lpwan Working Group A. Minaburo
Internet-Draft Acklio
Intended status: Standards Track L. Toutain
Expires: August 6, 2021 Institut MINES TELECOM; IMT Atlantique
February 02, 2021

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 6, 2021.

Table of Contents

1. Introduction

2. SCHC rules

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

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

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

- ...

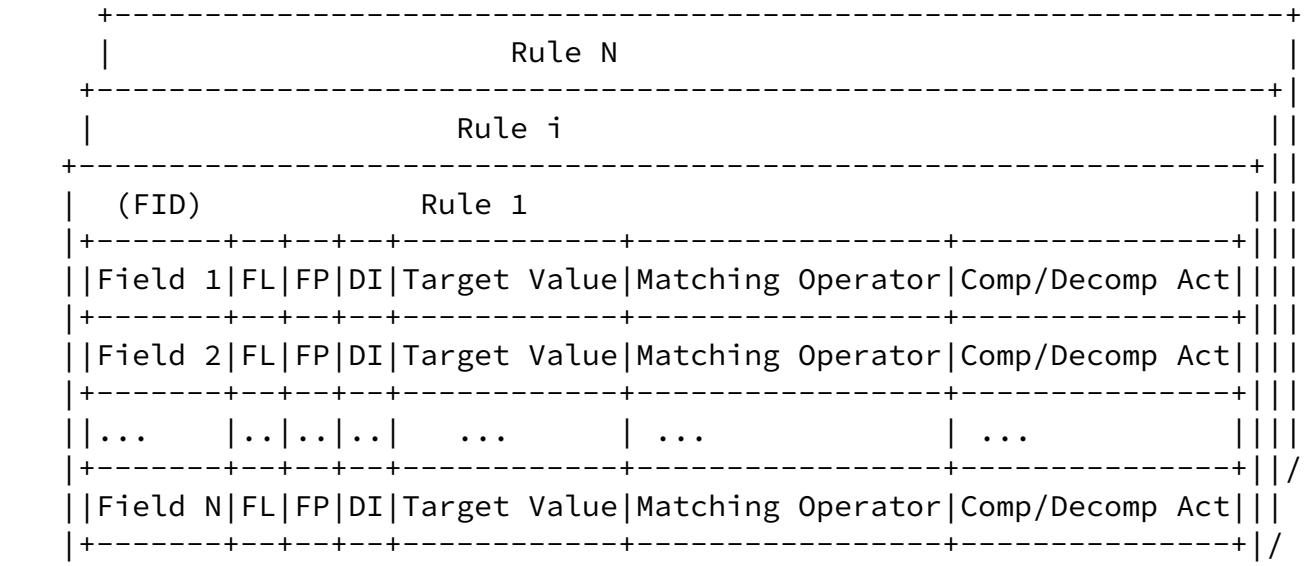
This document defines a YANG module to represent both compression and fragmentation rules, which leads to common representation for values for all the rules elements.

SCHC compression is generic, the main mechanism do no refers to a specific protocol. Any header field is abstracted through an ID, a position, a direction and a value that can be a numerical value or a string. [RFC8724] and [I-D.ietf-lpwan-coap-static-context-hc] specifies fields for IPv6, UDP, CoAP and OSCORE. [I-D.barthel-lpwan-oam-schc] decribes ICMPv6 header compression.

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

2.1. Compression Rules

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



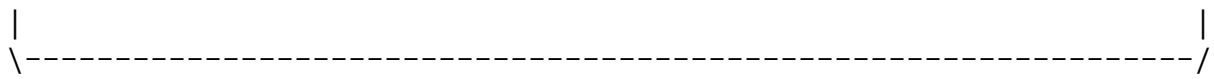


Figure 1: Compression Decompression Context

2.2. Field Identifier

In the process of compression, the headers of the original packet are first parsed to create a list of fields. This list of fields is matched against the rules to find the appropriate one and apply compression. The link between the list given by the parsed fields

and the rules is done through a field ID. [RFC8724] do not state how the field ID value can be constructed. In examples, identification is done through a string indexed by the protocol name (e.g. IPv6.version, CoAP.version,...).

Using the YANG model, each field MUST be identified through a global YANG identityref. A YANG field ID derives from the field-id-base-type. Figure 2 gives some field ID definitions. Note that some field IDs can be splitted is smaller pieces. This is the case for "fid-ipv6-trafficclass-ds" and "fid-ipv6-trafficclass-ecn" which are a subset of "fid-ipv6-trafficclass-ds".

```
   identity field-id-base-type  {
           description "Field ID with SID";
    }

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

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

  identity fid-ipv6-trafficclass-ds {
           base field-id-base-type;
           description "IPv6 Traffic Class field from RFC8200,
```

```
        DiffServ field from RFC3168";
    }

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

    ...
```

Figure 2: Definition of identityref for field IDs

Figure 2 gives some examples of field ID identityref definitions. The base identity is field-id-base-type, and field id are derived for it.

The naming convention is "fid" followed by the protocol name and the field name.
The yang model in annex (see Section 7) gives the full definition of the field ID for [RFC8724], [I-D.ietf-lpwan-coap-static-context-hc], and [I-D.barthel-lpwan-oam-schc].

The type associated to this identity is field-id-type (cf. Figure 3)

```
    typedef field-id-type {
        description "Field ID generic type.";
        type identityref {
            base field-id-base-type;
        }
    }
```

Figure 3: Type definition for field IDs

2.3. Field length

Field length is either an integer giving the size of a field in bits or a specific function. [RFC8724] defines the "var" function which allows variable length fields in byte and

[I-D.ietf-lpwan-coap-static-context-hc] defines the "tkl" function for managing the CoAP Token length field.

```
  identity field-length-base-type {
        description "used to extend field length functions";
  }

  identity fl-variable {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }

  identity fl-token-length {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }
```

Figure 4: Definition of identityref for field ILength

As for field ID, field length function can be defined as a identityref as shown in Figure 4.

Therefore the type for field length is a union between an integer giving in bits the size of the length and the identityref (cf. Figure 5).


```
 typedef field-length-type {
     description "Field length either a positive integer giving the size in
     or a function defined through an identityref.";
     type union {
         type int64; /* positive length in bits */
         type identityref { /* function */
             base field-length-base-type;
         }
     }
 }
```

Figure 5: Type definition for field Length

The naming convention is fl followed by the function name as defined in SCHC specifications.

2.4. Field position

Field position is a positive integer which gives the position of a field, the default value is 1, but if the field is repeated several times, the value is higher. value 0 indicates that the position is not important and is not taken into account during the rule selection process.

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

2.5. Direction Indicator

The Direction Indicator (DI) is used to tell if a field appears in both direction (Bi) or only uplink (Up) or Downlink (Dw).


```
identity direction-indicator-base-type {
      description "used to extend field length functions";
}

identity di-bidirectional {
         base direction-indicator-base-type;
         description "Direction Indication of bi directionality";
}
```

```
identity di-up {
         base direction-indicator-base-type;
         description "Direction Indication of upstream";
}

identity di-down {
         base direction-indicator-base-type;
         description "Direction Indication of downstream";
}
```

Figure 6: Definition of identityref for direction indicators

Figure 6 gives the identityref for Direction Indicators.

The type is "direction-indicator-type" (cf. Figure 7).

```
typedef direction-indicator-type {
    description "direction in LPWAN network, up when emitted by the device,
    down when received by the device, bi when emitted or received by the de
    type identityref {
        base direction-indicator-base-type;
    }
}
```

Figure 7: Type definition for direction indicators

2.6. Target Value

Target Value may be either a string or binary sequence. For match-mapping, several of these values can be contained in a Target Value field. In the data model, this is generalized by adding a position, which orders the list of values. By default the position is set to 0.

The leaf "value" is not mandatory to represent a non existing value in a TV.


```
grouping target-values-struct {
```

```
        description "defines the target value element. Can be either an arbitra
        binary or ascii element. All target values are considered as a matching
        Position is used to order values, by default position 0 is used when co
        a single element.";

        leaf value {
            type union {
                type binary;
                type string;
            }
        }
        leaf position {
            description "If only one element position is 0, otherwise position
            matching list.";
            type uint16;
        }
    }
```

Figure 8: Definition of target value

Figure 8 gives the definition of a single element of a Target Value. In the rule, this will be used as a list, with position as a key. The highest position value is used to compute the size of the index sent in residue.

2.7. Matching Operator

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

```
identity matching-operator-base-type {
   description "used to extend Matching Operators with SID values";
}

identity mo-equal {
   base matching-operator-base-type;
   description "RFC 8724";
}

identity mo-ignore {
   base matching-operator-base-type;
   description "RFC 8724";
}

identity mo-msb {
   base matching-operator-base-type;
   description "RFC 8724";
}

identity mo-matching {
   base matching-operator-base-type;
   description "RFC 8724";
}
```

Figure 9: Definition of identityref for Matching Operator

the type is "matching-operator-type" (cf. Figure 10)

```
typedef matching-operator-type {
    description "Matching Operator (MO) to compare fields values with targe
    type identityref {
        base matching-operator-base-type;
    }
}
```

Figure 10: Type definition for Matching Operator

2.7.1. Matching Operator arguments

Some Matching Operator such as MSB can take some values. Even if currently LSB is the only MO takes only one argument, in the future some MO may require several arguments. They are viewed as a list of target-values-type.

2.8. Compression Decompression Actions

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

```
        identity compression-decompression-action-base-type;

      identity cda-not-sent {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-value-sent {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-lsb {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-mapping-sent {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-compute-length {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }

      identity cda-compute-checksum {
          base compression-decompression-action-base-type;
              description "RFC 8724";
      }
```

```
    identity cda-deviid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

   identity cda-appiid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }
```

Figure 11: Definition of identityref for Compresion Decompression Action

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


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

Figure 12: Type definition for Compresion Decompression Action

2.8.1. Compression Decompression Action arguments

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

3. Rule definition

A rule is either a C/D or an F/R rule. A rule is identified by the rule ID value and its associated length. The YANG grouping rule-id-type defines the structure used to represent a rule ID. Length of 0 is allowed to represent an implicit rule.

```
// Define rule ID. Rule ID is composed of a RuleID value and a Rule ID Length

    grouping rule-id-type {
        leaf rule-id {
            type uint32;
            description "rule ID value, this value must be unique combined with
        }
        leaf rule-length {
            type uint8 {
                range 0..32;
            }
            description "rule ID length in bits, value 0 is for implicit rules"
        }
    }

// SCHC table for a specific device.

    container schc {
```

```
        leaf version{
            type uint64;
            mandatory false;
            description "used as an indication for versioning";
        }
        list rule {
            key "rule-id rule-length";
            uses rule-id-type;
            choice nature {
                case fragmentation {
                    uses fragmentation-content;
                }
                case compression {
                    uses compression-content;
                }
            }
        }
    }
```

Figure 13: Definition of a SCHC Context

To access to a specific rule, rule-id and its specific length is used as a key. The rule is either a compression or a fragmentation rule.

Each context can be identified though a version id.

3.1. Compression rule

A compression rule is composed of entries describing its processing (cf. Figure 14). An entry contains all the information defined in Figure 1 with the types defined above.

3.1.1. Compression context representation.

The compression rule described Figure 1 is associated to a rule ID. The compression rule entry is defined in Figure 14. Each column in the table is either represented by a leaf or a list. Note that

```
   Matching Operators and Compression Decompression actions can have
   arguments.  They are viewed a ordered list of strings and numbers as
   in target values.

    grouping compression-rule-entry {
        description "These entries defines a compression entry (i.e. a line)
        as defined in RFC 8724 and fragmentation parameters.

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

        An entry in a compression rule is composed of 7 elements:
        - Field ID: The header field to be compressed. The content is a YANG id
        - Field Length : either a positive integer of a function defined as a Y
        - 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 decompressi
        action, parameters may be associated to that action.
        ";

        leaf field-id {
            description "Field ID, identify a field in the header with a YANG i
            mandatory true;
            type schc:field-id-type;
        }
        leaf field-length {
            description "Field Length in bit or through a function defined as a
            mandatory true;
            type schc:field-length-type;
        }
        leaf field-position {
            description "field position in the header is a integer. If the fiel
            in the header the value is 1, and incremented for each repetition o
```



```
            0 means that the position is not important and order may change whe
            mandatory true;
            type uint8;
        }
```

```
        leaf direction-indicator {
            description "Direction Indicator, a YANG identityref to say if the
            up or down";
            mandatory true;
            type schc:direction-indicator-type;
        }
        list target-values {
            description "a list of value to compare with the header field value
            is a singleton, position must be 0. For matching-list, should be co
            values starting from 1.";
            key position;
            uses target-values-struct;
        }
        leaf matching-operator {
            mandatory true;
            type schc:matching-operator-type;
        }
        list matching-operator-value {
            key position;
            uses target-values-struct;
        }
        leaf comp-decomp-action {
            mandatory true;
            type schc:comp-decomp-action-type;
        }
        list comp-decomp-action-value {
            key position;
            uses target-values-struct;
        }
    }
```

Figure 14: Definition of a compression entry

3.1.2. Rule definition

A compression rule is a list of entries.

```
 grouping compression-content {
     description "define a compression rule composed of a list of entries.";
     list entry {
         key "field-id field-position direction-indicator";
         uses compression-rule-entry;
     }
 }
```

Figure 15: Definition of a compression rule

To identify a specific entry Field ID, position and direction are needed.

3.2. Fragmentation rule

Parameters for fragmentation are defined in Annex D of [RFC8724].

Figure 16 gives the first elements found in this structure. It starts with a direction. Since fragmentation rules work for a specific direction, they contain a mandatory direction. The type is the same as the one used in compression entries, but the use of bidirectionnal is forbidden.

The next elements describe size of SCHC fragmentation header fields. Only the FCN size is mandatory and value must be higher or equal to 1.

```
    grouping fragmentation-content {
        description "This grouping defines the fragmentation parameters for
        all the modes (No Ack, Ack Always and Ack on Error) specified in
        RFC 8724.";

        leaf direction {
            type schc:direction-indicator-type;
            description "should be up or down, bi directionnal is forbidden.";
            mandatory true;
        }
        leaf dtagsize {
            type uint8;
            description "size in bit of the DTag field";

        }
        leaf wsize {
            type uint8;
            description "size in bit of the window field";
        }
        leaf fcnsize {
            type uint8 {
                range 1..max;
            }
            description "size in bit of the FCN field";
            mandatory true;
        }
...
```

Figure 16: Definition of a fragmentation parameters, SCHC header

RCS algorithm is defined (Figure 17), by default with the CRC computation proposed in [RFC8724]. The algorithms are identified through an identityref specified in the SCHC Data Model and with the type RCS-algorithm-type (Figure 18).

```
...
        leaf RCS-algorithm {
            type RCS-algorithm-type;
            default schc:RFC8724-RCS;
            description "Algoritm used for RCS";
```

```
        }
...
```

Figure 17: Definition of a fragmentation parameters, RCS algorithm


```
    identity RCS-algorithm-base-type {
        description "identify which algorithm is used to compute RSC.
        The algorithm defines also the size if the RSC field.";
    }

    identity RFC8724-RCS {
        description "CRC 32 defined as default RCS in RFC8724.";
        base RCS-algorithm-base-type;
    }

    typedef RCS-algorithm-type {
        type identityref {
            base RCS-algorithm-base-type;
        }
    }
```

Figure 18: Definition of identityref for RCS Algorithm

Figure 19 gives the parameters used by the state machine to handle fragmentation:

- maximum-window-size contains the maximum FCN value that can be used.

- retransmission-timer gives in seconds the duration before sending an ack request (cf. section 8.2.2.4. of [RFC8724]). If specifed, value must be higher or equal to 1.

- inactivity-timer gives in seconds the duration before aborting (cf. section 8.2.2.4. of [RFC8724]), value of 0 explicitly indicates that this timer is disabled.

- max-ack-requests gives the number of attempts before aborting (cf.

section 8.2.2.4. of [RFC8724]).

- maximum-packet-size gives in bytes the larger packet size that can be reassembled.


```
...
        leaf maximum-window-size {
            type uint16;
            description "by default 2^wsize - 2";
        }

        leaf retransmission-timer {
            type uint64 {
                range 1..max;
            }
            description "duration in seconds of the retransmission timer"; // C
        }

        leaf inactivity-timer {
            type uint64;
            description "duration is seconds of the inactivity timer, 0 indicat
        }

        leaf max-ack-requests {
            type uint8 {
                range 1..max;
            }
            description "the maximum number of retries for a specific SCHC ACK.
        }

        leaf maximum-packet-size {
```

```
            type uint16;
            default 1280;
            description "When decompression is done, packet size must not stric
        }
...
```

Figure 19: Definition of a fragmentation state machine parameters

Figure 20 gives information related to a specific compression mode: fragmentation-mode MUST be set with a specific behavior. Identityref are given Figure 21.

For Ack on Error some specific information may be provided:

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

- tile-in-All1 indicates if All1 contains only the RCS (all1-data-no) or may contain a single tile (all1-data-yes). Since the reassembly process may detect this behavior, the choice can be left to the fragmentation process. In that case identityref all1-

data-sender-choice as to be specified. All possible values are given Figure 21.

- ack-behavior tells when the fragmentation process may send acknowledgments. When ack-behavior-after-All0 is specified, the ack may be sent after the reception of All-0 fragment. When ack-behavior-after-All1 is specified, the ack may be sent after the reception of All-1 fragment at the end of the fragmentation process. ack-behavior-always do not impose a limitation at the SCHC level. The constraint may come from the LPWAN technology. All possible values are given Figure 21.

```
...
        leaf fragmentation-mode {
            type schc:fragmentation-mode-type;
            description "which fragmentation mode is used (noAck, AckAlways, Ac
            mandatory true;
        }
```

```
        choice mode {
            case no-ack;
            case ack-always;
            case ack-on-error {
                leaf tile-size {
                    type uint8;
                    description "size in bit of tiles, if not specified or set
                }
                leaf tile-in-All1 {
                    type schc:all1-data-type;
                    description "When true, sender and receiver except a tile i
                }
                leaf ack-behavior {
                    type schc:ack-behavior-type;
                    description "Sender behavior to acknowledge, after All-0, A
                    LPWAN allows it (Always)";
                }
            }
        }
...
```

Figure 20: Definition of a fragmentation specific information

```
// -- FRAGMENTATION TYPE

// -- fragmentation modes

    identity fragmentation-mode-base-type {
        description "fragmentation mode";
```



```
    }

    identity fragmentation-mode-no-ack {
        description "No Ack of RFC 8724.";
        base fragmentation-mode-base-type;
    }

    identity fragmentation-mode-ack-always {
        description "Ack Always of RFC8724.";
        base fragmentation-mode-base-type;
    }
    identity fragmentation-mode-ack-on-error {
```

```
        description "Ack on Error of RFC8724.";
        base fragmentation-mode-base-type;
    }

    typedef fragmentation-mode-type {
        type identityref {
            base fragmentation-mode-base-type;
        }
    }

// -- Ack behavior

    identity ack-behavior-base-type {
        description "define when to send an Acknowledgment message";
    }

    identity ack-behavior-after-All0 {
        description "fragmentation expects Ack after sending All0 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-after-All1 {
        description "fragmentation expects Ack after sending All1 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-always {
        description "fragmentation expects Ack after sending every fragment.";
        base ack-behavior-base-type;
    }

    typedef ack-behavior-type {
        type identityref {
            base ack-behavior-base-type;
        }
    }
```



```
// -- All1 with data types

    identity all1-data-base-type {
        description "type to define when to send an Acknowledgment message";
    }
```

```
    identity all1-data-no {
        description "All1 contains no tiles.";
        base all1-data-base-type;
    }

    identity all1-data-yes {
        description "All1 MUST contain a tile";
        base all1-data-base-type;
    }

    identity all1-data-sender-choice {
        description "Fragmentation process choose to send tiles or not in all1.
        base all1-data-base-type;
    }

    typedef all1-data-type {
        type identityref {
            base all1-data-base-type;
        }
    }
```

Figure 21: Specific types for Ack On Error mode

YANG Tree

```
module: schc
  +--rw schc
     +--rw version?   uint64
     +--rw rule* [rule-id rule-length]
        +--rw rule-id                        uint32
        +--rw rule-length                    uint8
        +--rw (nature)?
           +--:(fragmentation)
           |  +--rw direction                schc:direction-indicator-type
           |  +--rw dtagsize?                uint8
           |  +--rw wsize?                   uint8
           |  +--rw fcnsize                  uint8
           |  +--rw RCS-algorithm?           RCS-algorithm-type
           |  +--rw maximum-window-size?     uint16
           |  +--rw retransmission-timer?    uint64
           |  +--rw inactivity-timer?        uint64
           |  +--rw max-ack-requests?        uint8
           |  +--rw maximum-packet-size?     uint16
           |  +--rw fragmentation-mode       schc:fragmentation-mode-type
           |  +--rw (mode)?
           |     +--:(no-ack)
           |     +--:(ack-always)
           |     +--:(ack-on-error)
           |        +--rw tile-size?         uint8
           |        +--rw tile-in-All1?      schc:all1-data-type
           |        +--rw ack-behavior?      schc:ack-behavior-type
           +--:(compression)
              +--rw entry* [field-id field-position direction-indicator]
                 +--rw field-id                     schc:field-id-type
                 +--rw field-length                 schc:field-length-type
                 +--rw field-position               uint8
                 +--rw direction-indicator          schc:direction-indicator-typ
                 +--rw target-values* [position]
                 |  +--rw value?      union
                 |  +--rw position    uint16
                 +--rw matching-operator            schc:matching-operator-type
                 +--rw matching-operator-value* [position]
                 |  +--rw value?      union
                 |  +--rw position    uint16
                 +--rw comp-decomp-action           schc:comp-decomp-action-type
                 +--rw comp-decomp-action-value* [position]
                    +--rw value?      union
                    +--rw position    uint16
```

Figure 22

4. IANA Considerations

This document has no request to IANA.

5. Security considerations

This document does not have any more Security consideration than the ones already raised on [RFC8724]

6. Acknowledgements

The authors would like to thank Dominique Barthel, Carsten Bormann, Alexander Pelov.

7. YANG Module

```
<code begins> file schc@2020-02-28.yang
module schc{
    yang-version "1";
    namespace "urn:ietf:lpwan:schc:rules-description";
    prefix "schc";

    description
    "Generic Data model for Static Context Header Compression Rule for SCHC,
    based on draft-ietf-lpwan-ipv6-static-context-hc-18. Include compression
    rules and fragmentation rules.

    This module is a YANG model for SCHC rules (RFc 8724).
    RFC 8724 describes a rule in a abstract way through a table.

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

```
    This module proposes a global data model that can be used for rule
    exchanges or modification. It proposes both the data model format and
    the global identifiers used to describes some operations in fields.
    This data model applies both to compression and fragmentation.";
```



```
    revision 2020-06-15 {
        description "clean up and add descriptions, merge schc-id to this file"
    }

    revision 2020-02-28 {
        description "Add Fragmentation parameters";
    }

    revision 2020-01-23 {
        description "Modified TV with binary and union";
    }

    revision 2020-01-07 {
        description "First version of the YANG model";
    }

// -------------------------
//  Field ID type definition
//-------------------------

// generic value TV definition

    identity field-id-base-type  {
           description "Field ID with SID";
    }

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

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

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

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



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

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

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

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

    identity fid-ipv6-devprefix {
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 8200. Depending if it is resp
                             a uplink or an downklink message.";
    }

    identity fid-ipv6-deviid {
```

```
        base field-id-base-type;
        description "correspond either to the source address or the desdination
                             address prefix of RFC 8200. Depending if it is resp
                             a uplink or an downklink message.";
    }

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

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



```
    identity fid-udp-dev-port {
        base field-id-base-type;
        description "UDP length from RFC 768";
    }

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

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

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

    identity fid-coap-version {
```

```
        base field-id-base-type;
        description "CoAP version from RFC 7252";
    }

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

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

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

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

    identity fid-coap-code-detail {
        base field-id-base-type;
        description "CoAP code detail from RFC 7252";
```



```
    }

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

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

    identity fid-coap-option-if-match {
        base field-id-base-type;
```

```
        description "CoAP option If-Match from RFC 7252";
    }

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

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

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

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

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

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

    identity fid-coap-option-uri-path {
```



```
        base field-id-base-type;
        description "CoAP option Uri-Path from RFC 7252";
    }

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

```
    }

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

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

    identity fid-coap-option-accept {
        base field-id-base-type;
        description "CoAP option Max-Age from RFC 7252";
    }

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

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

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

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

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

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

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

identity fid-coap-option-no-response {
    base field-id-base-type;
    description "CoAP option No response from RFC 7967";
}

identity fid-coap-option-oscore-flags {
    base field-id-base-type;
    description "CoAP option oscore flags (see draft schc coap, section 6.4
}

identity fid-coap-option-oscore-piv {
    base field-id-base-type;
    description "CoAP option oscore flags (see draft schc coap, section 6.4
}

identity fid-coap-option-oscore-kid {
    base field-id-base-type;
    description "CoAP option oscore flags (see draft schc coap, section 6.4
}

identity fid-coap-option-oscore-kidctx {
    base field-id-base-type;
    description "CoAP option oscore flags (see draft schc coap, section 6.4
}

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

identity fid-icmpv6-code {
    base field-id-base-type;
    description "ICMPv6 field (see draft OAM)";
}

identity fid-icmpv6-checksum {
    base field-id-base-type;
    description "ICMPv6 field (see draft OAM)";
```

```
    }

    identity fid-icmpv6-identifier {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }

    identity fid-icmpv6-sequence {
        base field-id-base-type;
        description "ICMPv6 field (see draft OAM)";
    }


/// !!!!!!! See future CoAP extentions

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

  identity field-length-base-type {
        description "used to extend field length functions";
  }

  identity fl-variable {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }

  identity fl-token-length {
           base field-length-base-type;
           description "residue length in Byte is sent";
  }

//---------------------------------
// Direction Indicator type
//---------------------------------

  identity direction-indicator-base-type {
        description "used to extend field length functions";
  }

  identity di-bidirectional {
           base direction-indicator-base-type;
```

```
            description "Direction Indication of bi directionality";
    }

    identity di-up {
            base direction-indicator-base-type;
```



```
            description "Direction Indication of upstream";
    }

    identity di-down {
            base direction-indicator-base-type;
            description "Direction Indication of downstream";
    }

//-----------------------------------
// Matching Operator type definition
//-----------------------------------

     identity matching-operator-base-type {
        description "used to extend Matching Operators with SID values";
     }

     identity mo-equal {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-ignore {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-msb {
        base matching-operator-base-type;
        description "RFC 8724";
     }

     identity mo-matching {
        base matching-operator-base-type;
        description "RFC 8724";
     }
```

```
//------------------------------
// CDA type definition
//------------------------------

    identity compression-decompression-action-base-type;

    identity cda-not-sent {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-value-sent {
```



```
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-lsb {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-mapping-sent {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-compute-length {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-compute-checksum {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

    identity cda-deviid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }
```

```
   identity cda-appiid {
        base compression-decompression-action-base-type;
            description "RFC 8724";
    }

// -- type definition

    typedef field-id-type {
        description "Field ID generic type.";
        type identityref {
            base field-id-base-type;
        }
    }

    typedef field-length-type {
        description "Field length either a positive integer giving the size in
        or a function defined through an identityref.";
        type union {
            type int64; /* positive length in bits */
```



```
            type identityref { /* function */
                base field-length-base-type;
            }
        }
    }

    typedef direction-indicator-type {
        description "direction in LPWAN network, up when emitted by the device,
        down when received by the device, bi when emitted or received by the de
        type identityref {
            base direction-indicator-base-type;
        }
    }

    typedef matching-operator-type {
        description "Matching Operator (MO) to compare fields values with targe
        type identityref {
            base matching-operator-base-type;
        }
    }

    typedef comp-decomp-action-type {
```

```
        description "Compression Decompression Action to compression or decompr
        type identityref {
            base compression-decompression-action-base-type;
        }
    }

// -- FRAGMENTATION TYPE

// -- fragmentation modes

    identity fragmentation-mode-base-type {
        description "fragmentation mode";
    }

    identity fragmentation-mode-no-ack {
        description "No Ack of RFC 8724.";
        base fragmentation-mode-base-type;
    }

    identity fragmentation-mode-ack-always {
        description "Ack Always of RFC8724.";
        base fragmentation-mode-base-type;
    }
    identity fragmentation-mode-ack-on-error {
        description "Ack on Error of RFC8724.";
        base fragmentation-mode-base-type;
```



```
    }

    typedef fragmentation-mode-type {
        type identityref {
            base fragmentation-mode-base-type;
        }
    }

// -- Ack behavior

    identity ack-behavior-base-type {
        description "define when to send an Acknowledgment message";
    }

    identity ack-behavior-after-All0 {
```

```
        description "fragmentation expects Ack after sending All0 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-after-All1 {
        description "fragmentation expects Ack after sending All1 fragment.";
        base ack-behavior-base-type;
    }

    identity ack-behavior-always {
        description "fragmentation expects Ack after sending every fragment.";
        base ack-behavior-base-type;
    }

    typedef ack-behavior-type {
        type identityref {
            base ack-behavior-base-type;
        }
    }

// -- All1 with data types

    identity all1-data-base-type {
        description "type to define when to send an Acknowledgment message";
    }

    identity all1-data-no {
        description "All1 contains no tiles.";
        base all1-data-base-type;
    }

    identity all1-data-yes {
        description "All1 MUST contain a tile";
```



```
        base all1-data-base-type;
    }

    identity all1-data-sender-choice {
        description "Fragmentation process choose to send tiles or not in all1.
        base all1-data-base-type;
    }
```

```
    typedef all1-data-type {
        type identityref {
            base all1-data-base-type;
        }
    }

// -- RCS algorithm types

    identity RCS-algorithm-base-type {
        description "identify which algorithm is used to compute RSC.
        The algorithm defines also the size if the RSC field.";
    }

    identity RFC8724-RCS {
        description "CRC 32 defined as default RCS in RFC8724.";
        base RCS-algorithm-base-type;
    }

    typedef RCS-algorithm-type {
        type identityref {
            base RCS-algorithm-base-type;
        }
    }



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

    grouping target-values-struct {
        description "defines the target value element. Can be either an arbitra
        binary or ascii element. All target values are considered as a matching
        Position is used to order values, by default position 0 is used when co
        a single element.";

        leaf value {
            type union {
                type binary;
                type string;
            }
        }
```



```
        leaf position {
```

```
            description "If only one element position is 0, otherwise position
            matching list.";
            type uint16;
        }
    }

    grouping compression-rule-entry {
        description "These entries defines a compression entry (i.e. a line)
        as defined in RFC 8724 and fragmentation parameters.

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

        An entry in a compression rule is composed of 7 elements:
        - Field ID: The header field to be compressed. The content is a YANG id
        - Field Length : either a positive integer of a function defined as a Y
        - 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, paramters may be
        associated to that operator.
        - Comp./Decomp. Action: A YANG id giving the compression or decompressi
        action, paramters may be associated to that action.
        ";

        leaf field-id {
            description "Field ID, identify a field in the header with a YANG r
            mandatory true;
            type schc:field-id-type;
        }
        leaf field-length {
            description "Field Length in bit or through a function defined as a
            mandatory true;
            type schc:field-length-type;
        }
        leaf field-position {
            description "field position in the header is a integer. If the fiel
            in the header the value is 1, and incremented for each repetition o
            0 means that the position is not important and order may change whe
            mandatory true;
            type uint8;
        }
        leaf direction-indicator {
            description "Direction Indicator, a YANG referenceid to say if the
            up or down";
            mandatory true;
```

```
            type schc:direction-indicator-type;
        }
        list target-values {
            description "a list of value to compare with the header field value
            is a singleton, position must be 0. For matching-list, should be co
            values starting from 1.";
            key position;
            uses target-values-struct;
        }
        leaf matching-operator {
            mandatory true;
            type schc:matching-operator-type;
        }
        list matching-operator-value {
            key position;
            uses target-values-struct;
        }
        leaf comp-decomp-action {
            mandatory true;
            type schc:comp-decomp-action-type;
        }
        list comp-decomp-action-value {
            key position;
            uses target-values-struct;
        }
    }

    grouping compression-content {
        description "define a compression rule composed of a list of entries.";
        list entry {
            key "field-id field-position direction-indicator";
            uses compression-rule-entry;
        }
    }

    grouping fragmentation-content {
        description "This grouping defines the fragmentation parameters for
        all the modes (No Ack, Ack Always and Ack on Error) specified in
        RFC 8724.";

        leaf direction {
            type schc:direction-indicator-type;
            description "should be up or down, bi directionnal is forbiden.";
            mandatory true;
        }
        leaf dtagsize {
```

```
            type uint8;
            description "size in bit of the DTag field";
```



```
        }
        leaf wsize {
            type uint8;
            description "size in bit of the window field";
        }
        leaf fcnsize {
            type uint8;
            description "size in bit of the FCN field";
            mandatory true;
        }
        leaf RCS-algorithm {
            type RCS-algorithm-type;
            default schc:RFC8724-RCS;
            description "Algoritm used for RCS";
        }
        leaf maximum-window-size {
            type uint16;
            description "by default 2^wsize - 1";
        }

        leaf retransmission-timer {
            type uint64 {
                range 1..max;
            }
            description "duration in seconds of the retransmission timer"; // C
        }

        leaf inactivity-timer {
            type uint64;
            description "duration is seconds of the inactivity timer, 0 indicat
        }

        leaf max-ack-requests {
            type uint8 {
                range 1..max;
            }
            description "the maximum number of retries for a specific SCHC ACK.
        }
```

```
        leaf maximum-packet-size {
            type uint16;
            default 1280;
            description "When decompression is done, packet size must not stric
        }

        leaf fragmentation-mode {
            type schc:fragmentation-mode-type;
            description "which fragmentation mode is used (noAck, AckAlways, Ac
```



```
            mandatory true;
        }

        choice mode {
            case no-ack;
            case ack-always;
            case ack-on-error {
                leaf tile-size {
                    type uint8;
                    description "size in bit of tiles, if not specified or set
                }
                leaf tile-in-All1 {
                    type schc:all1-data-type;
                    description "When true, sender and receiver except a tile i
                }
                leaf ack-behavior {
                    type schc:ack-behavior-type;
                    description "Sender behavior to acknowledge, after All-0, A
                    LPWAN allows it (Always)";
                }
            }
       }
    }


// Define rule ID. Rule ID is composed of a RuleID value and a Rule ID Length

    grouping rule-id-type {
        leaf rule-id {
            type uint32;
            description "rule ID value, this value must be unique combined with
```

```
        }
        leaf rule-length {
            type uint8 {
                range 0..32;
            }
            description "rule ID length in bits, value 0 is for implicit rules"
        }
    }

// SCHC table for a specific device.

    container schc {
        leaf version{
            type uint64;
            mandatory false;
            description "used as an indication for versioning";
```



```
        }
        list rule {
            key "rule-id rule-length";
            uses rule-id-type;
            choice nature {
                case fragmentation {
                    uses fragmentation-content;
                }
                case compression {
                    uses compression-content;
                }
            }
        }
    }


}
```

<code ends>

Figure 23

8. Normative References

[I-D.barthel-lpwan-oam-schc]

Barthel, D., Toutain, L., Kandasamy, A., Dujovne, D., and J. Zuniga, "OAM for LPWAN using Static Context Header Compression (SCHC)", draft-barthel-lpwan-oam-schc-02 (work in progress), November 2020.

[I-D.ietf-lpwan-coap-static-context-hc]
Minaburo, A., Toutain, L., and R. Andreasen, "LPWAN Static Context Header Compression (SCHC) for CoAP", draft-ietf-lpwan-coap-static-context-hc-18 (work in progress), January 2021.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

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

Authors' Addresses

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

Email: ana@ackl.io

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

Email: Laurent.Toutain@imt-atlantique.fr