Data Model for Static Context Header Compression (SCHC)
draft-ietf-lpwan-schc-yang-data-model-14

Abstract

This document describes a YANG data model for the SCHC (Static Context Header Compression) compression and fragmentation rules.

Status of This Memo

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

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

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

This Internet-Draft will expire on 12 January 2023.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

SCHC is a compression and fragmentation mechanism for constrained
networks defined in [RFC8724]. It is based on a static context shared by two entities at the boundary of the constrained network. [RFC8724] provides a non formal representation of the rules used either for compression/decompression (or C/D) or fragmentation/reassembly (or F/R). The goal of this document is to formalize the description of the rules to offer:

* the same definition on both ends, even if the internal representation is different.

* an update of the other end to set up some specific values (e.g. IPv6 prefix, destination address,...)

* ...

[I-D.ietf-lpwan-architecture] illustrates the exchange of rules using the YANG data model.

This document defines a YANG module [RFC7950] to represent both compression and fragmentation rules, which leads to common representation for values for all the rules elements.

2. Requirements Language

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

3. SCHC rules

This document defines a YANG module to represent both compression and fragmentation rules, which leads to common representation for values for all the rules elements.

SCHC compression is generic, the main mechanism does not refer to a specific protocol. Any header field is abstracted through an ID, a position, a direction, and a value that can be a numerical value or a string. [RFC8724] and [RFC8824] specify fields for IPv6 [RFC8200], UDP[RFC0768], CoAP [RFC7252] including options defined for no
serveur response [RFC7967] and OSCORE [RFC8613]. For the latter [RFC8824] splits this field into sub-fields.

SCHC fragmentation requires a set of common parameters that are included in a rule. These parameters are defined in [RFC8724].

The YANG data model allows to select the compression or the fragmentation using the feature command.

3.1. Compression Rules

[RFC8724] proposes a non formal representation of the compression rule. A compression context for a device is composed of a set of rules. Each rule contains information to describe a specific field in the header to be compressed.

```
+-----------------------------------------------------------------+
|                      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|||
|+-------+--+--+--+------------+-----------------+---------------+|||
||...    |..|..|..|   ...      | ...             | ...           |||
|+-------+--+--+--+------------+-----------------+---------------+|||
|                   Rule N                                     |
+-----------------------------------------------------------------+|
```

Figure 1: Compression Decompression Context

3.2. Identifier generation
Identifier used in the SCHC YANG data model are from the identityref statement to ensure to be globally unique and be easily augmented if needed. The principle to define a new type based on a group of identityref is the following:

* define a main identity ending with the keyword base-type.

* derive all the identities used in the Data Model from this base type.

* create a typedef from this base type.

The example (Figure 2) shows how an identityref is created for RCS (Reassembly Check Sequence) algorithms used during SCHC fragmentation.

```Yang
// -- RCS algorithm types

identity rcs-algorithm-base-type {
  description
    "Identify which algorithm is used to compute RCS. The algorithm also defines the size of the RCS field.";
}

identity rcs-RFC8724 {
  base rcs-algorithm-base-type;
  description
    "CRC 32 defined as default RCS in RFC8724. RCS is 4 byte-long";
}

typedef rcs-algorithm-type {
  type identityref {
    base rcs-algorithm-base-type;
  }
  description
    "type used in rules.";
}
Figure 2: Principle to define a type based on identityref.

3.3. Convention for Field Identifier

In the process of compression, the headers of the original packet are first parsed to create a list of fields. This list of fields is matched against the rules to find the appropriate rule and apply compression. [RFC8724] does not state how the field ID value is constructed. In examples, identification is done through a string indexed by the protocol name (e.g. IPv6.version, CoAP.version,...).

The current YANG data model includes fields definitions found in [RFC8724], [RFC8824].

Using the YANG data model, each field MUST be identified through a global YANG identityref. A YANG field ID for the protocol is always derived from the fid-base-type. Then an identity for each protocol is specified using the naming convention fid-<<protocol name>>-base-type. All possible fields for this protocol MUST derive from the protocol identity. The naming convention is "fid" followed by the protocol name and the field name. If a field has to be divided into sub-fields, the field identity serves as a base.

The full field-id definition is found in Section 5. A type is defined for IPv6 protocol, and each field is based on it. Note that the DiffServ bits derives from the Traffic Class identity.

3.4. Convention for Field length

Field length is either an integer giving the size of a field in bits or a specific function. [RFC8724] defines the "var" function which allows variable length fields (whose length is expressed in bytes) and [RFC8824] defines the "tkl" function for managing the CoAP Token length field.

The naming convention is "fl" followed by the function name.

The field length function can be defined as an identityref as described in Section 5. Therefore, the type for field length is a union between an integer giving in bits the size of the length and the identityref.
3.5. Convention for Field position

Field position is a positive integer which gives the position of a field, the default value is 1, and incremented at each repetition. Value 0 indicates that the position is not important and is not considered during the rule selection process.

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

3.6. Convention for Direction Indicator

The Direction Indicator (di) is used to tell if a field appears in both direction (Bi) or only uplink (Up) or Downlink (Dw). The naming convention is "di" followed by the Direction Indicator name.

The type is "di-type".

3.7. Convention for Target Value

The Target Value is a list of binary sequences of any length, aligned to the left. In the rule, the structure will be used as a list, with index as a key. The highest index value is used to compute the size of the index sent in residue for the match-mapping CDA (Compression Decompression Action). The index allows to specify several values:

* For Equal and LSB, Target Value contains a single element. Therefore, the index is set to 0.

* For match-mapping, Target Value can contain several elements. Index values MUST start from 0 and MUST be contiguous.

If the header field contains a text, the binary sequence uses the same encoding.

3.8. Convention for Matching Operator

Matching Operator (MO) is a function applied between a field value provided by the parsed header and the target value. [RFC8724] defines 4 MO.

The naming convention is "mo" followed by the MO name.
The type is "mo-type"

3.8.1. Matching Operator arguments

They are viewed as a list, built with a tv-struct (see chapter Section 3.7).

3.9. Convention for Compression Decompression Actions

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

The naming convention is "cda" followed by the CDA name.

3.9.1. Compression Decompression Action arguments

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

3.10. Fragmentation rule

Fragmentation is optional in the data model and depends on the presence of the "fragmentation" feature.

Most of the fragmentation parameters are listed in Annex D of [RFC8724].

Since fragmentation rules work for a specific direction, they MUST contain a mandatory direction indicator. The type is the same as the one used in compression entries, but bidirectional MUST NOT be used.

3.10.1. Fragmentation mode

[RFC8724] defines 3 fragmentation modes:

* No Ack: this mode is unidirectional, no acknowledgment is sent back.

* Ack Always: each fragmentation window must be explicitly
acknowledged before going to the next.

* Ack on Error: A window is acknowledged only when the receiver detects some missing fragments.

The type is "fragmentation-mode-type". The naming convention is "fragmentation-mode" followed by the fragmentation mode name.

3.10.2. Fragmentation Header

A data fragment header, starting with the rule ID can be sent on the fragmentation direction. [RFC8724] indicates that the SCHC header may be composed of (cf. Figure 3):

* a Datagram Tag (Dtag) identifying the datagram being fragmented if the fragmentation applies concurrently on several datagrams. This field is optional and its length is defined by the rule.

* a Window (W) used in Ack-Always and Ack-on-Error modes. In Ack-Always, its size is 1. In Ack-on-Error, it depends on the rule. This field is not needed in No-Ack mode.

* a Fragment Compressed Number (FCN) indicating the fragment/tile position within the window. This field is mandatory on all modes defined in [RFC8724], its size is defined by the rule.

|--- SCHC Fragment Header ----|
|--- T --|--M--|-- N --|
| RuleID | DTag | W | FCN | Fragment Payload | padding (as needed)
|--- ... ++ ... ++++++ ... ++++++++...+.................................|

Figure 3: Data fragment header from RFC8724

3.10.3. Last fragment format

The last fragment of a datagram is sent with an RCS (Reassembly Check Sequence) field to detect residual transmission error and possible losses in the last window. [RFC8724] defines a single algorithm based on Ethernet CRC computation.

The naming convention is "rcs" followed by the algorithm name.

For Ack-on-Error mode, the All-1 fragment may just contain the RCS or can include a tile. The parameters defines the behavior:

* all-1-data-no: the last fragment contains no data, just the RCS
* all-1-data-yes: the last fragment includes a single tile and the RCS

* all-1-data-sender-choice: the last fragment may or may not contain a single tile. The receiver can detect if a tile is present.

The naming convention is "all-1-data" followed by the behavior identifier.

3.10.4. Acknowledgment behavior

The acknowledgment fragment header goes in the opposite direction of data. [RFC8724] defines the header, composed of (see Figure 4):

* a Dtag (if present).

* a mandatory window as in the data fragment.

* a C bit giving the status of RCS validation. In case of failure, a bitmap follows, indicating the received tile.

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

+-- ... -+- ... --+-+-+-+-+------------------ ... --+-+-+-+-+------------------
| RuleID | DTag | W |C=0|Compressed Bitmap| pad. as needed (failure)
+-- ... -+- ... --+-+-+-+-+------------------ ... --+-+-+-+-+------------------

Figure 4: Acknowledgment fragment header for RFC8724

For Ack-on-Error, SCHC defines when an acknowledgment can be sent. This can be at any time defined by the layer 2, at the end of a window (FCN all-0) or as a response to receiving the last fragment (FCN all-1). The naming convention is "ack-behavior" followed by the algorithm name.

3.10.5. Timer values

The state machine requires some common values to handle correctly fragmentation.

* retransmission-timer gives the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specified, value
must be strictly positive.

* inactivity-timer gives the duration before aborting a fragmentation session (cf. section 8.2.2.4. of [RFC8724]). The value 0 explicitly indicates that this timer is disabled.

[RFC8724] do not specified any range for these timers. [RFC9011] recommends a duration of 12 hours. In fact, the value range should be between milliseconds for real time systems to several days. To allow a large range of applications, two parameters must be specified:

* the duration of a tick. It is computed by this formula $2^{\text{tick-duration}}/10^{6}$. When tick-duration is set to 0, the unit is the microsecond. The default value of 20 leads to a unit of 1.048575 second. A value of 32 leads to a tick duration of about 1 hour 11 minutes.

* the number of ticks in the predefined unit. With the default tick-duration value of 20, the timers can cover a range between 1.0 sec and 19 hours covering [RFC9011] recommendation.

3.10.6. Fragmentation Parameter

The SCHC fragmentation protocol specifies the the number of attempts before aborting through the parameter:

* max-ack-requests (cf. section 8.2.2.4. of [RFC8724]).

3.10.7. Layer 2 parameters

The data model includes two parameters needed for fragmentation:

* l2-word-size: [RFC8724] base fragmentation, in bits, on a layer 2 word which can be of any length. The default value is 8 and correspond to the default value for byte aligned layer 2. A value of 1 will indicate that there is no alignment and no need for padding.

* maximum-packet-size: defines the maximum size of a uncompressed datagram. By default, the value is set to 1280 bytes.
They are defined as unsigned integer, see Section 5.

4. Rule definition

A rule is identified by a unique rule identifier (rule ID) comprising both a Rule ID value and a Rule ID length. The YANG grouping rule-id-type defines the structure used to represent a rule ID. A length of 0 is allowed to represent an implicit rule.

Three types of rules are defined in [RFC8724]:

* Compression: a compression rule is associated with the rule ID.

* No compression: this identifies the default rule used to send a packet in extenso when no compression rule was found (see [RFC8724] section 6).

* Fragmentation: fragmentation parameters are associated with the rule ID. Fragmentation is optional and feature "fragmentation" should be set.

To access a specific rule, the rule ID length and value are used as a key. The rule is either a compression or a fragmentation rule.

4.1. Compression rule

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

The compression rule described Figure 1 is defined by compression-content. It defines a list of compression-rule-entry, indexed by their field id, position and direction. The compression-rule-entry element represent a line of the table Figure 1. Their type reflects the identifier types defined in Section 3.1

Some checks are performed on the values:

* target value must be present for MO different from ignore.

* when MSB MO is specified, the matching-operator-value must be
4.2. Fragmentation rule

A Fragmentation rule is composed of entries describing the protocol behavior. Some of them are numerical entries, others are identifiers defined in Section 3.10.

4.3. YANG Tree

module: ietf-schc
  +--rw schc
    +--rw rule* [rule-id-value rule-id-length]
      +--rw rule-id-value             uint32
      +--rw rule-id-length            uint8
      +--rw (nature)?

  +--:(fragmentation) {fragmentation}?  
    | +--rw fragmentation-mode          schc:fragmentation-mode-type
    | +--rw l2-word-size?               uint8
    | +--rw direction                   schc:di-type
    | +--rw dtag-size?                  uint8
    | +--rw w-size?                     uint8
    | +--rw fcn-size                    uint8
    | +--rw rcs-algorithm?              rcs-algorithm-type
    | +--rw maximum-packet-size?        uint16
    | +--rw window-size?                uint16
    | +--rw max-interleaved-frames?     uint8
    | +--rw inactivity-timer
    | | +--rw ticks-duration?           uint8
    | | +--rw ticks-numbers?            uint16
    | +--rw retransmission-timer
    | | +--rw ticks-duration?           uint8
    | | +--rw ticks-numbers?            uint16
    | +--rw max-ack-requests?           uint8
    +--rw (mode)?
    | +--:(no-ack)
    | +--:(ack-always)
    | +--:(ack-on-error)
    |   +--rw tile-size?               uint8
    |   +--rw tile-in-All1?            schc:all1-data-type
    |   +--rw ack-behavior?            schc:ack-behavior-type
Figure 5

5. YANG Module

<CODE BEGINS> file "ietf-schc@2022-07-11.yang"
module ietf-schc {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-schc";
  prefix schc;

  organization
    "IETF IPv6 over Low Power Wide-Area Networks (lpwan) working group";
  contact
    "WG Web:  <https://datatracker.ietf.org/wg/lpwan/about/>"
    WG List:  <mailto:p-wan@ietf.org>
    Editor:  Laurent Toutain
              <mailto:laurent.toutain@imt-atlantique.fr>
    Editor:  Ana Minaburo
              <mailto:ana@ackl.io>
  description
    ""
Generic Data model for Static Context Header Compression Rule for SCHC, based on RFC 8724 and RFC8824. Include compression, no compression and fragmentation rules.

This module is a YANG model for SCHC rules (RFC 8724 and RFC8824). RFC 8724 describes compression rules in a abstract way through a table.
This module proposes a global data model that can be used for rule exchanges or modification. It proposes both the data model format and the global identifiers used to describe some operations in fields. This data model applies to both compression and fragmentation.

revision 2022-07-11 {
    description
        "Initial version from RFC XXXX ";
    reference
        "RFC XXX: Data Model for Static Context Header Compression (SCHC)";
}

feature compression {
    description
        "SCHC compression capabilities are taken into account";
}

feature fragmentation {
    description
        "SCHC fragmentation capabilities are taken into account";
}

// -------------------------
//  Field ID type definition
// -------------------------
// generic value TV definition

identity fid-base-type {
    description
        "Field ID base type for all fields";
}

identity fid-ipv6-base-type {
    base fid-base-type;
    description
        "Field ID base type for IPv6 headers described in RFC 8200";
    reference
        "RFC 8200";
}

// RFC 8200: "IPv6 Header Compression (IHC)"
"RFC 8200" Internet Protocol, Version 6 (IPv6) Specification;

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

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

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

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

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

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

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

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

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

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

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

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

identity fid-udp-base-type {
  base fid-base-type;
  description
    "Field ID base type for UDP headers described in RFC 768";
  reference
    "RFC768 User Datagram Protocol";
}
identity fid-udp-dev-port {
    base fid-udp-base-type;
    description
        "UDP source or destination port from RFC 768, if uplink or
downlink communication, respectively.";
}

identity fid-udp-app-port {
    base fid-udp-base-type;
    description
        "UDP destination or source port from RFC 768, if uplink or
downlink communication, respectively.";
}

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

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

identity fid-coap-base-type {
    base fid-base-type;
    description
        "Field ID base type for UDP headers described in RFC 7252";
    reference
        "RFC 7252 The Constrained Application Protocol (CoAP)";
}

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

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

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

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

identity fid-coap-code-detail {
    base fid-coap-code;
    description
        "CoAP code detail from RFC 7252";
}

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

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

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

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

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

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

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

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

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

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

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

identity fid-coap-option-accept {
    base fid-coap-base-type;
    description
        "CoAP option Accept from RFC 7252";
}

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

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

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

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

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

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

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

identity fid-coap-option-no-response {
    base fid-coap-base-type;
    description
        "CoAP option No response from RFC 7967";
    reference
        "RFC7967 Constrained Application Protocol (CoAP) Option for No Server Response";
}
identity fid-oscore-base-type {
    base fid-coap-type;
    description "OSCORE options (RFC8613) split by RFC 8824 in sub options";
    reference "RFC8824 Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)";
}

identity fid-coap-option-oscore-flags {
    base fid-oscore-base-type;
    description "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

identity fid-coap-option-oscore-piv {
    base fid-oscore-base-type;
    description "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

identity fid-coap-option-oscore-kid {
    base fid-oscore-base-type;
    description "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

identity fid-coap-option-oscore-kidctx {
    base fid-oscore-base-type;
    description "CoAP option oscore flags (see RFC 8824, section 6.4)";
}

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

identity fl-base-type {
    description "Used to extend field length functions.";
}
identity fl-variable {
    base fl-base-type;
    description
        "Residue length in Byte is sent as defined for CoAP in RFC 8824 (cf. 5.3).";
}

identity fl-token-length {
    base fl-base-type;
    description
        "Residue length in Byte is sent as defined for CoAP in RFC 8824 (cf. 4.5).";
}

identity di-base-type {
    description
        "Used to extend direction indicators.";
}

identity di-bidirectional {
    base di-base-type;
    description
        "Direction Indication of bidirectionality in RFC 8724 (cf. 7.1).";
}

identity di-up {
    base di-base-type;
    description
        "Direction Indication of uplink defined in RFC 8724 (cf. 7.1).";
}

identity di-down {
    base di-base-type;
    description
        "Direction Indication of downlink defined in
identity mo-base-type {
    description
        "Used to extend Matching Operators with SID values";
}

identity mo-equal {
    base mo-base-type;
    description
        "Equal MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-ignore {
    base mo-base-type;
    description
        "Ignore MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-msb {
    base mo-base-type;
    description
        "MSB MO as defined in RFC 8724 (cf. 7.3)";
}

identity mo-match-mapping {
    base mo-base-type;
    description
        "match-mapping MO as defined in RFC 8724 (cf. 7.3)";
}

identity cda-base-type {
    description
        "Compression Decompression Actions.";
}

identity cda-not-sent {
    base cda-base-type;
    description
"not-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-value-sent {
  base cda-base-type;
  description
    "value-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-lsb {
  base cda-base-type;
  description
    "LSB CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-mapping-sent {
  base cda-base-type;
  description
    "mapping-sent CDA as defined in RFC 8724 (cf. 7.4).";
}

identity cda-compute {
  base cda-base-type;
  description
    "compute-* CDA as defined in RFC 8724 (cf. 7.4)";
}

identity cda-deviid {
  base cda-base-type;
  description
    "deviid CDA as defined in RFC 8724 (cf. 7.4)";
}

identity cda-appiid {
  base cda-base-type;
  description
    "appiid CDA as defined in RFC 8724 (cf. 7.4)";
}

// -- type definition
typedef fid-type {
typedef fl-type {
  type union {
    type int64; /* positive integer, expressing length in bits */
    type identityref { /* function */
      base fl-base-type;
    }
  }
}

description
  "Field length either a positive integer expressing the size in
   bits or a function defined through an identityref.";

reference
  "RFC 8724 SCHC: Generic Framework for Static Context Header  
   Compression and Fragmentation"

}

typedef di-type {
  type identityref {
  base di-base-type;
}

description
  "Direction in LPWAN network, up when emitted by the device, 
   down when received by the device, bi when emitted or 
   received by the device.";

reference
  "RFC 8724 SCHC: Generic Framework for Static Context Header  
   Compression and Fragmentation"

}

typedef mo-type {
  type identityref {
  base mo-base-type;
  }
typedef cda-type {
    type identityref {
        base cda-base-type;
    }
}

description
    "Matching Operator (MO) to compare fields values with
target values";
reference
    "RFC 8724 SCHC: Generic Framework for Static Context Header
     Compression and Fragmentation";

typedef cda-type {
    type identityref {
        base cda-base-type;
    }
}

description
    "Compression Decompression Action to compression or
decompress a field."
reference
    "RFC 8724 SCHC: Generic Framework for Static Context Header
     Compression and Fragmentation";

// -- FRAGMENTATION TYPE
// -- fragmentation modes

identity fragmentation-mode-base-type {
    description
        "fragmentation mode."
    reference
        "RFC 8724 SCHC: Generic Framework for Static Context Header
         Compression and Fragmentation";
}

identity fragmentation-mode-no-ack {
    base fragmentation-mode-base-type;
    description
        "No-ACK of RFC8724.";
}

identity fragmentation-mode-ack-always {
    base fragmentation-mode-base-type;
    description

typedef fragmentation-mode-type {
  type identityref {
    base fragmentation-mode-base-type;
  }
  description
    "type used in rules";
}

// -- Ack behavior

identity ack-behavior-base-type {
  description
    "Define when to send an Acknowledgment.";
  reference
    "RFC 8724 SCHC: Generic Framework for Static Context Header Compression and Fragmentation";
}

identity ack-behavior-after-all-0 {
  base ack-behavior-base-type;
  description
    "Fragmentation expects Ack after sending All-0 fragment.";
}

identity ack-behavior-after-all-1 {
  base ack-behavior-base-type;
  description
    "Fragmentation expects Ack after sending All-1 fragment.";
}

identity ack-behavior-by-layer2 {
  base ack-behavior-base-type;
typedef ack-behavior-type {
    type identityref {
        base ack-behavior-base-type;
    }
    description
        "Type used in rules.";
}

// -- All-1 with data types

typedef all-1-data-type {
    type identityref {
        base all-1-data-base-type;
    }
    description
        "Type to define when to send an Acknowledgment message.";
    reference
        "RFC 8724 SCH: Generic Framework for Static Context Header
           Compression and Fragmentation";
}

identity all-1-data-base-type {
    description
        "Type to define when to send an Acknowledgment message.";
    reference
        "RFC 8724 SCH: Generic Framework for Static Context Header
           Compression and Fragmentation";
}

identity all-1-data-no {
    base all-1-data-base-type;
    description
        "All-1 contains no tiles.";
}

identity all-1-data-yes {
    base all-1-data-base-type;
    description
        "All-1 MUST contain a tile.";
}

identity all-1-data-sender-choice {
    base all-1-data-base-type;
    description
        "Fragmentation process chooses to send tiles or not in All-1.";
}

typedef all-1-data-type {
    type identityref {
base all-1-data-base-type;
}
description
  "Type used in rules."
}

// -- RCS algorithm types

identity rcs-algorithm-base-type {
  description
    "Identify which algorithm is used to compute RCS.
    The algorithm also defines the size of the RCS field."
  reference
    "RFC 8724 SCHC: Generic Framework for Static Context Header
    Compression and Fragmentation";
}

identity rcs-RFC8724 {
  base rcs-algorithm-base-type;
  description
    "CRC 32 defined as default RCS in RFC8724. RCS is 4 byte-long";
}

typedef rcs-algorithm-type {
  type identityref {
    base rcs-algorithm-base-type;
  }
  description
    "type used in rules.";
}

// --------- TIMER DURATION -------------------

grouping timer-duration {
  leaf ticks-duration {
    type uint8;
    default "20";
    description
      "duration of one tick in micro-seconds:
      2^ticks-duration/10^6 = 1.048s";
  }
}
leaf ticks-numbers {
  type uint16;
  description "timer duration = ticks-numbers * 2^ticks-duration / 10^6";
}
description "used by inactivity and retransmission timer. Allows a precision from micro-second to year by sending the tick-duration value.
For instance:

<table>
<thead>
<tr>
<th>tick-duration / smallest value</th>
<th>highest value</th>
</tr>
</thead>
<tbody>
<tr>
<td>20: 00y 000d 00h 00m 01s.048575&lt;-&gt;00y 000d 19h 05m 18s.428159</td>
<td></td>
</tr>
<tr>
<td>21: 00y 000d 00h 00m 02s.097151&lt;-&gt;00y 001d 14h 10m 36s.856319</td>
<td></td>
</tr>
<tr>
<td>22: 00y 000d 00h 00m 04s.194303&lt;-&gt;00y 003d 04h 21m 13s.712639</td>
<td></td>
</tr>
<tr>
<td>23: 00y 000d 00h 00m 08s.388607&lt;-&gt;00y 006d 08h 42m 27s.425279</td>
<td></td>
</tr>
<tr>
<td>24: 00y 000d 00h 00m 16s.777215&lt;-&gt;00y 012d 17h 24m 54s.850559</td>
<td></td>
</tr>
<tr>
<td>25: 00y 000d 00h 00m 33s.554431&lt;-&gt;00y 025d 10h 49m 49s.701119</td>
<td></td>
</tr>
</tbody>
</table>

Note that the smallest value is also the incrementation step, so the timer precision.
",
reference "RFC 8724 SCHC: Generic Framework for Static Context Header Compression and Fragmentation";
}

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

grouping tv-struct {
  description "Defines the target value element. If the header field contains a text, the binary sequence uses the same encoding. field-id allows the conversion to the appropriate type.";
  leaf value {
    type binary;
    description "Target Value";
  }
  leaf index {

type uint16;

description
    "Index gives the position in the matching-list. If only one
    element is present, index is 0. Otherwise, indicia is the
    the order in the matching list, starting at 0."
}

reference
    "RFC 8724  SCHC: Generic Framework for Static Context Header
    Compression and Fragmentation"
}

grouping compression-rule-entry {

description
    "These entries defines a compression entry (i.e. a line)
    as defined in RFC 8724.

+-------+--+--+--+------------+-----------------+---------------+
|Field 1|FL|FP|DI|Target Value|Matching Operator|Comp/Decomp Act|
+-------+--+--+--+------------+-----------------+---------------+

An entry in a compression rule is composed of 7 elements:
- Field ID: The header field to be compressed. The content
  is a YANG identifier.
- Field Length : either a positive integer of a function
defined as a YANG id.
- Field Position: a positive (and possibly equal to 0)
  integer.
- Direction Indicator: a YANG identifier giving the direction.
- Target value: a value against which the header Field is
  compared.
- Matching Operator: a YANG id giving the operation,
  parameters may be associated to that operator.
- Comp./Decomp. Action: A YANG id giving the compression or
  decompression action, parameters may be associated to that
  action.
";

leaf field-id {
    type schc:fid-type;
    mandatory true;
    description
        "Field ID, identify a field in the header with a YANG
         referenceid.";
}

leaf field-length {
    type schc:fl-type;
    mandatory true;
    description
        "Field Length, expressed in number of bits or through a
function defined as a YANG referenceid.

leaf field-position {
  type uint8;
  mandatory true;
  description
    "Field position in the header is an integer. Position 1
     matches the first occurrence of a field in the header,
     while incremented position values match subsequent
     occurrences.
     Position 0 means that this entry matches a field
     irrespective of its position of occurrence in the
     header.
     Be aware that the decompressed header may have
     position-0 fields ordered differently than they
     appeared in the original packet."
}

leaf direction-indicator {
  type schc:di-type;
  mandatory true;
  description
    "Direction Indicator, a YANG referenceid to say if the packet
     is bidirectional, up or down"
}

list target-value {
  key "index";
  uses tv-struct;
  description
    "A list of value to compare with the header field value.
     If target value is a singleton, position must be 0.
     For use as a matching list for the mo-match-mapping matching
     operator, positions should take consecutive values starting
     from 1."
}

leaf matching-operator {
  type schc:mo-type;
  must "../target-value or derived-from-or-self(., 'mo-ignore')" {
    error-message
      "mo-equal, mo-msb and mo-match-mapping need target-value"
    description
      "target-value is not required for mo-ignore"
  }
  must "not (derived-from-or-self(., 'mo-msb')) or
Matching Operator Arguments, based on TV structure to allow several arguments.

In RFC 8724, only the MSB matching operator needs arguments (a single argument, which is the number of most significant bits to be matched);

CDA: Compression Decompression Action.

CDA arguments, based on a TV structure, in order to allow for several arguments. The CDAs specified in RFC 8724 require no argument.

SCHC: Generic Framework for Static Context Header Compression and Fragmentation;

grouping compression-content {
    list entry {
        key "field-id field-position direction-indicator";
        uses compression-rule-entry;
        description

..
"A compression rule is a list of rule entries, each
describing a header field. An entry is identified
through a field-id, its position in the packet and
its direction."

{ 

description
"Define a compression rule composed of a list of entries.";
reference
"RFC 8724 SCHC: Generic Framework for Static Context Header
Compression and Fragmentation"
}
// SCHC Frag header format
leaf dtag-size {
  type uint8;
  default "0";
  description "Size, in bits, of the DTag field (T variable from RFC8724).";
}
leaf w-size {
  when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')"
    or
    derived-from(../fragmentation-mode, 'fragmentation-mode-ack-always') ";
  type uint8;
  description "Size, in bits, of the window field (M variable from RFC8724).";
}
leaf fcn-size {
  type uint8;
  mandatory true;
  description "Size, in bits, of the FCN field (N variable from RFC8724).";
}
leaf rcs-algorithm {
  type rcs-algorithm-type;
  default "schc:rcs-RFC8724";
  description "Algorithm used for RCS. The algorithm specifies the RCS size";
}
// SCHC fragmentation protocol parameters
leaf maximum-packet-size {
  type uint16;
  default "1280";
  description "When decompression is done, packet size must not strictly exceed this limit, expressed in bytes.";
}
leaf window-size {
type uint16;
description
"By default, if not specified 2^w-size - 1. Should not exceed
this value. Possible FCN values are between 0 and
window-size - 1."
}
leaf max-interleaved-frames {
  type uint8;
default "1";
description
"Maximum of simultaneously fragmented frames. Maximum value
is 2^dtag-size. All DTAG values can be used, but at most
max-interleaved-frames must be active at any time."
}
container inactivity-timer {
  uses timer-duration;
description
"Duration is seconds of the inactivity timer, 0 indicates
that the timer is disabled."
}
container retransmission-timer {
  uses timer-duration;
  when "derived-from(../fragmentation-mode,
       'fragmentation-mode-ack-on-error')
     or
       derived-from(../fragmentation-mode,
           'fragmentation-mode-ack-always')"
    description
"Duration in seconds of the retransmission timer."
}
leaf max-ack-requests {
  when "derived-from(../fragmentation-mode,
       'fragmentation-mode-ack-on-error')
     or
       derived-from(../fragmentation-mode,
           'fragmentation-mode-ack-always')"
  type uint8 {
    range "1..max";
  }
description
"The maximum number of retries for a specific SCHC ACK.";
choice mode {
    case no-ack;
    case ack-always;
    case ack-on-error {
        leaf tile-size {
            when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')";
            type uint8;
            description "Size, in bits, of tiles. If not specified or set to 0, tiles fill the fragment."
        }
        leaf tile-in-all-1 {
            when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')";
            type schc:all-1-data-type;
            description "Defines whether the sender and receiver expect a tile in All-1 fragments or not, or if it is left to the sender's choice."
        }
        leaf ack-behavior {
            when "derived-from(../fragmentation-mode, 'fragmentation-mode-ack-on-error')";
            type schc:ack-behavior-type;
            description "Sender behavior to acknowledge, after All-0, All-1 or when the LPWAN allows it."
        }
    }
}

description "RFC 8724 defines 3 fragmentation modes.";

// Define rule ID. Rule ID is composed of a RuleID value and a
// Rule ID Length
grouping rule-id-type {
  leaf rule-id-value {
    type uint32;
    description
    "Rule ID value, this value must be unique, considering its length.";
  }
  leaf rule-id-length {
    type uint8 {
      range "0..32";
    }
    description
    "Rule ID length, in bits. The value 0 is for implicit rules.";
  }
  description
  "A rule ID is composed of a value and a length, expressed in bits.";
  reference
  "RFC 8724 SCHC: Generic Framework for Static Context Header Compression and Fragmentation";
}

// SCHC table for a specific device.

container schc {
  list rule {
    key "rule-id-value rule-id-length";
    uses rule-id-type;
    choice nature {
      case fragmentation {
        if-feature "fragmentation";
        uses fragmentation-content;
      }
      case compression {
        if-feature "compression";
        uses compression-content;
      }
      case no-compression {

      }
}
"RFC8724 requires a rule for uncompressed headers."

} description
"A rule is for compression, for no-compression or for fragmentation."

} description
"Set of rules compression, no compression or fragmentation rules identified by their rule-id."

} description
"a SCHC set of rules is composed of a list of rules which are used for compression, no-compression or fragmentation."

reference
"RFC 8724 SCHC: Generic Framework for Static Context Header Compression and Fragmentation"

}<CODE ENDS>

Figure 6

6. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

* Openschc is implementing the conversion between the local rule representation and the representation conform to the data model in JSON and CBOR (following -08 draft).
7. IANA Considerations

This document registers one URIs and one YANG modules.

7.1. URI Registration

This document requests IANA to register the following four URIs in the "IETF XML Registry" [RFC3688]:

Registrant Contact:  The IESG.
XML:  N/A; the requested URI is an XML namespace.

7.2. YANG Module Name Registration

This document registers the following four YANG modules in the "YANG Module Names" registry [RFC6020].

name:           ietf-schc
prefix:         schc
reference:      RFC XXXX Data Model for Static Context Header Compression (SCHC)

8. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.
This data model formalizes the rules elements described in [RFC8724] for compression and fragmentation. As explained in the architecture document [I-D.ietf-lpwan-architecture], a rule can be read, created, updated or deleted in response to a management request. These actions can be done between two instances of SCHC or between a SCHC instance and a rule repository.

```
create
(-------)   read  +========+ *
( rules )<-----|Rule   |<--|-------->
(-------)   update |Manager|   NETCONF, RESTCONF or CORECONF
     . read  delete +========+ request
     .
     +--------+
     <===| R & D |<===
     ===>| C & F |===>
     +--------+
```

The rule contains some sensible informations such as the application IPv6 address. An attacker by changing a rule content may block the communication or intercept the traffic. Therefore, the identity of the requester must be validated. This can be done through certificates or access lists.

The full tree is sensitive, since it represents all the elements that can be managed. This module aims to be encapsulated into a YANG module including access right and identities.

9. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Ivan Martinez, Alexander Pelov for their careful reading and valuable inputs. A special thanks for Carl Moberg, Tom Petch and Eric Vyncke for their explanations and wise advices when building the model.

10. References

10.1. Normative References


10.2. Informative References

[I-D.ietf-lpwan-architecture]
Pelov, A., Thubert, P., and A. Minaburo, "LPWAN Static


Authors' Addresses

Minaburo & Toutain       Expires 12 January 2023               [Page 41]

Internet-Draft           LPWAN SCHC YANG module                July 2022

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