# Packet Fragmentation over Sigfox: Implementation and Performance Evaluation of SCHC ACK-on-Error

Sergio Aguilar, *Member, IEEE,* Diego S. Wistuba La-Torre, Antonis Platis, Rafael Vidal, Carles Gomez, Sandra Céspedes, *Senior Member, IEEE,* and Juan Carlos Zúñiga, *Member, IEEE*

***Abstract*—The Static Context Header Compression and fragmentation (SCHC) framework has been recently designed by the IETF. SCHC provides adaptation layer functionality intended to efficiently support IPv6-based and other applications over LPWAN technologies. As of the writing, technology-specific profiles are being standardized in order to describe how SCHC can be tailored when used over a given underlying technology. In this paper, we provide the first performance evaluation of SCHC over Sigfox, a flagship LPWAN technology. We focus on the main SCHC over Sigfox fragmentation mode, called ACK-on-Error, which offers low overhead, reliability, and reassembly functionalities. We provide a theoretical analysis and an experimental evaluation in real environments that correspond to two geographical zones with different Sigfox radio settings: Barcelona (Spain) and Santiago (Chile). The study focuses on modeling and evaluating packet transfer times, and the required number of uplink and downlink messages. The results show that a small change in packet sizes may significantly affect the transfer times, especially when no duty-cycle restrictions are enforced. Also, we observe that, under certain conditions, an increase in the fragment loss rate may decrease the packet transfer time, and that the number of uplink and downlink messages is not proportional to such increase due to the fact that downlink messages are device-driven. The results provide useful insights for researchers, developers, implementers, and providers, with applicability to the application design, network planning, and resource management of IoT solutions.**

***Index Terms*—SCHC, Sigfox, IoT, LPWAN, fragmentation.**

## I. Introduction

Low Power Wide Area Network (LPWAN) technologies are gaining high momentum as de facto connectivity solutions for Internet of Things (IoT) devices. The main LPWAN advantages are the long link range (up to several kilometers), and the low power consumption that greatly extends the lifetime of devices, which allow reducing network infrastructure, maintenance and total cost of ownership. However, such benefits are achieved at the expense of a reduced frame size (e.g., in the order of tens of bytes) [1]. Furthermore, LPWAN technologies were designed without native IP support, which limits Internet connectivity. In order to enable and optimize IPv6 and other applications support for LPWAN devices, the Internet Engineering Task Force (IETF) LPWAN working group (WG) has recently standardized the Static Context Header Compression and fragmentation (SCHC) framework [2], [3].

SCHC is an adaptation layer located below IPv6 and atop a given LPWAN technology, and it offers two main components: i) Compression/Decompression (C/D) of upper layer headers, and ii) Fragmentation and Reassembly (F/R) of packets. The former aims to reduce the amount of bits transmitted over the radio interface, whereas the latter is intended to fulfill the IPv6 requirement of supporting a Maximum Transmission Unit (MTU) of up to 1280 bytes [4], despite the constrained LPWAN frame payload size.

Considering the diversity of LPWAN technologies and scenarios, SCHC was purposefully designed to offer generic, technology-independent functionality. To optimize the use of SCHC over a given LPWAN technology, a specific definition of the SCHC mechanism choices and parameter settings over that technology (called a SCHC Profile) is needed. As of the writing of this paper, the IETF LPWAN WG has worked on profiling SCHC over flagship LPWAN technologies such as Sigfox [5], LoRaWAN [6], and NB-IoT [7]. Since C/D depends on static context rules to be defined by the network operator/administrator on a per-deployment basis, SCHC Profiles mainly focus on specifying how F/R is performed over each particular technology.

In this paper, and for the first time to our best knowledge, we investigate the performance of SCHC over Sigfox. We focus on its most promising feature: a reliable, yet efficient F/R mode called ACK-on-Error, which is intended for uplink communication. We provide a theoretical analysis and an experimental evaluation of crucial performance metrics such as packet transfer time and number of Sigfox messages required when using ACK-on-Error. For the experimental evaluation, we implemented a real testbed and performed measurements in

Manuscript received July 30, 2021; revised October 12, 2021; accepted November 2, 2021. This work was supported in part by the Spanish Government through project TEC2016-79988-P funded by AEI / FSE / EU, and and grant PID2019-106808RA-I00 funded by MCIN / AEI / 10.13039/501100011033. This work was also supported by ANID Chile Project FONDECYT Regular 1201893, FONDEF/CONICYT 2019 Folio ID19I10363, and ANID Basal Project FB0008. (Corresponding author: Sergio Aguilar.)

S. Aguilar, A. Platis, C. Gomez and R. Vidal are with the Department of Network Engineering, Universitat Politècnica de Catalunya, 08860 Castelldefels, Barcelona, Spain (e-mail: sergio.aguilar.romero@upc.edu, antonios.platis@estudiantat.upc.edu, carlesgo@entel.upc.edu, rafael.vidal@entel.upc.edu)

D. S. Wistuba La-Torre and S. Céspedes are with the Department of Electrical Engineering and with the NIC Chile Research Labs, Universidad de Chile, Santiago, 8370451, Chile. (e-mail: scespedes@niclabs.cl, wistuba@niclabs.cl)

J.C. Zúñiga is with Sigfox SA, Canada-France (e-mail: juancarlos.zuniga@sigfox.com

two different world regions corresponding to different Sigfox geographical zones, which are characterized by different Sigfox radio features. Among others, our results show that in some cases fragment losses may reduce the transfer time, compared with a scenario without losses. Moreover, very small packet size changes can significantly impact the packet transfer time. Also, we found that the number of uplink and downlink messages is not proportional to a Fragment Loss Rate (FLR) increase due to the fact that downlink messages are device-driven. As a side-contribution of this work, we published our SCHC over Sigfox implementation source code [8], [9].

The remainder of the paper is organized as follows: Section II presents related work. Section III describes the Sigfox technology, detailing its transmission procedures. Section V overviews SCHC over Sigfox, focusing on its uplink ACK-on-Error mode. Section VI presents a theoretical analysis of the transfer time, and the total number of uplink and downlink messages required to carry a fragmented generic data packet over Sigfox. Section VII describes the environment used in our experimental evaluation and presents the results and discussion. Section VIII concludes the paper.

## II. Related work

Recently, SCHC has brought the attention of academia and industry as it enables IPv6 connectivity over LPWAN networks [2], [3], [5]–[7], [10]–[17]. With a standard network layer, the deployment of IoT applications, e.g., data transmission over CoAP, is expected to improve the interoperability of LPWAN technologies with existing Internet technologies.

The definition of specific SCHC Profiles (see section IV-F) is progressing rapidly after the publication of the generic SCHC specification, RFC 8724 [2]. The first profile, SCHC over LoRaWAN has become RFC 9011 [6]. Similar paths are following profile definitions of SCHC over Sigfox [5] and SCHC over NB-IoT [7]. At the same time, there is an increased interest in implementing SCHC and SCHC profiles in simulation, experimental and industrial settings. The Open-SCHC initiative [11] has been evolving together with the IETF's standardization track, with an open-source Python implementation of SCHC that also allows rapid testing via a simulation tool.

To compare SCHC performances to previous proposals of the IETF for different technologies, Ayoub et al. implemented a generic version of SCHC in the NS-3 simulator, focusing on the compression/decompression function [10]. In [12], [13], the performance of SCHC over LoRaWAN is modeled and validated with an experimental testbed limited to the ACK-on-Error mode for uplink traffic. Toutain et al. [14] provided an early implementation in Python of the compression/decompression SCHC function. The code was tested in pycom devices using LoRa, although the authors mentioned it could be easily adapted for Sigfox. In [15], the generic SCHC specifications where simulated using Open-SCHC, providing performance results of SCHC fragmentation, without fragment losses, while using LoRaWAN. Sanchez-Gomez et al. provided an evaluation of the SCHC profile over LoRaWAN using a real testbed that considers regular IPv6 and CoAP traffic [16]. The evaluation established the benefits of using SCHC in terms of delay and packet delivery ratio, and assessed the need for resources in the constrained devices (i.e., computing and memory requirements). To test SCHC in multimodal LPWAN solutions, Moons et al. implemented a generic library for constrained devices using the OSS-7 operating system and the Click modular router for packet processing and IPv6 forwarding [17].

The works mentioned above provide valuable insights into SCHC performance at both generic and profile levels, but are mainly focused on the LoRaWAN technology. In this work, we evaluate the performance of SCHC over Sigfox, analytically and experimentally. The experimental evaluation is based on the first SCHC implementation that addresses the specifics of the Sigfox technology in a scenario of packet fragmentation and reassembly.

## III. Sigfox overview

In this section we provide an overview of Sigfox. We first describe the network architecture and the main radio features of this technology. Then, we focus on the two types of frame transmission transactions offered by Sigfox: the Uplink Procedure (U-procedure) and the Bidirectional Procedure (B-Procedure).

### A. Network Architecture

The Sigfox Network architecture (see Fig. 1) is based on a star topology that includes devices, multiple Base Stations (BSs), the Sigfox Cloud, and application servers. Devices (e.g. sensors) transmit and receive messages using the Sigfox radio technology. Messages are received by one or more BSs, which allows exploiting space diversity [1]. Upon reception at the BS, messages are forwarded to the Sigfox Cloud, which is a single entity that allows global connectivity with minimum effect on the device and on the radio access network, as it connects all BSs, regardless of their geographical location.

The Sigfox Cloud functions as a gateway between a device and an application server. The latter is hosted by a third party, externally to the Sigfox domain. An application server receives messages generated by devices, processes them, and, if requested by the device, returns a response message. The response message is then forwarded by the Sigfox Cloud to the device.

The Sigfox Cloud also forwards to the application server events triggered by device messages (e.g., related with device battery status and temperature, etc.), events regarding network transmission status (e.g., connectivity, network and geolocation metadata, etc.) and messages reporting communication errors.

### B. Radio configuration and features

Sigfox is designed to operate in license-free frequency bands. Sharing unlicensed spectrum presents technical constraints, as use of these frequency bands is subject to local spectrum access regulations. To better handle the diversity of regulations across different world zones, Sigfox defines 7 geographical zones [18]. Each geographical zone is characterized by a specific set of radio features and parameter settings called Radio Configuration

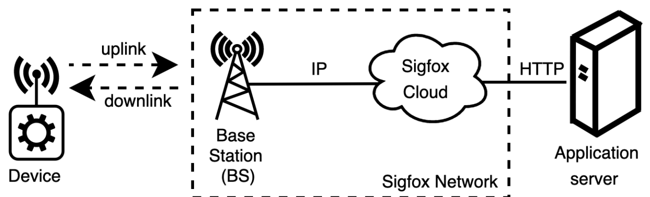


Figure 1. Sigfox Network architecture.

Table I
RADIO CONFIGURATION (RC) FOR DIFFERENT SIGFOX GEOGRAPHICAL ZONES.

| | RC1 | RC2 | RC3 | RC4 | RC5 | RC6 | RC7 |
|---|---|---|---|---|---|---|---|
| Regions or countries | Europe Middle East Africa | Brazil Canada Mexico USA | Japan | Latin America Asia Pacific | South Korea | India | Russia |
| Uplink frequency (MHz) | 868 | 902 | 923 | 920 | 923 | 865 | 868 |
| Downlink frequency (MHz) | 869 | 905 | 922 | 922 | 922 | 866 | 869 |
| Uplink data rate (bit/s) | 100 | 600 | 100 | 600 | 100 | 100 | 100 |
| Downlink data rate (bit/s) | 600 | 600 | 600 | 600 | 600 | 600 | 600 |
| Spectrum access | Duty Cycle (UL 1%, DL 10%) | Frequency Hopping | Listen Before Talk | Frequency Hopping | Listen Before Talk | - | Duty Cycle (UL 1%, DL 10%) |

(RC) [19]. Table I shows the RCs that correspond to the 7 Sigfox geographical zones.

As shown in Table I, Sigfox uses sub-GHz frequency bands in all RCs. The uplink data rate may be 100 bit/s or 600 bit/s, depending on the considered RC. For example, in RC1 (Europe and Africa) the uplink data rate is 100 bit/s, while in RC2 and RC4 (North/South America, Asia and Australia) the uplink data rate is 600 bit/s. The downlink data rate is 600 bit/s in all RCs.

A relevant feature of each RC is its associated spectrum access techniques. For example, in RC1 there is a 1% duty cycle (DC) constraint that must be enforced in the uplink, while in RC4 a radio transmitter must implement frequency hopping. The duty cycle restriction, under the strictest regulation, allows a maximum of 140 uplink messages and a very limited of downlink messages per day (e.g., less than 10, depending on the RC). [1]. However, constraints may vary depending on regulatory and system conditions.

## C. Uplink Procedure (U-procedure)

A device can send data towards the Sigfox Network anytime, provided that radio regulations are enforced. In Sigfox, there are two types of methods to perform an uplink frame transmission: U-procedure, and B-procedure. In the former, the uplink frame does not trigger a response from the Sigfox Network. In the latter, the device requests such a response. This subsection describes the U-procedure.

The uplink frame is transmitted three times using different frequencies. This approach provides time and frequency diversity. Therefore, Sigfox offers a triple diversity, i.e., diversity in time, in frequency, and in space [18]. A U-procedure comprises three different states (Fig. 2):

1) transmission (of duration $T_{Tx}$),
2) wait for next transmission (of duration $T_{Wait_{Tx}}$) and
3) cooldown (of duration $T_{Cool}$).

Fig. 3 shows the Sigfox uplink frame format, which is composed of 10 fields. The Preamble is a 19-bit predefined sequence. The Field Type (FT) is a 13-bit field which signals the message type (i.e., application or control messages). The Length Indicator (LI) is a 2-bit field that provides the length of the Message Authentication Code (MAUTH) (see Table II). The Bidirectional Flag (BF) is a 1-bit field that indicates if the uplink frame is starting a U-procedure (0b0) or a B-procedure (0b1). The Repeated Flag (REP) is a 1-bit field set to 0b0.

The Sigfox uplink frame format also includes a 12-bit Sequence Number. This number is incremented by the device for every uplink frame transmission. At reception, the number is registered by the Sigfox Cloud, which keeps a record of the sequence numbers received. The Device ID is a unique 32-bit value that identifies each device in the Sigfox Network.

The maximum uplink frame payload size ($L_{UL_{MAX}}$) is 12 bytes. The Message Authentication Code (MAUTH) provides frame authentication and has a variable length ($L_{MAUTH}$) which depends on the frame payload size. The total uplink frame size is fixed to a set of values, i.e., 14, 15, 18, 22, and 26 bytes, which lead to only 5 possible radio burst durations (for each data rate). This small and fixed set of total uplink frame sizes simplifies the radio transmitter and receiver design, and allows to better predict the power consumption (i.e., battery lifetime) of the device for a specific application. To accomplish this, considering that the uplink frame payload field is of variable size (between 0 and 12 bytes), the $L_{MAUTH}$ is selected so that its combination with the uplink frame payload always yields one of the possible total uplink frame sizes (see Table II). The uplink frame integrity is secured by using a Cyclic Redundancy Check (CRC) of 16 bits.

Table II shows the Sigfox uplink frame transmission time ($T_{Tx}$) for 100 bit/s and 600 bit/s, for all possible uplink frame payload sizes ($L_{UL}$). Note that $T_{Tx}$ has a variable duration and, due to a dependency between $L_{MAUTH}$ and $L_{UL}$, it exhibits a stepwise behavior as a function of the payload size.

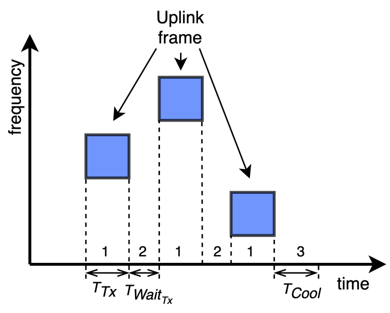


Figure 2. Illustration of a Sigfox U-procedure example. State numbers and their corresponding durations are indicated at the lower part of the figure. In the U-procedure, states 1 and 2 are present three and two times, respectively.

Table II
UPLINK FRAME TRANSMISSION TIME AS A FUNCTION OF THE PAYLOAD SIZE

| $L_{UL}$ (bytes) | $L_{MAUTH}$ (bytes) | LI value (MSB, LSB) | Total frame size (bytes) | $T_{Tx}$ 100 bit/s (ms) | $T_{Tx}$ 600 bit/s (ms) |
|---|---|---|---|---|---|
| 0 | 2 | 00 | 14 | 1120 | 186.67 |
| 1 | 2 | 00 | 15 | 1200 | 200 |
| 2 | 4 | 10 | | | |
| 3 | 3 | 01 | 18 | 1440 | 240 |
| 4 | 2 | 00 | | | |
| 5 | 5 | 11 | | | |
| 6 | 4 | 10 | 22 | 1760 | 293.34 |
| 7 | 3 | 01 | | | |
| 8 | 2 | 00 | | | |
| 9 | 5 | 11 | | | |
| 10 | 4 | 10 | 26 | 2080 | 346.67 |
| 11 | 3 | 01 | | | |
| 12 | 2 | 00 | | | |

### D. Bidirectional Procedure (B-procedure)

The B-procedure allows a device to send data to, and receive data from, the Sigfox Network. Fig. 4 shows a B-procedure example. In contrast with the U-procedure, the 3 uplink frames are now followed by the transmission of a downlink frame and an uplink confirmation frame. The confirmation frame is only sent by the device if the downlink frame is correctly received. The different states involved in a B-procedure, when a downlink frame is sent by the Sigfox Network and received by the device, are the following:

1) transmission (of duration $T_{Tx}$),
2) wait for next transmission (of duration $T_{Wait_{Tx}}$),
3) wait for next reception (of duration $T_{Wait_{Rx}}$),
4) reception (of duration $T_{Rx}$),
5) confirmation transmission (of duration $T_{Conf}$),
6) cooldown (of duration $T_{Cool}$).

The reception state has a variable duration ($T_{Rx}$), which depends on the moment in which the downlink frame is transmitted. If the downlink frame is not received by the device, or no downlink frame is sent by the Sigfox Network, the reception state will last for its maximum value ($T_{Rx_{MAX}}$) and the confirmation transmission state will not be present.

Fig. 5 shows the Sigfox downlink frame format, distributed as follows. The first field is a 91-bit predetermined Preamble. This field is followed by a 13-bit Field Type (FT) field that indicates a B-procedure is taking place. A downlink Error Correction Code (ECC) of 32 bits, which is the result of a BCH(15,11) error correction code applied over the concatenation of the downlink frame payload and the downlink frame MAUTH. The downlink frame payload size is fixed to 8 bytes. The downlink frame MAUTH has a size of 16 bits and it provides authentication for the downlink frame. The downlink frame integrity is protected by using an 8-bit CRC.

| 19 bits | 13 bits | 2 bits | 1 bit | 1 bit | 12 bits | 32 bits | 0 - 96 bits | 16 - 40 bits | 16 bits |
|---|---|---|---|---|---|---|---|---|---|
| Preamble | FT | LI | BF | REP | Sequence Number | Device ID | Frame Payload | MAUTH | CRC |

Figure 3. Sigfox uplink frame format.

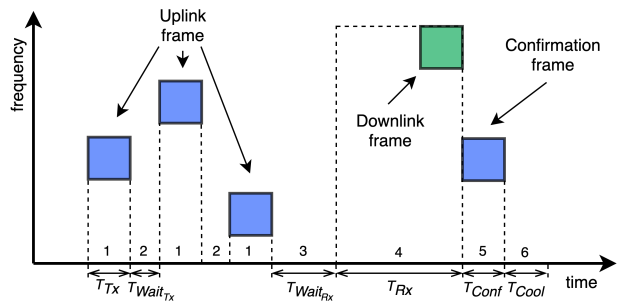


Figure 4. Sigfox B-procedure example illustration. State numbers and their corresponding durations are indicated at the lower part of the figure. In this example, a downlink frame is sent by the Sigfox Cloud and received by the device. The latter sends a confirmation frame. In a B-procedure, states 1 and 2 are present three and two times, respectively. $T_{Rx}$ can take values between $T_{RxMIN}$, when the downlink frame is received at the start of the reception window, and $T_{RxMAX}$, when no downlink frame is received.

| 91 bits | 13 bits | 32 bits | 64 bits | 16 bits | 8 bits |
|---|---|---|---|---|---|
| Preamble | FT | ECC | Frame Payload | MAUTH | CRC |

Figure 5. Sigfox downlink frame format.

The confirmation frame has the same format as the uplink frame (see Fig. 3), with a fixed payload size of 8 bytes. This payload contains information regarding the device battery voltage and Received Signal Strength Indicator (RSSI) estimation made by the device based on the downlink frame received from the BS.

## IV. STATIC CONTEXT HEADER COMPRESSION AND FRAGMENTATION (SCHC)

In this section, we provide an overview of SCHC, focusing on its fragmentation and reassembly functionality, which is the most relevant SCHC component when SCHC is used over Sigfox.

### A. SCHC Overview

SCHC is a generic framework designed to support IPv6 and other non-IP applications over LPWAN technologies. SCHC provides two main mechanisms (see Fig. 6): i) header compression and decompression (C/D), and ii) fragmentation and reassembly (F/R). The former intends to compress/decompress the headers (e.g., IPv6/UDP headers) of a packet before/after transmission. C/D is performed based on a static context shared between the sender (e.g., the device) and the receiver (e.g., located at the network side). A SCHC-compressed packet is called a SCHC Packet. F/R allows to transmit a SCHC Packet over a radio technology with a maximum frame payload size smaller than the SCHC Packet size. The SCHC Packet is broken into smaller data units of suitable size carried by SCHC Fragments.

### B. SCHC F/R Modes

SCHC provides three F/R modes. Each mode has different characteristics intended to adapt to different LPWAN scenarios.

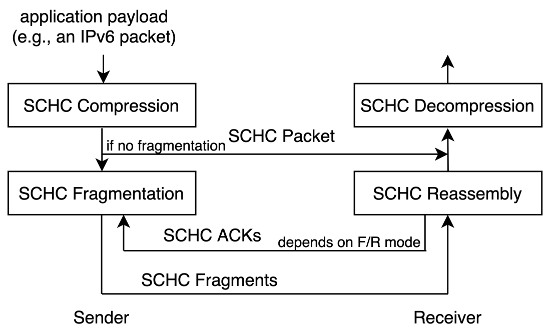


Figure 6. SCHC operation overview. SCHC ACK generation depends on the SCHC F/R mode.

The three F/R modes are, namely: No-ACK, ACK-Always and ACK-on-Error. No-ACK is designed for unidirectional links, with no feedback from the receiver to the sender. ACK-Always and ACK-on-Error offer reliability and require bidirectional links. In ACK-Always, a SCHC Acknowledgment (SCHC ACK) is sent after reception of a group of fragments called a window (see section IV-C), whether or not there are SCHC Fragment losses within the window. In ACK-on-Error, a SCHC ACK is usually only sent if there are SCHC Fragment losses in a window. In both modes, a SCHC ACK is unconditionally sent at the end of the final window to confirm the reception status of the SCHC Packet.

### C. RuleIDs, DTag, Tiles, Windows, and Bitmaps

A SCHC Fragment carries one or several tiles. Its generic format follows a SCHC Fragment header and a SCHC Fragment Payload, being the latter where the tile(s) are transported (see section IV-D1) SCHC F/R uses the concepts of RuleIDs, tiles, windows and bitmaps. We next present these concepts.

RuleIDs identify the rules used for C/D or for F/R. When both C/D and F/R are used, a subset of the RuleID space is dedicated to C/D, whereas the rest is used for F/R. When F/R is used, a RuleID is associated with each SCHC Packet to identify the F/R mode and its behavior, e.g., ACK-on-Error mode with a given configuration.

The Datagram Tag (DTag) is used to identify SCHC Fragments that correspond to the same SCHC Packet.

A SCHC Packet is fragmented in smaller data units called tiles. The tile size ($t$) can be fixed or variable, depending on the F/R mode. In F/R modes with fixed tile size, the last tile is typically the only one of smaller size.

A group of up to WINDOW_SIZE tiles is called a window. All windows, except the last one (which may be smaller), comprise WINDOW_SIZE tiles. Windows are identified by a window number (W) of size M bits. The tiles are numbered based on a unique combination of a sequence number called Fragment Compressed Number (FCN) of size N bits, that is decremented for every new fragment, and W field value. Fig. 7 depicts an example of how a SCHC Packet is fragmented into tiles, and how the tiles are numbered using FCNs. Tiles are then grouped into as many windows as needed.

A bitmap is a sequence of bits whereby each bit represents the reception status of a corresponding tile from a given window.

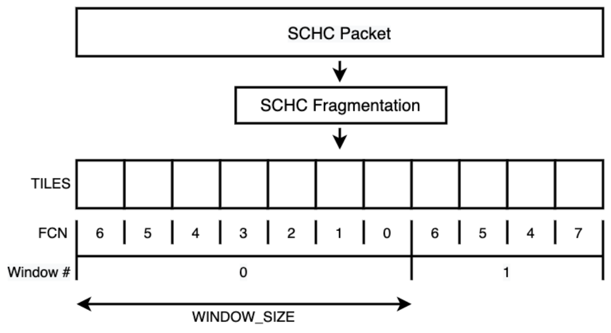


Figure 7. SCHC Packet fragmented in 11 tiles, with a WINDOW_SIZE equal to 7 tiles. FCN and Window number are depicted below each tile.

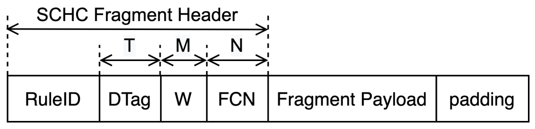


Figure 8. Regular SCHC Fragment format. The DTag field is used to distinguish the SCHC Fragments of a SCHC Packet from other SCHC Fragments using the same RuleID.

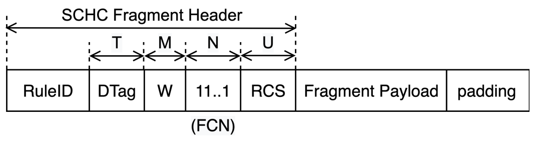


Figure 9. All-1 SCHC Fragment message format.

The bitmap is transported by a SCHC ACK associated to a window to signal missing received fragments from that window.

### D. SCHC Fragmentation messages

SCHC defines the following fragmentation message formats:

*1) Regular SCHC Fragment:* This SCHC Fragment message type is used to carry fragments (i.e., tiles) from the SCHC Packet that are not the last one. Regular SCHC Fragments may carry one or more tiles, depending on the tile size. Fig. 8 shows the Regular SCHC Fragment message format. At the end of a window that is not the last one, the SCHC Fragment FCN must have the value of all bits set to zero. This Regular SCHC Fragment is called an All-0 SCHC Fragment.

*2) All-1 SCHC Fragment:* The last SCHC Fragment of the last window, which means it is the last SCHC Fragment of the corresponding SCHC Packet, is especially signaled by means of setting the value of all FCN bits to 1 (see Fig. 9). This message type may include a Reassembly Check Sequence (RCS). When present, it is used, at the receiver, to check the integrity of the SCHC Packet after the reassembly process. As the All-1 SCHC Fragment carries the last tile, it can have a smaller size than a Regular one.

*3) SCHC ACK Message:* The sender can request SCHC ACKs to the receiver in the reliable SCHC F/R modes (i.e., ACK-Always and ACK-on-Error). How and when the sender requests SCHC ACKs is specific to each LPWAN technology profile. After receiving the last tile of the last window (i.e., upon reception of an All-1 SCHC Fragment) or when a SCHC

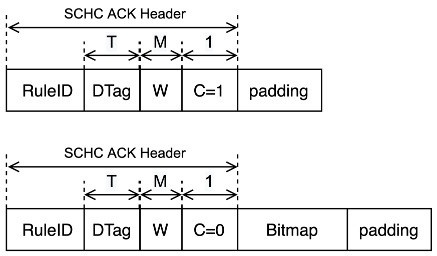


Figure 10. SCHC ACK Message format. The integrity check field (C) is set to 1 or 0 to indicate the success or failure of the SCHC Packet integrity check, respectively. When the integrity check fails (C=0), a bitmap is added to report missing SCHC Fragments.

ACK is requested, the receiver may perform an integrity check on the SCHC Packet and then sends a SCHC ACK (see Fig. 10). If reception of all fragments is successful, and if present, the integrity check is successful, the Integrity Check flag (C) is set to 1. Otherwise, C is set to 0 and SCHC Fragment losses are reported by using the bitmap field (see section IV-C).

Fig. 11a and 11b show examples of the transfer of a SCHC Packet fragmented into 11 tiles (see Fig. 7) when using ACK-on-Error mode, with and without SCHC Fragment losses, respectively.

### E. Counters and Timers

In all SCHC F/R modes, upon SCHC Fragment reception, the receiver resets a timer called Inactivity Timer. When this timer expires, the SCHC Packet transmission is aborted due to inactivity. Furthermore, in reliable SCHC F/R modes, the sender and the receiver keep a counter that is incremented when a SCHC ACK is requested and sent, respectively. The SCHC ACK will be requested and sent a maximum of MAX_ACK_REQUESTS times. The sender will wait, after requesting each SCHC ACK, for a time given by a timer called the Retransmission Timer. If MAX_ACK_REQUESTS is reached, the SCHC Packet transfer is aborted by either the sender or the receiver.

### F. SCHC Profiles

The SCHC framework specification was purposefully defined following a generic approach, in order to offer flexibility for adapting to different underlying LPWAN technologies and scenarios [2]. As shown earlier in this section, SCHC provides several F/R modes and a number of parameters, but it does not specify which F/R modes or parameter settings should be used over each particular LPWAN technology. Such details are deferred to SCHC Profiles, which are technology-specific documents that specify the parameter settings and F/R modes to be used over a given LPWAN technology. As of the writing, SCHC Profiles have been or are being defined for LoRaWAN, NB-IoT, and Sigfox [5]–[7]. Exploiting the generic design of SCHC [2], and its efficiency, SCHC Profiles are also being defined for use of SCHC over protocols and technologies beyond the LPWAN field, such as the Point-to-Point Protocol (PPP) or IEEE 802.15.4 [20], [21].

A SCHC Profile needs to provide information regarding the most common intended use cases, implementation recommendations, mapping of SCHC elements into the corresponding LPWAN architecture, and recommendations for C/D and/or F/R utilization. SCHC Profiles for LPWAN technologies focus mainly on specifying details for F/R functionality [5]–[7]. Such details comprise the F/R modes to be used and the size of each SCHC Fragment header field, among others.

## V. SCHC over Sigfox Profile

The SCHC framework defers to each LPWAN technology profile the proper choice and configuration of the F/R modes to be used (e.g., RuleID field size, N size, M size, padding bits), so that SCHC is suitably adapted to the characteristics of each radio technology. In this section, we describe how SCHC F/R is used over Sigfox.

### A. SCHC over Sigfox Overview

In Sigfox, as in other LPWAN technologies, uplink traffic is typically dominant. The SCHC over Sigfox specification [5] provides two modes for uplink fragmentation: No-ACK and ACK-on-Error. No-ACK mode is intended for the transmission of short, non-critical SCHC Packets, as it does not provide reliability. In contrast, ACK-on-Error defines two variants, offering reliability for short and long SCHC Packets, respectively. ACK-on-Error also reduces the number of ACKs when compared with the other reliable SCHC fragmentation mode (i.e. ACK-Always) [3], [22]. This is suitable considering that in Sigfox the downlink message rate is very limited (see section III-B). This paper focuses on the ACK-on-Error mode, as it provides reliability and efficiency for a very wide range of SCHC Packet sizes.

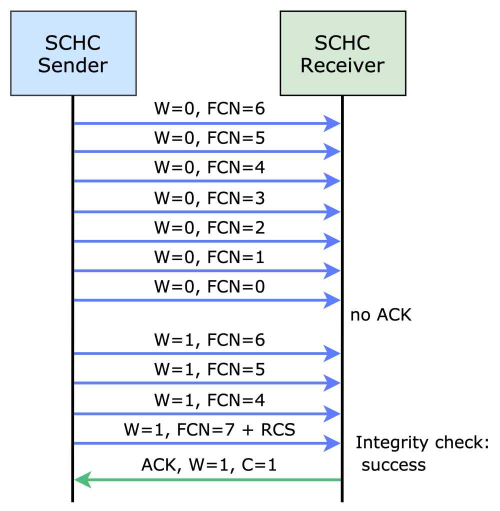
### B. Architecture

The SCHC over Sigfox architecture under study is composed of three main elements (see Fig. 12): the SCHC F/R Sender (hereinafter, the sender), the Sigfox Network, and the SCHC F/R Receiver (hereinafter, the receiver). The sender and the receiver are hosted by the device and the application server, respectively.

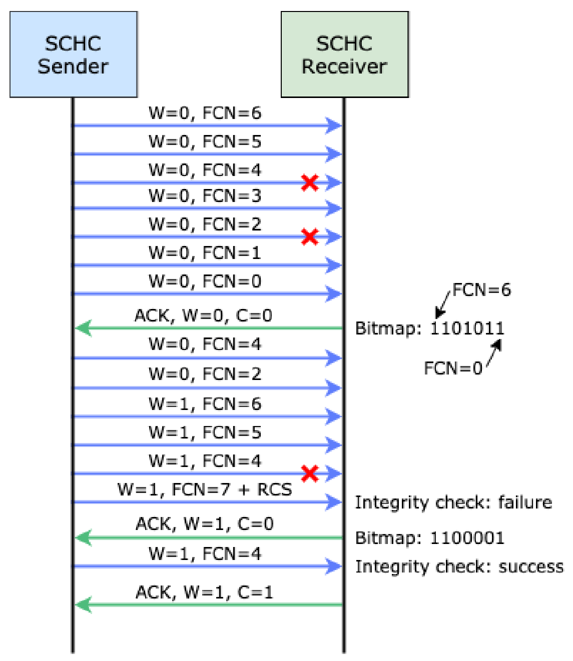
### C. ACK-on-Error mode over Sigfox

In this section we present the two variants of ACK-on-Error mode over Sigfox. These variants use a single-byte and a two-byte fragment header size, respectively.

*1) Single-byte-header ACK-on-Error:* SCHC over Sigfox recommends an 8-bit fragment header for SCHC Packets of a size up to 300 bytes. The fragment header (see Fig. 8) is composed of a 3-bit RuleID, a 2-bit W number size ($M = 2$), and a 3-bit FCN size ($N = 3$). Neither RCS and DTag are used. WINDOW_SIZE is equal to 7 tiles. The tile size is fixed to 11 bytes (i.e., one tile per SCHC Fragment). The SCHC ACK header includes a 3-bit RuleID, a 2-bit W and it may include a 7-bit bitmap (when there are SCHC Fragment losses), for a total of 5 or 13 bits, depending on whether the ACK reports success or failure, respectively (see Fig. 10). In Sigfox, the downlink payload size must always be 64 bits, therefore padding bits must be added.



(a)



(b)

Figure 11. SCHC Packet transfer timeline example, when using ACK-on-Error mode, (a) without SCHC Fragment losses and (b) with SCHC Fragment losses in Window 0 (FCN = 4, 2) and Window 1 (FCN = 4). In this example, the SCHC Fragment header size is 8 bits, distributed as 2 RuleID bits, 1 DTag bit (T=1), 2 W bits (M=2), and 3 FCN bits (N=3).

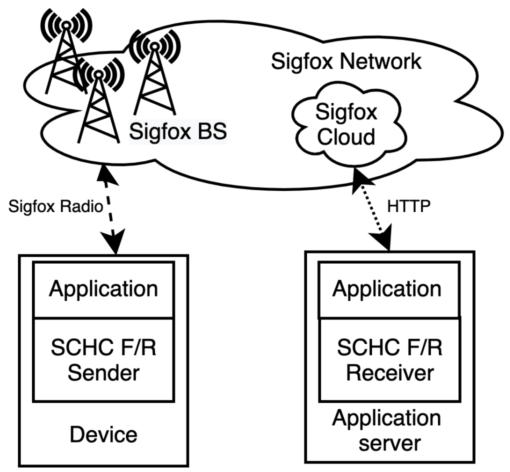


Figure 12. SCHC over Sigfox architecture.

*2) Two-byte-header ACK-on-Error:* Using the two-byte SCHC Fragment header supports the fragmentation and re-assembly of SCHC Packets with a size up to 2250 bytes. This is achieved by adding 1 bit to the W field and 2 bits to the FCN, compared with the single-byte SCHC header. The 16 SCHC Fragment header bits are organized as an 8-bit RuleID, a 3-bit W size ($M = 3$), and a 5-bit FCN size ($N = 5$). RCS and DTag are not used. WINDOW_SIZE is equal to 31 tiles. Compared with the single-byte SCHC header, the W field size now allows for up to 2 times more windows per SCHC Packet and the FCN field size allows up to 4 times more fragments per window. However, the tile size is now fixed to 10 bytes, one byte less. The SCHC ACK size is now of 11 or 43 bits, depending on whether SCHC ACK reports success or failure, respectively (see Fig. 10). Padding bits are added to complete the required downlink frame payload size of 64 bits.

The two-byte SCHC Fragment header may be used to comply with the IPv6 MTU of 1280 bytes, and may also be useful for applications that require long packet sizes, such as smart meters (e.g., gas, water, etc.), waveform captures, data logs, and large data packets using rich data types [23]–[26].

*3) Operation specifics:* In SCHC over Sigfox, in an intermediate window (i.e. a window that is not the last one), an All-0 SCHC Fragment signals the end of the window. The All-0 is sent by using a B-procedure, in order to open a downlink opportunity (see section III-D). The receiver may use this downlink opportunity to send a SCHC ACK if there are fragment losses in current or previous windows, and if the network assesses that radio conditions are favorable. After resending all lost fragments, or if the SCHC ACK is not received after the Retransmission Timer expires, the sender continues with the transmission of SCHC Fragments of the next window.

At the end of the last window (i.e., after the All-1 SCHC Fragment), the sender always expects a SCHC ACK, therefore, it opens a downlink opportunity. The All-1 SCHC Fragment is sent using a B-procedure (see section III-D), and is also used to request a SCHC ACK after retransmission of lost SCHC Fragments.

The SCHC over Sigfox draft specification recommends setting MAX_ACK_REQUESTS to 5. Both the Retransmission Timer and the Inactivity Timer are application-dependent and need to be consistent with spectrum access regulations (e.g. duty cycle constraints). RCS is not always recommended because Sigfox performs an integrity check for each uplink frame delivered by the Sigfox Cloud to the receiver (see section III-C). The DTag is not used, as different RuleIDs can be used to interleave different SCHC Packets sent simultaneously.

Finally, another Sigfox-specific feature is the use of the Sigfox Sequence Number to keep track of SCHC Fragment transmissions and to identify missing fragments.

*4) Single- vs two-byte-header trade-off:* SCHC over Sigfox provides different configurations of ACK-on-Error mode to serve different use cases and SCHC Packet sizes. On the one hand, the single-byte ACK-on-Error mode provides lesser overhead for small SCHC Packet sizes, and a smaller window size (with a smaller bitmap) which increases the downlink opportunities, i.e., the number of times the receiver can provide feedback to the sender and recover from eventual losses. On the other hand, the two-byte ACK-on-Error mode requires a larger overhead with lesser downlink opportunities, which is traded for a faster transfer time, for large SCHC Packet sizes.

## VI. Theoretical Analysis

In this section we present a theoretical performance analysis of the uplink SCHC Packet transfer over Sigfox by using the ACK-on-Error mode. First, we obtain the total delay of a Sigfox message transmission by using the U-procedure and the B-procedure. Based on the obtained results, we calculate the total SCHC Packet transfer time under the conditions considered. Finally, we derive the number of uplink (hereinafter, UL) and downlink (hereinafter, DL) messages required to perform a SCHC Packet transfer. This number is critical since message rates in Sigfox are limited, as aforementioned (see section III-B).

We assume that no fragment losses occur. Therefore, our theoretical model provides a lower bound on the SCHC Packet transfer time, as well as on the number of UL and DL messages required to transfer a SCHC Packet. Note that, in section VII, we evaluate the impact of fragment losses on performance by means of experiments.

### A. Sigfox message transmission procedures delay

We next derive the Sigfox message total delay for the U-procedure and the B-procedure.

*1) U-procedure delay:* The U-procedure (see Fig. 2) total delay ($T_{U-total}$) can be obtained as follows:

$$T_{U-total}(sec) = 3 * T_{Tx} + 2 * T_{Wait_{Tx}} + T_{Cool}. \quad (1)$$

The duration of the transmission state ($T_{TX}$) depends on the Sigfox frame length and the Bitrate (BR). Recall that each Sigfox RC defines an UL BR of either 100 bit/s or 600 bit/s (see section III-B) and the Sigfox protocol overhead per UL data unit is 96 bits (see Fig. 3). Then, $T_{Tx}$ can be calculated as:

$$T_{Tx} = \frac{96 + L_{MAUTH} + L_{UL}}{BR}, \quad (2)$$

where $L_{MAUTH}$ depends on the UL payload size ($L_{UL}$), and it can be obtained from Table II.

*2) B-procedure delay:* The B-procedure total delay ($T_{B-Total-DL}$) can be obtained as follows:

$$T_{B-total-DL}(sec) = 3 * T_{Tx} + 2 * T_{Wait_{Tx}} + T_{Wait_{Rx}} + T_{Rx} + T_{Conf} + T_{Cool}, \quad (3)$$

where $T_{Tx}$ is determined by using (2). Note that this calculation assumes that a DL frame is actually received after the three UL frame transmissions (see Fig. 4). However, in some cases, a DL frame might not be received during a B-procedure. This can happen when a DL message is actually not transmitted, or when such a message is not successfully delivered to the device by the radio link.

The B-procedure total delay when no DL is received ($T_{B-Total-no-DL}$) can be obtained by using (3), with $T_{Rx} = T_{Rx_{MAX}}$, and $T_{Conf} = 0$. Indeed, once $T_{Rx_{MAX}}$ time has passed, the radio proceeds to cooldown. Note that $T_{B-Total-no-DL} \geq T_{B-Total-DL}$ as $T_{Rx_{MAX}} \geq T_{Rx}$. This means that when a DL frame is received, the B-procedure delay may be smaller than when a DL frame is not received.

### B. SCHC Packet Transfer time: analysis

To obtain the SCHC Packet transfer time ($T_{SCHC}$), we first calculate the transmission time of each SCHC Fragment type and of the SCHC ACK. Then we determine the number of SCHC Fragments and SCHC ACKs required to transfer the SCHC Packet.

As explained in section V, a SCHC Packet is carried by different types of SCHC Fragments, such as the Regular SCHC Fragment, including the All-0, and the All-1 SCHC Fragment. All SCHC Fragments, except for the All-1, have a fixed tile size. We next determine the transfer delay of each type of SCHC Fragment.

A Regular SCHC Fragment that is not the All-0 is transmitted by using the Sigfox U-procedure (see section. VI-A1), therefore its transmission time ($T_{U-proc}$) is calculated by using (1), where $T_{Tx}$ is calculated by using (2) with $L_{UL} = 12$ bytes.

The transmissions of the All-0 and All-1 SCHC Fragments use the Sigfox B-procedure. When there are no fragment losses, the All-0 transmission time ($T_{All-0}$) is equal to $T_{B-Total-no-DL}$, and it can be calculated by using (3), with $L_{UL} = 12$ bytes.

The All-1 SCHC Fragment carries the last fragment of the SCHC Packet, and its size may be smaller than the Regular SCHC Fragment (which has a size of 12 bytes). The All-1 SCHC Fragment size $L_{All-1}$ can be calculated as:

$$L_{All-1} = L_{SCHC} - (\left\lceil \frac{L_{SCHC}}{L_{UL_{MAX}} - L_H} \right\rceil - 1) * (L_{UL_{MAX}} - L_H), \quad (4)$$

where $L_H$ is the size of the SCHC Fragment header, $L_{UL_{MAX}}$ is the maximum UL payload size and $L_{SCHC}$ is the size of the SCHC Packet. In UL ACK-on-Error mode, $L_H$ can be either 1 or 2 bytes. The All-1 SCHC Fragment transmission time ($T_{All-1}$) can be calculated by using (2)-(4).

The number of SCHC Fragments using a U-procedure required in a SCHC Packet transfer ($N_{U-proc}$) can be obtained as follows:

$$N_{U-proc} = \left\lceil \frac{L_{SCHC}}{L_{UL_{MAX}} - L_H} \right\rceil - N_{All-0} - 1. \quad (5)$$

The number of All-0 SCHC Fragments used in a fragmented SCHC Packet transmission ($N_{All-0}$) can be obtained as follows:

$$\begin{aligned} N_{All-0} &= N_{Windows} - 1 \\ &= \frac{L_{SCHC}}{WINDOW\_SIZE * t} - 1, \end{aligned} \tag{6}$$

where $N_{Windows}$ is the number of windows required to transmit the SCHC Packet. The number of All-1 SCHC Fragments ($N_{All-1}$), when there are no losses, is 1 for all SCHC Packet sizes.

Once the total number of SCHC Fragments and their transmission times have been calculated, $T_{SCHC}$ can be obtained as follows:

$$T_{SCHC} = N_{U-proc} * T_{U-proc} + N_{All-0} * T_{All-0} + N_{All-1} * T_{All-1}. \tag{7}$$

In Sigfox RCs where duty cycle restrictions apply, the device must keep the transmission radio off for a certain amount of time ($T_{Tx_{OFF}}$), after UL transmissions. This time can be calculated as follows:

$$T_{Tx_{OFF}} = \frac{3 * T_{Tx}}{DC} - 3 * T_{Tx}. \tag{8}$$

where DC denotes the duty cycle enforced (e.g., $DC = 0.01$ corresponds to a duty cycle of 1%).

The total time off for a SCHC Packet transfer ($T_{OFF}$) can be determined as follows:

$$T_{OFF} = (N_{U-proc} + N_{All-0} + N_{All-1}) * T_{Tx_{OFF}}, \tag{9}$$

where $T_{Tx_{OFF}}$ can be approximated to 600 s [1].

Finally, the total SCHC Packet transfer time including all inactive intervals due to enforcing duty cycle regulations ($T_{SCHC_{DC}}$) can be obtained as:

$$T_{SCHC_{DC}} = T_{SCHC} + T_{OFF}. \tag{10}$$

## C. SCHC Packet Transfer time: results

To obtain the SCHC Packet transfer time, first the U-Procedure and B-Procedure states were measured by using an Agilent N6750A power analyzer on a Pycom LoPy4 development board [27].

Table III shows the measured duration of each U-procedure state for Sigfox RC1 and RC4. Conversely, Table IV shows the measured B-procedure transmission states duration. Note that $T_{Rx}$ is variable within a range of values. From our experiments in RC1, the mean reception time $T_{Rx_{Mean}}$ is 14.5 s. Recall that, when no DL frame is received, $T_{Rx}$ takes its maximum value, $T_{Rx_{MAX}}$.

Figs. 13 and 14 show the SCHC Packet transfer time ($T_{SCHC}$) for SCHC Packet sizes from 0 to 2250 bytes, using the values from Tables III and IV, with $T_{Rx_{Mean}} = 14.5\ s$, and using (7), and (9), for RC1, and a duty cycle of 1%, and RC4, respectively. SCHC Packet transfer time tends to increase linearly with the SCHC Packet size. The stepwise behavior in $T_{SCHC}$ as SCHC Packet size increases happens because every 77 or 310 bytes, an additional window is needed (with the corresponding wait time after sending an All-0 message) for single-byte or two-byte SCHC Fragment headers, respectively. Therefore, a small change in SCHC Packet size may lead to a sudden increase in $T_{SCHC}$. However, in RC4, a slight SCHC Packet size increase from a SCHC Packet size slightly below 300 bytes leads to a $T_{SCHC}$ decrease by a factor of $\sim 2$ (see Fig. 14). This sudden discontinuity at 300 bytes occurs because, at that size, the UL ACK-on-Error mode changes from a single-byte to a two-byte SCHC Fragment header. The stepwise behavior is also reflected in $T_{SCHC_{DC}}$ (see Fig. 13). However, the number of fragments transmitted, and their subsequent $T_{Tx_{OFF}}$, become more dominant than the number of windows regarding their impact on $T_{Tx_{OFF}}$.

[1]see https://build.sigfox.com/study (accessed on 06/07/2021).

Table III
U-PROCEDURE TRANSMISSION STATES AND THEIR CORRESPONDING DURATIONS

| State number | State notation | Duration (ms) RC1 (100 bit/s) | Duration (ms) RC4 (600 bit/s) |
|---|---|---|---|
| 1 | $T_{TX}$ | [1120, 2080] | [186.67, 346.67] |
| 2 | $T_{Wait_{Tx}}$ | 1000 | 500 |
| 3 | $T_{Cool}$ | 1000 | 1000 |

Table IV
B-PROCEDURE FRAGMENT TRANSMISSION STATES AND THEIR CORRESPONDING DURATIONS

| State number | State notation | Duration (ms) RC1 (100 bit/s) | Duration (ms) RC4 (600 bit/s) |
|---|---|---|---|
| 1 | $T_{TX}$ | [1120, 2080] | [186.67, 346.67] |
| 2 | $T_{Wait_{Tx}}$ | 500 | 500 |
| 3 | $T_{Wait_{Rx}}$ | 15556 | 15556 |
| 4 | $T_{Rx}$ | [387,25000] | [387,25000] |
| 5 | $T_{Conf}$ | 1799 | 1799 |
| 6 | $T_{Cool}$ | 1000 | 1000 |

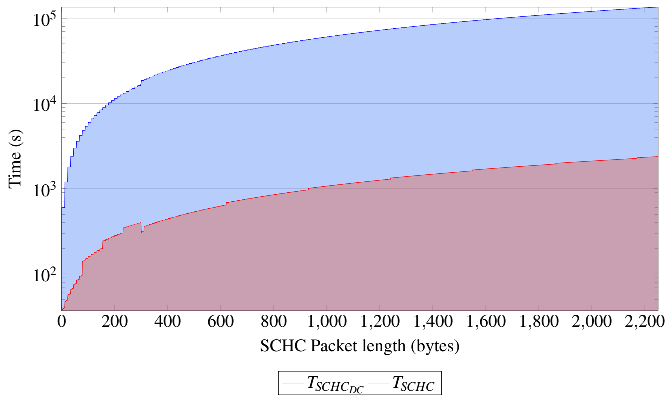


Figure 13. Theoretical SCHC Packet transfer time, for RC1, with duty cycle of 1%. The total time off ($T_{OFF}$) required to comply with duty cycle regulations is depicted in shaded blue.

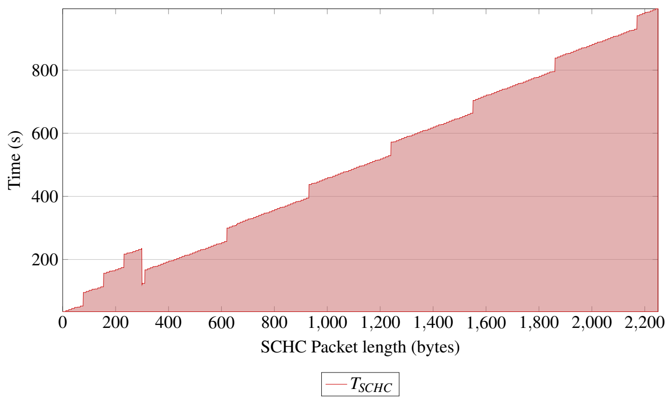


Figure 14. Theoretical SCHC Packet transfer time, for RC4.

Table V
NUMBER OF UL AND DL MESSAGES IN A NO LOSS SCENARIO

| SCHC Packet size (bytes) | $N_{UL}$ | $N_{DL}$ |
|---|---|---|
| 0 | 1 | 1 |
| 11 | 1 | 1 |
| 22 | 2 | 1 |
| 77 | 7 | 1 |
| 90 | 9 | 1 |
| 150 | 14 | 1 |
| 231 | 21 | 1 |
| 233 | 22 | 1 |
| 512 | 52 | 1 |
| 1280 | 128 | 1 |
| 2250 | 225 | 1 |

### D. Number of Messages (UL and DL)

The number of Sigfox UL messages ($N_{UL}$) required to transmit a SCHC Packet by using ACK-on-Error mode can be obtained as follows:

$$N_{UL} = N_{U-proc} + N_{All-0} + N_{All-1}. \quad (11)$$

In absence of fragment losses, $N_{UL}$ can be calculated as:

$$N_{UL} = \left\lceil \frac{L_{SCHC}}{L_{UL_{MAX}} - L_H} \right\rceil . \quad (12)$$

In the considered conditions, the number of DL messages ($N_{DL}$) is equal to 1 for all SCHC Packet sizes. This happens because the ACK-on-Error mode, when there are no fragment losses, only requires one SCHC ACK at the end of the SCHC Packet to confirm the correct reception of all SCHC Fragments.

Table V shows the number of UL and DL messages required to transfer SCHC Packets of different sizes when using the UL ACK-on-Error mode over Sigfox with no fragment losses. The results in Figs. 13, 14, and Table V represent a benchmark for the ones obtained experimentally in section VII.

## VII. EXPERIMENTAL EVALUATION

In this section, we present our experimental performance evaluation of the UL SCHC Packet transfer over Sigfox by using the ACK-on-Error mode. First, we present the experimental environment. Then, we measure the SCHC Packet transfer time and the number of UL and DL messages required for SCHC Packet transfer.

### A. Experimental Environment

In order to study the performance of UL ACK-on-Error mode over Sigfox, we implemented an experimental scenario consisting of the main elements of the SCHC over Sigfox architecture (see section V-B).

We developed the sender on a Pycom LoPy4 development board as the hardware platform. The LoPy4 uses the low-cost, low-power ESP32 system on a chip microcontroller [2] from Espressif, which is widely used in IoT applications. Furthermore, this platform is compatible with the Pycom SiPy development board [28], which has the Sigfox Verified certification.

The receiver was developed to run as a Cloud Function in the Google Cloud Platform [29], where the latter serves as the application server. The Sigfox Network forwards the SCHC Fragments received from the device to the application server by using HTTP messages. Both sender and receiver share the same Python-based implementation, adapted to the requirements of each host. For example, Cloud Functions retain no memory between executions. Because of this, and due to its fast response time, Firebase Realtime Database [30] was used to save all transmission data (e.g., SCHC Packets, SCHC Fragments, bitmaps, Sequence Numbers, etc).

The code implementation is class-based to ease modifications and extensions, and it corresponds to an updated version of the one presented in [31]. A `Profile` class is defined and is extended in the `Sigfox(Profile)` subclass, which sets all the parameters defined in the SCHC over Sigfox profile [5]. In the sender, an object named `Fragmenter` is instantiated to perform the fragmentation process. Each fragment obtained in the process belongs to the `Fragment` class. This class defines the header and payload format for every SCHC Fragment type, particularly specifying the Rule ID, the window number, and the FCN. In the receiver, the `Reassembler` object recovers the original message and performs the reassembly process. Fig. 15 represents the F/R process for an example transmission of three fragments per window.

In our implementation, RuleIDs for single-byte or two-byte ACK-on-Error start with 0b1 or 0b01, respectively. Note that the remaining RuleIDs may be used for C/D.

To keep track of the received fragments, a bitmap is created at the receiver (see Fig. 15). This bitmap becomes part of the SCHC ACK Message that is sent back to the sender, if any fragment is not received. When the transmission is completed, the last executed instance of the receiver performs the reassembly process, reading the fragments from Firebase Realtime Database and storing the SCHC Packet in the same service.

Moreover, at the application server we developed a fragment loss emulator, which enables the artificial injection of fragment losses in any transmission. At both sender and receiver, we developed a data tracker to facilitate the capture of SCHC

[2]https://www.espressif.com/en/products/socs/esp32, accessed on 05/06/2021.

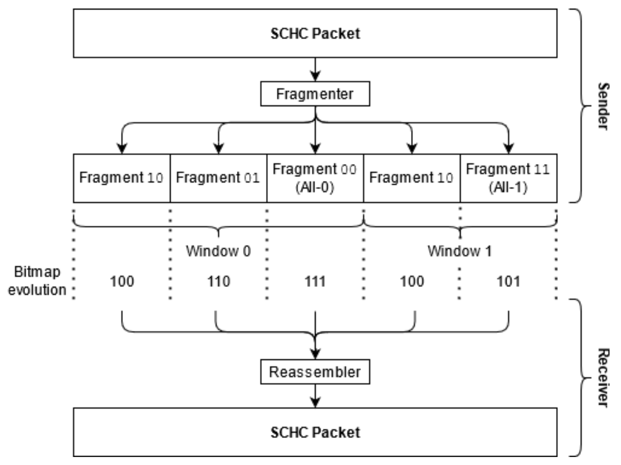


Figure 15. SCHC F/R mechanism diagram using `Fragmenter` and `Reassembler` objects.

Packet transfer metrics (e.g., Sigfox procedure transmission time, total number of fragments sent by the sender and receiver, fragment losses, etc). The source code of the implementations for both sender and receiver are publicly available [8], [9].

## B. Experiment Details

The experiments were performed in Sigfox RC1 (in Barcelona, Spain) and RC4 (in Santiago, Chile). In both scenarios, the Fragment Loss Rate (FLR) was found to be approximately 0%. This allows the validation of the theoretical model, which was developed assuming no fragment losses. However, to test the performance of the SCHC over Sigfox ACK-on-Error mode in the presence of losses, fragment losses following a Bernoulli distribution error distribution were artificially introduced at the receiver with FLR values of 10% and 20%. We performed experiments of three types: i) with no losses, ii) with losses in the UL only, and iii) with losses both in the UL and the DL (UL/DL).

Table VI shows the SCHC Packet sizes employed in the experiments, with the corresponding number of SCHC Fragments, $N_{Windows}$ and $L_H$. These SCHC Packet sizes allow analyzing the impact of the number of windows on the total SCHC Packet transfer time, and on the number of UL and DL messages.

For the experiments, the Retransmission Timer and the Inactivity Timer were set to 60 s and 200 s, respectively. SCHC Packet transfer time results (see section VII-D) were obtained from the average of 10 and 20 experiments for each combination of parameters (i.e., SCHC Packet size, FLR, and RC).

## C. SCHC Packet transfer time: analysis validation

To validate the theoretical SCHC Packet transfer time results presented in section VI, we compare such results with experimental results (Fig. 16). As it can be seen, experimental results of $T_{SCHC}$ are very close to the theoretical ones, with differences that do not exceed 3.38% in the worst case. For small SCHC Packet sizes (i.e., with a small number of U-procedures), these differences mainly happen because of the variability of $T_{Rx}$ in the experimental values of $T_{B-total-DL}$, whereas the theoretical value of $T_{Rx}$, obtained as an average value, is slightly smaller than the experimental ones. For large SCHC Packet sizes (i.e, with a larger number of U-procedures), the SCHC Packet transfer time differences are mainly due to the fact that the measured and experimental values of $T_{U-total}$ are not equal. For a Regular SCHC Fragment that is not an All-0, $T_{U-total_{theoretical}} = 9.24\ s$, whereas $T_{U-total_{experimental}} = 9.4\ s$. The greater value of the latter is due to the processing time overhead of the Sigfox API of LoPy4 [32]. As $N_{U-proc}$ increases, the difference in $T_{U-total}$ becomes more dominant than the differences in $T_{B-total-DL}$, making the experimental SCHC Packet transfer time slightly greater (up to 1.30% for a 2250-byte SCHC Packet) than the theoretical one.

Table VI
SCHC Packet Sizes Evaluated

| SCHC Packet size (bytes) | Number of SCHC Fragments | $N_{Windows}$ | $L_H$ (bytes) |
|---|---|---|---|
| 77 | 7 | 1 | 1 |
| 150 | 14 | 2 | 1 |
| 231 | 21 | 3 | 1 |
| 512 | 52 | 2 | 2 |

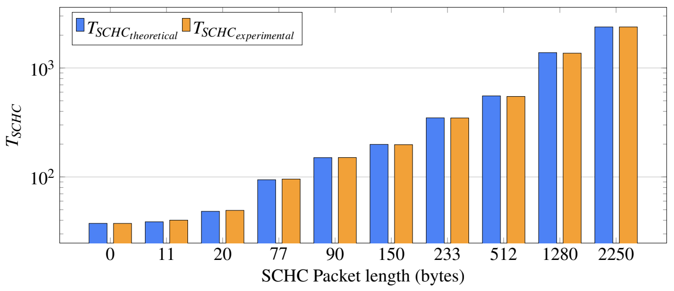


Figure 16. Theoretical and experimental SCHC Packet transfer time with 1% duty cycle. Sigfox zone RC1.

## D. SCHC Packet transfer time: experimental results

In the following, we present the experimental SCHC Packet transfer time results, $T_{SCHC}$, obtained in RC1 (see Fig. 17) and RC4 (see Fig. 18), respectively. Note that RC1, RC3, RC5 and RC6 have the same UL BR (100 bit/s), whereas RC4 and RC2 also share the same UL BR (600 bit/s). Therefore, $T_{SCHC}$ results for RC1 and RC4 can be mapped to all the other Sigfox RCs.

$T_{SCHC}$ increases with the UL FLR and $N_{Windows}$, as a greater number of SCHC Fragment transmissions and retransmissions are needed. SCHC Packet sizes with equal $N_{Windows}$, e.g., 150-byte and 512-byte, exhibit similar behavior with the FLR. The small $T_{SCHC}$ decrease as DL FLR increases is due to fact that Regular SCHC Fragment losses will yield a smaller time. This happens because $T_{B-total-DL}$ is smaller when a SCHC ACK is received (see section VI-C). $T_{SCHC}$ and $T_{SCHC_{DC}}$ show similar behavior with SCHC Packet size and FLR (see Fig. 17).

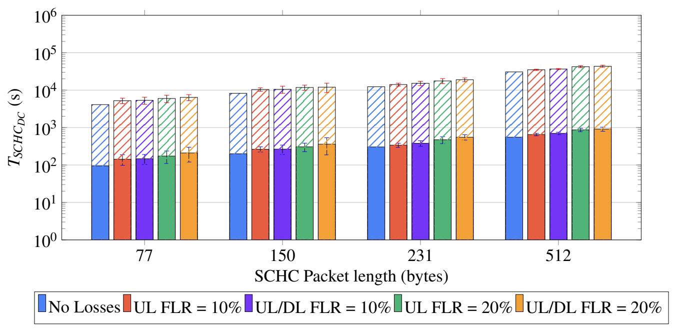


Figure 17. Experimental SCHC Packet transfer time: results obtained in RC1 (1% DC). The solid colored part of the bars represents $T_{SCHC}$, and the dashed part illustrates the time off ($T_{OFF}$) required to comply with duty cycle regulations.

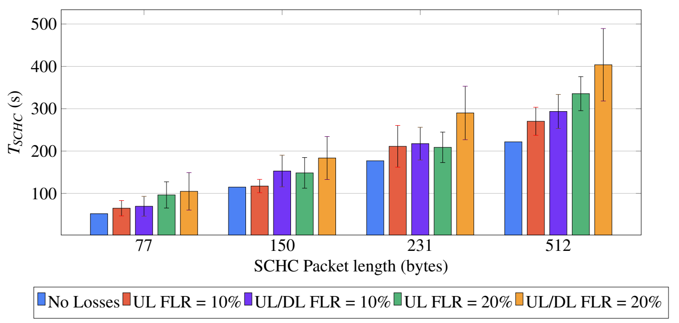


Figure 18. Experimental SCHC Packet transfer time: results obtained in RC4.

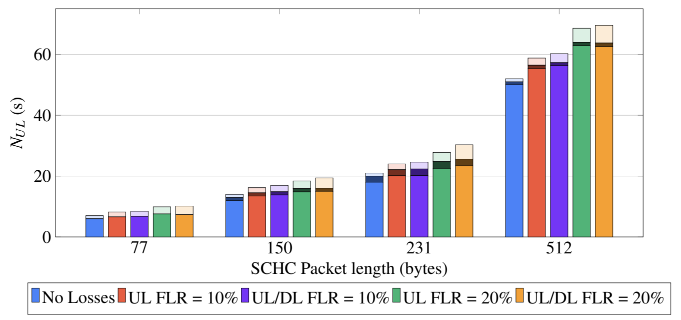


Figure 19. Number of UL messages: experimental results, Sigfox zones RC1 and RC4. The solid colored part, the darker colored part, and the lighter colored part represent the Regular SCHC Fragments, All-0 SCHC Fragments, and All-1 SCHC Fragments, respectively.

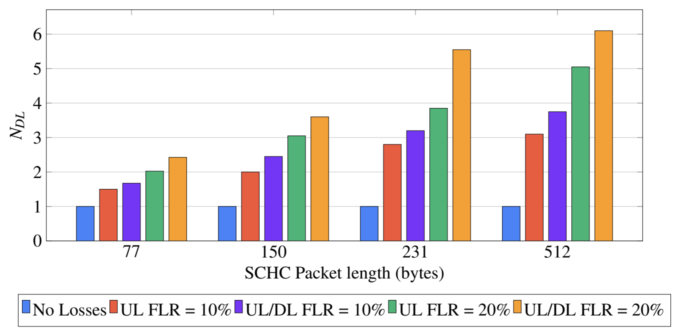


Figure 20. Number of DL messages, experimental results: Sigfox zones RC1 and RC4.

## *E. Average Number of Messages (UL and DL)*

Fig. 19 shows the experimental results for $N_{UL}$ in RC1 and RC4, for the different cases analyzed. Recall that $N_{UL}$ comprises $N_{U-proc}$, $N_{All-0}$ and $N_{All-1}$. $N_{U-proc}$ increases with UL FLR, due to retransmissions. Moreover, $N_{All-1}$ increases with both UL and DL FLR. This happens because Regular SCHC Fragment retransmission cycles trigger additional All-1 SCHC Fragments to request SCHC ACKs, and losses in the DL (i.e., in SCHC ACKs) trigger an UL message (i.e., All-1) that requests again the lost SCHC ACK.

The $N_{All-0}$ increase with UL and DL FLR is small, as this SCHC Fragment will only be requested again if lost. However, $N_{All-0}$ increases linearly with $N_{Windows}$. SCHC Packet sizes with the same number of windows, e.g., 150-byte and 512-byte, yield similar $N_{All-0}$ values, independently of the SCHC Packet size.

Fig. 20 presents the experimental results for $N_{DL}$ in RC1 and RC4, for the different cases analyzed. $N_{DL}$ increases with the UL and DL FLR. As UL FLR increases, more retransmissions are needed to successfully transmit all SCHC Fragments, increasing also the number of SCHC ACKs. Increasing the DL FLR also increases the number of SCHC ACK retransmissions.

Another observation from Fig. 20 is that, in the presence of losses, $N_{DL}$ increases with $N_{Windows}$ (which is also related with the SCHC Packet size). This is due to the greater probability that the transmission of a SCHC Packet involves losses of Regular SCHC Fragments, leading to a greater number of SCHC ACKs.

We also noted that errors in the first windows (for SCHC Packet sizes greater than one window) have a greater impact on performance than errors in the last window, which result in an increase in SCHC Packet transfer time and number of messages required. This happens because the last window is unconditionally acknowledged, and fragment losses within intermediate windows generate additional SCHC ACKs. Therefore, the location of the SCHC Fragment losses directly affects SCHC Packet transfer performance.

One possible future improvement intended to reduce $N_{DL}$ may be reporting losses from several intermediate windows in a single extended ACK. In turn, $N_{UL}$ would also decrease, as less All-1 SCHC Fragments would be required.

## *F. Results applicability and implementation considerations*

The results provided in this section are useful for application design, since they allow to verify the feasibility of running a given application using SCHC over Sigfox, considering SCHC Packet size, FLR, and the limitations in message rate and duty cycle (see section III-B). For example, an application that requires sending a 77-byte SCHC Packet every 120 minutes may be feasible in RC1, and with duty cycle restrictions enforced, since $T_{SCHC_{DC}}$ would be 89 minutes (see Fig. 17), for UL/DL FLR of up to 10%. In another example, sending a SCHC Packet of 231 bytes, for a UL FLR up to 20%, will require slightly less than 4 DL messages (see Fig. 20). If only 4 DL messages per day are allowed, sending such a SCHC Packet per day will be possible. However, an additional non-zero DL FLR may lead $N_{DL}$ to exceed the limit of 4 DL messages.

## VIII. Conclusions

In this paper we have presented the first performance evaluation of the SCHC ACK-on-Error fragmentation and reassembly mode over Sigfox. We provided a theoretical analysis and an experimental evaluation conducted on real testbeds, for different Sigfox radio settings. Through measurements, we obtained the time required by the LoPy4 device for the different transmission stages of a Sifgox uplink and downlink frame, and provide a theoretical model for errorless transmissions, that was validated with the experimental results.

We provided insights that are valid for all Sigfox RCs, regarding the transfer time and the number of uplink and downlink frames, for different SCHC Packet sizes. The number of uplink and downlink frames is critical in LPWAN networks, as message rates are limited. The transfer time of a SCHC Packet provides important insights for delay-sensitive applications, and it is especially critical when duty cycle restrictions are enforced.

## References

[1] S. Farrell, "Low-Power Wide Area Network (LPWAN) Overview," RFC 8376, May 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8376.txt

[2] A. Minaburo, L. Toutain, C. Gomez, D. Barthel, and J.-C. Zúñiga, "SCHC: Generic Framework for Static Context Header Compression and Fragmentation," RFC 8724, April 2020. [Online]. Available: https://rfc-editor.org/rfc/rfc8724.txt

[3] C. Gomez, A. Minaburo, L. Toutain, D. Barthel, and J. C. Zuniga, "IPv6 over LPWANs: Connecting Low Power Wide Area Networks to the Internet (of Things)," *IEEE Wireless Communications*, vol. 27, no. 1, pp. 206–213, 2020.

[4] S. Deering and R. Hinden, "Internet protocol, version 6 (ipv6) specification," Internet Requests for Comments, RFC Editor, STD 86, July 2017.

[5] J. C. Zúñiga, C. Gomez, S. Aguilar, L. Toutain, S. Céspedes, and D. Wistuba La Torre, "SCHC over Sigfox LPWAN," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-lpwan-schc-over-sigfox-05, February 2021, accessed on 04-05-2021. [Online]. Available: https://www.ietf.org/archive/id/draft-ietf-lpwan-schc-over-sigfox-05.txt

[6] O. Gimenez and I. Petrov, "Static Context Header Compression and Fragmentation (SCHC) over LoRaWAN," Internet Requests for Comments, RFC Editor, RFC 9011, April 2021.

[7] E. Ramos and A. Minaburo, "SCHC over NB-IoT," Working Draft, IETF Secretariat, Internet-Draft draft-ietf-lpwan-schc-over-nbiot-04, January 2021, accessed on 04-05-2021. [Online]. Available: https://www.ietf.org/internet-drafts/draft-ietf-lpwan-schc-over-nbiot-04.txt

[8] Aguilar, Sergio and Wistuba, Diego and Platis, Antonis. SCHC over Sigfox Implementation: Network SCHC F/R. Accessed on 04-05-2021. [Online]. Available: https://github.com/saguilarDevel/schc-sigfox

[9] ——. SCHC over Sigfox Implementation: Sender SCHC F/R. Accessed on 04-05-2021. [Online]. Available: https://github.com/saguilarDevel/schcfox

[10] W. Ayoub, F. Nouvel, S. Hmede, A. E. Samhat, M. Mroue, and J.-C. Prévotet, "Implementation of SCHC in NS-3 Simulator and Comparison with 6LoWPAN," in *Proc. of 26th ICT*, Hanoi, Vietnam, Apr. 2019.

[11] OpenSCHC. SCHC Implementation. Accessed on 04-05-2021. [Online]. Available: https://openschc.github.io/openschc/index.html

[12] R. Muñoz Lara, S. Céspedes, and A. Hafid, "Understanding and Characterizing Transmission Times for Compressed IP packets over LoRaWAN," in *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, 2019, pp. 1–6.

[13] R. Muñoz Lara, "Modelado y evaluación de la eficiencia del estándar SCHC para el transporte de paquetes IP sobre LoRaWAN," Master's thesis, Universidad de Chile, Santiago, Chile, 2020.

[14] Laurent Toutain. SCHC. Accessed on 04-05-2021. [Online]. Available: https://github.com/ltn22/SCHC

[15] S. Aguilar, A. Marquet, L. Toutain, C. Gomez, R. Vidal, N. Montavont, and G. Z. Papadopoulos, "LoRaWAN SCHC Fragmentation Demystified," in *Ad-Hoc, Mobile, and Wireless Networks*, M. R. Palattella, S. Scanzio, and S. Coleri Ergen, Eds. Springer International Publishing (Cham), 2019, pp. 213–227.

[16] J. Sanchez-Gomez, J. Gallego-Madrid, R. Sanchez-Iborra, J. Santa, and A. F. Skarmeta, "Impact of SCHC Compression and Fragmentation in LPWAN: A Case Study with LoRaWAN," *Sensors*, vol. 20, no. 1, 2020.

[17] B. Moons, A. Karaagac, J. Haxhibeqiri, E. De Poorter, and J. Hoebeke, "Using SCHC for an Optimized Protocol Stack in Multimodal LPWAN solutions," in *Proc. of IEEE WF-IoT*, Limerick, Ireland, 04 2019.

[18] Sigfox, "Sigfox Connected Objects: Radio specifications," https://build.sigfox.com/sigfox-device-radio-specifications, 2 2019, Rev. 1.3, Accessed on 04-05-2021.

[19] Sigfox. Radio Configurations. Accessed on 04-05-2021. [Online]. Available: https://build.sigfox.com/sigfox-radio-configurations-rc

[20] P. Thubert, "Schc over PPP," Working Draft, IETF Secretariat, Internet-Draft draft-thubert-intarea-schc-over-ppp-03, April 2021, https://www.ietf.org/archive/id/draft-thubert-intarea-schc-over-ppp-03.txt, Accessed on 04-10-2021.

[21] C. Gomez and A. Minaburo, "Transmission of SCHC-compressed packets over IEEE 802.15.4 networks," Working Draft, IETF Secretariat, Internet-Draft draft-gomez-6lo-schc-15dot4-00, July 2021, https://www.ietf.org/archive/id/draft-gomez-6lo-schc-15dot4-00.txt, Accessed on 04-10-2021.

[22] S. Aguilar, P. Maillé, L. Toutain, C. Gomez, R. Vidal, N. Montavont, and G. Z. Papadopoulos, "Performance Analysis and Optimal Tuning of IETF LPWAN SCHC ACK-on-Error Mode," *IEEE Sensors Journal*, vol. 20, no. 23, 2020.

[23] P. Thubert, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Selective Fragment Recovery," Internet Requests for Comments, RFC Editor, RFC 8931, November 2020.

[24] S. Aguilar, R. Vidal, and C. Gomez, "Evaluation of Receiver-Feedback Techniques for Fragmentation over LPWANs," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[25] D. F. Ramírez, S. Céspedes, C. Becerra, and C. Lazo, "Performance evaluation of future AMI applications in Smart Grid Neighborhood Area Networks," in *IEEE Colombian Conference on Communication and Computing (IEEE COLCOM 2015)*, 2015, pp. 1–6.

[26] LoRa Alliance and DLMS User Association, "A Solution for Successful Interoperability with DLMS/COSEM and LoRaWAN," https://lora-alliance.org/wp-content/uploads/2020/11/dlms-lorawan-whitepaper_v1.pdf, Accessed on 07-10-2021.

[27] Pycom. LoPy4 Datasheet. Accessed on 04-05-2021. [Online]. Available: https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf

[28] ——. SiPy Datasheet. Accessed on 04-05-2021. [Online]. Available: https://docs.pycom.io/.gitbook/assets/specsheets/Pycom_002_Specsheets_SiPy_v2.pdf

[29] Google. Google Cloud Function. Accessed on 04-05-2021. [Online]. Available: https://cloud.google.com/functions

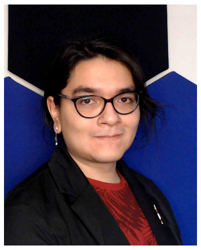
[30] ——. Google Firebase. Accessed on 04-05-2021. [Online]. Available: https://firebase.google.com/

[31] D. Wistuba, S. Céspedes, S. Aguilar, J. C. Zúñiga, , C. Gomez, and R. Vidal, "An Implementation of IoT LPWAN SCHC Message Fragmentation and Reassembly," in *School on Systems and Networks 2020 (SSN2020)*, Vitoria, Brasil, 12 2020.

[32] Pycom. Sigfox LoPy4 API Reference. Accessed on 04-05-2021. [Online]. Available: https://docs.pycom.io/firmwareapi/pycom/network/sigfox/

**Sergio Aguilar** (Member, IEEE) received the B.Sc. degree in electrical engineering and a Lic. degree in communication systems from the Universidad de Costa Rica (UCR) in 2006 and 2008, respectively. He received the M.Sc. degree in telecommunication engineering and management from the Universitat Politèctica de Catalunya (UPC) in 2016, where he is currently pursuing the Ph.D. degree in network engineering.

His interests include the Internet of Things, LPWAN technologies and the IETF SCHC framework.



**Diego S. Wistuba La-Torre** studied Electrical Engineering and Computer Science Engineering at Universidad de Chile, Santiago, Chile, receiving the respective B.Sc. degrees. They currently are a M.Sc. student at the Department of Electrical Engineering of the same university.

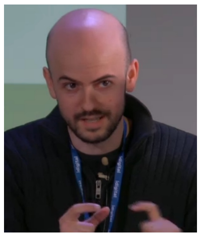
They have worked developing and testing the SCHC communication framework for Low-Power Wide-Area Networks. Their main research interests include communication protocols, information coding and wireless networks.

**Antonis Platis** received his Diploma in Electrical and Computer Engineering from Aristotle university of Thessaloniki (AUTh) in 2018. He also received his M.Sc. in Telecommunications Engineering and Management from Universitat Politècnica de Catalunya (UPC) in 2021.

His research interests include the Internet of Things, LPWANs, as well as Visible Light Communications.

**Rafael Vidal** received his M.Sc. and Ph.D. degrees from the Universitat Politèctica de Catalunya (UPC) in 1997 and 2009, respectively.

He is an associate professor at the same university. He has worked in several publicly funded research projects, and is co-author of several papers published in journals and conferences. His current research interests include performance of wireless networks (in particular, low-power wide-area networks) and the Internet of Things.



**Carles Gomez** received his Ph.D. degree from Universitat Politèctica de Catalunya (UPC) in 2007.

He is an associate professor at the same university. He is a co-author of numerous technical contributions including papers published in journals and conferences, IETF RFCs, and books. His current research interests focus mainly on the Internet of Things.

Dr. Gomez serves as an editorial board member of several journals. He is also an IETF 6Lo working group chair.

**Sandra Céspedes** (IEEE M'12, SM'17) received her B.Eng. (2003) and Specialization (2007) degrees in Telematics Engineering, and Management of Information Systems, from Universidad Icesi, Colombia, and a Ph.D. (2012) in Electrical and Computer Engineering from the University of Waterloo, Canada.

She is an Associate Professor with the Department of Electrical Engineering and the head of the Wireless Networking Research Group (WiNet), Universidad de Chile, Santiago, Chile. Dr. Céspedes holds an honorary Adjunct Professorship at Universidad Icesi, Cali, Colombia. She is the Head of Research at NIC Chile Research Labs and an Associate Researcher with the Advance Center of Electrical and Electronic Engineering (AC3E). Her research focuses on the topics of routing and protocols design for restricted devices and the Internet of Things, vehicular communications systems and networking, and cyberphysical systems.

She serves as an Associate Editor for the IEEE Internet of Things Journal and the IEEE Vehicular Technology Magazine.

**Juan Carlos Zúñiga** (Member, IEEE) received his engineering degree from UNAM, Mexico, and his M.Sc. from Imperial College London, UK.

He is Head of Standardization and IP Strategies at Sigfox, a global IoT Network and Service Provider. He is co-chair of the IETF Internet Area and MADINAS working groups.

He was chairman of the IEEE 802 Executive Committee Privacy SG, he is co-author of several IETF RFCs, and he has actively participated in other standards organizations such as ETSI, 3GPP and W3C. He has vast experience developing technology and business in IoT, NFV/SDN, 4G/5G and IEEE 802.11 (Wi-Fi). He has served as Guest Editor of IEEE Communications Magazine, and is inventor of over 70 granted USPTO/EPO patents. Before joining Sigfox, he worked at InterDigital, Harris Communications Canada, Nortel Networks UK, and Kb/Tel Mexico.