

lpwan Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: 7 November 2022

A. Minaburo  
Acklio  
L. Toutain  
Institut MINES TELECOM; IMT Atlantique  
6 May 2022

Data Model for Static Context Header Compression (SCHC)  
draft-ietf-lpwan-schc-yang-data-model-08

## 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 7 November 2022.

## Copyright Notice

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

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

## Table of Contents

<a href="#">1.</a>	<a href="#">Introduction . . . . .</a>	<a href="#">2</a>
<a href="#">2.</a>	<a href="#">SCHC rules . . . . .</a>	<a href="#">3</a>
<a href="#">2.1.</a>	<a href="#">Compression Rules . . . . .</a>	<a href="#">4</a>
<a href="#">2.2.</a>	<a href="#">Identifier generation . . . . .</a>	<a href="#">4</a>
<a href="#">2.3.</a>	<a href="#">Field Identifier . . . . .</a>	<a href="#">5</a>
<a href="#">2.4.</a>	<a href="#">Field length . . . . .</a>	<a href="#">7</a>
<a href="#">2.5.</a>	<a href="#">Field position . . . . .</a>	<a href="#">8</a>
<a href="#">2.6.</a>	<a href="#">Direction Indicator . . . . .</a>	<a href="#">8</a>
<a href="#">2.7.</a>	<a href="#">Target Value . . . . .</a>	<a href="#">10</a>
<a href="#">2.8.</a>	<a href="#">Matching Operator . . . . .</a>	<a href="#">10</a>
<a href="#">2.8.1.</a>	<a href="#">Matching Operator arguments . . . . .</a>	<a href="#">12</a>
<a href="#">2.9.</a>	<a href="#">Compression Decompression Actions . . . . .</a>	<a href="#">12</a>
<a href="#">2.9.1.</a>	<a href="#">Compression Decompression Action arguments . . . . .</a>	<a href="#">13</a>
<a href="#">2.10.</a>	<a href="#">Fragmentation rule . . . . .</a>	<a href="#">13</a>
<a href="#">2.10.1.</a>	<a href="#">Fragmentation mode . . . . .</a>	<a href="#">13</a>
<a href="#">2.10.2.</a>	<a href="#">Fragmentation Header . . . . .</a>	<a href="#">14</a>
<a href="#">2.10.3.</a>	<a href="#">Last fragment format . . . . .</a>	<a href="#">15</a>
<a href="#">2.10.4.</a>	<a href="#">Acknowledgment behavior . . . . .</a>	<a href="#">17</a>
<a href="#">2.10.5.</a>	<a href="#">Fragmentation Parameters . . . . .</a>	<a href="#">18</a>
<a href="#">2.10.6.</a>	<a href="#">Layer 2 parameters . . . . .</a>	<a href="#">19</a>
<a href="#">3.</a>	<a href="#">Rule definition . . . . .</a>	<a href="#">19</a>
<a href="#">3.1.</a>	<a href="#">Compression rule . . . . .</a>	<a href="#">21</a>
<a href="#">3.2.</a>	<a href="#">Fragmentation rule . . . . .</a>	<a href="#">24</a>
<a href="#">3.3.</a>	<a href="#">YANG Tree . . . . .</a>	<a href="#">27</a>
<a href="#">4.</a>	<a href="#">IANA Considerations . . . . .</a>	<a href="#">29</a>
<a href="#">5.</a>	<a href="#">Security considerations . . . . .</a>	<a href="#">29</a>
<a href="#">6.</a>	<a href="#">Acknowledgements . . . . .</a>	<a href="#">29</a>
<a href="#">7.</a>	<a href="#">YANG Module . . . . .</a>	<a href="#">29</a>
<a href="#">8.</a>	<a href="#">Normative References . . . . .</a>	<a href="#">51</a>
	<a href="#">Authors' Addresses . . . . .</a>	<a href="#">51</a>

[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,...)

- \* ...

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.

## [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 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,...)
- \* ...

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, UDP, CoAP and OSCORE.

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

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

```

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

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

```

Figure 1: Feature for compression and fragmentation.

## 2.1. 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

+-----+--+--+--+-----+-----+-----+-----+		
...  .. .. ..  ...   ...   ...		
+-----+--+--+--+-----+-----+-----+-----+		
Field N FL FP DI Target Value Matching Operator Comp/Decomp Act		
+-----+--+--+--+-----+-----+-----+-----+		
+-----+-----+-----+-----+		

Figure 2: Compression Decompression Context

## 2.2. 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 3) shows how an identityref is created for RCS algorithms used during SCHC fragmentation.

```
// -- 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 3: Principle to define a type based on identityref.

### 2.3. 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 model, each field MUST be identified through a global YANG identityref. A YANG field ID for the protocol always derives 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 7](#). The example Figure 4 gives the first field ID definitions. 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.

```

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";
}

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";
}
...

```

Figure 4: Definition of identityref for field IDs

The type associated to this identity is fid-type (cf. Figure 5)

```

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

```

Figure 5: Type definition for field IDs

#### [2.4.](#) 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.

```
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).";
}
```

Figure 6: Definition of identityref for Field Length

The field length function can be defined as an identityref as shown in Figure 6.

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 7).

```
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.";
}

```

Figure 7: Type definition for field Length

### [2.5.](#) 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.

### [2.6.](#) 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 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
    RFC 8724 (cf. 7.1).";
}
```

Figure 8: Definition of identityref for direction indicators

Figure 8 gives the identityref for Direction Indicators. The naming convention is "di" followed by the Direction Indicator name.

The type is "di-type" (cf. Figure 9).

```
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.";
}
```

Figure 9: Type definition for direction indicators

## [2.7.](#) Target Value

The Target Value is a list of binary sequences of any length, aligned to the left. Figure 10 shows the definition of a single element of a Target Value. In the rule, the structure 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 for the match-mapping CDA. The position allows to specify several values:

- \* For Equal and LSB, Target Value contains a single element. Therefore, the position is set to 0.
- \* For match-mapping, Target Value can contain several elements. Position values must start from 1 and MUST be contiguous.

```
grouping tv-struct {
  description
    "Defines the target value element. Always a binary type, strings
    must be converted to binary. field-id allows the conversion
    to the appropriate type.";
  leaf value {
    type binary;
    description
      "Target Value";
  }
  leaf position {
    type uint16;
    description
      "If only one element, position is 0. Otherwise, position is the
      the order in the matching list, starting at 1.";
  }
}
```

Figure 10: Definition of target value

## [2.8.](#) 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 as listed in Figure 11.

```
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)";
}
```

Figure 11: Definition of identityref for Matching Operator

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

The type is "mo-type" (cf. Figure 12)

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

Figure 12: Type definition for Matching Operator

### [2.8.1.](#) Matching Operator arguments

They are viewed as a list, built with a tv-struct (see chapter [Section 2.7](#)).

## [2.9.](#) Compression Decompression Actions

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

Figure 14 shows some CDA definition, the full definition is in [Section 7](#).

```
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)";
}
....

```

Figure 13: Definition of identityref for Compression Decompression Action

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

```

typedef cda-type {
    type identityref {
        base cda-base-type;
    }
    description
        "Compression Decompression Action to compression or
        decompress a field.";
}

```

Figure 14: Type definition for Compression Decompression Action

### [2.9.1.](#) 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.

## [2.10.](#) 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.

### [2.10.1.](#) Fragmentation mode

[\[RFC8724\]](#) defines 3 fragmentation modes:

- \* No Ack: this mode is unidirectionnal, 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.

Figure 15 shows the definition for identifiers from these three modes.

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

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
        "ACK-Always of RFC8724.";
}

identity fragmentation-mode-ack-on-error {
    base fragmentation-mode-base-type;
    description
        "ACK-on-Error of RFC8724.";
}

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

```

Figure 15: Definition of fragmentation mode identifier

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

#### [2.10.2.](#) Fragmentation Header

A data fragment header, starting with the rule ID can be sent on the fragmentation direction. The SCHC header may be composed of (cf. Figure 16):

- \* 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 on 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 16: Data fragment header from [RFC8724](#)

### [2.10.3](#). 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 identity of the RCS algorithm is shown in Figure 17.

```
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 17: type definition for RCS

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 defined in Figure 18 allows to define the behavior:

- \* all1-data-no: the last fragment contains no data, just the RCS
- \* all1-data-yes: the last fragment includes a single tile and the RCS
- \* all1-data-sender-choice: the last fragment may or may not contain a single tile. The receiver can detect if a tile is present.

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

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

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

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

typedef all1-data-type {
  type identityref {
    base all1-data-base-type;
  }
  description
    "Type used in rules.";
}
```

Figure 18: type definition for RCS

The naming convention is "all1-data" followed by the behavior identifier.

#### [2.10.4.](#) Acknowledgment behavior

The acknowledgment fragment header goes in the opposite direction of data. The header is composed of (see Figure 19):

- \* 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 19: 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 following identifiers (cf. Figure 20) define the acknowledgment behavior.

```
identity ack-behavior-base-type {
  description
    "Define when to send an Acknowledgment .";
}

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

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

identity ack-behavior-by-layer2 {
  base ack-behavior-base-type;
  description
    "Layer 2 defines when to send an Ack.";
}

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

Figure 20: bitmap generation behavior

The naming convention is "ack-behavior" followed by the algorithm name.

#### [2.10.5.](#) Fragmentation Parameters

The state machine requires some common values to handle fragmentation:

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

- \* inactivity-timer expresses, in seconds, 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.
- \* max-ack-requests expresses the number of attempts before aborting (cf. [section 8.2.2.4. of \[RFC8724\]](#)).
- \* maximum-packet-size reexpresses, in bytes, the larger packet size that can be reassembled.

They are defined as unsigned integers, see [Section 7](#).

#### [2.10.6.](#) Layer 2 parameters

The data model includes two parameters needed for fragmentation:

- \* l2-word-size: [\[RFC8724\]](#) base fragmentation 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 7](#).

### 3. 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.

```
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.";
}
```

```
// 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 {
        description
          "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.";
  }
}
```

Figure 21: Definition of a SCHC Context

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.

### [3.1.](#) Compression rule

A compression rule is composed of entries describing its processing

(cf. Figure 22). An entry contains all the information defined in Figure 2 with the types defined above.

The compression rule described Figure 2 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 2. Their type reflects the identifier types defined in [Section 2.1](#)

Some checks are performed on the values:

- \* target value must be present for MO different from ignore.
- \* when MSB MO is specified, the matching-operator-value must be present

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 or 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 define
        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 "position";
    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-value" {
    error-message "mo-msb requires length value";
  }
  mandatory true;
  description
    "MO: Matching Operator";
}
list matching-operator-value {
  key "position";
  uses tv-struct;
  description
    "Matching Operator Arguments, based on TV structure to allow
      several arguments.
      In RFC 8724, only the MSB matching operator needs arguments (a single
        number of most significant bits to be matched)";
}
leaf comp-decomp-action {
  type schc:cda-type;
  mandatory true;
  description
    "CDA: Compression Decompression Action.";
}
list comp-decomp-action-value {
  key "position";
  uses tv-struct;
  description
    "CDA arguments, based on a TV structure, in order to allow for
      several arguments. The CDAs specified in RFC 8724 require no
        argument.";
}
}

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.";
}
```

Figure 22: Definition of a compression entry

### [3.2.](#) Fragmentation rule

A Fragmentation rule is composed of entries describing the protocol behavior. Some on them are numerical entries, others are identifiers defined in [Section 2.10](#).

The definition of a Fragmentation rule is divided into three sub-parts:

- \* parameters such as the fragmentation-mode, the l2-word-size and the direction. Since Fragmentation rules are always defined for a specific direction, the value must be either di-up or di-down (di-bidirectional is not allowed).
- \* parameters defining the Fragmentation header format (dtag-size, w-size, fcn-size and rcs-algorithm).
- \* Protocol parameters for timers (inactivity-timer, retransmission-timer) or behavior (maximum-packet-size, max-interleaved-frames, max-ack-requests). If these parameters are specific to a single fragmentation mode, they are grouped in a structure dedicated to that Fragmentation mode. If some parameters can be found in several modes, typically ACK-Always and ACK-on-Error, they are defined in a common part and a when statement indicates which modes are allowed.

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

```

    description
        "which fragmentation mode is used (noAck, AckAlways,
        AckonError)";
}
leaf l2-word-size {
    type uint8;

```

```

    default "8";
    description
        "Size, in bits, of the layer 2 word";
}
leaf direction {
    type schc:di-type;
    must "derived-from-or-self(., 'di-up') or
        derived-from-or-self(., 'di-down')" {
        error-message
            "direction for fragmentation rules are up or down.";
    }
    mandatory true;
    description
        "Should be up or down, bidirectionnal is forbidden.";
}
// 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.";
}
leaf inactivity-timer {
    type uint64;
    description
        "Duration is seconds of the inactivity timer, 0 indicates
        that the timer is disabled.";
}
leaf retransmission-timer {
    when "derived-from(..fragmentation-mode,
        'fragmentation-mode-ack-on-error')";
}

```

```

        or
        derived-from(..fragmentation-mode,
                    'fragmentation-mode-ack-always') ";
    type uint64 {
        range "1..max";
    }
    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-All1 {
            when "derived-from(..fragmentation-mode,
                            'fragmentation-mode-ack-on-error')";
            type schc:all1-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.";
}
}

```

### [3.3.](#) 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?         uint64
|   +--rw retransmission-timer?     uint64
|   +--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
+--:(compression) {compression}?
|   +--rw entry* [field-id field-position direction-indicator]
|       +--rw field-id              schc:fid-type
|       +--rw field-length          schc:fl-type
|       +--rw field-position        uint8
|       +--rw direction-indicator   schc:di-type
|       +--rw target-value* [position]
|           | +--rw value?          binary
|           | +--rw position       uint16
|       +--rw matching-operator      schc:mo-type
|       +--rw matching-operator-value* [position]
|           | +--rw value?          binary
|           | +--rw position       uint16
|       +--rw comp-decomp-action      schc:cda-type
|       +--rw comp-decomp-action-value* [position]
|           +--rw value?            binary
|           +--rw position         uint16
+--:(no-compression)

```

Figure 23

#### [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 in [RFC8724] and [RFC8824].

## 6. Acknowledgements

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

## 7. YANG Module

```
<code begins> file ietf-schc@2022-02-15.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
    "
    Copyright (c) 2021 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

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

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

    The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
```

NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in [BCP 14](#) ([RFC 2119](#)) ([RFC 8174](#)) when, and only when, they appear in all capitals, as shown here.

\*\*\*\*\*

Generic Data model for Static Context Header Compression Rule for SCHC, based on [RFC 8724](#) and [RFC8824](#). Include compression, no compression and fragmentation rules.

This module is a YANG model for SCHC rules ([RFC 8724](#) and [RFC8824](#)). [RFC 8724](#) describes compression rules in a abstract way through a table.

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

This module 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 2022-02-15 {
  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";
  }

  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-payloadlength {  
    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  
        an uplink or a downlink 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";
}

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";
}

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";
}

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

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

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

identity fid-coap-option-oscore-kidctx {
    base fid-coap-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).";
}

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

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

identity di-bidirectional {
  base di-base-type;
  description
    "Direction Indication of bidirectionality 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
```

```
    RFC 8724 (cf. 7.1).";  
}
```

```
//-----  
// Matching Operator type definition  
//-----  
  
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)";  
}  
  
//-----  
// CDA type definition  
//-----  
  
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-length 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 {
    type identityref {
        base fid-base-type;
    }
    description
        "Field ID generic 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.";
}

```

```

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.";
}

```

```

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

```

```

        "Matching Operator (MO) to compare fields values with
        target values";
    }

typedef cda-type {
    type identityref {
        base cda-base-type;
    }
    description
        "Compression Decompression Action to compression or
        decompress a field.";
}

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

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

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
        "ACK-Always of RFC8724.";
}

identity fragmentation-mode-ack-on-error {
    base fragmentation-mode-base-type;
    description
        "ACK-on-Error of RFC8724.";
}

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 .";
}

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

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

identity ack-behavior-by-layer2 {
    base ack-behavior-base-type;
    description
        "Layer 2 defines when to send an Ack.";
}

```

```

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

// -- All1 with data types

identity all1-data-base-type {

```



```

    description
        "Type to define when to send an Acknowledgment message.";
}

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

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

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

typedef all1-data-type {
    type identityref {
        base all1-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.";
}

```

```

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.";
}

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

grouping tv-struct {
    description
        "Defines the target value element. Always a binary type, strings
        must be converted to binary. field-id allows the conversion
        to the appropriate type.";
    leaf value {
        type binary;
        description
            "Target Value";
    }
    leaf position {
        type uint16;
        description
            "If only one element, position is 0. Otherwise, position is the
            the order in the matching list, starting at 1.";
    }
}

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 or 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 define
      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 "position";
  uses tv-struct;
```

```
    "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-value" {
        error-message "mo-msb requires length value";
    }
    mandatory true;
    description
        "MO: Matching Operator";
}
list matching-operator-value {
    key "position";
    uses tv-struct;
    description
        "Matching Operator Arguments, based on TV structure to allow
        several arguments.
        In RFC 8724, only the MSB matching operator needs arguments (a single
        number of most significant bits to be matched)";
}
leaf comp-decomp-action {
    type schc:cda-type;
    mandatory true;
    description
        "CDA: Compression Decompression Action.";
}
list comp-decomp-action-value {
    key "position";
    uses tv-struct;
```

```

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

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.";
}

grouping fragmentation-content {
  description
    "This grouping defines the fragmentation parameters for
      all the modes (No-Ack, Ack-Always and Ack-on-Error) specified in
      RFC 8724.";
  leaf fragmentation-mode {
    type schc:fragmentation-mode-type;
    mandatory true;
    description
      "which fragmentation mode is used (noAck, AckAlways,
        AckonError)";
  }
  leaf l2-word-size {
    type uint8;
    default "8";
    description
      "Size, in bits, of the layer 2 word";
  }
  leaf direction {
    type schc:di-type;
    must "derived-from-or-self(., 'di-up') or

```

```

        derived-from-or-self(., 'di-down')" {
    error-message
        "direction for fragmentation rules are up or down.";
    }
    mandatory true;
    description
        "Should be up or down, bidirectionnal is forbidden.";
    }
// 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  $2^w\text{-size} - 1$ . 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
         $2^{\text{dtag-size}}$ . All DTAG values can be used, but at most
        max-interleaved-frames must be active at any time.";
  }
  leaf inactivity-timer {
    type uint64;
    description

```

```

      "Duration is seconds of the inactivity timer, 0 indicates
        that the timer is disabled.";
  }
  leaf retransmission-timer {
    when "derived-from(..fragmentation-mode,
                                'fragmentation-mode-ack-on-error')
        or
        derived-from(..fragmentation-mode,
                                'fragmentation-mode-ack-always') ";
    type uint64 {
      range "1..max";
    }
    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-All1 {
            when "derived-from(..fragmentation-mode,
                            'fragmentation-mode-ack-on-error')";
            type schc:all1-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.";
}

// 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 {
        description

```

```

        "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.";
    }
}
<code ends>

```

Figure 24

## 8. Normative References

- [RFC8724] Minaburo, A., Toutain, L., Gomez, C., Barthel, D., and JC. Zuniga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation", [RFC 8724](#), DOI 10.17487/RFC8724, April 2020, <<https://www.rfc-editor.org/info/rfc8724>>.
- [RFC8824] Minaburo, A., Toutain, L., and R. Andreasen, "Static Context Header Compression (SCHC) for the Constrained Application Protocol (CoAP)", [RFC 8824](#), DOI 10.17487/RFC8824, June 2021, <<https://www.rfc-editor.org/info/rfc8824>>.

## 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](mailto:Laurent.Toutain@imt-atlantique.fr)

