SCHC Working Group
Internet-Draft
Intended status: Informational
Expires: 13 March 2026

R. Munoz-Lara
Universidad de Chile
S. Cespedes
Concordia University

9 September 2025

Static Context Header Compression and Fragmentation over Direct-to-Satellite IoT
draft-munoz-schc-over-dts-iot-00

Abstract

This document uses the Static Context Header Compression and Fragmentation (SCHC) standard in prone to disruptions communications such as Direct-to-Satellite Internet of Things (DtS-IoT) communications or any scenario where communications are interrupted or experience significant delays. To this end, the document defines a new fragmentation sublayer for communications with prone to disruptions and delay. The new fragmentation sublayer optimizes the transfer delay of a SCHC packet through a Type II Hybrid ARQ/FEC mechanism.

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 13 March 2026.

Copyright Notice

Table of Contents

1. Introduction

Direct-to-satellite IoT (DtS-IoT) enables end devices to communicate directly with satellites in an IoT network without relying on ground-based Low Power Wide Area Network (LPWAN) gateways like LoRaWAN radio gateways or NB-IoT eNodeBs. In this setup, the satellite itself incorporates the LPWAN gateway functionality. When DtS-IoT relies on Low Earth Orbit (LEO) satellites or a limited LEO constellation, connectivity between the end devices and the satellite may experience interruptions. The interruptions between the ground nodes (end device and ground station) and the LEO satellite occur due to the fast speed and short line-of-sight duration between the satellite and

ground stations. End devices and LEO satellites use a store-and-forward mechanism to handle these disruptions. On the uplink, if an end device is outside satellite coverage, the end device temporarily stores messages in a queue until the satellite becomes visible. Similarly, once the satellite receives a message from an end device, it holds until it can establish a link with the ground station.

Using SCHC in a DtS-IoT context allows end devices to leverage the widespread coverage of satellite communications while enabling their applications to connect directly with TCP/IP networks. However, SCHC was not originally intended for delay- or disruption-tolerant environments. It supports fragmentation in cases where compressed SCHC packets still exceed the MTU size. In its reliable mode, SCHC includes mechanisms to acknowledge the successful receipt of each transmitted fragment. The effect of communication disruptions on these acknowledgment mechanisms remains uncertain. One possible outcome is the expiration of the SCHC timers, involving the unexpected termination of the SCHC session, and inefficiency use in the transmission time to the satellite, which involves greater energy consumption in the constrained nodes.

2. SCHC over DtS-IoT

The following sections show the current behavior of the SCHC standard in a Direct-to-Satellite IoT scenario. They define the concepts of visibility windows, revisit (pass-to-pass), and transfer delay.

2.1. Visibility and Pass-to-pass windows

From the point of view of the ground nodes (end device and ground station), there are two-period windows associated with interrupts.

* Visibility window (visibility period): the period in which a device on the ground can communicate with a satellite.

* Pass-to-pass window (revisit period): is the time between the end of a visibility window (i-th pass) and the beginning of the next visibility window ((i+1)th pass) for the same device on the ground. During this time, the ground device cannot communicate with the satellite. Figure 1 shows the message flow of SCHC to send a SCHC window with its corresponding acknowledgement through a DtS-IoT environment.

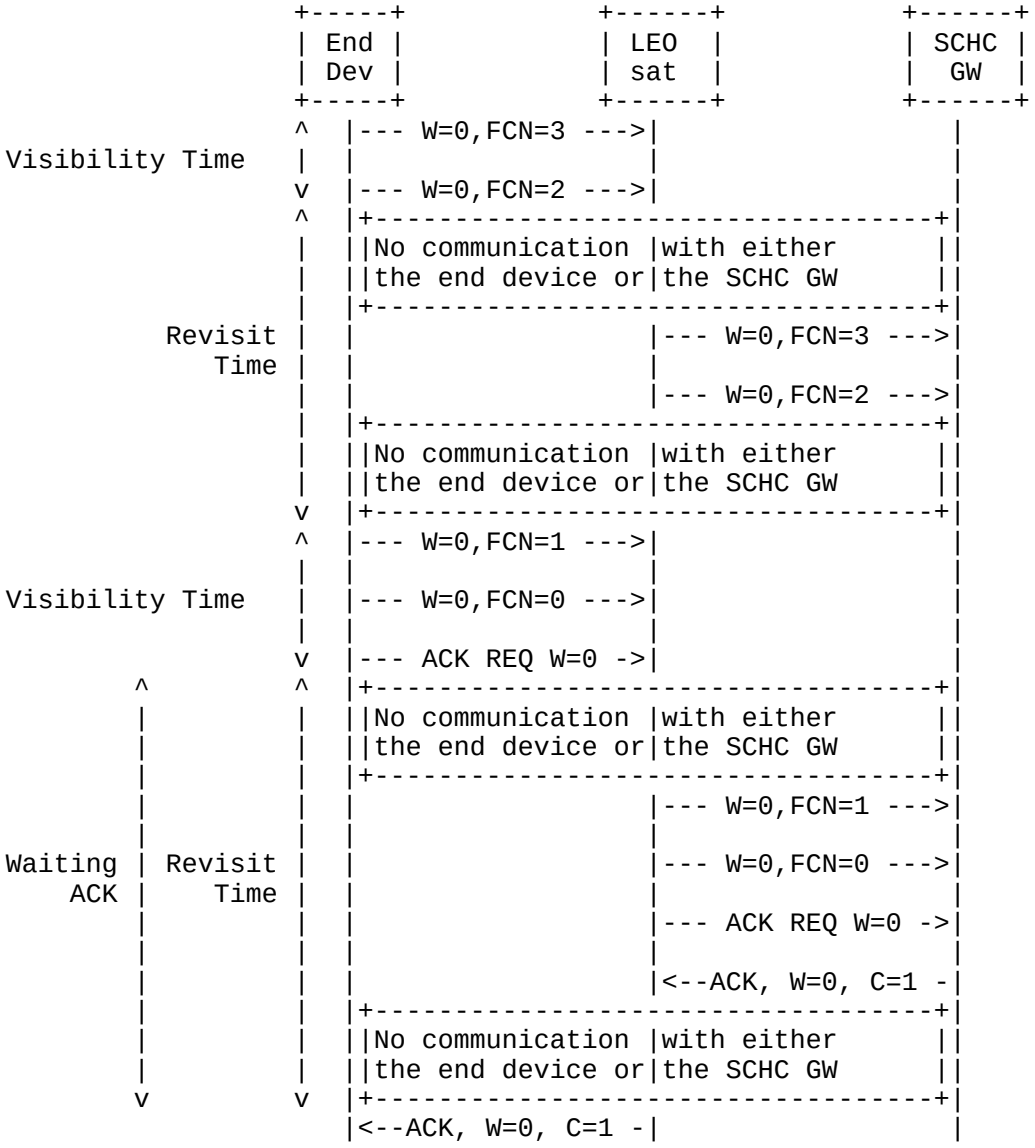


Figure 1: Message flow for a SCHC session inDtS-IoT.

2.2. SCHC packet transfer delay

The SCHC packet transfer delay is the time between the first SCHC fragment message sent by the fragmenter and the last SCHC ACK with successful acknowledgement received by the fragmenter.

In SCHC fragmentation modes with reliability (ACK-on-error and ACK-Always modes), each SCHC window MUST be acknowledged when there are errors in the reception of tiles. Due to the asynchronous communication provided by the visibility windows and the pass-to-pass window, the fragmenter must wait for the reception of a SCHC ACK. This period is known as waiting ACK. The SCHC packet transfer delay is directly proportional to waiting for ACK. Furthermore, if there is fragment loss, each retransmission window must be confirmed with a SCHC ACK, further increasing the SCHC packet transfer delay.

In this scenario, forward error correction (FEC), normally applied below the link layer, becomes important. Although it is not frequently used today in the transport layer on the Internet, FEC or erasure coding could be useful in future interplanetary transport protocols, as it would facilitate the recovery of lost data without requiring retransmissions or a permanent two-way connection.

3. SCHC ARQ-FEC fragmentation sublayer

The proposed new fragmentation sublayer uses channel coding to reduce the transfer delay of a SCHC packet. It new sublayer is designed to perform data delivery without retransmitting the lost data if the number of lost coded tiles is not higher than a configurable umbral.

The new fragmentation sublayer is based in a Type II Hybrid ARQ/FEC mechanism which redefine the fragmentation sublayer show in Section 5 of the document [RFC8724]. The new fragmentation sublayer consists of two processes: encoding and fragmentation. Figure 2 shows both processes in the sender and the receiver. The SCHC packet coming from the compression sublayer is encoded and fragmented. At the receiver, the process is reversed. The receiver first reassembles the messages and then decodes them to generate the SCHC packet sent. Hereinafter, this document will refer to the fragmentation process as ARQ-FEC mode.

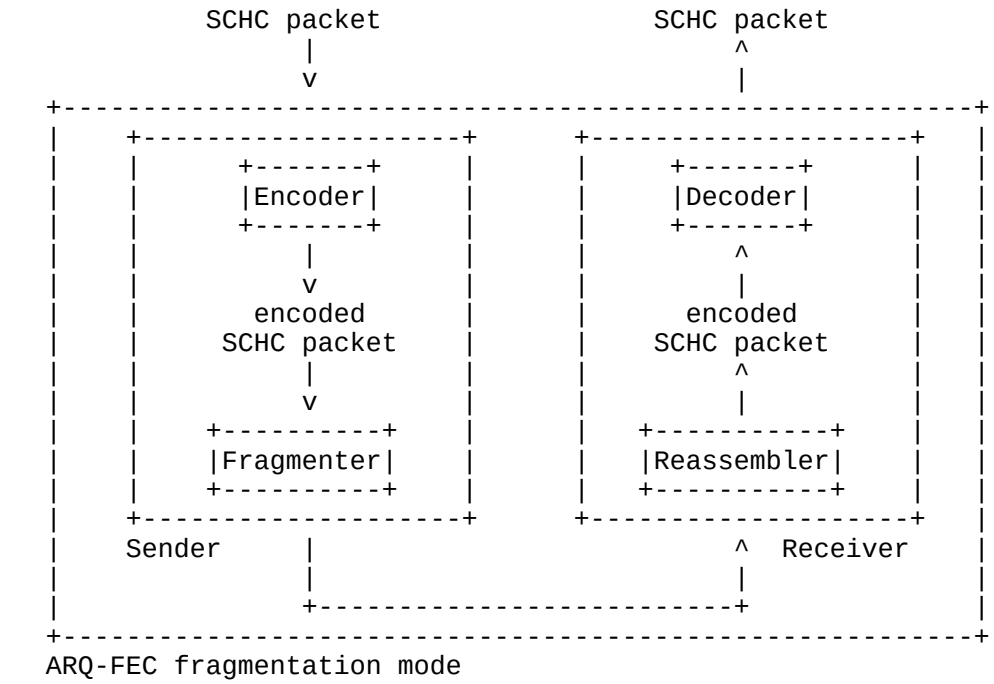


Figure 2: SCHC ARQ-FEC Fragmentation sublayer.

3.1. SCHC ARQ-FEC protocol elements

This subsection outlines the various components used to support the SCHC ARQ-FEC Fragmentation/Reassembly mode functionality defined in this document.

* SCHC Packet size -> P: is the size in bits of the data from compression sublayer (see Figure 24 in [RFC8724]). In this document the SCHC Packet size is called "P".

* Tile Size -> S: is the size in bytes of a regular tile (see 8.2.2.1 in [RFC8724]). The tile size is defined in each profile. In this document the tile size is called "S".

* Dataword length -> F: An SCHC packet is divided into smaller pieces of information called datawords. In this document the dataword size is called "F".

* Redundancy bytes -> rb: are extra bytes added to a dataword to detect and correct errors during transmission.

* Codeword length -> cwl: is the total number of bytes in a codeword, including both data and redundancy bytes. Its value is calculated as cwl = F + rb

3.2. Encoding and Decoding process

3.2.1. Sender Behavior

To encode a SCHC packet, the SCHC ARQ-FEC fragmentation mode implements the encode process in the sender. This process takes a SCHC packet (from the compression sublayer) and create an encoded SCHC packet. To do this, the sender divides a SCHC packet into fixed-size units called datawords. The datawords are encoded and arranged in a C-matrix. A C-matrix generate a encoded SCHC packet. The sender takes the bits from encoded SCHC packet and sends them to the receiver using the ARQ-FEC mode.

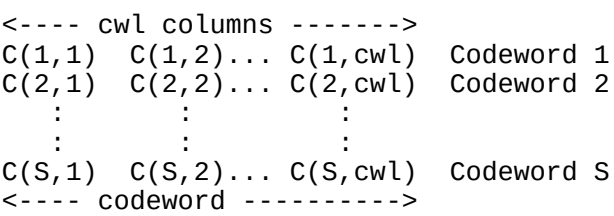
The steps for dividing the SCHC packet and creating the encoded SCHC packet is as follows:

* Divide the SCHC packet into S datawords of F bytes.
  - F is calculated as floor(P/(S*8))
  - If (P mod (S * 8)) equals zero, then there are no remaining bits.
  - If (P mod (S * 8)) is not equal to zero, then there should be (P mod (S * 8)) remaining bits.
* Arrange the datawords in a grid pattern or matrix. Each row of the matrix will be a dataword. The D-matrix has S rows and F columns. Each element of the matrix is a byte. The remaining bits MUST NOT be placed in the D-matrix. Figure 3 shows an example of the arrangement.

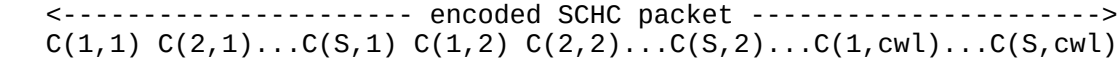
```
          <---- F columns ------->
       ^ B(1,1)  B(1,2)... B(1,F)   Dataword 1
       | B(2,1)  B(2,2)... B(2,F)   Dataword 2
S rows |    :       :        :
       |    :       :        :
       v B(S,1)  B(S,2)... B(S,F)   Dataword S
         <---- dataword -------->
```

Figure 3: Datawords arranged in a D-matrix.

* The channel coding algorithm encodes each dataword and generates a codeword. Each codeword corresponds to a row of the new C-matrix. Figure 4 shows an example of the arrangement. The C-matrix has S rows and cwl columns.



* To generate the encoded SCHC packet, the encoder selects the S-bytes of the first column of the C-matrix. These S-bytes will be the first S-bytes of the encoded SCHC packet. Then, it selects the S-bytes of the second column of the C-matrix. These will follow in the encoded SCHC packet, and so on. Figure 5 shows how the encoded SCHC packet is constructed from the C-matrix.



3.2.2. Receiver Behavior

To decode a encoded SCHC packet in the receiver, the decoding process assumes that there are enough encoded tiles in C-matrix to perform a successful decoding. The following two sections show how create a C-matrix with enought codewords, and how create a SCHC packet from the C-matrix.

3.2.2.1. Creating a C-matrix

On the receiver, a C-matrix is generated from an encoded SCHC packet. An encoded SCHC packet contains (S * cwl) bytes. Its bytes are counted from 1 to (S * cwl) as shown in Figure 6. To mapping between the byte index _i_ of the encoded SCHC packet and the C-matrix position (row, column), use the following equations: * row = ((i-1) mod S) + 1 * column = ceil(i/S)

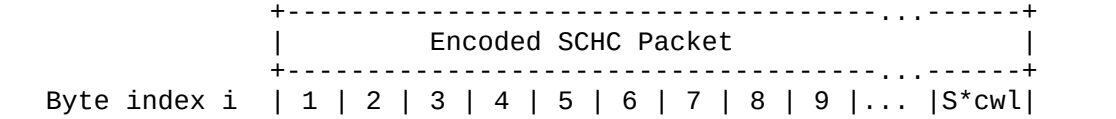


Figure 6: Bytes of a encoded SCHC packet.

3.2.2.2. Generating SCHC packet from C-matrix

If the ARQ-FEC mode receives an All-1 SCHC Fragment, or a All-0 SCHC Fragment, or a SCHC ACK REQ message then the ARQ-FEC mode instructs the decoding process that it can begin decoding from encoded SCHC packet. The decoding process can begin even if some tiles are missing.

To decode a C-matrix, follow these steps:

1. The receiver decodes the bytes in the first row of the C-matrix. The bytes decoded generate the F-bytes of the first row of the D-matrix.

2. The receiver decodes the bytes in the second row of the C-matrix. The bytes decoded generate the F-bytes of the second row of the D-matrix, and so on. If any row cannot be decoded due to missing symbols, continue with the next row.

3. The decoding process ends when the entire D-matrix has been constructed.

    * If any row could not be decoded due to missing symbols, then the decoding process MUST instruct the ARQ-FEC mode to respond to the sender with a SCHC Compound ACK message (c=0) only with the necessary number of tiles to decode. The SCHC Compound ACK message is defined in the document [RFC9441].

    * If all codewords can be decoded, then the decoding process must instruct the ARQ-FEC mode to respond to the sender with a SCHC Compound ACK message with the field c=1.

If all codewords could be decoded, then the receiver creates a SCHC packet following these steps:

1. Select the F-bytes from the first row of the D-matrix. This bytes will be the first F-bytes of the SCHC packet.

2. Select the F-bytes from the second row of the D-matrix. This bytes will be the next F-bytes of the SCHC packet.

3. Continue selecting the F-bytes from each row until you reach the last row of the D-matrix.

4. Add to the end of the SCHC packet the (P mod (S * 8)) remaining bits. This bits are transportes in the All-1 SCHC fragment.

3.3. Fragmentation and Reassembly

The new ARQ-FEC mode is based on the ACK-on-Error mode with some modifications. The following paragraphs detail each modification. Anything not covered in the modifications will be considered as defined in section 8.4.3 of document [RFC8724]:

* Like ACK-on-Error mode, the new ARQ-FEC mode supports L2 technologies that have variable MTU and out-of-order delivery. It requires an L2 that provides a feedback path from the reassembler to the fragmenter.

* Unlike other fragmentation modes, the success of the transfer of tiles is associated with the success of decoding the encoded SCHC packet. Thus, if the fagmentation process loses some tiles, the encoded SCHC packet can still be decoded, the SCHC packet obtained, and the fragmentation declared successful.

* Unlike ACK-on-Error mode, ARQ-FEC mode only confirms at the end of the session whether more tiles are needed to decode the encoded SCHC packet.

Transmission of the encoded SCHC packet concludes when:

* the integrity check shows that all tiles plus remaining bits have been successfully received, or

* the decoder indicates that it has received enough tiles to decode an encoded SCHC packet, or

* too many retransmission attempts were made, or

* the receiver determines that the transmission of the encoded SCHC packet fragmented has been inactive for too long.

The values and size of each parameter in a SCHC header are defined below:

* RuleID size: 30 for uplink and 31 for downlink.

* Tile Size: as defined in each profile, depending on the direction of communication (uplink or downlink).

* M: only one window, then W field not used, M = 0 bits.

* N: is sum between the bits defined for N and the bits defined for M in each profile, depending on the direction of communication (uplink or downlink). For example, in uplink fragmentation for LoRaWAN: M=2 and N=6, then the N for ARQ-FEC uplink fragmentation process over LoRaWAN is 2 + 6 = 8 bits.

* WINDOW_SIZE: (2^N)-1.

* Size and algorithm for the RCS field: as defined in each profile, depending on the direction of communication (uplink or downlink).

* DTag: datagram tag are not used, T = 0 bits.

* MAX_ACK_REQUESTS: 8

* Retransmission timer: Set by the implementation depending on the application requirements. The default RECOMMENDED duration of this timer is 12 hours; this value is mainly driven by application requirements and MAY be changed by the application.

* Inactivity timer: The SCHC gateway implements an "inactivity timer". The default RECOMMENDED duration of this timer is 12 hours; this value is mainly driven by application requirements and MAY be changed by the application.

For each active RuleID, the sender MUST maintain:

* one Attempts counter, and

* one Retransmission Timer.

For each active RuleID, the receiver MUST maintain:

* one Inactivity Timer, and

* one Attempts counter.

3.3.1. SCHC packet size

The fragmentation process support an encoded SCHC packet with size equal to or less than (WINDOW_SIZE * (2^M) * S). Then, the new ARQ-FEC fragmentation mode supports a SCHC packet, sent by the compression layer, that complies with the following size:

floor(P/(S*8))= (WINDOW_SIZE - rb)

3.3.2. Sender Behavior

At the beginning of the fragmentation of a encoded SCHC packet:

* The fragment sender MUST select a RuleID value for this encoded SCHC packet. In addition, the fragment sender MUST initialize the Attempts counter to 0 for that RuleID value.

* A Rule MUST NOT be selected if the WINDOW_SIZE value for that Rule is such that the encoded SCHC Packet cannot be fragmented in WINDOW_SIZE * (2^M) tiles or less.

A Regular SCHC Fragment message carries in its payload one or several contiguous tiles. If more than one tile is carried in one Regular SCHC Fragment:

* The selected tiles MUST be contiguous in the original encoded SCHC Packet, and

* The selected tiles MUST be placed in the SCHC fragment payload next to each other, in the same order as they appear in the encoded SCHC packet, and

* The FCN field MUST contain the tile index of the first tile sent in that SCHC Fragment.

The last tile depends on whether or not there are any bits remaining:

* If (P mod (S * 8)) is not equal to zero, then the conversion from a SCHC packet to a C-matrix generates (P mod (S * 8)) remaining bits. In this case, the last tile are the remaining bits and they MUST be tranport in an All-1 SCHC Fragment.

* If (P mod (S * 8)) is equal to zero, there are no bits remaining. In this case, the All-1 SCHC fragment does not carry a payload and does contain RCS. Because of this, the All-1 SCHC fragment can be distinguished from a SCHC Receiver-Abort.

* In both cases, the fragment sender MUST send a SCHC All-1 message with the W field equal to zero.

The fragment sender MUST listen for SCHC Compound ACK messages after having sent:

* an All-1 SCHC Fragment, or

* a SCHC ACK REQ.

Each time a fragment sender sends an All-1 SCHC Fragment or a SCHC ACK REQ:

* it MUST increment the Attempts counter, and

* it MUST reset the Retransmission Timer.

On Retransmission Timer expiration:

* if the Attempts counter is strictly less than MAX_ACK_REQUESTS, the fragment sender MUST send either the All-1 SCHC Fragment or a SCHC ACK REQ with the W field equal to zero.

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

On receiving a SCHC Compound ACK:

* if the C bit is set to 1, the sender MAY exit successfully.

* if the C bit is set to 0, the fragment sender MUST send SCHC Fragment messages containing all the tiles that are reported missing in the SCHC Compound ACK.

3.3.3. Receiver Behavior

When the reassembler receives a SCHC fragment with a RuleID that is not currently being processed, then:

* The receiver MUST start a process to assemble a new encoded SCHC Packet with that RuleID, and

* the receiver MUST start an Inactivity Timer and an Attempts counter to 0 for that RuleID, and

* if the receiver is under-resourced to do this, it MUST respond to the sender with a SCHC Receiver-Abort.

Whenever a SCHC F/R message arrives for the current RuleID, the receiver MUST reset the corresponding Inactivity Timer.

On receiving a SCHC Fragment message, the receiver determines what tiles were received, based on the payload length and on the FCN field of the SCHC Fragment. If the N bits of the FCN field are not all set to one, then the tiles MUST be assembled based on the a priori known tile size.

On receiving a SCHC ACK REQ or an All-1 SCHC Fragment, if the receiver knows of any missing tiles for the packet being reassembled, it MUST return a SCHC Compound ACK for window numbered 0 indicating which tiles are missing only if the decoder does not have enough tiles to reconstruct the SCHC packet.

Upon sending a SCHC Compound ACK, the receiver MUST increase the Attempts counter.

After receiving a SCHC All-1 fragment, the receiver MUST check the integrity of the reensambled encoded SCHC packet each time it sends a SCHC Compound ACK.

Upon receiving a SCHC Sender-Abort, the receiver MAY exit with an error condition.

Upon expiration of the Inactivity Timer, the receiver MUST send a SCHC Receiver-Abort, and it MAY exit with an error condition.

On the Attempts counter exceeding MAX_ACK_REQUESTS, the receiver MUST send a SCHC Receiver-Abort, and it MAY exit with an error condition.

Reassembly of the encoded SCHC Packet concludes when:

* a Sender-Abort has been received, or

* the Inactivity Timer has expired, or

* the Attempts counter has exceeded MAX_ACK_REQUESTS, or

* at least an All-1 SCHC Fragment has been received the decoder has enough tiles to reconstruct the SCHC packet.

At the end of each session, the reassembler MUST sends a SCHC Compound ACK to the fragmenter indicating which tiles are missing only if the decoder does not have enough tiles to reconstruct the SCHC packet. The fragmenter retransmits the tiles that are reported missing. If the decoder has enough tiles to reconstruct the SCHC packet, the reassembler MUST send a SCHC Compound ACK to the fragmenter indicating that the reconstruction was successful (c parameter equal to 1).

4. Conventions and Definitions

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

5. Security Considerations

This document does not add any security considerations and follows the [RFC8724].

6. IANA Considerations

This document has no IANA actions.

7. Appendix A. Example of an Encode/Decode process with Reed-Solomon

This example represents the encode of a SCHC packet generated by the compression sublayer. The encoding uses the Reed-Solomon algorithm over LoRaWAN technology.

* SCHC packet size: 8950 bits

Encode process parameters:

* Tile size (S): 10 bytes

* Dataword size (F): 111 bytes

* Remaining bits: 70 bits

* Redundance bytes (rb): 44 bytes

Divide the SCHC packet into 10 datawords of 111 bytes. Arrange the datawords in a grid pattern or matrix. Each row of the matrix will be a dataword. The D-matrix has 10 rows and 111 columns. Each element of the matrix is a byte. The remaining bits MUST NOT be placed in the D-matrix. Figure 7 shows an example of the arrangement.

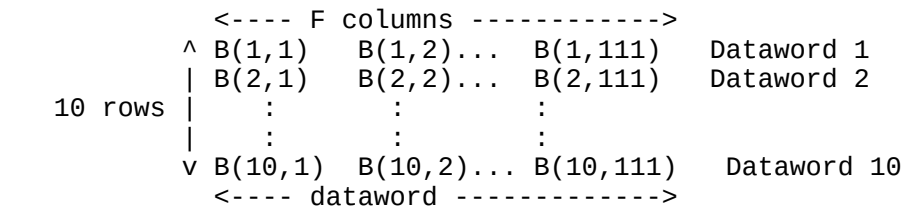


Figure 7: Datawords arranged in a D-matrix.

The Reed-Solomon algorithm generates symbols from a polynomial. The polynomial coefficients are the bytes of each D-matrix row. The polynomial is degree F-1, where F is the dataword size. For example, the polynomial built from the first row of the D-matrix is:

$$y = B(1,1) + B(1,2) * x + B(1,3) * x^2 + B(1,4) * x^3 ... B(1,F) * x^{(F-1)},$$

Moreover, the polynomial of the second row is:

$$y = B(2,1) + B(2,2) * x + B(2,3) * x^2 + B(2,4) * x^3 ... B(2,F) * x^{(F-1)},$$

And so on. Where B(i,j) is:

* the byte of row i and column j of the D-matrix, or

* the j-th byte of dataword i.

The algorithm needs to generate 155 symbols to recover the 111 coefficients of the polynomial. The size of each symbol is 1 byte. To determine the symbols, you can use any of the following methods:

* Message polynomial + field evaluation: direct symbol generation.

* Message polynomial + generator division: parity symbols.

* Generator matrix (Vandermonde): code symbols as a linear algebra product.

The symbols generated form a row of the C-matrix. Therefore, C-matrix has 10 rows and 155 columns and contains all the symbols by the S polynomials. Figure 8 shows an example of the arrangement. The C-matrix has S rows and cwl columns.

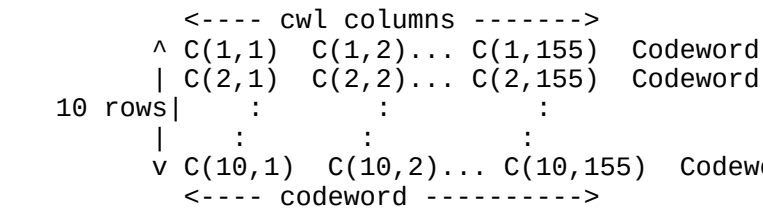


Figure 8: Codewords arranged in a C-matrix.

To generate the encoded SCHC packet, the encoder selects the S-bytes of the first column of the C-matrix. This S-bytes will be the first S-bytes of the encoded SCHC packet. Then, it selects the S-bytes of the second column of the C-matrix. This S-bytes will be the following S-bytes of the encoded SCHC packet and so on. Finally the encoded SCHC packet will have a size of (S x cwl) bytes. Figure 9 shows the bytes that conform an encoded SCHC packet respect to the bytes from C-matrix. At this point, the encoded SCHC packet must be delivered to the fragmentation process.

The encoded SCHC packet does not transport the remaining bits. These bits will be sent in a All-1 SCHC fragment.

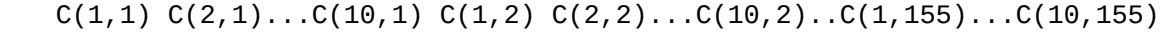


Figure 9: Bytes of a encoded SCHC packet.

8. Appendix B. Example of an ARQ-FEC fragmentation mode

This example represents the transmission of a SCHC packet generated by the compression sublayer. The transmission uses the ARQ-FEC fragmentation sublayer over LoRaWAN technology:

* SCHC packet size: 8950 bits

* Maximum Transmittion Unit (MTU) in AU915-928 LoRaWAN frequency band with DR5 (222 bytes) and DR3 (115 bytes)

ARQ-FEC fragmentation mode parameters: * Tile size (S): 10 bytes * Dataword size (F): 111 bytes * Remaining bits: 70 bits * Redundance bytes (rb): 44 bytes * encoded SCHC packet size (S*(F+rb)): 1550 bytes * W: 2 bits * N: 6 bits * WINDOW_SIZE: (2^N)-1 = 63

8.1. Case 1: Without loss and variable MTU

Each regular SCHC fragment carries a SCHC packet with a 1-byte SCHC header and a SCHC payload of 22 tiles (DR5) or 11 tiles (DR3). Figure 10 shows the first regular SCHC fragment.

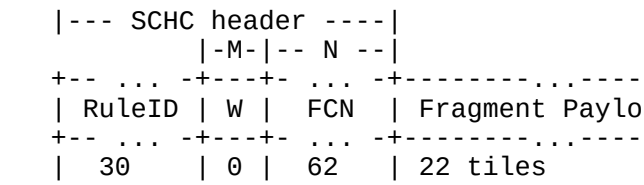


Figure 10: First regular SCHC fragment. Case 1: without loss and varible MTU.

The All-1 SCHC fragment carries the 70 remaining bits plus padding bits. Figure 11 shows the All-1 SCHC fragment of this example.

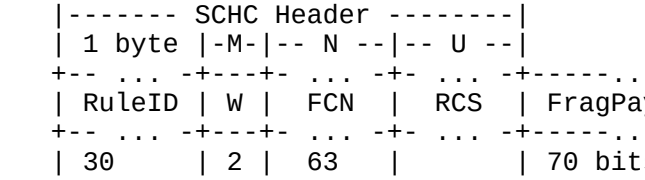


Figure 11: All-1 SCHC fragment. Case 1: without loss and varible MTU.

Figure 12 illustrates the transmission of a SCHC Packet using the ARQ-FEC fragmentation proccess. During the transmission of the encoded SCHC packet, the MTU is modified according to the LoRaWAN data rate.

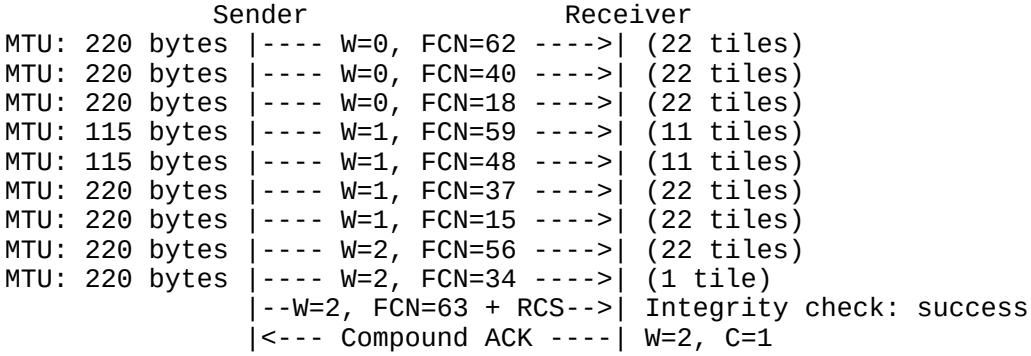


Figure 12: Message flow for ARQ-FEC mode (Case 1).

8.2. Case 2: With loss of fragments. Sufficient fragments in the decoding process.

In this case, there is a loss of fragments, but the decoding process indicate to ARQ-FEC fragmentation mode that there are enougth tiles to decode the encode SCHC packet. Figure 13 illustrates the transmission of a encode SCHC Packet using the ARQ-FEC fragmentation mode. During transmission of the encoded SCHC packet, fragments 2 and 4 are lost.

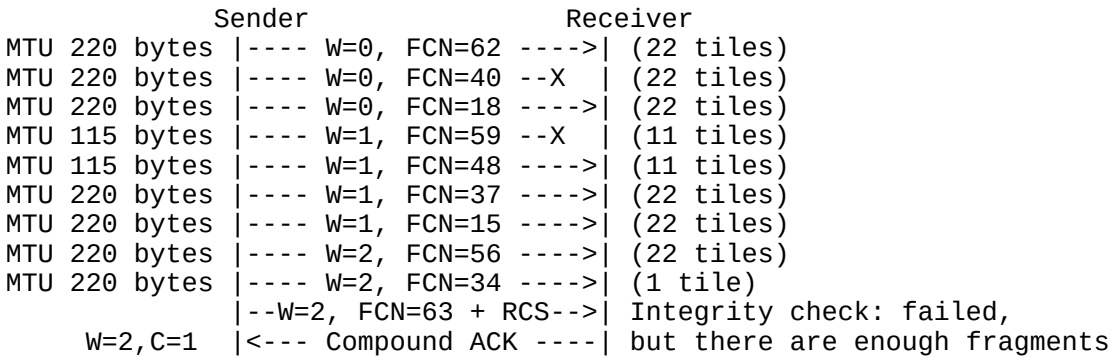


Figure 13: Message flow for ARQ-FEC mode (Case 2).

8.3. Case 3: With loss of fragments. Insufficient fragments in the decoding process.

In this case, fragment loss occurs and the decoding process signals to the ARQ-FEC fragmentation mode that there are not enough tiles to decode the encoded SCHC packet. Figure 13 illustrates the transmission of an encoded SCHC packet using the ARQ-FEC fragmentation mode. During the transmission of the encoded SCHC packet, fragments 2, 4, and 6 are lost. Since only 11 tiles are sufficient to decode the encoded SCHC packet, the receiver sends only one ACK for fragment 4. Figure 14 illustrates the transmission of a SCHC Packet using the ARQ-FEC fragmentation proccess. During the transmission of the encoded SCHC packet, there are fragments loss.

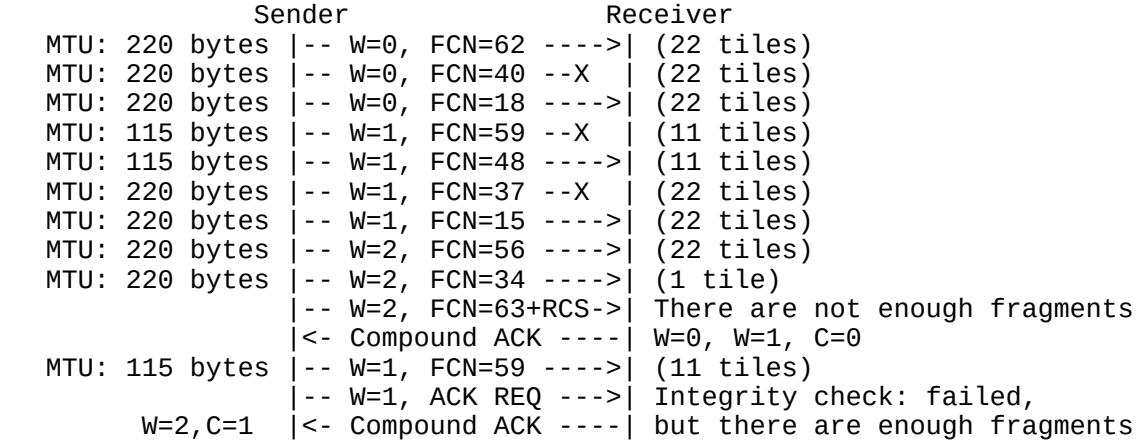


Figure 14: Message flow for ARQ-FEC modee (Case 3).

Acknowledgments

TODO acknowledge.

Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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>.

[RFC9441] Z炭单iga, J., Gomez, C., Aguilar, S., Toutain, L., C迦spedes, S., and D. Wistuba, "Static Context Header Compression (SCHC) Compound Acknowledgement (ACK)", RFC 9441, DOI 10.17487/RFC9441, July 2023, <https://www.rfc-editor.org/info/rfc9441>.

Authors' Addresses

Rodrigo Munoz-Lara
Universidad de Chile
Santiago
Chile
Email: rmunozlara@ing.uchile.cl

Sandra Cespedes
Concordia University
Montreal
Canada
Email: sandra.cespedes@concordia.ca