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 will expire on 29 December 2023.

Copyright Notice

Table of Contents

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Identifier          |        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Data ...
+-+-+-+-+-
```

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:

```
        Device       NGW     core SCHC C/D              Internet Host

                 SCHC packet
       ---|~~~~~~~~~~~~~~~~~~~~~~~>|    Echo Request, Code=0    |
    l t |  |           |            |<---------------------------|
    i i |  |           |            |                            |
    f m |  |           |            |--------------------------->|
    e e X  |           |            |    Echo Reply,   Code=0    |
      r    |           |            |                            |
           |           |            |                            |
           |           |            |    Echo Request, Code=0    |
           |           |            |O---------------------------|
           |           |            |                            |
           |           |            |                            |
```

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.

```
     Device       NGW     core SCHC C/D                Internet Host

        |           |             |    Destination Port=XXX    |
        |           |             |<---------------------------|
        |           |             |                            |
        |           |             |--------------------------->|
        |           |             | ICMPv6 Port Unreachable    |
        |           |             |                            |
        |           |             |                            |
```

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.

```
       Device       NGW     core SCHC C/D                Internet Server

         |           |            |                            |
         | SCHC compressed IPv6   |                            |
         |~~~~~~~~~~~|----------->|----------------------X     |
         |           |            |<---------------------      |
         |<~~~~~~~~~~|------------| ICMPv6 Host unreachable    |
         |SCHC compressed ICMPv6  | payload: IPv6 packet       |
         |payload: compressed IPv6|                            |
         |           |            |                            |
```

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Value/Unused                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+                as possible without the ICMPv6 packet          +
|                exceeding the minimum IPv6 MTU                 |
```

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

Network Working Group C. Gomez
Internet-Draft UPC
Intended status: Informational J. Crowcroft
Expires: September 11, 2019 University of Cambridge
March 10, 2019

RTO considerations in LPWAN
draft-gomez-rto-considerations-lpwan-00

Abstract

Low-Power Wide Area Network (LPWAN) technologies are characterized by very low physical layer bit and message transmission rates. Moreover, a response to a message sent by an LPWAN device may often only be received after a significant delay. As a result, Round-Trip Time (RTT) values in LPWAN are often (sometimes, significantly) greater than typical default values of Retransmission TimeOut (RTO) algorithms. Furthermore, buffering at network elements such as radio gateways may interact negatively with LPWAN technology transmission mechanisms, potentially exacerbating RTTs by up to several orders of magnitude. This document provides guidance for RTO settings in LPWAN, and describes an experimental dual RTO algorithm for LPWAN.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Low-Power Wide Area Network (LPWAN) technologies offer appealing features, such as multikilometer wireless link range, while allowing low energy consumption for Internet of Things (IoT) devices. However, these advantages come at the expense of reduced physical layer (PHY) bit and message rates, which in some regions are further affected by spectrum access regulatory constraints. In some LPWAN scenarios, with flagship LPWAN technologies such as LoRaWAN or Sigfox, PHY bit rates are lower than 1 kbit/s, and uplink message rates are lower than 1 message/minute [RFC8376].

Due to the aforementioned communication constraints, LPWAN technologies often exhibit high or very high Round Trip Times (RTTs). Even with negligible processing delays and in absence of communication errors, RTTs can be in the order of a few seconds or a few tens of seconds. Depending on the approach used to comply with spectrum access regulations, RTTs can grow to several minutes. Finally, when downlink responses are buffered in the radio gateway, RTTs will be in the order of the time between uplink messages (e.g. hours, if that is the time between two consecutive uplink messages).

The described RTTs, as well as their potential variability, are significantly greater than typical ones on the Internet. In TCP, the default RTO used to be 3 seconds and was reduced to 1 second [RFC7414]. In a similar order, the Constrained Application Protocol (CoAP), which is the preferred application-layer protocol for IPv6-based LPWAN, has a default RTO randomly chosen between 2 and 3 seconds [RFC7252]. At the adaptation layer between IPv6 and the LPWAN technology, some of the Static Context Header Compression (SCHC) fragmentation modes also use RTOs, which need to be defined suitably for each LPWAN technology [I-D.ietf-lpwan-ipv6-static-context-hc].

This document provides guidance for suitable RTO configuration and/or usage in LPWAN. First, the document characterizes the RTT for LoRaWAN and Sigfox in absence of communication errors, buffering delays or processing delays. Second, higher order RTTs are described, capturing the impact of message rate limitations due to regulatory constraints and radio gateway buffering delays. Finally, the document discusses suitable RTO settings in LPWAN, and describes an experimental LPWAN-specific dual RTO algorithm.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL","SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Ideal scenario RTT

This section provides an analysis of the RTT for relevant LPWAN technologies, such as LoRaWAN and Sigfox, assuming ideal conditions (i.e. no losses, as well as negligible buffering and processing delay). For detailed descriptions of LoRaWAN and Sigfox, the reader may refer to the literature [RFC8376][LoRaWAN][Sigfox].

In the analysis, the RTT comprises the time since the start of the transmission of an uplink message by an IoT device until a response is completely received by the IoT device. A 4-byte SCHC-compressed IPv6/UDP/CoAP packet is assumed for the downlink response. Of course, larger sized packets will lead to greater RTTs.

3.1. LoRaWAN

Figure 1 shows the minimum and maximum theoretical RTT values for LoRaWAN in the EU band in ideal conditions. For the minimum ones, we assume a 4-byte uplink frame payload, and a downlink response sent in the first receive window. For the maximum ones, we assume the maximum allowed uplink payload size for each Data Rate (DR), and a downlink response sent in the second receive window. Note that there is a 1- or 2-second delay between the uplink transmission and the first or second receive window, respectively.

	Maximum				
DR	Ulpld	TtxUL	TtxDL	RTTmin	RTTmax
0	51	2.79	0.99	4.52	5.81
1	51	1.56	0.58	2.99	4.15
2	51	0.70	0.29	1.92	3.00
3	115	0.68	0.14	1.73	2.82
4	242	0.70	0.07	1.66	2.78
5	242	0.40	0.04	1.37	2.44
6	242	0.20	0.02	1.19	2.22
7	242	0.04	0.003	1.00	2.05

ULpld: uplink frame payload, in bytes
TtxUL: uplink frame transmission time, in seconds
TtxDL: downlink frame transmission time, in seconds
RTTmin: minimum RTT, in seconds
RTTmax: maximum RTT, in seconds

Figure 1: Minimum and maximum RTT values for LoRaWAN in the EU, without losses, and in absence of buffering delay and processing delay.

As shown in Figure 1, and under the conditions assumed, the minimum RTT value for DR0 will always (for DR1, will almost always) exceed the default CoAP RTO. The maximum RTT will always exceed the default CoAP RTO for DR0-DR2, and will often exceed the default CoAP RTO for DR3-DR7. Note that since DR6 and DR7 are optional, they are not necessarily supported in real deployments.

3.2. Sigfox

Figure 2 shows the minimum and maximum theoretical RTT values for Sigfox in ideal conditions. For the minimum ones, we assume a 4-byte uplink frame payload, and a downlink response sent right at the beginning of the downlink receive window. For the maximum ones, we assume the maximum allowed uplink payload size, and a downlink response sent at the end of the receive window. Note that there is a 20-second delay between the frame uplink transmission and the start of the downlink receive window.

	Maximum				
UL BR	Ulpld	TtxUL	TtxDL	RTTmin	RTTmax
100	12	2.08	0.39	21.8	47.1
600	12	0.35	0.39	20.6	45.4

UL BR: uplink bit rate, in bit/s
ULpld: uplink frame payload, in bytes
TtxUL: uplink frame transmission time, in seconds
TtxDL: downlink frame transmission time, in seconds
RTTmin: minimum RTT, in seconds
RTTmax: maximum RTT, in seconds

Figure 2: Minimum and maximum RTT values for Sigfox, without losses, and in absence of buffering delay and processing delay.

As shown in Figure 2, and under the conditions assumed, the RTT in Sigfox will be one order of magnitude greater than the default CoAP RTO for all uplink bit rates and uplink frame payload sizes.

4. Higher order RTT

The high RTTs found in ideal conditions can be further exacerbated by two further behaviours of LPWAN networks: i) policies for compliance with duty cycle constraints, and ii) radio gateway buffering delays.

EU spectrum access regulations for some ISM bands used by LPWAN technologies state that, unless listen-before-talk is used, the duty cycle needs to be lower than some limit (e.g. 1% in some frequency bands). Both LoRaWAN and Sigfox need to comply with such regulations. There may be different applicable policies intended to ensure compliance with the regulations. In one of them, in order to comply with the 1% duty cycle limitation, after sending an uplink frame, an IoT device keeps an idle period equal to 99 times the transmission time of the uplink frame. Such a policy may increase the RTT by up to two orders of magnitude. For example, in LoRaWAN, this policy leads to RTTs that will always exceed the default CoAP RTO, leading to an RTT of up to 282 seconds in the worst case.

Another phenomenon that may happen in LPWAN relates with the fact that in some technologies and scenarios (e.g. the most typical LoRaWAN class, called class A, and in Sigfox), a downlink frame can only be sent during a given time interval (called receive window) after the uplink frame transmission. If a radio gateway misses the opportunity to send a downlink response to an uplink frame (e.g. because the radio gateway is busy sending other downlink messages or because it needs to refrain from transmitting immediately in order to comply with duty cycle regulations), the response to an uplink frame may be queued by the radio gateway until the next opportunity for sending a downlink frame. This problem has already been described in [I-D.toutain-core-time-scale]. If the problem occurs, the RTT will be tied to the time between two uplink consecutive frames. Depending on the application and its traffic pattern, such time may take values in the order of seconds, minutes, hours or even days.

5. Discussion and proposed dual RTO algorithm

The RTO needs to be greater than the RTT in order to avoid spurious timeouts. The latter are particularly expensive in LPWAN due to the message rate constraints in these networks, and also since they consume energy unnecessarily. However, as stated in [I-D.ietf-tcpm-rto-consider], "each implementation of a retransmission timeout mechanism represents a balance between correctness and timeliness and therefore no implementation suits all situations".

If delay is not relevant for an application, setting the default RTO to at least the highest frequently expected RTT, denoted HIGH_RTT, may be a suitable approach.

The problem arises when delay, even if at LPWAN scales, matters, and higher order RTTs (Section 3) are expected in addition to the ideal scenario ones (Section 2). At the very least, the default RTO needs to be greater than the corresponding ideal scenario RTT value shown in Section 2. If higher order RTTs are expected, one option is using a simple dual RTO approach as follows.

The LPWAN device keeps two RTO instances. One instance (called Low RTO) is initialized to a suitable ideal scenario RTT, denoted LOW_RTT. The other instance (called High RTO) is initialized to a value of at least HIGH_RTT. The dual RTO operates as follows (see Figure 3):

o Initially, the LPWAN device uses the High RTO.

o When the device uses the High RTO, after N_THRESH_LOW consecutive RTT samples lower than THRESH_LOW_RTT, the device switches to using the Low RTO.

o When the device uses the Low RTO, after N_THRESH_HIGH consecutive RTT samples greater than THRESH_HIGH_RTT, the device switches back to using the High RTO.

```
                            +-----------+
                       +--->| High RTO  |----+
                       |    +-----------+    |
if N_THRESH_HIGH       |                     | if N_THRESH_LOW
consecutive RTT samples|                     | consecutive RTT samples
 > THRESH_HIGH_RTT     |                     |  < THRESH_LOW_RTT
                       |    +-----------+    |
                       +----|  Low RTO  |<---+
                            +-----------+
```

Figure 3: State machine of the dual RTO algorithm.

The above described dual RTO algorithm may be applied to different RTO approaches, such as a constant RTO, a constant but dithered RTO (e.g. as in default CoAP), an adaptive RTO algorithm (e.g. as in TCP or CoCoA [I-D.ietf-core-cocoa]), etc. If an adaptive RTO is used, performance will benefit from separating lower RTT and higher RTT regimes, avoiding inaccuracy due to a too high RTT variance. Note that the phenomena described in Section 3 are expected to yield systematically large step function RTT distributions. These deviate significantly from the roughly normal/gaussian RTT statistics assumed by the TCP RTO algorithm.

Further refinement of the mechanism, to be discussed.

6. Security Considerations

TBD

7. Acknowledgments

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Ciencia, Innovacion y Universidades) through the Jose Castillejo grant CAS18/00170 and by European Regional Development Fund (ERDF) and the Spanish Government through project TEC2016-79988-P, AEI/FEDER, UE. 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.

8. References

8.1. 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>.

8.2. Informative References

[I-D.ietf-core-cocoa]
Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-03 (work in progress), February 2018.

[I-D.ietf-lpwan-ipv6-static-context-hc]
Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and J. Zuniga, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-18 (work in progress), December 2018.

[I-D.ietf-tcpm-rto-consider]
Allman, M., "Retransmission Timeout Requirements", draft-ietf-tcpm-rto-consider-08 (work in progress), February 2019.

[I-D.toutain-core-time-scale]
Minaburo, A. and L. Toutain, "CoAP Time Scale Option", draft-toutain-core-time-scale-00 (work in progress), October 2017.

[LoRaWAN] L. Casals, B. Mir, R. Vidal, C. Gomez, "Modeling the Energy Performance of LoRaWAN", Sensors, October 2017.

[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>.

[RFC7414] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <https://www.rfc-editor.org/info/rfc7414>.

[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>.

[Sigfox] C. Gomez, J.C. Veras, R. Vidal, L. Casals, J. Paradells, "A Sigfox Energy Consumption Model", Sensors, February 2019.

Authors' Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

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

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

March 08, 2021

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

Abstract

This draft defines how to compress the Constrained Application Protocol (CoAP) using the Static Context Header Compression (SCHC). SCHC is a header compression mechanism adapted for Constrained Devices. SCHC uses a static description of the header to reduce the header's redundancy and size. While RFC 8724 describes the SCHC compression and fragmentation framework, and its application for IPv6/UDP headers, this document applies SCHC for CoAP headers. The CoAP header structure differs from IPv6 and UDP since CoAP uses a flexible header with a variable number of options, themselves of variable length. The CoAP protocol messages format is asymmetric: the request messages have a header format different from the one in the response messages. This specification gives guidance on applying SCHC to flexible headers and how to leverage the asymmetry for more efficient compression Rules.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

A Rule describes the packets' entire header with an ordered list of fields descriptions; see section 7 of [RFC8724]. Thereby each description contains the field ID (FID), its length (FL), and its position (FP), a direction indicator (DI) (upstream, downstream, and bidirectional), and some associated Target Values (TV). The direction indicator is used for compression to give the best TV to the FID when these values differ in the transmission direction. So a field may be described several times.

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

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

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

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

SCHC is a general mechanism applied to different protocols, the exact Rules to be used depending on the protocol and the Application. Section 10 of the [RFC8724] describes the compression scheme for IPv6 and UDP headers. This document targets the CoAP header compression using SCHC.

1.1. Terminology

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

2. SCHC Applicability to CoAP

SCHC Compression for CoAP header MAY be done in conjunction with the lower layers (IPv6/UDP) or independently. The SCHC adaptation layers, described in Section 5 of [RFC8724], may be used as shown in Figure 1, Figure 2, and Figure 3.

In the first example, Figure 1, a Rule compresses the complete header stack from IPv6 to CoAP. In this case, the Device and the NGW perform SCHC C/D (Static Context Header Compression Compressor/

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

```
      (Device)            (NGW)                              (App)

      +--------+                                           +--------+
      |  CoAP  |                                           |  CoAP  |
      +--------+                                           +--------+
      |  UDP   |                                           |  UDP   |
      +--------+     +----------------+                    +--------+
      |  IPv6  |     |      IPv6      |                    |  IPv6  |
      +--------+     +--------+-------+                    +--------+
      |  SCHC  |     |  SCHC  |       |                    |        |
      +--------+     +--------+       +                    +        +
      |  LPWAN |     | LPWAN  |       |                    |        |
      +--------+     +--------+-------+                    +--------+
          ((((LPWAN))))             ------   Internet  ------
```

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.

```
         (Device)            (NGW)                              (App)

         +--------+                                           +--------+
         |  CoAP  |                                           |  CoAP  |
         +--------+                                           +--------+
         |  SCHC  |                                           |  SCHC  |
         +--------+                                           +--------+
         |  DTLS  |                                           |  DTLS  |
         +--------+                                           +--------+
         .  udp   .                                           .  udp   .
         ..........     ..................                    ..........
         .  ipv6  .     .      ipv6      .                    .  ipv6  .
         ..........     ..................                    ..........
         .  schc  .     .  schc  .       .                    .        .
         ..........     ..........       .                    .        .
         .  lpwan .     . lpwan  .       .                    .        .
         ..........     ..................                    ..........
             ((((LPWAN))))             ------   Internet  ------
```

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.

```
         (Device)            (NGW)                               (App)

         +--------+                                            +--------+
         |  CoAP  |                                            |  CoAP  |
         |  inner |                                            |  inner |
         +--------+                                            +--------+
         |  SCHC  |                                            |  SCHC  |
         |  inner |                                            |  inner |
         +--------+                                            +--------+
         |  CoAP  |                                            |  CoAP  |
         |  outer |                                            |  outer |
         +--------+                                            +--------+
         |  SCHC  |                                            |  SCHC  |
         |  outer |                                            |  outer |
         +--------+                                            +--------+
         .  udp   .                                            .  udp   .
         ..........     ..................                     ..........
         .  ipv6  .     .      ipv6      .                     .  ipv6  .
         ..........     ..................                     ..........
         .  schc  .     .  schc  .       .                     .        .
         ..........     ..........       .                     .        .
         .  lpwan .     . lpwan  .       .                     .        .
         ..........     ..................                     ..........
             ((((LPWAN))))             ------   Internet  ------
```

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.

- 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.

- 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.

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

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

Figure 6: OSCORE Option

The Figure 6 shows the OSCORE Option Value encoding defined in Section 6.1 of [RFC8613], where the first byte specifies the Content of the OSCORE options using flags. The three most significant bits of this byte are reserved and always set to 0. Bit h, when set, indicates the presence of the kid context field in the option. Bit k, when set, indicates the presence of a kid field. The three least significant bits n indicate the length of the piv (Partial Initialization Vector) field in bytes. When n = 0, no piv is present.

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

To better perform OSCORE SCHC compression, the Rule description needs to identify the OSCORE Option and the fields it contains. Conceptually, it discerns up to 4 distinct pieces of information within the OSCORE option: the flag bits, the piv, the kid context, and the kid. The SCHC Rule splits into four field descriptions the OSCORE option to compress them:

o CoAP OSCORE_flags,

o CoAP OSCORE_piv,

o CoAP OSCORE_kidctx,

o CoAP OSCORE_kid.

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

7. Examples of CoAP header compression

7.1. Mandatory header with CON message

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

RuleID 1

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

Figure 7: CoAP Context to compress header without Token

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

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

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

7.2. OSCORE Compression

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

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

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

- o Class E: Encrypted options moved to the Inner Plaintext,

- o Class I: Integrity-protected options included in the AAD for the encryption of the Plaintext but otherwise left untouched in the Outer Message,

- o Class U: Unprotected options left untouched in the Outer Message.

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

```
                         Original CoAP Packet
                      +-+-+---+-------+---------------+
                      |v|t|TKL| code  |  Msg Id.      |
                      +-+-+---+-------+---------------+....+
                      | Token                              |
                      +-------------------------------.....+
                      | Options (IEU)              |
                      .                            .
                      .                            .
                      +------+---------------------+
                      | 0xFF |
                      +------+------------------------+
                      |                               |
                      |     Payload                   |
                      |                               |
                      +-------------------------------+
                             /                \
                            /                  \
                           /                    \
                          /                      \
        Outer Header     v                        v  Plaintext
     +-+-+---+--------+---------------+          +-------+
     |v|t|TKL|new code|  Msg Id.      |          | code  |
     +-+-+---+--------+---------------+....+     +-------+-----......+
     | Token                               |     | Options (E)       |
     +--------------------------------.....+     +-------+------.....+
     | Options (IU)           |                  | 0xFF  |
     .                        .                  +-------+-----------+
     . OSCORE Option          .                  |                   |
     +------+-----------------+                  | Payload           |
     | 0xFF |                                    |                   |
     +------+                                    +-------------------+
```

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.

```
   Outer Header
+-+-+---+--------+---------------+
|v|t|TKL|new code|  Msg Id.      |
+-+-+---+--------+---------------+....+
| Token                                |
+--------------------------------.....+
| Options (IU)              |
.                           .
. OSCORE Option             .
+------+-------------------+
| 0xFF |
+------+---------------------------+
|                                  |
| Ciphertext: Encrypted Inner      |
|             Header and Payload   |
|             + Authentication Tag |
|                                  |
+----------------------------------+
```

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.

```
     Outer Message                             OSCORE Plaintext
  +-+-+---+--------+---------------+          +-------+
  |v|t|TKL|new code|  Msg Id.      |          | code  |
  +-+-+---+--------+---------------+....+     +-------+-----......+
  | Token                               |     | Options (E)       |
  +--------------------------------.....+     +-------+------.....+
  | Options (IU)              |               | OxFF  |
  .                           .               +-------+-----------+
  . OSCORE Option             .               |                   |
  +------+-------------------+                | Payload           |
  | 0xFF |                                    |                   |
  +------+------------+                       +-------------------+
  |  Ciphertext       |<---------\                     |
  |                   |          |                     v
  +-------------------+          |             +-----------------+
          |                      |             |   Inner SCHC    |
          v                      |             |   Compression   |
    +-----------------+          |             +-----------------+
    |   Outer SCHC    |          |                    |
    |   Compression   |          |                    v
    +-----------------+          |             +-------+
          |                      |             |RuleID |
          v                      |             +-------+-----------+
    +---------+            +-------------+     |Compression Residue|
    |RuleID'  |            | Encryption  | <-- +----------+--------+
    +---------+----------+ +-------------+     |                   |
    |Compression Residue'|                     |  Payload          |
    +-----------+--------+                     |                   |
    |  Ciphertext        |                     +-------------------+
    |                    |
    +--------------------+
```

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.

```
   ________________________________________________________
  |                                                        |
  | OSCORE Plaintext                                       |
  |                                                        |
  | 0x01bb74656d7065726174757265  (13 bytes)               |
  |                                                        |
  | 0x01 Request Code GET                                  |
  |                                                        |
  |      bb74656d7065726174757265 Option 11: URI_PATH      |
  |                               Value = temperature      |
  |________________________________________________________|

                             |
                             |
                             | Inner SCHC Compression
                             |
                             v
                _________________________________
               |                                 |
               | Compressed Plaintext            |
               |                                 |
               | 0x00                            |
               |                                 |
               | RuleID = 0x00 (1 byte)          |
               | (No Compression Residue)        |
               |_________________________________|

                             |
                             | AEAD Encryption
                             |  (piv = 0x04)
                             v
         _________________________________________________
        |                                                 |
        |  encrypted_plaintext = 0xa2 (1 byte)            |
        |  tag = 0xc54fe1b434297b62 (8 bytes)             |
        |                                                 |
        |  ciphertext = 0xa2c54fe1b434297b62 (9 bytes)    |
        |_________________________________________________|
```

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.

```
     +-------------------------------------------------------+
     |                                                       |
     | OSCORE Plaintext                                      |
     |                                                       |
     | 0x45ff32332043  (6 bytes)                             |
     |                                                       |
     | 0x45 Successful Response Code 69 "2.05 Content"       |
     |                                                       |
     |     ff Payload marker                                 |
     |                                                       |
     |       32332043 Payload                                |
     |                                                       |
     +-------------------------------------------------------+
                                |
                                | Inner SCHC Compression
                                |
                                v
           +---------------------------------------------+
           |                                             |
           | Compressed Plaintext                        |
           |                                             |
           | 0x001919902180 (6 bytes)                    |
           |                                             |
           |   00 RuleID                                 |
           |                                             |
           |  0b0 (1 bit match-map Compression Residue)  |
           |       0x32332043 >> 1 (shifted payload)     |
           |                        0b0000000 Padding    |
           |                                             |
           +---------------------------------------------+
                                |
                                | AEAD Encryption
                                |  (piv = 0x04)
                                v
       +--------------------------------------------------------+
       |                                                        |
       |  encrypted_plaintext = 0x10c6d7c26cc1 (6 bytes)        |
       |  tag = 0xe9aef3f2461e0c29 (8 bytes)                    |
       |                                                        |
       |  ciphertext = 0x10c6d7c26cc1e9aef3f2461e0c29 (14 bytes)|
       |                                                        |
       +--------------------------------------------------------+
```

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    |   MO  |   CDA  || Sent |
|                   |  |  |  |    Value     |       |        ||[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: June 7, 2020

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

December 05, 2019

Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6

draft-ietf-lpwan-ipv6-static-context-hc-24

Abstract

This document defines the Static Context Header Compression (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. 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 generic header compression mechanism and its application to compress IPv6/UDP headers.

This document also specifies an optional fragmentation and reassembly mechanism. It can be 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

Copyright Notice

Table of Contents

1. Introduction

This document defines the Static Context Header Compression (SCHC) framework, which provides both a header compression mechanism and an optional fragmentation mechanism. 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.

```
  ()    ()    ()        |
   ()   ()  ()  ()      / \       +---------+
 ()  ()  ()  ()  ()  ()  /   \======|    ^    |            +-----------+
  ()   ()    ()       |           | <--|--> |            |Application|
 ()   ()   ()   ()   / \==========|    v    |=============|   (App)   |
   ()   ()   ()    /   \          +---------+            +-----------+
  Dev            RGWs              NGW
```

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".

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

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

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

o 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.

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

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

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

o 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 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 Section 7.3.

- 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.

- o 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.

- o Rule: A set of Field Descriptions.

- o 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.

- o 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.

- o 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.

- o 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.

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

- o 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.

```
                +----------------+
                |      IPv6      |
             +- +----------------+
             |  |   Compression  |
      SCHC <    +----------------+
             |  |  Fragmentation |
             +- +----------------+
                |LPWAN technology|
                +----------------+
```

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.

```
  A packet (e.g., an IPv6 packet)
           |                                           ^
           v                                           |
  +------------------+                     +--------------------+
  | SCHC Compression |                     | SCHC Decompression |
  +------------------+                     +--------------------+
           |                                           ^
           |   If no fragmentation (*)                 |
           +-------------- SCHC Packet  -------------->|
           |                                           |
           v                                           |
  +--------------------+                      +-----------------+
  | SCHC Fragmentation |                      | SCHC Reassembly |
  +--------------------+                      +-----------------+
        |     ^                                     |     ^
        |     |                                     |     |
        |     +---------- SCHC ACK (+) -------------+     |
        |                                                 |
        +-------------- SCHC Fragments -------------------+

          Sender                                    Receiver


  *: 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.

```
   |------- Compressed Header -------|
   +---------------------------------+--------------------+
   |  Rule ID |  Compression Residue |      Payload       |
   +---------------------------------+--------------------+
```

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.

```
       Dev                                               App
  +----------------+                                +----+ +----+ +----+
  | App1 App2 App3 |                                |App1| |App2| |App3|
  |                |                                |    | |    | |    |
  |       UDP      |                                |UDP | |UDP | |UDP |
  |      IPv6      |                                |IPv6| |IPv6| |IPv6|
  |                |                                |    | |    | |    |
  |SCHC C/D and F/R|                                |    | |    | |    |
  +--------+-------+                                +----+ +----+ +----+
           |  +---+     +---+    +----+    +----+     .      .      .
           +~ |RGW| === |NGW| == |SCHC| == |SCHC|...... Internet ....
              +---+     +---+    |F/R |    |C/D |
                                 +----+    +----+
```

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 infrastructure 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 infrastructure 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:

- o 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 (i.e., no matching compression Rule was found).

- o 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 can 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).

```
  /-----------------------------------------------------------------\
  |                          Rule N                                 |
 /-----------------------------------------------------------------\|
 |                       Rule i                                    ||
/-----------------------------------------------------------------\||
|  (FID)            Rule 1                                        |||
|+-------+--+--+--+------------+-----------------+---------------+|||
||Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act||||
|+-------+--+--+--+------------+-----------------+---------------+|||
||Field 2|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act||||
|+-------+--+--+--+------------+-----------------+---------------+|||
||...    |..|..|..|   ...      | ...             | ...           ||||
|+-------+--+--+--+------------+-----------------+---------------+||/
||Field N|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|||
|+-------+--+--+--+------------+-----------------+---------------+|/
|                                                                 |
\-----------------------------------------------------------------/
```

Figure 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:

o 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.

o Field Length (FL) represents the length of the original 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...).

o 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.

o 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.

o 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. Which Rule to use among multiple valid Rules is 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 packet MUST be sent uncompressed using the Rule ID dedicated to this purpose (see Section 6). The entire packet header is the Compression Residue (see Figure 4). 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.

```
|------------------- Compression Residue -------------------|
+-----------------+-----------------+-----+-----------------+
| field 1 residue | field 2 residue | ... | field N residue |
+-----------------+-----------------+-----+-----------------+
```

Figure 7: Compression Residue structure

- o 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.

- o 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:

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

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

o 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.

o 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.

```
|- field residue -|
+-----------------+
|      value      |
+-----------------+
```

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).

```
|--- field residue ---|
+-------+-------------+
|  size |    value    |
+-------+-------------+
```

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:

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

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

- o 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.

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

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

- o 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.

```
                                SCHC Packet
       +----+--+-----+---+----+-+---+---+-----+...-----+----+---+------+
Tiles  |    |  |     |   |    | |   |   |     |         |    |   |      |
       +----+--+-----+---+----+-+---+---+-----+...-----+----+---+------+
```

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

o all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW_SIZE.

o WINDOW_SIZE MUST be specified in a Profile.

o the windows are numbered.

o their numbers MUST increment by 1 from 0 upward, from the start of the SCHC Packet to its end.

o the last window MUST contain WINDOW_SIZE tiles or less.

o tiles are numbered within each window.

o the tile indices MUST decrement by 1 from WINDOW_SIZE - 1 downward, looking from the start of the SCHC Packet toward its end.

o 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.

```
         +---------------------------------------------...-------------+
         |                       SCHC Packet                           |
         +---------------------------------------------...-------------+

Tile #   | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 |     | 0 | 4 | 3 |
Window # |-------- 0 --------|-------- 1 --------|- 2  ... 27 -|-- 28 -|
```

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

o Bitmaps (see Section 8.2.2.3) MAY be sent back by the receiver to the sender in a SCHC ACK message.

o 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. A Profile MUST specify how integrity checking is performed.

It is RECOMMENDED that integrity checking be 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 in the Ethernet standard [ETHERNET]) is RECOMMENDED as the default algorithm for computing the RCS.

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.

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. CRC libraries are usually byte-oriented. 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.

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. The Rule identifies

  * 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 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.

o 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.

o 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.

o 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 that the very last tile of a SCHC Packet has been

transmitted. 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.

- o 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.

- o 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.

- o 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).

```
+-----------------+-----------------------+~~~~~~~~~~~~~~~~~~~~~
| Fragment Header |   Fragment Payload    | padding (as needed)
+-----------------+-----------------------+~~~~~~~~~~~~~~~~~~~~~
```

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.

```
 |--- SCHC Fragment Header ----|
          |-- T --|-M-|-- N --|
 +-- ... --+- ... -+---+- ... -+--------...-------+~~~~~~~~~~~~~~~~~~~~~
 | Rule ID | DTag  | W |  FCN  | Fragment Payload | padding (as needed)
 +-- ... --+- ... -+---+- ... -+--------...-------+~~~~~~~~~~~~~~~~~~~~~
```

Figure 13: Detailed Header Format for Regular SCHC Fragments

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

Profiles MUST ensure that a SCHC Fragment with FCN equal to 0 (called an All-0 SCHC Fragment) is distinguishable by size, even in the presence of padding, from a SCHC ACK REQ message (see Section 8.3.3) with the same Rule ID value and with the same T, M and N values. 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.

```
|-------- SCHC Fragment Header -------|
          |-- T --|-M-|-- N --|-- U --|
+-- ... --+- ... -+---+- ... -+- ... -+------...-----+~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  | W | 11..1 |  RCS  | Frag Payload | pad. (as needed)
+-- ... --+- ... -+---+- ... -+- ... -+------...-----+~~~~~~~~~~~~~~~~~~
                        (FCN)
```

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

Profiles MUST ensure that an All-1 SCHC Fragment message is distinguishable by size, even in the presence of padding, from a SCHC Sender-Abort message (see Section 8.3.4) with the same Rule ID value and with the same T, M and N values. 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.

```
|---- SCHC ACK Header ----|
          |-- T --|-M-| 1 |
+--- ... -+- ... -+---+---+~~~~~~~~~~~~~~~~~~
| Rule ID |  DTag | W |C=1| padding as needed                (success)
+--- ... -+- ... -+---+---+~~~~~~~~~~~~~~~~~~

+--- ... -+- ... -+---+---+------ ... ------+~~~~~~~~~~~~~~~
| Rule ID |  DTag | W |C=0|Compressed Bitmap| pad. as needed (failure)
+--- ... -+- ... -+---+---+------ ... ------+~~~~~~~~~~~~~~~
```

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):

- o 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).

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

- o 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, o 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.

o 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.

```
|---- SCHC ACK Header ----|--------      Bitmap     --------|
          |-- T --|-M-| 1 |
+--- ... -+- ... -+---+---+---------------------------------+
| Rule ID |  DTag | W |C=0|1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1|
+--- ... -+- ... -+---+---+---------------------------------+
        next L2 Word boundary ->|
```

Figure 16: SCHC ACK Header plus uncompressed Bitmap

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

```
|---- SCHC ACK Header ----|CpBmp|
          |-- T --|-M-| 1 |
+--- ... -+- ... -+---+---+-----+
| Rule ID |  DTag | W |C=0|1 0 1|
+--- ... -+- ... -+---+---+-----+
        next L2 Word boundary ->|
```

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.

```
|---- SCHC ACK Header ----|--- Bitmap --|
          |-- T --|-M-| 1 |6 5 4 3 2 1 0| (tile #)
+---------+-------+---+---+-------------+
| Rule ID |  DTag | W |C=0|1 0 1 0 1 1 1|     uncompressed Bitmap
+---------+-------+---+---+-------------+
    next L2 Word boundary ->|<-- L2 Word -->|

+---------+-------+---+---+-------------+~~~+
| Rule ID |  DTag | W |C=0|1 0 1 0 1 1 1|Pad|  transmitted SCHC ACK
+---------+-------+---+---+-------------+~~~+
    next L2 Word boundary ->|<-- L2 Word -->|
```

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.

```
|---- SCHC ACK Header ----|--- Bitmap --|
          |-- T --|-M-| 1 |6 5 4 3 2 1 0| (tile #)
+---------+-------+---+---+-------------+
| Rule ID |  DTag | W |C=0|1 1 1 1 1 1 1|  with uncompressed Bitmap
+---------+-------+---+---+-------------+
    next L2 Word boundary ->|

+--- ... -+- ... -+---+---+-+
| Rule ID |  DTag | W |C=0|1|                transmitted SCHC ACK
+--- ... -+- ... -+---+---+-+
    next L2 Word boundary ->|
```

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.

```
|---- SCHC ACK REQ Header ----|
          |-- T --|-M-|-- N --|
+-- ... --+- ... -+---+- ... -+~~~~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  | W |  0..0 | padding (as needed)      (no payload)
+-- ... --+- ... -+---+- ... -+~~~~~~~~~~~~~~~~~~~~~
```

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.

```
|---- Sender-Abort Header ----|
          |-- T --|-M-|-- N --|
+-- ... --+- ... -+---+- ... -+~~~~~~~~~~~~~~~~~~~~~
| Rule ID | DTag  | W | 11..1 | padding (as needed)
+-- ... --+- ... -+---+- ... -+~~~~~~~~~~~~~~~~~~~~~
```

Figure 21: SCHC Sender-Abort format

If the W field is present,

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

```
|--- Receiver-Abort Header ---|
            |--- T ---|-M-| 1 |
+--- ... ---+-- ... --+---+---+-+-+-+-+-+-+-+-+-+-+-+
|  Rule ID  |   DTag  | W |C=1| 1..1|      1..1     |
+--- ... ---+-- ... --+---+---+-+-+-+-+-+-+-+-+-+-+-+
            next L2 Word boundary ->|<-- L2 Word -->|
```

Figure 22: SCHC Receiver-Abort format

If the W field is present,

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

- o 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

- o 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

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

Such a bit pattern never occurs in a legitimate 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

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

- o 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 State Machines implementing the SCHC F/R modes described 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

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

Each Profile, for each Rule ID value, MAY define

- o a value of N different from the recommended one,
- o 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 SCHC Fragment that carries the last tile MUST be an All-1 SCHC Fragment, described 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,

- o the receiver MUST reset the Inactivity Timer,

- o the receiver assembles the payloads of the SCHC Fragments

On receiving an All-1 SCHC Fragment,

- o 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

- o the receiver MUST perform the integrity check

- o if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet

- o 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

- o 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

o one Attempts counter

o one Retransmission Timer

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

o one Inactivity Timer

o 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,

o the Fragment MUST be of the Regular type specified in Section 8.3.1.1

o the FCN field MUST contain the tile index

o 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,

o 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.

o 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,

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

- o 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

- o 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. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

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

- o 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

- o 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.

o 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 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":

o 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 REQ, 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":

o 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.

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

At any time, on sending a SCHC ACK, the receiver MUST increment the Attempts counter.

At any time, on incrementing its window counter, the receiver MUST reset the Attempts counter.

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:

- o The penultimate tile size MUST be the regular tile size
- o 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.

```
        +---------------------------------------------...-----------+
        |                       SCHC Packet                         |
        +---------------------------------------------...-----------+

Tile #  | 4 | 3 | 2 | 1 | 0 | 4 | 3 | 2 | 1 | 0 | 4 |     | 0 | 4 |3|
Window # |-------- 0 --------|-------- 1 --------|- 2  ... 27 -|- 28-|


SCHC Fragment msg    |-----------|
```

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

- o 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,
- o 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
- whether 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 o an All-1 SCHC Fragment

o 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,

o it MUST increment the Attempts counter

o it MUST reset the Retransmission Timer

On Retransmission Timer expiration

o 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,

o 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,

o 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

        o MUST send a SCHC Sender-Abort

        o MAY exit with an error condition

      - otherwise

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

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

      + otherwise,

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

        - otherwise

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

          o 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.

o 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

o 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. The initial value of the Inactivity Timer is the RECOMMENDED lifetime for the DTag value at the receiver. If the SCHC Fragment is determined to be such a remnant, the receiver MAY silently ignore it and discard it.

- o 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.

- o 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.

- o 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,

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

o 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

o a Sender-Abort has been received

o or the Inactivity Timer has expired

o or the Attempts counter has exceeded MAX_ACK_REQUESTS

o 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.

```
A packet (e.g., an IPv6 packet)
         |                                   ^ (padding bits
         v                                   |       dropped)
+------------------+                      +--------------------+
| SCHC Compression |                      | SCHC Decompression |
+------------------+                      +--------------------+
         |                                   ^
         |   If no fragmentation             |
         +---- SCHC Packet + padding as needed ----->|
         |                                   | (integrity
         v                                   |  checked)
+--------------------+                       +-----------------+
| SCHC Fragmentation |                       | SCHC Reassembly |
+--------------------+                       +-----------------+
     |       ^                                   |       ^
     |       |                                   |       |
     |       +--- SCHC ACK + padding as needed --+       |
     |                                                   |
     +------- SCHC Fragments + padding as needed---------+

        Sender                                    Receiver
```

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:

- o 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".

- o 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:

- o 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".

- o 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, implementers 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 re-construct 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 susceptible to attacks, such as BREACH and CRIME. The attack involves injecting arbitrary data into the packet and observing the resulting compressed 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

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 predictable 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, assuming the tunnel provides confidentiality, neither the DTag field nor any other SCHC-introduced field is visible over the Internet.

13. Acknowledgements

Thanks to (in alphabetical order) Sergio Aguilar Romero, David Black, Carsten Bormann, Deborah Brungard, Brian Carpenter, Philippe Clavier, Alissa Cooper, Roman Danyliw, Daniel Ducuara Beltran, Diego Dujovne, Eduardo Ingles Sanchez, Rahul Jadhav, Benjamin Kaduk, Arunprabhu Kandasamy, Suresh Krishnan, Mirja Kuehlewind, Barry Leiba, Sergio Lopez Bernal, Antoni Markovski, Alexey Melnikov, Georgios Papadopoulos, 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.

```
 Management   Data
+----------+---------+---------+
|   CoAP   |  CoAP   | legacy  |
+----||----+---||----+---||----+
.   UDP    .  UDP    |   UDP   |
................................
.   IPv6   .  IPv6   .  IPv6   .
+------------------------------+
|    SCHC Header compression   |
|      and fragmentation       |
+------------------------------+
|      LPWAN L2 technologies   |
+------------------------------+
         Dev or NGW
```

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
  Special Rule ID used to tag an uncompressed UDP/IPV6 packet.

Rule 1

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 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	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 3

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

Figure 26 describes a example of a Rule set.

In this example, 0 was chosen as the special Rule ID that tags packets that cannot be compressed with any compression Rule.

All the fields described in Rules 1-3 are present in the IPv6 and UDP headers. The DevIID-DID value is found in the L2 header.

Rules 2-3 use global addresses. The way the Dev learns the prefix is not in the scope of the document.

Rule 3 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.

```
        Sender               Receiver
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-------FCN=0-------->|
          |-----FCN=1 + RCS --->| Integrity check: success
        (End)
```

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.

```
        Sender               Receiver
          |-----W=0, FCN=6----->|
          |-----W=0, FCN=5----->|
          |-----W=0, FCN=4----->|
          |-----W=0, FCN=3----->|
          |-----W=0, FCN=2----->|
          |-----W=0, FCN=1----->|
          |-----W=0, FCN=0----->|
      (no ACK)
          |-----W=1, FCN=6----->|
          |-----W=1, FCN=5----->|
          |-----W=1, FCN=4----->|
          |--W=1, FCN=7 + RCS-->| Integrity check: success
          |<-- ACK, W=1, C=1 ---| C=1
        (End)
```

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.

```
            Sender               Receiver
              |-----W=0, FCN=6----->|
              |-----W=0, FCN=5----->|
              |-----W=0, FCN=4--X-->|
              |-----W=0, FCN=3----->|
              |-----W=0, FCN=2--X-->|
              |-----W=0, FCN=1----->|
              |-----W=0, FCN=0----->|        6543210
              |<-- ACK, W=0, C=0 ---| Bitmap:1101011
              |-----W=0, FCN=4----->|
              |-----W=0, FCN=2----->|
         (no ACK)
              |-----W=1, FCN=6----->|
              |-----W=1, FCN=5----->|
              |-----W=1, FCN=4--X-->|
              |- W=1, FCN=7 + RCS ->| Integrity check: failure
              |<-- ACK, W=1, C=0 ---| C=0, Bitmap:1100001
              |-----W=1, FCN=4----->| Integrity check: success
              |<-- ACK, W=1, C=1 ---| C=1
           (End)
```

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.

```
      Sender                Receiver
        |-----W=0, FCN=27----->| 4 tiles sent
        |-----W=0, FCN=23----->| 4 tiles sent
        |-----W=0, FCN=19----->| 4 tiles sent
        |-----W=0, FCN=15--X-->| 4 tiles sent (not received)
        |-----W=0, FCN=11----->| 4 tiles sent
        |-----W=0, FCN=7 ----->| 4 tiles sent
        |-----W=0, FCN=3 ----->| 4 tiles sent
        |-----W=1, FCN=27----->| 4 tiles sent
        |-----W=1, FCN=23----->| 4 tiles sent
        |-----W=1, FCN=19----->| 4 tiles sent
        |-----W=1, FCN=15----->| 4 tiles sent
        |-----W=1, FCN=11----->| 4 tiles sent
        |-----W=1, FCN=7 ----->| 4 tiles sent
        |-----W=1, FCN=3 --X-->| 4 tiles sent (not received)
        |-----W=2, FCN=27----->| 4 tiles sent
        |-----W=2, FCN=23----->| 4 tiles sent
    ^   |-----W=2, FCN=19----->| 1 tile sent
    |   |-----W=2, FCN=18----->| 1 tile sent
    |   |-----W=2, FCN=17----->| 1 tile sent
        |-----W=2, FCN=16----->| 1 tile sent
    s   |-----W=2, FCN=15----->| 1 tile sent
    m   |-----W=2, FCN=14----->| 1 tile sent
    a   |-----W=2, FCN=13--X-->| 1 tile sent (not received)
    l   |-----W=2, FCN=12----->| 1 tile sent
    l   |---W=2, FCN=31 + RCS->| Integrity check: failure
    e   |<--- ACK, W=0, C=0 ---| C=0, Bitmap:1111111111110000111111111111
    r   |-----W=0, FCN=15----->| 1 tile sent
        |-----W=0, FCN=14----->| 1 tile sent
    L   |-----W=0, FCN=13----->| 1 tile sent
    2   |-----W=0, FCN=12----->| 1 tile sent
        |<--- ACK, W=1, C=0 ---| C=0, Bitmap:1111111111111111111111110000
    M   |-----W=1, FCN=3 ----->| 1 tile sent
    T   |-----W=1, FCN=2 ----->| 1 tile sent
    U   |-----W=1, FCN=1 ----->| 1 tile sent
        |-----W=1, FCN=0 ----->| 1 tile sent
    |   |<--- ACK, W=2, C=0 ---| C=0, Bitmap:1111111111111101000000000001
    |   |-----W=2, FCN=13----->| Integrity check: success
    V   |<--- ACK, W=2, C=1 ---| C=1
      (End)
```

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.

```
        Sender               Receiver
          |-----W=0, FCN=6----->|
          |-----W=0, FCN=5----->|
          |-----W=0, FCN=4----->|
          |-----W=0, FCN=3----->|
          |-----W=0, FCN=2----->|
          |-----W=0, FCN=1----->|
          |-----W=0, FCN=0----->|
          |<-- ACK, W=0, C=0 ---| Bitmap:1111111
          |-----W=1, FCN=6----->|
          |-----W=1, FCN=5----->|
          |-----W=1, FCN=4----->|
          |--W=1, FCN=7 + RCS-->| Integrity check: success
          |<-- ACK, W=1, C=1 ---| C=1
        (End)
```

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.

```
        Sender               Receiver
          |-----W=0, FCN=6----->|
          |-----W=0, FCN=5----->|
          |-----W=0, FCN=4--X-->|
          |-----W=0, FCN=3----->|
          |-----W=0, FCN=2--X-->|
          |-----W=0, FCN=1----->|
          |-----W=0, FCN=0----->|        6543210
          |<-- ACK, W=0, C=0 ---| Bitmap:1101011
          |-----W=0, FCN=4----->|
          |-----W=0, FCN=2----->|
          |<-- ACK, W=0, C=0 ---| Bitmap:1111111
          |-----W=1, FCN=6----->|
          |-----W=1, FCN=5----->|
          |-----W=1, FCN=4--X-->|
          |--W=1, FCN=7 + RCS-->| Integrity check: failure
          |<-- ACK, W=1, C=0 ---| C=0, Bitmap:11000001
          |-----W=1, FCN=4----->| Integrity check: success
          |<-- ACK, W=1, C=1 ---| C=1
        (End)
```

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.

```
          Sender               Receiver
             |-----W=0, FCN=6----->|
             |-----W=0, FCN=5----->|
             |-----W=0, FCN=4--X-->|
             |-----W=0, FCN=3--X-->|
             |-----W=0, FCN=2--X-->|
             |--W=0, FCN=7 + RCS-->| Integrity check: failure
             |<-- ACK, W=0, C=0 ---| C=0, Bitmap:1100001
             |-----W=0, FCN=4----->| Integrity check: failure
             |-----W=0, FCN=3----->| Integrity check: failure
             |-----W=0, FCN=2----->| Integrity check: success
             |<-- ACK, W=0, C=1 ---| C=1
           (End)
```

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.

```
          Sender               Receiver
             |-----W=0, FCN=6----->|
             |-----W=0, FCN=5----->|
             |-----W=0, FCN=4--X-->|
             |-----W=0, FCN=3--X-->|
             |-----W=0, FCN=2--X-->|
             |--W=0, FCN=7 + RCS-->| Integrity check: failure
             |<-- ACK, W=0, C=0 ---| C=0, Bitmap:1100001
             |-----W=0, FCN=4----->| Integrity check: failure
             |-----W=0, FCN=3----->| Integrity check: failure
             |-----W=0, FCN=2----->| Integrity check: success
             |<-X-ACK, W=0, C=1 ---| C=1
    timeout  |                     |
             |--- W=0, ACK REQ --->| ACK REQ
             |<-- ACK, W=0, C=1 ---| C=1
           (End)
```

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.

```
           Sender               Receiver
             |-----W=0, FCN=6----->|
             |-----W=0, FCN=5----->|
             |-----W=0, FCN=4--X-->|
             |-----W=0, FCN=3--X-->|
             |-----W=0, FCN=2--X-->|
             |--W=0, FCN=7 + RCS-->| Integrity check: failure
             |<-- ACK, W=0, C=0 ---| C=0, Bitmap:1100001
             |-----W=0, FCN=4----->| Integrity check: failure
             |-----W=0, FCN=3----->| Integrity check: failure
             |-----W=0, FCN=2--X-->|
      timeout|                     |
             |--- W=0, ACK REQ --->| ACK REQ
             |<-- ACK, W=0, C=0 ---| C=0, Bitmap: 1111101
             |-----W=0, FCN=2----->| Integrity check: success
             |<-- ACK, W=0, C=1 ---| C=1
           (End)
```

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.

```
      Sender               Receiver
        |-----W=0, FCN=23----->|
        |-----W=0, FCN=22----->|
        |-----W=0, FCN=21--X-->|
        |-----W=0, FCN=20----->|
        |-----W=0, FCN=19----->|
        |-----W=0, FCN=18----->|
        |-----W=0, FCN=17----->|
        |-----W=0, FCN=16----->|
        |-----W=0, FCN=15----->|
        |-----W=0, FCN=14----->|
        |-----W=0, FCN=13----->|
        |-----W=0, FCN=12----->|
        |-----W=0, FCN=11----->|
        |-----W=0, FCN=10--X-->|
        |-----W=0, FCN=9 ----->|
        |-----W=0, FCN=8 ----->|
        |-----W=0, FCN=7 ----->|
        |-----W=0, FCN=6 ----->|
        |-----W=0, FCN=5 ----->|
        |-----W=0, FCN=4 ----->|
        |-----W=0, FCN=3 ----->|
        |-----W=0, FCN=2 ----->|
        |-----W=0, FCN=1 ----->|
        |-----W=0, FCN=0 ----->|
        |                      |
        |<--- ACK, W=0, C=0 ---| Bitmap:110111111111101111111111
        |-----W=0, FCN=21----->|
        |-----W=0, FCN=10----->|
        |<--- ACK, W=0, C=0 ---| Bitmap:111111111111111111111111
        |-----W=1, FCN=23----->|
        |-----W=1, FCN=22----->|
        |-----W=1, FCN=21----->|
        |--W=1, FCN=31 + RCS-->| Integrity check: success
        |<--- ACK, W=1, C=1 ---| C=1
      (End)
```

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:

```
                +============+
   +------------+  Init      |
   |  FCN=0     +============+
   |  No Window
   |  No Bitmap
   |                   +--------+
   |          +========+==+     | More Fragments
   |          |           | <--+ ~~~~~~~~~~~~~~~~~~~~
   +--------> |    Send   |      send Fragment (FCN=0)
              +===+=======+
                  |  last fragment
                  |  ~~~~~~~~~~~~
                  |  FCN = 1
                  v  send fragment+RCS
              +============+
              |    END     |
              +============+
```

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

```
                         +------+ Not All-1
              +==========+=+    | ~~~~~~~~~~~~~~~~~~~
              |            + <--+ set Inactivity Timer
              |  RCV Frag  +-------+
              +=+===+======+       |All-1 &
     All-1 &    |   |              |RCS correct
    RCS wrong   |   |Inactivity    |
                |   |Timer Exp.    |
                v   |              |
     +==========++  |              v
     |   Error    |<-+      +========+==+
     +============+         |    END    |
                            +===========+
```

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

```
                 +=======+
                 | INIT  |       FCN!=0 & more frags
                 |       |       ~~~~~~~~~~~~~~~~~~~~~~
                 +======++  +--+ send Window + frag(FCN)
                    W=0 |   |  | FCN-
     Clear lcl_bm       |   |  v set lcl_bm
          FCN=max value |  ++==+========+
                        +> |            |
   +---------------------> |    SEND    |
   |                       +==+===+=====+
   |      FCN==0 & more frags |   | last frag
   |    ~~~~~~~~~~~~~~~~~~~~~ |   | ~~~~~~~~~~~~~~~
   |               set lcl_bm |   | set lcl_bm
   |   send wnd + frag(all-0) |   | send wnd+frag(all-1)+RCS
   |       set Retrans_Timer  |   | set Retrans_Timer
   |                          |   |
   |Recv_wnd == wnd &         |   |
   |lcl_bm==recv_bm &         |   |  +----------------------+
   |more frag                 |   |  | lcl_bm!=rcv-bm       |
   |~~~~~~~~~~~~~~~~~~~~~~    |   |  | ~~~~~~~~~            |
   |Stop Retrans_Timer        |   |  | Attempt++            v
   |clear lcl_bm              v   v  |                +=====+=+
   |window=next_window   +====+===+==+===+            |Resend |
   +---------------------+               |            |Missing|
                    +----+     Wait      |            |Frag   |
   not expected wnd |    |    Bitmap     |            +=======+
   ~~~~~~~~~~~~~~~~ +--->+               ++Retrans_Timer Exp  |
       discard frag      +==+=+===+=+==+=+| ~~~~~~~~~~~~~~~~~ |
                            | |   | ^  ^ |reSend(empty)All-*  |
                            | |   | |  | |Set Retrans_Timer   |
                            | |   | |  +--+Attempt++          |
     C_bit==1 &             | |   | +-------------------------+
   Recv_window==window &    | |   |   all missing frags sent
                no more frag| |   |   ~~~~~~~~~~~~~~~~~~~~~~
    ~~~~~~~~~~~~~~~~~~~~~~~~| |   |   Set Retrans_Timer
          Stop Retrans_Timer| |   |
    +=============+         | |   |
    |     END     +<--------+ |   |
    +=============+           |   | Attempt > MAX_ACK_REQUESTS
               All-1 Window & |   | ~~~~~~~~~~~~~~~~~~
                  C_bit ==0 & |   v Send Abort
             lcl_bm==recv_bm  | +=+===========+
                 ~~~~~~~~~~~~ +>|    ERROR    |
                   Send Abort   +=============+
```

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

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

```
                  +=======+
                  |       |
                  | INIT  |
                  |       |       FCN!=0 & more frags
                  +======++       ~~~~~~~~~~~~~~~~~~~~~~
     Frag RuleID trigger |   +--+ Send cur_W + frag(FCN);
     ~~~~~~~~~~~~~~~~~~~ |   |  | FCN--;
  cur_W=0; FCN=max_value;|   |  | set [cur_W, cur_Bmp]
    clear [cur_W, Bmp_n];|   |  v
          clear rcv_Bmp  |  ++==+==========+         **BACK_TO_SEND
                         +->+              |     cur_W==rcv_W &
      **BACK_TO_SEND        |     SEND     |     [cur_W,Bmp_n]==rcv_Bmp
+-------------------------->+              |     & more frags
|  +----------------------->+              |     ~~~~~~~~~~~~
|  |                        ++===+=========+     cur_W++;
|  |      FCN==0 & more frags|   |last frag      clear [cur_W, Bmp_n]
|  |  ~~~~~~~~~~~~~~~~~~~~~~~|   |~~~~~~~~~
|  |        set cur_Bmp;     |   |set [cur_W, Bmp_n];
|  |send cur_W + frag(All-0);|   |send cur_W + frag(All-1)+RCS;
|  |        set Retrans_Timer|   |set Retrans_Timer
|  |                         |   | +-----------------------------------+
|  |Retrans_Timer expires &  |   | |cur_W==rcv_W&[cur_W,Bmp_n]!=rcv_Bmp|
|  |more Frags               |   | |  ~~~~~~~~~~~~~~~~~~               |
|  |~~~~~~~~~~~~~~~~~~~~     |   | |  Attempts++; W=cur_W              |
|  |stop Retrans_Timer;      |   | | +--------+             rcv_W==Wn &|
|  |[cur_W,Bmp_n]==cur_Bmp;  v   v | |        v     [Wn,Bmp_n]!=rcv_Bmp|
|  |cur_W++            +=====+===+=+=+===+  +=+=========+   ~~~~~~~~~~~
|  +-------------------+                 |  | Resend    |   Attempts++;|
+----------------------+   Wait x ACK    |  | Missing   |          W=Wn|
+--------------------->+                 |  | Frags(W)  +<-------------+
|         rcv_W==Wn &+-+                 |  +======+====+
| [Wn,Bmp_n]!=rcv_Bmp| ++=+===+===+==+===+         |
|      ~~~~~~~~~~~~~~|  ^ |   |   |  ^            |
|        send (cur_W,+--+ |   |   |  +-------------+
|        ALL-0-empty)     |   |   |     all missing frag sent(W)
|                         |   |   |     ~~~~~~~~~~~~~~~~~
|  Retrans_Timer expires &|   |   |     set Retrans_Timer
|            No more Frags|   |   |
|           ~~~~~~~~~~~~~~|   |   |
|                         |   |   |
```

```
|       stop Retrans_Timer;|    |   |
|(re)send frag(All-1)+RCS |    |   |
+-------------------------+    |   |
                 cur_W==rcv_W& |   |
       [cur_W,Bmp_n]==rcv_Bmp& |   | Attempts > MAX_ACK_REQUESTS
  No more Frags & RCS flag==OK |   | ~~~~~~~~~~
                ~~~~~~~~~~~~~~~~~~ |   | send Abort
 +=========+stop Retrans_Timer|   |  +===========+
 |   END   +<-----------------+   +->+   ERROR   |
 +=========+                         +===========+
```

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.

```
                   +=======+        New frag RuleID received
                   |       |        ~~~~~~~~~~~~~
                   | INIT  +-------+cur_W=0;clear([cur_W,Bmp_n]);
                   +=======+       |sync=0
                                   |
      Not All* & rcv_W==cur_W+---+ | +---+
        ~~~~~~~~~~~~~~~~~~~~ |   | | |  (E)
        set[cur_W,Bmp_n(FCN)]|   v v v   |
                            ++===+=+=+===+=+
     +----------------------+              +--+ All-0&Full[cur_W,Bmp_n]
     |           ABORT *<---+  Rcv Window  |  | ~~~~~~~~~~
     |  +-------------------+              +<-+ cur_W++;set Inact_timer;
     |  |                +->+=+=+=+=+=+====+    clear [cur_W,Bmp_n]
     |  | All-0 empty(Wn)|    | | | ^ ^
     |  | ~~~~~~~~~~~~~~ +----+ | | | |rcv_W==cur_W & sync==0;
     |  | sendACK([Wn,Bmp_n])   | | | |& Full([cur_W,Bmp_n])
     |  |                       | | | |& All* || last_miss_frag
     |  |                       | | | |~~~~~~~~~~~~~~~~~~~~~~
     |  |    All* & rcv_W==cur_W|(C)| |sendACK([cur_W,Bmp_n]);
     |  |              & sync==0| | | |cur_W++; clear([cur_W,Bmp_n])
     |  |&no_full([cur_W,Bmp_n])| |(E)|
     |  |      ~~~~~~~~~~~~~~~~ | | | |              +========+
     |  | sendACK([cur_W,Bmp_n])| | | |              | Error/ |
     |  |                       | | | |   +----+     | Abort  |
     |  |                       v v | |   |    |     +===+====+
     |  |                   +===+=+=+=+===+=+ (D)        ^
     |  |                +--+    Wait x     |  |         |
     |  | All-0 empty(Wn)+->| Missing Frags |<-+         |
     |  | ~~~~~~~~~~~~~~    +=============+=+            |
     |  | sendACK([Wn,Bmp_n])             +--------------+
     |  |                                       *ABORT
     v  v
    (A)(B)
                                      (D) All* || last_miss_frag
      (C) All* & sync>0                   & rcv_W!=cur_W & sync>0
          ~~~~~~~~~~~~                    & Full([rcv_W,Bmp_n])
          Wn=oldest[not full(W)];         ~~~~~~~~~~~~~~~~~~~~
          sendACK([Wn,Bmp_n])             Wn=oldest[not full(W)];
                                          sendACK([Wn,Bmp_n]);sync--

                                ABORT-->* Uplink Only &
                                          Inact_Timer expires
      (E) Not All* & rcv_W!=cur_W         || Attempts > MAX_ACK_REQUESTS
          ~~~~~~~~~~~~~~~~~~~~            ~~~~~~~~~~~~~~~~~~~~~
          sync++; cur_W=rcv_W;            send Abort
          set[cur_W,Bmp_n(FCN)]
```

```
     (A)(B)
      |  |
      |  | All-1 & rcv_W==cur_W & RCS!=OK        All-0 empty(Wn)
      |  | ~~~~~~~~~~~~~~~~~~~~~~~~~~~~     +-+  ~~~~~~~~~~
      |  | sendACK([cur_W,Bmp_n],C=0)       | v  sendACK([Wn,Bmp_n])
      |  |                      +===========+=++
      |  +---------------------->+   Wait End   +-+
      |                         +=====+=+====+=+ | All-1
      |     rcv_W==cur_W & RCS==OK    | |    ^   | & rcv_W==cur_W
      |     ~~~~~~~~~~~~~~~~~~~~~~    | |    +---+ & RCS!=OK
      |  sendACK([cur_W,Bmp_n],C=1)   | |          ~~~~~~~~~~~~~~~~~~~
      |                               | | sendACK([cur_W,Bmp_n],C=0);
      |                               | |          Attempts++
      |All-1 & Full([cur_W,Bmp_n])    | |
      |& RCS==OK & sync==0            | +-->* ABORT
      |~~~~~~~~~~~~~~~~~~~            v
      |sendACK([cur_W,Bmp_n],C=1)   +=+=========+
      +---------------------------->+    END    |
                                    +===========+
```

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, the document must describe:

  * 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, lifetime of DTag at the receiver

  * 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

  * what makes an All-0 SCHC Fragment and a SCHC ACK REQ distinguishable (see Section 8.3.1.1).

  * what makes an All-1 SCHC Fragment and a SCHC Sender-Abort distinguishable (see Section 8.3.1.2).

  * 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.

- o 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. See Appendix F.

- 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: Informational
Expires: September 8, 2019

A. Minaburo
Acklio
E. Ramos
Ericsson
S. Shanmugalingam
Acklio

March 07, 2019

LPWAN Static Context Header Compression (SCHC) over NB-IoT
draft-minaburo-lpwan-nbiot-hc-02

Abstract

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Static Context Header Compression (SCHC) [I-D.ietf-lpwan-ipv6-static-context-hc] defines a header compression scheme and fragmentation functionality, both specially tailored for Low Power Wide Area Networks (LPWAN) networks defined in [RFC8376].

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

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

2. Terminology

This document will follow the terms defined in [I-D.ietf-lpwan-ipv6-static-context-hc], in [RFC8376], and the [TGPP23720].

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

- o HSS. Home Subscriber Server. It is a database that performs mobility management

- o PGW. Packet Data Node Gateway. An interface between the internal with the external network

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

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

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

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

3. Architecture

```
+--+
|UE| \              +------+     +-------+
+--+  \             | MME  |-----| HSS   |
       \          / +------+     +-------+
+--+    \+-----+ /        |
|UE| ----| eNB |-         |
+--+    /+-----+ \        |
       /          \ +-------+
      /            \|       |  +-------+   Service PDN
+--+ /              | S-GW  |--| P-GW  |-- e.g. Internet
|UE|                |       |  +-------+
+--+                +-------+
```

Figure 1: 3GPP network architecture

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

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

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

- o Necessary security procedures for efficient small data transmission.

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

- o Paging optimizations for coverage enhancements.

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

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

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

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

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

```
                                             +-------+
                                             |  HSS  |
                                             +-+-----+
                                              /
                              +---------+  __/S6a
                 +--------+   | +-----+ +_/
   +----+ C-Uu   |        +---+-+ MME | | T6i+--------+ T7 +----+
   |CIOT+--------+  eNB   |S1 | |     +-+----+IWK-SCEF+----+SCEF|
   |UE  |        |(NB-IoT)|   | +---+-+ |    +--------+    +----+
   +----+        +--------+   |     | |
                              |C-SGN| |
                              |     |S11
                   +------+   |     | |
   +--------+LTE-Uu|      |   |  +--+-+ |
   |LTE eMTC|(eMTC)|eNB   +---+--+SGW | | S8+---+    +-----------+
   |   UE   +------+(eMTC)|S1 |  |    +-+---+PGW|SGi |Application|
   +--------+      +------+   |  +----+ |   |   +----+Server (AS)|
                              +---------+   +---+    +-----------+
```

Figure 2: 3GPP optimized CIOT network architecture

4. Data Transmission

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

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

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

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

SCHC could be deployed differently depending on where the header compression and the fragmentation are applied. The SCHC functionalities could be applied to the packets about to be transmitted over the air, or to the whole end-to-end link. To accomplish the first, it is required to place SCHC compression and decompression entities in the eNB and in the UE for transmissions over the User Plane. Additionally, to handle the case of the transmissions over Control Plane or Data Over NAS, the network SCHC entity has to be placed in the C-SGN as well. For these two cases, the functions are to be standardized by 3GPP.

Another possibility is to apply SCHC functionalities to the end-to-end connection or at least up to the operator network edge. In that case, the SCHC entities would be placed in the application layer of the terminal in one end, and either in the application servers or in a broker function in the edge of the operator network in the other end. For the radio network, the packets are transmitted as non-IP traffic, which can be currently served utilizing IP tunneling or SCEF services. Since this option does not necessarily require 3GPP standardization, it is possible to also benefit legacy devices with SCHC by utilizing the non-IP transmission features of the operator network.

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

5. IP based Data Transmission

5.1. SCHC over User Plane transmissions

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

5.1.1. SCHC Entities Placing

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

In this scenario, RLC takes care of the handling of fragmentation (if transparent mode is not configured) when packets exceeds the transport block size at the time of transmission. Therefore SCHC fragmentation is not needed and should not be used to avoid additional protocol overhead. It is not common to configure RLC in Transparent Mode for IP based user plane data. But given the case in the future, SCHC fragmentation may be used. In that case, a SCHC tile would match the minimum transport block size minus the PDCP and MAC headers.

```
     +---------+                              +---------+  |
     |IP/non-IP+------------------------------+IP/non-IP+->+
     +---------+   |   +---------------+   |  +---------+  |
     | PDCP    +-------+ PDCP  | GTP|U +------+ GTP-U   |->+
     | (SCHC)  +       + (SCHC)|       +      +         |  |
     +---------+   |   +---------------+   |  +---------+  |
     | RLC     +-------+ RLC   |UDP/IP +------+ UDP/IP  +->+
     +---------+   |   +---------------+   |  +---------+  |
     | MAC     +-------+ MAC   | L2    +------+ L2      +->+
     +---------+   |   +---------------+   |  +---------+  |
     | PHY     +-------+ PHY   | PHY   +------+ PHY     +->+
     +---------+       +---------------+      +---------+  |
                  C-Uu/                    S1-U             SGi
        CIOT/     LTE+Uu     C-BS/eNB             C-SGN
       LTE eMTC
         UE
```

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

5.2. Data Over Control Plane

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

NAS has been adapted to provide support for user plane data transmissions to reduce the overhead when transmitting infrequent small quantities of data. This is known as Data over NAS (DoNAS) or Control Plane CIoT EPS optimization. In DoNAS the UE makes use of the pre-established NAS security and piggyback uplink small data into the initial NAS uplink message, and uses an additional NAS message to receive downlink small data response.

The data encryption from the network side is performed by the C-SGN in a NAS PDU. Depending on the data type signaled indication (IP or non-IP data), the network allocates an IP address or just establish a direct forwarding path. DoNAS (Data over NAS) is regulated under rate control upon previous agreement, meaning that a maximum number of bits per unit of time is agreed per device subscription beforehand and configured in the device.

The use of DoNAS is typically expected when a terminal in a power saving state requires to do a short transmission and receive an acknowledgment or short feedback from the network. Depending on the size of buffered data to transmit, the UE might be instructed to deploy the connected mode transmissions instead, limiting and controlling the DoNAS transmissions to predefined thresholds and a good resource optimization balance for the terminal and the network. The support for mobility of DoNAS is present but produces additional overhead. Additional details of DoNAS are given in the Appendix.

5.2.1. SCHC Entities Placing

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

```
+---------+                       +---------+--------+  +  +---------+
| IP/     +--+-----------------+--+  IP/    |   IP/  +------+   IP/   |
| Non-IP |  |                 |  | Non-IP | Non-IP |  |  | Non-IP |
+---------+  |                 |  +------------------+  |  +---------+
| NAS     +-----------------------+   NAS   |GTP|C/U +------+GTP|C/U |
|(SCHC)  |  |                 |  | (SCHC)  |        |  |  |         |
+---------+  |  +------------+  |  +------------------+  |  +---------+
| RRC     +------+RRC  |S1|AP+------+ S1|AP  |        |  |  |         |
+---------+  |  +------------+  |  +---------+  UDP   +------+  UDP   |
| PDCP*   +------+PDCP*|SCTP +------+ SCTP    |        |  |  |         |
+---------+  |  +------------+  |  +------------------+  |  +---------+
| RLC     +------+ RLC | IP  +------+ IP      | IP     +------+ IP      |
+---------+  |  +------------+  |  +------------------+  |  +---------+
| MAC     +------+ MAC | L2  +------+ L2      | L2     +------+ L2      |
+---------+  |  +------------+  |  +------------------+  |  +---------+
| PHY     +--+--+ PHY | PHY +--+--+ PHY     | PHY    +------+ PHY     |
+---------+     +-----+------+     +---------+--------+  |  +---------+
          C-Uu/          S1-lite                        SGi
   CIOT/  LTE-Uu   C-BS/eNB           C-SGN                  PGW
 LTE eMTC
   UE
```

*PDCP is bypassed until AS security is activated [TGPP36300].

Figure 4

5.3. Parameters for Static Context Header Compression (SCHC)

5.3.1. SCHC Context initialization

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

5.3.2. SCHC Rules

The number of rules in a context are defined by the network operator in these scenarios. For this, the operator must be aware of the type of IP traffic that the device will carry out. This means that the operator might provision sets of rules compatible with the use case of the device. For devices acting as gateways of other devices several rules that match the diversity of devices and protocols used by the devices associated to the gateway. Meanwhile than simpler devices (for example an electricity meter) may have a predetermined set of protocols and parameters fixed. Additionally, the deployment of IPV4 addresses in addition to IPV6 may force to provision separate rules to deal with each of the cases.

5.3.3. Rule ID

For these transmission scenarios in NB-IoT, a reasonable assumption of 9 bytes of radio protocol overhead can be expected. PDCP 5 bytes due to header and integrity protection, and 4 bytes of RLC and MAC. The minimum physical Transport Block (TB) that can withhold this overhead value according to 3GPP Release 15 specifications are: 88, 104, 120 and 144 bits. If it is wished to optimize the number of transmissions of a very small application packet so that in some cases can be transmitted using only one physical layer transmission, then the SCHC overhead should not exceed the available number of bits of the smallest utile physical TB available. The packets handled by 3GPP networks are byte-aligned, and therefore the minimum payload possible (including padding) is 8 bits. Therefore in order to utilize the smallest TB the maximum SCHC is 8 bits. This must include the Compression Residue in addition to the Rule ID. In the other hand, it is possible that more complex NB-IoT devices (such as a capillarity gateway) might require additional bits to handle the variety and multiple parameters the of higher layer protocols deployed. In that sense, the operator may want to have flexibility on the number and type of rules supported by each device independently, and consequently a configurable value is preferred for these scenarios. The configuration may be set as part of the operation profile agreed together with the context distribution. The Rule Id field size may range for example from 2 bits resulting in 4 rules to a 8 bits value that would yield up to 256 rules which can be used together with the operators and seems quite a reasonable maximum limit even for a device acting as a NAT. More bits could be configured, but it should take in account the byte-alignment of the expected Compression Residue too. In the minimum TB size case, 2 bits size of Rule Id leave only 6 bits available for Compression Residue.

5.3.4. SCHC MAX_PACKET_SIZE

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

5.3.5. Fragmentation

For these scenarios the SCHC fragmentation functions are recommend to be disabled. The RLC layer of NB-IoT can segment packets in suitable units that fit the selected transport blocks for transmissions of the physical layer. The selection of the blocks is done according to the input of the link adaptation function in the MAC layer and the quantity of data in the buffer. The link adaptation layer may produce different results at each Time Transmission Interval (TTI) resulting in varying physical transport blocks that depends of the network load, interference and number of bits to be transmitted and QoS. Even if setting a value that allows the construction of data units following SCHC tiles principle, the protocol overhead may be greater or equal than allowing the AS radio protocols to take care of the fragmentation natively.

5.3.5.1. Fragmentation in Transparent Mode

If RLC is configured to operate in Transparent Mode, there could be a case to activate a fragmentation function together with a light reliability function such as the ACK-Always mode. In practice , it is very rare to transmit user plane data using this configuration and it is mainly targeting control plane transmissions. In those cases the reliability is normally ensured by MAC based mechanisms, such as repetitions or automatic retransmissions, and additional reliability might only generate protocol overhead.

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

6. Non-IP based Data Transmission

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

6.1. SCHC Entities Placing

In the two scenarios using NIDD, SCHC entities are located almost in top of the stack. In the terminal, it may be implemented by a application utilizing the NB-IoT connectivity services. In the network side, the SCHC entities are located in the Application Server (AS). The IP tunneling scenario requires that the Application Server sends the compressed packet over an IP connection that is terminated by the 3GPP core network. If instead the SCEF services are used, then it is possible to utilize a API call to transfer the SCHC packets between the core network and the AS, also an IP tunnel could be established by the AS, if negotiated with the SCEF.

```
+---------+       XXXXXXXXXXXXXXXXXXXXXXXX          +--------+
| SCHC    |      XXX                    XXX         | SCHC   |
|(Non-IP) +-----XX......................XX....+--*---+(Non-IP)|
+---------+    XX                +----+  XX   |   |  +--------+
|         |    XX                |SCEF+-------+   |  |        |
|         |   XXX     3GPP RAN & +----+  XXX     +---+  UDP   |
|         |   XXX    CORE NETWORK        XXX      |  |        |
|   L2    +---+XX                +-------------+  |  +--------+
|         |     XX               |IP TUNNELING+--+   |        |
|         |      XXX             +-------------+  +---+  IP    |
+---------+       XXXX                 XXXX       |  +--------+
| PHY     +-------+ XXXXXXXXXXXXXXXXXXXXXXX       +---+  PHY   |
+---------+                                         +--------+
   UE                                                   AS
```

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

6.2. Parameters for Static Context Header Compression

6.2.1. SCHC Context initialization

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

6.2.2. SCHC Rules

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

6.2.3. Rule ID

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

6.2.4. SCHC MAX_PACKET_SIZE

In these scenarios the maximum recommended MTU size that applies is 1358 Bytes, since the SCHC packets (and fragments) are traversing the whole 3GPP network infrastructure (core and radio), and not only the radio as the IP transmissions case.

6.3. Fragmentation

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

6.3.1. Fragmentation modes

Depending of the QoS that has been assigned to the packets, it is possible that packets are lost before they arrive to 3GPP radio network transmission, for example in between the links of a capillarity gateway, or due to buffer overflow handling in a backhaul connection. In consequence, it is possible to secure additional reliability on the packets transmitted with a small trade-off on additional transmissions to signal the packets arrival indication end-to-end if no transport protocol takes care of retransmission. To achieve this, the packets fragmentation is activated with the ACK-on-Error mode enabled. In some cases, it is even desirable to keep track of all the SCHC packets delivered, in that case, the fragmentation function could be active for all packets transmitted by the applications (SCHC MAX_PACKET_SIZE == 1 Byte) and the ACK-on-Error mode.

6.3.2. Fragmentation Parameters(TBD)

- Rule ID
- DTag
- FCN
- W (number of bits)
- WINDOW_SIZE
- Retransmission Timer
- Inactivity Timer
- MAX_ACK_Retries
- MAX_ATTEMPS
- MIC (size and algorithm)

7. Padding

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

8. Security considerations

3GPP access security is specified in (TGPP33203).

9. 3GPP References

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

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

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

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

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

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

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

10. Appendix

10.1. NB-IoT User Plane protocol architecture

10.1.1. Packet Data Convergence Protocol (PDCP)

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

- o Header compression and decompression by means of ROHC (Robust Header Compression)

- o Transfer of user and control data to higher and lower layers

- o Duplicate detection of lower layer SDUs when re-establishing connection (when RLC with Acknowledge Mode in use for User Plane only)

- o Ciphering and deciphering

- o Timer-based SDU discard in uplink

10.1.2. Radio Link Protocol (RLC)

RLC is a layer-2 protocol that operates between the UE and the base station (eNB). It supports the packet delivery from higher layers to MAC creating packets that are transmitted over the air optimizing the Transport Block utilization. RLC flow of data packets is unidirectional and it is composed of a transmitter located in the transmission device and a receiver located in the destination device. Therefore to configure bi-directional flows, two set of entities, one in each direction (downlink and uplink) must be configured and they are effectively peered to each other. The peering allows the transmission of control packets (ex., status reports) between entities. RLC can be configured for data transfer in one of the following modes:

- o Transparent Mode (TM). In this mode RLC do not segment or concatenate SDUs from higher layers and do not include any header to the payload. When acting as a transmitter, RLC receives SDUs from upper layers and transmit directly to its flow RLC receiver via lower layers. Similarly, an TM RLC receiver would only deliver without additional processing the packets to higher layers upon reception.

- o Unacknowledged Mode (UM). This mode provides support for segmentation and concatenation of payload. The size of the RLC packet depends of the indication given at a particular transmission opportunity by the lower layer (MAC) and are octets aligned. The packet delivery to the receiver do not include support for reliability and the lost of a segment from a packet means a whole packet loss. Also in case of lower layer retransmissions there is no support for re-segmentation in case of change of the radio conditions triggering the selection of a smaller transport block. Additionally it provides PDU duplication detection and discard, reordering of out of sequence and loss detection.

- o Acknowledged Mode (AM). Additional to the same functions supported from UM, this mode also adds a moving windows based reliability service on top of the lower layer services. It also provides support for re-segmentation and it requires bidirectional communication to exchange acknowledgment reports called RLC Status Report and trigger retransmissions is needed. Protocol error detection is also supported by this mode. The mode uses depends of the operator configuration for the type of data to be transmitted. For example, data transmissions supporting mobility or requiring high reliability would be most likely configured using AM, meanwhile streaming and real time data would be map to a UM configuration.

10.1.3. Medium Access Control (MAC)

MAC provides a mapping between the higher layers abstraction called Logical Channels comprised by the previously described protocols to the Physical layer channels (transport channels). Additionally, MAC may multiplex packets from different Logical Channels and prioritize what to fit into one Transport Block if there is data and space available to maximize the efficiency of data transmission. MAC also provides error correction and reliability support by means of HARQ, transport format selection and scheduling information reporting from the terminal to the network. MAC also adds the necessary padding and piggyback control elements when possible additional to the higher layers data.

```
                                               <Max. 1600 bytes>
                       +---+         +---+           +------+
   Application         |AP1|         |AP1|           |  AP2 |
   (IP/non-IP)         |PDU|         |PDU|           |  PDU |
                       +---+         +---+           +------+
                       |   |         |   |           |      |
   PDCP           +--------+    +--------+      +-----------+
                  |PDCP|AP1|    |PDCP|AP1|      |PDCP|  AP2 |
                  |Head|PDU|    |Head|PDU|      |Head|  PDU |
                  +--------+    +--------+      +--------+--\
                  |    |   |    |     |  |      |    |   |\  `----\
            +---------------------------+      |    |(1)| `-----\(2)'-\
   RLC      |RLC |PDCP|AP1|RLC |PDCP|AP1| +-------------+    +----|---+
            |Head|Head|PDU|Head|Head|PDU| |RLC |PDCP|AP2|    |RLC |AP2|
            +-------------|-------------+ |Head|Head|PDU|    |Head|PDU|
            |         |   |         |   | +---------|---+    +--------+
            |         |   | LCID1   |   | /         /   /   /         /
           /         /   /        _/  _//        _/  _/    / LCID2   /
           |        |   |        |   | /       _/  _/     /      ___/
           |        |   |        |   ||       |   |      /      /
       +------------------------------------------+ +-----------+---+
   MAC |MAC|RLC|PDCP|AP1|RLC|PDCP|AP1|RLC|PDCP|AP2| |MAC|RLC|AP2|Pad|
       |Hea|Hea|Hea |PDU|Hea|Hea |PDU|Hea|Hea |PDU| |Hea|Hea|PDU|din|
       |der|der|der |   |der|der |   |der|der |   | |der|der|   |g  |
       +------------------------------------------+ +-----------+---+
                         TB1                               TB2
```

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

10.2. NB-IoT Data over NAS (DoNAS)

The AS protocol stack used by DoNAS is somehow special. Since the security associations are not established yet in the radio network, to reduce the protocol overhead, PDCP (Packet Data Convergence Protocol) is bypassed until AS security is activated. RLC (Radio Link Control protocol) is configured by default in AM mode, but depending of the features supported by the network and the terminal it may be configured in other modes by the network operator. For example, the transparent mode does not add any header or does not process the payload in any way reducing the overhead, but the MTU would be limited by the transport block used to transmit the data which is couple of thousand of bits maximum. If UM (only Release 15 compatible terminals) is used, the RLC mechanisms of reliability is disabled and only the reliability provided by the MAC layer by Hybrid Automatic Repeat reQuest (HARQ) is available. In this case, the protocol overhead might be smaller than for the AM case because the lack of status reporting but with the same support for segmentation up to 16000 Bytes. NAS packet are encapsulated within a RRC (Radio Resource Control)[TGPP36331] message.

Depending of the data type indication signaled (IP or non-IP data), the network allocates an IP address or just establish a direct forwarding path. DoNAS is regulated under rate control upon previous agreement, meaning that a maximum number of bits per unit of time is agreed per device subscription beforehand and configured in the device. The use of DoNAS is typically expected when a terminal in a power saving state requires to do a short transmission and receive an acknowledgment or short feedback from the network. Depending of the size of buffered data to transmit, the UE might be instructed to deploy the connected mode transmissions instead, limiting and controlling the DoNAS transmissions to predefined thresholds and a good resource optimization balance for the terminal and the network. The support for mobility of DoNAS is present but produces additional overhead.

```
       +--------+   +--------+   +--------+
       |        |   |        |   |        |       +-----------------+
       |   UE   |   |  C-BS  |   |  C-SGN |       |Roaming Scenarios|
       +----|---+   +--------+   +--------+       |  +--------+     |
            |            |            |           |  |        |     |
        +---------------|------------|+           |  |  P-GW  |     |
        |       Attach                |           |  +--------+     |
        +------------------------------+          |       |         |
            |            |            |           |       |         |
     +------|------------|--------+   |           |       |         |
     |RRC Connection Establishment|   |           |       |         |
     |with NAS PDU transmission   |   |           |       |         |
     |& Ack Rsp                   |   |           |       |         |
     +----------------------------+   |           |       |         |
            |            |            |           |       |         |
            |            |Initial UE  |           |       |         |
            |            |message     |           |       |         |
            |            |----------->|           |       |         |
            |            |            |           |       |         |
            |            | +---------------------+|       |         |
            |            | |Checks Integrity     ||       |         |
            |            | |protection, decrypts ||       |         |
            |            | |data                 ||       |         |
            |            | +---------------------+|       |         |
            |            |            |       Small data packet     |
            |            |            |-------------------------------->
            |            |            |       Small data packet     |
            |            |            |<-------------------------------
            |            | +----------|---------+ |       |         |
            |            | Integrity protection,| |       |         |
            |            | encrypts data        | |       |         |
            |            | +--------------------+ |       |         |
            |            |            |           |       |         |
            |            |Downlink NAS|           |       |         |
            |            |message     |           |       |         |
            |            |<-----------|           |       |         |
    +-----------------------+         |           |       |         |
    |Small Data Delivery,    |         |           |       |         |
    |RRC connection release |         |           |       |         |
    +-----------------------+         |           |       |         |
                                                  |                 |
                                                  |                 |
                                                  +-----------------+
```

Figure 7: DoNAS transmission sequence from an Uplink initiated access

```
                      +---+ +---+ +---+                  +----+
    Application       |AP1| |AP1| |AP2|                  |AP2 |
   (IP/non-IP)        |PDU| |PDU| |PDU|  ............... |PDU |
                      +---+ +---+ +---+                  +----+
                      |   |/   /  |    \                 |    |
   NAS /RRC      +--------+---|---+----+            +---------+
                 |NAS/|AP1|AP1|AP2|NAS/|            |NAS/|AP2 |
                 |RRC |PDU|PDU|PDU|RRC |            |RRC |PDU |
                 +--------+-|-+---+----+            +---------|
                 |          |\         |            |         |
                 |<--Max. 1600 bytes-->|__          |_        |
                 |          |  \__        \___        \_       \_
                 |          |     \           \         \__      \_
            +---------------|+-----|----------+            \       \
   RLC      |RLC | NAS/RRC  ||RLC  | NAS/RRC  |       +----|-------+
            |Head|  PDU(1/2)||Head | PDU (2/2)|       |RLC |NAS/RRC|
            +---------------++----------------+       |Head|PDU    |
            |    |          | \               |       +------------+
            |    |    LCID1 |  \              |       |           /
            |    |          |   \              \      |           |
            |    |          |    \              \     |           |
            |    |          |     \              \     \          |
       +----+----+----------++-----|----+---------++----+---------|---+
   MAC |MAC |RLC |    RLC   ||MAC  |RLC |  RLC    ||MAC |  RLC    |Pad|
       |Head|Head|  PAYLOAD ||Head |Head| PAYLOAD ||Head|  PDU    |   |
       +----+----+----------++-----+----+---------++----+---------+---+
                TB1                   TB2                     TB3
```

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

11. Informative References

[I-D.ietf-lpwan-ipv6-static-context-hc]
Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and J. Zuniga, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-18 (work in progress), December 2018.

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

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

Authors' Addresses

Ana Minaburo
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne Cedex
France

Email: ana@ackl.io

Edgar Ramos
Ericsson
Hirsalantie 11
02420 Jorvas, Kirkkonummi
Finland

Email: edgar.ramos@ericsson.com

Sivasothy Shanmugalingam
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne Cedex
France

Email: sothy@ackl.io

lpwan Working Group N. Sornin, Ed.
Internet-Draft M. Coracin
Intended status: Informational Semtech
Expires: August 17, 2019 I. Petrov
Acklio
A. Yegin
Actility
J. Catalano
Kerlink
V. Audebert
EDF R&D
February 13, 2019

Static Context Header Compression (SCHC) over LoRaWAN
draft-petrov-lpwan-ipv6-schc-over-lorawan-03

Abstract

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

This document provides the adaptation of SCHC for use in LoRaWAN networks, and provides elements such as efficient parameterization and modes of operation.

Copyright Notice

Table of Contents

1. Introduction

The Static Context Header Compression (SCHC) specification [I-D.ietf-lpwan-ipv6-static-context-hc] describes generic header compression and fragmentation techniques that can be used on all LPWAN (Low Power Wide Area Networks) technologies defined in [I-D.ietf-lpwan-overview]. Even though those technologies share a great number of common features like start-oriented topologies, network architecture, devices with mostly quite predictable communications, etc; they do have some slight differences in respect of payload sizes, reactiveness, etc.

SCHC gives a generic framework that enables those devices to communicate with other Internet 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 efficient parameters and modes of operation when SCHC is used over LoRaWAN networks.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This section defines the terminology and acronyms used in this document. For all other definitions, please look up the SCHC specification [I-D.ietf-lpwan-ipv6-static-context-hc].

o DevEUI: an IEEE EUI-64 identifier used to identify the device during the procedure while joining the network (Join Procedure)

o DevAddr: a 32-bit non-unique identifier assigned to a device statically or dynamically after a Join Procedure (depending on the activation mode)

o TBD: all significant LoRaWAN-related terms.

3. Static Context Header Compression Overview

This section contains a short overview of Static Context Header Compression (SCHC). For a detailed description, refer to the full specification [I-D.ietf-lpwan-ipv6-static-context-hc].

Static Context Header Compression (SCHC) avoids context synchronization, which is the most bandwidth-consuming operation in other header compression mechanisms such as RoHC [RFC5795]. Based on the fact that the nature of data flows is highly predictable in LPWAN networks, some static contexts may be stored on the Device (Dev). The contexts must be stored in both ends, and it can either be learned by a provisioning protocol or by out of band means or it can be pre-provisioned, etc. The way the context is learned on both sides is out of the scope of this document.

```
        Dev                                                   App
   +--------------+                                  +--------------+
   |APP1 APP2 APP3|                                  |APP1 APP2 APP3|
   |              |                                  |              |
   |      UDP     |                                  |      UDP     |
   |     IPv6     |                                  |     IPv6     |
   |              |                                  |              |
   |   SCHC C/D   |                                  |              |
   |   (context)  |                                  |              |
   +-------+------+                                  +-------+------+
           |   +--+     +----+     +---------+                 .
           +~~ |RG| === |NGW | === |SCHC C/D |... Internet ..
               +--+     +----+     |(context)|
                                   +---------+
```

Figure 1: Architecture

Figure 1 represents the architecture for compression/decompression, it is based on [I-D.ietf-lpwan-overview] terminology. The Device is sending applications flows using IPv6 or IPv6/UDP protocols. These flows are compressed by an Static Context Header Compression Compressor/Decompressor (SCHC C/D) to reduce headers size. Resulting information is sent on a layer two (L2) frame to a LPWAN Radio Network (RG) which forwards the frame to a Network Gateway (NGW). The NGW sends the data to a SCHC C/D for decompression which shares the same rules with the Dev. The SCHC C/D can be located on the Network Gateway (NGW) or in another place as long as a tunnel is established between the NGW and the SCHC C/D. The SCHC C/D in both sides 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 process is bidirectional, so the same principles can be applied in the other direction.

In a LoRaWAN network, the RG is called a Gateway, the NGW is Network Server, and the SCHC C/D can be embedded in different places, for example in the Network Server and/or the Application Server.

Next steps for this section: detailed overview of the LoRaWAN architecture and its mapping to the SCHC architecture.

4. LoRaWAN Architecture

An overview of LoRaWAN [lora-alliance-spec] protocol and architecture is described in [I-D.ietf-lpwan-overview]. Mapping between the LPWAN architecture entities as described in

[I-D.ietf-lpwan-ipv6-static-context-hc] and the ones in [lora-alliance-spec] is as follows:

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. This entity maps to the LoRaWAN End-device.

o The Radio Gateway (RGW), which is the end point 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 Internet. This entity maps to the LoRaWAN Network Server.

o LPWAN-AAA Server, which controls the user authentication and the applications. This entity maps to the LoRaWAN Join Server.

o Application Server (App). The same terminology is used in LoRaWAN.

```
 ()    ()    ()        |                      +------+
  ()   ()  ()  ()      / \       +---------+   | Join |
 ()  ()  ()  ()  ()    /   \======|    ^    |===|Server|  +-----------+
  ()  ()   ()        |            | <--|--> |   +------+  |Application|
 ()  ()   ()   ()   / \==========|    v    |=============|   Server  |
  ()   ()   ()    /   \          +---------+              +-----------+
 End-Devices  Gateways     Network Server
```

Figure 2: LPWAN Architecture

SCHC C/D (Compressor/Decompressor) and SCHC Fragmentation are performed on the LoRaWAN End-device and the Application Server. While the point-to-point link between the End-device and the Application Server constitutes 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 acts as the first hop IP router for the End-device. Note that 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 classA, some devices implement classA+B or class A+C. ClassB and classC are mutually exclusive.

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

- *ClassB*: classB devices implement all the functionalities of classA devices, but also schedule periodic listen windows. Therefore, as opposed the classA devices, classB devices can receive downlink that are initiated by the network and not following an uplink. There is a trade-off between the periodicity of those scheduled classB listen windows and the power consumption of the device. The lower the downlink latency, the higher the power consumption.

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

4.2. Device addressing

LoRaWAN devices use a 32bits network address (devAddr) to communicate with the network over the air. However that address might be reused several time on the same network at the same time for different devices. Devices using the same devAddr are distinguish by the network server based on the cryptographic signature appended to every single LoRaWAN MAC frame, as all devices use different security keys. To communicate with the SCHC gateway the network server MUST identify the devices by a unique 64bits device ID called the devEUI. Unlike devAddr, devEUI is guaranteed to be unique for every single device across all networks. The devEUI is assigned to the device during the manufacturing process by the device's manufacturer. The devEUI is built like an Ethernet MAC address by concatenating the manufacturer's IEEE 24bits OUI field with a 40bits serial number. The network server translates the devAddr into a devEUI in the uplink direction and reciprocally on the downlink direction.

```
+--------+         +---------------+        +--------------------+
| device | <=====> | Network Server| <====> | Application Server |
+--------+ devAddr +---------------+ devEUI +--------------------+
```

Figure 3: LoRaWAN addresses

4.3. General Message Types

- o Confirmed messages:

- o Unconfirmed messages:

4.4. LoRaWAN MAC Frames

- o JoinRequest

- o JoinAccept

- o Data

5. SCHC over LoRaWAN

5.1. Rule ID management

The LoRaWAN MAC layers features a port field in all frames. This port field (FPort) is 8bit long and the values from 1 to 220 can be used. SCHC over LoRaWAN uses 2 contiguous FPort value to separate the uplink SCHC traffic from the downlink and avoid any confusion. Those FPorts are called FPortUp and FPortDwn. Those FPorts can use arbitrary values inside the allowed Fport range but must be shared by the end-device and SCHC gateway.

SCHC over LoRAWAN SHOULD support encoding RuleID on 3 bits, there are therefore 8 possible RuleIds on both uplink and downlink direction.

The RuleID 0 is reserved for fragmentation in both directions. The 7 remaining RuleIDs are available for IPV6 header compression. Uplink (on FPortUp) and downlink (on FportDwn) RuleIDs are independent. The same RuleID may have different meanings on the uplink and downlink paths.

The only uplink messages using the FportDwn port are the fragmentation SCHC ACKs messages of a downlink fragmentation session. Similarly, the only downlink messages using the FportUp port are the fragmentation SCHC ACKs messages of an uplink fragmentation session

5.2. IID computation

TBD (To discuss with the SCHC authors).

5.3. No compression packets are sent using Rule ID 7.

5.4. Fragmentation

The L2 word size used by LoRaWAN is 1 octet (8 bits). The SCHC fragmentation over LoRaWAN exclusively uses the ACK-always mode. A LoRaWAN device cannot support simultaneous interleaved fragmentation sessions in the same direction (uplink or downlink). This means that only a single fragmented IPV6 datagram may be transmitted and/or received by the device at a given moment. The fragmentation parameters are different for uplink and downlink fragmentation sessions and are successively described in the next sections.

5.4.1. Uplink fragmentation: From device to gateway

In that case the device is the fragmentation transmitter, and the SCHC gateway the fragmentation receiver.

- SCHC fragmentation reliability mode : "ACK_ALWAYS"

- Window size: 8, the FCN field is encoded on 3 bits

- DTag : 1bit. this field is used to clearly separate two consecutive fragmentation sessions. A LoRaWAN device cannot interleave several fragmented SCHC datagrams.

- MIC calculation algorithm: CRC32 using 0xEDB88320 (i.e. the reverse representation of the polynomial used e.g. in the Ethernet standard [RFC3385])

- Retransmission Timer and inactivity Timer: LoRaWAN devices do not implement a "retransmission timer". At the end of a window the ACK corresponding to this window is transmitted by the network gateway in the RX1 or RX2 receive slot of the device. If this ACK is not received the device sends an all-0 (or an all-1) fragment with no payload to request an ACK retransmission. The periodicity between retransmission of the all-0/all-1 fragments is device/ application specific and may be different for each device (not specified). The gateway implements an "inactivity timer". The default recommended duration of this timer is 12h. This value is mainly driven by application requirements and may be changed.

```
| RuleID | DTag  | W     | FCN    | Payload |
+ ------ + ----- + ----- | ------ + ------- +
| 3 bits | 1 bit | 1 bit | 3 bits |         |
```

Figure 4: All fragment except the last one. Header size is 8 bits.

```
| RuleID | DTag  | W     | FCN    | MIC     | Payload |
+ ------ + ----- + ----- | ------ + ------- + ------- +
| 3 bits | 1 bit | 1 bit | 3 bits | 32 bits |         |
```

Figure 5: All-1 fragment detailed format for the last fragment. Header size is 8 bits.

The format of an all-0 or all-1 acknowledge is:

```
| RuleID | DTag  | W     | Encoded bitmap | Padding (0s) |
+ ------ + ----- + ----- | -------------- + ------------ +
| 3 bits | 1 bit | 1 bit | 3 or 8 bits    | 0 or 3 bits  |
```

Figure 6: ACK format for All-0 windows. Header size is 1 or 2 bytes.

```
| RuleID | DTag  | W     | C     | Encoded bitmap (if C = 0) | Padding (0s) |
+ ------ + ----- + ----- + ----- + ------------------------- + ------------ +
| 3 bits | 1 bit | 1 bit | 1 bit | 2 or 8 bits               | 0 or 2 bits  |
```

Figure 7: ACK format for All-1 windows. Header size is 1 or 2 bytes.

5.4.2. Downlinks: From gateway to device

In that case the device is the fragmentation receiver, and the SCHC gateway the fragmentation transmitter. The following fields are common to all devices.

- o SCHC fragmentation reliability mode : ACK_ALWAYS

- o Window size : 1 , The FCN field is encoded on 1 bits

- o DTag : 1bit. This field is used to clearly separate two consecutive fragmentation sessions. A LoRaWAN device cannot interleave several fragmented SCHC datagrams.

- o MIC calculation algorithm: CRC32 using 0xEDB88320 (i.e. the reverse representation of the polynomial used e.g. in the Ethernet standard [RFC3385]) o MAX_ACK_REQUESTS : 8

```
| RuleID | DTag  | W     | FCN    | Payload           |
+ ------ + ----- + ----- | ------ + ------- + ------- +
| 3 bits | 1 bit | 1 bit | 1 bits | X bytes + 2 bits  |
```

Figure 8: All fragments but the last one. Header size is 6 bits.

```
| RuleID | DTag  | W     | FCN    | MIC     | Payload | Padding (0s) |
+ ------ + ----- + ----- | ------ + ------- + ------- + ------------ +
| 3 bits | 1 bit | 1 bit | 1 bits | 32 bits | X bytes | 0 to 7 bits  |
```

Figure 9: All-1 Fragment Detailed Format for the Last Fragment. Header size is 6 bits.

The format of an all-0 or all-1 acknowledge is:

```
| RuleID | DTag  | W     | Encoded bitmap | Padding (0s) |
+ ------ + ----- + ----- | -------------- + ------------ +
| 3 bits | 1 bit | 1 bit | 1 bit          | 2 bits       |
```

Figure 10: ACK format for All-0 windows. Header size is 8 bits.

```
| RuleID | DTag  | W     | C = 1 | Padding (0s) |
+ ------ + ----- + ----- + ----- + ------------ +
| 3 bits | 1 bit | 1 bit | 1 bit | 2 bits       |
```

Figure 11: ACK format for All-1 windows, MIC is correct. Header size is 8 bits.

```
| RuleID | DTag  | W     | b'111  | 0xFF (all 1's) |
+ ------ + ----- + ----- + ------ + -------------- +
| 3 bits | 1 bit | 1 bit | 3 bits | 8 bits         |
```

Figure 12: Receiver ABORT packet (following an all-1 packet with incorrect MIC). Header size is 16 bits.

Class A and classB&C devices do not manage retransmissions and timers in the same way.

5.4.2.1. Class A devices

Class A devices can only receive in an RX slot following the transmission of an uplink. Therefore there cannot be a concept of "retransmission timer" for a gateway talking to classA devices for downlink fragmentation.

The device replies with an ACK fragment to every single fragment received from the gateway (because the window size is 1). Following the reception of a FCN=0 fragment (fragment that is not the last fragment of the packet or ACK-request), the device MUST transmit the ACK fragment until it receives the fragment of the next window. The device shall transmit up to MAX_ACK_REQUESTS ACK fragments before aborting. The device should transmit those ACK as soon as possible while taking into consideration eventual local radio regulation on duty-cycle, to progress the fragmentation session as quickly as possible. The ACK bitmap is 1 bit long and is always 1.

Following the reception of a FCN=1 fragment (the last fragment of a datagram) and if the MIC is correct, the device shall transmit the ACK with the "MIC is correct" indicator bit set. This message might be lost therefore the gateway may request a retransmission of this ACK in the next downlink. The device SHALL keep this ACK message in memory until it receives a downlink from the gateway different from an ACK-request indicating that the gateway has received the ACK message.

Following the reception of a FCN=1 fragment (the last fragment of a datagram) and if the MIC is NOT correct, the device shall transmit a receiver-ABORT fragment. The device SHALL keep this ABORT message in memory until it receives a downlink from the gateway different from an ACK-request indicating that the gateway has received the ABORT message. The fragmentation receiver (device) does not implement retransmission timer and inactivity timer.

The fragmentation sender (the gateway) implements an inactivity timer with default duration 12 hours. Once a fragmentation session is started, if the gateway has not received any ACK or receiver-ABORT message 12 hours after the last message from the device was received, the gateway may flush the fragmentation context. For devices with very low transmission rates (example 1 packet a day in normal operation) , that duration may be extended, but this is application specific.

5.4.2.2. Class B or C devices

Class B&C devices can receive in scheduled RX slots or in RX slots following the transmission of an uplink. The device replies with an ACK fragment to every single fragment received from the gateway (because the window size is 1). Following the reception of a FCN=0 fragment (fragment that is not the last fragment of the packet or ACK-request), the device MUST always transmit the corresponding ACK fragment even if that fragment has already been received. The ACK bitmap is 1 bit long and is always 1. If the gateway receives this ACK, it proceeds to send the next window fragment If the retransmission timer elapses and the gateway has not received the ACK of the current window it retransmits the last fragment. The gateway tries retransmitting up to MAX_ACK_REQUESTS times before aborting.

Following the reception of a FCN=1 fragment (the last fragment of a datagram) and if the MIC is correct, the device shall transmit the ACK with the "MIC is correct" indicator bit set. If the gateway receives this ACK, the current fragmentation session has succeeded and its context can be cleared.

If the retransmission timer elapses and the gateway has not received the all-1 ACK it retransmits the last fragment with the payload (not an ACK-request without payload). The gateway tries retransmitting up to MAX_ACK_REQUESTS times before aborting.

The device SHALL keep the all-1 ACK message in memory until it receives a downlink from the gateway different from the last (FCN=1) fragment indicating that the gateway has received the ACK message. Following the reception of a FCN=1 fragment (the last fragment of a datagram) and if the MIC is NOT correct, the device shall transmit a receiver-ABORT fragment. The retransmission timer is used by the gateway (the sender), the optimal value is very much application specific but here are some recommended default values. For classB devices, this timer trigger is a function of the periodicity of the classB ping slots. The recommended value is equal to 3 times the classB ping slot periodicity. For classC devices which are nearly constantly receiving, the recommended value is 30 seconds. This means that the device shall try to transmit the ACK within 30 seconds of the reception of each fragment. The inactivity timer is implemented by the device to flush the context in-case it receives nothing from the gateway over an extended period of time. The recommended value is 12 hours for both classB&C devices.

6. Security considerations

As this document is only providing parameters that are expected to be better suited for LoRaWAN networks for [I-D.ietf-lpwan-ipv6-static-context-hc]. As such, this parameters does not contribute to any new security issues in addition of those identified in [I-D.ietf-lpwan-ipv6-static-context-hc].

7. Acknowledgements

TBD

8. References

8.1. 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>.

[RFC3385] Sheinwald, D., Satran, J., Thaler, P., and V. Cavanna, "Internet Protocol Small Computer System Interface (iSCSI) Cyclic Redundancy Check (CRC)/Checksum Considerations", RFC 3385, DOI 10.17487/RFC3385, September 2002, <https://www.rfc-editor.org/info/rfc3385>.

[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>.

[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>.

8.2. Informative References

[I-D.ietf-lpwan-ipv6-static-context-hc]
Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and J. Zuniga, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-18 (work in progress), December 2018.

[I-D.ietf-lpwan-overview]
Farrell, S., "LPWAN Overview", draft-ietf-lpwan-overview-10 (work in progress), February 2018.

[lora-alliance-spec]
Alliance, L., "LoRaWAN Specification Version V1.0.2", <http://portal.lora-alliance.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=1398>.

Appendix A. Examples

Appendix B. Note

Authors' Addresses

Nicolas Sornin (editor)
Semtech
14 Chemin des Clos
Meylan
France

Email: nsornin@semtech.com

Michael Coracin
Semtech
14 Chemin des Clos
Meylan
France

Email: mcoracin@semtech.com

Ivaylo Petrov
Acklio
2bis rue de la Chataigneraie
35510 Cesson-Sevigne Cedex
France

Email: ivaylo@ackl.io

Alper Yegin
Actility
.
Paris, Paris
France

Email: alper.yegin@actility.com

Julien Catalano
Kerlink
1 rue Jacqueline Auriol
35235 Thorigne-Fouillard
France

Email: j.catalano@kerlink.fr

Vincent AUDEBERT
EDF R&D
7 bd Gaspard Monge
91120 PALAISEAU
FRANCE

Email: vincent.audebert@edf.fr

Network Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Informational L. Toutain
Expires: September 12, 2019 Institut MINES TELECOM ; IMT Atlantique
March 11, 2019

Data Model for Static Context Header Compression (SCHC)
draft-toutain-lpwan-schc-yang-data-model-00

Abstract

This document describes a YANG data model for the SCHC (Static Context Header Compression). A generic module is defined, that can be applied for any headers and also a specific model for the IPv6 UDP protocol stack is also proposed. Note that this draft is a first attempt to define a YANG data module for SCHC, more work is needed to use all the YANG facilities.

Status of This Memo

Copyright Notice

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

SCHC [I-D.ietf-lpwan-ipv6-static-context-hc] defines a compression technique for LPWAN networks based on static context. The context contains a list of rules (cf. Figure 1). Each rule contains itself a list of field descriptions composed of a field identifier (FID), a field length (FID), a field position (FP), a field direction (DI), a target value (TV), a matching operator (MO) and a Compression/ Decompression Action (CDA).

```
  +-----------------------------------------------------------------+
  |                      Rule N                                     |
 +-----------------------------------------------------------------+|
 |                    Rule i                                       ||
+-----------------------------------------------------------------+||
|  (FID)            Rule 1                                        |||
|+-------+--+--+--+------------+-----------------+---------------+|||
||Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act||||
|+-------+--+--+--+------------+-----------------+---------------+|||
||Field 2|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act||||
|+-------+--+--+--+------------+-----------------+---------------+|||
||...    |..|..|..|   ...      | ...             | ...           ||||
|+-------+--+--+--+------------+-----------------+---------------+||/
||Field N|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|||
|+-------+--+--+--+------------+-----------------+---------------+|/
|                                                                 |
\-----------------------------------------------------------------/
```

Figure 1: Compression Decompression Context

The goal of this document is to provide an YANG data model to represent SCHC Compression and Fragmentation rules, to allow management over a LPWAN network. The main constraints are:

- o since the device may be managed through the LPWAN network, management messages must be compact. COREconf offers a representation based on CBOR.

- o this data model can be extended with new values, such as new field id, new MO or CDA.

2. YANG types

2.1. Field Identifier

The field identifier is used to identify a specific field. It is viewed as an uint32.

2.2. Target Value field

A value may be associated for each field in a rule. The value's type depends on the field. It can be an integer, a prefix, a string, or any other type carried by the field. The LPWA-types regroups all the possibles values. Figure 2 gives its definition.

```
typedef lpwan-types {
  type union {
    type uint8;
    type uint16;
    type uint32;
    type uint64;
    type inet:ipv6-prefix;
    type string;
   }
}
```

Figure 2: Value types

Note that as defined in [I-D.ietf-lpwan-ipv6-static-context-hc], Dev and App Prefixes can be of type inet:ipv6-prefix-type, but this type derives from ASCII characters, a binary representation such as uint64 will be more compact.

2.3. Matching Operators

A matching operator is used to check the field value stored in the rule against the value contained in the header field. If there is no matching the rule is not selected. Two instances of matching operator are defined to allow the rule selection from informations contained either in the compressed header or the uncompressed header. The SCHC document [I-D.ietf-lpwan-ipv6-static-context-hc] defines four operators:

o equal: The rule's value must be equal to the packet header value for a specific field.

- ignore: There is no check for this field.

- MSB(x): This operator compare the most significant bits. The operator takes one argument representing the length of least significant bit part, which will be ignored during the matching but sent if the rule matches.

- match-mapping: From the list of values of the Target Value, This operator will match if one of those values is equal to the field value and will send the index of the list representing this value.

```
/**********************************/
/* Matching operator type         */
/**********************************/
typedef matching-operator-type {
    type enumeration {
        enum equal;
        enum ignore;
        enum msb;
        enum match-mapping;
    }
}
```

Figure 3: Matching operators

Figure 3 represents the Matching Operator type definition.

2.4. Compression Decompression Actions

The SCHC document [I-D.ietf-lpwan-ipv6-static-context-hc] defines some compression decompression actions (CDA). The CDA tells how to compress and decompress the field. They are defined in Figure 4. they are coded the same way as MO.

```
           /********************************************/
           /* Compression-Decompression action type     */
           /********************************************/
           typedef compression-decompression-action-type {
                           type enumeration {
                                   enum not-sent;
                                   enum value-sent;
                                   enum lsb;
                                   enum mapping-sent;
                                   enum compute-length;
                                   enum compute-checksum;
                                   enum esiid-did;
                                   enum laiid-did;
                           }
           }
```

Figure 4: Action functions

3. Generic rule definition

Each rule's row is defined by several leaves, composed of:

- a field key which will be used as a key,
- a field name that can be used for debugging purpose,
- a field length that containing the length of the field,
- a field position that gives the number of instances,
- a field direction indicates the packet direction,
- a field target value containing the value that will be compared,
- a matching operators for rule selection
- an compression/decompression action to compress/decompress the field.

Figure 5 defines the format.

```
        grouping rule-entry {
                leaf field-id {
                        type int32;
                        description "Field ID unique value representing the Fiel
d";
                }
```

```
                leaf field-length {
                        type uint8;
                        description "size in bits of the field";
                }

                leaf field-position {
                        type uint8;
                        description "For repeated fields, we need to be able to
                                     distinguish between successive occurences";
                }

                leaf direction {
                     type direction-type;
                }
                list target-values {
                     key tv-key;
                     leaf tv-key {
                        type int8;
                     }
                     leaf target-value {
                        type lpwan-types;
                     }
                     description "Target Values can be a list of value, for
                                  match-mapping. For other MO, only one entry is
 specified";
                }
                leaf matching-operator {
                     type matching-operator-type;
                }

                leaf matching-operator-parameter {
                     type lpwan-types;
                     description "If the matching operator requires a parameter
                                  (for example lsb or msb), the value is provide
d here.";
                }

                leaf compression-decompression-action {
                     type compression-decompression-action-type;
                }

                leaf compression-decompression-action-parameter {
                     type lpwan-types;
                     description "If the matching operator requires a parameter
                                  (for example lsb or msb), the value is provide
d here.";
                }
        }
```

Figure 5: Action functions

4. YANG static context model

This lead to the generic rule definition, represented Figure 7. It defines a set of rules.

```
  grouping compression-rule {

    leaf rule-id {
            type uint8;
            description "The number of the context rule that should be applied."
;
    }
    leaf rule-id-length {
            type uint8;
    }

    list rule-fields {
      key "field-id field-position direction";
      uses rule-entry;
    }

  }
```

Figure 6: YANG definition of the generic module

```
 module: ietf-lpwan-schc-rule
   +--rw rule-id?          uint8
   +--rw rule-id-length?   uint8
   +--rw rule-fields* [field-id field-position direction]
      +--rw field-id                                     int32
      +--rw field-length?                                uint8
      +--rw field-position                               uint8
      +--rw direction                                    direction-type
      +--rw target-values* [tv-key]
      |  +--rw tv-key          int8
      |  +--rw target-value?   lpwan-types
      +--rw matching-operator?                           m.-o.-type
      +--rw matching-operator-parameter?                 lpwan-types
      +--rw compression-decompression-action?            c.-d.-a.-type
      +--rw compression-decompression-action-parameter?  lpwan-types
```

Figure 7: Generic module tree

The YANG tree is given Figure 7.

```
SID        Assigned to
---------  --------------------------------------------------
60000      node /rule-fields
60001      node /rule-fields/compression-decompression-action
60002      node /rule-fields/compression-decompression-action-parameter
60003      node /rule-fields/direction
60004      node /rule-fields/field-id
60005      node /rule-fields/field-length
60006      node /rule-fields/field-position
60007      node /rule-fields/matching-operator
60008      node /rule-fields/matching-operator-parameter
60009      node /rule-fields/target-values
60010      node /rule-fields/target-values/target-value
60011      node /rule-fields/target-values/tv-key
60012      node /rule-id
60013      node /rule-id-length

File ietf-lpwan-schc-rule@2016-10-31.sid created
Number of SIDs available : 100
Number of SIDs assigned : 14
```

Figure 8: Example of SID allocation

Figure 8 gives a simple allocation for SID value. SID values from 100 to 113 are used for /generic-rules/context-rules/rule-fields/field-compression-decompression-action. SID value from 1009 to 1012 are used in /generic-rules/context-rules/rule-fields/field-matching-operator.

5. Acknowledgement

The authors would like to thank Michel Veillette, Alexander Pelov, Antoni Markovski for their help on COMI/CoOL and YANG.

6. Normative References

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-04 (work in progress), November 2018.

[I-D.ietf-lpwan-ipv6-static-context-hc]
Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and J. Zuniga, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-18 (work in progress), December 2018.

Authors' Addresses

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

Email: ana@ackl.io

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

Email: Laurent.Toutain@imt-atlantique.fr

lpwan Working Group JC. Zuniga
Internet-Draft SIGFOX
Intended status: Informational C. Gomez
Expires: September 12, 2019 Universitat Politecnica de Catalunya
L. Toutain
IMT-Atlantique
March 11, 2019

SCHC over Sigfox LPWAN
draft-zuniga-lpwan-schc-over-sigfox-06

Abstract

The Static Context Header Compression (SCHC) specification describes a header compression scheme and a fragmentation functionality for Low Power Wide Area Network (LPWAN) technologies. SCHC offers a great level of flexibility that can be tailored for different LPWAN technologies.

The present document provides the optimal parameters and modes of operation when SCHC is implemented over a Sigfox LPWAN.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Static Context Header Compression (SCHC) specification [I-D.ietf-lpwan-ipv6-static-context-hc] defines a header compression scheme and a fragmentation functionality. Both can be used on top of all the LWPAN systems defined in [RFC8376] . These LPWAN systems have similar characteristics such as star-oriented topologies, network architecture, connected devices with built-in applications, etc.

SCHC offers a great level of flexibility to accommodate all these LPWAN systems. Even though there are a great number of similarities between LPWAN technologies, some differences exist with respect to the transmission characteristics, payload sizes, etc. Hence, there are optimal parameters and modes of operation that can be used when SCHC is used on top of a specific LPWAN.

This document describes the recommended parameters and modes of operation to be used when SCHC is implemented over a Sigfox LPWAN.

2. Terminology

It is assumed that the reader is familiar with the terms and mechanisms defined in [RFC8376] and in [I-D.ietf-lpwan-ipv6-static-context-hc].

3. Static Context Header Compression

The Static Context Header Compression (SCHC) described in [I-D.ietf-lpwan-ipv6-static-context-hc] takes advantage of the predictability of data flows existing in LPWAN networks to avoid context synchronization. Nonetheless, these contexts must be stored and configured on both ends. This can be done either by using a provisioning protocol, by out of band means, or by pre-provisioning them (for instance at manufacturing time). The way the contexts are configured and stored on both ends is out of the scope of this document.

```
        Dev                                                  App
   +----------------+                                   +--------------+
   | APP1 APP2 APP3 |                                   |APP1 APP2 APP3|
   +----------------+                                   +--------------+
   |   UDP  |       |                                   |     UDP      |
   |  IPv6  |       |                                   |    IPv6      |
   +--------+       |                                   |              |
   |SCHC C/D and F/R|                                   |              |
   |                |                                   |              |
   +--------+-------+                                   +-------+------+
            $   +--+      +----+      +-----------+              .
            +~~ |RG| === |NGW | ===  |    SCHC   |... Internet ..
                +--+      +----+      |F/R and C/D|
                                      +-----------+
```

Figure 1: Architecture

Figure 1 represents the architecture for compression/decompression and fragmentation/reassembly, which is based on [RFC8376] terminology, where the Radio Gateway is a Sigfox Base Station and the Network Gateway is the Sigfox Cloud.

The Device is sending applications flows that are compressed and/or fragmented by a Static Context Header Compression Compressor/Decompressor (SCHC C/D) to reduce headers size and/or fragment the packet. The resulting information is sent over a layer two (L2) frame to a LPWAN Radio Gateway (RG) which forwards the frame to a Network Gateway (NGW).

4. SCHC over Sigfox

In the case of the global Sigfox network, RGs (or base stations) are distributed over the multiple countries where the Sigfox LPWAN service is provided. On the other hand, the NGW (or Cloud-based Core network) is a single entity that connects to all Sigfox base stations in the world.

Uplink Sigfox transmissions occur in repetitions over different times and frequencies. Besides these time and frequency diversities, the Sigfox network also provides space diversity, as potentially an uplink message will be received by several base stations. Since all messages are self-contained and base stations forward them all back to the same Core network (NGW), multiple input copies can be combined at the NGW and hence provide for extra reliability based on the triple diversity (i.e. time, space and frequency). A detailed description of the Sigfox Radio Protocol can be found in [sigfox-spec].

The NGW communicates with the Network SCHC C/D for compression/decompression and/or for fragmentation/reassembly. The Network SCHC C/D shares the same set of rules as the Dev SCHC C/D. The Network SCHC C/D can be collocated with the NGW or it could be in another place, as long as a tunnel is established between the NGW and the SCHC C/D. 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 process is bidirectional, so the same principles can be applied on both uplink and downlink.

4.1. SCHC Rules

The RuleID MUST be sent at the beginning of the SCHC header. The total number of rules to be used affects directly the Rule ID field size, and therefore the total size of the fragmentation header. For this reason, it is recommended to keep the number of rules that are defined for a specific device to the minimum possible.

4.2. Packet processing

TBD

5. Fragmentation

The SCHC specification [I-D.ietf-lpwan-ipv6-static-context-hc] defines a generic fragmentation functionality that allows sending data packets larger than the maximum size of a Sigfox data frame.

The functionality also defines a mechanism to send reliably multiple frames, by allowing to resend selectively any lost frames.

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 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.

5.1. Fragmentation headers

A list of fragmentation header fields, their sizes as well as suggested modes for SCHC fragmentation over Sigfox are provided in this section.

5.2. Uplink fragment transmissions

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

Since uplink transmissions occur asynchronously, an SCHC fragment can be transmitted at any given time by the Dev.

5.2.1. Uplink No-ACK mode

No-ACK is RECOMMENDED to be used for transmitting short, non-critical packets that require fragmentation.

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

The recommended Rule ID size is: 2 bits

The recommended DTag size (T) is: 2 bits

Fragment Compressed Number (FCN) size (N): 4 bits

As per [I-D.ietf-lpwan-ipv6-static-context-hc], in the No-ACK mode the W (window) field is not present.

When fragmentation is used to transport IP frames, the Message Integrity Check (MIC) size, M: TBD bits

The algorithm for computing the MIC field MUST be TBD.

5.2.2. Uplink ACK-Always mode

TBD

5.2.3. Uplink ACK-on-Error mode

ACK-on-Error is RECOMMENDED for larger packets that need to be sent reliably, since it leads to a reduced number of ACKs in the lower capacity downlink channel.

In the most generic case, the Fragmentation Header size is 8 bits and it is composed as follows:

The recommended Rule ID size is: 2 bits.

The recommended DTag size (T) is: 1 bit.

The recommended Window (W) size is: 2 bits.

Fragment Compressed Number (FCN) size (N): 3 bits.

For the ACK-on-Error fragmentation mode(s), a single window size is RECOMMENDED.

The value of MAX_ACK_REQUESTS SHOULD be 2, and the value of MAX_WIND_FCN SHOULD be 6 (or 0b110, which allows a maximum window size of 7 fragments).

When fragmentation is used to transport IP frames, the Message Integrity Check (MIC) size, M: TBD bits

The algorithm for computing the MIC field MUST be TBD.

5.3. Downlink fragment transmissions

In some LPWAN technologies, as part of energy-saving techniques, downlink transmission is only possible immediately after an uplink transmission. This allows the device to go in a very deep sleep mode and preserve battery, without the need to listen to any information from the network. This is the case for Sigfox-enabled devices, which can only listen to downlink communications after performing an uplink transmission and requesting a downlink.

When there are fragments to be transmitted in the downlink, an uplink message is required to trigger the downlink communication. In order to avoid potentially high delay for fragmented datagram transmission in the downlink, the fragment receiver MAY perform an uplink transmission as soon as possible after reception of a downlink fragment that is not the last one. Such uplink transmission MAY be triggered by sending a SCHC message, such as a SCHC ACK. However, other data messages can equally be used to trigger DL communications.

For reliable downlink fragment transmission, the ACK-Always mode is RECOMMENDED.

The recommended Fragmentation Header size is: 8 bits

The recommended Rule ID size is: 2 bits.

The recommended DTag size (T) is: 2 bits.

Fragment Compressed Number (FCN) size (N): 3 bits.

As per [I-D.ietf-lpwan-ipv6-static-context-hc], in the ACK-Always mode a Window (W) 1-bit field must be present.

For the ACK-Always fragmentation mode(s), a single window size is RECOMMENDED.

The value of MAX_ACK_REQUESTS SHOULD be 2, and the value of MAX_WIND_FCN SHOULD be 6 (or 0b110, which allows a maximum window size of 7 fragments).

When fragmentation is used to transport IP frames, the Message Integrity Check (MIC) size, M: TBD bits

The algorithm for computing the MIC field MUST be TBD.

Sigfox downlink frames have a fixed length of 8 bytes, which means that default SCHC algorithm for padding cannot be used. Therefore, the 3 last bits of the fragmentation header are used to indicate in bytes the size of the padding. A size of 000 means that the full ramaining frame is used to carry payload, a value of 001 indicates that the last byte contains padding, and so on.

6. Padding

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

Uplink frames can contain a payload size from 0 to 96 bits, that is 0 to 12 bytes. The radio protocol allows sending zero bits or one single bit of information for binary applications (e.g. status), or an integer number of bytes. Therefore, for 2 or more bits of payload it is required to add padding to the next integer number of bytes. The reason for this flexibility is to optimize transmission time and hence save battery consumption at the device.

Downlink frames on the other hand have a fixed length. The payload length must be 64 bits (i.e. 8 bytes). Hence, if less information bits are to be transmitted, padding would be necessary and it should be performed as described in the previous section.

7. Security considerations

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

Application data can be encrypted at the application level or not, depending on the criticality of the use case. This flexibility allows providing a balance between cost and effort vs. risk. AES-128 in counter mode is used for encryption. Cryptographic keys are independent for each device. These keys are associated with the device ID and separate integrity and confidentiality keys are pre-provisioned. A confidentiality key is only provisioned if confidentiality is to be used.

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

8. Acknowledgements

Carles Gomez has been funded in part by the ERDF and the Spanish Government through project TEC2016-79988-P.

9. Informative References

[I-D.ietf-lpwan-ipv6-static-context-hc]
Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "LPWAN Static Context Header Compression (SCHC) and fragmentation for IPv6 and UDP", draft-ietf-lpwan-ipv6-static-context-hc-17 (work in progress), October 2018.

[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>.

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

Authors' Addresses

Juan Carlos Zuniga
SIGFOX
425 rue Jean Rostand
Labege 31670
France

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

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

Email: carlesgo@entel.upc.edu

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

Email: Laurent.Toutain@imt-atlantique.fr