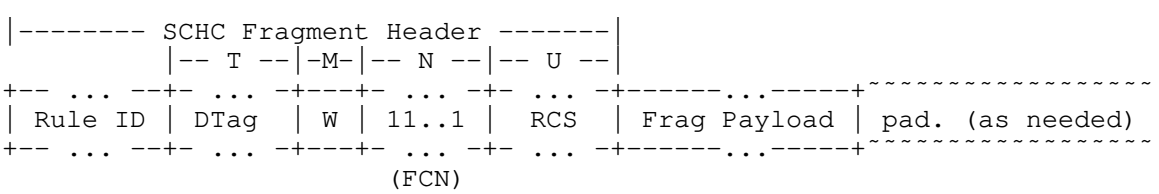


Figure 14: Detailed Header Format for the All-1 SCHC Fragment

The All-1 SCHC Fragment message MUST be distinguishable by size from a SCHC Sender-Abort message (see Section 8.3.4) that has the same T, M and N values, even in the presence of padding. This condition is met if the RCS is present and is at least the size of an L2 Word, or if the Payload is present and at least the size an L2 Word. This

condition is also met if the SCHC Sender-Abort Header is a multiple of L2 Words.

8.3.2. SCHC ACK format

The SCHC ACK message is shown in Figure 15. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The Compressed Bitmap field MUST be present in SCHC F/R modes that use windows, and MUST NOT be present in other modes.

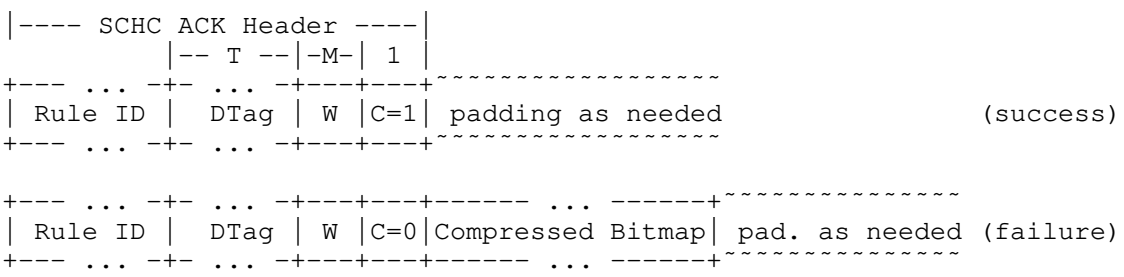


Figure 15: Format of the SCHC ACK message

The SCHC ACK Header contains a C bit (see Section 8.2.4).

If the C bit is set to 1 (integrity check successful), no Bitmap is carried.

If the C bit is set to 0 (integrity check not performed or failed) and if windows are used, a Compressed Bitmap for the window referred to by the W field is transmitted as specified in Section 8.3.2.1.

8.3.2.1. Bitmap Compression

For transmission, the Compressed Bitmap in the SCHC ACK message is defined by the following algorithm (see Figure 16 for a follow-along example):

- Build a temporary SCHC ACK message that contains the Header followed by the original Bitmap (see Section 8.2.2.3 for a description of Bitmaps).

- Position scissors at the end of the Bitmap, after its last bit.

- While the bit on the left of the scissors is 1 and belongs to the Bitmap, keep moving left, then stop. When this is done,

- While the scissors are not on an L2 Word boundary of the SCHC ACK message and there is a Bitmap bit on the right of the scissors, keep moving right, then stop.

- At this point, cut and drop off any bits to the right of the scissors

When one or more bits have effectively been dropped off as a result of the above algorithm, the SCHC ACK message is a multiple of L2 Words, no padding bits will be appended.

Because the SCHC Fragment sender knows the size of the original Bitmap, it can reconstruct the original Bitmap from the Compressed Bitmap received in the SCH ACK message.

Figure 16 shows an example where L2 Words are actually bytes and where the original Bitmap contains 17 bits, the last 15 of which are all set to 1.

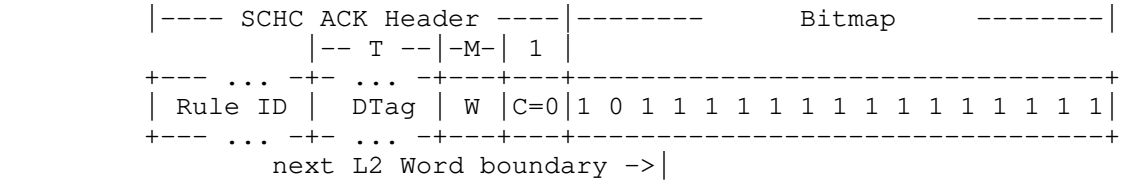


Figure 16: SCHC ACK Header plus uncompressed Bitmap

Figure 17 shows that the last 14 bits are not sent.

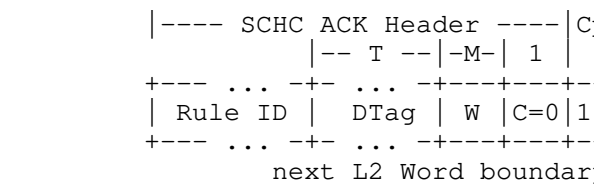


Figure 17: Resulting SCHC ACK message with Compressed Bitmap

Figure 18 shows an example of a SCHC ACK with tile indices ranging from 6 down to 0, where the Bitmap indicates that the second and the fourth tile of the window have not been correctly received.

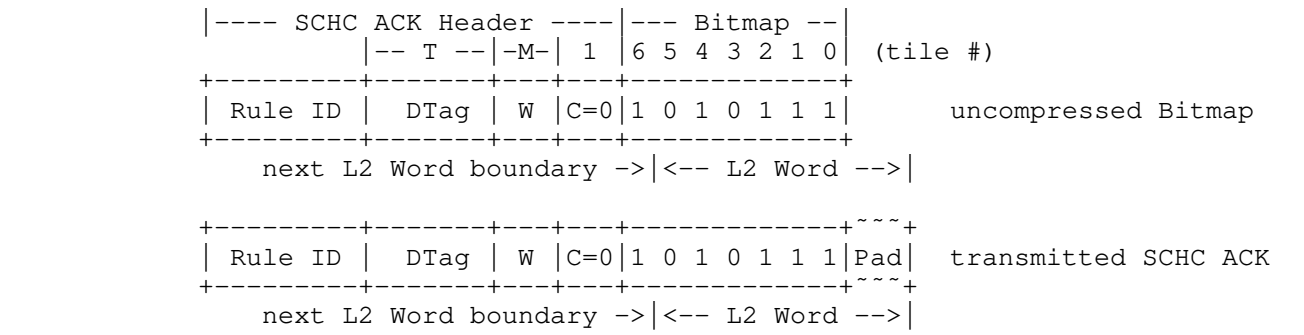


Figure 18: Example of a SCHC ACK message, missing tiles

Figure 19 shows an example of a SCHC ACK with FCN ranging from 6 down to 0, where integrity check has not been performed or has failed and the Bitmap indicates that there is no missing tile in that window.

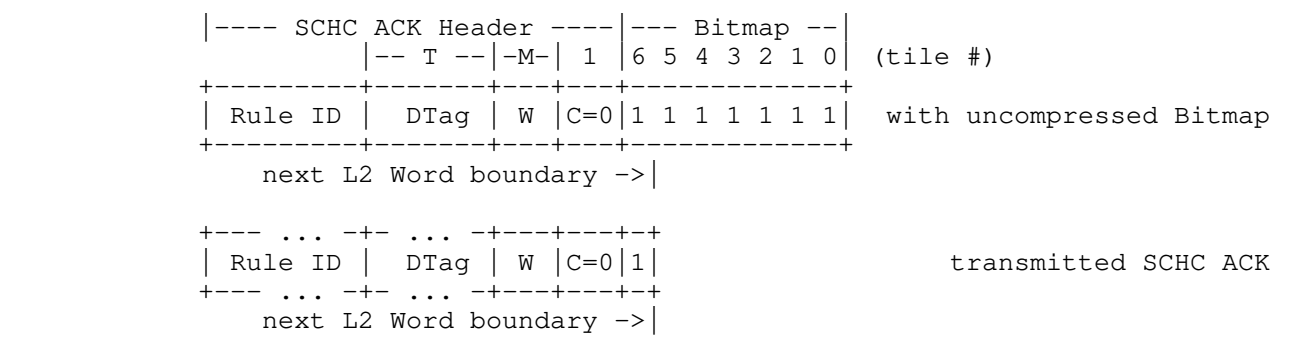


Figure 19: Example of a SCHC ACK message, no missing tile

8.3.3. SCHC ACK REQ format

The SCHC ACK REQ is used by a sender to request a SCHC ACK from the receiver. Its format is shown in Figure 20. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all zero.

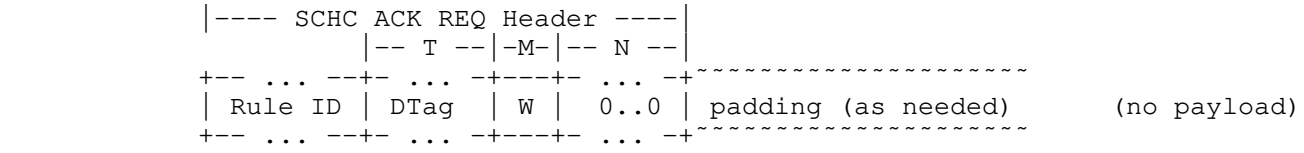


Figure 20: SCHC ACK REQ format

8.3.4. SCHC Sender-Abort format

When a SCHC Fragment sender needs to abort an on-going fragmented SCHC Packet transmission, it sends a SCHC Sender-Abort message to the SCHC Fragment receiver.

The SCHC Sender-Abort format is shown in Figure 21. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile. The FCN field is all ones.

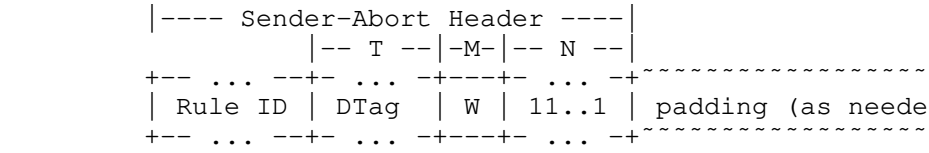


Figure 21: SCHC Sender-Abort format

If the W field is present,

- the fragment sender MUST set it to all ones. Other values are RESERVED.

- the fragment receiver MUST check its value. If the value is different from all ones, the message MUST be ignored.

The SCHC Sender-Abort MUST NOT be acknowledged.

8.3.5. SCHC Receiver-Abort format

When a SCHC Fragment receiver needs to abort an on-going fragmented SCHC Packet transmission, it transmits a SCHC Receiver-Abort message to the SCHC Fragment sender.

The SCHC Receiver-Abort format is shown in Figure 22. The DTag field and the W field are OPTIONAL, their presence is specified by each mode and Profile.

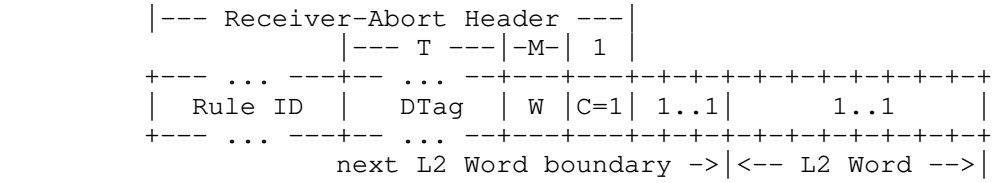


Figure 22: SCHC Receiver-Abort format

If the W field is present,

- the fragment receiver MUST set it to all ones. Other values are RESERVED.

- if the value is different from all ones, the fragment sender MUST ignore the message.

The SCHC Receiver-Abort has the same header as a SCHC ACK message. The bits that follow the SCHC Receiver-Abort Header MUST be as follows

- if the Header does not end at an L2 Word boundary, append bits set to 1 as needed to reach the next L2 Word boundary

- append exactly one more L2 Word with bits all set to ones

Such a bit pattern never occurs in a legit SCHC ACK. This is how the fragment sender recognizes a SCHC Receiver-Abort.

The SCHC Receiver-Abort MUST NOT be acknowledged.

8.4. SCHC F/R modes

This specification includes several SCHC F/R modes, which

- allow for a range of reliability options, such as optional SCHC Fragment retransmission

- support various LPWAN characteristics, such as links with variable MTU or unidirectional links.

More modes may be defined in the future.

Appendix B provides examples of fragmentation sessions based on the modes described hereafter.

Appendix C provides examples of Finite Sate Machines implementing the SCHC F/R modes decribed hereafter.

8.4.1. No-ACK mode

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. This mode supports LPWAN technologies that have a variable MTU.

In No-ACK mode, there is no communication from the fragment receiver to the fragment sender. The sender transmits all the SCHC Fragments

without expecting any acknowledgement. Therefore, No-ACK does not require bidirectional links: unidirectional links are just fine.

In No-ACK mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

The tile sizes are not required to be uniform. Windows are not used. The Retransmission Timer is not used. The Attempts counter is not used.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST NOT be present in the SCHC F/R messages. SCHC ACK MUST NOT be sent. SCHC ACK REQ MUST NOT be sent. SCHC Sender-Abort MAY be sent. SCHC Receiver-Abort MUST NOT be sent.

The value of N (size of the FCN field) is RECOMMENDED to be 1.

Each Profile, for each Rule ID value, MUST define

- the size of the DTag field,
- the size and algorithm for the RCS field,
- the expiration time of the Inactivity Timer

Each Profile, for each Rule ID value, MAY define

- a value of N different from the recommended one,
- the meaning of values sent in the FCN field, for values different from the All-1 value.

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer. If the receiver is under-resourced to do this, it MUST silently drop the related messages.

8.4.1.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. The tile MUST be at least the size of an L2 Word. The sender MUST transmit the SCHC Fragments messages in the order that the tiles appear in the SCHC Packet. Except for the last tile of a SCHC

Packet, each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits. Except for the last one, the SCHC Fragments MUST use the Regular SCHC Fragment format specified in Section 8.3.1.1. The last SCHC Fragment MUST use the All-1 format specified in Section 8.3.1.2.

The sender MAY transmit a SCHC Sender-Abort.

Figure 37 shows an example of a corresponding state machine.

8.4.1.2. Receiver behavior

Upon receiving each Regular SCHC Fragment,

- the receiver MUST reset the Inactivity Timer,
- the receiver assembles the payloads of the SCHC Fragments

On receiving an All-1 SCHC Fragment,

- the receiver MUST append the All-1 SCHC Fragment Payload and the padding bits to the previously received SCHC Fragment Payloads for this SCHC Packet
- the receiver MUST perform the integrity check
- if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet
- the reassembly operation concludes.

On expiration of the Inactivity Timer, the receiver MUST drop the SCHC Packet being reassembled.

On receiving a SCHC Sender-Abort, the receiver MAY drop the SCHC Packet being reassembled.

Figure 38 shows an example of a corresponding state machine.

8.4.2. ACK-Always mode

The ACK-Always mode has been designed under the following assumptions

- Data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly

- The L2 MTU value does not change while the fragments of a SCHC Packet are being transmitted.

- There is a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-Always mode on quasi-bidirectional links.

In ACK-Always mode, windows are used. An acknowledgement, positive or negative, is transmitted by the fragment receiver to the fragment sender at the end of the transmission of each window of SCHC Fragments.

The tiles are not required to be of uniform size. In ACK-Always mode, only the All-1 SCHC Fragment is padded as needed. The other SCHC Fragments are intrinsically aligned to L2 Words.

Briefly, the algorithm is as follows: after a first blind transmission of all the tiles of a window, the fragment sender iterates retransmitting the tiles that are reported missing until the fragment receiver reports that all the tiles belonging to the window have been correctly received, or until too many attempts were made. The fragment sender only advances to the next window of tiles when it has ascertained that all the tiles belonging to the current window have been fully and correctly received. This results in a per-window lock-step behavior between the sender and the receiver.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present and its size M MUST be 1 bit.

Each Profile, for each Rule ID value, MUST define

- the value of N (size of the FCN field),

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

- the size and algorithm for the RCS field,

- the size of the DTag field,

- the value of MAX_ACK_REQUESTS,

- the expiration time of the Retransmission Timer

- the expiration time of the Inactivity Timer

For each active pair of Rule ID and DTag values, the sender MUST maintain

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain

- one Inactivity Timer
- one Attempts counter

8.4.2.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet, the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet.

Each SCHC Fragment MUST contain exactly one tile in its Payload. All tiles with the index 0, as well as the last tile, MUST be at least the size of an L2 Word.

In all SCHC Fragment messages, the W field MUST be filled with the least significant bit of the window number that the sender is currently processing.

For a SCHC Fragment that carries a tile other than the last one of the SCHC Packet,

- the Fragment MUST be of the Regular type specified in Section 8.3.1.1
- the FCN field MUST contain the tile index
- each tile MUST be of a size that complements the SCHC Fragment Header so that the SCHC Fragment is a multiple of L2 Words without the need for padding bits.

The SCHC Fragment that carries the last tile MUST be an All-1 SCHC Fragment, described in Section 8.3.1.2.

The fragment sender MUST start by transmitting the window numbered 0.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

The sender starts by a "blind transmission" phase, in which it MUST transmit all the tiles composing the window, in decreasing tile index order.

Then, it enters a "retransmission phase" in which it MUST initialize an Attempts counter to 0, it MUST start a Retransmission Timer and it MUST await a SCHC ACK. Then,

- upon receiving a SCHC ACK,
  - if the SCHC ACK indicates that some tiles are missing at the receiver, then the sender MUST transmit all the tiles that have been reported missing, it MUST increment Attempts, it MUST reset the Retransmission Timer and MUST await the next SCHC ACK.
  - if the current window is not the last one and the SCHC ACK indicates that all tiles were correctly received, the sender MUST stop the Retransmission Timer, it MUST advance to the next fragmentation window and it MUST start a blind transmission phase as described above.
  - if the current window is the last one and the SCHC ACK indicates that more tiles were received than the sender sent, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - if the current window is the last one and the SCHC ACK indicates that all tiles were correctly received yet integrity check was a failure, the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.
  - if the current window is the last one and the SCHC ACK indicates that integrity checking was successful, the sender exits successfully.
- on Retransmission Timer expiration,
  - if Attempts is strictly less that MAX_ACK_REQUESTS, the fragment sender MUST send a SCHC ACK REQ and MUST increment the Attempts counter.
  - otherwise the fragment sender MUST send a SCHC Sender-Abort, and it MAY exit with an error condition.

At any time,

- on receiving a SCHC Receiver-Abort, the fragment sender MAY exit with an error condition.

- on receiving a SCHC ACK that bears a W value different from the W value that it currently uses, the fragment sender MUST silently discard and ignore that SCHC ACK.

Figure 39 shows an example of a corresponding state machine.

8.4.2.2. Receiver behavior

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

- the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

- the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair.

- the receiver MUST start an Inactivity Timer for that RuleID and DTag pair. It MUST initialize an Attempts counter to 0 for that RuleID and DTag pair. It MUST initialize a window counter to 0. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver Abort.

In the rest of this section, "local W bit" means the least significant bit of the window counter of the receiver.

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

The receiver MUST first initialize an empty Bitmap for the first window, then enter an "acceptance phase", in which

- on receiving a SCHC Fragment or a SCHC ACK REQ, either one having the W bit different from the local W bit, the receiver MUST silently ignore and discard that message.

- on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

- on receiving a SCHC Fragment with the W bit equal to the local W bit, the receiver MUST assemble the received tile based on the window counter and on the FCN field in the SCHC Fragment and it MUST update the Bitmap.

  * if the SCHC Fragment received is an All-0 SCHC Fragment, the current window is determined to be a not-last window, the receiver MUST send a SCHC ACK for this window and it MUST enter the "retransmission phase" for this window.

  * if the SCHC Fragment received is an All-1 SCHC Fragment, the padding bits of the All-1 SCHC Fragment MUST be assembled after the received tile, the current window is determined to be the last window, the receiver MUST perform the integrity check and it MUST send a SCHC ACK for this window. Then,

    + If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for this window.

    + If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for this window.

In the "retransmission phase":

- if the window is a not-last window

  * on receiving a SCHC Fragment that is not All-0 or All-1 and that has a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap and it MUST enter the "acceptance phase" for that new window.

  * on receiving a SCHC ACK REQ with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST send a SCHC ACK for that new window and it MUST enter the "acceptance phase" for that new window.

  * on receiving a SCHC All-0 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received and update the Bitmap, it MUST send a SCHC ACK for that new

window and it MUST stay in the "retransmission phase" for that new window.

* on receiving a SCHC All-1 Fragment with a W bit different from the local W bit, the receiver MUST increment its window counter and allocate a fresh Bitmap, it MUST assemble the tile received, including the padding bits, it MUST update the Bitmap and perform the integrity check, it MUST send a SCHC ACK for the new window, which is determined to be the last window. Then,

  + If the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST enter the "clean-up phase" for that new window.

  + If the integrity check indicates that the full SCHC Packet has not been correctly reassembled, the receiver enters the "retransmission phase" for that new window.

* on receiving a SCHC Fragment with a W bit equal to the local W bit,

  + if the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST silently ignore it and discard it.

  + otherwise, the receiver MUST assemble the tile received and update the Bitmap. If the Bitmap becomes fully populated with 1's or if the SCHC Fragment is an All-0, the receiver MUST send a SCHC ACK for this window.

* on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

o if the window is the last window

* on receiving a SCHC Fragment or a SCHC ACK, either one having a W bit different from the local W bit, the receiver MUST silently ignore and discard that message.

* on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver MUST send a SCHC ACK for this window.

* on receiving a SCHC Fragment with a W bit equal to the local W bit,

  + if the SCHC Fragment received is an All-0 SCHC Fragment, the receiver MUST silently ignore it and discard it.

+ otherwise, the receiver MUST update the Bitmap and it MUST assemble the tile received. If the SCHC Fragment received is an All-1 SCHC Fragment, the receiver MUST assemble the padding bits of the All-1 SCHC Fragment after the received tile, it MUST perform the integrity check and

  - if the integrity check indicates that the full SCHC Packet has been correctly reassembled, the receiver MUST send a SCHC ACK and it enters the "clean-up phase".

  - if the integrity check indicates that the full SCHC Packet has not been correctly reassembled,

    o if the SCHC Fragment received was an All-1 SCHC Fragment, the receiver MUST send a SCHC ACK for this window.

In the "clean-up phase":

- On receiving an All-1 SCHC Fragment or a SCHC ACK REQ, either one having the W bit equal to the local W bit, the receiver MUST send a SCHC ACK.

- Any other SCHC Fragment received MUST be silently ignored and discarded.

At any time, on expiration of the Inactivity Timer, on receiving a SCHC Sender-Abort or when Attempts reaches MAX_ACK_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit the receive process for that SCHC Packet.

Figure 40 shows an example of a corresponding state machine.

8.4.3. ACK-on-Error mode

The ACK-on-Error mode supports LPWAN technologies that have variable MTU and out-of-order delivery. It operates with links that provide a feedback path from the reassembler to the fragmenter. See Appendix F for a discussion on using ACK-on-Error mode on quasi-bidirectional links.

In ACK-on-Error mode, windows are used.

All tiles, but the last one and the penultimate one, MUST be of equal size, hereafter called "regular". The size of the last tile MUST be smaller than or equal to the regular tile size. Regarding the penultimate tile, a Profile MUST pick one of the following two options:

- The penultimate tile size MUST be the regular tile size
- or the penultimate tile size MUST be either the regular tile size or the regular tile size minus one L2 Word.

A SCHC Fragment message carries one or several contiguous tiles, which may span multiple windows. A SCHC ACK reports on the reception of exactly one window of tiles.

See Figure 23 for an example.

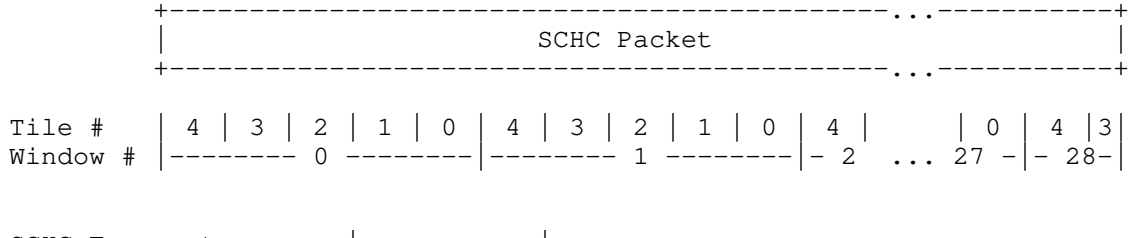


Figure 23: a SCHC Packet fragmented in tiles, ACK-on-Error mode

The W field is wide enough that it unambiguously represents an absolute window number. The fragment receiver sends SCHC ACKs to the fragment sender about windows for which tiles are missing. No SCHC ACK is sent by the fragment receiver for windows that it knows have been fully received.

The fragment sender retransmits SCHC Fragments for tiles that are reported missing. It can advance to next windows even before it has ascertained that all tiles belonging to previous windows have been correctly received, and can still later retransmit SCHC Fragments with tiles belonging to previous windows. Therefore, the sender and the receiver may operate in a decoupled fashion. The fragmented SCHC Packet transmission concludes when

- integrity checking shows that the fragmented SCHC Packet has been correctly reassembled at the receive end, and this information has been conveyed back to the sender,
- or too many retransmission attempts were made,
- or the receiver determines that the transmission of this fragmented SCHC Packet has been inactive for too long.

Each Profile MUST specify which Rule ID value(s) correspond to SCHC F/R messages operating in this mode.

The W field MUST be present in the SCHC F/R messages.

Each Profile, for each Rule ID value, MUST define

- the tile size (a tile does not need to be multiple of an L2 Word, but it MUST be at least the size of an L2 Word)
- the value of M (size of the W field),
- the value of N (size of the FCN field),
- the value of WINDOW_SIZE, which MUST be strictly less than 2^N,
- the size and algorithm for the RCS field,
- the size of the DTag field,
- the value of MAX_ACK_REQUESTS,
- the expiration time of the Retransmission Timer
- the expiration time of the Inactivity Timer
- if the last tile is carried in a Regular SCHC Fragment or an All-1 SCHC Fragment (see Section 8.4.3.1)
- if the penultimate tile MAY be one L2 Word smaller than the regular tile size. In this case, the regular tile size MUST be at least twice the L2 Word size.

For each active pair of Rule ID and DTag values, the sender MUST maintain

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain

- one Inactivity Timer
- one Attempts counter

8.4.3.1. Sender behavior

At the beginning of the fragmentation of a new SCHC Packet,

- the fragment sender MUST select a Rule ID and DTag value pair for this SCHC Packet. A Rule MUST NOT be selected if the values of M and WINDOW_SIZE for that Rule are such that the SCHC Packet cannot be fragmented in (2^M) * WINDOW_SIZE tiles or less.

- the fragment sender MUST initialize the Attempts counter to 0 for that Rule ID and DTag value pair.

A Regular SCHC Fragment message carries in its payload one or more tiles. If more than one tile is carried in one Regular SCHC Fragment

- the selected tiles MUST be contiguous in the original SCHC Packet

- they MUST be placed in the SCHC Fragment Payload adjacent to one another, in the order they appear in the SCHC Packet, from the start of the SCHC Packet toward its end.

Tiles that are not the last one MUST be sent in Regular SCHC Fragments specified in Section 8.3.1.1. The FCN field MUST contain the tile index of the first tile sent in that SCHC Fragment.

In a Regular SCHC Fragment message, the sender MUST fill the W field with the window number of the first tile sent in that SCHC Fragment.

Depending on the Profile, the last tile of a SCHC Packet MUST be sent either

- in a Regular SCHC Fragment, alone or as part of a multi-tiles Payload

- alone in an All-1 SCHC Fragment

In an All-1 SCHC Fragment message, the sender MUST fill the W field with the window number of the last tile of the SCHC Packet.

The fragment sender MUST send SCHC Fragments such that, all together, they contain all the tiles of the fragmented SCHC Packet.

The fragment sender MUST send at least one All-1 SCHC Fragment.

The fragment sender MUST listen for SCHC ACK messages after having sent

- an All-1 SCHC Fragment

- or a SCHC ACK REQ.

A Profile MAY specify other times at which the fragment sender MUST listen for SCHC ACK messages. For example, this could be after sending a complete window of tiles.

Each time a fragment sender sends an All-1 SCHC Fragment or a SCHC ACK REQ,

- it MUST increment the Attempts counter
- it MUST reset the Retransmission Timer

On Retransmission Timer expiration

- if Attempts is strictly less than MAX_ACK_REQUESTS, the fragment sender MUST send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window,
- otherwise the fragment sender MUST send a SCHC Sender-Abort and it MAY exit with an error condition.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

On receiving a SCHC ACK,

- if the W field in the SCHC ACK corresponds to the last window of the SCHC Packet,
  - if the C bit is set, the sender MAY exit successfully
  - otherwise,
    - if the Profile mandates that the last tile be sent in an All-1 SCHC Fragment,
      - if the SCHC ACK shows no missing tile at the receiver, the sender
        - MUST send a SCHC Sender-Abort
        - MAY exit with an error condition
      - otherwise

- the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

- if the last message in this sequence of SCHC Fragment messages is not an All-1 SCHC Fragment, then the fragment sender MUST in addition send a SCHC ACK REQ with the W field corresponding to the last window, after the sequence.

+ otherwise,

- if the SCHC ACK shows no missing tile at the receiver, the sender MUST send the All-1 SCHC Fragment

- otherwise

- the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC ACK.

- the fragment sender MUST then send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field corresponding to the last window.

- otherwise, the fragment sender

* MUST send SCHC Fragment messages containing the tiles that are reported missing in the SCHC ACK

* then it MAY send a SCHC ACK REQ with the W field corresponding to the last window

See Figure 41 for one among several possible examples of a Finite State Machine implementing a sender behavior obeying this specification.

8.4.3.2. Receiver behavior

On receiving a SCHC Fragment with a Rule ID and DTag pair not being processed at that time

- the receiver SHOULD check if the DTag value has not recently been used for that Rule ID value, thereby ensuring that the received SCHC Fragment is not a remnant of a prior fragmented SCHC Packet transmission. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

- the receiver MUST start a process to assemble a new SCHC Packet with that Rule ID and DTag value pair. The receiver MUST start an Inactivity Timer for that Rule ID and DTag value pair. It MUST initialize an Attempts counter to 0 for that Rule ID and DTag value pair. If the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver Abort.

On reception of any SCHC F/R message for the RuleID and DTag pair being processed, the receiver MUST reset the Inactivity Timer pertaining to that RuleID and DTag pair.

All message receptions being discussed in the rest of this section are to be understood as "matching the RuleID and DTag pair being processed", even if not spelled out, for brevity.

On receiving a SCHC Fragment message, the receiver determines what tiles were received, based on the payload length and on the W and FCN fields of the SCHC Fragment.

- if the FCN is All-1, if a Payload is present, the full SCHC Fragment Payload MUST be assembled including the padding bits. This is because the size of the last tile is not known by the receiver, therefore padding bits are indistinguishable from the tile data bits, at this stage. They will be removed by the SCHC C/D sublayer. If the size of the SCHC Fragment Payload exceeds or equals the size of one regular tile plus the size of an L2 Word, this SHOULD raise an error flag.

- otherwise, tiles MUST be assembled based on the a priori known tile size.

  * If allowed by the Profile, the end of the payload MAY contain the last tile, which may be shorter. Padding bits are indistinguishable from the tile data bits, at this stage.

  * the payload may contain the penultimate tile that, if allowed by the Profile, MAY be exactly one L2 Word shorter than the regular tile size.

  * Otherwise, padding bits MUST be discarded. The latter is possible because

    + the size of the tiles is known a priori,

    + tiles are larger than an L2 Word

    + padding bits are always strictly less than an L2 Word

On receiving a SCHC ACK REQ or an All-1 SCHC Fragment,

- if the receiver has at least one window that it knows has tiles missing, it MUST return a SCHC ACK for the lowest-numbered such window,

- otherwise,

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

  * otherwise it MUST return a SCHC ACK for window numbered 0

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

Upon sending a SCHC ACK, the receiver MUST increase the Attempts counter.

After receiving an All-1 SCHC Fragment, a receiver MUST check the integrity of the reassembled SCHC Packet at least every time it prepares for sending a SCHC ACK for the last window.

Upon receiving a SCHC Sender-Abort, the receiver MAY exit with an error condition.

Upon expiration of the Inactivity Timer, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

On the Attempts counter exceeding MAX_ACK_REQUESTS, the receiver MUST send a SCHC Receiver-Abort and it MAY exit with an error condition.

Reassembly of the SCHC Packet concludes when

- a Sender-Abort has been received

- or the Inactivity Timer has expired

- or the Attempts counter has exceeded MAX_ACK_REQUESTS

- or when at least an All-1 SCHC Fragment has been received and integrity checking of the reassembled SCHC Packet is successful.

See Figure 42 for one among several possible examples of a Finite State Machine implementing a receiver behavior obeying this specification, and that is meant to match the sender Finite State Machine of Figure 41.

9. Padding management

SCHC C/D and SCHC F/R operate on bits, not bytes. SCHC itself does not have any alignment prerequisite. The size of SCHC Packets can be any number of bits.

If the layer below SCHC constrains the payload to align to some boundary, called L2 Words (for example, bytes), the SCHC messages MUST be padded. When padding occurs, the number of appended bits MUST be strictly less than the L2 Word size.

If a SCHC Packet is sent unfragmented (see Figure 24), it is padded as needed for transmission.

If a SCHC Packet needs to be fragmented for transmission, it is not padded in itself. Only the SCHC F/R messages are padded as needed for transmission. Some SCHC F/R messages are intrinsically aligned to L2 Words.

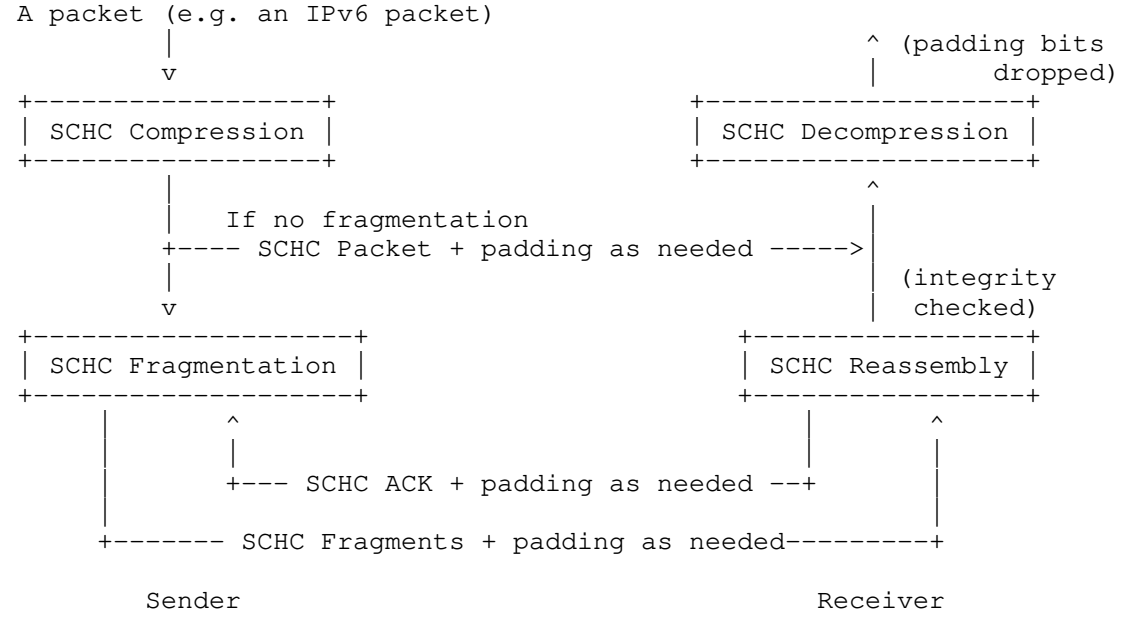


Figure 24: SCHC operations, including padding as needed

Each Profile MUST specify the size of the L2 Word. The L2 Word might actually be a single bit, in which case no padding will take place at all.

A Profile MAY define the value of the padding bits. The RECOMMENDED value is 0.

10. SCHC Compression for IPv6 and UDP headers

This section lists the IPv6 and UDP header fields and describes how they can be compressed. An example of a set of Rules for UDP/IPv6 header compression is provided in Appendix A.

10.1. IPv6 version field

The IPv6 version field is labeled by the protocol parser as being the "version" field of the IPv6 protocol. Therefore, it only exists for IPv6 packets. In the Rule, TV is set to 6, MO to "ignore" and CDA to "not-sent".

10.2. IPv6 Traffic class field

If the DiffServ field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this well-known value, an "equal" MO and a "not-sent" CDA.

Otherwise (e.g. ECN bits are to be transmitted), two possibilities can be considered depending on the variability of the value:

- One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".

- If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

  ECN functionality depends on both bits of the ECN field, which are the 2 LSBs of this field, hence sending only a single LSB of this field is NOT RECOMMENDED.

10.3. Flow label field

If the flow label is not set, i.e. its value is zero, the Field Descriptor in the Rule SHOULD contain a TV set to zero, an "equal" MO and a "not-sent" CDA.

If the flow label is set to a pseudo-random value according to [RFC6437], in the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".

If the flow label is set according to some prior agreement, i.e. by a flow state establishment method as allowed by [RFC6437], the Field Descriptor in the Rule SHOULD contain a TV with this agreed-upon value, an "equal" MO and a "not-sent" CDA.

10.4. Payload Length field

This field can be elided for the transmission on the LPWAN network. The SCHC C/D recomputes the original payload length value. In the Field Descriptor, TV is not set, MO is set to "ignore" and CDA is "compute-*".

10.5. Next Header field

If the Next Header field does not vary and is known by both sides, the Field Descriptor in the Rule SHOULD contain a TV with this Next Header value, the MO SHOULD be "equal" and the CDA SHOULD be "not-sent".

Otherwise, TV is not set in the Field Descriptor, MO is set to "ignore" and CDA is set to "value-sent". Alternatively, a matching-list MAY also be used.

10.6. Hop Limit field

The field behavior for this field is different for uplink (Up) and downlink (Dw). In Up, since there is no IP forwarding between the Dev and the SCHC C/D, the value is relatively constant. On the other hand, the Dw value depends on Internet routing and can change more frequently. The Direction Indicator (DI) can be used to distinguish both directions:

- in the Up, elide the field: the TV in the Field Descriptor is set to the known constant value, the MO is set to "equal" and the CDA is set to "not-sent".

- in the Dw, the Hop Limit is elided for transmission and forced to 1 at the receiver, by setting TV to 1, MO to "ignore" and CDA to "not-sent". This prevents any further forwarding.

10.7. IPv6 addresses fields

As in 6LoWPAN [RFC4944], IPv6 addresses are split into two 64-bit long fields; one for the prefix and one for the Interface Identifier (IID). These fields SHOULD be compressed. To allow for a single Rule being used for both directions, these values are identified by their role (Dev or App) and not by their position in the header (source or destination).

10.7.1. IPv6 source and destination prefixes

Both ends MUST be configured with the appropriate prefixes. For a specific flow, the source and destination prefixes can be unique and stored in the Context. In that case, the TV for the source and destination prefixes contain the values, the MO is set to "equal" and the CDA is set to "not-sent".

If the Rule is intended to compress packets with different prefix values, match-mapping SHOULD be used. The different prefixes are listed in the TV, the MO is set to "match-mapping" and the CDA is set to "mapping-sent". See Figure 26.

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

10.7.2. IPv6 source and destination IID

If the Dev or App IID are based on an LPWAN address, then the IID can be reconstructed with information coming from the LPWAN header. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "DevIID" or "AppIID". On LPWAN technologies where the frames carry a single identifier (corresponding to the Dev.), AppIID cannot be used.

As described in [RFC8065], it may be undesirable to build the Dev IPv6 IID out of the Dev address. Another static value is used instead. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDA is set to "not-sent".

If several IIDs are possible, then the TV contains the list of possible IIDs, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

It may also happen that the IID variability only expresses itself on a few bytes. In that case, the TV is set to the stable part of the IID, the MO is set to "MSB" and the CDA is set to "LSB".

Finally, the IID can be sent in its entirety on the LPWAN. In that case, the TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

10.8. IPv6 extension headers

This document does not provide recommendations on how to compress IPv6 extension headers.

10.9. UDP source and destination ports

To allow for a single Rule being used for both directions, the UDP port values are identified by their role (Dev or App) and not by their position in the header (source or destination). The SCHC C/D MUST be aware of the traffic direction (Uplink, Downlink) to select the appropriate field. The following Rules apply for Dev and App port numbers.

If both ends know the port number, it can be elided. The TV contains the port number, the MO is set to "equal" and the CDA is set to "not-sent".

If the port variation is on few bits, the TV contains the stable part of the port number, the MO is set to "MSB" and the CDA is set to "LSB".

If some well-known values are used, the TV can contain the list of these values, the MO is set to "match-mapping" and the CDA is set to "mapping-sent".

Otherwise the port numbers are sent over the LPWAN. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

10.10. UDP length field

The UDP length can be computed from the received data. The TV is not set, the MO is set to "ignore" and the CDA is set to "compute-*".

10.11. UDP Checksum field

The UDP checksum operation is mandatory with IPv6 for most packets but there are exceptions [RFC8200].

For instance, protocols that use UDP as a tunnel encapsulation may enable zero-checksum mode for a specific port (or set of ports) for sending and/or receiving. [RFC8200] requires any node implementing zero-checksum mode to follow the requirements specified in "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums" [RFC6936].

6LoWPAN Header Compression [RFC6282] also specifies that a UDP checksum can be elided by the compressor and re-computed by the decompressor when an upper layer guarantees the integrity of the UDP payload and pseudo-header. A specific example of this is when a Message Integrity Check protects the compressed message between the compressor that elides the UDP checksum and the decompressor that

computes it, with a strength that is identical or better to the UDP checksum.

Similarly, a SCHC compressor MAY elide the UDP checksum when another layer guarantees at least equal integrity protection for the UDP payload and the pseudo-header. In this case, the TV is not set, the MO is set to "ignore" and the CDA is set to "compute-*".

In particular, when SCHC fragmentation is used, a fragmentation RCS of 2 bytes or more provides equal or better protection than the UDP checksum; in that case, if the compressor is collocated with the fragmentation point and the decompressor is collocated with the packet reassembly point, and if the SCHC Packet is fragmented even when it would fit unfragmented in the L2 MTU, then the compressor MAY verify and then elide the UDP checksum. Whether and when the UDP Checksum is elided is to be specified in the Profile.

Since the compression happens before the fragmentation, implementors should understand the risks when dealing with unprotected data below the transport layer and take special care when manipulating that data.

In other cases, the checksum SHOULD be explicitly sent. The TV is not set, the MO is set to "ignore" and the CDA is set to "value-sent".

11. IANA Considerations

This document has no request to IANA.

12. Security considerations

As explained in Section 5, SCHC is expected to be implemented on top of LPWAN technologies, which are expected to implement security measures.

In this section, we analyze the potential security threats that could be introduced into an LPWAN by adding the SCHC functionalities.

12.1. Security considerations for SCHC Compression/Decompression

12.1.1. Forged SCHC Packet

Let’s assume that an attacker is able to send a forged SCHC Packet to a SCHC Decompressor.

Let’s first consider the case where the Rule ID contained in that forged SCHC Packet does not correspond to a Rule allocated in the

Rule table. An implementation should detect that the Rule ID is invalid and should silently drop the offending SCHC Packet.

Let's now consider that the Rule ID corresponds to a Rule in the table. With the CDAs defined in this document, the reconstructed packet is at most a constant number of bits bigger than the SCHC Packet that was received. This assumes that the compute-* decompression actions produce a bounded number of bits, irrespective of the incoming SCHC Packet. This property is true for IPv6 Length, UDP Length and UDP Checksum, for which the compute-* CDA is recommended by this document.

As a consequence, SCHC Decompression does not amplify attacks, beyond adding a bounded number of bits to the SCHC Packet received. This bound is determined by the Rule stored in the receiving device.

As a general safety measure, a SCHC Decompressor should never reconstruct a packet larger than MAX_PACKET_SIZE (defined in a Profile, with 1500 bytes as generic default).

12.1.2. Compressed packet size as a side channel to guess a secret token

Some packet compression methods are known to be victims of attacks, such as BREACH and CRIME. The attack involves injecting arbitrary data into the packet and observing the resulting compresssed packet size. The observed size potentially reflects correlation between the arbitrary data and some content that was meant to remain secret, such as a security token, thereby allowing the attacker to get at the secret.

By contrast, SCHC Compression takes place header field by header field, with the SCHC Packet being a mere concatenation of the compression residues of each of the individual field. Any correlation between header fields does not result in a change in the SCHC Packet size compressed under the same Rule.

If SCHC C/D is used to compress packets that include a secret information field, such as a token, the Rule set should be designed so that the size of the compression residue for the field to remain secret is the same irrespective of the value of the secret information. This is achieved by e.g. sending this field in extenso with the "ignore" MO and the "value-sent" CDA. This recommendation is disputable if it is ascertained that the Rule set itself will remain secret.

12.1.3. decompressed packet different from the original packet

The attention of Rule designers is drawn to situation As explained in Section 7.3, using FPs with value 0 in Field Descriptors in a Rule may result in header fields appearing in the decompressed packet in an order different from that in the original packet. Likewise, as stated in Section 7.5.3, using an "ignore" MO together with a "not-sent" CDA will result in the header field taking the TV value, which is likely to be different from the original value.

Depending on the protocol, the order of header fields in the packet may be functionally significant or not.

Furthermore, if the packet is protected by a checksum or a similar integrity protection mechanism, and if the checksum is transmitted instead of being recomputed as part of the decompression, these situations may result in the packet being considered corrupt and dropped.

12.2. Security considerations for SCHC Fragmentation/Reassembly

12.2.1. Buffer reservation attack

Let’s assume that an attacker is able to send a forged SCHC Fragment to a SCHC Reassembler.

A node can perform a buffer reservation attack: the receiver will reserve buffer space for the SCHC Packet. If the implementation has only one buffer, other incoming fragmented SCHC Packets will be dropped while the reassembly buffer is occupied during the reassembly timeout. Once that timeout expires, the attacker can repeat the same procedure, and iterate, thus creating a denial of service attack. An implementation may have multiple reassembly buffers. The cost to mount this attack is linear with the number of buffers at the target node. Better, the cost for an attacker can be increased if individual fragments of multiple SCHC Packets can be stored in the reassembly buffer. The finer grained the reassembly buffer (downto the smallest tile size), the higher the cost of the attack. If buffer overload does occur, a smart receiver could selectively discard SCHC Packets being reassembled based on the sender behavior, which may help identify which SCHC Fragments have been sent by the attacker. Another mild counter-measure is for the target to abort the fragmentation/reassembly session as early as it detects a non-identical SCHC Fragment duplicate, anticipating for an eventual corrupt SCHC Packet, so as to save the sender the hassle of sending the rest of the fragments for this SCHC Packet.

12.2.2. Corrupt Fragment attack

Let's assume that an attacker is able to send a forged SCHC Fragment to a SCHC Reassembler. The malicious node is additionally assumed to be able to hear an incoming communication destined to the target node.

It can then send a forged SCHC Fragment that looks like it belongs to a SCHC Packet already being reassembled at the target node. This can cause the SCHC Packet to be considered corrupt and be dropped by the receiver. The amplification happens here by a single spoofed SCHC Fragment rendering a full sequence of legit SCHC Fragments useless. If the target uses ACK-Always or ACK-on-Error mode, such a malicious node can also interfere with the acknowledgement and repetition algorithm of SCHC F/R. A single spoofed ACK, with all bitmap bits set to 0, will trigger the repetition of WINDOW_SIZE tiles. This protocol loop amplification depletes the energy source of the target node and consumes the channel bandwidth. Similarly, a spoofed ACK REQ will trigger the sending of a SCHC ACK, which may be much larger than the ACK REQ if WINDOW_SIZE is large. These consequences should be borne in mind when defining profiles for SCHC over specific LPWAN technologies.

12.2.3. Fragmentation as a way to bypass Network Inspection

Fragmentation is known for potentially allowing to force through a Network Inspection device (e.g. firewall) packets that would be rejected if unfragmented. This involves sending overlapping fragments to rewrite fields whose initial value led the Network Inspection device to allow the flow go through.

SCHC F/R is expected to be used over one LPWAN link, where no Network Inspection device is expected to sit. As described in Section 5.2, even if the SCHC F/R on the Network infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW.

12.2.4. Privacy issues associated with SCHC header fields

SCHC F/R allocates a DTag value to fragments belonging to the same SCHC Packet. Concerns were raised that, if DTag is a wide counter that is incremented in a predictible fashion for each new fragmented SCHC Packet, it might lead to a privacy issue, such as enabling tracking of a device across LPWANs.

However, SCHC F/R is expected to be used over exactly one LPWAN link. As described in Section 5.2, even if the SCHC F/R on the Network infrastructure side is located in the Internet, a tunnel is to be established between it and the NGW. Therefore, neither the DTag

field nor any other SCHC-introduced field is visible over the Internet.

13. Acknowledgements

Thanks to Sergio Aguilar Romero, Brian Carpenter, Carsten Bormann, David Black, Deborah Brungard, Philippe Clavier, Alissa Cooper, Roman Danyliw, Daniel Ducuara Beltran, Diego Dujovne, Eduardo Ingles Sanchez, Benjamin Kaduk, Arunprabhu Kandasamy, Suresh Krishnan, Mirja Kuehlewind, Rahul Jadhav, Barry Leiba, Sergio Lopez Bernal, Antony Markovski, Alexey Melnikov, Alexander Pelov, Charles Perkins, Edgar Ramos, Alvaro Retana, Adam Roach, Shoichi Sakane, Joseph Salowey, Pascal Thubert, and Eric Vyncke for useful design considerations, reviews and comments.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CAS15/00336, and by the ERDF and the Spanish Government through project TEC2016-79988-P. Part of his contribution to this work has been carried out during his stay as a visiting scholar at the Computer Laboratory of the University of Cambridge.

14. References

14.1. Normative References

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

[RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums", RFC 6936, DOI 10.17487/RFC6936, April 2013, <https://www.rfc-editor.org/info/rfc6936>.

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[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>.

14.2. Informative References

[ETHERNET]
"IEEE Standard for Ethernet", IEEE standard, DOI 10.1109/ieeestd.2018.8457469, n.d..

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

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

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.

[RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6 Interface Identifiers", RFC 7136, DOI 10.17487/RFC7136, February 2014, <https://www.rfc-editor.org/info/rfc7136>.

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

Appendix A. Compression Examples

This section gives some scenarios of the compression mechanism for IPv6/UDP. The goal is to illustrate the behavior of SCHC.

The mechanisms defined in this document can be applied to a Dev that embeds some applications running over CoAP. In this example, three flows are considered. The first flow is for the device management based on CoAP using Link Local IPv6 addresses and UDP ports 123 and 124 for Dev and App, respectively. The second flow will be a CoAP server for measurements done by the Dev (using ports 5683) and Global IPv6 Address prefixes alpha::IID/64 to beta::1/64. The last flow is for legacy applications using different ports numbers, the destination IPv6 address prefix is gamma::1/64.

Figure 25 presents the protocol stack. IPv6 and UDP are represented with dotted lines since these protocols are compressed on the radio link.

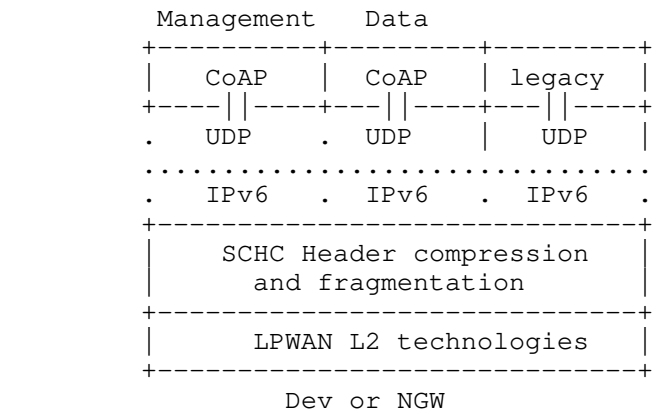


Figure 25: Simplified Protocol Stack for LP-WAN

In some LPWAN technologies, only the Devs have a device ID. When such technologies are used, it is necessary to statically define an IID for the Link Local address for the SCHC C/D.

Rule 0

Field	FL	FP	DI	Value	Match Opera.	Comp Decomp Action	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 DiffServ	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent	
IPv6 DevPrefix	64	1	Bi	FE80::/64	equal	not-sent	
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	FE80::/64	equal	not-sent	
IPv6 AppIID	64	1	Bi	::1	equal	not-sent	
UDP DevPort	16	1	Bi	123	equal	not-sent	
UDP AppPort	16	1	Bi	124	equal	not-sent	
UDP Length	16	1	Bi		ignore	compute-*	
UDP checksum	16	1	Bi		ignore	compute-*	

Rule 1

Field	FL	FP	DI	Value	Match Opera.	Action Action	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 DiffServ	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent	
IPv6 DevPrefix	64	1	Bi	[alpha/64, fe80::/64]	match-mapping	mapping-sent	1
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	[beta/64, alpha/64, fe80::64]	match-mapping	mapping-sent	2
IPv6 AppIID	64	1	Bi	::1000	equal	not-sent	
UDP DevPort	16	1	Bi	5683	equal	not-sent	
UDP AppPort	16	1	Bi	5683	equal	not-sent	
UDP Length	16	1	Bi		ignore	compute-*	
UDP checksum	16	1	Bi		ignore	compute-*	

Rule 2

Field	FL	FP	DI	Value	Match Opera.	Action Action	Sent [bits]
IPv6 Version	4	1	Bi	6	ignore	not-sent	
IPv6 DiffServ	8	1	Bi	0	equal	not-sent	
IPv6 Flow Label	20	1	Bi	0	equal	not-sent	
IPv6 Length	16	1	Bi		ignore	compute-*	
IPv6 Next Header	8	1	Bi	17	equal	not-sent	
IPv6 Hop Limit	8	1	Up	255	ignore	not-sent	
IPv6 Hop Limit	8	1	Dw		ignore	value-sent	8
IPv6 DevPrefix	64	1	Bi	alpha/64	equal	not-sent	
IPv6 DevIID	64	1	Bi		ignore	DevIID	
IPv6 AppPrefix	64	1	Bi	gamma/64	equal	not-sent	
IPv6 AppIID	64	1	Bi	::1000	equal	not-sent	
UDP DevPort	16	1	Bi	8720	MSB(12)	LSB	4
UDP AppPort	16	1	Bi	8720	MSB(12)	LSB	4
UDP Length	16	1	Bi		ignore	compute-*	
UDP checksum	16	1	Bi		ignore	compute-*	

Figure 26: Context Rules

All the fields described in the three Rules depicted on Figure 26 are present in the IPv6 and UDP headers. The DevIID-DID value is found in the L2 header.

The second and third Rules use global addresses. The way the Dev learns the prefix is not in the scope of the document.

The third Rule compresses each port number to 4 bits.

Appendix B. Fragmentation Examples

This section provides examples for the various fragment reliability modes specified in this document. In the drawings, Bitmaps are shown in their uncompressed form.

Figure 27 illustrates the transmission in No-ACK mode of a SCHC Packet that needs 11 SCHC Fragments. FCN is 1 bit wide.

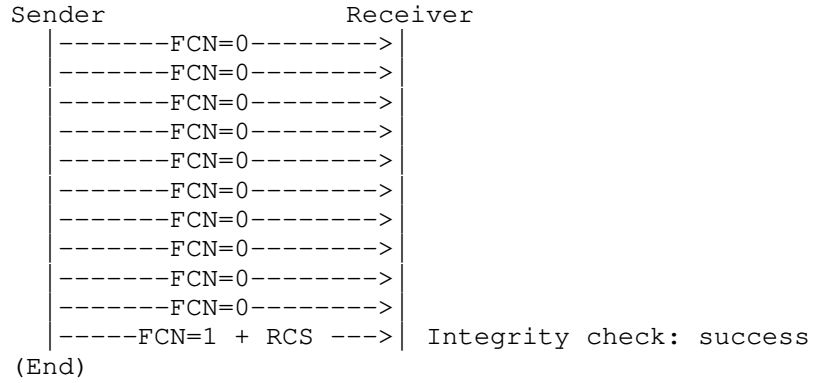


Figure 27: No-ACK mode, 11 SCHC Fragments

In the following examples, N (the size of the FCN field) is 3 bits. The All-1 FCN value is 7.

Figure 28 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7 and no lost SCHC Fragment.

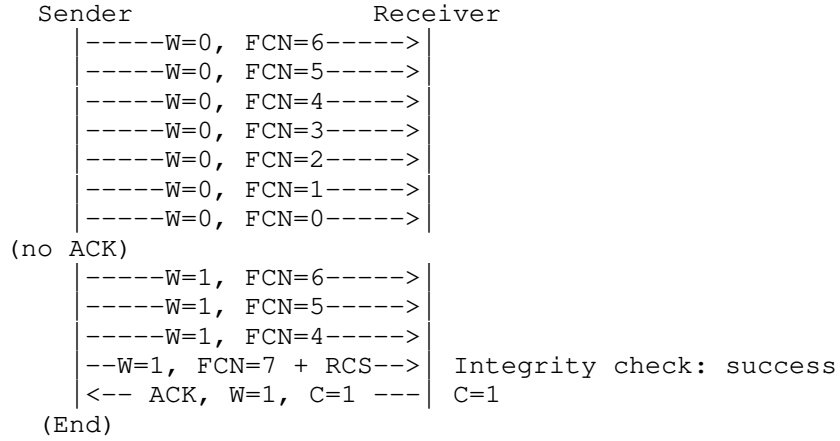


Figure 28: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, no lost SCHC Fragment.

Figure 29 illustrates the transmission in ACK-on-Error mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, WINDOW_SIZE=7 and three lost SCHC Fragments.

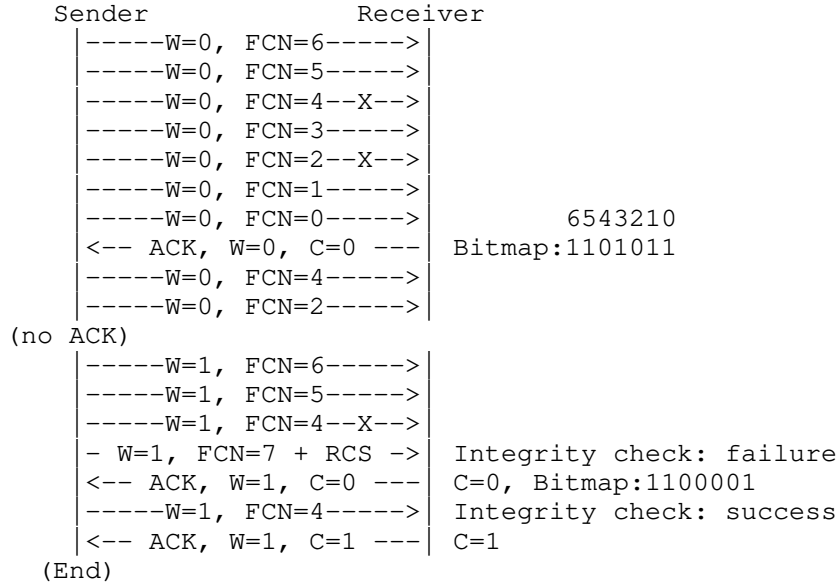


Figure 29: ACK-on-Error mode, 11 tiles, one tile per SCHC Fragment, lost SCHC Fragments.

Figure 30 shows an example of a transmission in ACK-on-Error mode of a SCHC Packet fragmented in 73 tiles, with N=5, WINDOW_SIZE=28, M=2 and 3 lost SCHC Fragments.

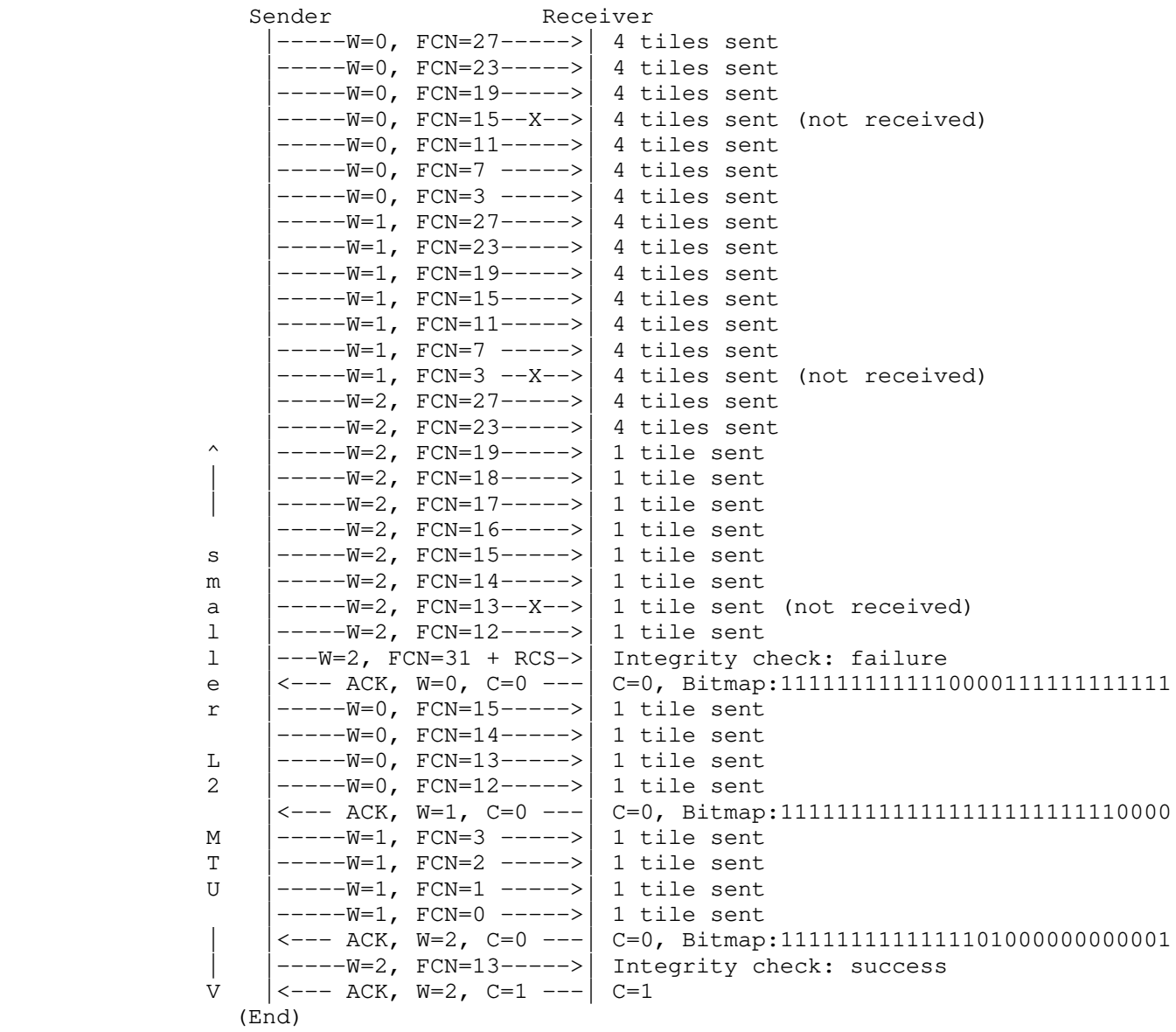


Figure 30: ACK-on-Error mode, variable MTU.

In this example, the L2 MTU becomes reduced just before sending the "W=2, FCN=19" fragment, leaving space for only 1 tile in each forthcoming SCHC Fragment. Before retransmissions, the 73 tiles are carried by a total of 25 SCHC Fragments, the last 9 being of smaller size.

Note: other sequences of events (e.g. regarding when ACKs are sent by the Receiver) are also allowed by this specification. Profiles may restrict this flexibility.

Figure 31 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, with N=3, WINDOW_SIZE=7 and no loss.

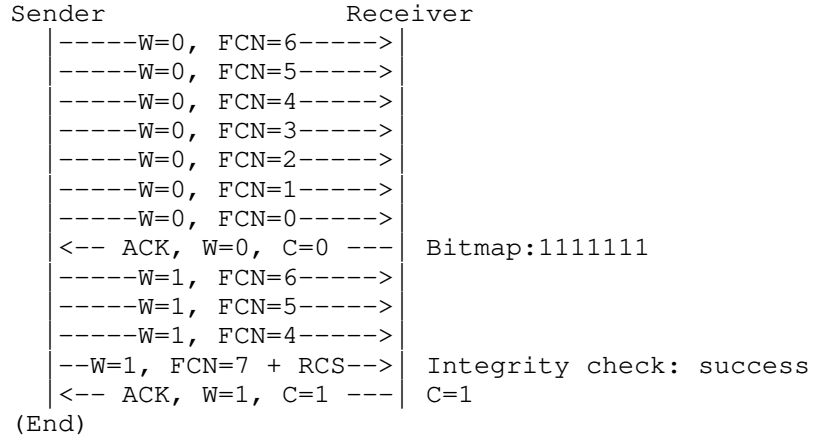


Figure 31: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, no loss.

Figure 32 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 11 tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7 and three lost SCHC Fragments.

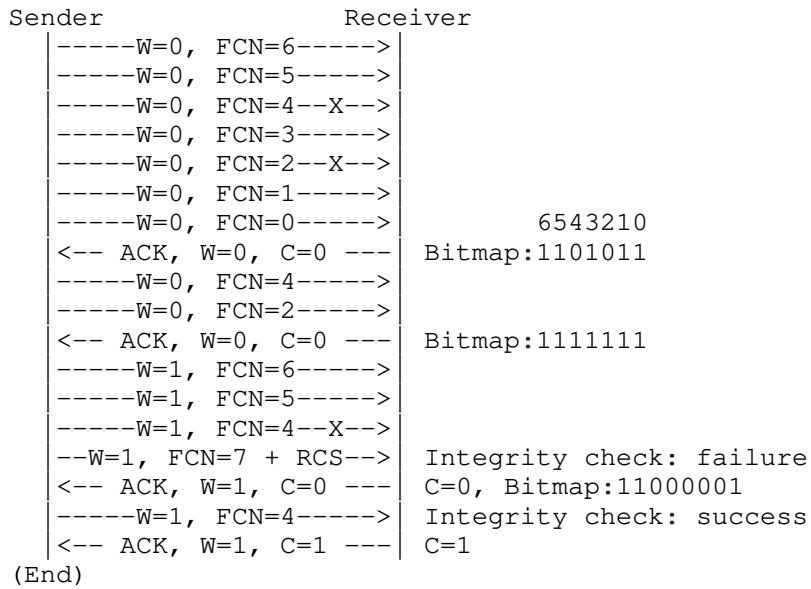


Figure 32: ACK-Always mode, 11 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 33 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3, WINDOW_SIZE=7, three lost SCHC Fragments and only one retry needed to recover each lost SCHC Fragment.

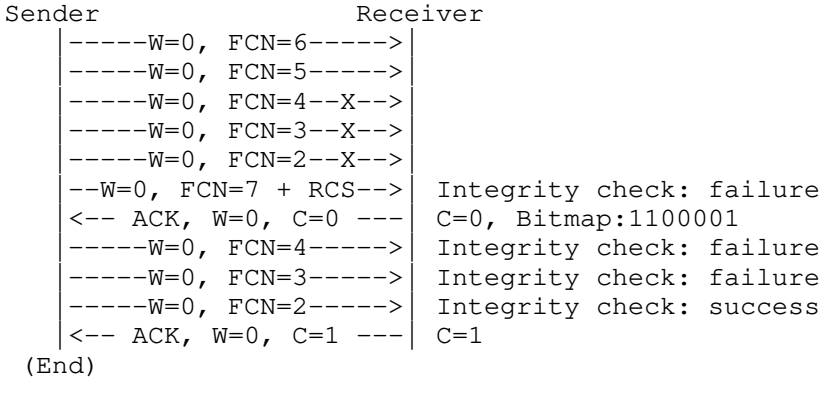


Figure 33: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, three lost SCHC Fragments.

Figure 34 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3,

WINDOW_SIZE=7, three lost SCHC Fragments, and the second SCHC ACK lost.

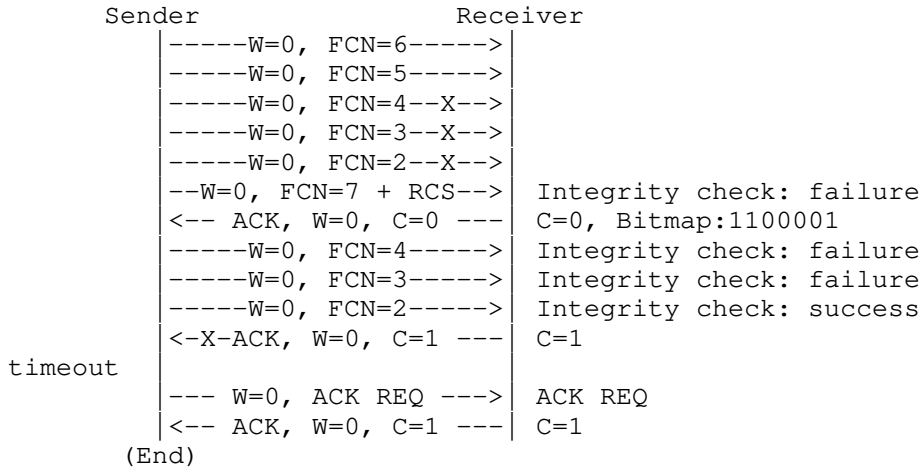


Figure 34: ACK-Always mode, 6 tiles, one tile per SCHC Fragment, SCHC ACK loss.

Figure 35 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with N=3, WINDOW_SIZE=7, with three lost SCHC Fragments, and one retransmitted SCHC Fragment lost again.

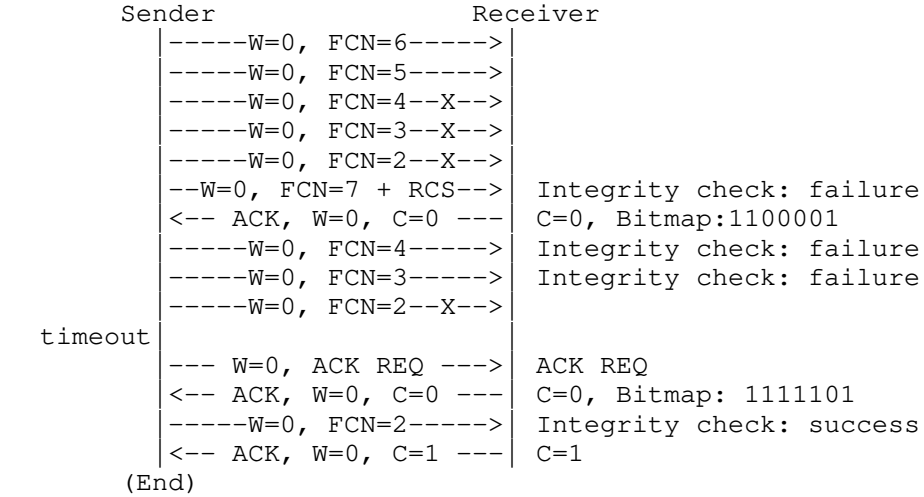


Figure 35: ACK-Always mode, 6 tiles, retransmitted SCHC Fragment lost again.

Figure 36 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 28 tiles, with one tile per SCHC Fragment, N=5, WINDOW_SIZE=24 and two lost SCHC Fragments.

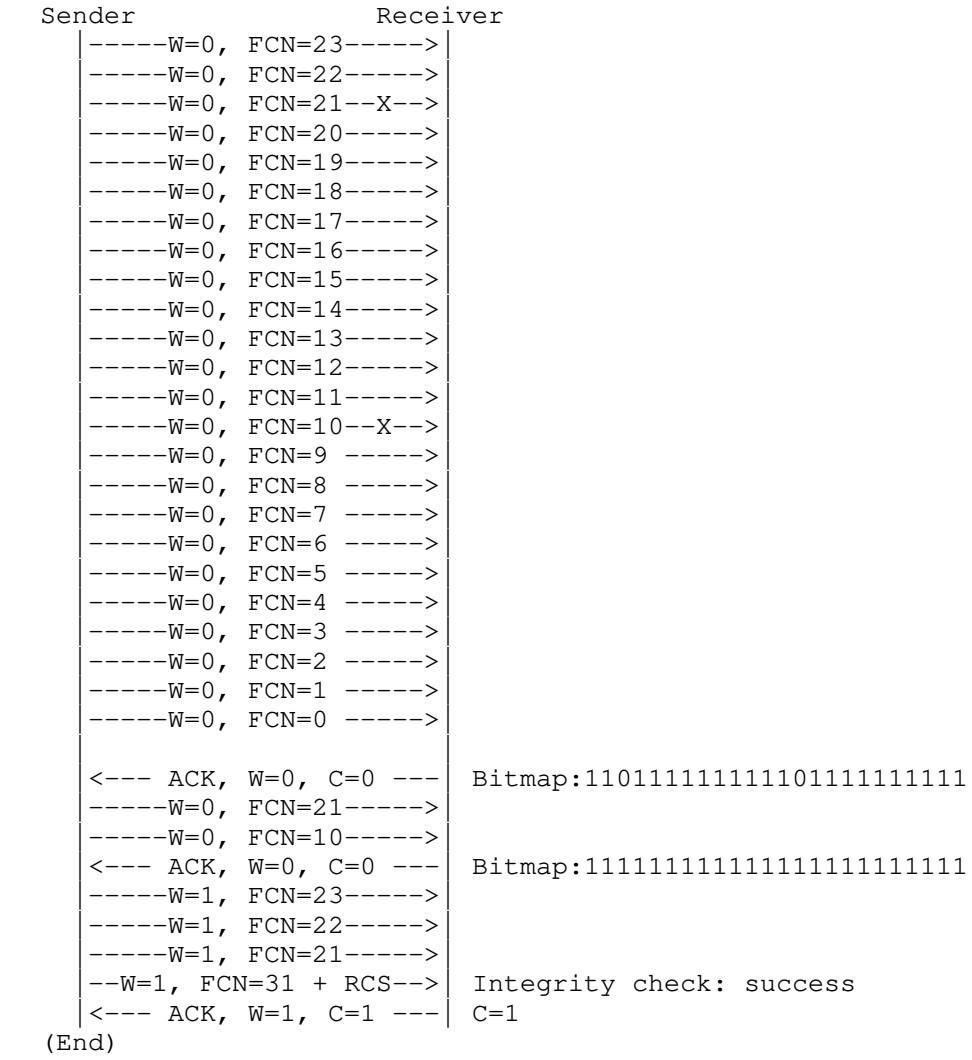


Figure 36: ACK-Always mode, 28 tiles, one tile per SCHC Fragment, lost SCHC Fragments.

Appendix C. Fragmentation State Machines

The fragmentation state machines of the sender and the receiver, one for each of the different reliability modes, are described in the following figures:

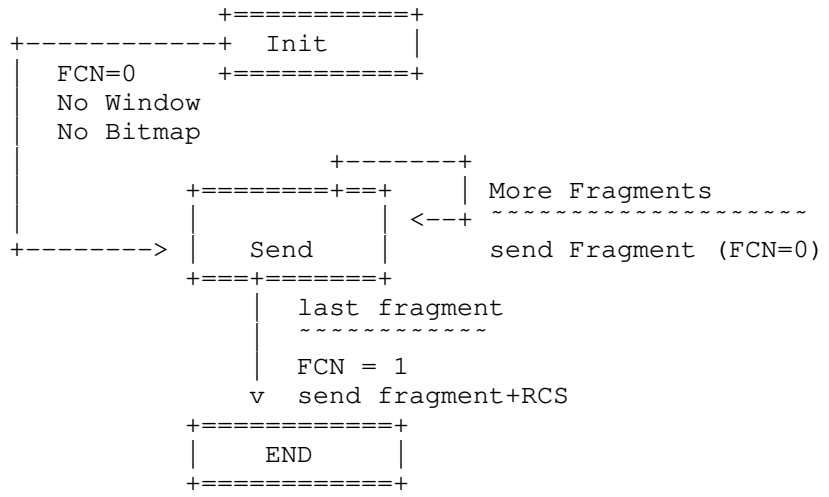


Figure 37: Sender State Machine for the No-ACK Mode

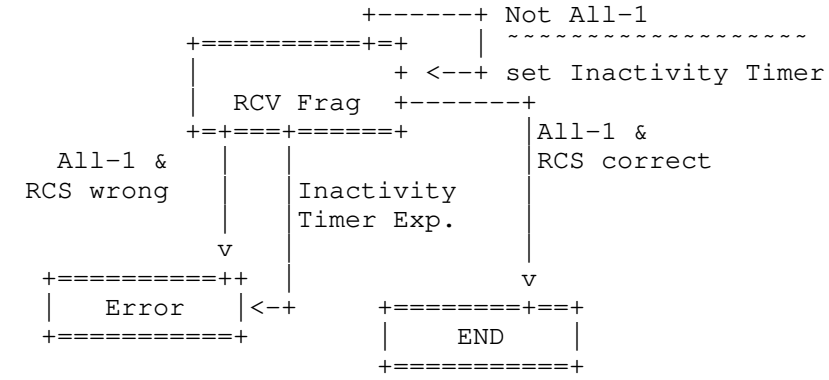


Figure 38: Receiver State Machine for the No-ACK Mode

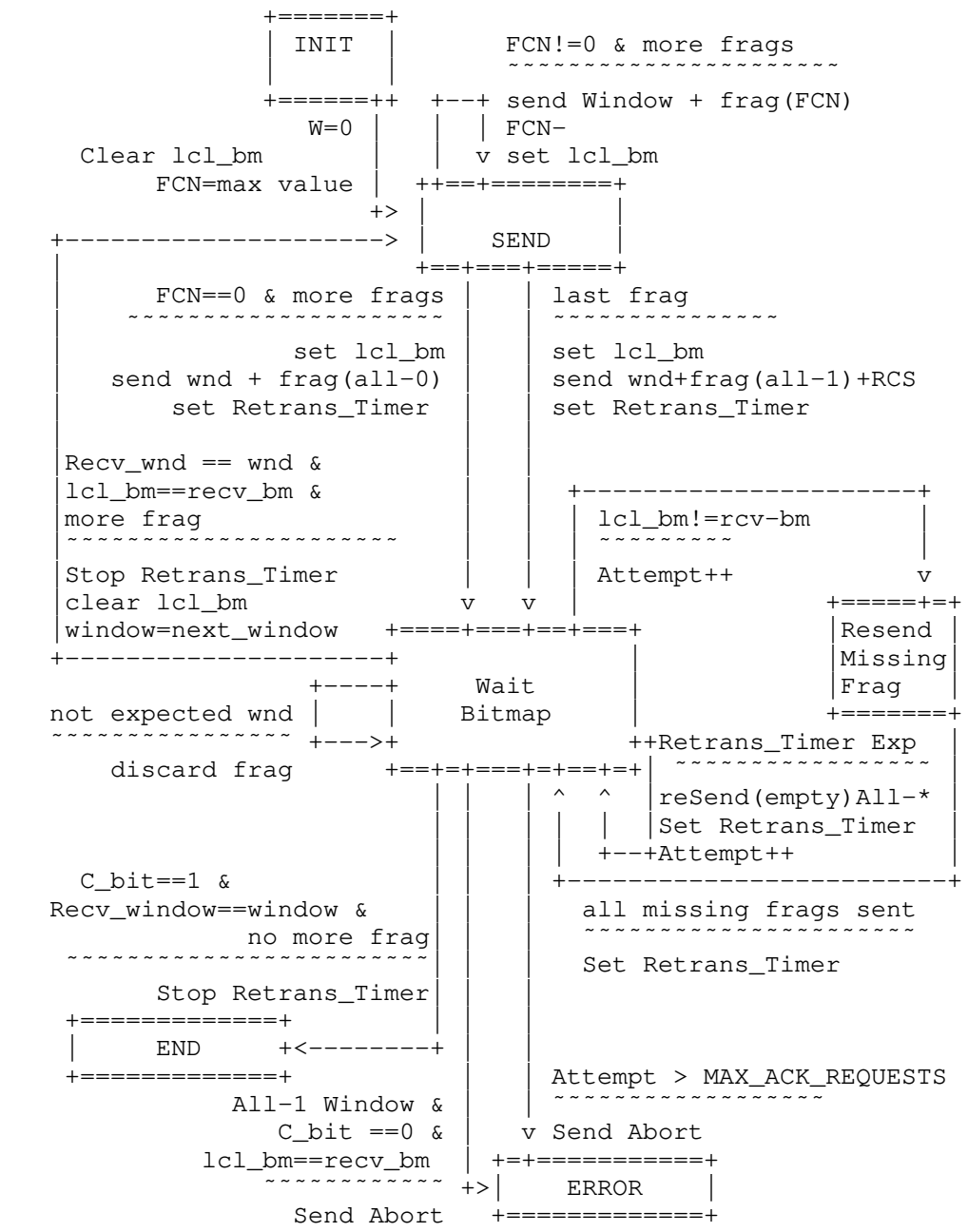
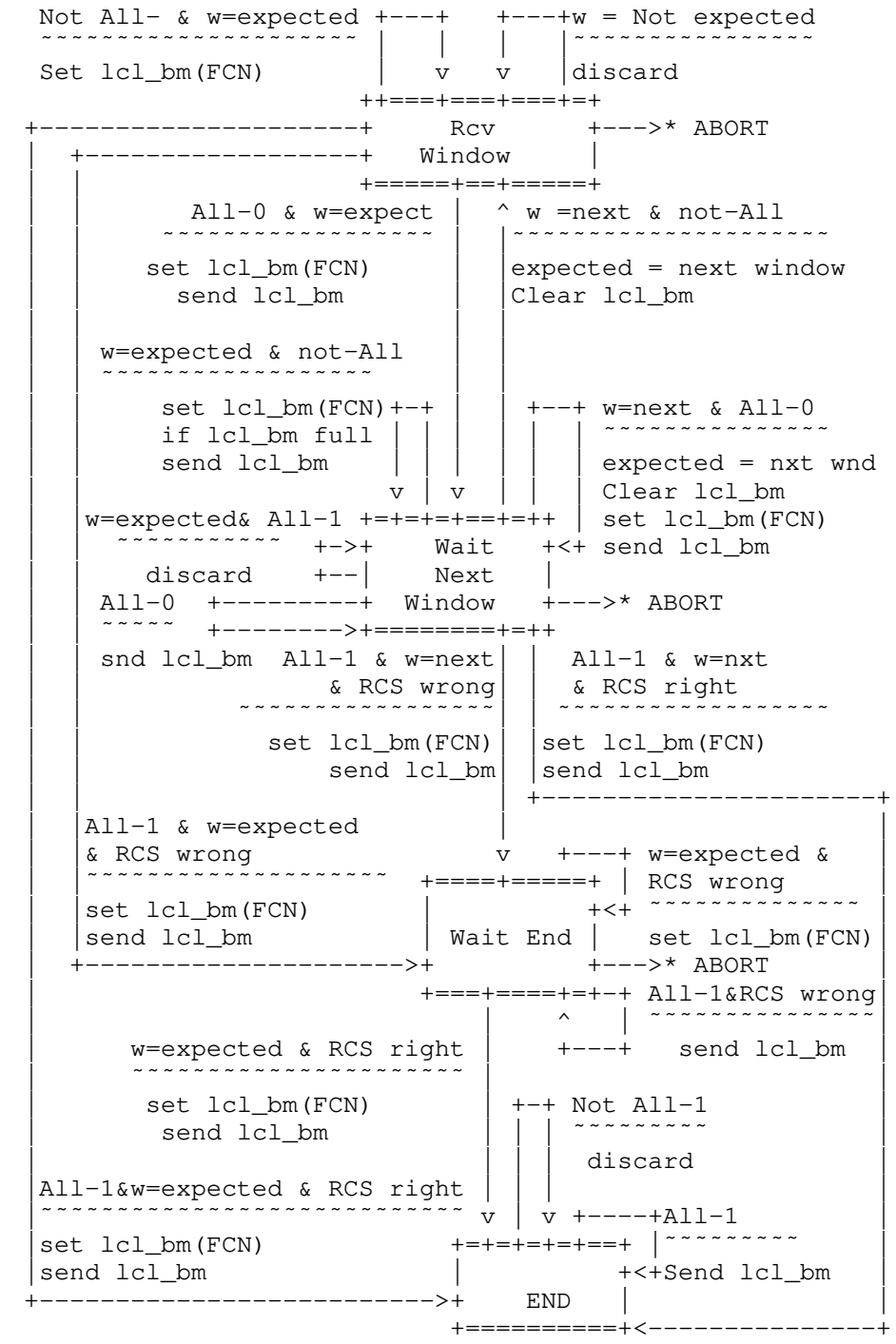


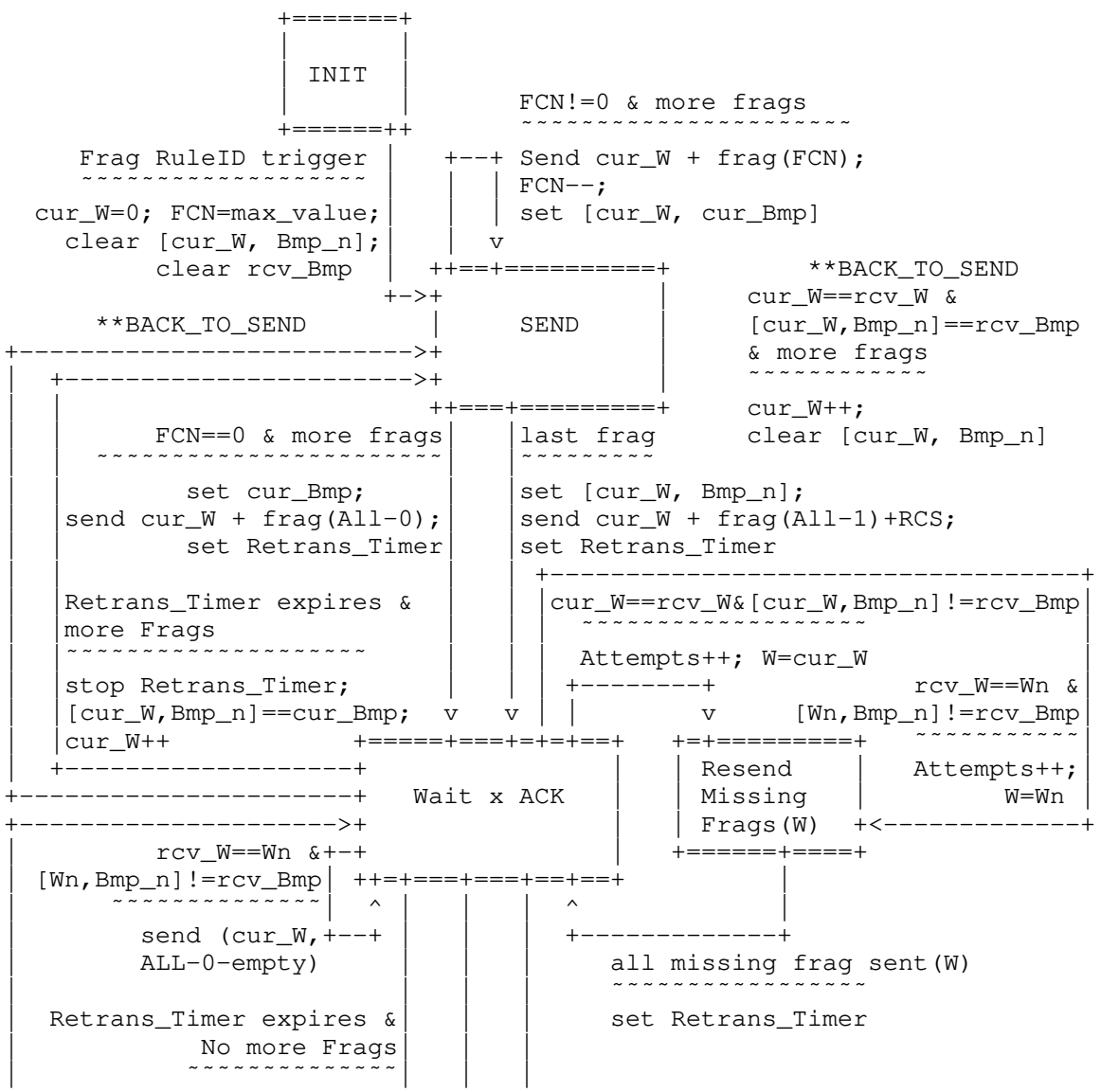
Figure 39: Sender State Machine for the ACK-Always Mode


Not All- & w=expected
Set lcl_bm(FCN)
w = Not expected
discard
Rcv Window
* ABORT
All-0 & w=expect
set lcl_bm(FCN)
send lcl_bm
^ w =next & not-All
expected = next window
Clear lcl_bm
w=expected & not-All
set lcl_bm(FCN)
if lcl_bm full
send lcl_bm
w=next & All-0
expected = nxt wnd
Clear lcl_bm
set lcl_bm(FCN)
send lcl_bm
w=expected& All-1
discard
Wait Next
All-0
Window
* ABORT
snd lcl_bm
All-1 & w=next & RCS wrong
All-1 & w=nxt & RCS right
set lcl_bm(FCN)
send lcl_bm
set lcl_bm(FCN)
send lcl_bm
All-1 & w=expected & RCS wrong
set lcl_bm(FCN)
send lcl_bm
Wait End
w=expected & RCS wrong
set lcl_bm(FCN)
* ABORT
All-1&RCS wrong
w=expected & RCS right
set lcl_bm(FCN)
send lcl_bm
Not All-1
discard
send lcl_bm
All-1&w=expected & RCS right
All-1
set lcl_bm(FCN)
send lcl_bm
Send lcl_bm
END

```
         --->* ABORT

         In any state
            on receiving a SCHC ACK REQ
               Send a SCHC ACK for the current window
```

Figure 40: Receiver State Machine for the ACK-Always Mode



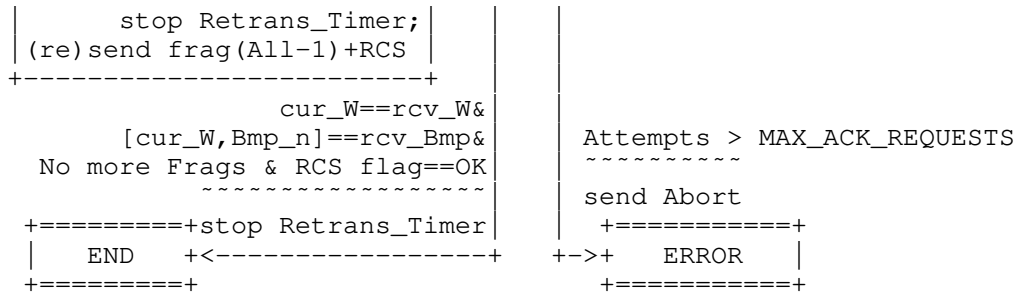
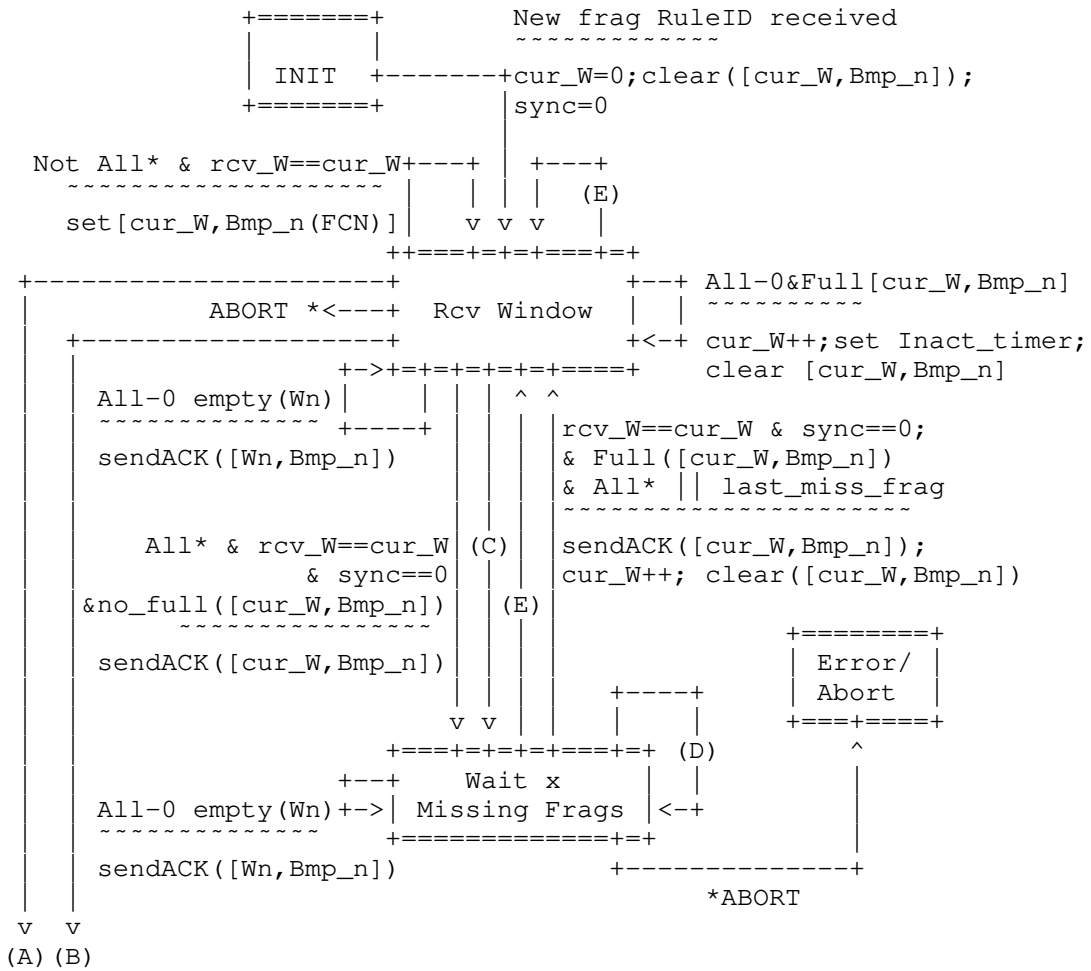


Figure 41: Sender State Machine for the ACK-on-Error Mode

This is an example only. It is not normative. The specification in Section 8.4.3.1 allows for sequences of operations different from the one shown here.



(C) All* & sync>0

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n])

(D) All* || last_miss_frag
& rcv_W!=cur_W & sync>0
& Full([rcv_W,Bmp_n])

Wn=oldest[not full(W)];
sendACK([Wn,Bmp_n]);sync--

ABORT-->* Uplink Only &
Inact_Timer expires
|| Attempts > MAX_ACK_REQUESTS

send Abort

(E) Not All* & rcv_W!=cur_W

sync++; cur_W=rcv_W;
set[cur_W,Bmp_n(FCN)]

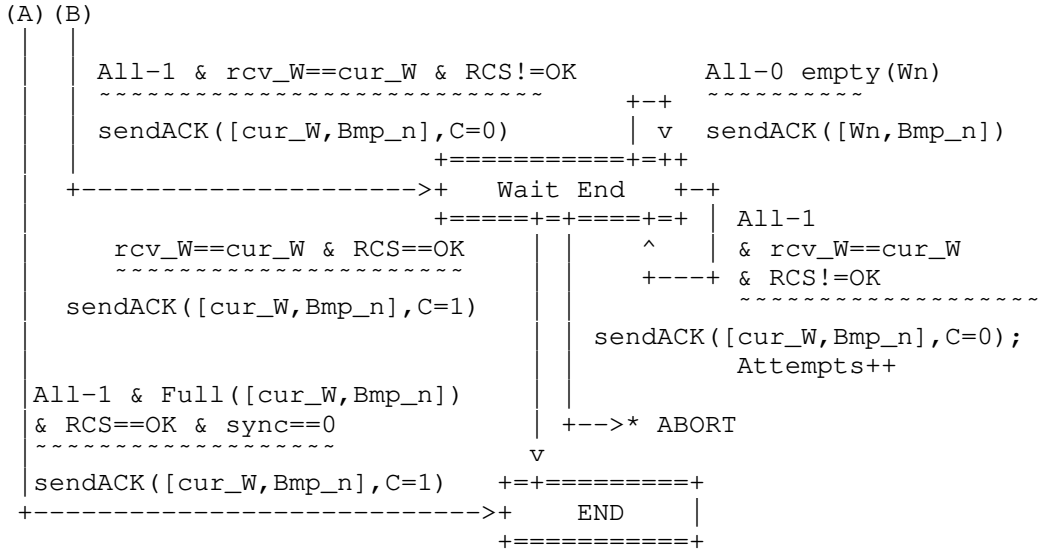


Figure 42: Receiver State Machine for the ACK-on-Error Mode

Appendix D. SCHC Parameters

This section lists the information that needs to be provided in the LPWAN technology-specific documents.

- Most common uses cases, deployment scenarios
- Mapping of the SCHC architectural elements onto the LPWAN architecture
- Assessment of LPWAN integrity checking
- Various potential channel conditions for the technology and the corresponding recommended use of SCHC C/D and F/R

This section lists the parameters that need to be defined in the Profile.

- Rule ID numbering scheme, fixed-sized or variable-sized Rule IDs, number of Rules, the way the Rule ID is transmitted
- maximum packet size that should ever be reconstructed by SCHC Decompression (MAX_PACKET_SIZE). See Section 12.

- Padding: size of the L2 Word (for most LPWAN technologies, this would be a byte; for some technologies, a bit)

- Decision to use SCHC fragmentation mechanism or not. If yes:

  * reliability mode(s) used, in which cases (e.g. based on link channel condition)

  * Rule ID values assigned to each mode in use

  * presence and number of bits for DTag (T) for each Rule ID value

  * support for interleaved packet transmission, to what extent

  * WINDOW_SIZE, for modes that use windows

  * number of bits for W (M) for each Rule ID value, for modes that use windows

  * number of bits for FCN (N) for each Rule ID value

  * size of RCS and algorithm for its computation, for each Rule ID, if different from the default CRC32. Byte fill-up with zeroes or other mechanism, to be specified.

  * Retransmission Timer duration for each Rule ID value, if applicable to the SCHC F/R mode

  * Inactivity Timer duration for each Rule ID value, if applicable to the SCHC F/R mode

  * MAX_ACK_REQUESTS value for each Rule ID value, if applicable to the SCHC F/R mode

- if L2 Word is wider than a bit and SCHC fragmentation is used, value of the padding bits (0 or 1). This is needed because the padding bits of the last fragment are included in the RCS computation.

A Profile may define a delay to be added after each SCHC message transmission for compliance with local regulations or other constraints imposed by the applications.

- In some LPWAN technologies, as part of energy-saving techniques, downlink transmission is only possible immediately after an uplink transmission. In order to avoid potentially high delay in the downlink transmission of a fragmented SCHC Packet, the SCHC Fragment receiver may perform an uplink transmission as soon as

possible after reception of a SCHC Fragment that is not the last one. Such uplink transmission may be triggered by the L2 (e.g. an L2 ACK sent in response to a SCHC Fragment encapsulated in a L2 PDU that requires an L2 ACK) or it may be triggered from an upper layer.

- o the following parameters need to be addressed in documents other than this one but not necessarily in the LPWAN technology-specific documents:
  - * The way the Contexts are provisioned
  - * The way the Rules are generated

Appendix E. Supporting multiple window sizes for fragmentation

For ACK-Always or ACK-on-Error, implementers may opt to support a single window size or multiple window sizes. The latter, when feasible, may provide performance optimizations. For example, a large window size should be used for packets that need to be split into a large number of tiles. However, when the number of tiles required to carry a packet is low, a smaller window size, and thus a shorter Bitmap, may be sufficient to provide reception status on all tiles. If multiple window sizes are supported, the Rule ID signals the window size in use for a specific packet transmission.

Appendix F. ACK-Always and ACK-on-Error on quasi-bidirectional links

The ACK-Always and ACK-on-Error modes of SCHC F/R are bidirectional protocols: they require a feedback path from the reassembler to the fragmenter.

Some LPWAN technologies provide quasi-bidirectional connectivity, whereby a downlink transmission from the Network Infrastructure can only take place right after an uplink transmission by the Dev.

When using SCHC F/R to send fragmented SCHC Packets downlink over these quasi-bidirectional links, the following situation may arise: if an uplink SCHC ACK is lost, the SCHC ACK REQ message by the sender could be stuck indefinitely in the downlink queue at the Network Infrastructure, waiting for a transmission opportunity.

There are many ways by which this deadlock can be avoided. The Dev application might be sending recurring uplink messages such as keep-alive, or the Dev application stack might be sending other recurring uplink messages as part of its operation. However, these are out of the control of this generic SCHC specification.

In order to cope with quasi-bidirectional links, a SCHC-over-foo specification may want to amend the SCHC F/R specification to add a timer-based retransmission of the SCHC ACK. Below is an example of the suggested behavior for ACK-Always mode. Because it is an example, [RFC2119] language is deliberately not used here.

For downlink transmission of a fragmented SCHC Packet in ACK-Always mode, the SCHC Fragment receiver may support timer-based SCHC ACK retransmission. In this mechanism, the SCHC Fragment receiver initializes and starts a timer (the UplinkACK Timer) after the transmission of a SCHC ACK, except when the SCHC ACK is sent in response to the last SCHC Fragment of a packet (All-1 fragment). In the latter case, the SCHC Fragment receiver does not start a timer after transmission of the SCHC ACK.

If, after transmission of a SCHC ACK that is not an All-1 fragment, and before expiration of the corresponding UplinkACK timer, the SCHC Fragment receiver receives a SCHC Fragment that belongs to the current window (e.g. a missing SCHC Fragment from the current window) or to the next window, the UplinkACK timer for the SCHC ACK is stopped. However, if the UplinkACK timer expires, the SCHC ACK is resent and the UplinkACK timer is reinitialized and restarted.

The default initial value for the UplinkACK Timer, as well as the maximum number of retries for a specific SCHC ACK, denoted MAX_ACK_REQUESTS, is to be defined in a Profile. The initial value of the UplinkACK timer is expected to be greater than that of the Retransmission timer, in order to make sure that a (buffered) SCHC Fragment to be retransmitted finds an opportunity for that transmission. One exception to this recommendation is the special case of the All-1 SCHC Fragment transmission.

When the SCHC Fragment sender transmits the All-1 SCHC Fragment, it starts its Retransmission Timer with a large timeout value (e.g. several times that of the initial UplinkACK Timer). If a SCHC ACK is received before expiration of this timer, the SCHC Fragment sender retransmits any lost SCHC Fragments as reported by the SCHC ACK, or if the SCHC ACK confirms successful reception of all SCHC Fragments of the last window, the transmission of the fragmented SCHC Packet is considered complete. If the timer expires, and no SCHC ACK has been received since the start of the timer, the SCHC Fragment sender assumes that the All-1 SCHC Fragment has been successfully received (and possibly, the last SCHC ACK has been lost: this mechanism assumes that the Retransmission Timer for the All-1 SCHC Fragment is long enough to allow several SCHC ACK retries if the All-1 SCHC Fragment has not been received by the SCHC Fragment receiver, and it also assumes that it is unlikely that several ACKs become all lost).

Authors' Addresses

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

Email: ana@ackl.io

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

Email: Laurent.Toutain@imt-atlantique.fr

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

Email: carlesgo@entel.upc.edu

Dominique Barthel
Orange Labs
28 chemin du Vieux Chene
38243 Meylan
France

Email: dominique.barthel@orange.com

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
Labege 31670
France

Email: JuanCarlos.Zuniga@sigfox.com

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

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

January 25, 2021

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

Abstract

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on July 29, 2021.

Copyright Notice

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
2. Terminology 3
3. Static Context Header Compression Overview 4
4. LoRaWAN Architecture 6
4.1. Device classes (A, B, C) and interactions 7
4.2. Device addressing 8
4.3. General Frame Types 8
4.4. LoRaWAN MAC Frames 9
4.5. LoRaWAN FPort 9
4.6. LoRaWAN empty frame 9
4.7. Unicast and multicast technology 9
5. SCHC-over-LoRaWAN 10
5.1. LoRaWAN FPort and RuleID 10
5.2. Rule ID management 10
5.3. Interface IDentifier (IID) computation 11
5.4. Padding 12
5.5. Decompression 12
5.6. Fragmentation 13
5.6.1. DTag 13
5.6.2. Uplink fragmentation: From device to SCHC gateway . . 13
5.6.3. Downlink fragmentation: From SCHC gateway to device . 17
5.7. SCHC Fragment Format 20
5.7.1. All-0 SCHC fragment 20
5.7.2. All-1 SCHC fragment 21
5.7.3. Delay after each LoRaWAN frame to respect local regulation 21
6. Security Considerations 21
7. IANA Considerations 21
Acknowledgements 21
Contributors 21
10. References 22
10.1. Normative References 22
10.2. Informative References 23
10.3. URIs 23
Appendix A. Examples 23
A.1. Uplink - Compression example - No fragmentation 23
A.2. Uplink - Compression and fragmentation example 24
A.3. Downlink 26
Authors' Addresses 28

1. Introduction

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

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

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

2. Terminology

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

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

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

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

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

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

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

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

3. Static Context Header Compression Overview

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

It defines:

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

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

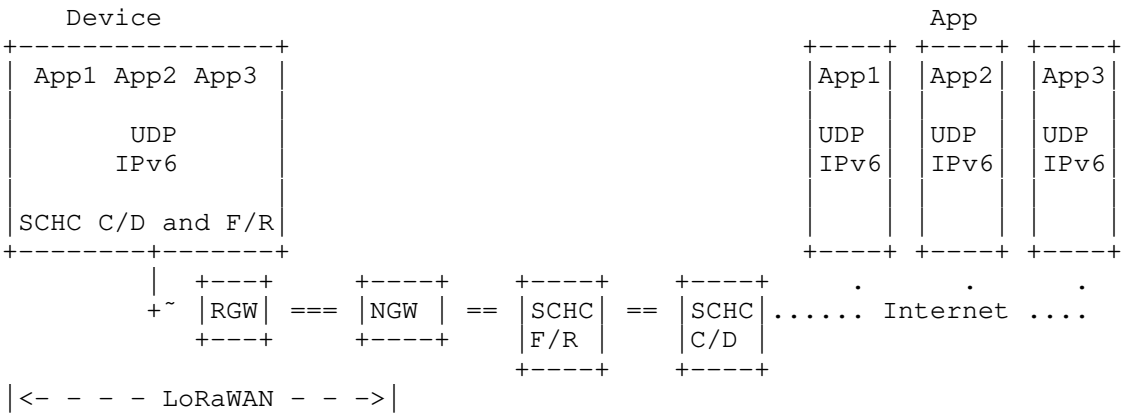


Figure 1: Architecture

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

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

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

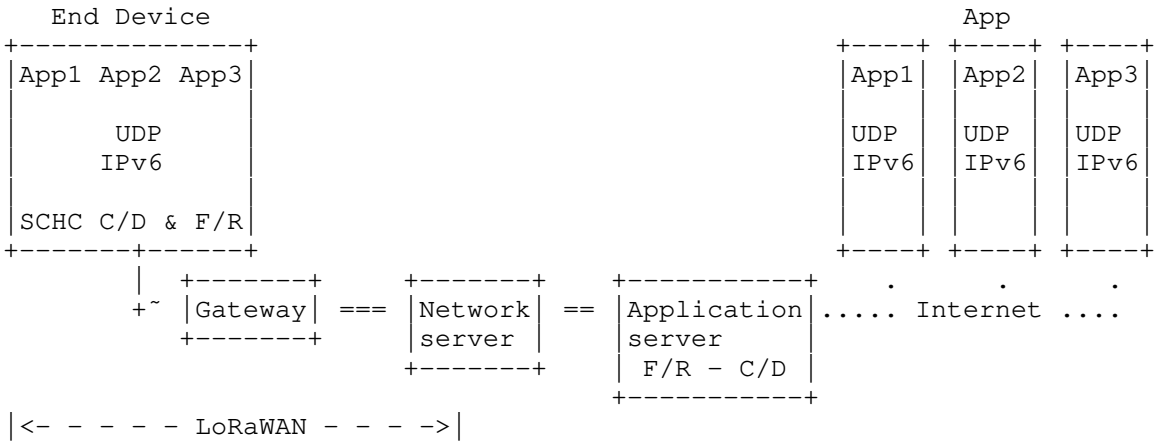


Figure 2: SCHC Architecture mapped to LoRaWAN

4. LoRaWAN Architecture

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

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

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

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

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

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

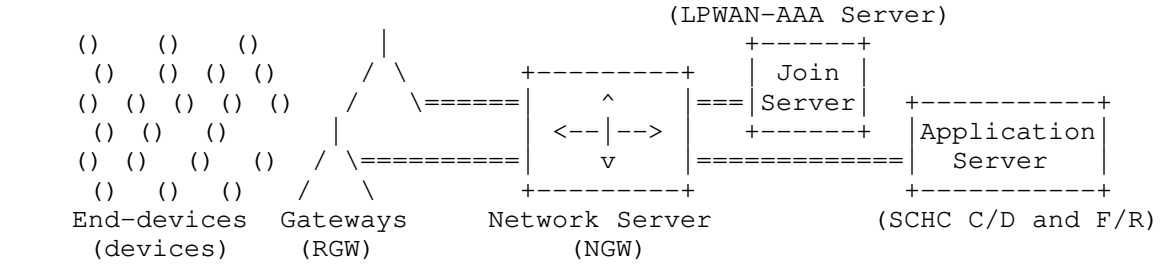


Figure 3: LPWAN Architecture

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

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

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

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

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

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

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

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

4.2. Device addressing

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

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

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

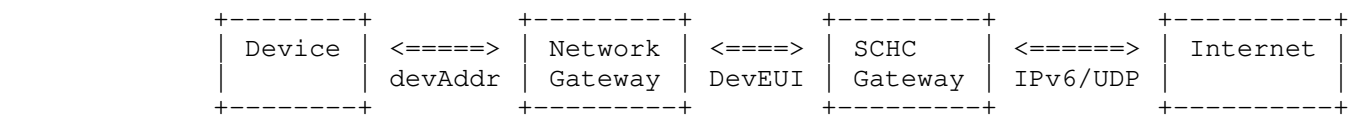


Figure 4: LoRaWAN addresses

4.3. General Frame Types

LoRaWAN implements the possibility to send confirmed or unconfirmed frames:

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

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

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

4.4. LoRaWAN MAC Frames

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

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

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

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

4.5. LoRaWAN FPort

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

4.6. LoRaWAN empty frame

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

4.7. Unicast and multicast technology

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

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

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

5. SCHC-over-LoRaWAN

5.1. LoRaWAN FPort and RuleID

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

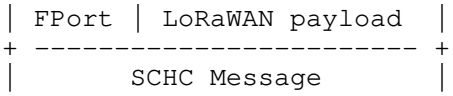


Figure 5: SCHC Message in LoRaWAN

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

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

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

5.2. Rule ID management

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

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

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

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

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

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

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

5.3. Interface IDentifier (IID) computation

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

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

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

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

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

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

Example with:

- DevEUI: 0x1122334455667788

- appSKey: 0x00AABBCCDDEEFF00AABBCCDDEEFFAABB

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

Figure 6: Example of IID computation.

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

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

5.4. Padding

All padding bits MUST be 0.

5.5. Decompression

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

RuleIDs matching FPortUp and FPortDown are reserved for SCHC Fragmentation.

5.6. Fragmentation

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

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

5.6.1. DTag

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

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

5.6.2. Uplink fragmentation: From device to SCHC gateway

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

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

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

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

- Penultimate tile MUST be equal to the regular size.

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

- Tile: size is 10 bytes.

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

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

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

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

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

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

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

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

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

5.6.2.1. Regular fragments

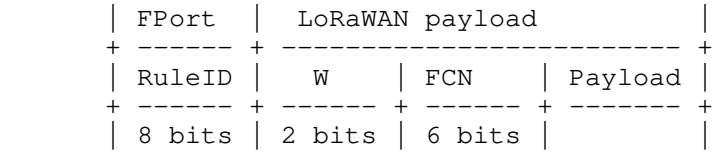


Figure 7: All fragments except the last one. SCHC header size is 16 bits, including LoRaWAN FPort.

5.6.2.2. Last fragment (All-1)

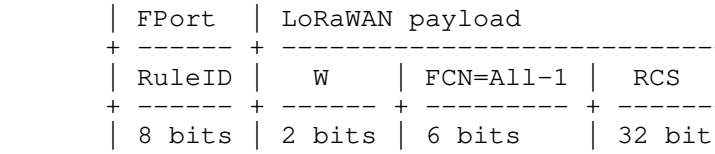


Figure 8: All-1 SCHC Message: the last fragment without last tile.

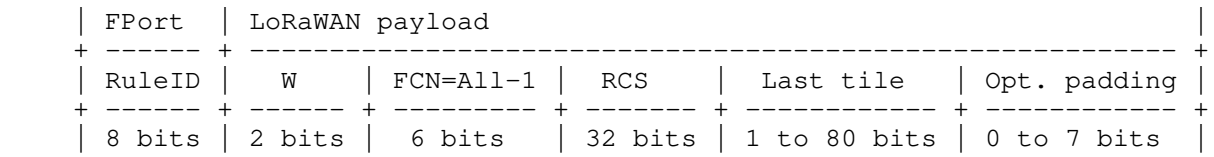


Figure 9: All-1 SCHC Message: the last fragment with last tile.

5.6.2.3. SCHC ACK

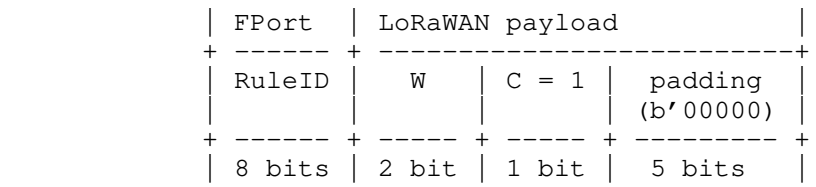


Figure 10: SCHC ACK format, correct RCS check.

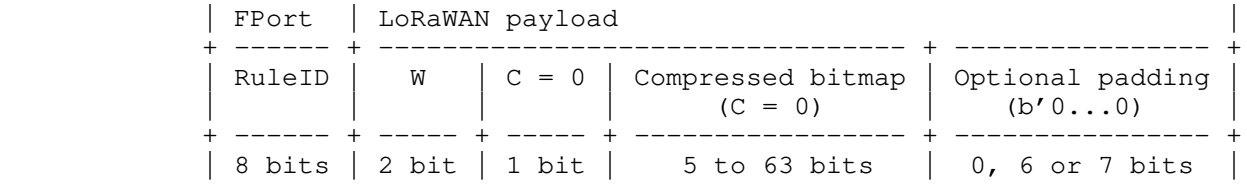


Figure 11: SCHC ACK format, failed RCS check.

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

5.6.2.4. Receiver-Abort

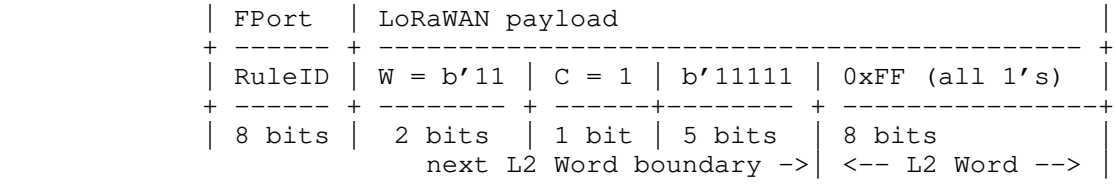


Figure 12: Receiver-Abort format.

5.6.2.5. SCHC acknowledge request

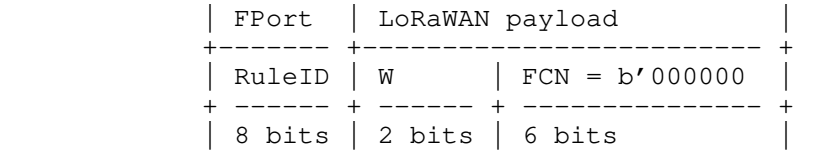


Figure 13: SCHC ACK REQ format.

5.6.3. Downlink fragmentation: From SCHC gateway to device

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

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

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

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

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

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

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

5.6.3.1. Regular fragments

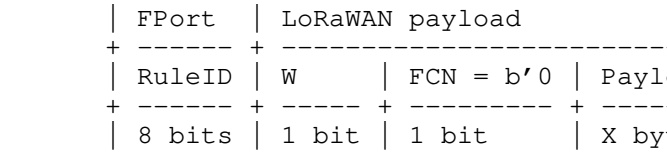


Figure 14: All fragments but the last one. Header size 10 bits, including LoRaWAN FPort.

5.6.3.2. Last fragment (All-1)

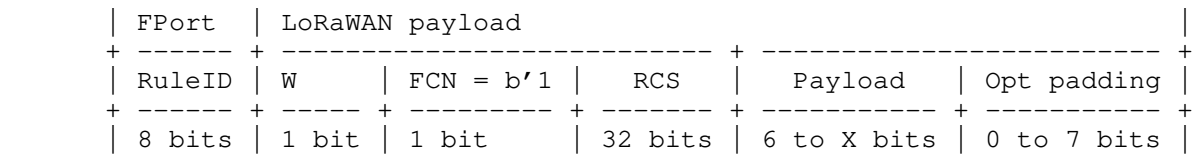


Figure 15: All-1 SCHC Message: the last fragment.

5.6.3.3. SCHC ACK

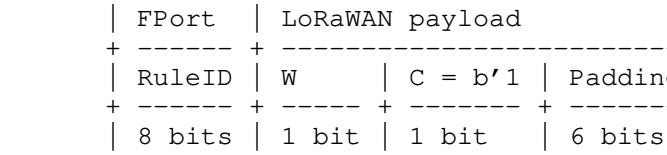


Figure 16: SCHC ACK format, RCS is correct.

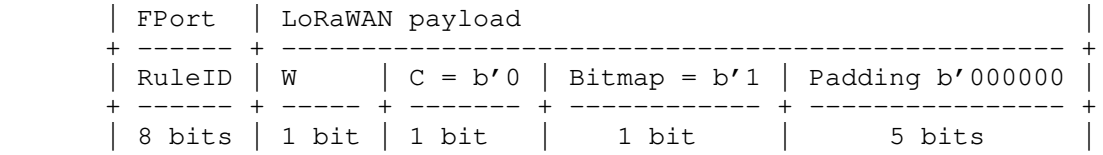


Figure 17: SCHC ACK format, RCS is incorrect.

5.6.3.4. Receiver-Abort

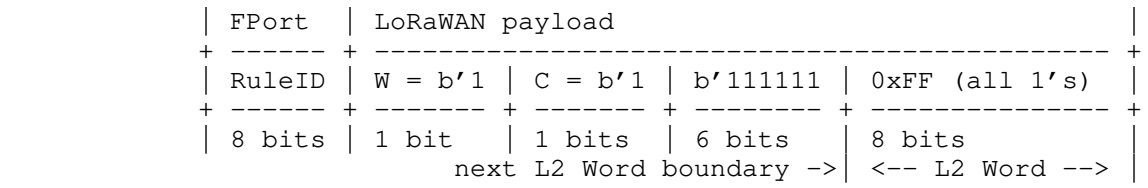


Figure 18: Receiver-Abort packet (following an All-1 SCHC Fragment with incorrect RCS).

5.6.3.5. Downlink retransmission timer

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

5.6.3.5.1. Class A devices

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

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

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

SCHC All-0 (FCN=0)

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

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

SCHC All-1 (FCN=1)

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

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

5.6.3.6. Class B or Class C devices

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

RECOMMENDED retransmission timer value:

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

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

5.7. SCHC Fragment Format

5.7.1. All-0 SCHC fragment

Uplink fragmentation (Ack-On-Error):

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

Downlink fragmentation (Ack-always):

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

5.7.2. All-1 SCHC fragment

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

5.7.3. Delay after each LoRaWAN frame to respect local regulation

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

6. Security Considerations

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

7. IANA Considerations

This document has no IANA actions.

Acknowledgements

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

Contributors

Contributors ordered by family name.

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

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

Michael Coracin
Semtech
Email: mcoracin@semtech.com

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

Nicolas Sornin
Semtech
Email: nsornin@semtech.com

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

10. References

10.1. Normative References

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

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

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

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

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

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

10.2. Informative References

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

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

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

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

10.3. URIs

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

Appendix A. Examples

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

A.1. Uplink - Compression example - No fragmentation

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

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

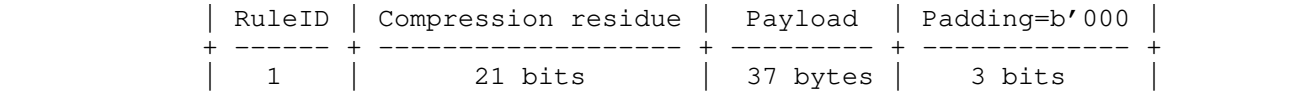


Figure 19: Uplink example: SCHC Message

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

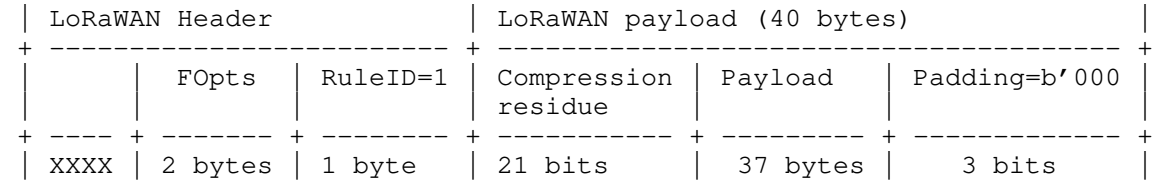


Figure 20: Uplink example: LoRaWAN packet

A.2. Uplink - Compression and fragmentation example

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

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

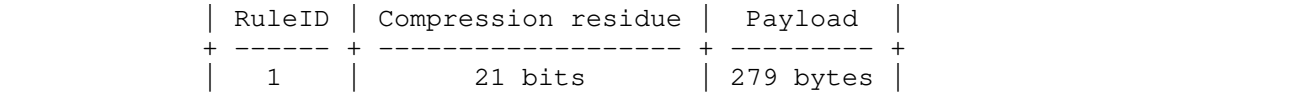


Figure 21: Uplink example: SCHC Message

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

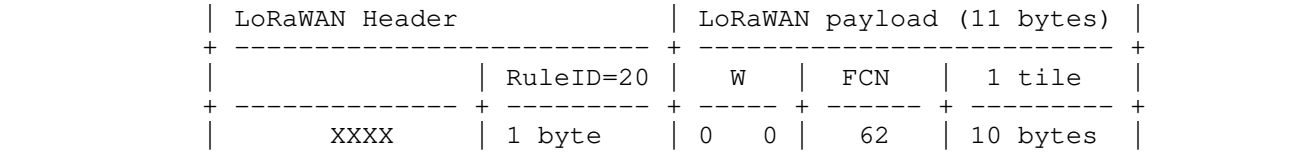


Figure 22: Uplink example: LoRaWAN packet 1

Content of the tile is:

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

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

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

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

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

Figure 24: Uplink example: LoRaWAN packet 2

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

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

Figure 25: Uplink example: LoRaWAN packet 3

Then All-1 message can be transmitted:

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

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

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

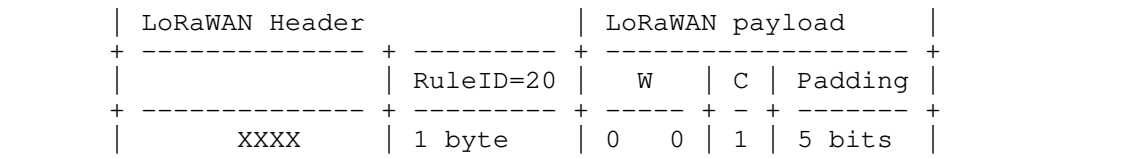


Figure 27: Uplink example: LoRaWAN packet 5 - SCHC ACK

A.3. Downlink

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

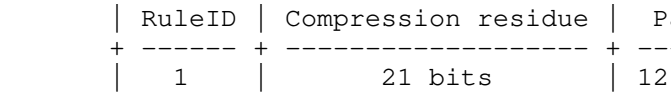


Figure 28: Downlink example: SCHC Message

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

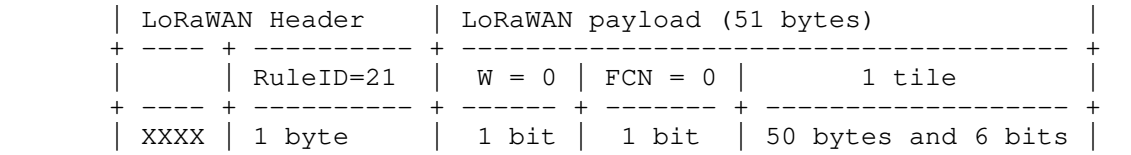


Figure 29: Downlink example: LoRaWAN packet 1 - SCHC Fragment 1

Content of the tile is:

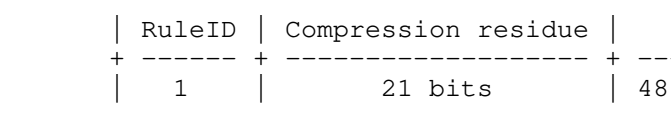


Figure 30: Downlink example: LoRaWAN packet 1: Tile content

The receiver answers with a SCHC ACK:

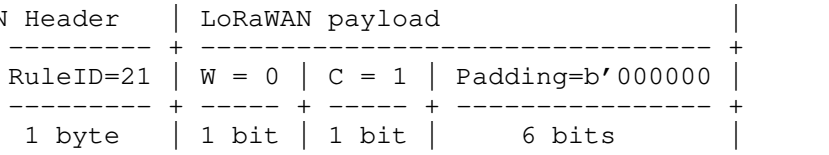


Figure 31: Downlink example: LoRaWAN packet 2 - SCHC ACK

The second downlink is sent, two FOpts:

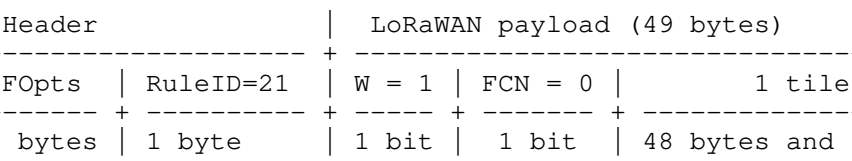


Figure 32: Downlink example: LoRaWAN packet 3 - SCHC Fragment 2

The receiver answers with an SCHC ACK:

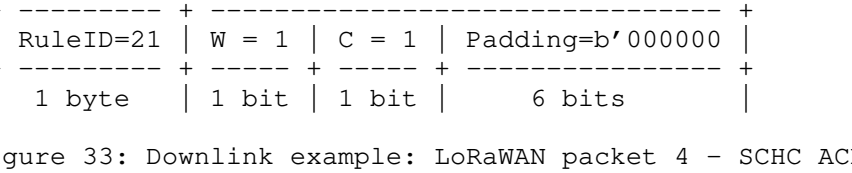


Figure 33: Downlink example: LoRaWAN packet 4 - SCHC ACK

The last downlink is sent, no FOpts:

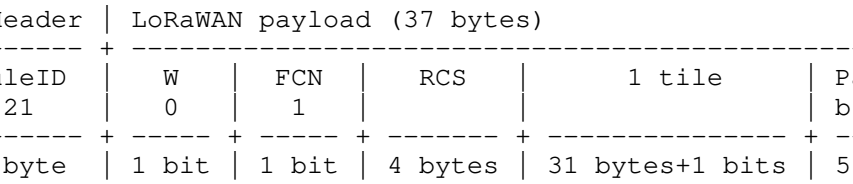


Figure 34: Downlink example: LoRaWAN packet 5 - All-1 SCHC message

The receiver answers to the sender with an SCHC ACK:

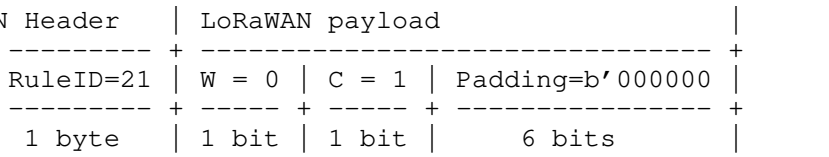


Figure 35: Downlink example: LoRaWAN packet 6 - SCHC ACK

Authors' Addresses

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

Email: ogimenez@semtech.com

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

Email: ivaylo@ackl.io

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 August 2023

JC. Zuniga
C. Gomez
S. Aguilar
Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
S. Cespedes
Concordia University
D. Wistuba
NIC Labs, Universidad de Chile
J. Boite
Unabiz (Sigfox)

3 February 2023

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

Abstract

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

Status of This Memo

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 7 August 2023.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction 3
2. Terminology 4
3. SCHC over Sigfox 4
3.1. Network Architecture 4
3.2. Uplink 6
3.3. Downlink 7
3.3.1. SCHC ACK on Downlink 8
3.4. SCHC Rules 9
3.5. Fragmentation 9
3.5.1. Uplink Fragmentation 10
3.5.2. Downlink Fragmentation 15
3.6. SCHC over Sigfox F/R Message Formats 16
3.6.1. Uplink No-ACK Mode: Single-byte SCHC Header 16
3.6.2. Uplink ACK-on-Error Mode: Single-byte SCHC Header . . 18
3.6.3. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1 20
3.6.4. Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 2 22
3.6.5. Downlink ACK-Always Mode: Single-byte SCHC Header . . 25
3.7. Padding 27
4. Fragmentation Rules Examples 27
4.1. Uplink Fragmentation Rules Examples 27
4.2. Downlink Fragmentation Rules Example 29
5. Fragmentation Sequence Examples 29
5.1. Uplink No-ACK Examples 29
5.2. Uplink ACK-on-Error Examples: Single-byte SCHC Header . . 30
5.3. SCHC Abort Examples 36
6. Security considerations 37
7. IANA Considerations 38
8. Acknowledgements 38
9. References 38
9.1. Normative References 38

9.2. Informative References 39
Authors' Addresses 40

1. Introduction

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

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

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

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

2. Terminology

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

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

3. SCHC over Sigfox

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

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

3.1. Network Architecture

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

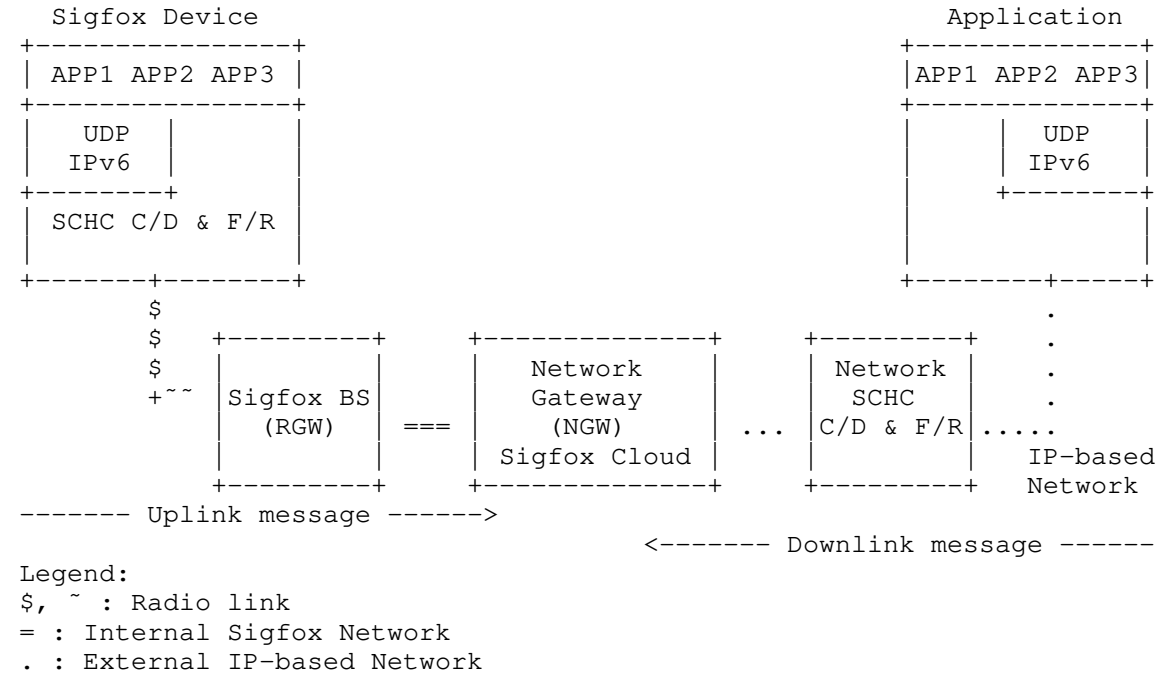


Figure 1: Network Architecture

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

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

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

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

3.2. Uplink

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

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

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

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

* Device ID
* Message Sequence Number
* Message Payload
* Message Timestamp
* Device Geolocation (optional)
* Received Signal Strength Indicator (RSSI) (optional)
* Device Temperature (optional)
* Device Battery Voltage (optional)

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

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

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

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

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

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

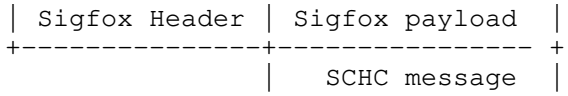


Figure 2: SCHC Message in Sigfox

3.3. Downlink

Downlink transmissions are Device-driven and can only take place following an Uplink communication that so indicates. Hence, a Sigfox Device explicitly indicates its intention to receive a Downlink message (with a size of 8 bytes) using a Downlink request flag when sending the preceding Uplink message to the network. The Downlink request flag is part of the Sigfox protocol headers. After completing the Uplink transmission, the Device opens a fixed window for Downlink reception. The delay and duration of the reception opportunity window have fixed values. If there is a Downlink message to be sent for this given Device (e.g., either a response to the Uplink message or queued information waiting to be transmitted), the network transmits this message to the Device during the reception

window. If no message is received by the Device after the reception opportunity window has elapsed, the Device closes the reception window opportunity and gets back to the normal mode (e.g., continue Uplink transmissions, sleep, stand-by, etc.)

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

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

3.3.1. SCHC ACK on Downlink

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

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

3.4. SCHC Rules

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

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

3.5. Fragmentation

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

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

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

3.5.1. Uplink Fragmentation

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

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

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

3.5.1.1. SCHC Sender-Abort

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

3.5.1.2. SCHC Receiver-Abort

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

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

MAX_ACK_REQUESTS can be increased when facing high error rates.

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

3.5.1.3. Single-byte SCHC Header for Uplink Fragmentation

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

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

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

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

* RuleID size: 3 bits
* DTag size (T): 0 bit
* Fragment Compressed Number (FCN) size (N): 5 bits

Other F/R parameters MUST be configured as follows:

* As per [RFC8724], in the No-ACK mode the W (window) field is not present.
* Regular tile size: 11 bytes
* All-1 tile size: 0 to 10 bytes

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

* RCS size: 5 bits

The maximum SCHC Packet size is 340 bytes.

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

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

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

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

* RuleID size: 3 bits

* DTag size (T): 0 bit

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

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

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

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

* Regular tile size: 11 bytes

* All-1 tile size: 0 to 10 bytes

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

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

* RCS size: 3 bits

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

3.5.1.4. Two-byte SCHC Header for Uplink Fragmentation

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

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

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

* RuleID size is: 6 bits

* DTag size (T) is: 0 bit

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

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

* RCS size: 4 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5

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

* Regular tile size: 10 bytes

* All-1 tile size: 1 to 10 bytes

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

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

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

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

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

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

* RuleID size is: 8 bits
* DTag size (T) is: 0 bit
* Window index (W) size (M): 3 bits
* Fragment Compressed Number (FCN) size (N): 5 bits.
* RCS size: 5 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5
* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)
* Regular tile size: 10 bytes
* All-1 tile size: 0 to 9 bytes
* Retransmission Timer: Application-dependent. The default value is 12 hours.
* Inactivity Timer: Application-dependent. The default value is 12 hours.

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

3.5.1.5. All-1 SCHC Fragment and RCS behaviour

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

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

3.5.2. Downlink Fragmentation

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

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

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

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

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

* RuleID size: 3 bits
* DTag size (T): 0 bit
* Window index (W) size (M) is: 0 bit
* Fragment Compressed Number (FCN) size (N): 5 bits

Other F/R parameters MUST be configured as follows:

* MAX_ACK_REQUESTS: 5
* WINDOW_SIZE: 31 (with a maximum value of FCN=0b11110)
* Regular tile size: 7 bytes
* All-1 tile size: 0 to 6 bytes
* Retransmission Timer: Application-dependent. The default value is 12 hours.
* Inactivity Timer: Application-dependent. The default value is 12 hours.
* RCS size: 5 bits

3.6. SCHC over Sigfox F/R Message Formats

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

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

3.6.1.1. Regular SCHC Fragment

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

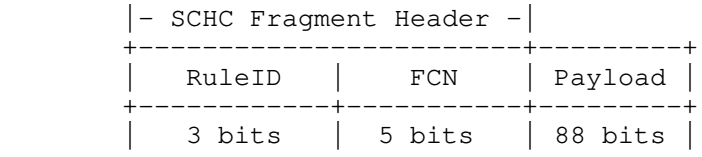


Figure 3: Regular SCHC Fragment format for all fragments except the last one

3.6.1.2. All-1 SCHC Fragment

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

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

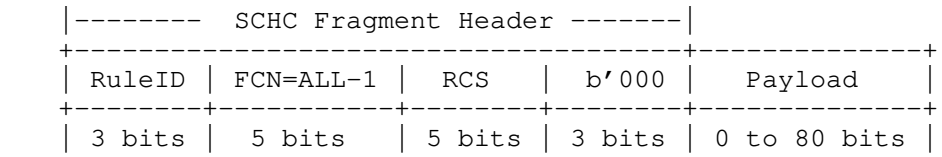


Figure 4: All-1 SCHC Message format with last tile

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

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

3.6.1.3. SCHC Sender-Abort Message format

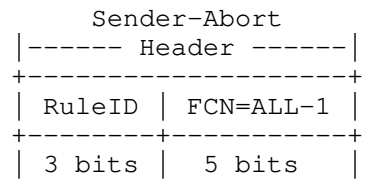


Figure 5: SCHC Sender-Abort message format

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

3.6.2.1. Regular SCHC Fragment

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

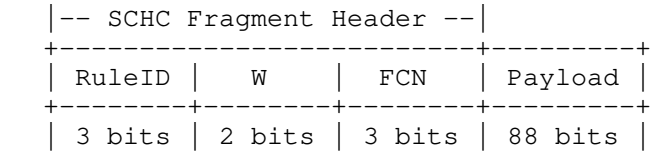


Figure 6: Regular SCHC Fragment format for all fragments except the last one

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

3.6.2.2. All-1 SCHC Fragment

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

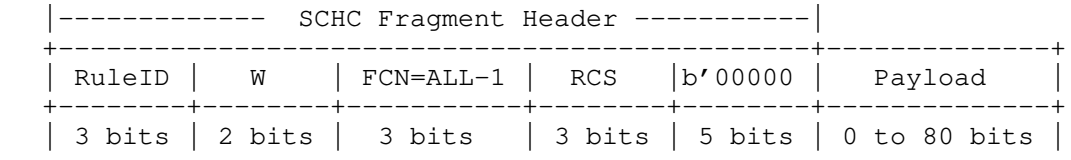


Figure 7: All-1 SCHC Message format with last tile

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

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

3.6.2.3. SCHC ACK Format

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

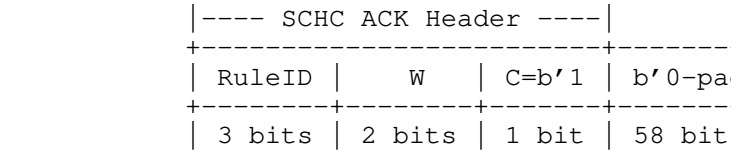
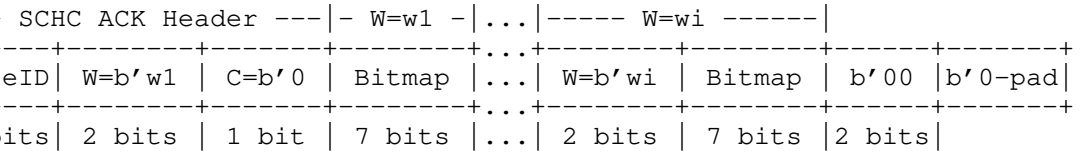


Figure 8: SCHC Success ACK message format

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



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

Figure 9: SCHC Compound ACK message format

3.6.2.4. SCHC Sender-Abort Message format

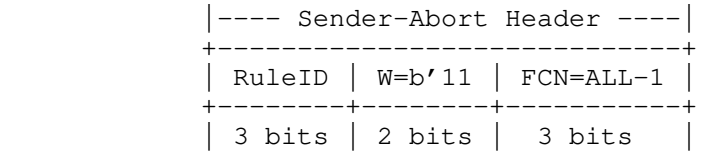


Figure 10: SCHC Sender-Abort message format

3.6.2.5. SCHC Receiver-Abort Message format

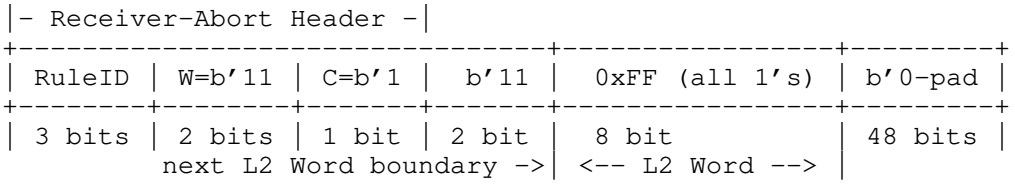


Figure 11: SCHC Receiver-Abort message format

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

3.6.3.1. Regular SCHC Fragment

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

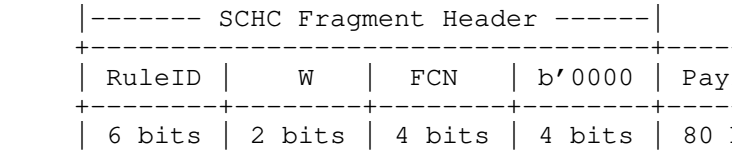


Figure 12: Regular SCHC Fragment format for all fragments except the last one

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

3.6.3.2. All-1 SCHC Fragment

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

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

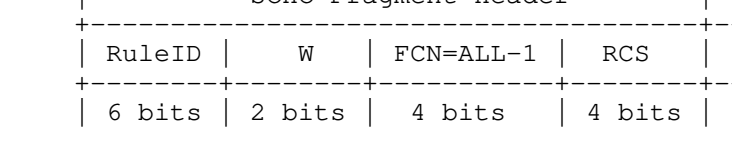


Figure 13: All-1 SCHC message format with last tile

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

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

3.6.3.3. SCHC ACK Format

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

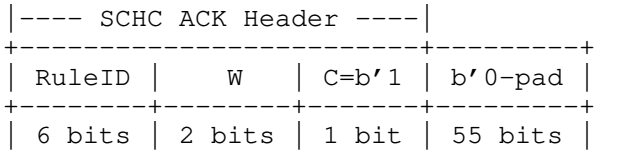
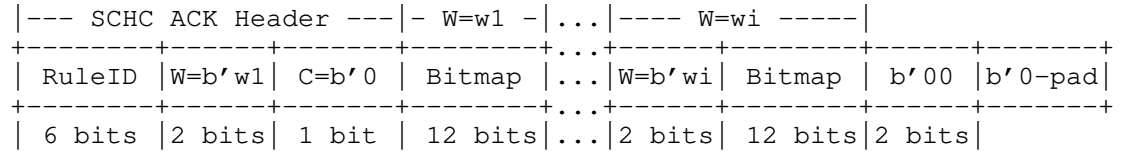


Figure 14: SCHC Success ACK message format

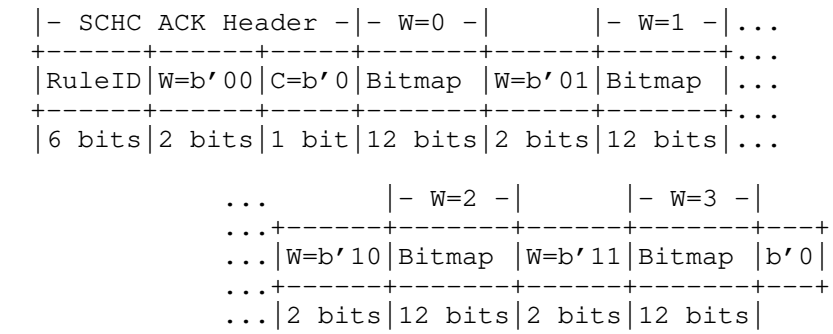
The SCHC Compound ACK message MUST be used in case SCHC fragment losses are found in any window of the SCHC Packet (C=0). The SCHC Compound ACK message format is shown in Figure 15. The SCHC Compound ACK can report up to 4 windows with losses. as shown in Figure 16.

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



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

Figure 15: SCHC Compound ACK message format



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

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

3.6.3.4. SCHC Sender-Abort Message Format

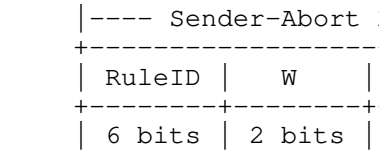


Figure 17: SCHC Sender-Abort message format

3.6.3.5. SCHC Receiver-Abort Message Format

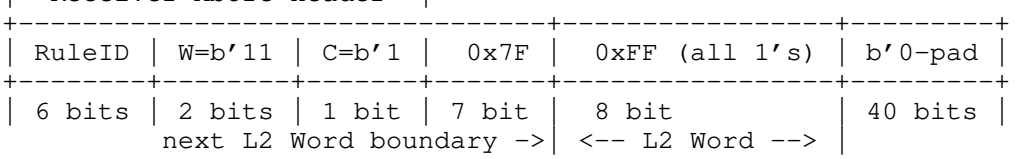


Figure 18: SCHC Receiver-Abort message format

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

3.6.4.1. Regular SCHC Fragment

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

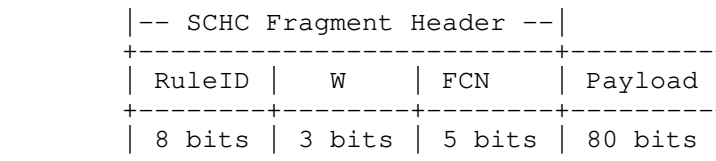


Figure 19: Regular SCHC Fragment format for all fragments except the last one

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

3.6.4.2. All-1 SCHC Fragment

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

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

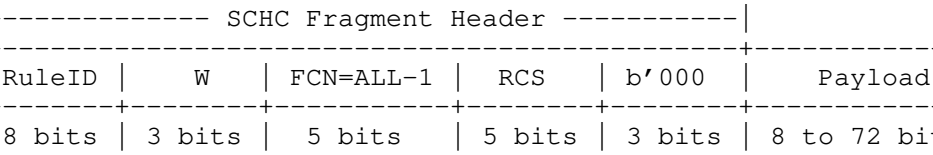


Figure 20: All-1 SCHC message format with last tile

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

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

3.6.4.3. SCHC ACK Format

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

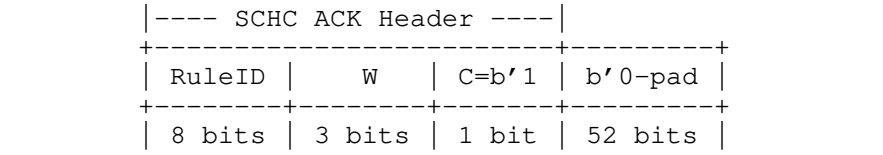
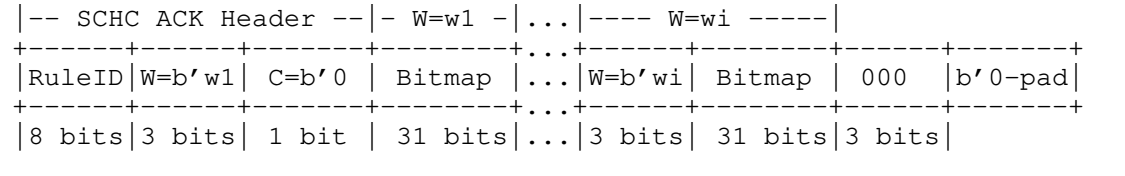


Figure 21: SCHC Success ACK message format

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

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



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

Figure 22: SCHC Compound ACK message format

3.6.4.4. SCHC Sender-Abort Message Format

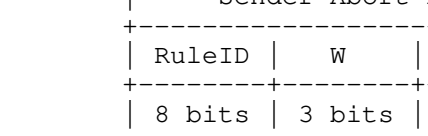


Figure 23: SCHC Sender-Abort message format

3.6.4.5. SCHC Receiver-Abort Message Format

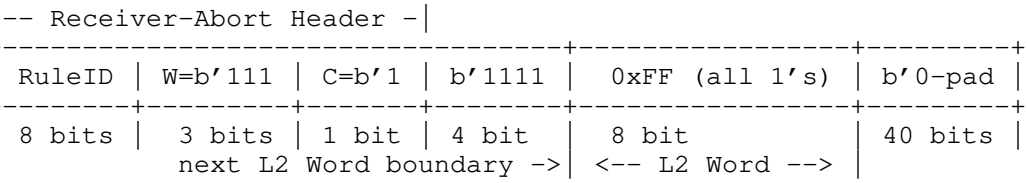


Figure 24: SCHC Receiver-Abort message format

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

3.6.5.1. Regular SCHC Fragment

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

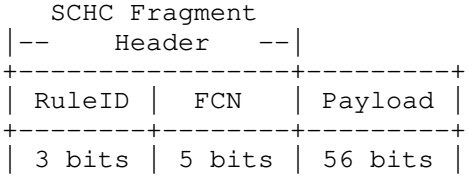


Figure 25: Regular SCHC Fragment format for all fragments except the last one

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

3.6.5.2. All-1 SCHC Fragment

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

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

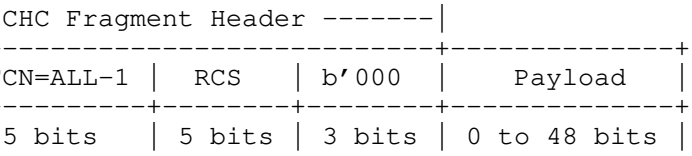


Figure 26: All-1 SCHC message format with last tile

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

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

3.6.5.3. SCHC ACK Format

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

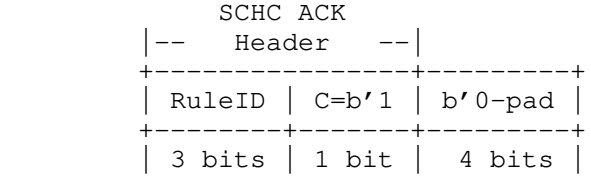


Figure 27: SCHC Success ACK message format

The SCHC ACK message format is shown in Figure 28.

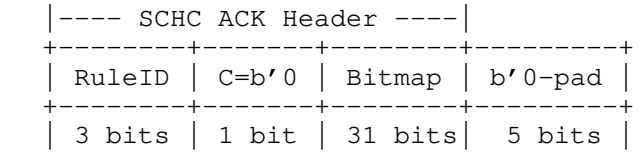


Figure 28: SCHC Compound ACK message format

3.6.5.4. SCHC Sender-Abort Message Format

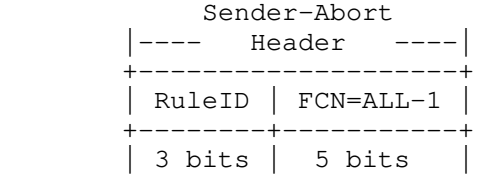


Figure 29: SCHC Sender-Abort message format

3.6.5.5. SCHC Receiver-Abort Message Format

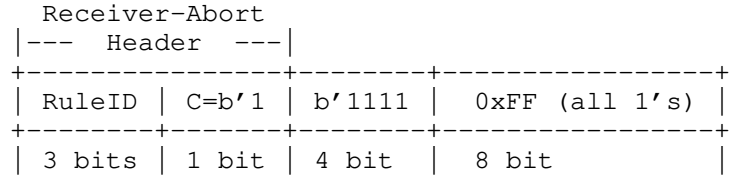


Figure 30: SCHC Receiver-Abort message format

3.7. Padding

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

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

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

4. Fragmentation Rules Examples

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

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

4.1. Uplink Fragmentation Rules Examples

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

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

* For Single-byte SCHC Header F/R modes, it is expected a RuleID field of 3 bits:
  - if the first 3 leftmost bits have a value different than 0b'111, then it signals a Single-byte SCHC Header F/R mode,
  - if their value is 0b'111, then it signals a Two-byte SCHC Header F/R mode.
* For Single-byte SCHC Header F/R modes:
  - there are 7 RuleIDs available (with values from 0b'000-0b'110), the RuleID with value 0b'111 is reserved to indicate a Two-byte SCHC Header.
  - This set of rules is called "standard rules", and it is used to implement Single-byte SCHC header modes.
  - Each RuleID is associated with a set of properties defining if Uplink F/R is used and which Uplink F/R mode is used. As an example, the RuleID 0b'000 is mapped onto Uplink No-ACK Mode: Single-byte SCHC Header, and the RuleIDs 0b'001 and 0b'002 are mapped onto Uplink ACK-on-Error Mode: Single-byte SCHC Header (2 RuleIDs to allow for SCHC Packet interleaving).
* For Two-byte SCHC Header F/R modes at least 6 bits for the RuleID field are expected:
  - the 3 first leftmost bits are always 0b'111,
    - if the following 3 bits have a different value than 0b'111, then it signals the Two-byte SCHC Header Option 1,
    - if the following 3 bits are 0b'111, then it signals the Two-byte SCHC Header Option 2.
  - For the Two-byte SCHC Header Option 1, there are 7 RuleIDs available (0b'111000-0b'111110), 0b'111111 being reserved to indicate the Two-byte SCHC Header Option 2. This set of rules is called "extended rules", and it is used to implement the Uplink ACK-on-Error Mode: Two-byte SCHC Header Option 1.

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

4.2. Downlink Fragmentation Rules Example

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

5. Fragmentation Sequence Examples

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

5.1. Uplink No-ACK Examples

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

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

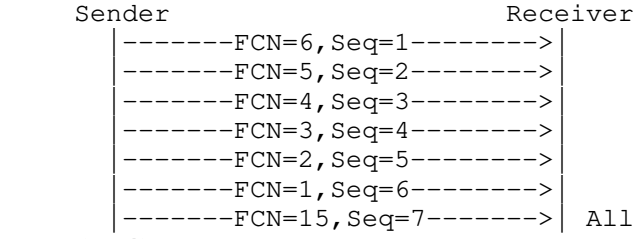


Figure 31: Uplink No-ACK No-Losses

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

Case losses on any fragment except the first.

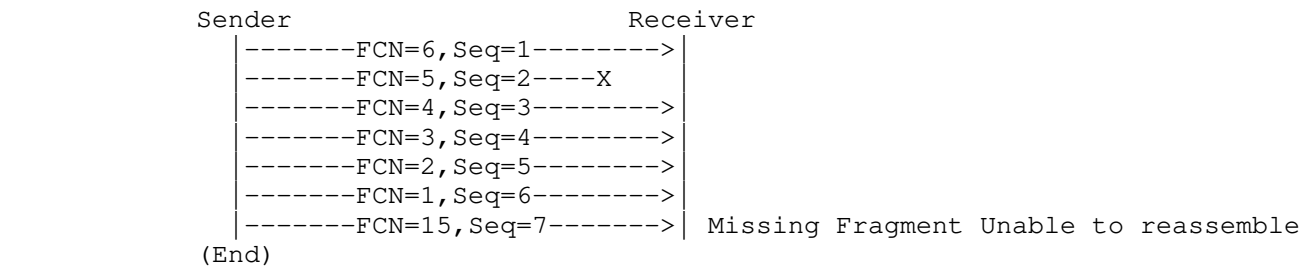


Figure 32: Uplink No-ACK Losses (scenario 1)

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

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

Case No losses

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

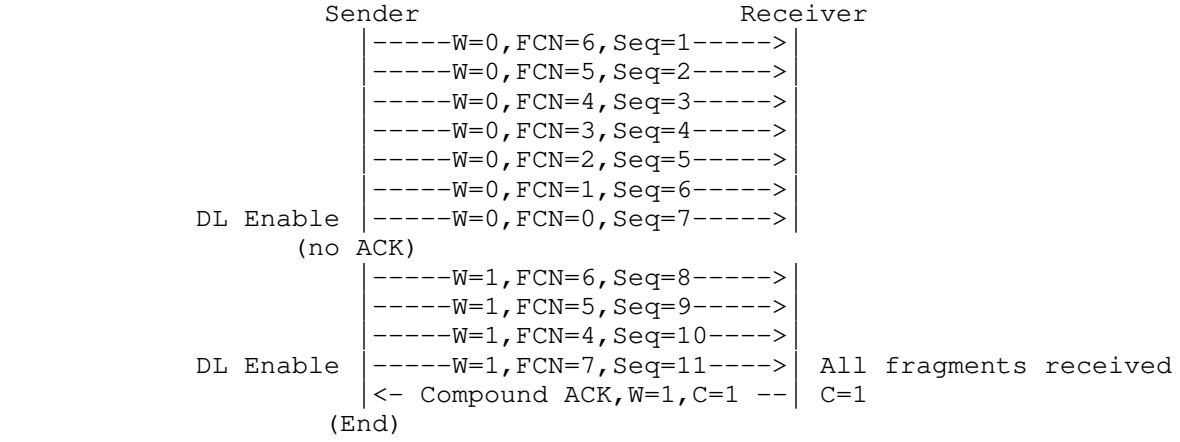


Figure 33: Uplink ACK-on-Error No-Losses

Case Fragment losses in first window

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

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

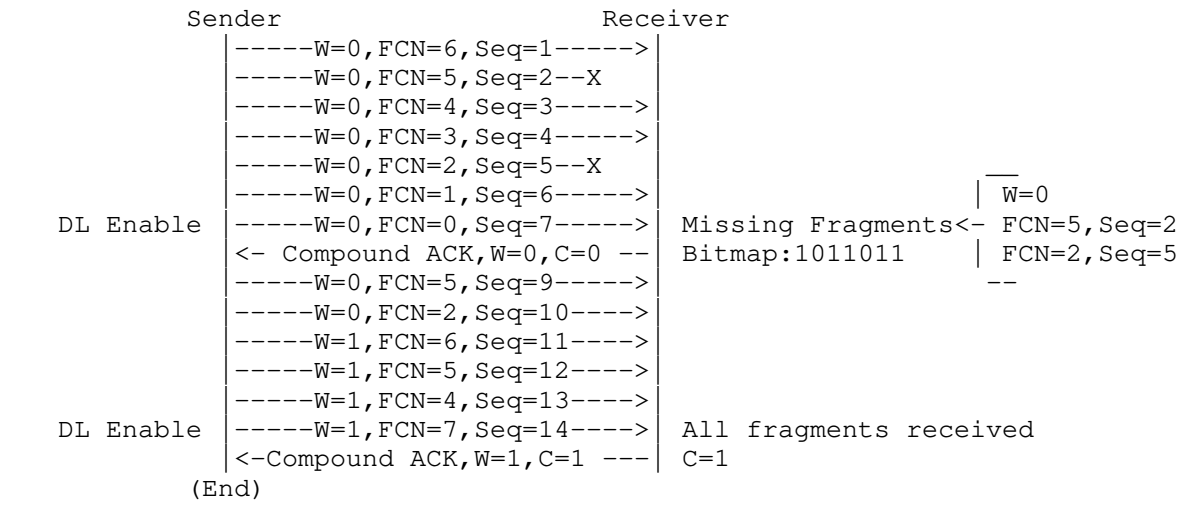


Figure 34: Uplink ACK-on-Error Losses on First Window

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

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

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

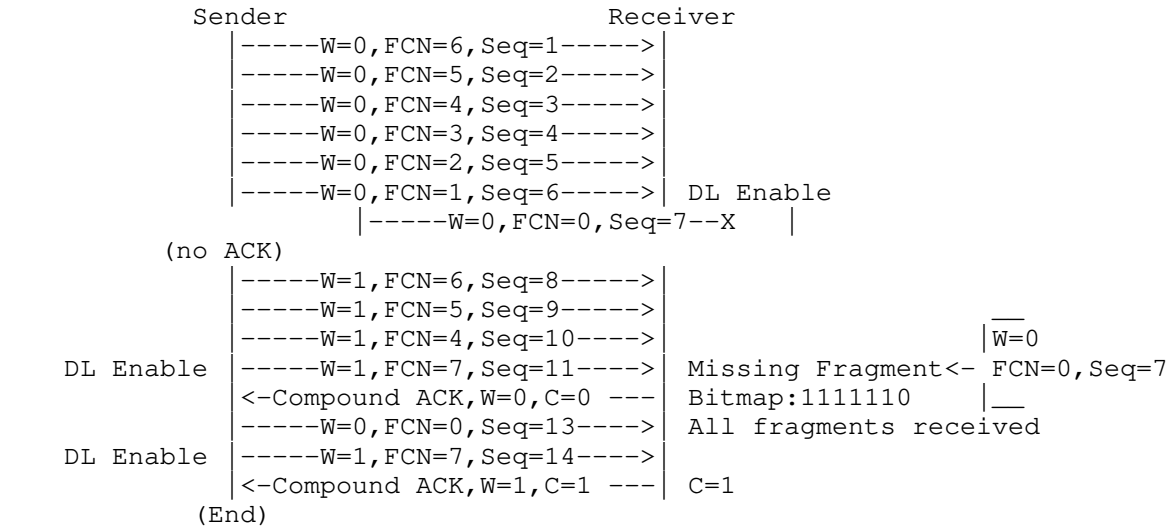


Figure 35: Uplink ACK-on-Error All-0 Lost on First Window

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

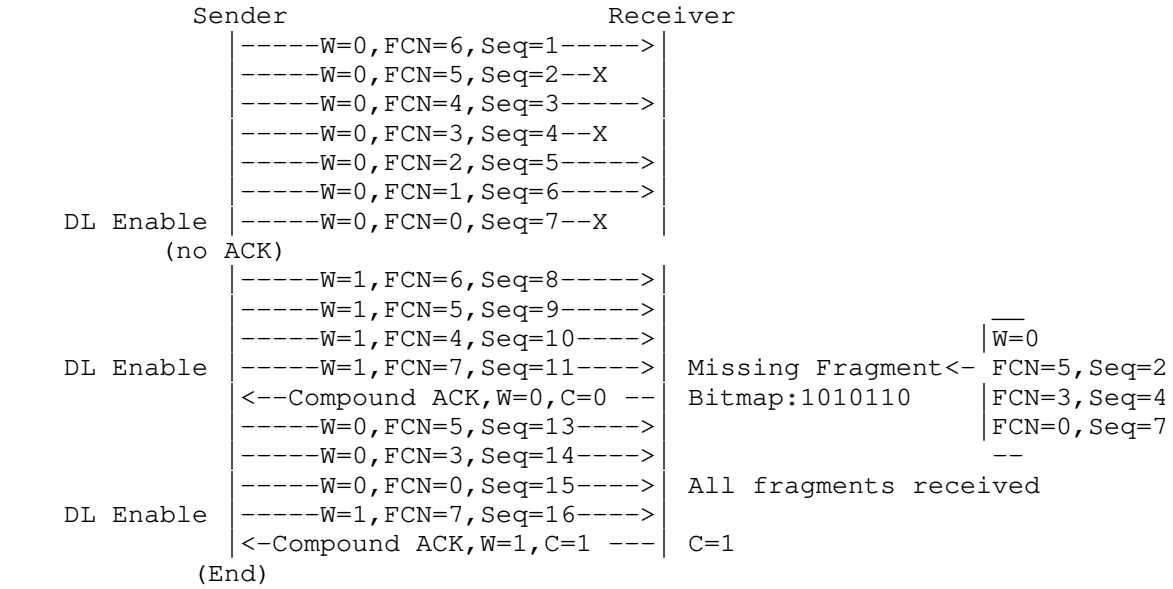


Figure 36: Uplink ACK-on-Error All-0 and other Fragments Lost on First Window

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

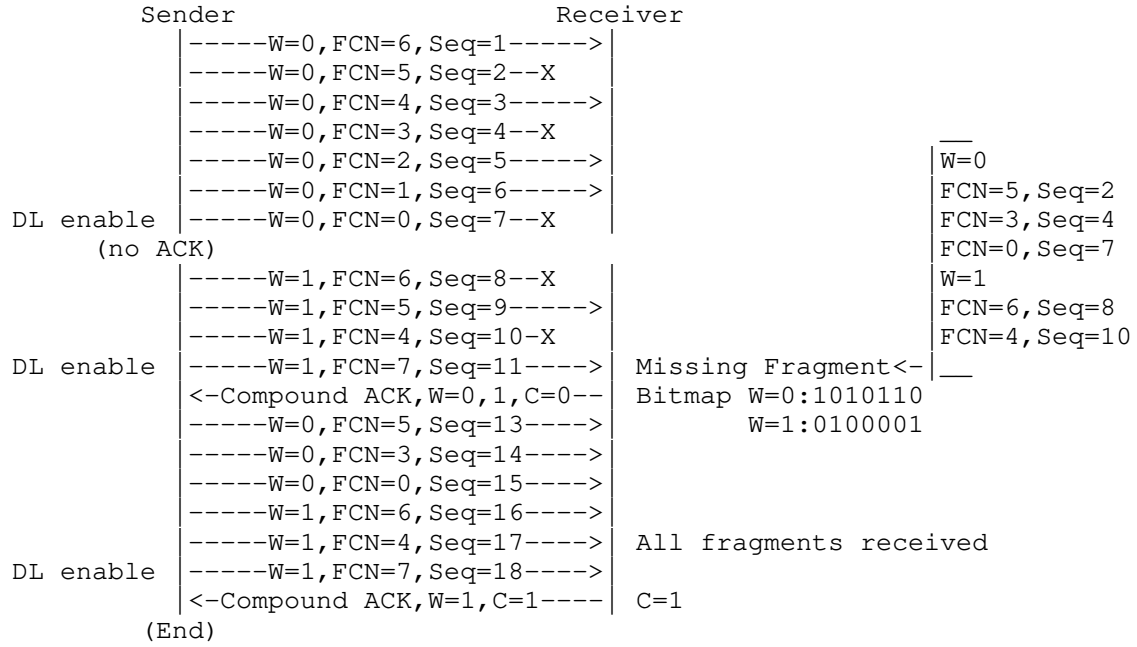


Figure 37: Uplink ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (1)

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

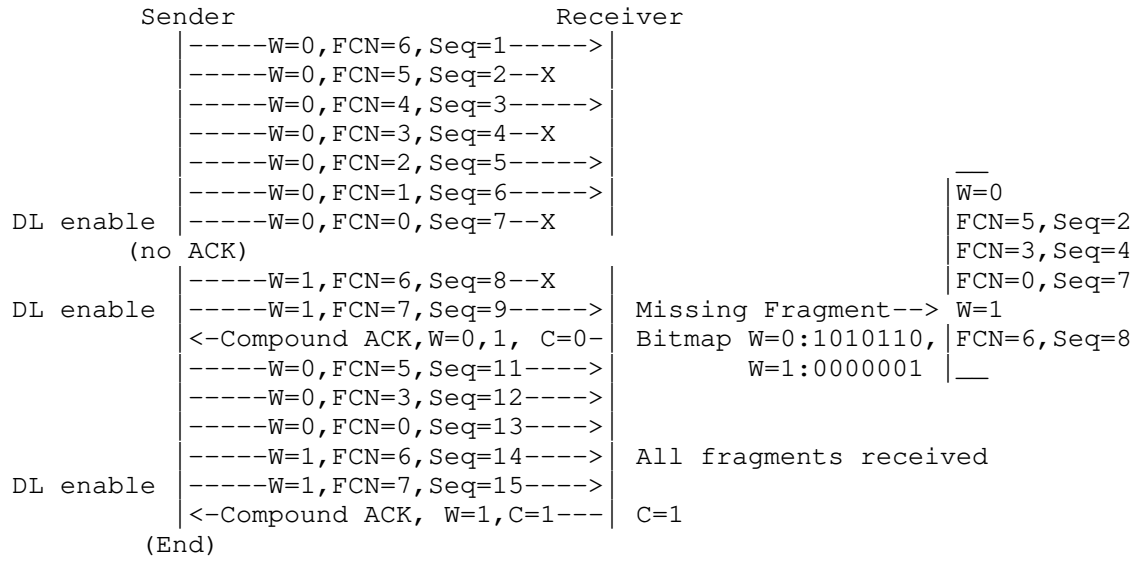


Figure 38: Uplink ACK-on-Error All-0 and other Fragments Lost on First and Second Windows (2)

Case SCHC ACK is lost

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

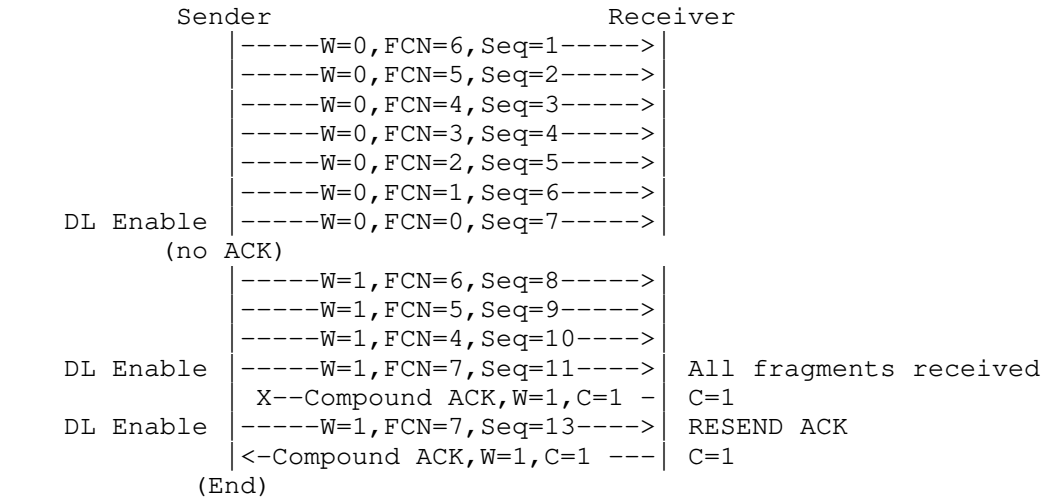


Figure 39: Uplink ACK-on-Error ACK Lost

Case SCHC Compound ACK at the end

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

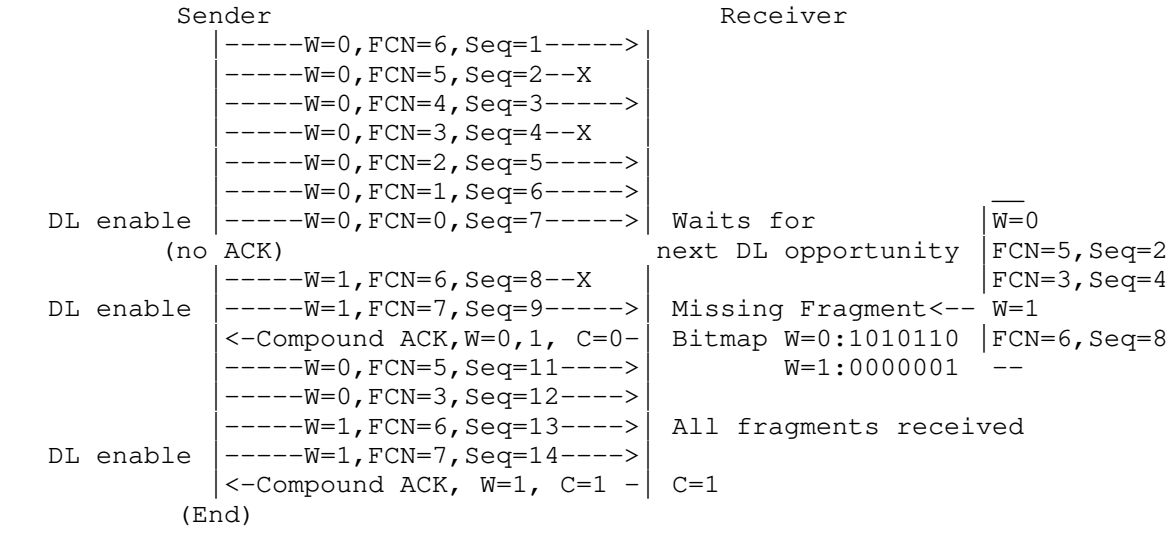


Figure 40: Uplink ACK-on-Error Fragments Lost on First and Second Windows with one Compound ACK

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

5.3. SCHC Abort Examples

Case SCHC Sender-Abort

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

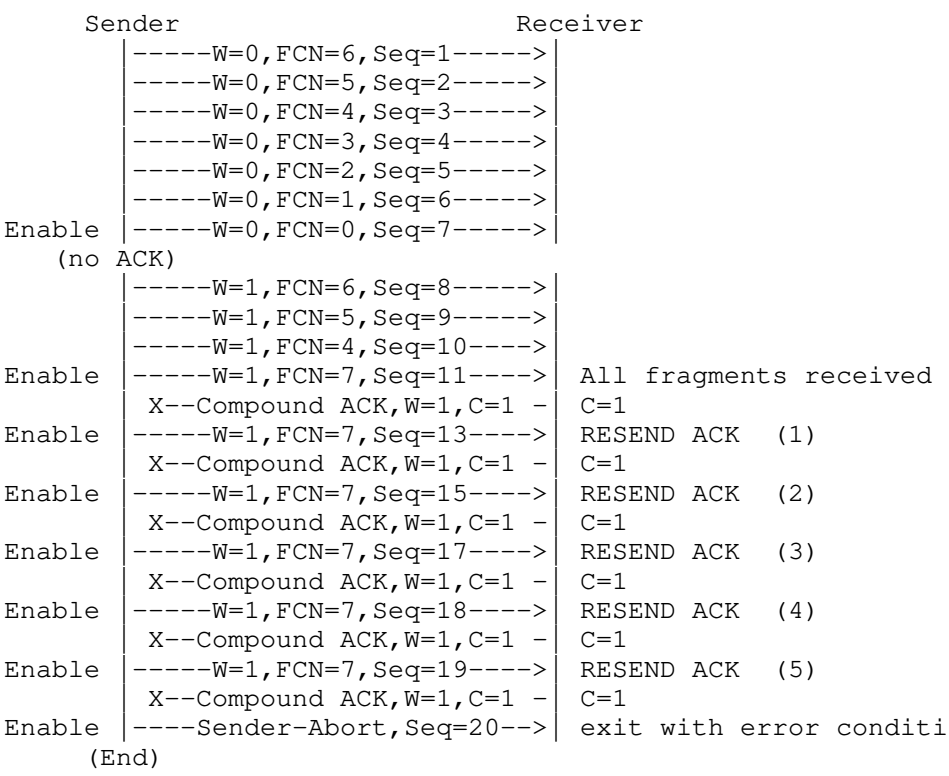


Figure 41: Uplink ACK-on-Error Sender-Abort

Case Receiver-Abort

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

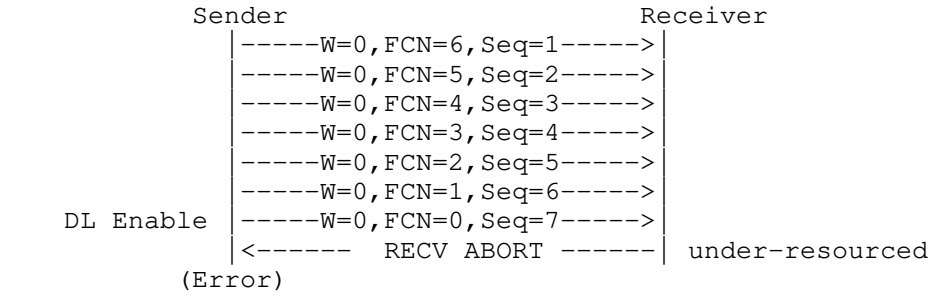


Figure 42: Uplink ACK-on-Error Receiver-Abort

6. Security considerations

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

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

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

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

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

7. IANA Considerations

This document has no IANA actions.

8. Acknowledgements

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

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

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

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

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

9. References

9.1. Normative References

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

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

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

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

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

9.2. Informative References

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

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

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

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

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

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

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

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

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

Authors' Addresses

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

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

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

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

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

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

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

lpwan Working Group
Internet-Draft
Intended status: Standards Track
Expires: 12 April 2023

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

9 October 2022

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

Abstract

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

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

Status of This Memo

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 12 April 2023.

Copyright Notice

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

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

Table of Contents

1. Introduction 3
2. Requirements Language 3
3. Terminology 3
4. SCHC rules 5
4.1. Compression Rules 5
4.2. Identifier generation 6
4.3. Convention for Field Identifier 7
4.4. Convention for Field length 8
4.5. Convention for Field position 8
4.6. Convention for Direction Indicator 8
4.7. Convention for Target Value 9
4.8. Convention for Matching Operator 9
4.8.1. Matching Operator arguments 9
4.9. Convention for Compression Decompression Actions 9
4.9.1. Compression Decompression Action arguments 9
4.10. Fragmentation rule 9
4.10.1. Fragmentation mode 10
4.10.2. Fragmentation Header 10
4.10.3. Last fragment format 11
4.10.4. Acknowledgment behavior 11
4.10.5. Timer values 12
4.10.6. Fragmentation Parameter 12
4.10.7. Layer 2 parameters 12
5. Rule definition 13
5.1. Compression rule 13
5.2. Fragmentation rule 14
5.3. YANG Tree 14
6. YANG Module 15
7. Implementation Status 45
8. IANA Considerations 46
8.1. URI Registration 46
8.2. YANG Module Name Registration 46
9. Security Considerations 47
10. Annex A : Example 48
11. Acknowledgements 51
12. References 51
12.1. Normative References 51

12.2. Informative References 53
Authors' Addresses 54

1. Introduction

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

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

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

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

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

2. Requirements Language

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

3. Terminology

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

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

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

* CDA: Compression/Decompression Action. Describes the pair of actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field.

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

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

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

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

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

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

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

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

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

* L2 Word: this is the minimum subdivision of payload data that the L2 will carry. In most L2 technologies, the L2 Word is an octet. In bit-oriented radio technologies, the L2 Word might be a single bit. The L2 Word size is assumed to be constant over time for each device.

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

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

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

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

4. SCHC rules

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

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

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

4.1. Compression Rules

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

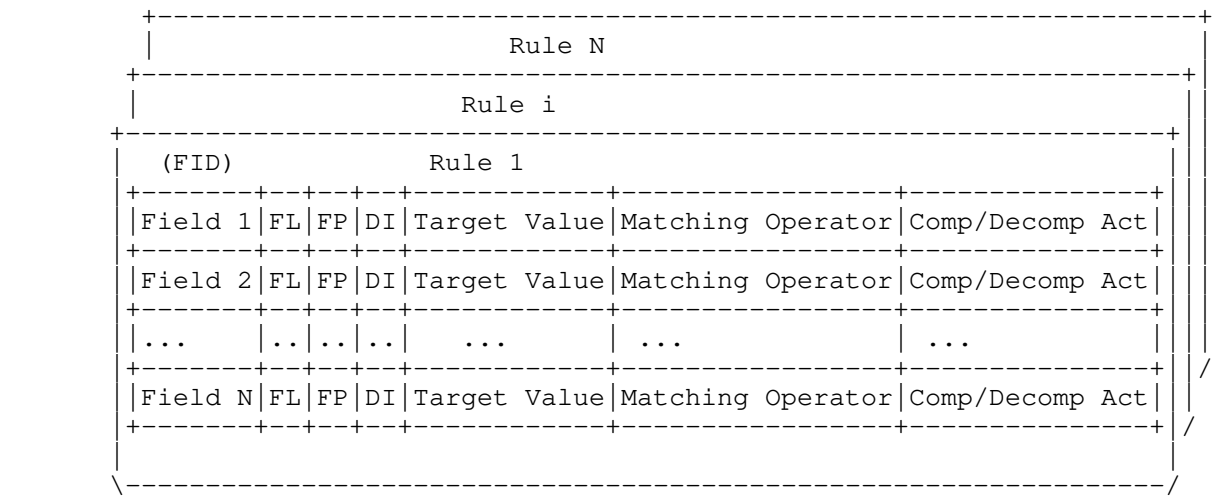


Figure 1: Compression Decompression Context

4.2. Identifier generation

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

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

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

* create a typedef from this base type.

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

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

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

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

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

4.3. Convention for Field Identifier

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

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

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

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

4.4. Convention for Field length

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

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

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

4.5. Convention for Field position

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

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

4.6. Convention for Direction Indicator

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

The type is "di-type".

4.7. Convention for Target Value

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

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

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

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

4.8. Convention for Matching Operator

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

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

The type is "mo-type"

4.8.1. Matching Operator arguments

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

4.9. Convention for Compression Decompression Actions

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

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

4.9.1. Compression Decompression Action arguments

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

4.10. Fragmentation rule

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

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

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

4.10.1. Fragmentation mode

[RFC8724] defines 3 fragmentation modes:

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

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

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

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

4.10.2. Fragmentation Header

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

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

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

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

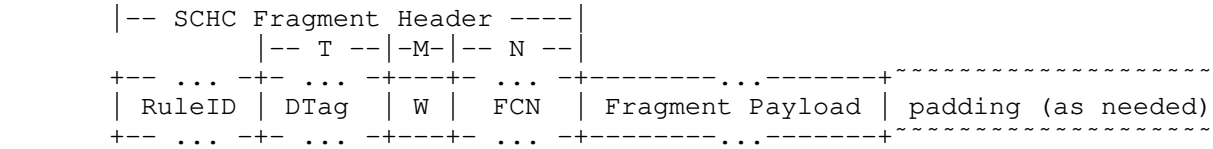


Figure 3: Data fragment header from RFC8724

4.10.3. Last fragment format

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

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

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

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

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

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

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

4.10.4. Acknowledgment behavior

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

* a Dtag (if present).

* a mandatory window as in the data fragment.

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

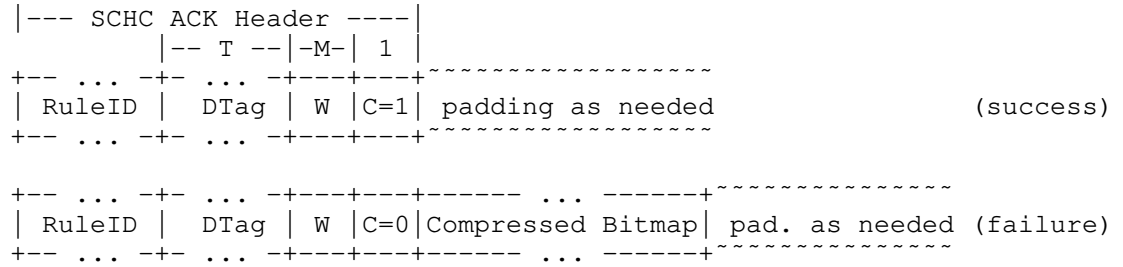


Figure 4: Acknowledgment fragment header for RFC8724

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

4.10.5. Timer values

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

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

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

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

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

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

4.10.6. Fragmentation Parameter

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

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

4.10.7. Layer 2 parameters

The data model includes two parameters needed for fragmentation:

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

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

They are defined as unsigned integers, see Section 6.

5. Rule definition

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

Three natures of rules are defined in [RFC8724]:

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

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

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

The YANG data model introduces respectively these three identities :

* nature-compression

* nature-no-compression

* nature-fragmentation

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

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

5.1. Compression rule

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

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

Some checks are performed on the values:

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

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

5.2. Fragmentation rule

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

5.3. YANG Tree

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

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

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

Figure 5: Overview of SCHC data model

6. YANG Module

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

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

```
description
  "
```

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

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

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

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

****

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

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

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

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

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

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

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

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

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

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

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

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

    identity fid-ipv6-trafficclass {
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    }

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

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

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

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

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

    identity fl-base-type {
      description
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    // -- type definition

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

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

    typedef di-type {
      type identityref {
        base di-base-type;
      }
      description
        "Direction in LPWAN network, up when emitted by the device,
         down when received by the device, bi when emitted or
         received by the device.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
```

```
                Compression and Fragmentation";
    }

    typedef mo-type {
      type identityref {
        base mo-base-type;
      }
      description
        "Matching Operator (MO) to compare fields values with
         target values.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    typedef cda-type {
      type identityref {
        base cda-base-type;
      }
      description
        "Compression Decompression Action to compression or
         decompress a field.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

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

    identity fragmentation-mode-base-type {
      description
        "Define the fragmentation mode.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    identity fragmentation-mode-no-ack {
      base fragmentation-mode-base-type;
      description
        "No-ACK mode.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    identity fragmentation-mode-ack-always {
```

```
      base fragmentation-mode-base-type;
      description
        "ACK-Always mode.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    identity fragmentation-mode-ack-on-error {
      base fragmentation-mode-base-type;
      description
        "ACK-on-Error mode.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    typedef fragmentation-mode-type {
      type identityref {
        base fragmentation-mode-base-type;
      }
      description
        "Define the type used for fragmentation mode in rules.";
    }

    // -- Ack behavior

    identity ack-behavior-base-type {
      description
        "Define when to send an Acknowledgment .";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    identity ack-behavior-after-all-0 {
      base ack-behavior-base-type;
      description
        "Fragmentation expects Ack after sending All-0 fragment.";
    }

    identity ack-behavior-after-all-1 {
      base ack-behavior-base-type;
      description
        "Fragmentation expects Ack after sending All-1 fragment.";
    }

    identity ack-behavior-by-layer2 {
```

```
      base ack-behavior-base-type;
      description
        "Layer 2 defines when to send an Ack.";
    }

    typedef ack-behavior-type {
      type identityref {
        base ack-behavior-base-type;
      }
      description
        "Define the type used for Ack behavior in rules.";
    }

    // -- All-1 with data types

    identity all-1-data-base-type {
      description
        "Type to define when to send an Acknowledgment message.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    identity all-1-data-no {
      base all-1-data-base-type;
      description
        "All-1 contains no tiles.";
    }

    identity all-1-data-yes {
      base all-1-data-base-type;
      description
        "All-1 MUST contain a tile.";
    }

    identity all-1-data-sender-choice {
      base all-1-data-base-type;
      description
        "Fragmentation process chooses to send tiles or not in All-1.";
    }

    typedef all-1-data-type {
      type identityref {
        base all-1-data-base-type;
      }
      description
        "Define the type used for All-1 format in rules.";
    }
```

```
     // -- RCS algorithm types

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

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

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

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

     grouping tv-struct {
       description
         "Defines the target value element. If the header field
          contains a text, the binary sequence uses the same encoding.
          field-id allows the conversion to the appropriate type.";
       leaf index {
         type uint16;
         description
           "Index gives the position in the matching-list. If only one
            element is present, index is 0. Otherwise, index is the
            the order in the matching list, starting at 0.";
       }
       leaf value {
         type binary;
         description
           "Target Value content as an untyped binary value.";
       }
```

```
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }

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

      +-------+--+--+--+------------+-----------------+---------------+
      |Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|
      +-------+--+--+--+------------+-----------------+---------------+

          An entry in a compression rule is composed of 7 elements:
          - Field ID: The header field to be compressed.
          - Field Length : Either a positive integer of a function.
          - Field Position: A positive (and possibly equal to 0)
            integer.
          - Direction Indicator: An indication in which direction
            compression and decompression process is effective.
          - Target value: A value against which the header Field is
            compared.
          - Matching Operator: The comparison operation and optional
            associate parameters.
          - Comp./Decomp. Action: The compression or decompression
            action, and optional parameters.
         ";
       leaf field-id {
         type schc:fid-type;
         mandatory true;
         description
           "Field ID, identify a field in the header with a YANG
            identity reference.";
       }
       leaf field-length {
         type schc:fl-type;
         mandatory true;
         description
           "Field Length, expressed in number of bits if the length is
            known when the Rule is created or through a specific
            function if the length is variable.";
       }
       leaf field-position {
         type uint8;
         mandatory true;
         description
           "Field position in the header is an integer. Position 1
```

```
            matches the first occurrence of a field in the header,
            while incremented position values match subsequent
            occurrences.
            Position 0 means that this entry matches a field
            irrespective of its position of occurrence in the
            header.
            Be aware that the decompressed header may have
            position-0 fields ordered differently than they
            appeared in the original packet.";
        }
        leaf direction-indicator {
          type schc:di-type;
          mandatory true;
          description
            "Direction Indicator, indicate if this field must be
             considered for rule selection or ignored based on the
             direction (bi directionnal, only uplink, or only
             downlink).";
        }
        list target-value {
          key "index";
          uses tv-struct;
          description
            "A list of value to compare with the header field value.
             If target value is a singleton, position must be 0.
             For use as a matching list for the mo-match-mapping matching
             operator, index should take consecutive values starting
             from 0.";
        }
        leaf matching-operator {
          type schc:mo-type;
          must "../target-value or derived-from-or-self(.,
                                                  'mo-ignore')" {
            error-message
              "mo-equal, mo-msb and mo-match-mapping need target-value";
            description
              "target-value is not required for mo-ignore.";
          }
          must "not (derived-from-or-self(., 'mo-msb')) or
                ../matching-operator-value" {
            error-message "mo-msb requires length value";
          }
          mandatory true;
          description
            "MO: Matching Operator.";
          reference
            "RFC 8724 SCHC: Generic Framework for Static Context Header
                      Compression and Fragmentation (see Section 7.3).";
```

```
      }
      list matching-operator-value {
        key "index";
        uses tv-struct;
        description
          "Matching Operator Arguments, based on TV structure to allow
           several arguments.
           In RFC 8724, only the MSB matching operator needs arguments
           (a single argument, which is the number of most significant
           bits to be matched).";
      }
      leaf comp-decomp-action {
        type schc:cda-type;
        must "../target-value or
                  derived-from-or-self(., 'cda-value-sent') or
                  derived-from-or-self(., 'cda-compute') or
                  derived-from-or-self(., 'cda-appiid') or
                  derived-from-or-self(., 'cda-deviid')" {
          error-message
            "cda-not-sent, cda-lsb, cda-mapping-sent need
             target-value";
          description
            "target-value is not required for some CDA.";
          }
        mandatory true;
        description
          "CDA: Compression Decompression Action.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 7.4)";
      }
      list comp-decomp-action-value {
        key "index";
        uses tv-struct;
        description
          "CDA arguments, based on a TV structure, in order to allow
           for several arguments. The CDAs specified in RFC 8724
           require no argument.";
      }

    }

    // --Rule nature

    identity nature-base-type {
      description
        "A rule, identified by its RuleID, are used for a single
         purpose. RFC 8724 defines 2 natures:
```

```
          compression, no compression and fragmentation.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-compression {
        base nature-base-type;
        description
          "Identify a compression rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-no-compression {
        base nature-base-type;
        description
          "Identify a no compression rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      identity nature-fragmentation {
        base nature-base-type;
        description
          "Identify a fragmentation rule.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation (see section 6).";
      }

      typedef nature-type {
        type identityref {
          base nature-base-type;
        }
        description
          "defines the type to indicate the nature of the rule.";
      }

      grouping compression-content {
        list entry {
          must "derived-from-or-self(../rule-nature,
                                     'nature-compression')" {
            error-message "Rule nature must be compression";
          }
          key "field-id field-position direction-indicator";
```

```
        uses compression-rule-entry;
        description
          "A compression rule is a list of rule entries, each
           describing a header field. An entry is identified
           through a field-id, its position in the packet, and
           its direction.";
      }
      description
        "Define a compression rule composed of a list of entries.";
      reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
                  Compression and Fragmentation";
    }

    grouping fragmentation-content {
      description
        "This grouping defines the fragmentation parameters for
         all the modes (No-ACK, ACK-Always and ACK-on-Error) specified
         in RFC 8724.";
      leaf fragmentation-mode {
        type schc:fragmentation-mode-type;
        must "derived-from-or-self(../rule-nature,
                                   'nature-fragmentation')" {
          error-message "Rule nature must be fragmentation";
        }
        mandatory true;
        description
          "Which fragmentation mode is used (No-Ack, ACK-Always,
           ACK-on-Error).";
      }
      leaf l2-word-size {
        type uint8;
        default "8";
        description
          "Size, in bits, of the layer 2 word.";
      }
      leaf direction {
        type schc:di-type;
        must "derived-from-or-self(., 'di-up') or
              derived-from-or-self(., 'di-down')" {
          error-message
            "Direction for fragmentation rules are up or down.";
        }
        mandatory true;
        description
          "MUST be up or down, bidirectional MUST NOT be used.";
      }
      // SCHC Frag header format
```

```
      leaf dtag-size {
        type uint8;
        default "0";
        description
          "Size, in bits, of the DTag field (T variable from
           RFC8724).";
      }
      leaf w-size {
        when "derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-on-error')
              or
              derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-always') ";
        type uint8;
        description
          "Size, in bits, of the window field (M variable from
           RFC8724).";
      }
      leaf fcn-size {
        type uint8;
        mandatory true;
        description
          "Size, in bits, of the FCN field (N variable from RFC8724).";
      }
      leaf rcs-algorithm {
        type rcs-algorithm-type;
        default "schc:rcs-crc32";
        description
          "Algorithm used for RCS. The algorithm specifies the RCS
           size.";
      }
      // SCHC fragmentation protocol parameters
      leaf maximum-packet-size {
        type uint16;
        default "1280";
        description
          "When decompression is done, packet size must not
           strictly exceed this limit, expressed in bytes.";
      }
      leaf window-size {
        type uint16;
        description
          "By default, if not specified 2^w-size - 1. Should not exceed
           this value. Possible FCN values are between 0 and
           window-size - 1.";
      }
      leaf max-interleaved-frames {
        type uint8;
```

```
          default "1";
          description
            "Maximum of simultaneously fragmented frames. Maximum value
             is 2^dtag-size. All DTAG values can be used, but more than
             max-interleaved-frames MUST NOT be active at any time";
        }
        container inactivity-timer {
          leaf ticks-duration {
            type uint8;
            default "20";
            description
              "Duration of one tick in micro-seconds:
                  2^ticks-duration/10^6 = 1.048s.";
          }
          leaf ticks-numbers {
            type uint16 {
              range "0..max";
            }
            description
              "Timer duration = ticks-numbers*2^ticks-duration / 10^6.";
          }

          description
            "Duration is seconds of the inactivity timer, 0 indicates
             that the timer is disabled.

             Allows a precision from micro-second to year by sending the
             tick-duration value. For instance:

           tick-duration /  smallest value          highest value
           v
           20: 00y 000d 00h 00m 01s.048575<->00y 000d 19h 05m 18s.428159
           21: 00y 000d 00h 00m 02s.097151<->00y 001d 14h 10m 36s.856319
           22: 00y 000d 00h 00m 04s.194303<->00y 003d 04h 21m 13s.712639
           23: 00y 000d 00h 00m 08s.388607<->00y 006d 08h 42m 27s.425279
           24: 00y 000d 00h 00m 16s.777215<->00y 012d 17h 24m 54s.850559
           25: 00y 000d 00h 00m 33s.554431<->00y 025d 10h 49m 49s.701119

            Note that the smallest value is also the incrementation step,
            so the timer precision.";
        }
        container retransmission-timer {
          leaf ticks-duration {
            type uint8;
            default "20";
            description
              "Duration of one tick in micro-seconds:
                  2^ticks-duration/10^6 = 1.048s.";
```

```
        }
        leaf ticks-numbers {
          type uint16 {
            range "1..max";
          }
          description
            "Timer duration = ticks-numbers*2^ticks-duration / 10^6.";
        }

        when "derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-on-error')
              or
              derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-always') ";
        description
          "Duration in seconds of the retransmission timer.
           See inactivity timer.";
      }
      leaf max-ack-requests {
        when "derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-on-error')
              or
              derived-from-or-self(../fragmentation-mode,
                                   'fragmentation-mode-ack-always') ";
        type uint8 {
          range "1..max";
        }
        description
          "The maximum number of retries for a specific SCHC ACK.";
      }
      choice mode {
        case no-ack;
        case ack-always;
        case ack-on-error {
          leaf tile-size {
            when "derived-from-or-self(../fragmentation-mode,
                                'fragmentation-mode-ack-on-error')";
            type uint8;
            description
              "Size, in bits, of tiles. If not specified or set to 0,
               tiles fill the fragment.";
          }
          leaf tile-in-all-1 {
            when "derived-from-or-self(../fragmentation-mode,
                                'fragmentation-mode-ack-on-error')";
            type schc:all-1-data-type;
            description
              "Defines whether the sender and receiver expect a tile in
```

```
                All-1 fragments or not, or if it is left to the sender's
                choice.";
            }
            leaf ack-behavior {
              when "derived-from-or-self(../fragmentation-mode,
                                'fragmentation-mode-ack-on-error')";
              type schc:ack-behavior-type;
              description
                "Sender behavior to acknowledge, after All-0, All-1 or
                 when the LPWAN allows it.";
            }
          }
          description
            "RFC 8724 defines 3 fragmentation modes.";
        }
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation";
      }

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

      grouping rule-id-type {
        leaf rule-id-value {
          type uint32;
          description
            "Rule ID value, this value must be unique, considering its
             length.";
        }
        leaf rule-id-length {
          type uint8 {
            range "0..32";
          }
          description
            "Rule ID length, in bits. The value 0 is for implicit
             rules.";
        }
        description
          "A rule ID is composed of a value and a length, expressed in
           bits.";
        reference
          "RFC 8724 SCHC: Generic Framework for Static Context Header
                    Compression and Fragmentation";
      }

      // SCHC table for a specific device.
```

```
     container schc {
       list rule {
         key "rule-id-value rule-id-length";
         uses rule-id-type;
         leaf rule-nature {
           type nature-type;
           mandatory true;
           description
             "Specify the rule's nature.";
         }
         choice nature {
           case fragmentation {
             if-feature "fragmentation";
             uses fragmentation-content;
           }
           case compression {
             if-feature "compression";
             uses compression-content;
           }
           description
             "A rule is for compression, for no-compression or for
              fragmentation.";
         }
         description
           "Set of rules compression, no compression or fragmentation
            rules identified by their rule-id.";
       }
       description
         "A SCHC set of rules is composed of a list of rules which are
          used for compression, no-compression or fragmentation.";
       reference
         "RFC 8724 SCHC: Generic Framework for Static Context Header
                   Compression and Fragmentation";
     }
   }
   <CODE ENDS>
```

Figure 6

7. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort

has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

* Openschc is implementing the conversion between the local rule representation and the representation conforming to the data model in JSON and CBOR (following -08 draft).

8. IANA Considerations

This document registers one URI and one YANG modules.

8.1. URI Registration

This document requests IANA to register the following URI in the "IETF XML Registry" [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-schc

Registrant Contact: The IESG.

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

8.2. YANG Module Name Registration

This document registers the following one YANG modules in the "YANG Module Names" registry [RFC6020].

name: ietf-schc

namespace: urn:ietf:params:xml:ns:yang:ietf-schc

prefix: schc

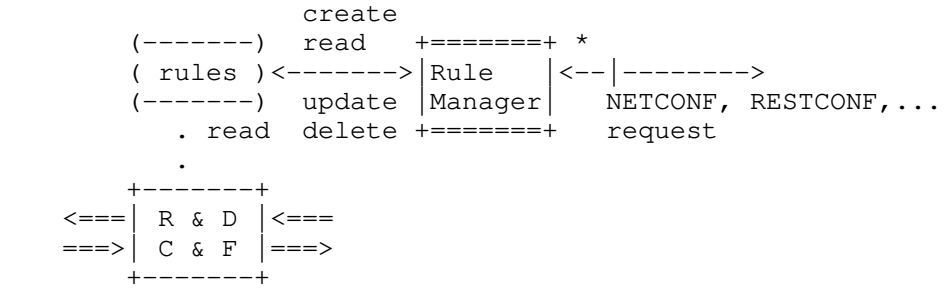
reference: RFC XXXX Data Model for Static Context Header Compression (SCHC)

9. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

This data model formalizes the rules elements described in [RFC8724] for compression, and fragmentation. As explained in the architecture document [I-D.ietf-lpwan-architecture], a rule can be read, created, updated or deleted in response to a management request. These actions can be done between two instances of SCHC or between a SCHC instance and a rule repository.



The rule contains sensitive information such as the application IPv6 address where the device's data will be sent after decompression. A device may try to modify other devices' rules by changing the application address and may block communication or allows traffic eavesdropping. Therefore, a device must be allowed to modify only its own rules on the remote SCHC instance. The identity of the requester must be validated. This can be done through certificates or access lists. By reading a module, an attacker may know the traffic a device can generate and learn about application addresses or REST API.

The full tree is sensitive, since it represents all the elements that can be managed. This module aims to be encapsulated into a YANG module including access controls and identities.

10. Annex A : Example

The informal rules given Figure 7 will represented in XML as shown in Figure 8.

```
/-------------------------\
|Rule 6/3            110  |
|--------------+---+--+--+----------------+-------+---------------\
|IPV6.VER      |  4| 1|BI|               6|EQUAL  |NOT-SENT       |
|IPV6.TC       |  8| 1|BI|               0|EQUAL  |NOT-SENT       |
|IPV6.FL       | 20| 1|BI|               0|IGNORE |NOT-SENT       |
|IPV6.LEN      | 16| 1|BI|                |IGNORE |COMPUTE-LENGTH |
|IPV6.NXT      |  8| 1|BI|              58|EQUAL  |NOT-SENT       |
|IPV6.HOP_LMT  |  8| 1|BI|             255|IGNORE |NOT-SENT       |
|IPV6.DEV_PREFIX| 64| 1|BI|200104701f2101d2|EQUAL  |NOT-SENT       |
|IPV6.DEV_IID  | 64| 1|BI|0000000000000003|EQUAL  |NOT-SENT       |
|IPV6.APP_PREFIX| 64| 1|BI|                |IGNORE |VALUE-SENT     |
|IPV6.APP_IID  | 64| 1|BI|                |IGNORE |VALUE-SENT     |
\--------------+---+--+--+----------------+-------+---------------/
/-------------------------\
|Rule 12/11     00001100  |
!=========================+=========================================\
!^ Fragmentation mode : NoAck   header dtag 2 Window  0 FCN  3  UP ^!
!^ No Tile size specified                                          ^!
!^ RCS Algorithm: RCS_CRC32                                        ^!
\====================================================================/
/-------------------------\
|Rule 100/8     01100100  |
| NO COMPRESSION RULE     |
\-------------------------/
```

Figure 7: Rules example

```
<?xml version='1.0' encoding='UTF-8'?>
  <schc xmlns="urn:ietf:params:xml:ns:yang:ietf-schc">
  <rule>
    <rule-id-value>6</rule-id-value>
    <rule-id-length>3</rule-id-length>
    <rule-nature>nature-compression</rule-nature>
    <entry>
      <field-id>fid-ipv6-version</field-id>
      <field-length>4</field-length>
      <field-position>1</field-position>
      <direction-indicator>di-bidirectional</direction-indicator>
      <matching-operator>mo-equal</matching-operator>
      <comp-decomp-action>cda-not-sent</comp-decomp-action>
      <target-value>
        <index>0</index>
```

```
          <value>AAY=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-trafficclass</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AA==</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-flowlabel</field-id>
        <field-length>20</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AA==</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-payload-length</field-id>
        <field-length>16</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-compute</comp-decomp-action>
      </entry>
      <entry>
        <field-id>fid-ipv6-nextheader</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>ADo=</value>
        </target-value>
      </entry>
      <entry>
```

```
        <field-id>fid-ipv6-hoplimit</field-id>
        <field-length>8</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AP8=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-devprefix</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>IAEEcB8hAdI=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-deviid</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-equal</matching-operator>
        <comp-decomp-action>cda-not-sent</comp-decomp-action>
        <target-value>
          <index>0</index>
          <value>AAAAAAAAAAM=</value>
        </target-value>
      </entry>
      <entry>
        <field-id>fid-ipv6-appprefix</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-value-sent</comp-decomp-action>
      </entry>
      <entry>
        <field-id>fid-ipv6-appiid</field-id>
        <field-length>64</field-length>
        <field-position>1</field-position>
        <direction-indicator>di-bidirectional</direction-indicator>
```

```
        <matching-operator>mo-ignore</matching-operator>
        <comp-decomp-action>cda-value-sent</comp-decomp-action>
      </entry>
    </rule>
    <rule>
      <rule-id-value>12</rule-id-value>
      <rule-id-length>11</rule-id-length>
      <rule-nature>nature-fragmentation</rule-nature>
      <direction>di-up</direction>
      <rcs-algorithm>rcs-crc32</rcs-algorithm>
      <dtag-size>2</dtag-size>
      <fcn-size>3</fcn-size>
      <fragmentation-mode>fragmentation-mode-no-ack</fragmentation-mode>
    </rule>
    <rule>
      <rule-id-value>100</rule-id-value>
      <rule-id-length>8</rule-id-length>
      <rule-nature>nature-no-compression</rule-nature>
    </rule>
   </schc>
```

Figure 8: XML representation of the rules.

11. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Ivan Martinez, Alexander Pelov for their careful reading and valuable inputs. A special thanks for Joe Clarke, Carl Moberg, Tom Petch, Martin Thomson, and Eric Vyncke for their explanations and wise advices when building the model.

12. References

12.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

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

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

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

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

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7136] Carpenter, B. and S. Jiang, "Significance of IPv6 Interface Identifiers", RFC 7136, DOI 10.17487/RFC7136, February 2014, <https://www.rfc-editor.org/info/rfc7136>.

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

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

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

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

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

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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

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

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

12.2. Informative References

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

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

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

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

Authors' Addresses

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

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