An Implementation of IoT LPWAN SCHC Message Fragmentation and Reassembly

Diego Wistuba[1,2], Sandra Céspedes[1,2], Juan Carlos Zúñiga[3], Rodrigo Muñoz[1], Sergio Aguilar[4], Carles Gomez[4], and Rafael Vidal[4]
[1]Department of Electrical Engineering, Universidad de Chile, Chile
[2]NIC Chile Research Labs, Universidad de Chile, Chile
[3]Sigfox, Canada
[4]Universitat Politècnica de Catalunya, Spain

{wistuba,scespedes}@niclabs.cl, juancarlos.zuniga@sigfox.com, rodrigo.munoz.lara@gmail.com, sergio.aguilar.romero@upc.edu, {carlesgo, rvidal}@entel.upc.edu

Abstract

This extended abstract reports about the implementation of the fragmentation and reassembly mechanisms of the SCHC framework for Low Power Wide Area Networks (LPWANs). The project aims to program the methods and routines described in the SCHC standard protocol for the fragmentation and reassembly mechanism while being independent of the LPWAN technology and remaining as an open-source project. The implementation addresses the ACK-on-Error mode, where an acknowledgement message is sent back to the sender device only when lost fragments are detected. Initial tests have been successful using a local communication setting with network sockets. We are currently porting the code to support the Sigfox LPWAN using Google Cloud Platform as a cloud computing service and the Python framework Flask for easy testing.

In: Proceedings of the IV School of Systems and Networks (SSN 2020), Vitória, Brazil, December 14-15, 2020. Published at http://ceur-ws.org.

1 Introduction

The Low Power Wide Area Networks (LPWANs) comprise a number of technologies to support the exponential growth of the Internet of Things (IoT) end devices and associated applications, providing low transmission data rates, large coverage areas, and long battery life. Such technologies are, however, constrained by small packet data sizes. The need to expand the potential of LPWANs leads to the requirement of Internet connectivity and the definition of mechanisms for supporting IPv6 packet transmissions over LPWAN [GMT+20].

The Static Context Header Compression and Fragmentation (SCHC) is a generic framework applicable to LPWANs and recently defined in the IETF RFC 8724 [MTG+a]. It is destined to compress IPv6/UDP headers and support the transport of IPv6 packets if the datagram, after SCHC compression, still exceeds the Layer 2 Maximum Transmission Unit (MTU). To achieve this, SCHC defines a set of static Rules (or Context), containing information about the compression/decompression (C/D), as well as the application of an optional fragmentation/reassembly (F/R) mechanism. SCHC can be implemented over any of the LPWAN radio technologies considered by the IETF LPWAN Working Group, namely: LoRaWAN, Sigfox, NB-IoT, and IEEE 802.15.4-based solutions. Any LPWAN technology that wants to use SCHC must define a profile—a set of specific parameters needed to support the framework.

This project aims to implement the F/R mechanism computationally and test it over real LPWANs,

being independent of the technology used and staying as open-source code once finished. In particular, we describe the implementation of the ACK-on-Error mode over the Sigfox LPWAN. The architecture of SCHC over the Sigfox network is represented in Figure 1, composed by a Sigfox device, base station and network gateway, which is the Sigfox cloud-based Network. Messages transmitted towards the Sigfox network gateway are labeled "uplink", and messages going back to the device are called "downlink" [MTG+b].

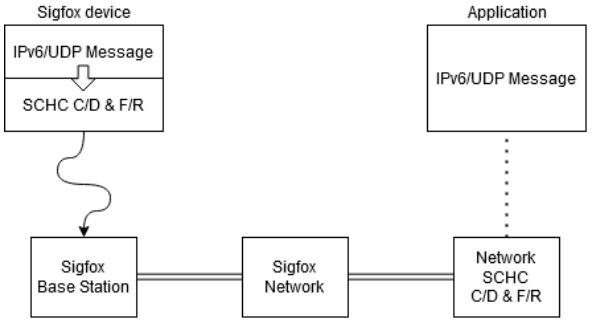

Figure 1: SCHC over Sigfox architecture. The curved line represents wireless communication. The dotted line represents communication towards an application [MTG+b].

This implementation of SCHC over Sigfox starts applying the F/R mechanism to a message exceeding the MTU of an LPWAN profile splitting it into fragments of size less or equal to this MTU. The fragments are grouped in windows of a certain length determined by the profile, and numbered in descending order from this length to 0. These fragments are then sequentially sent into the network as uplink messages. In the ACK-on-Error mode (see Section 2.1), if fragments are lost then the network sends a downlink message requesting retransmission of those fragments.

2 Description of SCHC implementation over Sigfox

2.1 Architecture description

With the goal of being independent of the LPWAN technology, object-oriented classes were chosen to work with. A class named `Profile` was created, meant to be extended for every SCHC profile. To simulate an LPWAN network in this situation, the Sigfox profile for SCHC was chosen and instantiated, assigning parameters as needed [MTG+b]. Eventually, more extensions of the `Profile` class can be created.

SCHC defines various types of messages, of which the Fragment and the ACK are the most frequent. Similarly as before, a `Message` class was created with a header and a payload as parameters, extended in `Fragment` and `ACK`.

Any message has a header and a payload. The header is a bit field of a certain length that contains information about the window to which the fragment belongs, its position inside that window, and optional parameters like error detection fields. It is implemented in a `Header` class, and its instantiations take part as an attribute of the `Message` class. This aims for easy management of the previously mentioned parameters. On the other hand, the payload has a part of the original message, and padding may be added as needed. It is added directly as an attribute of the `Message` class.

The fragment's position in the window is represented in the Fragment Compressed Number (FCN), and two types of messages are distinguishable: when the FCN is composed of only zeros (All-0) or only ones (All-1). These fragments are located at the end of each window, and particularly the All-1 is at the end of the last window. The full representation in bits of this header and payload is defined accordingly inside of the `Message` class.

The functionalities of generating all the fragments from an original message following this format are implemented in the `Fragmenter` class, and the process of obtaining the original message given a list of fragments is implemented in the `Reassembler` class. A simple view of the F/R mechanism is represented in Figure 2

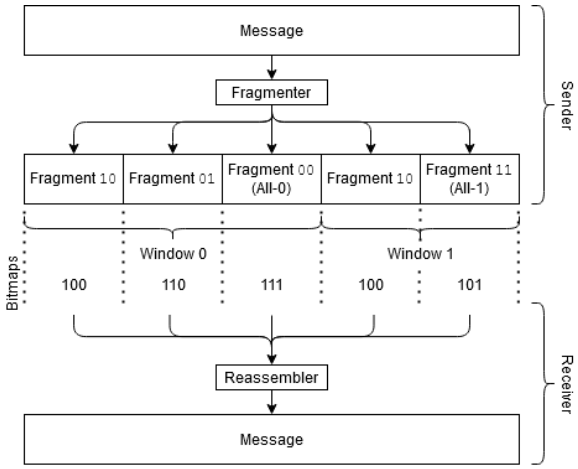

Figure 2: SCHC F/R mechanism with a window of size 3. The FCN and ACK bitmap are displayed for every fragment. This considers no loss of fragments.

To keep track of the received fragments at the receiver end of the network, a bitmap is defined. Every bit in this bitmap stands for a fragment in each window, being 0 initially and changing to 1 when the corresponding fragment is received. This bitmap is part of the header of the acknowledgement (ACK) message, which may be sent after each fragment is received (ACK-Always), when a fragment is detected

to be lost (ACK-on-Error), or not be sent at all (No-ACK). These reliability configurations are named F/R modes. The `ACK` class has the bitmap as a parameter within its header. This project addresses the ACK-on-Error mode, defined in [MTG+a]. A representation of the ACK-on-Error mode is displayed in Figure 3.

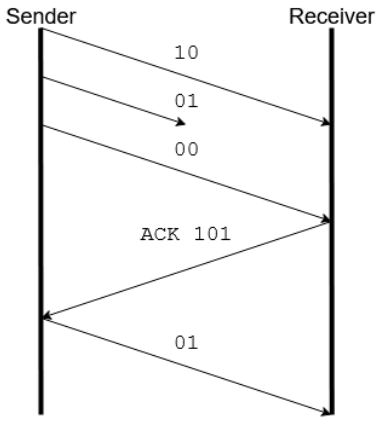

Figure 3: ACK-on-Error mode with a non-final window of size 3. The FCN and ACK bitmap are displayed for every fragment. Here, the second fragment (with FCN 01) is lost. Then the receiver sends a downlink ACK with its bitmap and the sender responds with the corresponding fragment.

2.2 Implementation description

The project is being developed in the Python programming language due to its easy handling of classes and network sockets. Initially the code was developed in a local environment, with no network connectivity other than communication within the same computer between network sockets, running sender and receiver scripts. After this stage, the code is currently being ported to support actual LPWAN end devices.

A `sender` script and a `receiver` script are written. The `sender` receives a message meant to be fragmented as its input, instantiates the `Fragmenter` class and executes its main method, `fragment()`, which returns an array of all the SCHC fragments. Then the sender follows the algorithm described in section 8.4.3.1 in [MTG+a], in order to send these fragments and acting accordingly when an ACK message is returned by the receiver.

The `receiver` script constantly waits for fragments on the network and follows the algorithm described in section 8.4.3.2 in [MTG+a], sending ACK messages for every undetected fragment. When every fragment has been received and located in an ordered array, this script instantiates the `Reassembler` class and similarly executes its main method, `reassemble()`, which returns the concatenation of every payload.

In local testing, the receiver script is meant to be run once. This script binds a socket to a port and enters a loop while hearing for new fragments, storing them in an array and breaking out of the loop when the transmission is complete and then executing the reassembly routine. This script can also test the ACK message system, emulating fragment loss randomly ignoring received fragments with a certain probability. These tests were made with the attributes of the Sigfox SCHC profile [MTG+b].

3 Preliminary results

The local tests with this implementation were successful, and the scripts are currently being modified in order to be run over an actual LPWAN. Using the Sigfox technology, a callback between its backend and a cloud computing service was created. Google Cloud Platform (GCP) was chosen for this purpose, and the Cloud Functions and Cloud Storage APIs are being used. The callback, via HTTP, forwards a message received by the Sigfox backend to GCP, where a Cloud Function gets triggered and executes a script. These scripts are stateless and retain no memory between executions.

Various changes in the scripts were proposed to support this new architecture. The receiver script is now executed as a Cloud Function in the Python framework Flask every time a fragment is received and writes it into a file in Cloud Storage, constantly reading and writing the bitmaps as files. When the final message of the transmission is received and no lost fragment is detected, the script reads every fragment into memory and executes the reassembly routine, saving the result as another file. When needed, an ACK message is generated and returned to the Sigfox network as an HTTP response in order to be sent back to the device that runs the sender script.

The sender script has not yet been ported to LPWAN end devices, but it has been refactored as an HTTP client that simulates the callback between the Sigfox network and GCP. The fragmentation routine is not affected, and the fragments are sent directly to GCP (or Flask) using an HTTP POST request.

4 Conclusions and future work

The fragmentation and compression functionalities that SCHC offers have the potential to enable LPWANs to support several industrial IoT applications. It can help for instance supporting bigger payloads for utilities' smart meters or visual sensors, recovering messages lost due to radio interference or coverage issues which may be critical for asset-tracking applications, adding IP connectivity to sensors/devices that

have very limited resources, enabling device management functionalities, etc.

New functionalities are currently being tested with the setup presented in this document, such as the timeout between fragments, SCHC abort messages and various corner cases of the communication.

It is expected that the outcome of this project will become available as open-source code, which can be used as baseline to develop industrial applications to support the aforementioned use cases.

Acknowledgements

This project was supported in part by the ANID Chile Project FONDEF ID19I10363.

Sergio Aguilar, Rafael Vidal and Carles Gomez were funded in part by the Spanish Government through projects TEC2016-79988-P, PID2019-106808RA-I00 (AEI/FEDER, and UE); and the Generalitat de Catalunya Grant 2017 SGR 376.

References

[GMT+20] Carles Gomez, Ana Minaburo, Laurent Toutain, Dominique Barthel, and Juan Carlos Zúñiga. IPv6 over LPWANs: Connecting Low Power Wide Area Networks to the Internet (of Things). *IEEE Wireless Communications*, 27(1):206–213, 2020.

[MTG+a] Ana Minaburo, Laurent Toutain, Carles Gomez, Dominique Barthel, and Juan Carlos Zúñiga. RFC 8724 - SCHC: Generic Framework for Static Context Header Compression and Fragmentation. https://datatracker.ietf.org/doc/rfc8724/.

[MTG+b] Ana Minaburo, Laurent Toutain, Carles Gomez, Dominique Barthel, and Juan Carlos Zúñiga. SCHC over Sigfox LPWAN. https://datatracker.ietf.org/doc/draft-ietf-lpwan-schc-over-sigfox/.