Abstract

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 in 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.

Two scenarios can be investigated:

- OAM coming from internet. In that case, the NGW should act as a proxy and handle specifically the OAM traffic.
- OAM coming from LPWAN devices: This can be included into regular devices but some specific devices may be installed in the LPWAN network to measure its quality.
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 describes 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.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 1, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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


[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 [I-D.ietf-lpwan-ipv6-static-context-hc].

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 (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 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 Section 4.2.1 what SCHC compression should be applied.

- the Device is the originator of an Echo Request message, and therefore the destination of the Echo Reply message.

- the Device is the destination of an Echo Request message, and therefore the purported source of an Echo Reply message.

These cases are further described in Section 4.

4. Detailed behavior

4.1. Device is the source of an ICMPv6 error message

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

The general intent of this document is to spare both the Device and the LPWAN network this un-necessary traffic. The incorrect packets should be caught at the core SCHC C/D and the ICMPv6 notification should be sent back from there.
Figure 1: Example of ICMPv6 error message sent back to the Internet

Figure 1 shows an example of an IPv6 packet trying to reach a Device. Let’s assume that the port number used as destination port is not "known" (needs better definition) from the core SCHC C/D. 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.

TODO: This assumes that all ports that the Device listens to will be matched by a SCHC rule. Is this the basic assumption of SCHC that all packets that do not match a rule are rejected? If yes, why do have fragmentation also for uncompressed packets?

TODO: discuss the various Type/Code that are expected to be generated in response to various errors.

4.2. 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 an intermediate router. 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.
Figure 2: Example of ICMPv6 error message sent back to the Device

Figure 2 illustrates this behavior. The ICMPv6 error message is compressed as described in Section 4.2.1 and forwarded over the LPWAN to the Device.

4.2.1. ICMPv6 error message compression.

The ICMPv6 error messages defined in [RFC4443] contain the fields shown in Figure 3.

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Value</td>
</tr>
<tr>
<td>As much of invoking packet as possible without the ICMPv6 packet exceeding the minimum IPv6 MTU</td>
</tr>
</tbody>
</table>

Figure 3: 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 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.

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.

[RFC4443] states that the "ICMPv6 error message MUST include as much of the IPv6 offending (invoking) packet ... as possible". In order to comply with this requirement, if there is enough information in the incoming ICMPv6 message for the core SCHC C/D to identify the rule that has been used to decompress the erroneous IPv6 packet, this Rule Id must be sent in the compressed ICMPv6 message to the Device. TODO: the erroneous IPv6 packet header (not just the Rule Id) should be sent back. This includes the Rule Id and the compression residue. This means the SCHC C/D uses the context backwards (in the reverse direction). How does the Device know it must also use the context backwards?

TODO: how does one know that the "payload" of a compressed-header packet is in fact another compressed header?

4.3. Device does a ping

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 4, with Type=128 and Code=0.
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".

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.

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 4).

[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.

TODO: how about a shared rule for Echo Request and Echo Reply with an LSB(1) CDA on the Type field? Or exploiting the Up/Down direction field in the rule?
4.4. Device is ping’ed

If the Device is ping’ed (i.e., is the destination of an Echo Request message), the default behavior is to avoid propagating the Echo Request message over the LPWAN.

This is the recommended behavior with the Code 0 (default value) of the Echo Request message.

The resulting behavior is shown on Figure 5 and described below:

```
Device       NGW     core SCHC C/D                 Internet Host
|           |            |    Echo Request, Code=0    |
|           |            |<---------------------------|
|           |            |--------------------------->|
|           |            |    Echo Reply,   Code=0    |
```

Figure 5: Examples of ICMPv6 Echo Request/Reply

- Code = 0: The Echo Request message is not propagated on the LPWAN to the Device. If the SCHC C/D finds a rule in the context with the IPv6 address of the Device, it responds with an Echo Reply on behalf of the Device. If no rule is found with that IPv6 address, the SCHC C/D does not respond.

TODO: again, we are assuming that no compression rule is equivalent to the device not providing the service.

5. Traceroute

The traceroute6 program sends successive probe packets destined to a chosen target but with the Hop Limit value successively incremented from the initial value 1.

It expects to receive a "Time Exceeded" (Type = 3) "Hop Limit" (Code = 0) ICMPv6 error message back from the successive routers along the path to the destination.

The probe packet is usually a UDP datagram, but can also be a TCP datagram or even an ICMPv6 message. The destination port is chosen in the unassigned range in hope that the destination, when eventually reached, will respond with a "Destination Unreachable" (Type = 1) "Port Unreachable" (Code = 4) ICMPv6 error message.
It is not anticipated that a Device will want to traceroute a destination on the Internet.

By contrast, a host on the Internet may attempt to traceroute an IPv6 address that is assigned to an LPWAN device. This is described in Figure 6.

```
<table>
<thead>
<tr>
<th>Device</th>
<th>NGW</th>
<th>core SCHC C/D</th>
<th>Internet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hop Limit=1, Dest Port=XXX</td>
<td>&lt;---------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICMPv6 Hop Limit error</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hop Limit=2, Dest Port=XXX</td>
<td>&lt;---------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICMPv6 Port Unreachable</td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 6: Example of traceroute to the LPWAN Device

When the probe packet first reaches the core SCHC C/D, its remaining Hop Limit is 1. The core SCHC C/D will respond back with a "Time Exceeded" (Type = 3) "Hop Limit" (Code = 0) ICMPv6 error message. Later on, when the probe packet reaches the core SCHC C/D with a Hop Limit value of 2, the core SCHC C/D will, as explained in Section 4.1, answer back with a "Destination Unreachable" (Type = 1) "Port Unreachable" (Code = 4) ICMPv6 error message. This is what the traceroute6 command expects. Therefore, the traceroute6 command will work with LPWAN IPv6 destinations, except for the time displayed for the destination, which is actually the time to its proxy.

However, if the probe packet happens to hit a port that matches a SCHC rule for that Device, the packet will be compressed with this rule and sent over the LPWAN, which is unfortunate. Forwarding of packets to the Device over the LPWAN should only be done from authenticated/trusted sources anyway. Rate-limitation on top of authentication will mitigate this nuisance.
6. Security considerations

TODO

7. IANA Considerations

TODO

8. References

8.1. Normative References

[I-D.ietf-lpwan-ipv6-static-context-hc]


8.2. Informative References


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
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

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

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

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

This Internet-Draft will expire on January 5, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents

Gomez & Crowcroft Expires January 5, 2020 [Page 1]
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 in LPWAN. Both the Uplink RTT (U-RTT) and the Downlink RTT (D-RTT) are considered. The former refers to an RTT where the first message in the RTT is sent in the uplink (and the response is sent in the downlink), whereas the latter refers to the opposite. First, the document characterizes the U-RTT for LoRaWAN and Sigfox in absence of communication errors, buffering delays or processing delays. Second, higher order U-RTTs are described, capturing the impact of message rate limitations due to regulatory constraints and radio gateway buffering delays. Third, D-RTT is analyzed for both LoRaWAN and Sigfox. 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 U-RTT

This section provides an analysis of the U-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 U-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 U-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 U-RTT, in seconds
RTTmax: maximum U-RTT, in seconds

Figure 1: Minimum and maximum U-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 U-RTT value for DR0 will always (for DR1, will almost always) exceed the default CoAP RTO. The maximum U-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 U-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.

<table>
<thead>
<tr>
<th>UL BR</th>
<th>Ulpld</th>
<th>TtxUL</th>
<th>TtxDL</th>
<th>RTTmin</th>
<th>RTTmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>12</td>
<td>2.08</td>
<td>0.39</td>
<td>21.8</td>
<td>47.1</td>
</tr>
<tr>
<td>600</td>
<td>12</td>
<td>0.35</td>
<td>0.39</td>
<td>20.6</td>
<td>45.4</td>
</tr>
</tbody>
</table>

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 U-RTT, in seconds
RTTmax: maximum U-RTT, in seconds

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

4. Higher order U-RTT

The high U-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 U-RTTs that will always exceed the default CoAP RTO, leading to a U-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 U-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. D-RTT analysis

D-RTTs may be greater than U-RTTs, due to the feature in some LPWANs that a downlink message can only be sent as a response to an uplink message (as an energy conservation technique for LPWAN devices). A downlink message may need to be buffered at the gateway until the opportunity for downlink transmission occurs. Therefore, a D-RTT comprises two main components: i) the wait time since a message is ready for downlink transmission until the next uplink transmission is complete, denoted $T_{\text{wait}}$; ii) the time since the uplink transmission is complete, until the D-RTT can be completed, called Basic D-RTT (BD-RTT).

$T_{\text{wait}}$ can be characterized as a random variable, which depends on the time between two consecutive uplink messages, and has a distribution that depends on the specific application in use. The message rates at which applications using LPWAN technologies operate may be in the order of seconds, minutes, hours, etc. In some cases, it is possible to program scheduled uplink transmissions, which allow minimizing $T_{\text{wait}}$, ideally down to zero.
Figure 3 and Figure 4 provide the values for BD-RTT for LoRaWAN (in the EU) and Sigfox, respectively. We assume the same packet sizes considered in the ideal scenario U-RTT study. In LoRaWAN, the BD-RTT does not contain the 1- or 2-second delay between the uplink transmission and the downlink response, therefore BD-RTT is smaller than the ideal scenario U-RTT. In Sigfox, the 1.4-second delay between a downlink transmission and its subsequent uplink confirmation is now added, compared to the ideal scenario U-RTT.

<table>
<thead>
<tr>
<th></th>
<th>BD-RTT</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3.52</td>
<td>3.81</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1.99</td>
<td>2.15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.92</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.73</td>
<td>0.82</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.66</td>
<td>0.78</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.37</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.19</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0.01</td>
<td>0.05</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Minimum and maximum BD-RTT values (in seconds) for LoRaWAN in the EU, without losses, and in absence of buffering delay and processing delay.
6. 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 (e.g. see Section 3) are expected in addition to the ideal scenario ones (e.g. see 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 5):

- Initially, the LPWAN device uses the High RTO.
o When the device uses the High RTO, after \( N_{\text{THRESH_LOW}} \) consecutive RTT samples lower than \( \text{THRESH_LOW}_{\text{RTT}} \), the device switches to using the Low RTO.

o When the device uses the Low RTO, after \( N_{\text{THRESH_HIGH}} \) consecutive RTT samples greater than \( \text{THRESH_HIGH}_{\text{RTT}} \), the device switches back to using the High RTO.

```
+----------+
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
+----------+

if \( N_{\text{THRESH_HIGH}} \) consecutive RTT samples > \( \text{THRESH_HIGH}_{\text{RTT}} \)

if \( N_{\text{THRESH_LOW}} \) consecutive RTT samples < \( \text{THRESH_LOW}_{\text{RTT}} \)

+----------+
|          |
|          |
|          |
|          |
|          |
|          |
|          |
|          |
+----------+

Figure 5: 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.

7. Security Considerations

TBD

8. 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.

9. References

9.1. Normative References


9.2. Informative References


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
This draft defines the way SCHC header compression can be applied to CoAP headers. The CoAP header structure differs from IPv6 and UDP protocols since CoAP uses a flexible header with a variable number of options, themselves of variable length. The CoAP protocol is asymmetric in its message format: the format of the packet header in the request messages is different from that in the response messages. Most of the compression mechanisms have been introduced in [I-D.ietf-lpwan-ipv6-static-context-hc], this document explains how to use the SCHC compression for CoAP.
Table of Contents

1. Introduction .................................................. 3
2. SCHC Compression Process .................................... 3
3. CoAP Compression with SCHC .................................... 4
4. Compression of CoAP header fields ............................ 6
   4.1. CoAP version field ........................................... 6
   4.2. CoAP type field ............................................. 6
   4.3. CoAP code field ............................................. 6
   4.4. CoAP Message ID field ...................................... 6
   4.5. CoAP Token fields .......................................... 7
5. CoAP options .................................................... 7
   5.1. CoAP Content and Accept options. .......................... 7
   5.2. CoAP option Max-Age, Uri-Host and Uri-Port fields ..... 7
   5.3. CoAP option Uri-Path and Uri-Query fields ............... 8
       5.3.1. Variable length Uri-Path and Uri-Query .......... 8
       5.3.2. Variable number of path or query elements ....... 9
   5.4. CoAP option Size1, Size2, Proxy-URI and Proxy-Scheme fields ....................................................... 9
   5.5. CoAP option ETag, If-Match, If-None-Match, Location-Path and Location-Query fields .................................... 9
6. Other RFCs ......................................................... 9
   6.1. Block ....................................................... 9
   6.2. Observe ..................................................... 10
   6.3. No-Response ............................................... 10
   6.4. OSCORE .................................................... 10
7. Examples of CoAP header compression .......................... 11
   7.1. Mandatory header with CON message ...................... 11
   7.2. OSCORE Compression ........................................ 12
   7.3. Example OSCORE Compression ................................ 16
8. IANA Considerations ............................................. 26
9. Security considerations ........................................ 26
10. Acknowledgements ............................................... 26
11. Normative References ........................................ 26
Authors’ Addresses ............................................... 27
1. Introduction

CoAP [rfc7252] is an implementation of the REST architecture for constrained devices. Although CoAP was designed for constrained devices, the size of a CoAP header may still be too large for the constraints of Low Power Wide Area Networks (LPWAN) and some compression may be needed to reduce the header size.

[I-D.ietf-lpwan-ipv6-static-context-hc] defines a header compression mechanism for LPWAN network based on a static context. The context is said static since the field description composing the Rules are not learned during the packet exchanges but are previously defined. The context(s) is(are) known by both ends before transmission.

A context is composed of a set of rules that are referenced by Rule IDs (identifiers). A rule contains an ordered list of the fields descriptions containing a field ID (FID), its length (FL) and its position (FP), a direction indicator (DI) (upstream, downstream and bidirectional) and some associated Target Values (TV). Target Value indicates the value that can be expected. TV can also be a list of values. A Matching Operator (MO) is associated to each header field description. The rule is selected if all the MOs fit the TVs for all fields of the incoming packet. In that case, a Compression/Decompression Action (CDA) associated to each field defines how the compressed and the decompressed values are computed out of each other, for each of the header fields. Compression mainly results in one of 4 actions: send the field value, send nothing, send some least significant bits of the field or send an index. Values sent are called Compression Residues and follow the rule ID in the transmitted message.

The compression rules define a generic way to compress and decompress the fields. If the device is modified, for example, to introduce new functionalities or new CoAP options, the rules must be updated to reflect the evolution. There is no risk to lock a device in a particular version of CoAP.

2. SCHC Compression Process

The SCHC Compression rules can be applied to CoAP flows. SCHC Compression of the CoAP header MAY be done in conjunction with the lower layers (IPv6/UDP) or independently. The SCHC adaptation layers as described in [I-D.ietf-lpwan-ipv6-static-context-hc] may be used as shown in Figure 1.
Figure 1: rule scope for CoAP

Figure 1 shows some examples for CoAP architecture and the SCHC rule’s scope.

In the first example, a rule compresses all headers from IPv6 to CoAP. In this case, SCHC C/D is performed at the device and at the LPWAN boundary.

In the second example, an end-to-end encryption mechanisms is used between the device and the application. CoAP is compressed independently of the other layers. The rule ID and the compression residue are encrypted using a mechanism such as DTLS. Only the other end can decipher the information. Layers below may also be compressed using other SCHC rules (this is out of the scope of this document).

In the third example, OSCORE [I-D.ietf-core-object-security] is used. 2 rulesets are used to compress the CoAP message. A first ruleset focuses on the inner header and is end to end, a second ruleset compresses the outer header and the layers below. SCHC C/D for inner header is done by both ends, and SCHC C/D for outer header and other headers is done between the device and the LPWAN boundary.

3. CoAP Compression with SCHC

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

- IPv6 and UDP are symmetrical protocols. The same fields are found in the request and in the response, with the value of some fields being swapped on the return path (e.g. source and destination fields). A CoAP request is intrinsically different from a response. For example, the URI-path option is mandatory in the request and is not found in the response, a request may contain an Accept option and the response a Content option.
[I-D.ietf-lpwan-ipv6-static-context-hc] defines the use of a message direction (DI) in the Field Description, which allows a single Rule to process message headers differently depending on the direction.

- Even when a field is "symmetric" (i.e. found in both directions) the values carried in each direction are different. Combined with a matching list in the TV, this allows reducing the range of expected values in a particular direction and therefore reduce the size of the compression residue. For instance, if a client sends only CON request, the type can be elided by compression and the answer may use one single bit to carry either the ACK or RST type. The same behavior can be applied to the CoAP Code field (0.0X code are present in the request and Y.ZZ in the answer). The direction allows splitting in two parts the possible values for each direction.

- In IPv6 and UDP, header fields have a fixed size. In CoAP, Token size may vary from 0 to 8 bytes, the length being given by a field in the header. More systematically, the CoAP options are described using the Type-Length-Value.

[I-D.ietf-lpwan-ipv6-static-context-hc] offers the possibility to define a function for the Field Length in the Field Description.

- In CoAP headers, a field can appear several times. This is typical for elements of a URI (path or queries). [I-D.ietf-lpwan-ipv6-static-context-hc] allows a Field ID to appears several times in the rule, the Field Position (FP) identifies the proper instance, thereby removing the ambiguity of the matching operation.

- Field sizes defined in the CoAP protocol can be too large regarding LPWAN traffic constraints. This is particularly true for the message ID field or Token field. The MSB MO can be used to reduce the information carried on LPWANs.

- CoAP also obeys the client/server paradigm and the compression ratio can be different if the request is issued from an LPWAN device or from a non LPWAN device. For instance a Device (Dev) aware of LPWAN constraints can generate a 1 byte token, but a regular CoAP client will certainly send a larger token to the Dev. The SCHC compression-decompression process does not modify the values. Nevertheless, a proxy placed before the compressor may change some field values to allow SCHC achieving a better compression ratio, while maintaining the necessary context for interoperability with existing CoAP implementations.
4. Compression of CoAP header fields

This section discusses the compression of the different CoAP header fields.

4.1. CoAP version field

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

4.2. CoAP type field

[rfc7252] defines 4 types of messages: CON, NON, ACK and RST. The last two are a response to the first two. If the device plays a specific client or server role, a rule can exploit these properties with the mapping list: [CON, NON] for one direction and [ACK, RST] for the other direction. The compression residue is reduced to 1 bit.

The field SHOULD be elided if for instance a client is sending only NON or only CON messages.

In any case, a rule MUST be defined to carry RST to a client.

4.3. CoAP code field

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

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

All the response codes MUST be compressed with a SCHC rule.

4.4. CoAP Message ID field

This field is bidirectional and is used to manage acknowledgments. The server memorizes the value for an EXCHANGE_LIFETIME period (by default 247 seconds) for CON messages and a NON_LIFETIME period (by default 145 seconds) for NON messages. During that period, a server receiving the same Message ID value will process the message as a retransmission. After this period, it will be processed as a new message.
In case the Device is a client, the size of the message ID field may be too large regarding the number of messages sent. The client SHOULD use only small message ID values, for instance 4 bit long. Therefore, a MSB can be used to limit the size of the compression residue.

In case the Device is a server, the client may be located outside of the LPWAN area and view the Device as a regular device connected to the internet. The client will generate Message ID using the 16 bits space offered by this field. A CoAP proxy can be set before the SCHC C/D to reduce the value of the Message ID, to allow its compression with the MSB matching operator and LSB CDA.

4.5. CoAP Token fields

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

Token Length is processed as any protocol field. If the value remains the same during all the transaction, the size can be stored in the context and elided during the transmission. Otherwise, it will have to the sent as a compression residue.

Token Value size cannot be defined directly in the rule in the Field Length (FL). Instead, a specific function designated as "TKL" MUST be used and length does not have to the sent with the residue. During the decompression, this function returns the value contained in the Token Length field.

5. CoAP options

5.1. CoAP Content and Accept options.

These fields are both unidirectional and MUST NOT be set to bidirectional in a rule entry.

If a single value is expected by the client, it can be stored in the TV and elided during the transmission. Otherwise, if several possible values are expected by the client, a matching-list SHOULD be used to limit the size of the residue. If is not possible, the value has to be sent as a residue (fixed or variable length).

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

These fields are unidirectional and MUST NOT be set to bidirectional in a rule entry. They are used only by the server to inform of the caching duration and is never found in client requests.
If the duration is known by both ends, the value can be elided on the LPWAN.

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

Otherwise these options SHOULD be sent as a residue (fixed or variable length).

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

These fields are unidirectional and MUST NOT be set to bidirectional in a rule entry. They are used only by the client to access a specific resource and are never found in server responses.

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

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

```
FID   FL   FP   DI    TV         MO        CDA
URI-Path 1  up  ["/a/b", equal  not-sent
               "/c/d"]
URI-Path 3  up  ignore value-sent
```

Figure 2: complex path example

In Figure 2 a single bit residue can be used to code one of the 2 paths. If regrouping were not allowed, a 2 bits residue would be needed.

5.3.1. Variable length Uri-Path and Uri-Query

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

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

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

For instance for a CORECONF path /c/X6?k="eth0" the rule can be set to:
Figure 3 shows the parsing and the compression of the URI, where c is not sent. The second element is sent with the length (i.e. 0x2 X 6) followed by the query option (i.e. 0x05 "eth0").

5.3.2. Variable number of path or query elements

The number of Uri-path or Uri-Query elements in a rule is fixed at the rule creation time. If the number varies, several rules SHOULD be created to cover all the possibilities. Another possibility is to define the length of Uri-Path to variable and send a compression residue with a length of 0 to indicate that this Uri-Path is empty. This adds 4 bits to the compression residue.

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

These fields are unidirectional and MUST NOT be set to bidirectional in a rule entry. They are used only by the client to access a specific resource and are never found in server response.

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

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

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

These fields are unidirectional.

These fields values cannot be stored in a rule entry. They MUST always be sent with the compression residues.

6. Other RFCs

6.1. Block

Block [rfc7959] allows a fragmentation at the CoAP level. SCHC also includes a fragmentation protocol. They are compatible. If a block option is used, its content MUST be sent as a compression residue.
6.2. Observe

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

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

6.3. No-Response

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

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

6.4. OSCORE

OSCORE [I-D.ietf-core-object-security] 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_kidctxt --------> |<-- CoAP OSCORE_kid -->|
```

Figure 4: OSCORE Option
The encoding of the OSCORE Option Value defined in Section 6.1 of [I-D.ietf-core-object-security] is repeated in Figure 4.

The first byte is used for flags that specify the contents of the OSCORE option. The 3 most significant bits 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 3 least significant bits n indicate the length of the piv field in bytes. When n = 0, no piv is present.

After the flag byte follow the piv field, kid context field and kid field in order and if present; the length of the kid context field is encoded in the first byte denoting by s the length of the kid context in bytes.

This draft recommends to implement a parser that is able 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. It is thus recommended that the parser split the OSCORE option into the 4 subsequent fields:

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

These fields are shown superimposed on the OSCORE Option format in Figure 4, the CoAP OSCORE_kidctxt field including the size bits s. Their size SHOULD be reduced using SCHC compression.

7. Examples of CoAP header compression

7.1. Mandatory header with CON message

In this first scenario, the LPWAN compressor at the Network Gateway side receives from a client on the Internet a POST message, which is immediately acknowledged by the Device. For this simple scenario, the rules are described Figure 5.
The version and Token Length fields are elided. The 26 method and response codes defined in [rfc7252] has been shrunk to 5 bits using a matching list. Uri-Path contains a single element indicated in the matching operator.

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

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

7.2. OSCORE Compression

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

Conceptually this is achieved by splitting the CoAP message into an Inner Plaintext and Outer OSCORE Message. The Inner Plaintext contains sensible information which is not necessary for proxy operation. This, in turn, is the part of the message which can be encrypted until it reaches its end destination. The Outer Message acts as a shell matching the format of a regular CoAP message, and
includes all Options and information needed for proxy operation and caching. This decomposition is illustrated in Figure 6.

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

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

Additionally, the OSCORE Option is added as an Outer option, signaling that the message is OSCORE protected. This option carries the information necessary to retrieve the Security Context with which the message was encrypted so that it may be correctly decrypted at the other end-point.
Figure 6: A CoAP message is split into an OSCORE outer and plaintext.

Figure 6 shows the message format for the OSCORE Message and Plaintext.

In the Outer Header, the original message code is hidden and replaced by a default dummy value. As seen in sections 4.1.3.5 and 4.2 of [I-D.ietf-core-object-security], the message code is replaced by POST for requests and Changed for responses when Observe is not used. If Observe is used, the message code is replaced by FETCH for requests and Content for responses.

The original message code is put into the first byte of the Plaintext. Following the message code, the class E options comes and
if present the original message Payload is preceded by its payload marker.

The Plaintext is now encrypted by an AEAD algorithm which integrity protects Security Context parameters and eventually any class I options from the Outer Header. Currently no CoAP options are marked class I. The resulting Ciphertext becomes the new Payload of the OSCORE message, as illustrated in Figure 7.

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

```
Outer Header
+---+---+---+-------------+---+---------------------------+
| v | t | tkl | new code | Msg Id. |
+---+---+---+-------------+---+---------------------------+
| Token |
+---------------------------+...+
| Options (IU) |
.                          .
. OSCORE Option .
+---------------------------+
| 0xFF |
+---------------------------+

Ciphertext: Encrypted Inner Header and Payload
+ Authentication Tag

Figure 7: OSCORE message
```

The SCHC Compression scheme consists of compressing both the Plaintext before encryption and the resulting OSCORE message after encryption, see Figure 8.

This translates into a segmented process where SCHC compression is applied independently in 2 stages, each with its corresponding set of rules, with the Inner SCHC Rules and the Outer SCHC Rules. This way compression is applied to all fields of the original CoAP message.
Note that since the Inner part of the message can only be decrypted by the corresponding end-point, this end-point will also have to implement Inner SCHC Compression/Decompression.

Figure 8: OSCORE Compression Diagram

7.3. Example OSCORE Compression

An example is given with a GET Request and its consequent CONTENT Response from a device-based CoAP client to a cloud-based CoAP server. A possible set of rules for the Inner and Outer SCHC Compression is shown. A dump of the results and a contrast between SCHC + OSCORE performance with SCHC + COAP performance is also listed. This gives an approximation to the cost of security with SCHC-OSCORE.
Our first example CoAP message is the GET Request in Figure 9.

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

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

0x0001 = mid
0x82 = token

Options:
0xbb74656d7065726174757265
Option 11: URL_PATH
Value = temperature

Original msg length: 17 bytes.

Figure 9: CoAP GET Request

Its corresponding response is the CONTENT Response in Figure 10.

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 10: CoAP CONTENT Response
The SCHC Rules for the Inner Compression include all fields that are already present in a regular CoAP message, what is important is the order of appearance and inclusion of only those CoAP fields that go into the Plaintext, Figure 11.

Rule ID 0

<table>
<thead>
<tr>
<th>Field</th>
<th>FP</th>
<th>DI</th>
<th>Target Value</th>
<th>MO</th>
<th>CDA</th>
<th>Sent [bits]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoAP Code</td>
<td>up</td>
<td>1</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CoAP Code</td>
<td>dw</td>
<td>[69,132]</td>
<td>match-map</td>
<td>match-sent</td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>CoAP Uri-Path</td>
<td>up</td>
<td>temperature</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COAP Option-End</td>
<td>dw</td>
<td>0xFF</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: Inner SCHC Rules

Figure 12 shows the Plaintext obtained for our example GET Request and follows the process of Inner Compression and Encryption until we end up with the Payload to be added in the outer OSCORE Message.

In this case the original message has no payload and its resulting Plaintext can be compressed up to only 1 byte (size of the Rule ID). The AEAD algorithm preserves this length in its first output, but also yields a fixed-size tag which cannot be compressed and has to be included in the OSCORE message. This translates into an overhead in total message length, which limits the amount of compression that can be achieved and plays into the cost of adding security to the exchange.
OSCORE Plaintext
0x01bb74656d7065726174757265 (13 bytes)

0x01 Request Code GET
bb74656d7065726174757265 Option 11: URI_PATH
Value = temperature

Figure 12: Plaintext compression and encryption for GET Request

In Figure 13 we repeat the process for the example CONTENT Response. In this case the misalignment produced by the compression residue (1 bit) makes it so that 7 bits of padding have to be applied after the payload, resulting in a compressed Plaintext that is the same size as before compression. This misalignment also causes the hexcode from the payload to differ from the original, even though it has not been
compressed. On top of this, the overhead from the tag bytes is incurred as before.

```
OSCORE Plaintext
0x45ff32332043 (6 bytes)
0x45 Successful Response Code 69 "2.05 Content"
    ff Payload marker
    32332043 Payload

<table>
<thead>
<tr>
<th>Inner SCHC Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Plaintext</td>
</tr>
<tr>
<td>0x001919902180 (6 bytes)</td>
</tr>
<tr>
<td>00 Rule ID</td>
</tr>
<tr>
<td>0b0 (1 bit match-map residue)</td>
</tr>
<tr>
<td>0x32332043 &gt;&gt; 1 (shifted payload)</td>
</tr>
<tr>
<td>0b0000000 Padding</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AEAD Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>(piv = 0x04)</td>
</tr>
</tbody>
</table>

encrypted_plaintext = 0x10c6d7c26cc1 (6 bytes)
tag = 0xe9aef3f2461e0c29 (8 bytes)
ciphertext = 0x10c6d7c26cc1e9aef3f2461e0c29 (14 bytes)
```

Figure 13: Plaintext compression and encryption for CONTENT Response
The Outer SCHC Rules (Figure 16) MUST process the OSCORE Options fields. In Figure 14 and Figure 15 we show a dump of the OSCORE Messages generated from our example messages once they have been provided with the Inner Compressed Ciphertext in the payload. These are the messages that are to go through Outer SCHC Compression.

Protected message:
==================
0x4102000182d7080904636c69656e74ffa2c54fe1b434297b62
(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 14: Protected and Inner SCHC Compressed GET Request
Protected message:

0x6144000182d008ff10c6d7c26cc1e9aeff3f2461e0c29
(22 bytes)

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

0x0001 = mid
0x82 = token

Options:
0x0008 (2 bytes)
Option 21: OBJECT_SECURITY
Value = b''

0xFF Payload marker
Payload:
0x10c6d7c26cc1e9aeff3f2461e0c29 (14 bytes)

Figure 15: Protected and Inner SCHC Compressed CONTENT Response

For the flag bits, a number of compression methods could prove to be useful depending on the application. The simplest alternative is to provide a fixed value for the flags, combining MO equal and CDA not-sent. This saves most bits but could hinder flexibility. Otherwise, match-mapping could allow to choose from a number of configurations of interest to the exchange. If neither of these alternatives is desirable, MSB could be used to mask off the 3 hard-coded most significant bits.

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

The piv field lends itself to having a number of bits masked off with MO MSB and CDA LSB. This could prove useful in applications where the message frequency is low such as that found in LPWAN technologies. Note that compressing the sequence numbers effectively reduces the maximum amount of sequence numbers that can be used in an exchange. Once this amount is exceeded, the SCHC Context would need to be re-established.
The size $s$ included in the kid context field MAY be masked off with CDA MSB. The rest of the field could have additional bits masked off, or have the whole field be fixed with MO equal and CDA not-sent. The same holds for the kid field.

Figure 16 shows a possible set of Outer Rules to compress the Outer Header.

<table>
<thead>
<tr>
<th>Rule ID 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>CoAP version</td>
</tr>
<tr>
<td>CoAP Type</td>
</tr>
<tr>
<td>CoAP Type</td>
</tr>
<tr>
<td>CoAP TKL</td>
</tr>
<tr>
<td>CoAP Code</td>
</tr>
<tr>
<td>CoAP Code</td>
</tr>
<tr>
<td>CoAP MID</td>
</tr>
<tr>
<td>CoAP Token</td>
</tr>
<tr>
<td>CoAP OSCORE_flags</td>
</tr>
<tr>
<td>CoAP OSCORE_piv</td>
</tr>
<tr>
<td>COAP OSCORE_kid</td>
</tr>
<tr>
<td>COAP OSCORE_kidctxt</td>
</tr>
<tr>
<td>CoAP OSCORE_flags</td>
</tr>
<tr>
<td>CoAP OSCORE_piv</td>
</tr>
<tr>
<td>CoAP OSCORE_kid</td>
</tr>
<tr>
<td>CoAP Option-End</td>
</tr>
</tbody>
</table>

Figure 16: Outer SCHC Rules

These Outer Rules are applied to the example GET Request and CONTENT Response. The resulting messages are shown in Figure 17 and Figure 18.
Compressed message:
===================
0x001489458a9fc3686852f6c4 (12 bytes)
0x00 Rule ID
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 17: SCHC-OSCORE Compressed GET Request

Compressed message:
===================
0x0014218daf84d983d35de7e48c3c1852 (16 bytes)
0x00 Rule ID
  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 18: SCHC-OSCORE Compressed CONTENT Response

For contrast, we compare these results with what would be obtained by SCHC compressing the original CoAP messages without protecting them with OSCORE. To do this, we compress the CoAP messages according to the SCHC rules in Figure 19.
Rule ID 1

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

Figure 19: SCHC-CoAP Rules (No OSCORE)

This yields the results in Figure 20 for the Request, and Figure 21 for the Response.

Compressed message:

0x0114

0x01 = Rule ID

Compression residue:

0b00010100 (1 byte)

Compressed msg length: 2

Figure 20: CoAP GET Compressed without OSCORE
Compressed message:
=====
0x010a32332043
0x01 = Rule ID

Compression residue:
0b00001010 (1 byte)

Payload
0x32332043

Compressed msg length: 6

Figure 21: CoAP CONTENT Compressed without OSCORE

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

8. IANA Considerations

This document has no request to IANA.

9. Security considerations

This document does not have any more Security consideration than the ones already raised on [I-D.ietf-lpwan-ipv6-static-context-hc]

10. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Thomas Fossati, Klaus Hartke, Francesca Palombini, Alexander Pelov, Goran Selander.

11. Normative References

[I-D.ietf-core-object-security]


Authors' Addresses

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

Email: ana@ackl.io
Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6
draft-ietf-lpwan-ipv6-static-context-hc-21

Abstract

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

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

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

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

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

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

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

This Internet-Draft will expire on January 24, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ........................................... 4
2. Requirements Notation .................................. 5
3. LPWAN Architecture ....................................... 5
4. Terminology ............................................. 6
5. SCHC overview ........................................... 8
  5.1. SCHC Packet format ................................ 10
  5.2. Functional mapping .................................. 10
6. Rule ID ................................................ 11
7. Compression/Decompression ................................. 12
  7.1. SCHC C/D Rules ..................................... 12
  7.2. Rule ID for SCHC C/D ............................... 14
  7.3. Packet processing .................................. 15
  7.4. Matching operators ................................. 17
  7.5. Compression Decompression Actions (CDA) .......... 17
7.5.1. processing fixed-length fields ........................................ 18
7.5.2. processing variable-length fields ................................... 18
7.5.3. not-sent CDA ............................................................... 19
7.5.4. value-sent CDA ............................................................. 19
7.5.5. mapping-sent CDA .......................................................... 19
7.5.6. LSB CDA ................................................................. 20
7.5.7. DevIID, AppIID CDA ...................................................... 20
7.5.8. Compute-* ................................................................. 20
8. Fragmentation/Reassembly ......................................................... 21
  8.1. Overview ................................................................. 21
  8.2. SCHC F/R Protocol Elements ............................................ 21
     8.2.1. Messages ............................................................. 21
     8.2.2. Tiles, Windows, Bitmaps, Timers, Counters ....................... 22
     8.2.3. Integrity Checking .................................................. 24
     8.2.4. Header Fields ........................................................ 25
  8.3. SCHC F/R Message Formats ................................................ 27
     8.3.1. SCHC Fragment format ............................................... 27
     8.3.2. SCHC ACK format .................................................... 28
     8.3.3. SCHC ACK REQ format ............................................... 31
     8.3.4. SCHC Sender-Abort format .......................................... 31
     8.3.5. SCHC Receiver-Abort format ....................................... 32
  8.4. SCHC F/R modes ............................................................ 33
     8.4.1. No-ACK mode ......................................................... 33
     8.4.2. ACK-Always mode ................................................... 35
     8.4.3. ACK-on-Error mode ................................................ 41
9. Padding management ............................................................... 48
10. SCHC Compression for IPv6 and UDP headers ................................... 49
  10.1. IPv6 version field ......................................................... 49
  10.2. IPv6 Traffic class field ............................................... 50
  10.3. Flow label field ......................................................... 50
  10.4. Payload Length field .................................................... 50
  10.5. Next Header field ........................................................ 51
  10.6. Hop Limit field .......................................................... 51
  10.7. IPv6 addresses fields ................................................... 51
     10.7.1. IPv6 source and destination prefixes ............................. 51
     10.7.2. IPv6 source and destination IID .................................. 52
  10.8. IPv6 extensions .......................................................... 52
  10.9. UDP source and destination port ...................................... 52
  10.10. UDP length field ........................................................ 53
  10.11. UDP Checksum field ..................................................... 53
11. IANA Considerations ............................................................ 54
12. Security considerations ....................................................... 54
  12.1. Security considerations for SCHC Compression/Decompression ......... 54
  12.2. Security considerations for SCHC Fragmentation/Reassembly .......... 55
13. Acknowledgements ............................................................... 56
14. References ................................................................. 56
1. Introduction

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

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

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

o 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).

o 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 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 and product-specific choices 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 technologies have similar network architectures but different terminologies. Using the terminology defined in [RFC8376], we can identify different types of entities in a typical LPWAN network, see Figure 1:

- 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.

- The Radio Gateway (RGW), which is the end point of the constrained link.

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

- Application Server (App)
4. Terminology

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

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

- **App**: LPWAN Application, as defined by [RFC8376]. An application sending/receiving packets to/from the Dev.
- **AppIID**: Application Interface Identifier. The IID that identifies the application server interface.
- **Bi**: Bidirectional. Characterizes a Field Descriptor that applies to headers of packets traveling in either direction (Up and Dw, see this glossary).
- **CDA**: Compression/Decompression Action. Describes the pair of inverse actions that are performed at the compressor to compress a header field and at the decompressor to recover the original value of the header field.
- **Compression Residue**: The bits that remain to be sent (beyond the Rule ID itself) after applying the SCHC compression.
- **Context**: A set of Rules used to compress/decompress headers.
- **Dev**: Device, as defined by [RFC8376].
- **DevIID**: Device Interface Identifier. The IID that identifies the Dev interface.
- **DI**: Direction Indicator. This field tells which direction of packet travel (Up, Dw or Bi) a Field Description applies to. This allows for asymmetric processing, using the same Rule.

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

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

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

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

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

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

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

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

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

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

- **Profile**: SCHC offers variations in the way it is operated, with a number of parameters listed in Appendix D. A Profile indicates a
particular setting of all these parameters. Both ends of a SCHC communication must be provisioned with the same Profile information and with the same set of Rules before the communication starts, so that there is no ambiguity in how they expect to communicate.

- **Rule**: A set of Field Descriptions.
- **Rule ID** (Rule Identifier): An identifier for a Rule. SCHC C/D on both sides share the same Rule ID for a given packet. A set of Rule IDs are used to support SCHC F/R functionality.
- **SCHC C/D**: SCHC Compressor/Decompressor. A mechanism used on both sides, at the Dev and at the network, to achieve Compression/Decompression of headers.
- **SCHC F/R**: SCHC Fragmentation / Reassembly. A mechanism used on both sides, at the Dev and at the network, to achieve Fragmentation / Reassembly of SCHC Packets.
- **SCHC Packet**: A packet (e.g. an IPv6 packet) whose header has been compressed as per the header compression mechanism defined in this document. If the header compression process is unable to actually compress the packet header, the packet with the uncompressed header is still called a SCHC Packet (in this case, a Rule ID is used to indicate that the packet header has not been compressed). See Section 7 for more details.
- **TV**: Target value. A value contained in a Rule that will be matched with the value of a header field.
- **Up**: Uplink direction for compression/decompression, from the Dev SCHC C/D to the network SCHC C/D.

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

5. **SCHC overview**

SCHC can be characterized as an adaptation layer between IPv6 and the underlying LPWAN technology. SCHC comprises two sublayers (i.e. the Compression sublayer and the Fragmentation sublayer), as shown in Figure 2.
Before a packet (e.g. an IPv6 packet) is transmitted, header compression is first applied. The resulting packet is called a SCHC Packet, whether or not any compression is performed. If the SCHC Packet is to be fragmented, 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)

\[
\begin{align*}
\text{Sender} & \quad \text{IPv6} \quad \text{Compression} \quad \text{SCHC} < \quad \text{Fragmentation} \quad \text{LPWAN technology} \\
\text{v} & \quad \text{If no fragmentation (*)} \\
\text{^} & \quad \text{SCHC Packet} \quad \text{----------------->} \\
\text{v} & \quad \text{SCHC Fragmentation} \\
\text{^} & \quad \text{SCHC Reassembly} \\
\text{^} & \quad \text{SCHC ACK} \\
\text{v} & \quad \text{SCHC Fragments} \\
\end{align*}
\]

\begin{itemize}
\item Sender
\item Receiver
\end{itemize}

*: the decision to use Fragmentation or not is left to each Profile.
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 below 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, i.e. on the Dev side and on the Network 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 side can be located in the NGW, or somewhere else 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 reverting 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 a Rule ID is the link between the SCHC Compressor and the SCHC Decompressor, or between the SCHC Fragmenter and the SCHC Reassembler.

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

The Rule IDs are used:

- For SCHC C/D, to identify the Rule (i.e., the set of Field Descriptions) that is used to compress a packet header.
  - At least one Rule ID MUST be allocated to tagging packets for which SCHC compression was not possible (no matching Rule was found).

- In SCHC F/R, to identify the specific mode and settings of F/R for one direction of traffic (Up or Dw).
  - When F/R is used for both communication directions, at least two Rule ID values are needed for F/R, one per direction of traffic.
7. Compression/Decompression

Compression with SCHC is based on using a set of Rules, called the Context, to compress or decompress headers. SCHC avoids Context synchronization traffic, which consumes considerable bandwidth in other header compression mechanisms such as RoHC [RFC5795]. Since the content of packets is highly predictable in LPWAN networks, static Contexts may be stored beforehand. The Contexts MUST be stored at both ends, and they can be learned by a provisioning protocol or by out of band means, or they can be pre-provisioned. The way the Contexts are provisioned is out of the scope of this document.

7.1. SCHC C/D Rules

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

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

Each Rule itself contains a list of Field Descriptions composed of a Field Identifier (FID), a Field Length (FL), a Field Position (FP), a Direction Indicator (DI), a Target Value (TV), a Matching Operator (MO) and a Compression/Decompression Action (CDA).
A Rule does not describe how the compressor parses a packet header to find and identify each field (e.g. the IPv6 Source Address, the UDP Destination Port or a CoAP URI path option). It is assumed that there is a protocol parser alongside SCHC that is able to identify all the fields encountered in the headers to be compressed, and to label them with a Field ID. Rules only describe the compression/decompression behavior for each header field, after it has been identified.

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

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

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

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

- A Direction Indicator (DI) indicates the packet direction(s) this Field Description applies to. Three values are possible:
  - UPLINK (Up): this Field Description is only applicable to packets sent by the Dev to the App,
  - DOWNLINK (Dw): this Field Description is only applicable to packets sent from the App to the Dev,
  - BIDIRECTIONAL (Bi): this Field Description is applicable to packets traveling both Up and Dw.

- Target Value (TV) is the value used to match against the packet header field. The Target Value can be a scalar value of any type (integer, strings, etc.) or a more complex structure (array, list, etc.). The types and representations are out of scope for this document.

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

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

7.2. Rule ID for SCHC C/D

Rule IDs are sent by the compression function in one side and are received for the decompression function in the other side. In SCHC C/D, the Rule IDs are specific to the Context related to one Dev. Hence, multiple Dev instances, which refer to different header compression Contexts, MAY reuse the same Rule ID for different Rules. On the network 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 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 steps:

- Compression Rule selection: the set of Rules is browsed to identify which Rule will be used to compress the packet header. The Rule is selected by matching the Fields Descriptions to the packet header. The detailed steps are 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 used for compressing the header. Otherwise, the Rule MUST be disregarded.
* If no eligible compression Rule is found, then the header MUST be sent in its entirety using the Rule ID of the "default" Rule dedicated to this purpose. Sending an uncompressed header may require SCHC F/R.

- Compression: each field of the header is compressed according to the Compression/Decompression Actions (CDAs). 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.

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

Figure 7: Compression Residue structure

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

- Decompression: when decompressing, on the network side the SCHC C/D needs to find the correct Rule based on the L2 address; 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.

The receiver identifies the sender through its device-id or source identifier (e.g. MAC address, if it exists) and selects the Rule using the Rule ID. This Rule describes the compressed header format and associates the received residues to each of the header fields. For each field in the header, the receiver applies the CDA action associated to that field in order to reconstruct the original header field value. The CDA application order can be different from the order in which the fields are listed in the Rule. In particular, Compute-* MUST be applied after the application of the CDAs of all the fields it computes on.
7.4. Matching operators

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

- **equal**: The match result is True if the field value in the packet matches the TV.
- **ignore**: No matching is attempted between the field value in the packet and the TV in the Rule. The result is always true.
- **MSB(x)**: A match is obtained if the most significant 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 length must be a multiple of the unit. For example, x must be multiple of 8 if the unit of the variable length is in bytes.
- **match-mapping**: With match-mapping, the Target Value is a list of values. Each value of the list is identified by an index. Compression is achieved by sending the index instead of the original header field value. This operator matches if the header field value is equal to one of the values in the target list.

7.5. Compression Decompression Actions (CDA)

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

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

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 ---|
+-------+-------------+
|  size  |    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 first (left of the residue bit 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:

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

- Larger sizes are encoded as 0xfff followed by the 16 bits unsigned integer.

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

7.5.3. not-sent CDA

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

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

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

7.5.4. value-sent CDA

The value-sent action can be used when the field value is not known by both the Compressor and the Decompressor. The value is sent in its entirety.

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 first (left of the residue bit 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.

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 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. If it is not needed, its description can be safely ignored.

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 SCHC Packet. This document does not specify which mode(s) are to be used over a specific LPWAN technology. That information will be given in Profiles.

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

8.2. SCHC F/R Protocol Elements

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

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

8.2.1. Messages

SCHC F/R defines the following messages:

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

- SCHC ACK REQ: A request by the sender for a SCHC ACK from the receiver.
- SCHC Sender-Abort: A message by the sender telling the receiver that it has aborted the transmission of a fragmented SCHC Packet.
- SCHC Receiver-Abort: A message by the receiver to tell the sender to abort the transmission of a fragmented SCHC Packet.

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.

```
+----+--+-----+---+----+-+---+---+-----+...-----+----+---+------+
Tiles |    |  |     |   |    | |   |   |     |        |    |   |      |
+----+--+-----+---+----+-+---+---+-----+...-----+----+---+------+
```

Figure 10: a SCHC Packet fragmented in tiles

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

8.2.2.2. Windows

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

If windows are used

- all the windows of a SCHC Packet, except the last one, MUST contain the same number of tiles. This number is WINDOW_SIZE.
- WINDOW_SIZE MUST be specified in a Profile.
- the windows are numbered.
o their numbers MUST increase 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 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.

<table>
<thead>
<tr>
<th>Tile #</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window #</td>
<td>-------- 0 --------</td>
<td>-------- 1 --------</td>
<td>2 ... 27</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: a SCHC Packet fragmented in tiles grouped in 28 windows, with WINDOW_SIZE = 5

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
8.2.2.4. Timers and counters

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

- **Inactivity Timer**: A SCHC Fragment receiver uses this timer to abort waiting for a SCHC F/R message.
- **Retransmission Timer**: A SCHC Fragment sender uses this timer to abort waiting for an expected SCHC ACK.
- **Attempts**: This counter counts the requests for SCHC ACKs, up to `MAX_ACK_REQUESTS`.

8.2.3. Integrity Checking

The integrity of the fragmentation-reassembly process of a SCHC Packet MUST be checked at the receive end. By default, integrity checking is performed by computing a Reassembly Check Sequence (RCS) of the SCHC Packet at the sender side before fragmentation 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 reverse representation of the polynomial used e.g. in the Ethernet standard [RFC3385]) is RECOMMENDED as the default algorithm for computing the RCS. Nevertheless, other RCS lengths or other algorithms MAY be required by the Profile.

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

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

8.2.4. Header Fields

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

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

The Rule ID allows SCHC F/R interleaving non-fragmented SCHC Packets and SCHC Fragments that carry other SCHC Packets, or interleaving SCHC Fragments that use different SCHC F/R modes or different parameters.

- **Datagram Tag (DTag)**. Its size (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 only one fragmented SCHC Packet in transit for a given 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 the transmission of which may overlap.
A sequence counter that is incremented for each new fragmented SCHC Packet, counting from 0 to up to \((2^T)-1\) and wrapping back to 0 is RECOMMENDED for maximum traceability and avoidance of ambiguity.

A flow of SCHC F/R messages with a given Rule ID and DTag value pair MUST NOT interfere with the operation of a SCHC F/R instance that uses another Rule ID and DTag value pair.

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

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

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

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

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

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

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

  See Section 8.2.3 for the RCS default size, default polynomial and details on RCS computation.
- C (integrity Check): C is a 1-bit field. This field is used in the SCHC ACK message to report on the reassembled SCHC Packet integrity check (see Section 8.2.3).

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

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

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

8.3. SCHC F/R Message Formats

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

8.3.1. SCHC Fragment format

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

```
+-----------------+-----------------------+˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜
| 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.

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

8.3.1.2. All-1 SCHC Fragment

The All-1 SCHC Fragment format is shown in Figure 14. The sender generally 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. 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

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

8.3.2. SCHC ACK format

The SCHC ACK message is shown in Figure 15. The DTag field, the W field and the Compressed Bitmap field are optional. The Compressed Bitmap field can only be present in SCHC F/R modes that use windows.
The SCHC ACK Header contains a C bit (see Section 8.2.4). If the C bit is set to 1 (integrity check successful), no Bitmap is carried. If the C bit is set to 0 (integrity check not performed or failed) and if windows are used, a Compressed Bitmap for the window referred to by the W field is transmitted as specified in Section 8.3.2.1.

8.3.2.1. Bitmap Compression

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

- Build a temporary SCHC ACK message that contains the Header followed by the original Bitmap (see Section 8.2.2.3 for a description of Bitmaps).
- Position scissors at the end of the Bitmap, after its last bit.
- While the bit on the left of the scissors is 1 and belongs to the Bitmap, keep moving left, then stop. When this is done,
- While the scissors are not on an L2 Word boundary of the SCHC ACK message and there is a Bitmap bit on the right of the scissors, keep moving right, then stop.
- At this point, cut and drop off any bits to the right of the scissors

When one or more bits have effectively been dropped off as a result of the above algorithm, the SCHC ACK message is a multiple of L2 Words, no padding bits will be appended.
Because the SCHC Fragment sender knows the size of the original Bitmap, it can reconstruct the original Bitmap from the Compressed Bitmap received in the SCH ACK message.

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

|---- 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 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 |
| Rule ID | DTag | W |C=0|1 0 1 0 1 1 1|

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 #)
```

```
next L2 Word boundary ->|
```

```
| Rule ID | DTag | W |C=0|1 1 1 1 1 1 1| with uncompressed Bitmap
```

```
|--- ... -+- ... -+---+- ... -+˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜˜
```

```
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. 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. 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,

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

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

The SCHC Sender-Abort MUST NOT be acknowledged.

8.3.5. SCHC Receiver-Abort format

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

The SCHC Receiver-Abort format is shown in Figure 22. The DTag field and the W field are optional.

|--- 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,

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

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

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

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

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

Such a bit pattern never occurs in a regular SCHC ACK. This is how the fragment sender recognizes a SCHC Receiver-Abort.
The SCHC Receiver-Abort MUST NOT be acknowledged.

8.4. SCHC F/R modes

This specification includes several SCHC F/R modes, which
- allow for a range of reliability options, such as optional SCHC Fragment retransmission
- support various LPWAN characteristics, including variable MTU.

More modes may be defined in the future.

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 acknowledgement.

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

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

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

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

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

Each Profile, for each Rule ID value, MUST define
- the size of the DTag field,
- the size and algorithm for the RCS field,
- the expiration time of the Inactivity Timer
Each Profile, for each Rule ID value, MAY define
  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.

8.4.1.1. Sender behavior

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

Each SCHC Fragment MUST contain exactly one tile in its Payload. The
tile MUST be at least the size of an L2 Word. The sender MUST
transmit the SCHC Fragments messages in the order that the tiles
appear in the SCHC Packet. Except for the last tile of a SCHC
Packet, each tile MUST be of a size that complements the SCHC
Fragment Header so that the SCHC Fragment is a multiple of L2 Words
without the need for padding bits. Except for the last one, the SCHC
Fragments MUST use the Regular SCHC Fragment format specified in
Section 8.3.1.1. The last SCHC Fragment MUST use the All-1 format
specified in Section 8.3.1.2.

The sender MAY transmit a SCHC Sender-Abort.

Figure 37 shows an example of a corresponding state machine.

8.4.1.2. Receiver behavior

Upon receiving each Regular SCHC Fragment,
  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
if integrity checking fails, the receiver MUST drop the reassembled SCHC Packet

the reassembly operation concludes.

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

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

Figure 38 shows an example of a corresponding state machine.

8.4.2. ACK-Always mode

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

- Data unit out-of-sequence delivery does not occur between the entity performing fragmentation and the entity performing reassembly.
- The L2 MTU value does not change while the fragments of a SCHC Packet are being transmitted.

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 \( \text{WINDOW\_SIZE} \), 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 \( \text{MAX\_ACK\_REQUESTS} \),
- the expiration time of the Retransmission Timer
- the expiration time of the Inactivity Timer

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

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer.

8.4.2.1. Sender behavior

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

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

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

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

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

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

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

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 wait a SCHC ACK. Then,

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

- on Retransmission Timer expiration,
if Attempts is strictly less than MAX_ACK_REQUESTS, the
fragment sender MUST send a SCHC ACK REQ and MUST increment the
Attempts counter.

otherwise the fragment sender MUST send a SCHC Sender-Abort,
and it MAY exit with an error condition.

At any time,

- on receiving a SCHC Receiver-Abort, the fragment sender MAY exit
  with an error condition.
- on receiving a SCHC ACK that bears a W value different from the W
  value that it currently uses, the fragment sender MUST silently
discard and ignore that SCHC ACK.

Figure 39 shows an example of a corresponding state machine.

8.4.2.2. Receiver behavior

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

- the receiver SHOULD check if the DTag value has not recently been
  used for that Rule ID value, thereby ensuring that the received
  SCHC Fragment is not a remnant of a prior fragmented SCHC Packet
  transmission. If the SCHC Fragment is determined to be such a
  remnant, the receiver MAY silently ignore it and discard it.
- the receiver MUST start a process to assemble a new SCHC Packet
  with that Rule ID and DTag value pair.
- the receiver MUST start an Inactivity Timer. It MUST initialize
  an Attempts counter to 0. It MUST initialize a window counter to
  0.

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, the receiver MUST reset the
Inactivity Timer.

Entering an "acceptance phase", the receiver MUST first initialize an
empty Bitmap for this window, then

- on receiving a SCHC Fragment or SCHC ACK REQ with the W bit
different from the local W bit, the receiver MUST silently ignore
and discard that message.
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, and the receiver MUST send a SCHC ACK for this window. Then,

+ If the Bitmap indicates that all the tiles of the current window have been correctly received, the receiver MUST increment its window counter and it enters the "acceptance phase" for that new window.

+ If the Bitmap indicates that at least one tile is missing in the current window, the receiver enters 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".

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

o on receiving a SCHC ACK REQ with the W bit equal to the local W bit, the receiver has not yet determined if the current window is a not-last one or the last one, the receiver MUST send a SCHC ACK for this window, and it keeps accepting incoming messages.

In the "retransmission phase":

o if the window is a not-last window

* on receiving a SCHC Fragment or SCHC ACK REQ with 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 a 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-1 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.

* on the Bitmap becoming fully populated with 1’s, the receiver MUST send a SCHC ACK for this window, it MUST increment its window counter and it enters the "acceptance phase" for the new window.

- if the window is the last window

* on receiving a SCHC Fragment or SCHC ACK REQ with 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 a 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. Then

- 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,

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

  - it keeps accepting incoming messages.
In the "clean-up phase":

- Any received SCHC F/R message with a W bit different from the local W bit MUST be silently ignored and discarded.
- Any received SCHC F/R message different from an All-1 SCHC Fragment or a SCHC ACK REQ MUST be silently ignored and discarded.
- On receiving an All-1 SCHC Fragment or a SCHC ACK REQ, the receiver MUST send a SCHC ACK.

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.

In ACK-on-Error mode, windows are used. All tiles MUST be of equal size, except for the last one, which MUST be of the same size or smaller than the regular ones. If allowed in a Profile, the penultimate tile MAY be exactly one L2 Word smaller than the regular tile size.

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

See Figure 23 for an example.

Figure 23: a SCHC Packet fragmented in tiles, Ack-on-Error mode
The W field is wide enough that it unambiguously represents an absolute window number. The fragment receiver sends SCHC ACKs to the fragment sender about windows for which tiles are missing. No SCHC ACK is sent by the fragment receiver for windows that it knows have been fully received.

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

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

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

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

Each Profile, for each Rule ID value, MUST define

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

- if the penultimate tile MAY be one L2 Word smaller than the regular tile size. In this case, the regular tile size MUST be at least twice the L2 Word size.

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

- one Attempts counter
- one Retransmission Timer

For each active pair of Rule ID and DTag values, the receiver MUST maintain an Inactivity Timer.

### 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 \((2^M) \times \text{WINDOW}_\text{SIZE}\) 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 consecutive 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
o in a Regular SCHC Fragment, alone or as part of a multi-tiles Payload

o 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.

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

+ otherwise,

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

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

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

- the receiver MUST start an Inactivity Timer. It MUST initialize an Attempts counter to 0.

On receiving any SCHC F/R message, the receiver MUST reset the Inactivity Timer.

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

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

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

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

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

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

    + the size of the tiles is known a priori,
On receiving a SCHC ACK REQ or an All-1 SCHC Fragment,
  o if the receiver has at least one window that it knows has tiles missing, it MUST return a SCHC ACK for the lowest-numbered such window,
  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 dropped)
+------------------+                      +--------------------+
| SCHC Compression |                      | SCHC Decompression |
+------------------+                      +--------------------+
                             +-----------------+
                             |       |
                             |       +------------- SCHC ACK ------------+
                             |       |
                             |       +-----------------+
------------ 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.

10.1. IPv6 version field

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

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

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

- One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".
- If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

10.3. Flow label field

If the Flow Label 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, two possibilities can be considered:

- One possibility is to not compress the field and send the original value. In the Rule, TV is not set to any particular value, MO is set to "ignore" and CDA is set to "value-sent".
- If some upper bits in the field are constant and known, a better option is to only send the LSBs. In the Rule, TV is set to a value with the stable known upper part, MO is set to MSB(x) and CDA to LSB.

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

10.4. Payload Length field

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

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

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

10.6. Hop Limit field

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

- in the Up, elide the field: the TV in the Field Descriptor is set to the known constant value, the MO is set to "equal" and the CDA is set to "not-sent".
- in the Dw, the Hop Limit is elided for transmission and forced to 1 at the receiver, by setting TV to 1, MO to "ignore" and CDA to "not-sent". This prevents any further forwarding.

10.7. IPv6 addresses fields

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

10.7.1. IPv6 source and destination prefixes

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

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

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

10.7.2. IPv6 source and destination IID

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

As described in [RFC8065], it may be undesirable to build the Dev IPv6 IID out of the Dev address. Another static value is used instead. In that case, the TV contains the static value, the MO operator is set to "equal" and the CDA is set to "not-sent". [RFC7217] provides some methods to derive this static identifier.

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 extensions

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

10.9. UDP source and destination port

To allow for a single Rule being used for both directions, the UDP port values are identified by their role (Dev or App) and not by their position in the header (source or destination). The SCHC C/D MUST be aware of the traffic direction (Uplink, Downlink) to select the appropriate field. The following Rules apply for Dev and App port numbers.
If both ends know the port number, it can be elided. The TV contains the port number, the MO is set to "equal" and the CDA is set to "not-sent".

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

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

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

10.10. UDP length field

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

10.11. UDP Checksum field

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

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

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

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

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

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

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

11. IANA Considerations

This document has no request to IANA.

12. Security considerations

Wireless networks are subjects to various sorts of attacks, which are not specific to SCHC. In this section, we’ll assume that an attacker was able to break into the network despite the latter’s security measures and that it can now send packets to a target node. What is specific to SCHC is the amplification of the effects that this break-in could allow. Our analysis equally applies to legitimate nodes "going crazy".

12.1. Security considerations for SCHC Compression/Decompression

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-decompress actions produce a bounded number of bits, irrespective of the incoming SCHC Packet. This property is true for IPv6 Length, UDP Length and UDP Checksum, for which the compute-* CDA is recommended by this document.
As a consequence, SCHC Decompression does not amplify attacks, beyond adding a bounded number of bits to the SCHC Packet received. This bound is determined by the Rule stored in the receiving device.

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

12.2. Security considerations for SCHC Fragmentation/Reassembly

Let’s assume that an attacker is able to send to 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 (down to 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.

In another type of attack, 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...
ssending 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.

13. Acknowledgements

Thanks to Sergio Aguilar Romero, Carsten Bormann, David Black, Philippe Clavier, Daniel Ducuara Beltran Diego Dujovne, Eduardo Ingles Sanchez, Arunprabhu Kandasamy, Suresh Krishnan, Rahul Jadhav, Sergio Lopez Bernal, Antony Markovski, Alexander Pelov, Charles Perkins, Edgar Ramos, Shoichi Sakane, and Pascal Thubert for useful design consideration and comments.

Carles Gomez has been funded in part by the Spanish Government (Ministerio de Educacion, Cultura y Deporte) through the Jose Castillejo grant CA15/00336, 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


14.2. Informative References
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.

![Protocol Stack Diagram]

In some LPWAN technologies, only the Devis 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

<table>
<thead>
<tr>
<th>Field</th>
<th>FL</th>
<th>FP</th>
<th>DI</th>
<th>Value</th>
<th>Match Opera.</th>
<th>Comp Decomp Action</th>
<th>Sent [bits]</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6 Version</td>
<td>4</td>
<td>1</td>
<td>Bi</td>
<td>6</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 DiffServ</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Flow Label</td>
<td>20</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Length</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td></td>
<td>ignore</td>
<td>compute--*</td>
<td></td>
</tr>
<tr>
<td>IPv6 Next Header</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>17</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Hop Limit</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>255</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 DevPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>FE80::/64</td>
<td>ignore</td>
<td>DevIID</td>
<td></td>
</tr>
<tr>
<td>IPv6 DevIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td></td>
<td>ignore</td>
<td>DevIID</td>
<td></td>
</tr>
<tr>
<td>IPv6 AppPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>FE80::/64</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 AppIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>::1</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Field</th>
<th>FL</th>
<th>FP</th>
<th>DI</th>
<th>Value</th>
<th>Match</th>
<th>Action</th>
<th>Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6 Version</td>
<td>4</td>
<td>1</td>
<td>Bi</td>
<td>6</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 DiffServ</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Flow Label</td>
<td>20</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Length</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>17</td>
<td>ignore</td>
<td>compute-*</td>
<td></td>
</tr>
<tr>
<td>IPv6 Hop Limit</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>255</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 DevPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>[alpha/64, match-</td>
<td>mapping-sent</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>IPv6 DevIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>ignore</td>
<td>DevIID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv6 AppPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>[beta/64, alpha/64,</td>
<td>mapping-sent</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IPv6 AppIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>::1000</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>UDP DevPort</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>5683</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>UDP AppPort</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>5683</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>UDP Length</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>ignore</td>
<td>compute-*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP checksum</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>ignore</td>
<td>compute-*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rule 2

<table>
<thead>
<tr>
<th>Field</th>
<th>FL</th>
<th>FP</th>
<th>DI</th>
<th>Value</th>
<th>Match</th>
<th>Action</th>
<th>Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv6 Version</td>
<td>4</td>
<td>1</td>
<td>Bi</td>
<td>6</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 DiffServ</td>
<td>8</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Flow Label</td>
<td>20</td>
<td>1</td>
<td>Bi</td>
<td>0</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Length</td>
<td>16</td>
<td>1</td>
<td>Bi</td>
<td>17</td>
<td>ignore</td>
<td>compute-*</td>
<td></td>
</tr>
<tr>
<td>IPv6 Hop Limit</td>
<td>8</td>
<td>1</td>
<td>Up</td>
<td>255</td>
<td>ignore</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 Hop Limit</td>
<td>8</td>
<td>1</td>
<td>Dw</td>
<td></td>
<td>ignore</td>
<td>value-sent</td>
<td>8</td>
</tr>
<tr>
<td>IPv6 DevPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>alpha/64</td>
<td>equal</td>
<td>DevIID</td>
<td></td>
</tr>
<tr>
<td>IPv6 DevIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>ignore</td>
<td>DevIID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPv6 AppPrefix</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>gamma/64</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
<tr>
<td>IPv6 AppIID</td>
<td>64</td>
<td>1</td>
<td>Bi</td>
<td>::1000</td>
<td>equal</td>
<td>not-sent</td>
<td></td>
</tr>
</tbody>
</table>
All the fields described in the three Rules depicted on Figure 26 are present in the IPv6 and UDP headers. The DevIID-DID value is found in the L2 header.

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

The third Rule compresses each port number to 4 bits.

Appendix B. Fragmentation Examples

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

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

Sender  
---FCN=0---
---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.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----W=0, FCN=6-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=5-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=4-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=3-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=2-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=1-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=0-----&gt;</td>
<td></td>
</tr>
<tr>
<td>(no ACK)</td>
<td></td>
</tr>
<tr>
<td>-----W=1, FCN=6-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=1, FCN=5-----&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=1, FCN=4-----&gt;</td>
<td></td>
</tr>
<tr>
<td>--W=1, FCN=7 + RCS--&gt;</td>
<td>Integrity check: success</td>
</tr>
<tr>
<td>&lt;-- ACK, W=1, C=1 ---</td>
<td>C=1</td>
</tr>
<tr>
<td>(End)</td>
<td></td>
</tr>
</tbody>
</table>

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

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

Figure 30 shows an example of a transmission in ACK-on-Error mode of a SCHC Packet fragmented in 73 tiles, with $N=5$, $\text{WINDOW\_SIZE}=28$, $M=2$ and 3 lost SCHC Fragments.
<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----W=0, FCN=27-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=0, FCN=23-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=0, FCN=19-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=0, FCN=15--X--&gt;</td>
<td>4 tiles sent (not received)</td>
</tr>
<tr>
<td>-----W=0, FCN=11-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=0, FCN=7 -----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=0, FCN=3 -----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=27-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=23-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=19-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=15-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=11-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=7 -----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=1, FCN=3--X--&gt;</td>
<td>4 tiles sent (not received)</td>
</tr>
<tr>
<td>-----W=2, FCN=27-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>-----W=2, FCN=23-----&gt;</td>
<td>4 tiles sent</td>
</tr>
<tr>
<td>^ -----W=2, FCN=19-----&gt;</td>
<td>1 tile sent</td>
</tr>
<tr>
<td></td>
<td>-----W=2, FCN=18-----&gt;</td>
</tr>
<tr>
<td></td>
<td>-----W=2, FCN=17-----&gt;</td>
</tr>
<tr>
<td></td>
<td>-----W=2, FCN=16----&gt;</td>
</tr>
<tr>
<td>s</td>
<td>-----W=2, FCN=15----&gt;</td>
</tr>
<tr>
<td>m</td>
<td>-----W=2, FCN=14----&gt;</td>
</tr>
<tr>
<td>a</td>
<td>-----W=2, FCN=13--X--&gt;</td>
</tr>
<tr>
<td>l</td>
<td>-----W=2, FCN=12----&gt;</td>
</tr>
<tr>
<td>e &lt;---- ACK, W=0, C=0 ---</td>
<td>C=0, Bitmap:1111111111110000111111111111</td>
</tr>
<tr>
<td>r</td>
<td>-----W=0, FCN=15----&gt;</td>
</tr>
<tr>
<td>r</td>
<td>-----W=0, FCN=14----&gt;</td>
</tr>
<tr>
<td>L</td>
<td>-----W=0, FCN=13----&gt;</td>
</tr>
<tr>
<td>2</td>
<td>-----W=0, FCN=12----&gt;</td>
</tr>
<tr>
<td>L &lt;---- ACK, W=1, C=0 ---</td>
<td>C=0, Bitmap:1111111111111111111111111000</td>
</tr>
<tr>
<td>M</td>
<td>-----W=1, FCN=3-----&gt;</td>
</tr>
<tr>
<td>T</td>
<td>-----W=1, FCN=2----&gt;</td>
</tr>
<tr>
<td>U</td>
<td>-----W=1, FCN=1----&gt;</td>
</tr>
<tr>
<td>U</td>
<td>-----W=1, FCN=0-----&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;---- ACK, W=2, C=0 ---</td>
</tr>
<tr>
<td></td>
<td>-----W=2, FCN=13----&gt;</td>
</tr>
<tr>
<td>V</td>
<td>&lt;---- ACK, W=2, C=1 ---</td>
</tr>
</tbody>
</table>

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

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

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

Figure 34 illustrates the transmission in ACK-Always mode of a SCHC Packet fragmented in 6 tiles, with one tile per SCHC Fragment, N=3,
WINDOW_SIZE=7, three lost SCHC Fragments, and the second SCHC ACK lost.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----W=0, FCN=6-------&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=5-------&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=4--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=3--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=2--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>--W=0, FCN=7 + RCS--&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>&lt;- ACK, W=0, C=0 ---</td>
<td>C=0, Bitmap:1100001</td>
</tr>
<tr>
<td>-----W=0, FCN=4-------&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>-----W=0, FCN=3-------&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>-----W=0, FCN=2-------&gt;</td>
<td>Integrity check: success</td>
</tr>
<tr>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>--- W=0, ACK REQ -----&gt;</td>
<td>ACK REQ</td>
</tr>
<tr>
<td>&lt;- ACK, W=0, C=1 ---</td>
<td>C=1</td>
</tr>
</tbody>
</table>

(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.

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----W=0, FCN=6-------&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=5-------&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=4--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=3--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>-----W=0, FCN=2--X---&gt;</td>
<td></td>
</tr>
<tr>
<td>--W=0, FCN=7 + RCS--&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>&lt;- ACK, W=0, C=0 ---</td>
<td>C=0, Bitmap:1100001</td>
</tr>
<tr>
<td>-----W=0, FCN=4-------&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>-----W=0, FCN=3-------&gt;</td>
<td>Integrity check: failure</td>
</tr>
<tr>
<td>-----W=0, FCN=2-------&gt;</td>
<td></td>
</tr>
<tr>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>--- W=0, ACK REQ -----&gt;</td>
<td>ACK REQ</td>
</tr>
<tr>
<td>&lt;- ACK, W=0, C=1 ---</td>
<td>C=1</td>
</tr>
</tbody>
</table>

(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, \( N=5 \), \( \text{WINDOW\_SIZE}=24 \) and two lost SCHC Fragments.

![Diagram](image.png)

Figure 36: ACK-Always mode, 28 tiles, one tile per SCHC Fragment, lost SCHC Fragments.
Appendix C. Fragmentation State Machines

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

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

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>FCN=0</td>
</tr>
<tr>
<td>No Window</td>
<td>No Bitmap</td>
</tr>
<tr>
<td>More Fragments</td>
<td>Send Fragment (FCN=0)</td>
</tr>
<tr>
<td></td>
<td>last fragment</td>
</tr>
<tr>
<td></td>
<td>FCN = 1</td>
</tr>
<tr>
<td></td>
<td>send fragment + RCS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not All-1</td>
<td>RCV Frag</td>
</tr>
<tr>
<td></td>
<td>All-1 &amp; RCS correct</td>
</tr>
<tr>
<td></td>
<td>Inactivity Timer Exp.</td>
</tr>
<tr>
<td></td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>END</td>
</tr>
</tbody>
</table>

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

Internet-Draft                 LPWAN SCHC                      July 2019

Inactivity_Timer = expires
When DWL
IF Inactivity_Timer expires
Send DWL Request
Attempt++

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

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

This is an example only. It is not normative. The specification in Section 8.4.3.1 allows for sequences of operations different from the one shown here.
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 use cases, deployment scenarios
- Mapping of the SCHC architectural elements onto the LPWAN architecture
- Assessment of LPWAN integrity checking
- Various potential channel conditions for the technology and the corresponding recommended use of SCHC C/D and F/R

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

- Rule ID numbering scheme, fixed-sized or variable-sized Rule IDs, number of Rules, the way the Rule ID is transmitted
- Maximum packet size that should ever be reconstructed by SCHC Decompression (MAX_PACKET_SIZE). See Section 12.
Padding: size of the L2 Word (for most LPWAN technologies, this would be a byte; for some technologies, a bit)

Decision to use SCHC fragmentation mechanism or not. If yes:
* reliability mode(s) used, in which cases (e.g. based on link channel condition)
* Rule ID values assigned to each mode in use
* presence and number of bits for DTag (T) for each Rule ID value
* support for interleaved packet transmission, to what extent
* WINDOW_SIZE, for modes that use windows
* number of bits for W (M) for each Rule ID value, for modes that use windows
* number of bits for FCN (N) for each Rule ID value
* size of RCS and algorithm for its computation, for each Rule ID, if different from the default CRC32. Byte fill-up with zeroes or other mechanism, to be specified.
* Retransmission Timer duration for each Rule ID value, if applicable to the SCHC F/R mode
* Inactivity Timer duration for each Rule ID value, if applicable to the SCHC F/R mode
* MAX_ACK_REQUEST value for each Rule ID value, if applicable to the SCHC F/R mode

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

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

In some LPWAN technologies, as part of energy-saving techniques, downlink transmission is only possible immediately after an uplink transmission. In order to avoid potentially high delay in the downlink transmission of a fragmented SCHC Packet, the SCHC Fragment receiver may perform an uplink transmission as soon as
possible after reception of a SCHC Fragment that is not the last one. Such uplink transmission may be triggered by the L2 (e.g. an L2 ACK sent in response to a SCHC Fragment encapsulated in a L2 PDU that requires an L2 ACK) or it may be triggered from an upper layer.

- 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 may signal the window size in use for a specific packet transmission.

The same window size MUST be used for the transmission of all tiles that belong to the same SCHC Packet.

Appendix F. Downlink SCHC Fragment transmission

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 Inactivity Timer is used) 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 Inactivity 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 Inactivity timer for the SCHC ACK is stopped. However, if the Inactivity timer expires, the SCHC ACK is resent and the Inactivity timer is reinitialized and restarted.
The default initial value for the Inactivity Timer, as well as the maximum number of retries for a specific SCHC ACK, denoted MAX_ACK_RETRIES, are not defined in this document, and need to be defined in a Profile. The initial value of the Inactivity 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 can find 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 Inactivity Timer). If a SCHC ACK is received before expiration of this timer, the SCHC Fragment sender retransmits any lost SCHC Fragments 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

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. This is called a profile.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents
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 [RFC8376]. Even though those technologies share a great number of common features like star-oriented topologies, network architecture, devices with mostly quite predictable communications, etc; they do have some slight differences in respect of payload sizes, reactivity, 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].

- DevEUI: an IEEE EUI-64 identifier used to identify the end-device during the procedure while joining the network (Join Procedure)
- DevAddr: a 32-bit non-unique identifier assigned to a end-device statically or dynamically after a Join Procedure (depending on the activation mode)
- 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, 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.
Figure 1 represents the architecture for compression/decompression, it is based on [RFC8376] terminology. The Device is sending applications flows using IPv6 or IPv6/UDP protocols. These flow might be fragmented (SCHC F/R), and 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 F/R for defragmentation, if required, then C/D for decompression which shares the same rules with the device. The SCHC F/R and 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 F/R, then SCHC F/R and 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 F/R and 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 is an Application Server. It can be provided by the Network Server or any third party software. Figure 1 can be map in LoRaWAN terminology to:
4. LoRaWAN Architecture

An overview of LoRaWAN [lora-alliance-spec] protocol and architecture is described in [RFC8376]. 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:

- 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 (LoRaWAN gateway). This entity maps to the LoRaWAN End-Device.

- The Radio Gateway (RGW), which is the end point of the constrained link. This entity maps to the LoRaWAN Gateway.

- The Network Gateway (NGW) is the interconnection node between the Radio Gateway and the Internet. This entity maps to the LoRaWAN Network Server.

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

- Application Server (App). The same terminology is used in LoRaWAN. In that case, the application server will be the SCHC gateway, doing C/D and F/R.
SCHC C/D (Compressor/Decompressor) and SCHC F/R (Fragmentation/Reassembly) are performed on the LoRaWAN End-Device and the Application Server (called SCHC gateway). 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 (SCHC gateway) acts as the first hop IP router for the End-Device. The Application Server and Network Server may be co-located, which effectively turns the Network/Application Server into the first hop IP router.

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

The LoRaWAN MAC layer supports 3 classes of end-devices named A, B and C. All end-devices implement the classA, some end-devices implement classA+B or class A+C. ClassB and classC are mutually exclusive.

- **ClassA**: The classA is the simplest class of end-devices. The end-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 end-device to get a downlink opportunity. The classA is the lowest power end-device class.

- **ClassB**: classB end-devices implement all the functionalities of classA end-devices, but also schedule periodic listen windows. Therefore, as opposed the classA end-devices, classB end-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 end-device. The lower the downlink latency, the higher the power consumption.

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

4.2. End-Device addressing

LoRaWAN end-devices use a 32 bits 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 end-devices. End-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 end-devices use different security keys. To communicate with the SCHC gateway the Network Server MUST identify the end-devices by a unique 64bits device ID called the devEUI. Unlike devAddr, devEUI is guaranteed to be unique for every single end-device across all networks. The devEUI is assigned to the end-device during the manufacturing process by the end-device’s manufacturer. It is built like an Ethernet MAC address by concatenating the manufacturer’s IEEE OUI field with a vendor unique number. ex: 24bits OUI is concatenated with a 40 bits serial number. The Network Server translates the devAddr into a devEUI in the uplink direction and reciprocally on the downlink direction.

```
+--------+         +----------+        +---------+            +----------+
| End-   | <=====> | Network  | <====> | SCHC    | <========> | Internet |
| Device | devAddr | Server   | devEUI | Gateway |  IPv6/UDP  |          |
+--------+         +----------+        +---------+            +----------+
```

Figure 4: LoRaWAN addresses

4.3. General Message Types

- *Confirmed messages*: The sender asks the receiver to acknowledge the message.
- *Unconfirmed messages*: The sender does not ask the receiver to acknowledge the message.

As SCHC defines its own acknowledgment mechanisms, SCHC does not require to use confirmed messages.
4.4. LoRaWAN MAC Frames

- **JoinRequest**: This message is used by an end-device to join a network. It contains the end-device’s unique identifier devEUI and a random nonce that will be used for session key derivation.

- **JoinAccept**: To onboard an end-device, the Network Server responds to the JoinRequest end-device’s message with a JoinAccept message. That message is encrypted with the end-device’s AppKey and contains (amongst other fields) the major network’s settings and a network random nonce used to derive the session keys.

- **Data**

5. SCHC-over-LoRaWAN

5.1. LoRaWAN FPort

The LoRaWAN MAC layers feature a frame port field in all frames. This field (FPort) is 8-bit long and the values from 1 to 223 can be used. It allows LoRaWAN network and application to identify data.

A fragmentation session with application payload transferred from device to server, is called uplink fragmentation session. It uses FPortUpShort or FPortUpDefault for data uplink and its associated SCHC control downlinks. The other way, a fragmentation session with application payload transferred from server to device, is called downlink fragmentation session. It uses FPortDown for data downlink and its associated SCHC control uplinks.

FPorts can use arbitrary values inside the allowed FPort range and must be shared by the end-device, the Network Server and SCHC gateway. The uplink and downlink SCHC ports must be different. In order to improve interoperability, it is recommended to use:

- FPortUpShort = 20
- FPortUpDefault = 21
- FPortDown = 22

Those are recommended values and are application defined. Also application can have multiple fragmentation session between a device and one or several SCHC gateways. A set of three FPort values is required for each gateway instance the device is required to communicate with.
The only uplink messages using the FPortDown port are the fragmentation SCHC control messages of a downlink fragmentation session (ex ACKs). Similarly, the only downlink messages using the FPortUpShort or FPortUpDefault ports are the fragmentation SCHC control messages of an uplink fragmentation session.

5.2. Rule ID management

SCHC-over-LoRaWAN SHOULD support encoding RuleID on 6 bits (64 possible rules).

The RuleID 0 is reserved for fragmentation. The RuleID 63 is used to tag packets for which SCHC compression was not possible (no matching Rule was found).

The remaining RuleIDs are available for compression. RuleIDs are shared between uplink and downlink sessions. A RuleID different from 0 means that the fragmentation is not used, thus the packet should be send to C/D layer.

5.3. IID computation

As LoRaWAN network uses unique EUI-64 per end-device, the Interface IDentifier is the LoRaWAN DevEUI. It is compliant with [RFC4291] and IID starting with binary 000 must enforce the 64-bits rule. TODO: Derive IID from DevEUI with privacy constraints ? Ask working group ?

5.4. Fragmentation

The L2 word size used by LoRaWAN is 1 byte (8 bits). The SCHC fragmentation over LoRaWAN uses the ACK-on-Error for uplink fragmentation and Ack-Always for downlink fragmentation. A LoRaWAN end-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 end-device at a given moment.

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

5.5. DTag

A LoRaWAN device cannot interleave several fragmented SCHC datagrams. This one bit field is used to distinguish two consecutive fragmentation sessions.
5.5.1. Uplink fragmentation: From device to SCHC gateway

In that case the device is the fragmentation transmitter, and the SCHC gateway the fragmentation receiver. Two fragmentation rules are defined regarding the *FPort*:

- **FPortUpShort**: SCHC header is only one byte. Used when fragmentation is required and payload size is less than 381 bytes.
- **FPortUpDefault**: SCHC header is two bytes. Used for all other cases: no fragmentation required or payload size is between 382 and 1524 byte.

*Both rules share common parameters:*

- **SCHC fragmentation reliability mode**: "ACK-on-Error"
- **DTag**: size is 1 bit.
- **FCN**: The FCN field is encoded on $N = 7$ bits, so $\text{WINDOW\_SIZE} = 127$ tiles are allowed in a window (FCN=All-1 is reserved for SCHC).
- **MIC calculation algorithm**: CRC32 using 0xEDB88320 (i.e. the reverse representation of the polynomial used e.g. in the Ethernet standard [RFC3385]) as suggested in [I-D.ietf-lpwan-ipv6-static-context-hc].
- **MAX\_ACK\_REQUESTS**: 8
- **Tile**: size is 3 bytes (24 bits)
- **Retransmission and inactivity timers**: LoRaWAN end-devices do not implement a "retransmission timer". At the end of a window or a fragmentation session, corresponding ACK(s) is (are) transmitted by the network gateway (LoRaWAN application server) in the RX1 or RX2 receive slot of end-device. If this ACK is not received the end-device sends an all-0 (or an all-1) fragment with no payload to request an SCHC 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 SCHC gateway implements an "inactivity timer". The default recommended duration of this timer is 12 hours. This value is
mainly driven by application requirements and may be changed by
the application.

*The following fields are different:*

- RuleID size
- Window index size W

### 5.5.1.1. FPortUpShort - 1 byte header

In that case RuleID size is 0, the rule is the FPort=FPortUpShort and
only fragmented payload can be transported.

- *RuleID*: size is 0 bit in SCHC header, not used.
- *Window index*: encoded on W = 0 bit, not used

With this set of parameters, the SCHC fragment header overhead is 1
byte (8 bits). MTU is: _127 tiles * 3 bytes per tile = 381 bytes_

*Regular fragments*

```
| DTag | FCN     | Payload |
+-----+--------+---------+
| 1 bit| 7 bits |          |
```

Figure 5: All fragment except the last one. Header size is 8 bits (1
byte).

*SCHC ACK*

```
| DTag | C       | Encoded bitmap (if C = 0) | Padding (0s) |
+-----+--------+--------------------------+---------------+
| 1 bit| 1 bit  | 0 to 127 bits             | 7 or 0 bits   |
```

Figure 6: SCHC ACK format, failed mic check.

### 5.5.1.2. FPortUpDefault - 2 bytes header

- *RuleID*: size is 6 bits (64 possible rules, 62 available for
  compression)

- *Window index*: encoded on W = 2 bits. So 4 windows are
  available.
With this set of parameters, the SCHC fragment header overhead is 2 bytes (16 bits). MTU is: _4 windows * 127 tiles * 3 bytes per tile = 1524 bytes_

_Note_: Even if it is less efficient, this rule can also be used for fragmented payload size less than 382 bytes.

*Regular fragments*

<table>
<thead>
<tr>
<th>RuleID</th>
<th>DTag</th>
<th>W</th>
<th>FCN</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>2 bits</td>
<td>7 bits</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: All fragment except the last one. Header size is 16 bits (2 bytes).

*Last fragment (All-1)*

<table>
<thead>
<tr>
<th>RuleID</th>
<th>DTag</th>
<th>W</th>
<th>FCN=All-1</th>
<th>MIC</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>2 bits</td>
<td>7 bits</td>
<td>32 bits</td>
<td>Last tile, if any</td>
</tr>
</tbody>
</table>

Figure 8: All-1 fragment detailed format for the last fragment.

*SCHC ACK*

<table>
<thead>
<tr>
<th>RuleID</th>
<th>DTag</th>
<th>W</th>
<th>C</th>
<th>Encoded bitmap (if C = 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>2 bit</td>
<td>1 bit</td>
<td>0 to 127 bits</td>
</tr>
</tbody>
</table>

Figure 9: SCHC formats, failed MIC check.

*Receiver-Abort*

<table>
<thead>
<tr>
<th>RuleID</th>
<th>DTag</th>
<th>W = b’11</th>
<th>C = 1</th>
<th>b’111111</th>
<th>0xFF (all 1’s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>2 bits</td>
<td>1 bit</td>
<td>6 bits</td>
<td>8 bits</td>
</tr>
</tbody>
</table>

Figure 10: Receiver-Abort format.

*SCHC acknowledge request*
<table>
<thead>
<tr>
<th>RuleID</th>
<th>DTag</th>
<th>W</th>
<th>FCN = b’0000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>2 bits</td>
<td>7 bits</td>
</tr>
</tbody>
</table>

Figure 11: SCHC ACK REQ format.

5.5.2. Downlink fragmentation: From SCHC 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.

- **SCHC fragmentation reliability mode**: ACK-Always.
- **RuleID**: size is 6 bits (64 possible rules, 62 for compression).
- **Window index**: encoded on W=1 bit, as per [I-D.ietf-lpwan-ipv6-static-context-hc].
- **DTag**: Not used, so its size is 0 bit.
- **FCN**: The FCN field is encoded on N=1 bits, so WINDOW_SIZE = 1 tile (FCN=All-1 is reserved for SCHC).
- **MIC calculation algorithm**: CRC32 using 0xEDB88320 (i.e. the reverse representation of the polynomial used e.g. in the Ethernet standard [RFC3385]), as per [I-D.ietf-lpwan-ipv6-static-context-hc].
- **MAX_ACK_REQUESTS**: 8

As only 1 tile is used, its size can change for each downlink, and will be maximum available MTU minus header (1 byte)

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

*Regular fragments*
| RuleID | W     | FCN = b’0 | Payload |
+ ------ + ----- + --------- + ------- +
| 6 bits | 1 bit | 1 bits    | X bytes |

Figure 12: All fragments but the last one. Header size 1 byte (8 bits).

*Last fragment (All-1)*

| RuleID | W     | FCN = b’1 | MIC     | Payload            |
+ ------ + ----- + --------- + ------- + ----------------- +
| 6 bits | 1 bit | 1 bit     | 32 bits | Last tile, if any |

Figure 13: All-1 SCHC ACK detailed format for the last fragment.

*SCHC acknowledge*

| RuleID | W     | C = b’1 |
+ ------ + ----- + ------- +
| 6 bits | 1 bit | 1 bit   |

Figure 14: SCHC ACK format, MIC is correct.

*Receiver-Abort*

| RuleID | W    | C = b’0 | b’11111111 |
+ ------ + ----- + ------- + ---------- +
| 6 bits | 1 bit | 1 bits  | 8 bits     |

Figure 15: Receiver-Abort packet (following an all-1 packet with incorrect MIC).

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

5.5.2.1. ClassA end-devices

Class A end-devices can only receive in an RX slot following the transmission of an uplink. Therefore there cannot be a concept of "retransmission timer" for an SCHC gateway. The SCHC gateway cannot initiate communication to a classA end-device.

The device replies with an ACK message to every single fragment received from the SCHC 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, but the end of a window), the device MUST transmit the SCHC ACK fragment until it receives the fragment of the next window. The device shall transmit up to MAX_ACK_REQUESTS ACK messages before aborting. The device should transmit those ACK as soon as possible while taking into consideration potential 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=All-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 (C=1). This message might be lost therefore the SCHC 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, on SCHC FPortDown from the SCHC gateway different from an ACK-request: it indicates that the SCHC gateway has received the ACK message.

Following the reception of a FCN=All-1 fragment (the last fragment of a datagram), if all fragments have been received and 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, on SCHC FPortDown, from the SCHC gateway different from an ACK-request indicating that the SCHC gateway has received the Abort message. The fragmentation receiver (device) does not implement retransmission timer and inactivity timer.

The fragmentation sender (the SCHC gateway) implements an inactivity timer with default duration of 12 hours. Once a fragmentation session is started, if the SCHC gateway has not received any ACK or Receiver-Abort message 12 hours after the last message from the device was received, the SCHC 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.

5.5.2.2. Class B or C end-devices

Class B&C end-devices can receive in scheduled RX slots or in RX slots following the transmission of an uplink. The device replies with an ACK message to every single fragment received from the SCHC 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 SCHC ACK message even if that fragment has already been received. The ACK bitmap is 1 bit long and is always 1. If the SCHC gateway receives this ACK, it proceeds to send the next window fragment. If
the retransmission timer elapses and the SCHC gateway has not received the ACK of the current window it retransmits the last fragment. The SCHC gateway tries retransmitting up to \texttt{MAX\_ACK\_REQUESTS} times before aborting.

Following the reception of a FCN=All-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 SCHC gateway receives this ACK, the current fragmentation session has succeeded and its context can be cleared.

If the retransmission timer elapses and the SCHC gateway has not received the SCHC ACK it retransmits the last fragment with the payload (not an ACK-request without payload). The SCHC gateway tries retransmitting up to \texttt{MAX\_ACK\_REQUESTS} times before aborting.

The device SHALL keep the SCHC ACK message in memory until it receives a downlink from the SCHC gateway different from the last (FCN>0 and different DTag) fragment indicating that the SCHC gateway has received the ACK message.

Following the reception of a FCN=All-1 fragment (the last fragment of a datagram), if all fragments have been received and if the MIC is NOT correct, the device shall transmit a Receiver-Abort fragment. The retransmission timer is used by the SCHC gateway (the sender), the optimal value is very much application specific but here are some recommended default values. For classB end-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 end-devices which are nearly constantly receiving, the recommended value is 30 seconds. This means that the end-device shall try to transmit the ACK within 30 seconds of the reception of each fragment. The inactivity timer is implemented by the end-device to flush the context in-case it receives nothing from the SCHC gateway over an extended period of time. The recommended value is 12 hours for both classB&C end-devices.

6. Security considerations

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].
Acknowledgements

Thanks to all those listed in the Contributors section for the excellent text, insightful discussions, reviews and suggestions.

Contributors

Contributors ordered by family name.

- ins: J. Catalano name: Julien Catalano org: Kerlink street: 1 rue Jacqueline Auriol city: 35235 Thorigne-Fouillard country: France email: j.catalano@kerlink.fr
- ins: M. Coracin name: Michael Coracin org: Semtech street: 14 Chemin des Clos city: Meylan country: France email: mcoracin@semtech.com
- ins: M. Le Gourrierec name: Marc Le Gourrierec org: SagemCom street: 250 Route de l’Empereur city: 92500 Rueil Malmaison country: FRANCE email: marc.legourrierec@sagemcom.com
- ins: N. Sornin name: Nicolas Sornin org: Semtech street: 14 Chemin des Clos city: Meylan country: France email: nsornin@semtech.com

9. References

9.1. Normative References


9.2. Informative References

[I-D.ietf-lpwan-ipv6-static-context-hc]

[lora-alliance-spec]

Appendix A. Examples

A.1. Uplink - Compression example - No fragmentation

Figure 16 is representing an applicative payload going through SCHC, no fragmentation required
An applicative payload of 78 bytes is passed to SCHC compression layer using rule 1, allowing to compress it to 40 bytes and 5 bits: 21 bits residue + 38 bytes payload.

<table>
<thead>
<tr>
<th>RuleID</th>
<th>Compression residue</th>
<th>Payload</th>
<th>Padding=0b000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21 bits</td>
<td>38 bytes</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>LoRaWAN Header</th>
<th>RuleID</th>
<th>Compression residue</th>
<th>Payload</th>
<th>Padding=b’000</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>1</td>
<td>21 bits</td>
<td>38 bytes</td>
<td>3 bits</td>
</tr>
</tbody>
</table>

Figure 16: Uplink example: compression without fragmentation

A.2. Uplink - Compression and fragmentation example

Figure 17 is representing an applicative payload going through SCHC, with fragmentation.

An applicative payload of 478 bytes is passed to SCHC compression layer using rule 1, allowing to compress it to 440 bytes: 21 bits residue + 138 bytes payload.

<table>
<thead>
<tr>
<th>RuleID</th>
<th>Compression residue</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21 bits</td>
<td>138 bytes</td>
</tr>
</tbody>
</table>

Given the size of the payload, FPortUpDefault will be used. The current LoRaWAN MTU is 11 bytes, although 2 bytes FOpts are used by LoRaWAN protocol: 9 bytes are available for SCHC payload. SCHC header is 2 bytes so 2 tiles are send in first fragment.

<table>
<thead>
<tr>
<th>LoRaWAN Header</th>
<th>FOpts</th>
<th>RuleID</th>
<th>DTag</th>
<th>W</th>
<th>FCN</th>
<th>2 tiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>2 bytes</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>126</td>
<td>6 bytes</td>
</tr>
</tbody>
</table>

Content of the two tiles is:

<table>
<thead>
<tr>
<th>RuleID</th>
<th>Compression residue</th>
<th>Payload</th>
</tr>
</thead>
</table>

Gimenez & Petrov Expires January 9, 2020 [Page 19]
Next transmission MTU is 242 bytes, no FOpts. 80 tiles are transmitted:

| LoRaWAN Header | RuleID | DTag  | W    | FCN   | 80 tiles |
|----------------+--------+-------+------|-------|---------|
|                | XXXX   | 0     | 0    | 0     | 240 bytes |

Next transmission MTU is 242 bytes, no FOpts. All 65 remaining tiles are transmitted, last tile is only 2 bytes. Padding is added for the remaining 6 bits.

| LoRaWAN Header | RuleID | DTag  | W    | FCN   | MIC  | 65 tiles | Padding |
|----------------+--------+-------+------|-------|------|---------|---------|
|                | XXXX   | 0     | 0    | 0     | 127  | CRC32    | 3 bits  |

All packets have been received by the SCHC gateway, computed MIC is correct so the following ACK is send to the device:

| LoRaWAN Header | RuleID | DTag  | W    | C    |
|----------------+--------+-------+------|------|
|                | XXXX   | 0     | 0    | 1    |

Figure 17: Uplink example: compression and fragmentation

A.3. Downlink

An applicative payload of 43 bytes is passed to SCHC compression layer using rule 1, allowing to compress it to 24 bytes and 5 bits: 21 bits residue + 22 bytes payload.

| RuleID | Compression residue | Payload |
|--------+--------------------+---------|
| 1      | 21 bits             | 18 bytes |

The current LoRaWAN MTU is 11 bytes, although 2 bytes FOpts are used by LoRaWAN protocol: 9 bytes are available for SCHC payload => it has to be fragmented.

| LoRaWAN Header | FOpts | RuleID | W    | FCN   | 1 tile |
|----------------+-------+--------+------|-------|--------|
|                | XXXX   | 2 bytes | 0    | 0     | 8 bytes |
Content of the two tiles is:

<table>
<thead>
<tr>
<th>RuleID</th>
<th>Compression residue</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21 bits</td>
<td>2 bytes + 5 bits</td>
</tr>
</tbody>
</table>

The receiver answers with an SCHC ACK

<table>
<thead>
<tr>
<th>RuleID</th>
<th>W = 0</th>
<th>C = b’1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

The second downlink is send, no FOpts:

<table>
<thead>
<tr>
<th>LoRaWAN Header</th>
<th>RuleID</th>
<th>W</th>
<th>FCN</th>
<th>1 tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10 bytes</td>
</tr>
</tbody>
</table>

The receiver answers with an SCHC ACK

<table>
<thead>
<tr>
<th>RuleID</th>
<th>W = 1</th>
<th>C = b’1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

The third downlink is send, no FOpts:

<table>
<thead>
<tr>
<th>LoRaWAN Header</th>
<th>RuleID</th>
<th>W</th>
<th>FCN</th>
<th>1 tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXXX</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10 bytes</td>
</tr>
</tbody>
</table>

The receiver answers with an SCHC ACK

<table>
<thead>
<tr>
<th>RuleID</th>
<th>W = 0</th>
<th>C = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>1 bit</td>
<td>1 bit</td>
</tr>
</tbody>
</table>

The last downlink is send, no FOpts:
The receiver answers with an SCHC ACK

```
| RuleID | W = 1 | C = 1 |
+ ------ + ----- + ------- +
| 6 bits | 1 bit | 1 bit |
```

Figure 18: Downlink example: compression and fragmentation

Appendix B. Note

Authors’ Addresses

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

Email: ogimenez@semtech.com

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

Email: ivaylo@ackl.io
SCHC over Sigfox LPWAN
draft-ietf-lpwan-schc-over-sigfox-00

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

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

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

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

This Internet-Draft will expire on October 26, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of
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 LPWAN 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.

![Diagram of architecture](image)

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 remaining 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.
Internet-Draft SCHC over Sigfox LPWAN April 2019

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]


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/Estève 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
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

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

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

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

This Internet-Draft will expire on September 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

Minaburo & Toutain Expires September 12, 2019
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 (FL), 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:

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

- 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.
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 Field";
    }
}

Minaburo & Toutain     Expires September 12, 2019
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 occurrences";
}

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.

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

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


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