UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

IMPLEMENTACIÓN Y EVALUACIÓN DE FRAGMENTACIÓN DE PAQUETES IPV6 PARA UN ENLACE LORAWAN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

JUAN JOSÉ MILLÁN CASTILLO

PROFESOR GUÍA:
SANDRA CÉSPEDES UMAÑA

MIEMBROS DE LA COMISIÓN:
CLAUDIO ESTÉVEZ MONTERO
ALBERTO CASTRO ROJAS

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: JUAN JOSÉ MILLÁN CASTILLO
FECHA: 2019
PROF. GUÍA: SANDRA CÉSPEDES UMAÑA

IMPLEMENTACIÓN Y EVALUACIÓN DE FRAGMENTACIÓN DE PAQUETES IPV6 PARA UN ENLACE LORAWAN

La Internet de las Cosas es un nuevo paradigma tecnológico que nace de la necesidad de interconectar una gran cantidad de dispositivos a Internet, ya sea máquinas u objetos de uso diario, con el fin de crear soluciones que faciliten la vida de las personas de forma cotidiana o en la industria. Su auge trae consigo nuevos desafíos en cuanto al desarrollo de redes pues cada vez se requieren sistemas más seguros, escalables e interconectables entre sí. Uno de estos desafíos corresponde a la implementación a gran escala de *Internet Protocol* version 6 (IPv6).

En la actualidad se han desarrollado diferentes redes para suplir la conectividad requerida por las tecnologías del Internet de las Cosas, dentro de las cuales aparece la categoría de *Low Power Wide Area Networks* (LPWAN), redes cuyos principales características son el bajo consumo de potencia y su gran rango de cobertura, ideales para el despliegue de redes de monitoreo.

Una de las principales tecnologías LPWAN es la conocida como *Long Range Wide Area Network* (LoRaWAN) que, además de cumplir con las características de las LPWAN, es bastante flexible pues funciona en una banda de frecuencia libre dada por la banda industrial, científica y médica (ISM) y es de fácil acceso económico, características bien vistas por los desarrolladores. El rápido crecimiento en el uso de esta tecnología como también el uso de otras LPWAN hace aún mas apremiante la necesidad de implementar el protocolo IPv6 para así asegurar interoperabilidad entre todos los sistemas emergentes.

Para lo anterior, la Internet Engineering Task Force (IETF) se encuentra actualmente trabajando en el desarrollo de un mecanismo de compresión y fragmentación de paquetes IPv6 para redes LPWAN denominado Static Context Header Compression (SCHC). Se han realizado varias implementaciones de la etapa de compresión pero no así de la etapa de fragmentación, por lo que en este trabajo se plantea realizar una demostración experimental del algoritmo de fragmentación de paquetes propuesto por la IETF.

La plataforma creada logra implementar el enlace para mandar grandes paquetes de datos a través de fragmentos y evalúa la implementación para diferentes configuraciones de parámetros, además de sugerir modificaciones al algoritmo. Con las pruebas realizadas se midieron las tasas de error de transmisión de paquetes y fragmentos y se concluyó que el algoritmo tiene un desempeño aceptable pero puede ser mejorado, principalmente en el comportamiento de los mensajes de *acknowledgment* que dependerá de la entidad que sea elegida como receptor, que pueden ser tanto *Network Server* (administrador de la red) o *gateway*.

A mi mamá, por confiar en mi y escucharme siempre que lo necesité.

A mi papá, por impulsarme en este camino de la ingeniería con su gran ejemplo.

A mi tata Pablo y abuelita Yola, por recibirme en su hogar todos estos años y hacerme sentir como en casa.

A la profe Sandra, por su paciencia y buena disposición como guía en este proceso.

A mi hermana, por sus consejos de vida y su buena energía.

A mi hermano, por sus risas y buena onda que siempre me recordaron lo hermoso de la familia.

A Nico y Sole, por apañar en todas durante mi paso por eléctrica y dar cara siempre. Cuanto aguante!

A Dani y Yorch, por regalarme su amistad cuando me vi solo en una ciudad desconocida.

A los cabros de la sección 8 y la ebria, con quienes compartí tantas risas y motivamos tantos viernes. Los mejores recuerdos los tengo junto ustedes.

Y a todos los que con su granito de arena hicieron de estos últimos años y mi paso por la universidad una época que jamás olvidaré.

Agradecimientos

Agradezco a mi compañero de eléctrica Ricardo Ramos por ayudarme, sin esperar nada a cambio, a realizar algunas de las pruebas necesarias para llevar a cabo este trabajo, pruebas que no hubiera podido hacer yo solo dada las características de éstas. También quiero agradecer a mis otros dos compañeros de eléctrica, Nicolás Sepúlveda y Solange Vivanco por acompañarme en este proceso y ayudarme a resolver algunos problemas a los cuales me enfrenté.

Tabla de Contenido

Índice de Ilustraciones

Introducción

0.1. Motivación y Antecedentes

La Internet de las Cosas nace de la necesidad de conectar una gran cantidad de dispositivos a Internet con diferentes aplicaciones. Por ejemplo, en la industria, se pueden monitorear más fácilmente los procesos presentes y tomar acciones frente a los datos adquiridos; en los domicilios se puede facilitar la vida de las personas automatizando una gran cantidad de actividades diarias; en el ámbito de la salud, se pueden monitorear procesos biológicos; los ejemplos de aplicación pueden ser extendidos a muchos otros dominios, como el transporte o la agricultura.

Dado su gran número de aplicaciones se vio como necesario desarrollar diferentes tecnologías que permitieran establecer las redes a las cuales se podrían conectar estos dispositivos de medición y acción, motivando a empresas a investigar sobre diferentes protocolos de comunicación que se ajustaran a las necesidades de estas redes y dispositivos, que en algunos casos están restringidos en su comunicación y capacidades computacionales.

Este impulso permitió el desarrollo de las *Low Power Wide Area Networks* (LPWAN), redes cuya principal característica es permitir la comunicación de dispositivos a través de largas distancias con un consumo mínimo de potencia, permitiendo que los objetos que se comuniquen usando estos protocolos puedan funcionar por varios meses o años con una misma carga de batería, lo que puede ser muy útil, por ejemplo, para el monitoreo de sistemas naturales, el seguimiento de personas u objetos o la modernización de las ciudades.

Se espera que las LPWAN, al ser tan útiles, promuevan un crecimiento en los dipositivos conectados de un 109 % por año hasta el año 2023 alcanzando más de mil millones de dispositivos conectados [7]. Este crecimiento hace que se vuelva importante pensar en la escalabilidad, seguridad y conectividad de estos sistemas, problemas comunes de la implementación de estas redes. Los actuales protocolos utilizados en estas redes en su mayoría no aseguran la protección de la información, la gran escalabidad de los sistemas y la capacidad de interconección entre ellos, pero sí existe uno que los resuelve, el protocolo de red IPv6 [1].

Las características del protocolo IPv6 hacen que su implementación sobre las LPWAN sea muy útil y urgente, sin embargo esto aún no se ha realizado debido a que estas redes no están capacitadas para transportar paquetes del tamaño que necesita este protocolo. Al maximizar las distancias y minimizar el consumo de potencia, se hizo necesario disminuir

considerablemente el tamaño de los paquetes a enviar, haciendo imposible que un paquete IPv6 sea transmitido directamente ya que éstos son muy grandes para los enlaces establecidos.

Dada la problemática anterior, la *Internet Engineering Task Force* (IETF) se encuentra trabajando en un mecanismo de compresión y fragmentación de paquetes para transmitir paquetes IPv6 sobre las redes LPWAN [5], en particular, en las redes LoRaWAN, un tipo de LPWAN cuya principal característica es, aparte de las típicas de este tipo de redes, el bajo costo de sus dispositivos terminales. Esta implementación aún no se ha llevado a la práctica ya que se encuentra en desarrollo (IETF Draft) [22], aún en proceso de convertirse en un estándar, por lo que sería un gran aporte para el estado del arte probar su implementación y evaluar su funcionamiento en la medida que la propuesta estándar se va afinando.

La implementación del IETF Draft consiste en introducir una capa de adaptabilidad entre la capa MAC y la capa de red de las comunicaciones, que disminuya el tamaño de los paquetes IPv6 a enviar. Esta capa consta de dos etapas, la primera consiste en la compresión de los paquetes y la segunda en la fragmentación de éstos. La etapa de compresión fue abordada en un Trabajo de Memoria de Título anterior, realizado por Nicolás Maturana Araneda [21], mientras que la etapa de fragmentación no ha sido implementada aún. Es así como en este Trabajo de Memoria de Título se propone la implementación y evaluación de la etapa de fragmentación de paquetes para completar la capa de adaptación.

0.2. Definición del Problema

LoRaWAN no soporta el protocolo de red IPv6 debido a que el tamaño de los paquetes IP es mas grande que los que pueden enviarse a través del protocolo de capa física LoRa, por lo que los dispositivos finales de este tipo de redes no pueden conectarse directamente a Internet. Esto hace necesaria la presencia de un nodo Gateway que establezca la conexión a las redes externas y un *Network Server* que se encargue de administrar estas conexiones, complejizando la arquitectura de la red e impidiendo aprovechar las ventajas de IPv6 para conectividad extremo a extremo.

Para habilitar el uso de IPv6 en los dispositivos finales de LoRaWAN se debe crear una capa de adaptación que modifique el tamaño de los paquetes IPv6 a enviar. La IETF se encuentra trabajando en una implementación sugerida (IETF Draft) [22] de la capa de adaptación de IPv6 para LoRaWAN, pero actualmente no se tienen implementaciones prácticas de esta sugerencia en su totalidad, sólamente se tienen implementaciones de la etapa de compresión en un Trabajo de Memoria de Título anterior y en códigos libres resultantes de *hackatones* organizadas por la IETF [11], pero ninguna de ellas integra la etapa de fragmentación.

0.3. Objetivo General

Implementar el algoritmo de fragmentación y reensamblaje de paquetes IPv6 sugerido por la IETF sobre paquetes transmitidos por un nodo LoRaWAN, evaluar su desempeño para diferentes contextos de comunicación (especialmente el contexto presente en una *smart city*) y permitir, en un futuro, la conexión a Internet de éste con IPv6 extremo a extremo.

0.4. Objetivos Específicos

1. Implementar el algoritmo de fragmentación en un paquete IPv6 y transmitir sus fragmentos a través de un nodo LoRaWAN.
2. Probar la fragmentación de paquetes para diferentes contextos de comunicación y evaluar su desempeño calculando las tasa de éxitos de transmisión y pérdidas de paquetes y la cantidad de mensajes necesarios para transmitir un paquete fragmentado.
3. Configurar los parámetros necesarios para simular el enlace que se encontraría en una aplicación real y evaluar qué tan efectivo sería el uso del algoritmo de fragmentación para dicha aplicación.

0.5. Metodología

Para realizar este trabajo se ha decidido proceder como sigue:

1. Estudiar los RFC de los protocolos utilizados y el IETF Draft para preparar la implementación del sistema. Esto incluye estudiar el contenido de cada header y la secuencia del algoritmo de fragmentación.
2. Implementar en Micro-Python los algoritmos de fragmentación y reensamblaje de paquetes para emisor (nodo) y receptor (*gateway*) presentados en el borrador IETF *Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6 version 21* [22] y en el borrador *IETF Static Context Header Compression (SCHC) over LoRaWAN version 3* [28].
3. Cargar las implementaciones en un nodo y *gateway* basados en la tarjeta de desarrollo Pycom LoPy 4 programable en el lenguaje Micro-Python.
4. Generar un paquete en el Pycom nodo y aplicar la fragmentación para diferentes escenarios de prueba para luego transmitir los fragmentos al Pycom gateway.
5. Recibir todos los fragmentos a través del *gateway* e inspeccionar su contenido. Luego aplicar el algoritmo de reensamblaje a los fragmentos recibidos y reconstruir el paquete IPv6 original generado.
6. Evaluar el desempeño del algoritmo sugerido por la IETF calculando tasas de pérdidas de paquetes y fragmentos y el número total de mensajes transmitidos por paquete fragmentado, esto realizando pruebas en un contexto de ciudad a una distancia fija entre nodo y *gateway* variando los parámetros más importantes del enlace y para dos tamaños de paquetes, 500 y 1500 bytes.

Capítulo 1

Marco Teórico

Este capítulo presenta los conceptos fundamentales necesarios para el estudio de las redes LPWAN, específicamente LoRaWAN, y la implementación del protocolo IPv6 sobre este tipo de redes.

1.1. Internet de las Cosas

La internet de las cosas (*Internet of Things*, IoT) corresponde a una nueva área tecnológica que abarca todo tipo de redes de objetos físicos o "cosas" provistas de circuitos electrónicos, sensores y software capaz de establecer comunicación con Internet y otros objetos de iguales características. Se refiere a dispositivos, usualmente restringidos en su comunicación y capacidades computacionales, que se conectan a Internet con el fin de ofrecer diferentes servicios, como por ejemplo la recolección de datos.

El auge de esta área y su relación con otras tecnologías emergentes ocasionó que varias compañías se volcaran a desarrollar dispositivos y tecnologías de acceso que suplieran esta nueva necesidad de interconexión de objetos, obligando a la Internet Engineering Task Force (IETF) a establecer estándares que aseguraran la interoperabilidad de estos sistemas.

Dentro de estas tecnologías de acceso se establecieron las *Low Power Wide Area Networks* (LPWAN) correspondientes a redes cuyos dispositivos pueden comunicarse entre sí a grandes distancias y con gastos energéticos muy bajos. Este tipo de redes son el foco de esta memoria.

1.2. Tecnologías LPWAN

Las características principales de estas redes son [19]:

- Bajo consumo de potencia en los dispositivos terminales.
- Baja tasa de transferencia de datos.

- Gran cobertura en distancia.

Para obtener estas características los dispositivos tienden comúnmente a restringir fuértemente el ancho de banda de sus comunicaciones y hacer uso del canal en ciclos de trabajo muy cortos, quitándole importancia a la latencia de comunicación, que suele ser alta en este tipo de redes, pero que no es un problema para muchas de las aplicaciones *machine-to-machine* para las que fueron diseñadas.

Varias empresas de tecnología han desarrollado protocolos y dispositivos para este tipo de redes. A continuación se nombrarán las principales tecnologías LPWAN consideradas por la IETF [19]:

- LoRaWAN
- Narrowband IoT (NB-IoT)
- Sigfox
- Wi-SUN Alliance Field Area Network (Wi-SUN)

Este estudio está enfocado a la tecnología LoRaWAN la cual será explicada a continuación.

1.3. LoRa

Se le llama LoRa (*Long Range*) al nivel de capa física (PHY) de la tecnología LPWAN LoRaWAN [27]. Esta tecnología permite la comunicación punto a punto entre dos dispositivos y se caracteriza por su largo alcance. Para ello usa una técnica de modulación propietaria de espectro ensanchado llamada *Chirp Spread Spectrum* (CSS) [26] que a diferencia de otro tipo de modulaciones maximiza el alcance para bajas potencias de transmisión. La ventaja de LoRa es el largo alcance de las comunicaciones. Un solo nodo puede cubrir una pequeña ciudad o cientos de kilómetros cuadrados [31]. El rango depende áltamente de las condiciones del ambiente y las obstrucciones. Sin embargo, LoRa es la tecnología que tiene el mayor presupuesto de alcance entre las tecnologías de baja potencia.

La comunicación entre dispositivos y *gateways* se extiende a diferentes canales de frecuencia y tasas de transferencia de datos. La selección de la tasa de transferencia de datos se realiza mediante la compensación entre el rango de comunicación y la duración de los mensajes. Las tasas varían entre 0,3 kbps a 50 kbps. Para maximizar la capacidad de las baterías y la capacidad general de la red, la infraestructura de la red LoRa puede manejar las tasas y frecuencias de cada dispositivo individualmente usando un esquema de tasa de transmisión adaptativa (*Adaptive Data Rate*, ADR) [27].

Las frecuencias que LoRa usa son principalmente las de la banda ISM (*Industrial, Scientific and Medical*), aunque la tecnología puede operar en cualquier frecuencia por debajo de 1 GHz. Las bandas ISM disponibles varían dependiendo de la región del mundo en que se esté por lo que existen diferentes módulos LoRa fabricados especialmente para cada una de ellas: EU868 (Union Europea, 868MHz), EU433 (Union Europea, 433MHz), US915 (Estados Unidos, 915MHz), AU915 (Australia 915 Mhz) y AS430 (Asia, 430MHz)[31]. Particularmente,

en Chile, se permite el uso de dispositivos LoRa en la banda de los 915Mhz [9] tomando como parámetros los correspondientes a la región AU915.

1.3.1. Características de la modulación: Chirp Spread Spectrum

Técnica de modulación cuyas características principales son su bajo requerimiento en potencia para la transmisión de mensajes y su robustez frente a diferentes mecanismos de degradación de canal como lo son los multicaminos (*multipath*), el desvanecimiento de la señal (*fading*) y efecto Doppler, lo que la hace muy útil para aplicaciones de LPWAN's [26].

Su funcionamiento se basa en la transmisión de pulsos modulados en frecuencia de duración arbitraria T. Durante este tiempo la frecuencia del pulso varía desde un valor bajo a uno alto o viceversa. La diferencia entre estas dos frecuencias es una buena aproximación para su ancho de banda. En la figura 1.1 se tiene un ejemplo del pulso anteriormente mencionado.

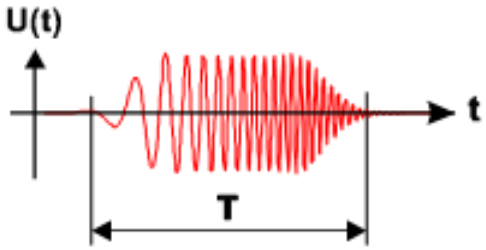


Figura 1.1: Ejemplo de pulso chirp. Fuente [4]

Ancho de Banda, Spreading Factor y Data Rate

A medida que aumenta el producto entre el ancho de banda (BW) y el tiempo en el aire de un pulso (T) la señal se hará cada vez más robusta al ruido y por ende podrá ser transmitida a mayores distancias. La variación de estos parámetros influye además en la tasa de transmisión de datos en [bits/s] de forma tal que al aumentar el producto $BW * T$ la tasa de datos disminuirá reflejándose en la transmisión de payloads más pequeños. LoRa se beneficia de esto para poder tener las características de robustez anteriormente mencionadas [4].

Para la modulación en LoRa se fijaron tres anchos de banda (BW): 125 Khz, 250 Khz y 500 Khz, y además se fijaron los valores de los *time on air* posibles resumiéndolos en un factor llamado *Spreading Factor* (SF), que en LoRa puede tomar valores enteros entre 7 y 12. De la combinación de estos dos valores se obtienen diferentes tasas de transmisiones de datos dadas por la ecuación 1.1 [26].

$$R_b = SF * \frac{1}{\frac{2^{SF}}{BW}} (bits/s) \tag{1.1}$$

Como los valores de SF y BW son fijos, las tasas de datos también lo son, siendo resumidas en valores llamados Data Rates (DR). Para las diferentes regiones se usan diferentes combinaciones de SF y BW por lo que los DR son diferentes también. Para la norma que rige en Chile dada por la banda AU915 se tienen los valores de DR de la tabla 1.1.

Tabla 1.1: Tabla de Data Rates para AU915. Fuente [20]

Data Rate	**Configuración**	**bits/s**	**Max Payload [bytes]**
DR0	SF12/125KHz	250	51
DR1	SF11/125KHz	440	51
DR2	SF10/125KHz	980	51
DR3	SF9/125KHz	1760	115
DR4	SF8/125KHz	3125	222
DR5	SF7/125KHz	5470	222
DR6	SF8/500KHz	12500	222
DR7	Sin definir		
DR8	SF12/500kHZ	980	33
DR9	SF11/500kHZ	1760	109
DR10	SF10/500kHZ	3900	222
DR11	SF9/500kHZ	7000	222
DR12	SF8/500kHZ	12500	222
DR13	SF7/500kHZ	21900	222

1.3.2. Formato de mensaje de capa PHY

Los mensajes son clasificados según su dirección, *uplink* si es que van de nodo a servidor y *downlink* si es que van en dirección contraria.

Para esta capa, los formatos de los mensajes varían para *uplink* y *downlink*. Estos se pueden ver en las figuras 1.2 y 1.3 respectivamente. En ellos se pueden distinguir los siguientes campos:

- **Preamble:** Campo de sincronización
- **PHDR:** Header de capa física
- **PHDR_CRC:** Header de capa física del tipo *Cyclic Redundancy Check*
- **PHYPayload:** *Payload* de capa física
- **CRC:** *Cyclic Redundancy Check*

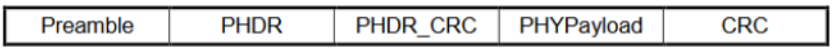


Figura 1.2: Formato de paquete de capa física, *uplink*. Fuente [27]

Dependiendo del *Spreading Factor* de la modulación elegido se tendrán diferentes tamaños para PHYPayload, los que pueden estar entre 33 y 222 bytes [16].

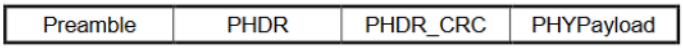


Figura 1.3: Formato de paquete de capa física, *downlink*. Fuente [27]

1.4. LoRaWAN

Se le llama LoRaWAN a la capa MAC, la encargada del control de acceso al medio, define el protocolo de comunicación y su arquitectura mientras la capa física LoRa habilita la comunicación a largas distancias. Esta parte de la tecnología es la que determina la duración de las baterías de cada nodo, la capacidad de la red, la calidad de servicio, la seguridad y otros cuantos servicios provistos por la red [31].

1.4.1. Arquitectura

La arquitectura definida por LoRaWan determina nodos terminales organizados en topología estrella alrededor de nodos *gateway* los cuales se organizan también en topología estrella en torno a los servidores, formando estrellas de estrellas. Los dispositivos terminales se comunican a través de LoRa con los *gateways* mientras que estos últimos se conectan a los servidores de red (o Internet) mediante una conexión IP estándar.

En la figura 1.4 se muestra la interacción entre las distintas partes de la red.

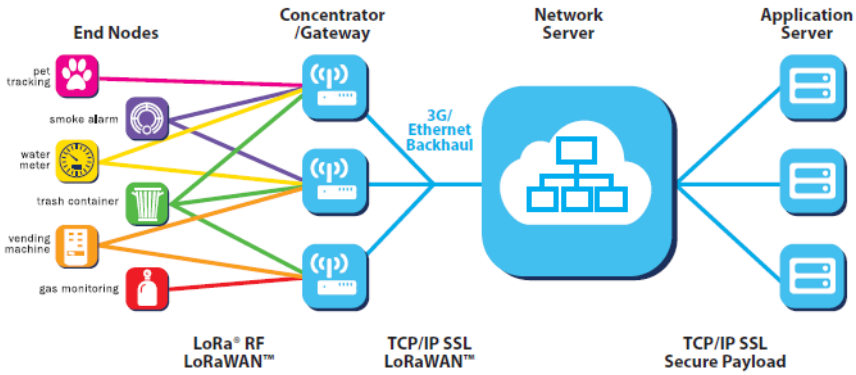


Figura 1.4: Arquitectura de red LoRaWAN. Fuente [31]

En esta red, el Network Server es un ente centralizado que recibe todos los mensajes provenientes de todos los nodos y aplicaciones y se encarga de redireccionarlos según corresponda ya que, dado que los nodos no manejan el protocolo IP, éstos son invisibles a las aplicaciones, que se entienden como cualquier servicio alojado en la Internet.

En el Network Server se deben registrar los nodos con sus respectivos identificadores a las aplicaciones que se quieran y es por él que la red puede mantenerse segura de extremo a extremo pues maneja las claves de encriptación de todas las sesiones.

1.4.2. Conceptos clave de la red

Para entender cómo se establecen las sesiones de comunicación entre aplicación y nodo es necesario explicar los siguientes conceptos .

Device EUI, Gateway EUI, Application EUI, DevAddr

Para la identificación de los dispositivos, red, aplicaciones y generación de claves de encriptación se tienen diferentes identificadores:

- **Device EUI:** Identificador único de los dispositivos otorgado por el fabricante. Tiene 64 bits de largo.
- **Gateway EUI:** Identificador único de los *gateways* otorgado por el Network Server. Tiene 64 bits de largo.
- **Application EUI:** Identificador único de la aplicación otorgado por el Network Server. Tiene 64 bits de largo.
- **DevAddr:** identificador único del dispositivo para una red específica. Asignado por el Network Server. Tiene 32 bits de largo.

Con los identificadores anteriormente descritos es posible identificar cada uno de los elementos de la red, sin embargo para establecer las sesiones es necesario conocer además las claves de encriptación y comprobación de mensajes. En algunos casos éstas son generadas por el Network Server en el momento de establecer la sesión directamente con los identificadores o en otros pueden ser generadas con anterioridad, en ambos casos, tanto dispositivos como Network Server deben conocerlas.

Network Session Key, Application Session Key, Application Key

Para la encriptación y comprobación de mensajes se establecen dos claves de 128 bits únicas para cada dispositivo y su respectiva sesión de comunicación.

- **Network Session Key:** Esta clave es usada para realizar el chequeo y validación de mensajes (MIC) entre nodo y Network Server, tambien sirve para mapear cada DevAddr con sus respectivos Device EUI y Application EUI.
- **Application Sesion Key:** Esta clave es usada para encriptar y desencriptar los mensajes enviados entre nodo y aplicación.

Además existe una tercera clave llamada **Application Key** que es usada para generar las dos claves anteriormente mencionadas en el caso de que éstas deban ser generadas automáticamente para cada sesión.

Dispositivos OTAA y ABP

Para establecer las sesiones entre nodo y aplicación existen 2 formas de autenticarse con la red: *Over-the-Air Activation (OTAA)* y *Activation by Personalization* (ABP).

- **OTAA:** En este método los usuarios deben ingresar manualmente en el dispositivo la Application Key, Application EUI y Device EUI. Cuando el dispositivo es encendido, iniciará un proceso en el cual negocia la Network Session Key y la Application Session Key con el Network Server que son generadas en el momento para aquella sesión en particular, además la red le asigna un DevAddr al nodo.
- **ABP:** En este método el usuario debe ingresar manualmente al dispositivo la Network Session Key, la Application Session Key y la DevAddr entregadas por el Network Server con anterioridad. Al momento de encender el dispositivo no es necesario negociar las claves por lo que el nodo puede empezar la transmisión de datos inmediatamente.

1.4.3. Clases LoRaWAN

LoRaWAN define 3 clases diferentes que pueden implementar los dispositivos. Todos los dispositivos deben implementar al menos la clase A. En adición ellos pueden implementar las clases B y C opcionales [27].

En la figura 1.5 se muestra el esquema general de las capas que se involucran en este protocolo.

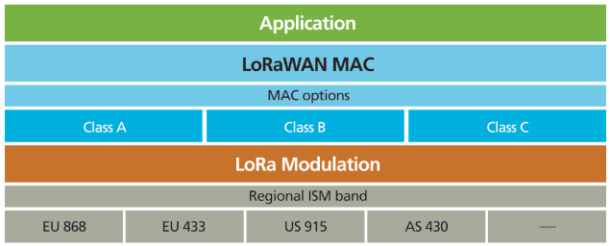


Figura 1.5: Esquema de capas involucradas en el protocolo LoRaWAN. Fuente [15]

A continuación se describirá cada una de las clases:

Dispositivos terminales bidireccionales (Clase A)

Los dispositivos terminales de la clase A permiten comunicación bidireccional donde cada ventana de transmisión del tipo *uplink* es seguida por dos ventanas cortas del tipo *downlink*. Los tiempos de las ventanas de recepción están dados por tiempos variables aleatorios (tipo ALOHA). Esta clase es la que presenta el menor consumo de potencia ya que sólo habilita el *downlink* luego de una transmisión *uplink*. Si el servidor necesita enviarle información al dispositivo éste deberá esperar a que se habilite otro ciclo *downlink*.

En la figura 1.6 se tiene un esquema que muestra cómo actúa esta clase.

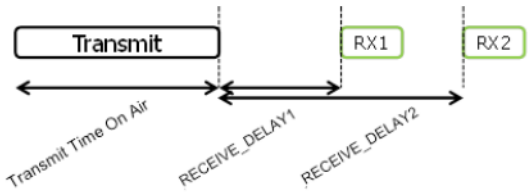


Figura 1.6: Esquema de transmisión para Clase A. Fuente [27]

Dispositivos bidireccionales con ventanas de tiempo programadas (Clase B)

Los dispositivos terminales de la clase B habilitan más ventanas de tiempo de transmisión del tipo *downlink*. En adición a las dos ventanas aleatorias de la clase A, la clase B habilita ventanas extras en tiempos programados.

Dispositivos bidireccionales con ventanas de recepción continuas (Clase C)

Los dispositivos terminales de la clase C mantienen continuamente habilitadas las ventanas de recepción siendo cerradas sólo al momento de transmitir. Estos dipositivos son los que más consumen energía pero son los que ofrecen la menor latencia de comunicación entre el servidor y el dispositivo.

1.4.4. Formato de mensaje de capa MAC

El *payload* del mensaje de la capa física (PHYPayload) mostrado anteriormente contiene a su vez el *payload* de la capa MAC y otra información adicional:

- **MHDR:** Header de capa MAC
- **MACPayload:** *Payload* de capa MAC
- **MIC:** *Message Integrity Check*

A su vez, el *payload* de la capa MAC (MACPayload) está compuesto por los siguientes campos:

- **FHDR:** Encabezado del *frame*
- **FPort:** Puerto del *frame*
- **FRMPayload:** *Payload* del *frame*

Finalmente, el encabezado del *frame* (FHDR) contiene la dirección del dispositivo terminal, específicamente en el campo DevAddr.

En la figura 1.7 se puede ver el desglose de los campos anteriormente mencionados.

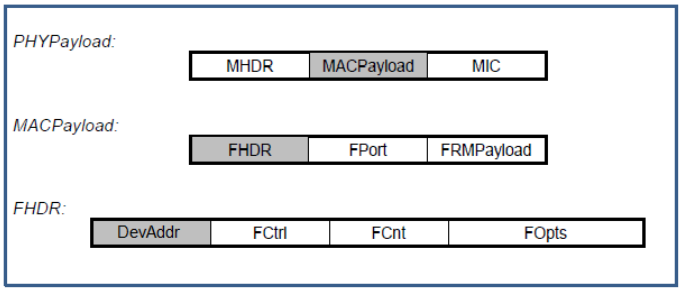


Figura 1.7: Formato de paquete de capa MAC desglosado. Fuente [27]

1.5. IPv6

IPv6 es la nueva versión del Internet Protocol (IP), protocolo de la capa de red, diseñado como sucesor de IP versión 4 (IPv4). Nace de la necesidad de disponer de una cantidad mayor de direcciones IP dado el crecimiento en la cantidad de dispositivos conectados que aumentó aún mas con la aparición del IoT.

Los cambios hechos a IPv4 para pasar a ser IPv6 pueden ser enmarcados en alguna de las siguientes categorías [17]:

- **Expansión de capacidad de direccionamiento:** Incremento del tamaño de una dirección IP de 32 bits a 128 bits.
- **Simplificación del formato del header:** Algunos campos del header IPv4 se eliminaron o se hicieron opcionales para así reducir el tamaño de los paquetes.
- **Mejora en el soporte de extensiones y opciones:** Cambios en la codificación de las opciones del header IP para facilitar la adición de éstas y eliminar las limitaciones.
- **Etiquetado de flujos de paquetes:** Se agregó la capacidad de etiquetar paquetes en secuencia para ser tratados como un solo flujo.
- **Capacidad de autenticación y privacidad:** Se agregaron extensiones para soportar autenticación, integridad de datos y confidencialidad.

1.5.1. Formato de encabezado

El encabezado consiste en una estructura de 40 bytes de tamaño mínimo y contiene la información indispensable del paquete. Estos son la version del protocolo (6 en este caso), el tamaño del *payload*, la dirección de origen, la de destino, entre otros. En la figura 1.8 se tiene una descripción básica del encabezado.

Además del encabezado original, IPv6 da la opción de agregar encabezados para el uso de extensiones, la identificación del encabezado siguiente va en el campo Next Header. Una de estas extensiones es la fragmentación de paquetes.

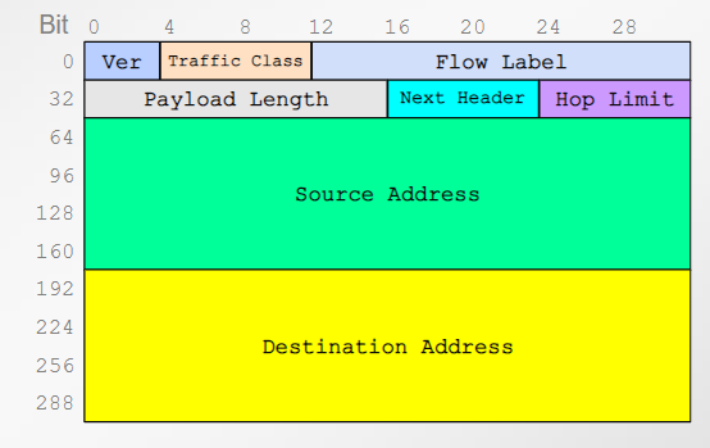


Figura 1.8: Formato de encabezado IPv6.

1.5.2. Fragmentación

La fragmentación en IPv6 a diferencia de IPv4 es realizada directamente por el nodo de origen y no por los enrutadores de la red. Esta se lleva a cabo sólo si la unidad máxima de transferencia (MTU) de los nodos en cuestión es menor a 1280 bytes [17], algo común para los dispositivos que usan tecnologías LPWAN pues ya se vió que el tamaño del *payload* LoRa tiene una extensión máxima de 222 bytes.

Para efectuarla, en primer lugar se toma el paquete original identificando 2 partes dentro de él, la parte infragmentable correspondiente al header IPv6 mas los headers de extensión si es que existen y la parte fragmentable que consiste al resto del paquete luego de todos los headers. Hecho esto se procede a dividir la parte fragmentable en partes mas pequeñas para luego montarla en otros paquetes manteniendo la parte infragmentable del paquete original agregando además un header de fragmentación que identifica la posición del *payload* del paquete dentro del paquete original y otra información útil para el reensamblaje. En la figura 1.9 se tiene una descripción gráfica de este proceso.

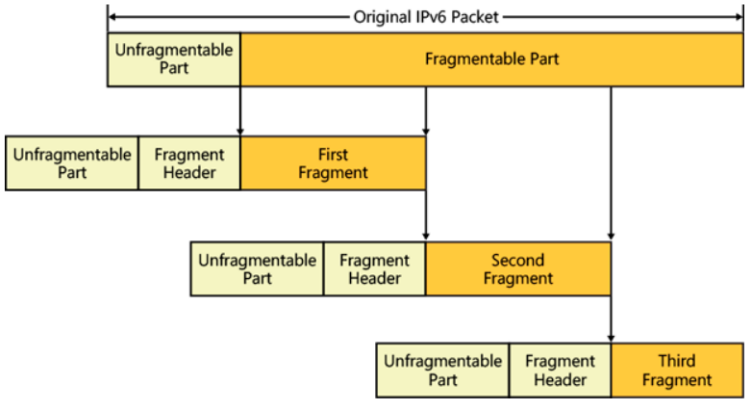


Figura 1.9: Esquema de fragmentación IPv6. Fuente [14]

.

Para el proceso de reensamblaje se debe aclarar que los paquetes fragmentados pueden

llegar en cualquier orden, siendo importante esperar que lleguen todos ellos para luego ordenarlos. Esto en un tiempo máximo de 60 segundos. Si han transcurrido 60 segundos y aún no llegan todos los paquetes, los paquetes son descartados y se envía un mensaje ICMP Time Exceeded al nodo de origen.

Luego de que todos los paquetes llegaron a su destino, se calcula el tamaño del *payload* original usando la información de todos los headers y luego se unen siguiendo el esquema de la figura 1.10.

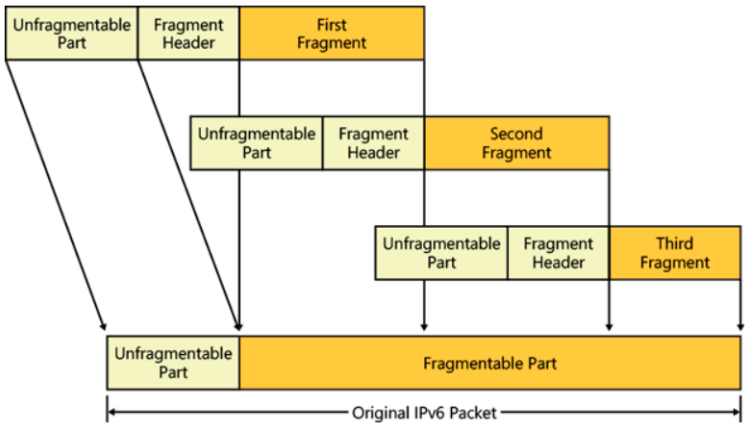


Figura 1.10: Esquema de reensamblaje de paquetes IPv6 fragmentados. Fuente [14].

1.6. Adaptaciones de IPv6 sobre redes restringidas

El protocolo IPv6 es una solución aceptable a los problemas actuales que se presentan en el ambiente del IoT, uno de ellos es la seguridad. La seguridad en las redes se hará fundamental en un futuro donde será común la existencia de *Smart Cities* (ciudades inteligentes) y *Smart Houses* (casas inteligentes), donde la mayoría de los sensores se comuniquen inalámbricamente. Un ciber-ataque a estos sistemas podría traducirse en grandes catástrofes, paralizando industrias y ciudades. El problema de escalabilidad también es solucionado al disponer de una cantidad mucho mayor de direcciones IP para todos los dispositivos que deseen conectarse. Finalmente, se podrá mejorar considerablemente la conectividad al simplificar la arquitectura de las redes existentes [1].

Las redes restringidas, particularmente las LPWAN, soportan paquetes pequeños por lo que la implementación de IPv6 sobre ellas no se puede efectuar de forma directa. En particular, el protocolo LoRaWAN soporta payloads de 222 bytes máximo además de los 8 bytes que ocupa el header de la capa MAC [16] siendo imposible que quepa dentro de él un paquete IPv6 cuyo tamaño mínimo es de 1280 bytes. Además se tiene que el header de IPv6 tiene un tamaño mínimo de 40 bytes, ocupando gran parte del tamaño del paquete LoRa y dejando muy pocos bytes para los protocolos de capas superiores, como por ejemplo UDP.

Debido a lo anterior se han diseñado e implementado técnicas para disminuir el tamaño de los datagramas IPv6 a transmitir y que estas redes restringidas los soporten. Una de

estas implementaciones es la realizada para las *Low Power Personal Area Networks* (LoWPAN) llamada 6LoWPAN [23]. A la fecha del desarrollo de este documento el estándar para LoRaWAN aún se encuentra en desarrollo por la IETF [22].

1.6.1. 6LoWPAN

LoWPAN al igual que LoRaWAN es una tecnología bajo el estándar IEE 800.15.4 que se caracteriza por su bajo consumo de potencia pero con la diferencia de que las comunicaciones son de corto alcance. El tamaño de los paquetes que soporta es de un máximo de 127 bytes disminuyendo a 81 bytes efectivos cuando no se cuentan los bytes del header de la capa MAC [23], alejándose bastante de los 1280 bytes necesarios para un paquete IP.

Para su implementación, en primer lugar se definen direcciones mas cortas (de 16 o 64 bits) para establecer el link entre los dispositivos terminales y el *gateway* de la red personal (*Personal Area Network*, PAN). Luego se define una capa de adaptación con un nuevo formato de paquete. El nuevo formato de paquete elimina algunos contenidos del *header* original IPv6 y sólo deja los campos para ruteo, saltos, fragmentación, el *payload* y agrega *headers* opcionales para indicar si el ruteo es para una red del tipo mesh, otro para indicar si el mensaje es del tipo *broadcast* y finalmente uno para indicar fragmentación de paquetes.

Luego, disminuidos los campos del *header* IPv6 se procede a comprimirlo. Para la compresión se asume que los dispositivos que comparten la misma red también comparten algún tipo de estado, es decir, comparten algunos campos del *header* IPv6. Además de eliminar ciertos parámetros se infieren otros desde los campos de los *headers* de otras capas y así eliminar algunos bits de información redundante. El único campo que no puede ser modificado y debe ser llevado completamente es el de límite de saltos.

Finalmente, si el paquete sigue siendo demasiado grande para el *payload*, se procede a realizar la fragmentación de la misma forma como se hace la fragmentación estándar de paquetes IPv6 vista anteriormente.

1.7. 6LoRaWAN

Para la implementación de IPv6 sobre LoRaWAN la IETF se encuentra trabajando en un documento borrador titulado *Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6* que luego será convertido en estándar [22]. Este documento sugiere una implementación, que similar a 6LoWPAN, se basa en una capa de adaptación que comprime el *header* IPv6 para luego aplicar la fragmentación de paquetes en caso de que el paquete comprimido siga siendo demasiado grande.

La implementación de la compresión busca aprovechar las siguientes características de las LPWAN:

- La tipología tipo estrella de estas redes implica que todos los paquetes siguen el mismo

camino.

- El tráfico de paquetes se puede saber de antemano dependiendo de la aplicación.

Aprovechándose de aquellas características la etapa de compresión ocupa el algoritmo *Static Context Header Compression* (SCHC), el cual propone comprimir el *header* IPv6 según el contexto de comunicación. En las LPWAN el contexto de comunicación no varía, por lo que varios campos del *header* IPv6 permanecen constantes en el tiempo. Este tipo de compresión se basa en la naturaleza predecible de los paquetes circulantes de una red LPWAN y actúa omitiendo información que se presume conocida para ambas partes de la comunicación para el contexto elegido. El parámetro Rule ID es el que selecciona el contexto a utilizar para luego aplicar la compresión/descompresión según la regla correspondiente a ese contexto, esta regla debe ser conocida por ambas partes de la comunicación, emisor y receptor [22].

La etapa de fragmentación se realiza siempre después de la etapa de compresión por lo que siempre actúa sobre un paquete bajo el efecto del algoritmo SCHC, siendo necesario que almacene en el *header* de fragmentación la Rule ID de compresión y el número de paquete comprimido además de habilitar la retransmisión de paquetes para aumentar la fiabilidad de las transmisiones, obligando a agregar parámetros para manejar un temporizador de retransmisión. Al igual que para la fragmentación de paquetes IPv6 vista anteriormente, esta fragmentación establece un *header* de fragmentación para cada fragmento, los campos principales de este *header* son:

- **Rule ID:** Es la identificación que señala el tipo de contexto en que el paquete fue comprimido.
- **Fragment Compressed Number (FCN):** Número que sirve para representar la posición del fragmento para su reensamblaje, pero no corresponde exactamente al número de fragmento. Sirve para denotar si un fragmento es el último del total de fragmentos enviados o si es el último de una ventana de fragmentos.
- **Datagram Tag (DTag):** Los paquetes fragmentados que tengan igual DTag tienen el mismo paquete de origen, mientras que paquetes con diferentes DTag vienen de paquetes originales diferentes.
- **Window (W):** Este número lleva el mismo valor para todos los paquetes de una misma ventana de transmisión.

Como este algoritmo de fragmentación requiere implementar la retransmisión de paquetes, éste define 3 niveles de confiabilidad:

- **No-ACK:** El receptor no genera mensajes de *acknowledgment* (ACK) para los mensajes fragmentados.
- **ACK-Always:** El receptor genera mensajes ACK luego de recibir una ventana de paquetes fragmentados menor a la cantidad de fragmentos totales a recibir. De esta forma es posible retransmitir fragmentos antes de enviar la totalidad de estos.
- **ACK-no-Error:** El receptor genera un mensaje ACK al recibir todos los fragmentos del paquete o si es que al menos uno de los fragmentos se perdió.

Información mas detallada sobre la fragmentación de paquetes sobre LoRaWAN será expuesta en el capítulo 2.

1.8. Estado actual de implementaciones de 6LoRaWAN

Las implementaciones que se tienen hasta el momento de 6LoRaWAN sólo aplican la capa de adaptabilidad en su primera etapa, es decir la etapa de compresión de paquetes usando SCHC. En [25] y [24] se tienen implementaciones basadas en el IETF Draft usando UDP en la capa de transporte en módulos dedicados usando su propia implementación en código, mientras que en [30] se realiza una implementación similar pero usando el sistema operativo Contiki OS. Finalmente se tiene la implementación realizada por Nicolás Maturana Araneda en su Trabajo de Memoria de Título [21] en el cual se implementó la capa de compresión en Python para dispositivos propietarios de Everynet. En estos casos la capa de fragmentación no fue implementada.

En cuanto al grupo de trabajo *IPv6 over Low Power Wide-Area Networks* de la IETF se tienen implementaciones en Python del algoritmo SCHC resultantes de juntas masivas de programación realizadas el año 2018 y 2019 ([2] y [3]). Actualmente, este grupo de trabajo se encuentra trabajando en la recopilación del código para obtener la implementación definitiva de SCHC en la librería OpenSCHC [11] mientras que la implementación de la fragmentación aún se encuentra pendiente por integrar a la capa de adaptabilidad propuesta por el IETF Draft.

Capítulo 2

Fragmentación/Reensamblaje

Este capítulo describe el funcionamiento de los algoritmos de fragmentación y reensamblaje (F/R) usados en la implementación de la capa de adaptación para el soporte de mensajes IPv6 en redes LoRaWAN. La capa de Compresión/Descompresión (C/D) será omitida ya que se escapa de los alcances de este trabajo.

La información plasmada en este capítulo resultó de la interpretación de la versión 21 del documento IETF Draft que aborda la implementación de SCHC F/R para redes LPWAN [22] y la versión 3 del documento IETF Draft que especifica su uso para las redes LoRaWAN [28].

2.1. Static Context Header Compression (SCHC)

El algoritmo *Static Context Header Compression* (SCHC) consiste en una capa de adaptación entre IPv6 y las tecnologías LPWAN para habilitar el soporte del protocolo IP en este tipo de redes. SCHC consta de dos subcapas, capa C/D y capa F/R, como se ve en la figura 2.1.

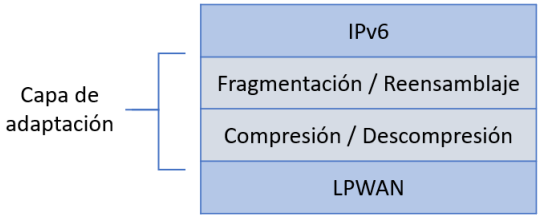


Figura 2.1: Esquema de capas para SCHC

Antes de que un paquete sea transmitido se aplica la compresión de su *header*. El resultado es un paquete SCHC sin importar si la compresión fue realizada con éxito o no. Luego, el paquete pasa por una capa de fragmentación opcional para el caso en que el paquete no supere el máximo establecido por el enlace, pero necesaria para los casos en que el enlace

deba cubrir el requerimiento de 1280 bytes máximos que solicita la implementación de IPv6. Las operaciones inversas toman lugar en el receptor. Este proceso se ve ilustrado en la figura 2.2.

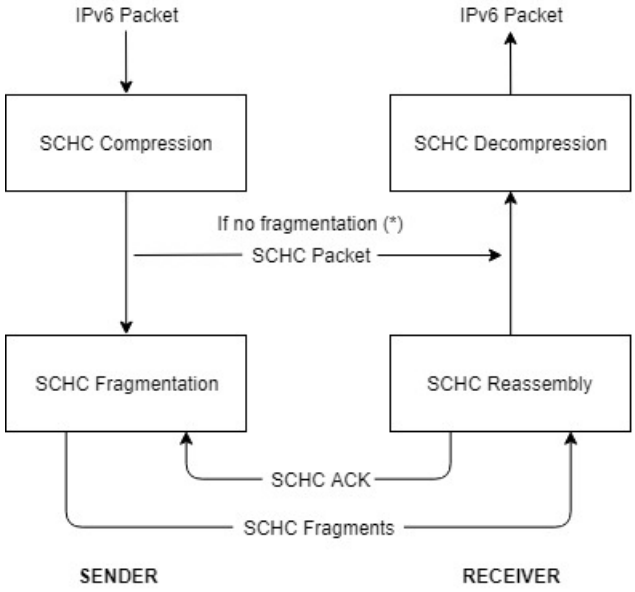


Figura 2.2: Operaciones en el receptor y emisor para SCHC. Fuente [22]

Para los paquetes que necesiten ser fragmentados se les asignará una Rule ID y un perfil de fragmentación que identificarán al paquete como un paquete SCHC F/R y le asignarán los parámetros necesarios para poder realizar el reensamblaje en el lado del receptor. Esos parámetros corresponden a opciones que determinarán los valores y tamaños en bits de los campos del encabezado (*header*) que se le agregará a cada fragmento antes de ser transmitido y otros datos como los valores de los contadores y temporizadores que se utilizarán en el proceso de transmisión.

Las especificaciones del algoritmo de F/R incluyen además la implementación de 3 modos de operación que habilitan diferentes opciones de confiabilidad como por ejemplo la opción de retransmitir fragmentos perdidos. Estos modos de operación serán explicados más adelante en el documento.

Para la transmisión de mensajes se emplea un tamaño mínimo de símbolo (L2 *word size*), siendo de 1 octeto (8 bits) para el caso del protocolo LoRaWAN, por lo que todos los fragmentos transmitidos deben ser de un tamaño múltiplo de L2 *word size*. Si es que esto no ocurriese se deben agregar bits de relleno al final del paquete para que, al momento de fragmentar, se obtengan fragmentos con el tamaño adecuado.

2.2. Elementos del protocolo SCHC F/R

Para habilitar la funcionalidad del algoritmo de F/R se definieron diferentes elementos: mensajes SCHC F/R, *tiles*, ventanas, bitmaps, contadores, temporizadores y campos del

encabezado. Estos elementos se aplican de diferente forma para cada modo de operación por lo que en esta sección se hará sólo una descripción general de cada uno.

2.2.1. Mensajes SCHC F/R

El algoritmo define los siguientes tipos de mensajes [22]:

- **SCHC Fragment:** mensaje que porta parte del paquete SCHC desde el emisor al receptor.
- **SCHC ACK:** mensaje de confirmación desde el receptor al emisor. Este mensaje se usa para indicar la recepción de los fragmentos.
- **SCHC ACK REQ:** solicitud del emisor al receptor del envío de un SCHC ACK.
- **SCHC Sender-Abort:** mensaje en el cual el emisor le dice al receptor que ha abortado la transmisión de un paquete SCHC fragmentado.
- **SCHC Receiver-Abort:** Mensaje en el cual el receptor le dice al receptor que aborte la transmisión de un paquete SCHC fragmentado.

2.2.2. Tiles

El paquete SCHC es fragmentado en diferentes piezas llamadas *tiles* (figura 2.3). Estas deben estar no vacías y unidas deben ser iguales al paquete SCHC. Cada mensaje SCHC Fragment que incluya el campo de *payload* lleva como carga al menos un *tile*.

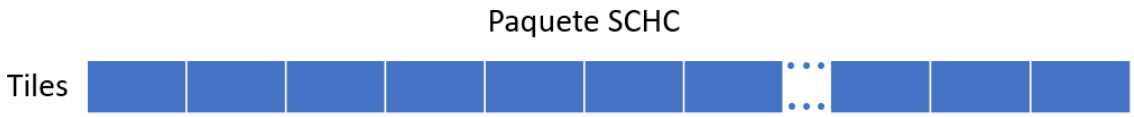


Figura 2.3: Tiles de un paquete SCHC

2.2.3. Ventanas

Algunos modos de F/R deben manejar el envío de *tiles* sucesivos, llamados ventanas. Si las ventanas son usadas:

- Todas las ventanas excepto la última deben contener el mismo número de *tiles*. Este número debe ser WINDOW_SIZE.
- El valor de WINDOW_SIZE debe ser especificado en el perfil.
- Las ventanas deben estar numeradas.
- Los *tiles* deben ser numerados por cada ventana.
- Cada *tile* será identificado por su número de ventana y su índice en ella. Como se puede ver en la figura 2.4

Figura 2.4: Enumeración de *tiles* y ventanas

Cuando se utilicen ventanas:

- Se deberán transmitir Bitmaps desde el receptor al emisor en un mensaje SCHC ACK.
- Un Bitmap corresponde exactamente con una ventana.

2.2.4. Bitmaps

Cada bit en el Bitmap para una ventana corresponde a un *tile* de esa ventana, por lo tanto, cada Bitmap tiene un tamaño de WINDOW_SIZE bits. Un bit del Bitmap identifica la recepción de un *tile* de la ventana enviada por lo que su uso se justifica en la necesidad del emisor de saber si es que los fragmentos enviados fueron recibidos correctamente por el emisor o si se perdieron en el proceso de transmisión. En el caso de pérdidas se utilizaría esta información para informar al emisor sobre cuáles son los fragmentos que deberían ser retransmitidos para completar la recepción de la ventana.

En el receptor:

- Un bit fijado en 1 en el Bitmap indica que el *tile* asociado a esa posición de bit ha sido correctamente recibido.
- Un bit fijado en 0 en el Bitmap indica que el *tile* asociado a esa posición de bit NO ha sido correctamente recibido.

2.2.5. Temporizadores y contadores

Algunos modos de F/R pueden usar los siguientes temporizadores y contadores:

- **Temporizador de Inactividad (Inactivity Timer):** el receptor usa este temporizador para abortar el proceso de espera de mensajes SCHC F/R.
- **Temporizador de Retransmisión (Retransmission Timer):** el emisor usa este temporizador para abortar la espera de un SCHC ACK por parte del receptor.
- **Contador de Intentos (Attempts):** contador usado por el emisor para contar el número de SCHC ACK REQ enviadas al receptor. Este número puede llegar a un máximo de MAX_ACK_REQUEST definido en el perfil de F/R.

2.2.6. Integrity Checking

El paquete reensamblado debe pasar por un proceso de chequeo al final del proceso de reensamblaje en el receptor. Por defecto, el *integrity checking* se realiza computando un *Message Integrity Check* (MIC) sobre el paquete original en el lado del emisor para luego ser transmitido al receptor y comparar su valor con un MIC calculado en el lado del receptor sobre el paquete reensamblado. Estos cálculos deben ser efectuados sobre el paquete que incluye los bits de relleno agregados al final de este. El IETF Draft recomienda el uso del algoritmo CRC32 con el polinomio 0xEDB88320, sin embargo, se puede usar cualquier otro MIC que se quisiera.

2.2.7. Campos del encabezado

Para el correcto reensamblaje de paquetes por parte del receptor es necesario agregar a cada paquete un encabezado que le proporcione al receptor la información necesaria para unir los paquetes en el orden adecuado. De esta forma se definieron los campos siguientes.

Rule ID

Campo presente en todos los mensajes SCHC F/R. Se usa para identificar que un mensaje SCHC F/R está siendo transmitido.

Datagram Tag (DTag)

Campo utilizado para informar al receptor sobre los fragmentos que corresponden con una Rule ID. Su tamaño (T en bits) esta definido en cada perfil para cada Rule ID y éste establece 2 tipos de comportamientos para el proceso de transmisión:

- Cuando T es 0, el campo DTag no aparecerá en los mensajes SCHC F/R y se le asignará un valor de 0 para emisor y receptor. Cuando esto ocurre, sólo puede haber un paquete SCHC en proceso de transmisión en ese momento para aquella Rule ID.
- En el caso de que T sea distinto de 0, el campo DTag debe ser fijado con el mismo valor para todos los mensajes SCHC F/R relacionados al mismo paquete SCHC fragmentado y debe ser diferente entre mensajes que estén relacionados a diferentes paquetes SCHC. De esta forma se podrá tener en tránsito diferentes fragmentos para diferentes paquetes fragmentados.

Window (W)

Este campo es opcional. Solo se presenta si en el modo de F/R se ocupan ventanas. Lleva la información de la ventana a la que pertenece cada *tile* por lo que para *tiles* de una misma

ventana el campo W debe ser idéntico.

Su tamaño (M en bits) lo define cada modo de F/R y cada perfil para cada Rule ID. Dependiendo del modo de operación y el perfil, W puede llevar el número completo de la ventana o una representación parcial de su número, como por ejemplo llevar sólo el valor del bit menos significativo de éste.

Fragment Compressed Number (FCN)

Este campo lleva la información sobre el progreso en la secuencia de *tiles* siendo transmitidos por los SCHC Fragments. Por ejemplo, puede contener una representación parcial y eficiente del número de *tile* siendo transmitido. La descripción del uso exacto de este campo se deja para cada modo de F/R. Sin embargo, se tienen reservados 2 valores que se ocupan para el control del proceso de F/R, estos son:

- El valor de FCN con todos sus bits en 1 (llamado All-1) identifica al último *tile* del paquete SCHC. Si se ocupan ventanas, la última ventana de un paquete es llamada Ventana All-1.
- Si se usan ventanas, el valor de FCN con todos sus bits en 0 (llamado All-0) identifica el último *tile* de una ventana que no es la última del paquete. Por extensión, aquella ventana es llamada Ventana All-0.

Message Integrity Check (MIC)

Este campo sólo aparece en el fragmento All-1 (el último del paquete). Su tamaño (U en bits) es definido por cada perfil para cada Rule ID.

Integrity Check (C)

Este campo aparece en un mensaje SCHC ACK para reportar el chequeo de integridad del paquete reensamblado por el receptor. Un valor de 1 señala que el chequeo fue realizado con éxito, mientras que un valor de 0 señala que el chequeo, o no se realizó o falló.

Compressed Bitmap

El Bitmap Compreso es usado junto con las ventanas y los Bitmaps. Su presencia y tamaño es definida por cada modo de F/R para cada Rule ID.

Este campo aparece en los mensajes SCHC ACK para reportar el estado del Bitmap, y entonces, el estado de los paquetes recibidos en el receptor.

2.3. Formatos de mensajes SCHC F/R

Esta sección describe los formatos de los diferentes mensajes nombrados anteriormente en la sección 2.2.1.

2.3.1. Formato de SCHC Fragment general

Este tipo de mensaje tiene el formato que se ve en la figura 2.5. Consta de un encabezado y un payload en el cual pueden ir uno o más *tiles*.

Figura 2.5: Formato general de SCHC Fragment

2.3.2. Formato de SCHC Fragment regular

Un SCHC Fragment regular tiene el formato que se ve en la figuran 2.6. Este tipo de mensajes se usa para transportar *tiles* que no son la última *tile* del paquete. Los campos DTag y W son opcionales.

Figura 2.6: Formato detallado de SCHC Fragment

El campo FCN no debe tener todos sus bits en 1 ya que en ese caso estaría portando el último *tile* del paquete convirtiéndose en un fragmento All-1.

2.3.3. Formato de SCHC Fragment All-1

El fragmento All-1 se muestra en la figura 2.7. Este mensaje es enviado por el emisor para completar la emisión de un paquete SCHC fragmentado. Los campos DTag, W, MIC y Payload son opcionales, sin embargo, el mensaje debe tener al menos el campo MIC o el campo Paylaod. Todos los bits del campo FCN están en 1.

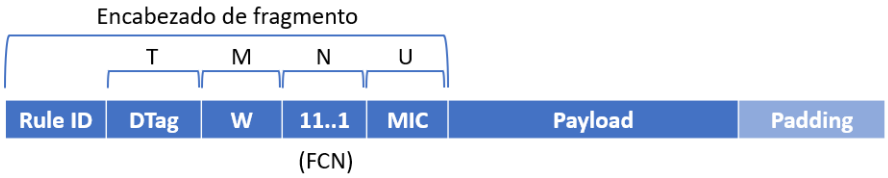


Figura 2.7: Formato de mensaje All-1

2.3.4. Formato de SCHC ACK

El formato del mensaje SCHC ACK se muestra en la figura 2.8. Los campos Dtag, W y Compressed Bitmap son opcionales. El campo Compressed Bitmap solo puede estar presente en los modos F/R que usen ventanas.

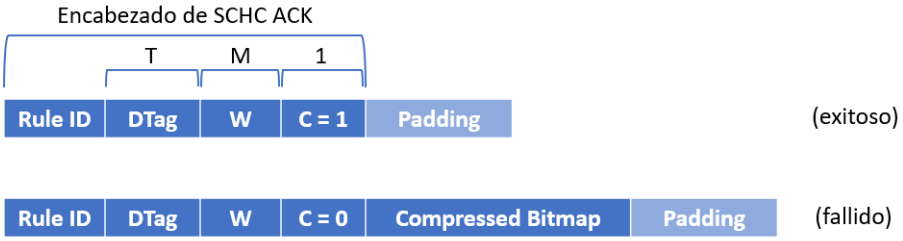


Figura 2.8: Formato de mensajes SCHC ACK para Integrity Check exitoso y fallido

El encabezado de este mensaje contiene el bit C, que señala o no el éxito del Integrity Check. Si C es 1 (*Integrity Check* exitoso), el Bitmap no es enviado. Si C es 0 (*Integrity Check* no realizado o fallido) y se está en un modo que ocupe ventanas, se enviará un Bitmap comprimido (Compressed Bitmap) para la ventana referida en el campo W.

Compresión de Bitmap

Para su transmisión, el Bitmap comprimido en el mensaje SCHC ACK se define con el siguiente algoritmo:

- Construir un mensaje SCHC ACK temporal cuyo encabezado contiene el bitmap original.
- Posicionar un cursor al final del Bitmaps, después del último bit.
- Mientras que el bit a la izquierda del cursor sea 1 y pertenezca al Bitmap, mover el cursor hacia la izquierda, luego parar.
- Mientras el cursor no esté en una posición tal de que el mensaje SCHC ACK tenga un tamaño múltiplo del tamaño L2 *word size*, mover el cursor hacia la derecha, luego parar.
- Finalmente, eliminar todos los bits que se encuentren a la derecha del cursor.

Cuando se eliminen 1 o más bits del mensaje SCHC ACK como resultado del algoritmo, se tendrá que su tamaño sera múltiplo del tamaño L2 *word size* por lo que no será necesario

agregar bits de relleno. Si ocurre el caso de que no se han eliminado bits del Bitmap debido a que no se alcanzó el tamaño múltiplo del tamaño L2 *word size*, se deberán agregar bits de relleno.

Como el emisor de fragmentos conoce el tamaño original del Bitmap, podrá reconstruirlo desde el Compressed Bitmap recibido.

2.3.5. Formato de mensaje SCHC ACK REQ

El formato de este mensaje se muestra en la figura 2.9. Los campos Dtag y W son opcionales. El campo FCN tiene todos sus bits en 0.

Figura 2.9: Formato de mensaje SCHC ACK REQ

Este mensaje se diferencia del fragmento All-0 en que el mensaje SCHC ACK REQ no presenta el campo Payload.

2.3.6. Formato de mensaje SCHC Sender-Abort

El formato de este mensaje se muestra en la figura 2.10. Los campos Dtag y W son opcionales mientras que el campo FCN tiene todos sus bits en 1.

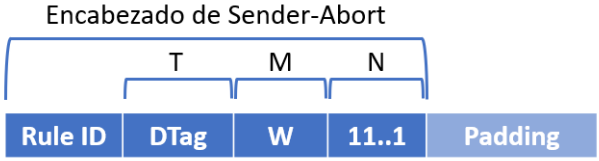


Figura 2.10: Formato de mensaje SCHC Sender-Abort

Si el campo W está presente:

- El emisor debe fijar todos los bits de este campo en 1. Otros valores están reservados.
- El receptor debe chequear su valor. Si su valor es diferente al de todos los bits en 1, el mensaje debe ser ignorado.

El mensaje SCHC Sender-Abort no debe ser contestado con un mensaje SCHC ACK desde el receptor.

Este mensaje se diferencia del mensaje All-1 en que el mensaje Sender-Abort no contiene ni el campo MIC ni el campo Payload.

2.3.7. Formato de SCHC Receiver-Abort

El formato de este mensaje se muestra en la figura 2.11. Los campos DTag y W son opcionales.

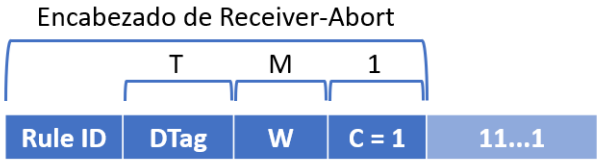


Figura 2.11: Formato de mensaje SCHC Receiver-Abort

Si el campo W está presente:

- El receptor debe fijar todos sus bits en 1. Otros valores están reservados.
- El emisor debe chequear que todos los bits de W sean 1, en caso contrario deberá ignorar el mensaje.

Este mensaje tiene el mismo encabezado que el mensaje SCHC ACK por lo que para diferenciarlos se debe modificar el mensaje Receiver-Abort con lo siguiente:

- Si el *header* NO es de tamaño múltiplo al tamaño L2 *word size*, se deben agregar bits fijados en 1 hasta que se cumpla la condición.
- Si el *header* es de tamaño múltiplo al tamaño L2 *word size*, se debe agregar exactamente un símbolo de tamaño L2 *word size* con todos sus bits fijados en 1.

El mensaje SCHC Receiver-Abort no debe ser contestado con un mensaje SCHC ACK desde el emisor.

2.4. Modos de F/R

El IETF Draft de SCHC para LPWAN especifica 3 tipos de modos de operación los cuales son:

- **No-ACK**
- **ACK-Always**
- **ACK-on-Error**

2.4.1. Modo No-ACK

En este modo no existe comunicación entre el emisor de fragmentos y el receptor. El emisor transmite todos los SCHC fragments sin esperar un mensaje ACK por lo que no se utilizan

los mensajes SCHC ACK, SCHC ACK REQ ni tampoco el mensaje SCHC Receiver-Abort que va de receptor a emisor.

Los *tiles* pueden no ser de tamaño uniforme. No se usan ventanas ni el temporizador de retransmisión. Tampoco se usa el contador de intentos. Como no se usan ventanas, el campo W no aparece en los encabezados de los mensajes y tampoco se ocupan los mensajes All-0. En resumen, sólo se utilizan los mensajes SCHC Fragments incluyendo el mensaje All-1, los mensajes SCHC Sender-Abort y el temporizador de inactividad por parte del receptor.

Cada perfil debe especificar qué valores de Rule ID corresponden con este modo de operación además de los siguientes valores:

- El tamaño del campo DTag.
- El tamaño y el algoritmo para el campo MIC.
- El tiempo de expiración del temporizador de inactividad.

Por cada par de valores Rule ID y DTag, el receptor debe mantener un temporizador de inactividad.

Comportamiento de emisor

Al inicio de la fragmentación, el emisor debe seleccionar un valor de Rule ID y un DTag para el paquete SCHC a fragmentar. Cada SCHC Fragment debe contener exactamente un *tile* en su Payload, con cada *tile* de un tamaño mínimo de L2 *word size*. El transmisor debe enviar los SCHC Fragments en el orden en que las *tiles* se encuentran en el paquete SCHC realizando una transmisión ciega. Todos los SCHC Fragments deben tener el formato regular a excepción del último fragmento, que debe ser del tipo All-1.

El emisor puede transmitir un SCHC Sender-Abort en caso de que sea necesario.

En la figura 2.12 se tiene un diagrama que describe el comportamiento del emisor.

Comportamiento de receptor

Al recibir un SCHC Fragment regular el receptor debe:

- Reiniciar el temporizador de inactividad.
- Ensamblar los Payloads de los fragmentos.

Al recibir un fragmento All-1, el receptor debe:

- Agregar el Payload del fragmento a los Payloads de los fragemntos anteriormente recibidos.
- Realizar el chequeo de integridad del paquete.
- Si el chequeo falla, el receptor debe desechar el paquete reensamblado.

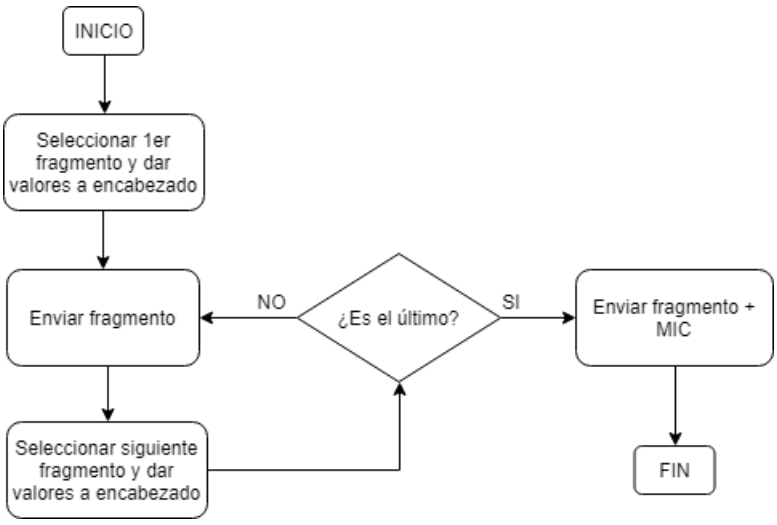


Figura 2.12: Diagrama que muestra el comportamiento del emisor en el modo No-ACK. Adaptado de [22]

- Terminar la operación de reensamblado.

Al expirar el temporizador de inactividad o recibir un mensaje SCHC Sender-Abort, el receptor debe desechar el paquete siendo reensamblado.

En la figura 2.13 se tiene un diagrama que describe el comportamiento del receptor.

2.4.2. Modo ACK-Always

En este modo se ocupan ventanas, además los *tiles* no deben ser necesariamente de tamaño uniforme. Un mensaje ACK es transmitido desde receptor a emisor luego de que el emisor ha transmitido cada ventana. Cada mensaje ACK debe contener el Bitmap correspondiente a la ventana recibida. En este modo se utilizan todos los elementos descritos en la sección 2.2.

De forma superficial, el algoritmo utilizado en este modo se puede describir como sigue: Despues de la transmisión ciega de la primera ventana, el emisor de fragmentos itera transmitiendo los fragmentos faltantes luego de recibir el mensaje ACK que contiene el Bitmap de la ventana que fue transmitida hasta que todos los fragmentos de la ventana hayan sido recibidos correctamente o hasta que se hayan hecho demasiados intentos de retransmisión. En el caso de que la ventana haya sido recibida completamente, el emisor pasa a la siguiente ventana repitiendo el proceso. Finalmente, luego de enviar el último fragmento de la última ventana el emisor debe esperar un mensaje ACK con la respuesta del emisor, la cual dirá si el mensaje se pudo reensamblar correctamente.

Cada perfil debe especificar qué valores de Rule ID corresponden con este modo de operación además de los siguientes valores:

- El tamaño del campo FCN N.
- El tamaño de WINDOW_SIZE, que debe ser estrictamente menor a 2^N.

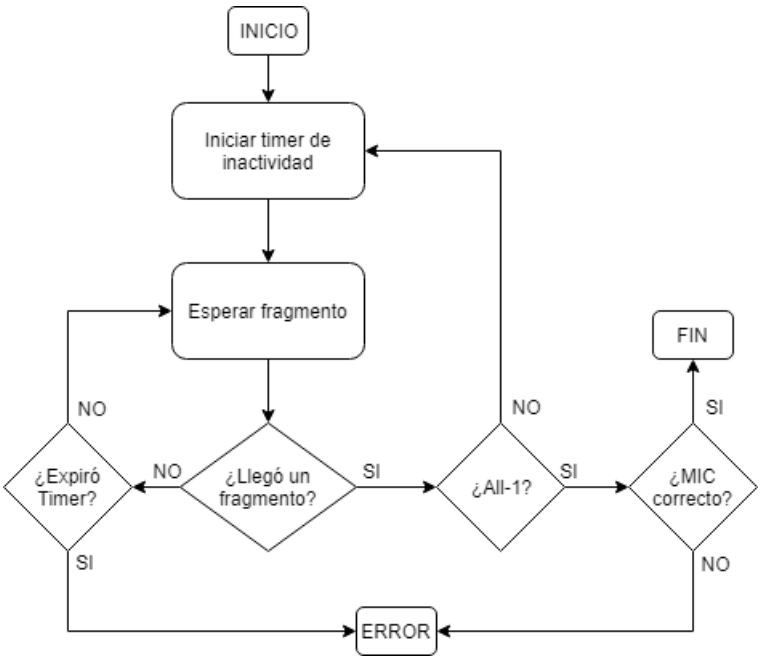


Figura 2.13: Diagrama que muestra el comportamiento del emisor en el modo No-ACK. Adaptado de [22]

- El tamaño y el algoritmo para el campo MIC.
- El tamaño del campo DTag.
- El valor de MAX_ACK_REQUESTS.
- El tiempo de expiracion para el temporizador de retransmisión.
- El tiempo de retransmisión para el temporizador de inactividad.

Por cada par de valores Rule ID y DTag, el emisor debe mantener:

- Un contador de intentos.
- Un temporizador de retransmisión.

Por cada par de valores de RuleID y DTag, el receptor debe mantener un temporizador de inactividad.

Comportamiento de emisor

Al inicio de la fragmentación, el emisor debe seleccionar un valor de Rule ID y un DTag para el paquete SCHC a fragmentar. Cada SCHC Fragment debe contener exactamente un *tile* en su Payload. Todos los *tiles* de índice 0, al igual que el último *tile*, deben ser al menos de tamaño L2 *word size*. En todos los SCHC Fragments debe estar presente el campo W conteniendo la información completa o parcial del número de la ventana que esta siendo transmitida, si la representación es parcial, ésta debe llenar el campo W con los bits menos significativos del número de ventana.

Para los SCHC Fragments que llevan *tiles* diferentes al último *tile* del paquete:

- El fragmento debe tener el formato regular de un SCHC Fragment.
- El campo FCN debe contener el índice del *tile*.
- Cada *tile* debe ser de tamaño tal que complemente el tamaño del encabezado para que el fragmento tenga un tamaño múltipo de L2 *word size*.

El fragmento que contiene el último *tile* debe ser un fragmento All-1.

El emisor debe empezar la transmisión con la ventana número 0 y transmitir las ventanas en orden creciente, mientras que los *tiles* de cada ventana deben ser transmitidos disminuyendo su índice.

El diagrama de la figura 2.14 describe superficialmente el comportamiento del emisor. Para un mayor detalle en cuanto al comportamiento de los índices de cada ventana y *tile*, contadores y temporizadores, se debe revisar el IETF Draft.

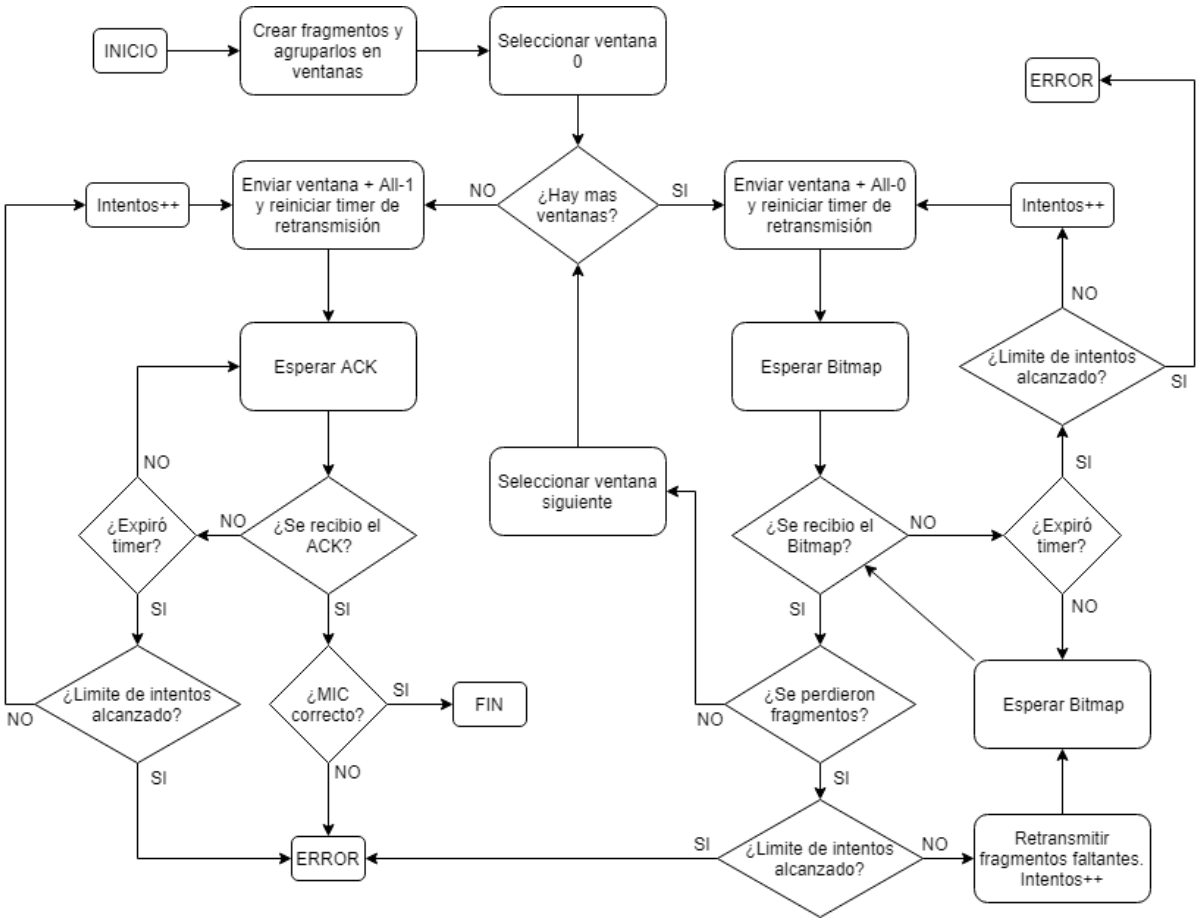


Figura 2.14: Diagrama que muestra el comportamiento del emisor en el modo ACK-Always. Adaptado de [22]

Comportamiento de receptor

Al recibir un fragmento con un par de valores Rule ID y DTag no procesado anteriormente el receptor debe:

- Chequear que el valor de DTag no ha sido usado recientemente para el mismo valor de Rule ID, es decir, asegurarse de que el fragmento recibido no es un remanente de un paquete SCHC recibido con anterioridad. Si lo es, el receptor debe descartar dicho fragmento.
- Iniciar un nuevo proceso de reensamblaje para aquel par de valores de Rule ID y DTag.
- El receptor debe iniciar el temporizador de inactividad. Inicializar un contador de intentos en 0 e inicializar un contador de ventanas también en 0.

Al recibir un nuevo mensaje desde el emisor, el receptor debe reiniciar el temporizador de inactividad.

El receptor se puede encontrar en 3 fases diferentes: fase de aceptación, fase de retransmisión y fase de limpieza.

En la fase de aceptación el receptor se encontrará recibiendo los fragmentos que correspondan con el número de ventana que espera e irá reconstruyendo el paquete a medida que llegan los fragmentos de dicha ventana hasta recibir un mensaje All-0 o All-1. Si el fragmento es un All-0 se tratará del último paquete de la ventana pero no así del paquete por lo que deberá transmitir un SCHC ACK con el Bitmap de la ventana recibida. Si el Bitmap indica que todos los fragmentos de dicha ventana fueron recibidos correctamente, el receptor incrementará su contador de ventanas y entrará nuevamente a la fase de aceptación. Si el Bitmap indica que faltan fragmentos por recibir, el receptor pasará a la fase de retransmisión. Si el fragmento es un All-1, el fragmento será ensamblado junto con los fragmentos anteriores y se realizará el chequeo de integridad sobre el paquete completo, el cual deberá incluir los bits de relleno. Si el chequeo indica que el paquete fue reensamblado correctamente, se pasará a la fase de limpieza, en cambio, si el chequeo indica lo contrario, se pasará a la fase de retransmisión para esta ventana (la última).

Si el receptor recibe un mensaje SCHC ACK REQ con el campo W igual a su contador de ventanas, el receptor deberá enviar un SCHC ACK para esta ventana mientras sigue aceptando nuevos fragmentos.

En la fase de retransmisión, si la ventana no es la última: el receptor sólo aceptará los fragmentos cuyo campo W sea igual a su contador de ventanas hasta que el Bitmap de la ventana indique que toda la ventana fue recibida con éxito, luego de eso transmitirá un SCHC ACK para dicha ventana. Si la ventana es la última: el receptor aceptará fragmentos hasta recibir el fragmento All-1, luego realizará el chequeo de integridad y pasará a la fase de limpieza si el chequeo fue correcto o enviará un SCHC ACK en caso contrario, para luego seguir recibiendo fragmentos en la fase de retransmisión.

En la fase de limpieza, sólo se aceptarán mensajes All-1 y SCHC ACK REQ que además deberán cumplir que su campo W tenga el mismo valor que su contador de ventanas. Al recibir uno de estos mensajes el receptor deberá enviar un SCHC ACK.

En cualquier momento, si el temporizador de inactividad expira, se recibe un mensaje SCHC Sender-Abort o el número de intentos llega al máximo dado por MAX_ACK_REQUESTS, el receptor deberá enviar un mensaje SCHC Receviver-Abort y de deberá terminar el proceso de recepción de mensajes para dicho paquete SCHC.

El diagrama de la figura 2.15 describe superficialmente el comportamiento del emisor. Para un mayor detalle en cuanto al comportamiento de los índices de cada ventana y *tile*, contadores y temporizadores, se debe revisar el IETF Draft.

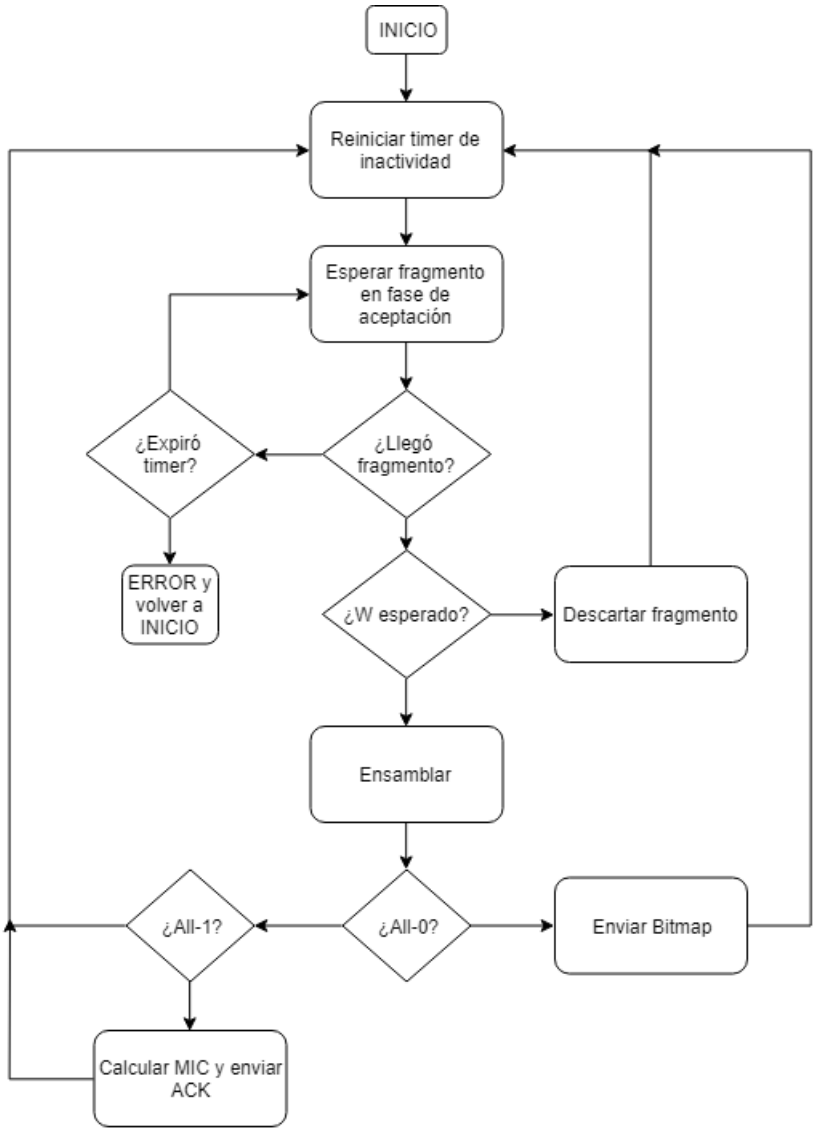


Figura 2.15: Diagrama que muestra el comportamiento del receptor en el modo ACK-Always. Adaptado de [22]

2.4.3. Modo ACK-on-Error

En este modo se utilizan ventanas. Todos los *tiles* deben ser del mismo tamaño a excepción del último, que debe ser de igual o menor tamaño, además todos deben ser de un tamaño mínimo L2 *word size*. Si es especificado en el perfil, el penúltimo *tile* puede tener un tamaño menor a los fragmentos regulares, siendo menor en un L2 *word size*.

En este modo el campo W debe ser lo suficientemente grande como para representar sin ambigüedades el número de la ventana transmitida. El receptor envía mensajes ACK al

emisor para informar los *tiles* faltantes de alguna ventana. No se envían mensajes ACK para ventanas que han sido recibidas completamente.

El emisor de fragmentos retransmite los SCHC Fragments para los *tiles* que fueron reportadas como perdidos. Puede avanzar a la siguiente ventana antes de tener la certeza de que todos los *tiles* de la ventana previamente transmitida fueron recibidos y puede, además, retransmitir SCHC Fragments con *tiles* pertenecientes a ventanas anteriores. De esta forma, el emisor y el receptor pueden trabajar de forma desacoplada.

El proceso de transmisión de fragmentos termina cuando:

- El chequeo de integridad informa que el paquete SCHC fue reensamblado correctamente por el receptor.
- Se han realizado demasiados intentos de retransmisión.
- El receptor determina que la transmisión del paquete fragmentado ha estado inactiva demasiado tiempo.

Cada perfil debe especificar qué valores de Rule ID corresponden con este modo de operación además de los siguientes valores:

- El tamaño de los *tile*.
- El tamaño del campo W (M).
- El tamaño del campo FCN N.
- El tamaño de WINDOW_SIZE, que debe ser estrictamente menor a 2^N.
- El tamaño y el algoritmo para el campo MIC.
- El tamaño del campo DTag.
- El valor de MAX_ACK_REQUESTS.
- El tiempo de expiracion para el temporizador de retransmisión.
- El tiempo de retransmisión para el temporizador de inactividad.
- Si el último *tile* es llevado por un SCHC Fragment regular o un mensaje All-1.
- Si el tamaño del penúltimo *tile* es menor en un L2 *word size* con respecto a los demás *tiles*.

Por cada par de valores Rule ID y DTag, el emisor debe mantener:

- Un contador de intentos.
- Un temporizador de retransmisión.

Para ese mismo par de valores el receptor debe mantener un temporizador de inactividad.

Comportamiento de emisor

Al inicio del proceso de fragmentación, el emisor debe elegir un par de valores Rule ID y DTag para el paquete e inicializar el contador de intentos en 0. Un mensaje SCHC Fragment puede llevar en su Payload uno o más *tiles*. Si lleva mas de un *tile*, se debe cumplir que:

- Los *tiles* seleccionados deben ser consecutivos.
- Los *tiles* deben aparecer en el SCHC Fragment adyacentes el uno del otro de igual forma como aparecen en el paquete SCHC original.

Dependiendo del perfil, el último *tile* de un paquete SCHC puede ser transmitido de 2 formas:

- En un mensaje SCHC Fragment regular, solo o en un Payload multi-*tile*.
- Solo en un fragmento All-1.

El emisor debe enviar al menos un fragmento All-1 después de haber transmitido todos los demás fragmentos del paquete. En el caso que el paquete conste de un solo fragmento, éste debe ser un All-1.

El emisor debe entrar en el proceso de espera de un mensaje SCHC ACK despues de enviar:

- Un fragmento All-1.
- Un mensaje SCHC ACQ REQ cuyo campo W corresponda con el de la última ventana.

Un perfil puede especificar otras situaciones en que el emisor debe esperar un mensaje SCHC ACK, como por ejemplo, después de enviar una ventana completa.

Cada vez que el emisor envía un mensaje All-1 o un SCHC ACK REQ, debe:

- Incrementar su contador de intentos.
- Reiniciar el temporizador de retransmisión.

Al expirar el temporizador de retransmisión:

- Si el contador de intentos es estrictamente menor a MAX_ACK_REQUESTS, el emisor debe enviar un SCHC ACK REQ y aumentar en uno el contador.
- Si no se cumple lo anterior, el emisor debe enviar un SCHC Sender-Abort y terminar el proceso de transmisión de fragmentos con un error.

Al recibir un mensaje SCHC ACK:

- Si el valor del campo W corresponde con el número de la última ventana del paquete SCHC:
  - Si el bit C está en 1, el emisor termina el proceso de transmisión de forma exitosa.
  - Si el bit C está en 0:
    - Si el SCHC ACK indica que no se ha perdido ninguna *tile*, el emisor:
      - Debe enviar un mensaje SCHC Sender-Abort.
      - Puede terminar el proceso de transmisión con un error.
    - Si se han perdido *tiles*:
      - El emisor debe retransmitir todos los fragmentos con los *tiles* faltantes.

· Si el último mensaje de esta secuencia no es un fragmento All-1, el emisor debe enviar un SCHC ACK REQ cuyo campo W indique el valor correspondiente a la última ventana transmitida.

- Si el valor del campo W no corresponde con el número de la última ventana del paquete SCHC:
  - El emisor debe enviar los fragmento que contienen los *tiles* faltantes.
  - Luego puede enviar un SCHC ACK REQ cuyo valor de W corresponda con el de la última ventana.

Para un mayor detalle en cuanto al comportamiento de los índices de cada ventana y *tile*, contadores y temporizadores, se debe revisar el IETF Draft.

Comportamiento de receptor

Al recibir un nuevo SCHC Fragment, el receptor, al igual que en los otros modos de operación, debe chequear que el par de valores Rule ID y DTag no correspondan a los valores de un fragmento remanente de un paquete anterior para luego empezar el proceso de reensamblaje si es que corresponde e inicializar el temporizador de Inactividad y el contador de intentos.

Al recibir un nuevmo mensaje SCHC, el receptor debe reiniciar el temporizador de inactividad.

Al recibir un SCHC Fragment, se debe determinar que *tiles* fueron recibidas basado en el largo del Payload y los valores de los campos W y FCN:

- Si el campo FCN indica que es un mensaje All-1, si el Payload está presente éste debe ensamblarse completamente al paquete incluyendo sus bits de relleno debido a que el receptor no conoce el tamaño del último *tile* del paquete y los bits de relleno son indistinguibles. Si el tamaño del Payload super el tamaño regular de un *tile* mas el tamaño L2 *word size*, se debe levantar una alarma de error.
- Si no es un mensaje All-1, los *tiles* deben ensamblarse basado en su tamaño, que es conocido por el receptor con anterioridad.
  - El Payload puede contener el último *tile* cuyo tamaño será menor, por lo que se debe ensamblar incluyendo los bits de relleno pues en este punto éstos son indistinguibles.
  - El Payload puede contener el penúltimo *tile* que, si fue especificado en el perfil, puede ser exactamente un L2 *word size* mas pequeño que los *tiles* regulares.
  - De otra forma los bits de relleno pueden ser descartados ya que el tamaño de los *tiles* es conocido con anterioridad, los *tiles* son mas grandes que el tamaño L2 *word size* y el tamaño de los bits de relleno es siempre menor al tamaño L2 *word size*.

Al recibir un SCHC ACK REQ o un fragmento All-1:

- Si el receptor tiene al menos una ventana incompleta, debe retornar un mensaje SCHC ACK para la ventana con el número menor.

- En caso contrario:
  - Si ha recibido al menos un *tile*, debe retornar un SCHC ACK con el número de la ventana mayor de la cual posee *tiles*.
  - De otra forma debe retornar un SCHC ACK de la ventana 0.

Un perfil puede especificar otras situaciones en las cuales el receptor debe enviar o no un SCHC ACK y qué ventana debe reportar en aquella circunstancia. Si se recibe un fragmento All-1, el receptor debe realizar el chequeo de integridad del paquete reensamblado y enviar el mensaje SCHC ACK informando el estado del reensamblaje. Si el receptor recibe un SCHC Sender-Abort, debe terminar el proceso de recepción con un mensaje de error y no enviar nada. Finalmente, si el temporizador de inactividad expira, el receptor debe enviar un SCHC Receiver-Abort y terminar el proceso con un mensaje de error, al igual que al exceder el número MAX_ACK_REQUESTS con el contador de intentos.

El reensamblado del paquete SCHC concluye cuando:

- Se recibe un mensaje SCHC Sender-Abort.
- El temporizador de inactividad expira.
- El contador de intentos excede el número MAX_ACK_REQUESTS.
- Cuando se ha recibido un fragmento All-1 y el chequeo de integridad fue realizado con éxito.

Para un mayor detalle en cuanto al comportamiento de los índices de cada ventana y *tile*, contadores y temporizadores, se debe revisar el IETF Draft.

2.5. F/R para LoRaWAN

2.5.1. Parámetros de configuración

Para LoRaWAN, la IETF mantiene un Draft donde especifica las configuraciones sobre el modo de F/R y el perfil a ocupar para esta tecnología en los casos de *Uplink* (nodo a gateway) y *Downlink* (gateway a nodo) [28].

Para Uplink las especificaciones son:

- El modo de fragmentación a ocupar será ACK-Always.
- El L2 *word size* es de 1 octeto (8 bits).
- El campo Rule ID debe tener un tamaño de 3 bits, donde sólo el valor 0 está reservado para la fragmentación.
- El valor de WINDOW_SIZE es 7.
- El campo FCN tiene un tamaño de 3 bits.
- El campo DTag es de sólo 1 bit.
- El algoritmo MIC a utilizar es CRC32 usando el polinomio 0xEDB88320.

- Los emisores no implementan el temporizador de retransmisión. Al terminar de transmitir una ventana, el ACK correspondiente a ella debe ser transmitido en uno de los dos espacios de tiempo destinados para la recepción de mensajes luego de transmitir.
- Los gateway deben implementar el temporizador de inactividad con un valor recomendado de 12 horas. Este valor depende del requerimiento de la aplicación y puede ser cambiado.

Para Downlink las especificaciones son:

- El modo de fragmentación a ocupar será ACK-Always [1].
- El L2 *word size* es de 1 octeto (8 bits).
- El campo Rule ID debe tener un tamaño de 3 bits, donde sólo el valor 0 está reservado para la fragmentación.
- El valor de WINDOW_SIZE es 1.
- El campo FCN tiene un tamaño de 1 bit.
- El campo DTag es de sólo 1 bit.
- El algoritmo MIC a utilizar es CRC32 usando el polinomio 0xEDB88320.

2.5.2. Arquitectura de fragmentación

Para habilitar la fragmentación se deben implementar tanto emisor (*Sender*) como receptor (*Receiver*). Para el caso *Uplink* el emisor debe ser implementado en un nodo LoRaWAN mientras que el receptor puede ser implementado tanto en los *Gateway* de la red como en la *Application* ligada al *Network Server*. Para el caso de *Downlink* el emisor puede ser implementado tanto en los *Gateway* como en la *Application* mientras que el receptor debe se implementado en el nodo LoraWAN.

[1] En la versión 3 del documento Draft para LoRaWAN se cambia al modo ACK-On-Error

Capítulo 3

Implementación del sistema

Este capítulo cubre en detalle la implementación de las etapas de fragmentación y reensamblaje tanto de emisor como de receptor para un enlace LoRaWAN en el caso *uplink* siguiendo la arquitectura mostrada en el figura 3.1 y los parámetros recomendados para LoRaWAN mencionados en la sección 2.5.1.

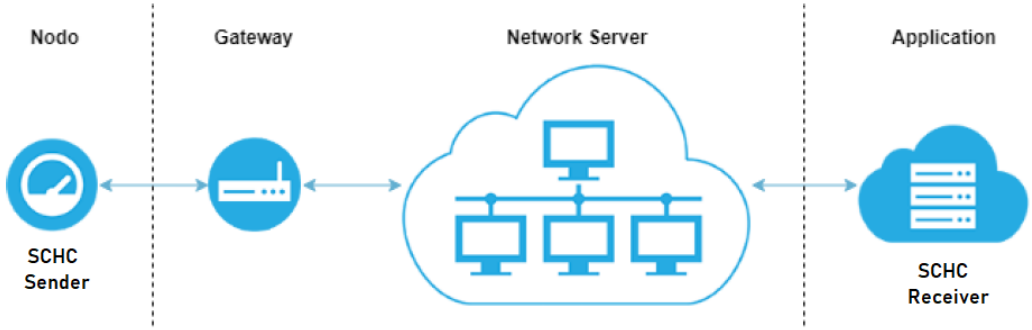


Figura 3.1: Arquitectura de fragmentación para Uplink

En las siguientes secciones se mostrarán las herramientas de hardware y software utilizado, se explicarán las clases utilizadas para la creación de la librería encargada del sistema F/R y su implementación, para luego explicar la implementación tanto de *Sender* como *Receiver* en Nodo y *Application*, finalizando con una sección dedicada a explicar la plataforma experimental montada para probar el sistema.

3.1. Herramientas de hardware y software

Para montar la arquitectura mostrada en la figura 3.1 fue necesario habilitar por separado cada una de las partes que la componen. Para nodo y *gateway* fue necesario utilizar componentes tanto de hardware como de software mientras que para *network server* y *application* sólo fue necesario usar componentes de software.

3.1.1. Pycom LoPy4

El hardware utilizado tanto para nodo como para *gateway* fue el dispositivo LoPy4 desarrollado por Pycom [6]. Esta tarjeta de desarrollo está diseñada principalmente para el desarrollo de proyectos relacionados con el Internet de las Cosas ya que cuenta con módulos LoRa, Sigfox, WiFi y Bluetooth y un microcontrolador potente como lo es el chip ESP32 de Espressif [29]. Además viene habilitado con un firmware capaz de ser programado en Micropython, una versión *lite* de Python 3.

Este dispositivo es capaz de operar en un solo canal LoRa a la vez (al ser diseñado principalmente como nodo) por lo que al ser configurado como *gateway* recibirá mensajes en un solo canal.

Especificaciones LoRa

- Semtech LoRa transceiver SX1276 [18].
- LoRaWAN stack.
- Dispositivos clase A y C.
- Apto para la region AU915.
- Rango de nodo: hasta 40Km.
- Rango de gateway: hasta 22Km.

En la figura 3.2 se tiene una imagen de la tarjeta de desarrollo utilizada en el montaje experimental.

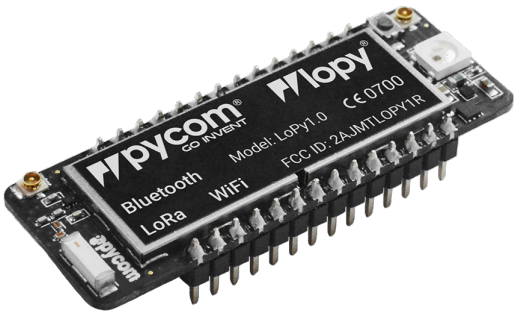


Figura 3.2: Pycom LoPy4

3.1.2. The Things Network

Como *network server* se utilizó *The Things Network* (TTN), el cual es un *network server* de código abierto donde se puede administrar fácilmente una red LoRaWAN que conste de nodos, *gateways* y *applications*. Se eligió debido a que consta de una documentación extensa y una gran comunidad que lo hacen un servidor bastante amigable para el desarrollo de proyectos.

Para este proyecto se registró en TTN una *application* dedicada a la recepción de paquetes y fragmentos SCHC.

3.1.3. Python 3

Todo el software de este proyecto fue desarrollado en Python 3. Para nodo y *gateway* se uso una variante *lite* llamada MicroPython [8], que es una versión compacta de Python 3 que porta las funciones fundamentales, mientras que para la *application* se uso la versión completa pues esta corre directamente en la máquina del usuario y no en un microcontrolador.

3.2. Configuración inicial

Para establecer la comunicación entre nodo, *Gateway* y *Application* dentro del *Network Server* fue necesario registrar cada uno de los elementos en la plataforma *The Things Network* para lo cual se creó una cuenta en dicha plataforma.

3.2.1. Registro de Gateway

Para el registro del *Gateway* fue necesario ingresar un Gateway EUI en forma de un identificador único de 64 bits. TTN recomienda usar una versión expandida de 64 bits de la dirección WiFi MAC de largo 48 bits que se puede obtener llamando un método a través de la terminal de MicroPython del LoPy4. Además fue necesario elegir el plan de frecuencias dado por la región, en este caso AU915, elegir la ruta de direccionamiento de mensajes de uno de los servidores de TTN y la locación del dispositivo, que en este caso es el Departamento de Ingeniería Eléctrica de la Universidad de Chile.

En la figura 3.3 se tienen los parámetros ingresados mientras que en la figura 3.4 se tiene una imagen de la locación.

Figura 3.3: Parámetros de registro de *Gateway* en TTN

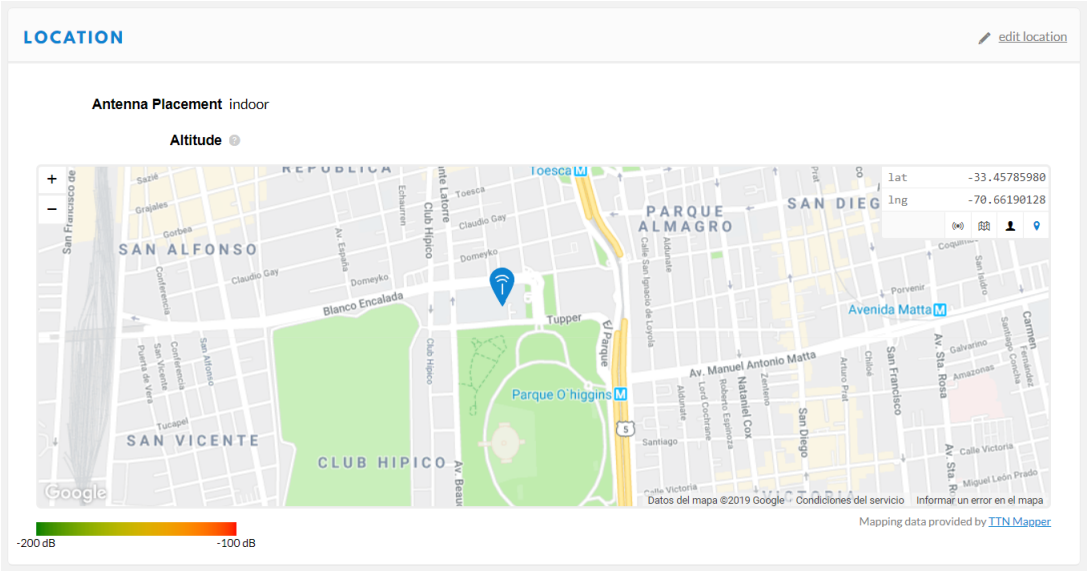


Figura 3.4: Locación de *Gateway*

3.2.2. Registro de Nodo y Application

En primer lugar, se creó una nueva *application* dedicada en su totalidad a la recepción de paquetes SCHC fragmentados. Para esto sólo fue necesario elegir un nombre y asignarle la dirección de alguno de los servidores de TTN puesto que el identificador Application EUI es generado automáticamente, a diferencia del identificador del *Gateway*.

En la figura 3.5 se tienen los parámetros de la *application* registrada.

Luego se registró un nodo dentro de la *application* asignándole un Device ID correspondiente al nombre del dispositivo dentro de la *application* y un Device EUI único correspon-

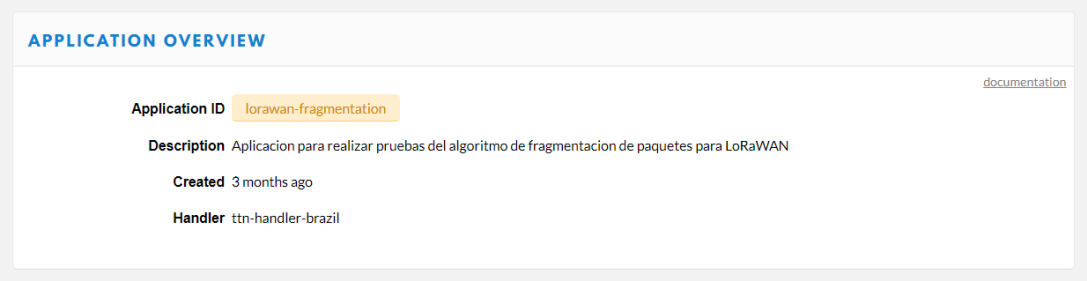


Figura 3.5: Parámetros de registro de *Application* en TTN

diente a la dirección LoRa MAC del dispositivo obtenida mediante un método ingresado en la terminal de MicroPython del LoPy4.

El dispositivo registrado puede conectarse a la red mediante los métodos ABP u OTAA, esto puede ser elegido dentro de la configuración del dispositivo. En este caso se eligió ABP para simplificar el proceso de conexión y poder realizar las pruebas más rápidamente.

En la figura 3.6 se muestran los parámetros del dispositivo.

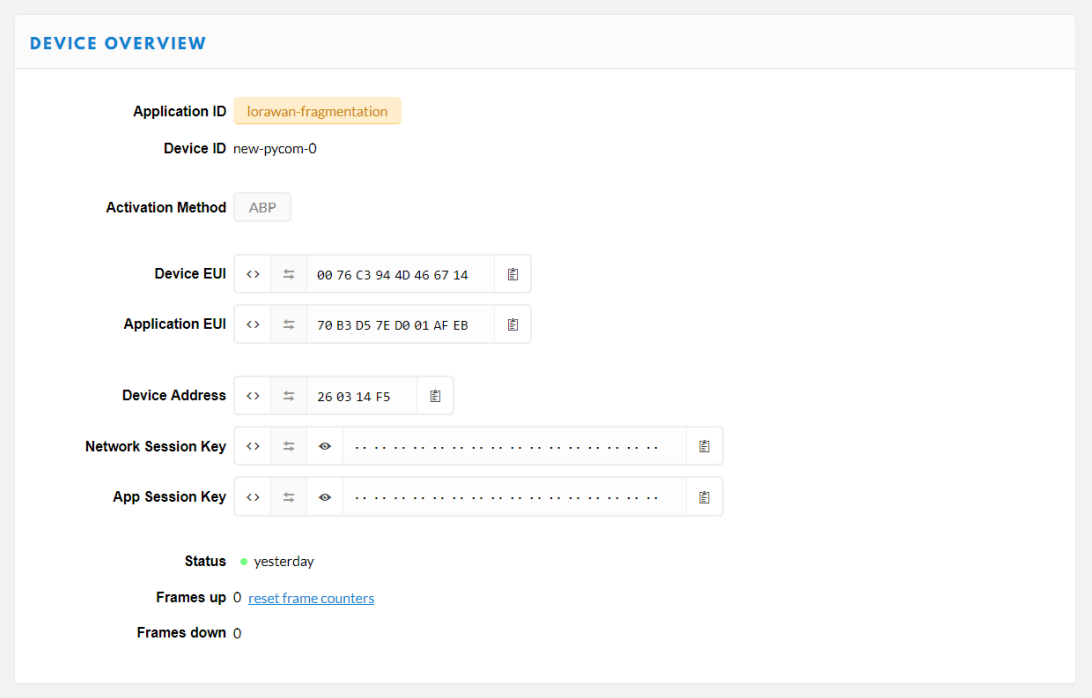


Figura 3.6: Parámetros de configuración de Nodo en TTN

3.3. Librería LoRaWAN Fragmentation

Para la implementación de la etapa F/R se creó la librería *LoRaWAN Fragmentation* (Ver apéndice A) que implementa todos los elementos descritos en la sección 2.2 en diferentes clases que son instanciadas por una clase principal llamada *FREngine*.

Para la implementación de esta librería se usó sólo una librería externa llamada ubistring [13] para poder manejar la información de cada byte en bits dado que la unidad mínima de Python está dada por el objeto `bytes`. A continuación se hará una breve descripción de cada clase y su función dentro del algoritmo.

3.3.1. FRProfile

Clase encargada de guardar todos los parámetros que identifican a un perfil de fragmentación:

- **Rule ID size:** tamaño en bits del campo Rule ID.
- **L2 word size:** tamaño en bits del L2 word.
- **Fragmentation flag:** flag que indica el uso de fragmentación.
- **Modo:** modo de fragmentación.
- **DTag size:** tamaño en bits del campo DTag.
- **W size:** tamaño en bits del campo W.
- **Use Windows flag:** flag que indica el uso de ventanas de fragmentos.
- **WINDOW SIZE:** tamaño de ventana de fragmentos.
- **FCN size:** tamaño en bits del campo FCN.
- **MIC size:** tamaño en bits del campo MIC.
- **MIC algorithm:** tipo de algoritmo MIC a utilizar.
- **Retransmission timer:** segundos de espera para el timer de retransmisión.
- **Inactivity timer:** segundos de espera para el timer de inactividad.
- **MAX ACK REQUEST:** cantidad máxima de mensajes ACK request a enviar.
- **Penultimate tile smaller flag:** flag que indica el tamaño del penultimo fragmento para el modo ACK on Error

Esta clase además cuenta con métodos para seleccionar el modo de fragmentación a utilizar y habilitar/deshabilitar el uso de la fragmentación de paquetes.

3.3.2. FRTile

Clase que instancia un *tile*. Esta clase guarda principalmente 2 arreglos, uno de `bytes` y uno de enteros. El primer arreglo guarda los valores de bytes que se quieren almacenar en el *tile* mientras que el arreglo de enteros guarda el numero de bits menos significativos que se deben tomar de cada byte almacenado que serán utilizados para construir un objeto del tipo `Bits` que una todos los bits en un arreglo. Para entender mejor el funcionamiento de esta clase se dará un pequeño ejemplo: si se quieren almacenar los primeros 3 bits de un arreglo de 4 bytes se deberá obtener en primer lugar el valor entero de los primeros 3 bits del primer `byte` del arreglo realizando los correspondientes desplazamientos de bits, hecho esto se guardará dicho valor entero en la primera casilla del arreglo de bytes del objeto `Tile` y

se guardará un 3 en el arreglo de enteros del objeto `Tile` ya que solamente los 3 bits menos significativos del valor en el arreglo de bytes contienen la información que se quería almacenar en primer lugar.

La clase FRTile se construyó de esta forma para poder ahorrar tiempo de cómputo en la creación de los *tiles* ya que transformar un arreglo grande de `bytes` a `Bits` en primera instancia y luego dividirlo resultó ser muy costoso en cuanto a operaciones por lo que se decidió hacerlo por partes.

3.3.3. FRPacket

Clase que instancia un paquete SCHC. En ella se guardan las siguientes variables:

- **Packet:** arreglo de bytes con el paquete a enviar.
- **Tiles:** arreglo de objetos FRTile que guarda los *tiles* del paquete luego de aplicar el método `get_tiles()` de la clase FRPacket.
- **Packet padding:** variable usada para computar el MIC sobre el paquete. Dependiendo del padding asignado a la última *tile* del paquete ésta variable le agrega un byte más al paquete para poder computar el MIC sobre una cantidad entera de bytes. Si es que la última *tile* no necesita padding esta variable guarda un byte nulo.
- **Last tile padding:** variable que guarda la cantidad de bits de padding del último tile.
- **MIC:** variable que guarda el valor del MIC computado sobre el paquete.

Esta clase cuenta con métodos para obtener los *tiles* dependiendo del modo de fragmentación y el tamaño requerido de cada *tile*, un método para calcular el MIC , un método para reconstruir el arreglo de bytes dependiendo de los *tiles* en la variable `Tile`, un setter y getter del arreglo de bytes `Packet`.

3.3.4. FRFragmentEngine

Clase utilizada para poder crear cada tipo de mensaje dependiendo de los parámetros de un FRProfile, calcular el padding necesario para el último *tile* e identificar el tipo de mensaje recibido tanto para *Sender* como para *Receiver* entregando además los valores de cada campo para dicho mensaje.

Se inicia entregándole un objeto del tipo FRProfile para poder obtener los parámetros correspondietes a los tamaños de cada campo del header. Con este objeto es posible crear los mensajes:

- Regular SCHC fragment
- All-1 fragment
- ACK
- ACK Request

- Sender-Abort
- Receiver-Abort

Además se crearon los métodos:

- `sender_message_recovery(fragment)`
- `receiver_message_recovery(fragment)`

El primero identifica los mensajes ACK y Receiver-Abort mientras que el segundo identifica los mensajes Regular SCHC Fragment, All-1 fragment, Sender-Abort y ACK Request.

3.3.5. FRBitmap

Clase utilizada para instanciar un Bitmap, trabajar sobre él para la creación del mensaje SCHC ACK, obtenerlo desde un mensaje SCHC ACK e informar el estado de recepción de los fragmentos por ventana para el *Receiver*.

3.3.6. CRC32

Clase utilizada para calcular el MIC por defecto dado por el algoritmo CRC32.

3.3.7. FRCommon

Archivo que guarda constantes y métodos comunes entre clases. En ella se define el método `take_field_from_fragment()` utilizado para poder separar un arreglo de `bytes` en diferentes objetos del tipo FRTile y así obtener los valores de cada campo en caso de que se use para tratar los mensajes recibidos por *Sender* o *Receiver* o para obtener los *tiles* de un paquete.

En él tambien se definen las clases FRModes con las constantes que definen a cada modo de fragmentación, la clase FRMessages con las constantes que definen a cada tipo de mensaje y la clase FRHeaders que identifica cada *header* con su índice dentro del arreglo `headers` entregado por los métodos de identificación de mensajes de la clase FRFragmentEngine.

Finalmente este archivo almacena un diccionario que mapea los valores de *Data Rate* con su tamaño máximo de *payload* para la región AU915 y el contenido de un mensaje llamado DUMMY MSG cuya utilidad se explica en la sección 3.4.2.

3.3.8. FREngine

Clase principal que se encarga de manejar todo el proceso de fragmentación y reensamblaje de un paquete dependiendo de si es utilizada en un *Sender* o en un *Receiver*. En ella se guardan las siguientes variables de configuración:

- **DR:** valor del *data rate* a utilizar.
- **ID Profiles:** diccionario que mapea Rule ID con su respectivo Profile.
- **Packet:** variable que almacena el paquete en un arreglo de `bytes`.
- **Fragments:** variable que almacena los fragmentos generados/recibidos.

También se guardan variables para contar los mensajes enviados, la cantidad de bits enviados, los mensajes recibidos y la cantidad de bits recibidos con los cuáles se evaluará el sistema.

Para el proceso de recepción se guarda un *flag* para identificar que se esté recibiendo un paquete, un *buffer* para la recepción de fragmentos, un flag para encolar un mensaje ACK y los parámetros de valor de ventana, bitmap, bit C del chequeo de integridad para poder discriminar los mensajes recibidos y poder armar los mensajes ACK.

La inicialización de esta clase debe tener como parámetros el valor del *Data Rate* de transmisión en el caso de que sea un *Sender* y opcionalmente un arreglo de `bytes` correspondiente al paquete a enviar. Es necesario entregar el DR a utilizar debido a que este valor determina el tamaño máximo de mensaje a enviar y por consiguiente el tamaño de cada *tile*.

Para el proceso de transmisión se creó el método `send_packet_always_ack()` que se encarga de manejar la transmisión del paquete para el modo Always-ACK que se utilizará para las pruebas.

Para el proceso de recepción se crearon los métodos `start_receiving()`, `stop_receiving()` y `continue_receiving()` para manejar el estado del receptor al momento de recibir un mensaje. Además se creó el método `receive()`, el cual se ejecuta cada vez que se recibe algún mensaje si es que el receptor se encuentra en estado de recepión de un paquete.

3.4. Implementación de enlace con fragmentación y reensamblaje

3.4.1. Enlace LoRaWAN

Para establecer el enlace LoRaWAN y poder realizar las primeras pruebas de transmisión se usaron los ejemplos `main_AU915.py` y el ejemplo de *Nano Gateway* disponibles en el repositorio de Pycom [10]. Dado que el gateway utilizado corresponde a un Pycom LoPy4 de un solo canal fue necesario configurar tanto nodo como *gateway* con la misma frecuencia procurando que ésta estuviera dentro del rango de las frecuencias dadas para la región AU915.

La frecuencia elegida fue de 915,2MHz.

También fue necesario configurar ambos dispositivos con el mismo valor de *data rate*. Se hicieron pruebas de enlace para los DR del 0 al 6 siendo el 6 el único incapaz de establecer un enlace correcto.

En el lado del *Gateway* fue necesario además agregar la ruta del servidor de TTN que se encargará de manejar los mensajes que entran y salen del *gateway*, eligiéndose el servidor de Europa por su estabilidad.

3.4.2. Limitaciones de TTN

Para la implementación del algoritmo F/R se tuvo que modificar levemente el comportamiento del *Sender* debido a una limitante dada por TTN y la integración de Python. A continuación se contextualizará dicha limitante.

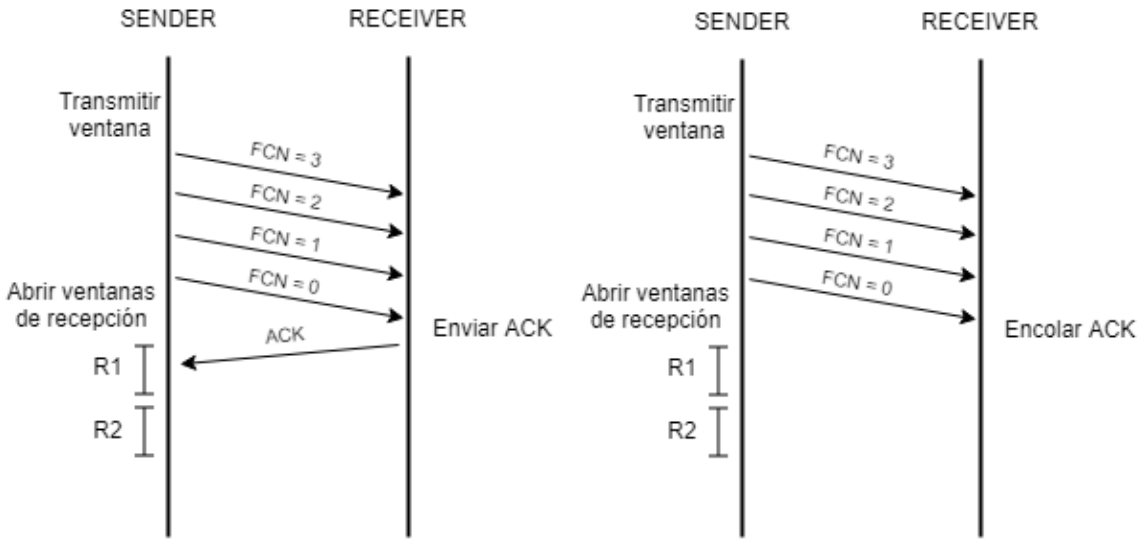
Para los modos con ventanas, el algoritmo indica que luego de la transmisión completa de una ventana por parte del *Sender*, el *Sender* se pondrá inmediatamente en espera de un mensaje ACK por parte del *gateway*, por lo que luego de que el *Receiver* haya recibido el último mensaje de una ventana éste debe responder inmediatamente con un mensaje ACK. Para que pueda ser recibido por el *Sender*, este mensaje ACK deberá llegar al *Sender* en alguno de los intervalos de tiempo dados por las ventanas de recepción que se abrirán.

Como el *Receiver* fue montado en una *application* del *network server* es necesario esperar a que el último fragmento de la ventana sea procesado por la *application* para luego ordenar el envío de un mensaje ACK. La limitación se encuentra en el modo en que se envía esa orden a TTN (dada por la integración de Python), que hace que esta orden no pueda llegar a tiempo al *gateway*, por lo que el mensaje queda en cola hasta que se abran nuevamente las ventanas de recepción. Dicha apertura sólo ocurrirá cuando el *Sender* envíe un nuevo mensaje. Este problema se ilustra en la figura 3.7. En 3.7a, el comportamiento esperado indica que el ACK es enviado luego de la recepción del último fragmento de la ventana, mientras que en 3.7b se ilustra el comportamiento real que indica que el ACK no es transmitido, sino puesto en cola.

Para arreglar este problema se planteó como solución transmitir un DUMMY MSG desde el *Sender* con la única finalidad de provocar el envío del ACK encolado. En la figura 3.8 se ilustra la solución planteada.

3.4.3. Sender

Para la implementación del *Sender* se siguió la referencia del documento Draft de LoRa [28] por lo que sólo se implementó el modo Always-ACK usando el perfil por defecto dado por este mismo documento (ver sección 2.5.1). Esta implementación se realizó usando la librería LoRaWAN Fragmentation descrita en la sección 3.3. El código debe correr en un Pycom LoPy4.



(a) Comportamiento esperado para envío de ACK. (b) Comportamiento real para envío de ACK.

Figura 3.7: Ilustración de problema de *Downlink* dado por TTN

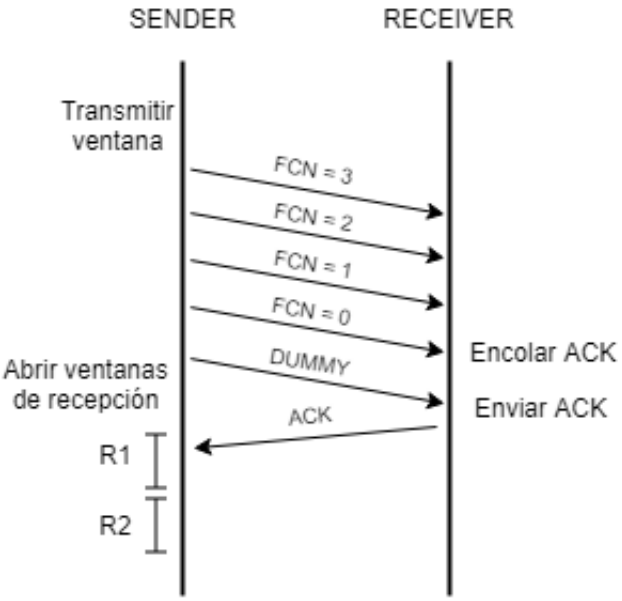


Figura 3.8: Solución planteada para manejar el envío de ACK

Como la implementación del algoritmo F/R no depende del contenido del paquete a enviar, se decidió simular un paquete IPv6 como una lista de `bytes` aleatoria de un tamaño de 1500 bytes, que es el tamaño recomendado de paquete, superior al MTU de 1280 bytes [17].

A continuación se describen los pasos para realizar la transmisión de un paquete:

1: Elegir Data Rate
2: Establecer enlace LoRaWAN entre Nodo y Gateway
3: Crear Perfil por defecto instanciando FRProfile
4: Habilitar fragmentación para el Perfil
5: Instanciar FREngine e iniciarlo con el DR elegido
6: Agregar Perfil por defecto con RuleID=0 al objeto FREngine
7: Fijar RuleID=0 a la transmisión del paquete
8: Crear paquete aleatorio de bytes
9: Fijar paquete a enviar en FREngine
10: Ejecutar `send_packet_always_ack()`

Terminado el proceso de transmisión, el método `send_packet_always_ack()` imprimirá en la terminal del LoPy4 la cantidad de mensajes enviados, la cantidad de bits enviados, la cantidad de mensajes recibidos y la cantidad de bits recibidos.

3.5. Receiver

Para el *Receiver* se ocupó la librería TTN para Python 3, con la cual se puede recibir el payload de los mensajes enviados desde *Sender* a *Receiver* y encolar los mensajes ACK. Dicha librería se basa en el protocolo MQTT para el tratado de los mensajes y ocupa funciones de *callback* que pueden ser asociados a los eventos de recepción y transmisión de mensajes desde y hacia TTN [12]. El código debe correr en alguna máquina con Python 3 instalado y con conexión a Internet.

A continuación se describen los pasos para iniciar un *Receiver* e iniciar la recepción de un paquete fragmentado.

1: Asignar Application ID, Device ID y Access Key de la Application
2: Iniciar un cliente MQTT con los parámetros de la Application
3: Crear perfil por defecto instanciando FRProfile
4: Habilitar fragmentación para el Perfil
5: Instanciar FREngine
6: Agregar Perfil por defecto con RuleID=0 al objeto FREngine
7: Asignar método Callback a la recepción de mensajes

El método Callback empleado tiene el comportamiento.

```
1: if messagge == DUMMY_MSG then
2:     pass
3: else
4:     if no se esta recibiendo un paquete then
5:         Obtener RuleID y DTag desde message
6:         if DTag != DTag_anterior then
7:             Iniciar recepción de nuevo paquete
8:         else
9:             Continuar recibiendo paquete anterior para encolar nuevos ACK
10:        end if
11:    end if
12:    if se esta recibiendo un paquete then
13:        Almacenar message en buffer
14:        Ejecutar receive(message)
15:        if se debe enviar ACK then
16:            Encolar ACK con Bitmap, Bit C y número de ventana actuales
17:        end if
18:    end if
19: end if
```

Finalizado un proceso de recepción de un paquete, el método `receive()` imprimirá en la terminal del LoPy4 la cantidad de mensajes enviados, la cantidad de bits enviados, la cantidad de mensajes recibidos y la cantidad de bits recibidos.

3.6. Plataforma experimental

Se montó el *gateway* en una estación base ubicada en el Departamento de Ingeniería Eléctrica de la Universidad de Chile en el laboratorio de telecomunicaciones en la misma ubicación que se mostró en la figura 3.4 junto con un computador con conexión a Internet corriendo el programa del *Receiver*.

El nodo se trasladó al patio del edificio Beauchef Poniente tal como se ve en la figura 3.9 a una distancia aproximada de 235 metros del *gateway*. Se debe destacar que no hay línea de vista entre Nodo y *gateway* y que además entre ellos se encuentran 2 edificaciones de gran envergadura como lo son el edifico Escuela de Beauchef y la Torre Central de la facultad de Ingeniería de la Universidad de Chile.

En la figura 3.10 se tiene una imagen del Nodo en terreno.

Para las pruebas de transmisión de paquetes se enviaron 5 paquetes de 1500 bytes de tamaño para los valores de *Data Rate* 0, 3, 4 y 5 y se obtuvieron la cantidad de mensajes recibidos y transmitidos y la cantidad de bits intercambiados entre *Sender* y *Receiver*.

Figura 3.9: Ubicación de nodo para las pruebas de transmisión usando fragmentación

Figura 3.10: Imagen del nodo en terreno, ubicado en el patio del edificio Beauchef Poniente

Capítulo 4

Resultados y análisis

Para la obtención de los resultados se calculó lo siguiente:

- Tiempo de transmisión de fragmentos.
- Tasa de error de transmisión de fragmentos.
- Tasa de error de transmisión de paquetes.

En todos los casos el contenido de los paquetes enviados correspondió a arreglos de tamaño fijo cuyo contenido fueron bytes aleatorios. Se hizo de esta forma ya que el objetivo de este trabajo fue probar la implementación del algoritmo de fragmentación independiente del contenido de los mensajes, además así es posible montar cualquier tipo de mensaje dentro el algoritmo al tratarse sólo de arreglos de bytes.

Para los cálculos de las tasas de error se transmitieron paquetes de 500 y 1500 bytes. Las razones para elegir estos tamaños fueron: paquetes de 500 bytes para probar la transmisión de paquetes que sólo necesitaran una ventana de transmisión para ser enviados completamente (en el caso de los DR 3, 4 y 5) y así probar el caso más simple de transmisión y paquetes de 1500 bytes ya que este es el MTU recomendado para la transmisión de paquetes IPv6 [17].

4.1. Tiempo de transmisión de fragmentos

Para esta prueba se midió el tiempo de transmisión de 30 fragmentos para cada *Data Rate* (DR) entre el 0 y el 5. El tamaño de cada fragmento fue el máximo aceptado por cada DR por lo que para los DR 0, 1 y 2 fue de 51 bytes, para DR 3 fue de 115 bytes y para los DR 4 y 5 fue de 222 bytes. En la tabla 4.1 se pueden observar los resultados de las pruebas.

Tabla 4.1: Tabla de tiempos de transmisión para cada DR

Data Rate	**Tiempo promedio [seg]**	**Desviación estándar [seg]**
0	3,019	0,341
1	2,049	0,250
2	1,273	0,235
3	1,257	0,368
4	1,127	0,438
5	1,694	0,744

4.2. Tasas de error de fragmentos y paquetes

Para estas pruebas se calcularon la cantidad de fragmentos perdidos y paquetes reensamblados para paquetes de 500 y 1500 bytes. Se enviaron 10 paquetes de 500 bytes y 10 paquetes de 1500 bytes para cada DR del 0 al 5 obteniéndose los resultados de la tabla 4.2

Tabla 4.2: Tasas de error de transmisión de fragmentos para cada DR

Data Rate	**Tasa de error de Tx de fragmentos**	**Desviación estándar**
0	0,054	0,092
1	0,016	0,044
2	0,077	0,083
3	0,078	0,214
4	0,149	0,208
5	0,837	1,120

En cuanto a la tasa de error de transmisión de paquetes, ésta fue nula para todos los DR excepto el DR 5 el cual tuvo un porcentaje de error de transmisión de paquetes de un 70 % para los paquetes de 1500 bytes. En este caso la calidad del enlace bajó considerablemente pues se tuvieron casos de problemas en la transmisión de la última ventana que desencadenó el envío de un mensaje Sender-Abort por parte del *sender* y casos de error en el MIC del paquete que sólo se vieron en las pruebas hechas con este DR.

4.3. Cantidad de fragmentos por paquete

Finalmente, para realizar un análisis mas completo se muestran en la tabla 4.3 la cantidad de fragmentos necesarios para transmitir un paquete de 1500 bytes para cada DR.

Tabla 4.3: Cantidad de fragmentos por paquete de 1500 bytes para cada DR

Data Rate	**Número de fragmentos**
0	38
1	38
2	38
3	15
4	7
5	7

4.4. Análisis de resultados

Comparando los resultados de las tablas anteriores se deduce que los mejores DR para las condiciones del experimento son los DR 3 y 4 ya que en ambos se tienen bajas tasas de errores de transmisión de fragmentos y una cantidad pequeña de fragmentos necesarios para transmitir paquetes de 1500 bytes. La elección entre uno y otro dependerá exclusivamente de la distancia que se tenga entre nodo y *gateway* ya que en las condiciones del experimento ambos DR tienen un buen desempeño, pero aumentando la distancia la calidad del enlace dada por el DR 4 disminuirá siendo necesario realizar pruebas de transmisión antes de implementar este sistema.

En cuanto a la cantidad de fragmentos necesarios para la transmisión de un paquete SCHC, los primeros tres valores de DR tienen un tamaño de Payload de 51 bytes (ver tabla 1.1) necesitándose aproximadamente 38 fragmentos para poder completar un proceso de transmisión. Al necesitarse un mayor número de fragmentos para transmitir un paquete la cantidad de energía necesaria para esto aumentaría provocando que los niveles de carga de las baterías disminuyan rápidamente pues el gasto de energía es directamente proporcional a la cantidad de mensajes enviados. Generalmente, los DR más bajos se utilizan en aplicaciones donde se necesite abarcar mayores rangos de distancia lo que puede quererse por ejemplo en aplicaciones donde se necesiten medir variables ambientales. En dichas aplicaciones es posible que los nodos se encuentren en lugares poco accesibles y lejanos por lo que sería importante maximizar el uso de las baterías para disminuir la frecuencia de mantenimiento de los dispositivos. Si se aplicara el algoritmo de fragmentación en aplicaciones con DR bajos se eliminaría una de las grandes ventajas que ofrece LoRaWAN que es el poco gasto energético.

En cuanto a los tiempos de transmisión, los DR 0 y 1 tienen tiempos de transmisión por fragmento por sobre los 2 segundos haciendo que el tiempo de transmisión de un paquete sea demasiado alto considerando además la cantidad de fragmentos que se envían al usar estos DR. Para los DR restantes los tiempos de transmisión de un fragmento permanecen

constantes por lo que el tiempo de transmisión de un paquete dependerá casi en su totalidad de la cantidad de fragmentos a enviar que dependerán del tamaño completo del paquete y el DR a utilizar.

Finalmente, de los resultados obtenidos se puede deducir que el algoritmo de fragmentación sólo será útil para DRs de valores medios, como 3 o 4, ya que con estos valores de DR las pérdidas llegan a su mínimo al igual que la cantidad de mensajes enviados. Si se aumenta el valor de DR las pérdidas de fragmentos son mayores por lo que vuelve a aumentar la cantidad de mensajes enviados aumentando los gastos de energía.

4.5. Discusión

Esta sección será dividida en diferentes secciones dedicadas a discutir algunos aspectos relevantes sobre la implementación del algoritmo F/R en redes LoRaWAN y concluir sobre los resultados de este trabajo.

4.5.1. ¿Por qué implementar un algoritmo de fragmentación para el soporte de IPv6 y no un NAT (Network Address Translation)?

Un NAT traduce direcciones IP privadas a direcciones IP públicas para poder comunicar redes cerradas a Internet. No es posible implementar esto en la arquitectura LoRaWAN ya que sólo el enlace entre *gateway*, *network server* y la Internet soportan el protocolo IP mientras que el enlace Gateway-Nodo no lo soporta. Si se quisiera implementar un NAT en los *gateway* el enlace Nodo-Gateway deberia soportar IP siendo necesario la implementación de la fragmentación de paquetes.

4.5.2. Implementación de Receiver en Network Server o Gateway

Por simplicidad se decidió implementar el lado del Receiver en una *application* del *network server*, pero se descubrió, al realizar las pruebas de *downlink*, que no es posible responder directamente un mensaje de *uplink* con un *downlink* cuyo valor dependa del mensaje *uplink* recibido debido a los problemas de encolamiento de mensajes *downlink*. Este problema ocurre por el retraso de la orden de encolamiento desde el *network server* al *gateway*. Para arreglar esto fue necesario añadir la transmisión de un mensaje más en el proceso de transmisión de ventanas a cuestas de disminuir el rendimiento del algoritmo.

Este problema podría ser suprimido si es que el *gateway* tomara por su cuenta la decisión de enviar un mensaje ACK, implementando el *Receiver* directamente en él. El problema de implementar el *Receiver* en el *gateway* es que el *gateway* deberá conocer el contenido de los mensajes LoRaWAN para lo que tendría que tener las claves *Network Session Key*

y *Application Session Key* de cada *application* para cada Nodo lo que no es posible de antemano. Para esto, entonces, se debería diseñar un método para poder intercambiar las claves de forma segura entre Nodo y *gateway*.

4.5.3. Conveniencia del uso del algoritmo de fragmentación en aplicaciones con DR bajos

Para DR bajos (0, 1 y 2) el algoritmo de fragmentación no debería ser implementado para aplicaciones donde sea necesario mantener bajo los gastos energéticos ya que para estos DR los pequeños tamaños de los payload provocan que sea necesario enviar muchos mensajes para poder transmitir un paquete SCHC afectando así el rendimiento de las baterías.

4.5.4. Efectividad del uso de fragmentación para DR altos

Para DR altos, si es que las condiciones del canal no son lo suficientemente buenas, se podría tener una gran tasa de pérdida de fragmentos lo que disminuiría considerablemente el rendimiento del algoritmo de fragmentación ya que provocaría muchas retransmisiones aumentando así el consumo energético de los dispositivos.

4.5.5. Diferencias de la implementación de fragmentación en enlace LoRaWAn y en red LoRaWAN

Para poder implementar la fragmentación en una red LoRaWAN no basta con sólo implementarla en un enlace ya que al tenerse más de un nodo es posible que puedan ocurrir transmisiones simultáneas de paquetes SCHC y sea necesario compartir los recursos del *Receiver* de forma eficiente. Una solución rápida sería levantar un proceso en el *Receiver* cada vez que se tenga que recibir un nuevo paquete pero esta solución haría muy ineficiente el sistema cuando se tuvieran varias transmisiones en simultáneo. En conclusión, se necesitaría además agregar un algoritmo de planificación para el uso de recursos.

Capítulo 5

Conclusiones y trabajo futuro

En este trabajo se presentó una prueba experimental de la propuesta del algoritmo SCHC F/R dada por la IETF. Se implementó el algoritmo de fragmentación y reensamblaje usando el modo de operación Always-ACK usando un nodo Pycom como *SCHC sender* y el Network Server *The Things Network* como *SCHC receiver* y se evaluó su desempeño para un contexto de ciudad transmitiendo paquetes de tamaño típico en la implementación de IPv6.

En los resultados obtenidos se puede observar que la implementación del algoritmo SCHC F/R logra habilitar satisfactoriamente la transmisión de grandes paquetes de datos a través de un enlace LoRaWAN, por lo que la integración de IPv6 a estas redes es totalmente factible, sin embargo es necesario estudiar además qué tan factible es su implementación dadas las características de los dispositivos y las aplicaciones en las que estas redes se usan. Si es que el uso de batería no es un problema y las distancias entre nodo y *gateway* son bajas, la implementación de IPv6 usando este algoritmo se puede realizar de forma directa, sin embargo, si no se tienen estas condiciones se deberá realizar una evaluación más detallada de su implementación ya que el uso del algoritmo de fragmentación aumenta considerablemente la cantidad de mensajes a enviar por un nodo y los tiempos de transmisión de los paquetes. Entonces, la evaluación hecha en este trabajo puede ser utilizada en futuras discusiones sobre implementaciones reales de este algoritmo en redes LoRaWAN.

Como trabajo futuro se plantea la implementación del *SCHC receiver* en un *gateway* LoRaWAN para no tener que modificar el comportamiento de los mensajes de *acknowledgment*. Para esto es necesario implementar un mecanismo para el intercambio de llaves entre nodo y *gateway* ya que para la implementacion del algoritmo F/R es necesario que el *receiver* sea capaz de desencriptar los mensajes enviados por el nodo, lo que ahora no es posible ya que las llaves las posee sólo el *Network Server*.

Finalmente, se puede concluir que el diseño del algoritmo de F/R está bien pensado a excepción del encolamiento de los mensajes *downlink* para el envío de los ACK en la implementación del *sender* en el Network Server. Al arreglar este problema no se encontraron más problemáticas en su implementación.

Apéndice A

Librería LoRaWAN Fragmentation

El codigo usado en este proyecto se encuentra en un repositorio *Github*. La última versión se encuentra en:

`https://github.com/juanolete/schc-fragmentation-lorawan`

Bibliografía

[1] 3 reasons why ipv6 is important for the internet of things. https://www.link-labs.com/blog/why-ipv6-is-important-for-internet-of-things. Accessed: 2018-11-12.

[2] Ietf 102 hackathon. https://trac.ietf.org/trac/ietf/meeting/wiki/102hackathon. Accessed: 2018-11-12.

[3] Ietf 103 hackathon. https://trac.ietf.org/trac/ietf/meeting/wiki/103hackathon. Accessed: 2018-11-12.

[4] Introduction to chirp spread spectrum (css) technology. http://www.ieee802.org/802_tutorials/03-November/15-03-0460-00-0040-IEEE-802-CSS-Tutorial-part1.ppt. Accessed: 2019-07-05.

[5] Ipv6 over low power wide-area networks (lpwan) working group. https://datatracker.ietf.org/wg/lpwan/about/. Accessed: 2018-11-12.

[6] Lopy4 development board. https://pycom.io/product/lopy4/. Accessed: 2019-07-09.

[7] Lpwan market report. https://iot-analytics.com/lpwan-market-report-2018-2023-new-report. Accessed: 2018-11-12.

[8] Micropython. https://github.com/micropython/micropython-lib. Accessed: 2019-07-09.

[9] Norma técnica de equipos de alcance reducido, ministerio de transporte y telecomunicaciones; subsecretarÍa de telecomunicaciones. https://www.leychile.cl/Navegar?idNorma=1109333. Accessed: 2018-11-12.

[10] Pycom libraries. https://github.com/pycom/pycom-libraries. Accessed: 2019-07-09.

[11] Schc implementation (for micropython and python). https://github.com/openschc/openschc. Accessed: 2019-08-23.

[12] Ttn python api reference. https://www.thethingsnetwork.org/docs/applications/python/api-reference.html. Accessed: 2019-07-09.

[13] ubitstring library. https://github.com/mjuenema/micropython-bitstring. Acces-

sed: 2019-07-09.

[14] Understanding the ipv6 header. https://www.microsoftpressstore.com/articles/article.aspx?p=2225063&seqNum=4. Accessed: 2018-11-12.

[15] What is lora? https://www.semtech.com/lora/what-is-lora. Accessed: 2018-11-12.

[16] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, 55(9):34–40, 2017.

[17] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. STD 86, RFC Editor, July 2017.

[18] General Description, K E Y Product Features, and Ordering Information. Sx1276/77/78/79. (August), 2016.

[19] S. Farrell. Low-power wide area network (lpwan) overview. RFC 8376, RFC Editor, May 2018.

[20] Lora Alliance. LoRaWAN 1.1 Regional Parameters. 2018.

[21] N. Maturana Araneda. Implementation and evaluation of static context header compression for IPv6 packets within a LoRaWAN network. 2019.

[22] Ana Minaburo, Laurent Toutain, Carles Gomez, Dominique Barthel, and Juan Zuniga. Static context header compression (schc) and fragmentation for lpwan, application to udp/ipv6. Internet-Draft draft-ietf-lpwan-ipv6-static-context-hc-21, IETF Secretariat, July 2019. http://www.ietf.org/internet-drafts/draft-ietf-lpwan-ipv6-static-context-hc-21.txt.

[23] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of ipv6 packets over ieee 802.15.4 networks. RFC 4944, RFC Editor, September 2007. http://www.rfc-editor.org/rfc/rfc4944.txt.

[24] Ramon Sanchez-Iborra, Jesus Sanchez-Gomez, Salvador Perez, Jose Santa, Pedro J. Fernande, Jose-Luis Hernandez-Ramos, and Antonio Skarmeta. Internet access for LoRaWAN devices considering security issues. *Global IoT Summit (GIoTS)*, pages 13 – 18, 2018.

[25] Ramon Sanchez-Iborra, Jesus Sanchez-Gomez, Jose Santa, Pedro J. Fernandez, and Antonio F. Skarmeta. IPv6 communications over LoRa for future IoV services. *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings*, 2018-Janua:92–97, 2018.

[26] Semtech. LoRa^{TM}Modulation Basics Semtech. AN1200.22(May):1–26, 2015.

[27] N Sornin, M Luis, T Eirich, and T Kramp. LoRaWAN ^{TM} Specification. *LoRa^{TM} Alliance*, pages 1–70, 2016.

[28] Nicolas Sornin, Michael Coracin, Ivaylo Petrov, Alper Yegin, Julien Catalano, and Vincent AUDEBERT. Static context header compression (schc) over lorawan. Internet-Draft draft-petrov-lpwan-ipv6-schc-over-lorawan-03, IETF Secretariat, February 2019. `http://www.ietf.org/internet-drafts/draft-petrov-lpwan-ipv6-schc-over-lorawan-03.txt`.

[29] Espressif Systems. ESP32 Series Datasheet. 2019.

[30] Steffen Thielemans, Maite Bezunartea, and Kris Steenhaut. Establishing transparent IPv6 communication on LoRa based low power wide area networks (LPWANS). *Wireless Telecommunications Symposium*, 2017.

[31] Technical Marketing Workgroup. LoRaWAN101. (November), 2015.