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

MODELADO Y EVALUACIÓN DE LA EFICIENCIA DEL ESTÁNDAR SCHC PARA EL TRANSPORTE DE PAQUETES IP SOBRE LORAWAN

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA MENCIÓN ELÉCTRICA

RODRIGO EUGENIO MUÑOZ LARA

PROFESORA GUÍA:
DRA. SANDRA CÉSPEDES UMAÑA

MIEMBROS DE LA COMISIÓN:
DR. JORGE SILVA SÁNCHEZ
DR. DIEGO DUJOVNE

SANTIAGO DE CHILE
2020

RESUMEN DE LA TESIS PARA OPTAR
AL GRADO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA MENCIÓN ELÉCTRICA
POR: RODRIGO EUGENIO MUÑOZ LARA
FECHA: 2020
PROF. GUÍA: DRA. SANDRA CÉSPEDES UMAÑA

MODELADO Y EVALUACIÓN DE LA EFICIENCIA DEL ESTÁNDAR SCHC PARA EL TRANSPORTE DE PAQUETES IP SOBRE LORAWAN

En la actualidad, han surgido varias tecnologías que están diseñadas para comunicaciones de largo alcance y bajo consumo. Estas características son muy valoradas en el *Internet of Things* debido a que los dispositivos finales deben mantener un régimen de consumo tal que sea sustentable respecto de la generación. Con la aparición de la red global de cosas, *the Internet of Things (IoT)*, han surgido nuevas tecnologías, destacando LoRaWAN™, Sigfox y NB-IoT; todas ellas tienen la característica común de no soportar de forma nativa direccionamiento IP debido a los pequeños tamaños de memoria para almacenar una pila de protocolos y la incapacidad para poder fragmentar mensajes en la capa de enlace.

El *Internet Engineering Task Force (IETF)* ha trabajado en un estándar llamado *Static Context Header Compression o SCHC* que permite comprimir los encabezados de los protocolos IPv6, UDP y CoAP para las tecnologías *Low Power Wide Area Networks (LPWANs)* permitiendo la comunicación con un *end device* directamente a su dirección IP.

Esta tesis realiza el modelamiento, evaluación empírica y teórica de los tiempos de transmisión en el canal de subida de un enlace LoRaWAN™usando el estándar *SCHC*. Este modelamiento permite obtener una relación entre la eficiencia, el spread factor de LoRa™y el tamaño de tile del protocolo SCHC.

A mi compañera y esposa Hadee, por ser paciente y ayudarme a cumplir mis sueños. A mis hijos Benjamín y Cristóbal por aplazar más de una vez un juego de autos o un documental de la segunda guerra mundial.

A mis padres por entregarme todas las herramientas para cumplir mis metas. A mis hermanas por mirarme con admiración. Esa admiración era parte del combustible que me permitía avanzar.

A Sandra por saber guiarme en esta tesis y comprender que mis tiempos eran también para ser padre y trabajador.

A todos ustedes, muchas gracias

Este trabajo ha sido apoyado por el Centro Avanzado en Ingeniería Eléctrica y Electrónica, AC3E, Proyecto Basal FB0008, ANID, y por el Proyecto FONDECYT 1201893

Tabla de Contenido

1. Introducción **1**
- 1.1. Motivación y Antecedentes . . . 2
- 1.2. Definición del problema . . . 3
- 1.3. Hipótesis . . . 4
- 1.4. Objetivos . . . 4
  - 1.4.1. Objetivos Generales . . . 4
  - 1.4.2. Objetivos Particulares . . . 4
- 1.5. Metodologías y herramientas . . . 5
  - 1.5.1. Herramientas de software y hardware . . . 7

2. Marco Teórico y Estado del Arte **11**
- 2.1. Marco Teórico . . . 11
  - 2.1.1. Low-Power Wide-Area Network (LPWAN) . . . 11
  - 2.1.2. Long Range Wide-Area Network (LoRaWAN™) . . . 14
- 2.2. Estado del Arte . . . 21
  - 2.2.1. Internet Protocol (IP) sobre LPWAN . . . 21

3. Static Context Header Compression (SCHC) **25**
- 3.1. Introducción . . . 25
- 3.2. Compresión . . . 26
- 3.3. Fragmentación . . . 30
  - 3.3.1. Aspectos Generales . . . 30
  - 3.3.2. Modos de Fragmentación . . . 31

4. Modelado teórico de la compresión y fragmentación para LoRaWAN **34**
- 4.1. Compresión . . . 34
  - 4.1.1. Caracterización de porcentajes de compresión y tiempos de transmisión 35
- 4.2. Fragmentación . . . 38
  - 4.2.1. Modo Ack-on-Error sin pérdida de fragmentos . . . 38
  - 4.2.2. Modo Ack-on-Error con pérdida de fragmentos . . . 41
  - 4.2.3. Eficiencia en el modo Ack-on-Error . . . 45

5. PySCHC: Implementación experimental de SCHC **48**
- 5.1. Plataforma experimental . . . 48
- 5.2. Desarrollo de PySCHC . . . 49
  - 5.2.1. Arquitectura de Software . . . 49
  - 5.2.2. Problemas en el desarrollo . . . 51

5.3. Despliegue de PySCHC 55
5.3.1. Instalación en Google Cloud Platform 55

6. Análisis y Resultados **56**
6.1. Análisis de la compresión 56
6.1.1. Diseño de los experimentos - Time on Air vs Porcentaje de compresión 56
6.1.2. Resultados - Time on Air versus Porcentaje de compresión 56
6.1.3. Discusión - Time on Air versus Porcentaje de compresión 57
6.2. Análisis de la fragmentación 59
6.2.1. Diseño de los experimentos: Simulación 1 - Eficiencia versus Probabilidad de Error 59
6.2.2. Diseño de los experimentos: Experimento 1 - Eficiencia versus spreading factor versus Tamaño de *Tile* 61
6.2.3. Resultados - Eficiencia vs Probabilidad de Error 62
6.2.4. Discusión - Eficiencia vs probabilidad - comportamiento *spreading factor* 10 63
6.2.5. Resultados - Eficiencia vs spreading factor 65
6.2.6. Discusión - Eficiencia vs spreading factor 66
6.2.7. Resultados - Eficiencia vs Tamaño de Tile 67
6.2.8. Discusión - Eficiencia vs Tamaño de *tile* 69
6.3. Resumen de resultados 71

7. Conclusiones y Trabajo a futuro **73**
7.1. Conclusiones 73
7.2. Trabajos Futuros 74
7.2.1. Eficiencia para enlace de bajada en LoRaWAN 74
7.2.2. Eficiencia Empírica en ambiente con pérdidas 74
7.2.3. Implementación completa del estándar SCHC 74
7.2.4. Estándar SCHC en modo *Fog Computing* 75

Anexos **76**

A. Código para el cálculo de Eficiencia vs Probabilidad de error **77**

B. Código para el cálculo de Eficiencia Teórica vs Spread Factor **84**

C. Código para el cálculo de Eficiencia Empírica vs Spread Factor **89**

D. Código para el cálculo de Eficiencia Teórica vs Tamaño de Tile **93**

E. Código para el cálculo de Eficiencia Empírica vs Tamaño de Tile **98**

F. Proceso de instalación de PySCHC **102**

Bibliografía **104**

Índice de Tablas

2.1. Terminologías usadas segun tecnología LPWAN [elaboración propia] 13
2.2. Plan de bandas LoRaWAN™[11] 16
2.3. Tasa de Datos para la banda AU915-928 [11] 18

3.1. Ejemplo de Regla de compresión [elaboración propia] 27

6.1. Parámetros de evaluación [elaboración propia] 57
6.2. Parámetros usados para la simulación de Eficiencia v/s Probabilidad [elaboración propia] 60
6.3. Parámetros usados para el experimento de Eficiencia v/s spreading factor [elaboración propia] 62
6.4. Eficiencia vs *spreading factor* para un tamaño de *tile* de 10 bytes y una probabilidad de error igual a cero [elaboración propia] 63
6.5. Eficiencia Teórica vs *spreading factor* para un tamaño de *tile* de 10 bytes [elaboración propia] 65
6.6. Modelo teórico y empírico para la eficiencia vs *spreading factor* para un tamaño de *tile* de 10 bytes [elaboración propia] 67
6.7. Cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* para un *SCHC packet* de 400 *bytes* usando el *spreading factor* 12 [elaboración propia] 70

Índice de Ilustraciones

1.1. Ejemplo de diagrama de comunicación utilizado para modelar la fragmentación de SCHC [elaboración propia] . . . 5
1.2. División de un *SCHC packet* en *tiles* y ventanas [elaboración propia] . . . 6
1.3. Arquitectura *LPWAN* [elaboración propia] . . . 6
1.4. Arquitectura utilizada en la implementación de *SCHC* para la evaluación empírica. [elaboración propia] . . . 7
1.5. Modulo de desarrollo LoPy4 de la empresa Pycom [7] . . . 8
1.6. Diagrama de bloques para el modulo de desarrollo LoPy4 [7] . . . 8
1.7. Gateway RAK7258 de la empresa RAK Wireless (vista externa)[8] . . . 9
1.8. Gateway RAK7258 de la empresa RAK Wireless (vista interna)[8] . . . 10

2.1. Arquitectura *LPWAN* [elaboración propia] . . . 13
2.2. Tipos de dispositivos *LoRaWAN*^{TM} [elaboración propia] . . . 15
2.3. Dispositivo Clase C [elaboración propia] . . . 15
2.4. Canales *LoRaWAN*^{TM} en la banda AU915-928 [11] . . . 17
2.5. Paquete LoRa (capa física) [elaboración propia] . . . 19
2.6. IPv6 sobre LoRaWAN (Weber, 2016) [15] . . . 21

3.1. Subcapas SCHC . . . 25
3.2. Operacion de SCHC . . . 26
3.3. SCHC Context [elaboración propia] . . . 27
3.4. Estructura de un paquete SCHC [elaboración propia] . . . 28
3.5. Proceso de compresión SCHC [elaboración propia] . . . 29
3.6. División de un *SCHC packet* en *tiles* y ventanas [elaboración propia] . . . 30
3.7. Modo Ack-on-Error [elaboración propia] . . . 33

4.1. Agrupación de reglas según encabezado comprimido [elaboración propia] . . . 36
4.2. Flujo de mensajes en el modo Ack-on-Error (sin pérdidas de fragmentos) [elaboración propia] . . . 40
4.3. Flujo de mensajes en el modo Ack-on-Error (con pérdidas de fragmentos) [elaboración propia] . . . 43

5.1. Arquitectura utilizada [elaboración propia] . . . 49
5.2. Diagrama de Clases módulo PySCHC Compression [elaboración propia] . . . 50
5.3. Diagrama de Clases módulo PySCHC Node [elaboración propia] . . . 51
5.4. Diagrama de Clases módulo PySCHC Gateway [elaboración propia] . . . 52
5.5. Flujo de mensajes exitosos - Modo Ack on error [elaboración propia] . . . 54

6.1. Time on Air según el porcentaje de compresión para diferentes spreading factor 57
6.2. Time on Air variation vs Spead Factor 58
6.3. Time on Air range vs porcentaje de compresión 58
6.4. Eficiencia vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia] 62
6.5. Factor $\left(T_1 + T_2 \cdot \frac{p}{(1-p)}\right)$ vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia] 64
6.6. Factor $\left(T_1 + T_2 \cdot \frac{p}{(1-p)}\right) \cdot R$ vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia] 64
6.7. Eficiencia Teórica vs spreading factor [elaboración propia] 65
6.8. Eficiencia Empírica vs *spreading factor* [elaboración propia] 66
6.9. Eficiencia Teórica vs Tamaño de *Tile* [elaboración propia] 68
6.10. Eficiencia Empírica vs tamaño de *tile* [elaboración propia] 68
6.11. Variación en la eficiencia para el spreading factor 12 [elaboración propia] . . 69
6.12. Cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia] 70
6.13. Curvas polinómicas que se ajustan a la cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia] 71
6.14. Derivada de la cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia] 72

Capítulo 1

Introducción

En 1999, Kevin Ashton utilizó por primera vez el concepto *Internet of Things o IoT* asociado, en ese tiempo, a los envases inteligentes y a los tags RFID. Ya han pasado 20 años y este concepto está causando una revolución en la adquisición de datos, lo que sumado al *Data Analytics*, permite un automatismo nunca antes pensado.

En la actualidad existen varias tecnologías de acceso que sirven para IoT. Pero ha surgido un grupo de tecnologías como LoRaWAN^{TM}, Sigfox y NB-IoT que están diseñadas para entregar una gran cobertura y al mismo tiempo un bajo consumo de energía. Estas características son muy valoradas en las soluciones del *Internet of Things* debido a la gran densidad de dispositivos finales que no pueden estar cambiando sus baterías de forma periódica o necesitan tener un consumo menor a lo que entregan sus fuentes de alimentación.

Todos estas tecnologías diseñadas para IoT tienen la característica de no soportar direccionamiento IP debido a los pequeños tamaños de memoria para almacenar una pila de protocolos y la incapacidad para poder fragmentar mensajes en la capa de enlace. El problema con esto, es no poder conectar los nodos finales directamente a Internet[1]. Por lo anterior, no se pueden utilizar protocolos que faciliten la gestión y el acceso a los dispositivos finales.

El *Internet Engineering Task Force (IETF)* ha trabajado en un estándar llamado *Static Context Header Compression o SCHC* [1] el cual permite comprimir los encabezados de los protocolos IPv6, UDP y CoAP para *Low Power Wide Area Networks (LPWANs)* permitiendo comunicar con un *end device* directamente a su dirección IP [2].

Si bien existe una literatura en torno al estándar *SCHC* y otros mecanismos de compresión en *Low Power Wide Area Networks (LPWANs)*, todavía son escasos los estudios sobre el modelamiento del flujo de mensajes y tiempos asociados en su transmisión, considerando que el protocolo fue recién aprobado como estándar el mes de abril de 2020. Estos flujos de mensajes son definidos en el estándar, pero no así el impacto del *spreading factor* de *LoRa*^{TM} sobre los tiempos de transmisión.

Dado que la eficiencia de un protocolo es una característica que permite medir el desem-

[1]La conexión de las *LPWANs* a internet se hacía a nivel de aplicación casi exclusivamente.

peño del mismo, esta tesis entrega la relación entre el *spreading factor*, un parámetro físico de la tecnología *LoRa*^{TM}, el tamaño del *tile*[2] y la eficiencia del protocolo *SCHC*. Lo anterior se consigue evaluando los tiempos de transmisión de un fragmentos SCHC cuando se modifican valores en la subcapa de compresión y fragmentación. En el caso de la subcapa de compresión, se evalúa el *time on air* para cada regla en función de los distintos *spreading factors* de la tecnología *LoRa*^{TM}. En el caso de la fragmentación, se realiza el modelamiento y evaluación empírica y teórica de los tiempos de transmisión en el canal de subida de un enlace LoRaWAN$^{\text{TM}}$usando el modo de fragmentación *Ack-on-error* del estándar *SCHC*. Este modelamiento permite obtener la relación entre la eficiencia del protocolo *SCHC*, la latencia y el *spreading factor*.

1.1. Motivación y Antecedentes

Sin lugar a dudas estamos en una década donde los datos han tomando relevancia en nuestra sociedad[3]. Las tecnologías como *Data Analytics* y *Machine Learning* nos permiten automatizar procesos y crear modelos que sirven para anticipar eventos. En muchos casos, los datos que alimentan estas tecnologías provienen de variables de campo, como temperatura, velocidad de viento, cantidad de vehículos en una carretera, etc. Estas variables son obtenidas a través de sensores que en conjunto crean el llamado *Internet of Things*. Por lo tanto, la relación de interdependencia entre *Data Analytics* e *IoT* revela la importancia de contar con tecnologías de comunicaciones confiables y robustas. Estas serán las encargadas de entregar los datos y crear el *Big Data* para posterior análisis.

Dentro de todas las soluciones y arquitecturas que han surgido para satisfacer las necesidades del *Internet of Things* destacan las redes *Low Power Wide Area Networks (LPWANs)*. Estas redes están diseñadas para tener una amplia cobertura y un mecanismo de comunicación que permite a los sensores o dispositivos finales tener un consumo bajo, haciendo a la solución sustentable en el tiempo. Por ejemplo, cada nodo puede durar años sin tener que renovar su fuente de poder. Debido al requerimiento de tener un bajo consumo de energía, los nodos pueden transmitir muy pocos mensajes por día y además poseen un bajo ancho de banda, haciendo que este tipo de redes sean también llamadas redes con restricciones o *resource-constrained networks*.

Long Range Wide Area Networks o LoRaWAN^{TM}, es una tecnología perteneciente a la familia *LPWAN*. Segun *LoRa Alliance*, al mes de abril de 2019, en todo el mundo hay más de 113 redes LoRaWAN$^{\text{TM}}$en 55 países en diversas etapas de implementación comercial. Según Semtech Corporation, al mes de abril de 2019, el número acumulado de nodos finales de *LoRa* alcanzó los 97 millones. El crecimiento de la solución LoRaWAN$^{\text{TM}}$se está acelerando porque el ecosistema de dispositivos se ha expandido rápidamente para abordar un conjunto más amplio de mercados verticales como son *Utilities* (Electricidad, Agua Potable y Gas), edificios inteligentes, logística, seguimiento de activos y agricultura inteligente, entre otros [3].

[2]Un mensaje SCHC está fragmentado en partes, en lo sucesivo denominados *tiles*

[3]https://velogig.com/entendiendo-la-relacion-entre-internet-de-las-cosas-iot-y-big-data/

El protocolo LoRaWAN™, posee una MTU[4] del orden de las decenas de bytes y no existe la fragmentación en capa de enlace. Debido a estas limitaciones, no es posible la transmisión de protocolos de capas superiores típicos de Internet, como: *Internet Protocol (IP)*, *Transmission Control Protocol (TCP)*, *User Datagram Protocol (UDP)*, *Hypertext Transfer Protocol (HTTP)* y *Constrained Application Protocol (CoAP)*. Además, la arquitectura está diseñada para que cada nodo final no posea una dirección IP a la cual acceder directamente desde Internet; en cambio, un elemento central denominado *Network Server* recibe los mensajes destinados a los nodos finales y es el encargado de hacer la traducción entre una *Uniform Resource Locator (URL)* y un identificador del dispositivo final para el protocolo LPWAN.

A pesar que las limitaciones mencionadas no han frenado el despliegue de redes LoRaWAN, sí han creado islas de dispositivos que dependen de entidades "traductoras"para poder interoperar con la Internet. Por ejemplo, existe una variedad de protocolos para el acceso remoto como *Secure SHell (SSH)*, *Secure File Transfer Protocol (SFTP)* los cuáles permitirían a un usuario remoto acceder a un nodo final LoRaWAN™a través de Internet para actualizar su *firmware*, ejecutar un script de forma remota u obtener un archivo almacenado en la memoria del dispositivo. Sin embargo, dichas funcionalidades hoy no son soportadas, haciendo compleja la administración remota de redes LoRaWAN e incrementado la dependencia de soluciones propietarias que no contribuyen a crear sistemas interoperables. Sumado a lo anterior, el interés de poder asignar una dirección IP a un nodo final *LPWAN* y además permitir el envío de mensajes IP radica en poder usar las características de movilidad, segmentación y transporte que entrega *IPv6* como protocolo de la capa de red de Internet.

1.2. Definición del problema

Como se indicó en la introducción, la implementación del estándar SCHC en un nodo LoRaWAN™permite que un *end device* pueda comunicarse directamente con un nodo en Internet [4]. Esta comunicación directa, al usar una nueva capa de adaptación, trae consigo nuevos roles a nivel de capa de enlace de datos como la compresión y la fragmentación.

A nivel de fragmentación el estándar SCHC presenta tres modos de funcionamiento [1] y en el caso de LoRaWAN™se utilizan dos de ellos, uno para *uplink* (*Ack-on-Error*) y otro para *downlink* (*Ack-Always*) [5]. Para ninguno de ellos se ha determinado la relación que existe entre los distintos *spreading factors* y la eficiencia que pueda existir en un enlace LoRaWAN™. Determinar los modelos apropiados que describan el desempeño de la tecnología bajo la operación de SCHC para sus diferentes configuraciones de canal, permitiría escoger un *spreading factor* que garantice una entrega exitosa dependiendo del modo de fragmentación usado.

En resumen, al no existir estudios que entreguen una relación entre las variables de desempeño de un enlace *LoRaWAN*™y los distintos modos de operación definidos en el estándar *SCHC*, no se pueden determinar los parámetros de *LoRa* que optimizan el trabajo del protocolo *SCHC* en función de la entrega exitosa y oportuna de los mensajes que transporta.

[4]Maximum Transmission Unit

1.3. Hipótesis

Al modelar los flujos de mensajes para la compresión y fragmentación del estándar *SCHC*, enviados a través de una red *LoRaWAN*^{TM}, se puede determinar que existe una relación lineal e inversamente proporcional entre la eficiencia del protocolo *SCHC* y el *spreading factor* de la tecnología *LoRa*.

1.4. Objetivos

1.4.1. Objetivos Generales

- Modelar y evaluar teórica y empíricamente la eficiencia de un enlace de subida LoRaWAN para distintos *spreading factors*, dadas las reglas de compresión y modos de fragmentación del estándar SCHC.

1.4.2. Objetivos Particulares

- Proponer métricas que permitan evaluar la eficiencia de la compresión de *SCHC* en LoRaWAN$^{\mathrm{TM}}$
- Determinar un modelo para la eficiencia del modo de fragmentación *Ack-on-Error* del estándar *SCHC*.
- Caracterizar el impacto de la compresión *SCHC* en los tiempos de transmisión de LoRaWAN$^{\mathrm{TM}}$mediante evaluación de las métricas propuestas.
- Evaluar el desempeño de un enlace LoRaWAN$^{\mathrm{TM}}$con fragmentación *SCHC* en modo *Ack-on-Error*, aplicando el modelo teórico a diferentes condiciones de canal y validando los resultados en un ambiente experimental.

1.5. Metodologías y herramientas

Para la capa de compresión, el estándar *SCHC* utiliza reglas para poder indicar qué encabezados se deben o no comprimir. Esta tesis realiza un análisis teórico asignando un porcentaje de compresión a cada una de las reglas. Dada la cantidad finita de encabezados, existirá una cantidad finita de reglas de compresión y para cada una de ellas se le asigna un porcentaje de compresión. A pesar que la cantidad de reglas es finita, existe un gran número de combinaciones para indicar cuáles encabezados se comprimen y cuáles no. Esto quiere decir que si se cuenta la cantidad de reglas que definen que encabezados se deben comprimir y cuáles no, se obtienen $2^{12} = 4096$ reglas. Por ejemplo, una regla puede indicar que se comprimen los encabezados *Version*, *Traffic Class* y *Flow Label* y otra regla puede indicar que no se comprime ningún encabezado *IPv6* pero si los 4 encabezados de *UDP*. Así, la cantidad total de reglas será 4096 ya que existen 12 headers que se pueden comprimir (8 de *IPv6* y 4 de *UDP*)

Para la capa de fragmentación, el estudio está basada en el **análisis teórico** de los mensajes SCHC intercambiados en un enlace de subida o *uplink* para LoRaWAN. El protocolo SCHC define los flujos en los documentos *draft-ietf-lpwan-schc-over-lorawan-07* [5] y *RFC8724* [1]. Con esta información, se crea un diagrama de comunicación entre el Transmisor (nodo) y el Receptor (Application Server) como muestra la figura 1.1.

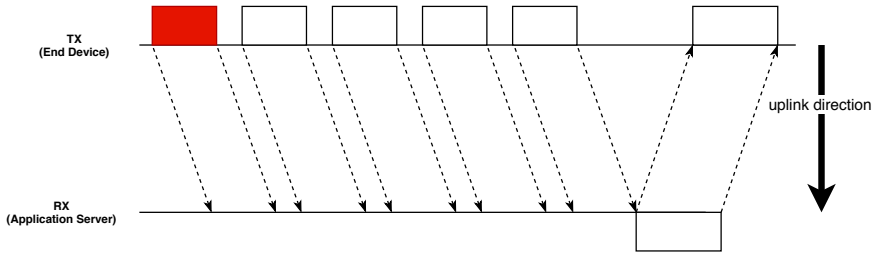


Figura 1.1: Ejemplo de diagrama de comunicación utilizado para modelar la fragmentación de SCHC [elaboración propia]

Con el diagrama de comunicación, se calculan los tiempos de transmisión y recepción de cada uno de los mensajes. Este cálculo requiere conocer cuánto tiempo toma transmitir una cierta cantidad de bits en un canal *LoRa*. Para ello se utiliza el concepto *Time on Air* definido por *Semtech* en su documento *SX1272/3/6/7/8: LoRa Modem Designer's. Guide* [6]

Para poder soportar el mecanismo de fragmentación el estándar define algunos elementos comunes como son: *tiles*, ventanas, mensajes y campos. Un *SCHC packet* o paquete SCHC es dividido en partes llamadas *tiles*. Los grupos de *tiles* son llamados ventanas. La figura 3.6 presenta este concepto.

Con los tiempos de cada mensaje se grafica la eficiencia teórica promedio del protocolo en función del *spreading factor* y el tamaño de cada *tile* medido en bytes para una enlace sin pérdidas. Además se grafica la eficiencia teórica promedio en función de la probabilidad de error para un enlace con pérdidas a través del método Montecarlo. Para modelar el funcionamiento de la fragmentación se toman como base los protocolos para control de errores y retransmisión de mensajes o ARQ. Así, el cálculo de la eficiencia está basado en la división

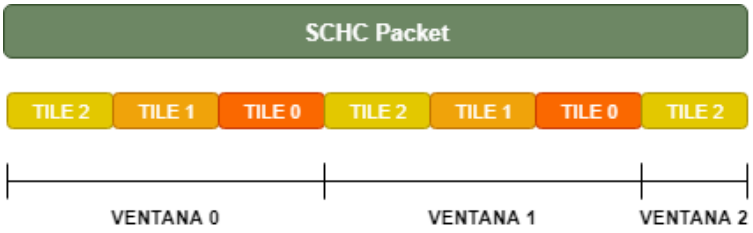


Figura 1.2: División de un *SCHC packet* en *tiles* y ventanas [elaboración propia]

entre el tiempo de envío de un *SCHC Regular fragment* y el tiempo de enviar exitosamente una ventana SCHC.

La segunda etapa del estudio considera una **análisis empírico**, a través de la implementación del estándar *SCHC* tanto de su capa de compresión como fragmentación. El código es llamado PySCHC, está escrito en Python y es un de los tres códigos que implementan el estándar a la fecha de publicar este trabajo. Además, se realiza la construcción de un escenario LoRaWAN para pruebas del protocolo. Este escenario considera solamente el enlace de subida o *uplink* el cual utiliza el modo Ack-on-Error definido en el estándar SCHC.

El escenario implementado está basado en la topología mostrada en la figura 1.3. Cada uno de los componentes están descritos en detalle en la sección 1.5.1. El diseño considera cuatro bloques: *End-device*, *Radio Gateway*, *Network Server* y *Application Server*.

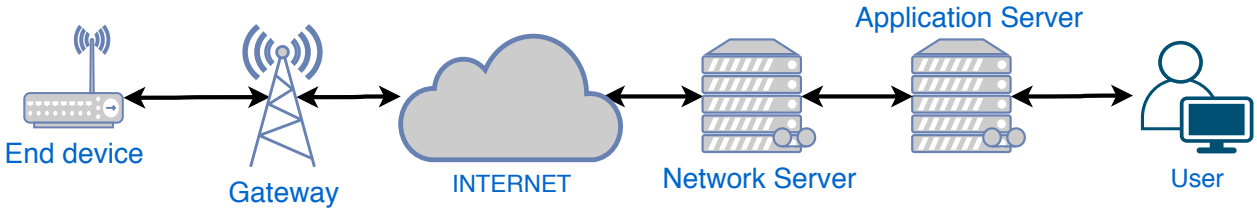


Figura 1.3: Arquitectura *LPWAN* [elaboración propia]

Como se dijo anteriormente, en la implementación del estándar SCHC, se considera solo el uso del enlace uplink, esto es, el rol de compresión/fragmentación está ubicado en el *end-device* y el rol de descompresión/desfragmentación está ubicado en el *application server*, tal como lo muestra la figura 1.4.

Se ha seleccionado a Python como lenguaje de desarrollo por las siguientes razones:

- La placa de desarrollo que representa al *end-device* es programada a través de Micropython.
- Al mismo tiempo, el código desarrollado para el *end-device* puede ser implementado en Google Cloud Platform reutilizando gran parte del desarrollo.

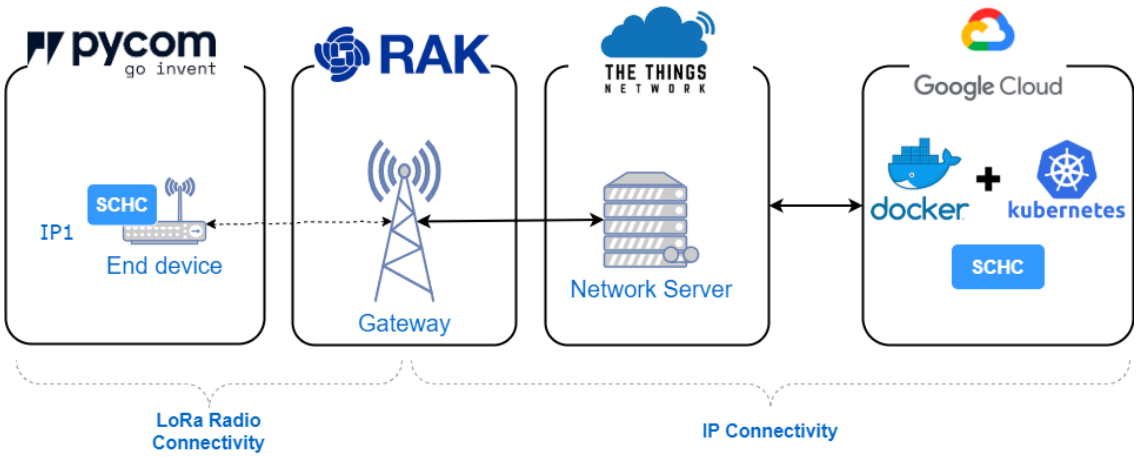


Figura 1.4: Arquitectura utilizada en la implementación de *SCHC* para la evaluación empírica. [elaboración propia]

1.5.1. Herramientas de software y hardware

Modelado

Para poder realizar el modelado se utiliza el software de cálculo numérico Matlab R2018a (versión 9.4). Este software trae incluido un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). A nivel de dependencias no se utilizan *toolboxes* y solo se usan funciones incluidos en el *core* de Matlab.

Implementación experimental

Para poder realizar la implementación se considera el diagrama de una red LPWA definida en [1], el cual considera cuatro bloques: *End-device*, *Radio Gateway*, *Network Server* y *Application Server* tal como lo muestra la figura 1.3

Para el caso del *End-device* se utiliza un nodo LoPy4 de la empresa Pycom[5]. Este nodo es una placa de desarrollo que soporta cuatro tipos de red (LoRa, Sigfox, WiFi, Bluetooth). Además implementa MicroPython, una versión eficiente del lenguaje de programación Python 3 que incluye un pequeño subconjunto de la biblioteca estándar de Python y está optimizado para ejecutarse en microcontroladores y en entornos restringidos, como el microcontrolador ESP32. La figura 1.5 muestra el modulo de desarrollo LoPy4 con sus principales características.

Las especificaciones de esta placa de desarrollo son:

- CPU: Xtensa^{TM}dual–core 32–bit LX6 microprocessor(s), hasta 600 DMIPS
- Memoria: RAM: 520KB + 4MB y External flash: 8MB

[5]https://pycom.io/product/lopy4/

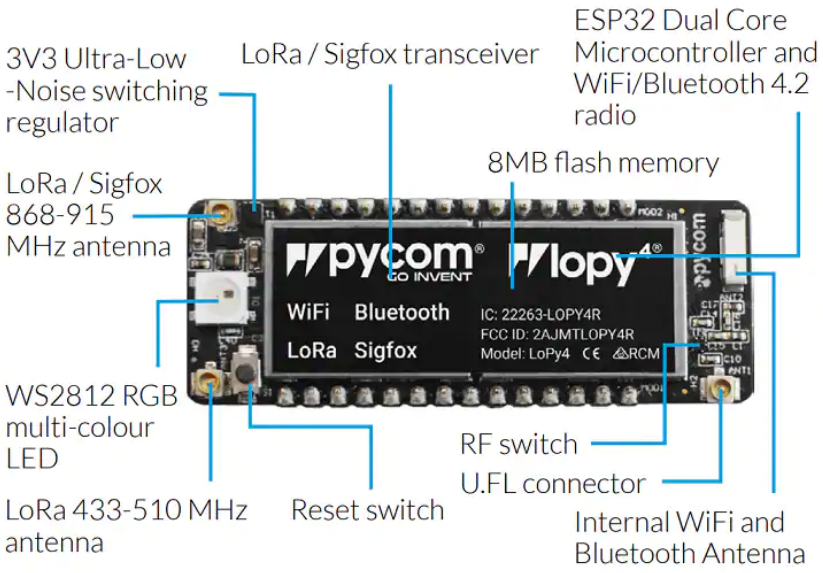


Figura 1.5: Modulo de desarrollo LoPy4 de la empresa Pycom [7]

- WiFi: 802.11b/g/n 16mbps
- RTC: Corriendo a 150kHz
- Seguridad: Soporte de SSL/TLS y WPA Enterprise security
- Hash/encryption: SHA, MD5, DES, AES

El diagrama de bloques de la LoPy4 es el mostrado en la figura 1.6. Este diagrama presenta los bloques agrupados por funcionalidad.

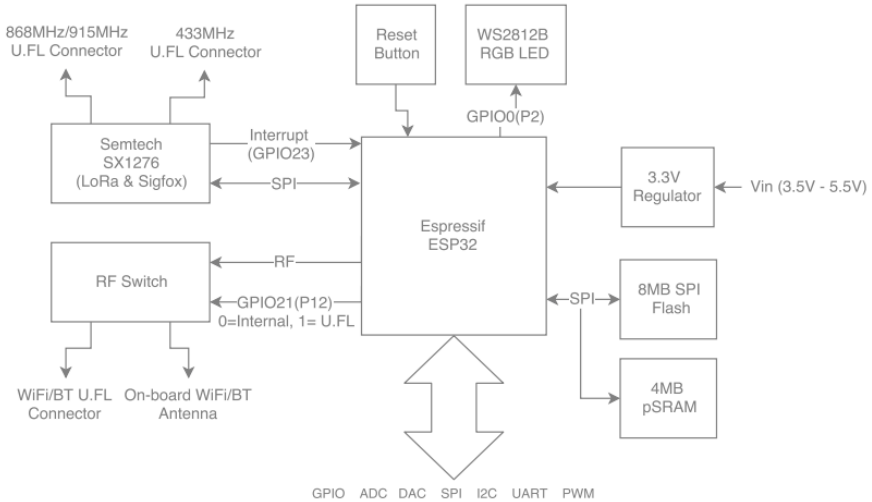


Figura 1.6: Diagrama de bloques para el modulo de desarrollo LoPy4 [7]

Para el *Radio Gateway* se utiliza un gateway LoRaWAN™modelo RAK7258 de la marca RAK Wireless. Está diseñado para interiores y viene con una opción de conectividad Ethernet estándar (compatible con Power over Ethernet). Además, el soporte de WiFi lo hace fácilmente configurable. Por lo tanto, hay múltiples opciones de conectividad a las que recurrir en caso

de una interrupción de la red. El software de gestión y configuración se basa en OpenWRT. Este gateway tiene 8 canales LoRa, soporta la versión 1.0.2 del stack LoRaWAN^{TM}y está diseñado para la banda AU915 que va desde los 915 MHz hasta los 928 MHz. La figura 1.7 muestra una imagen externa y la figura 1.8 una imagen interna del gateway con la descripción de cada puerto y bloque respectivamente.

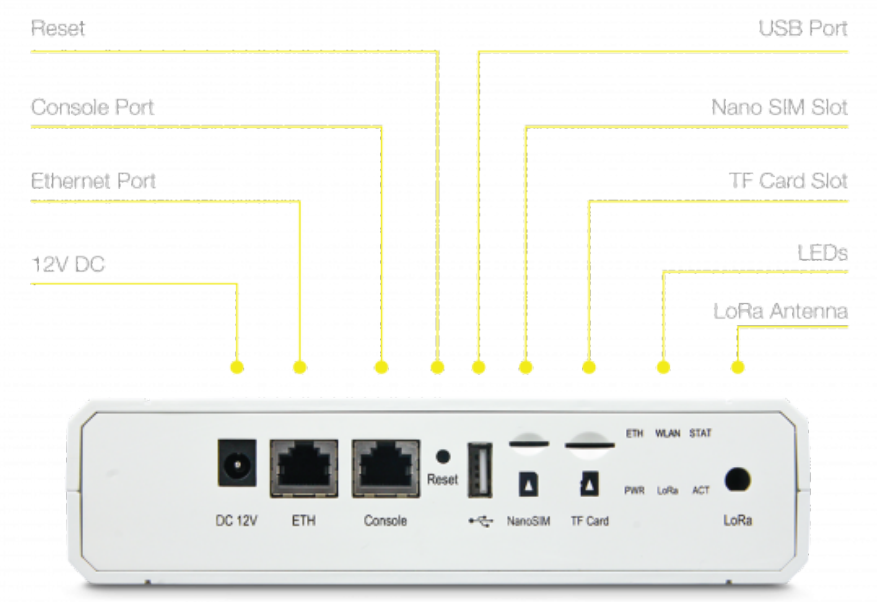


Figura 1.7: Gateway RAK7258 de la empresa RAK Wireless (vista externa)[8]

El *Network Server* es implementado por *The Things Network* o TTN. *The Things Network* es una red global de gateways que forman una comunidad mundial para entregar cobertura LoRaWAN. Todos estos *gateways* están conectados a los *network servers* de *TTN* desde donde se puede obtener la información enviada por cada sensor o *end device*. Además de la tecnología de comunicaciones, *TTN* ha hecho un esfuerzo de desarrollo en la parte de servidor, construyendo todo el *backend* de la red y dando soporte a los *gateways* distribuidos por el globo. Este *backend* es el que maneja la duplicidad de mensajes, orquestación de mensaje de bajada y gestión de integraciones con plataformas. *TTN* ofrece también capacidades de integración por *HTTP*, además de una serie de APIs en distintos lenguajes como: *Go*, *Java*, *Node-RED* y *Node.js*.

Finalmente, el *Application Server* es implementado a través de *Doker Container* en la plataforma de *Google* llamada *Google Kubernetes Engine*. *Docker container* permite ejecutar una aplicación a través del concepto de microservicio. Un *Docker container* permite la abstracción del sistema operativo y del hardware a nivel de aplicación tal como lo haría una máquina virtual, pero a diferencia de virtualizar el sistema operativo, en un container se virtualiza la aplicación y sus bibliotecas. Cuando el container ya está creado, es posible desplegarlo en un ambiente como *Google Kubernetes Engine*. La explicación detallada de *Docker Container* y *Google Kubernetes Engine* está fuera del alcance de esta tesis.

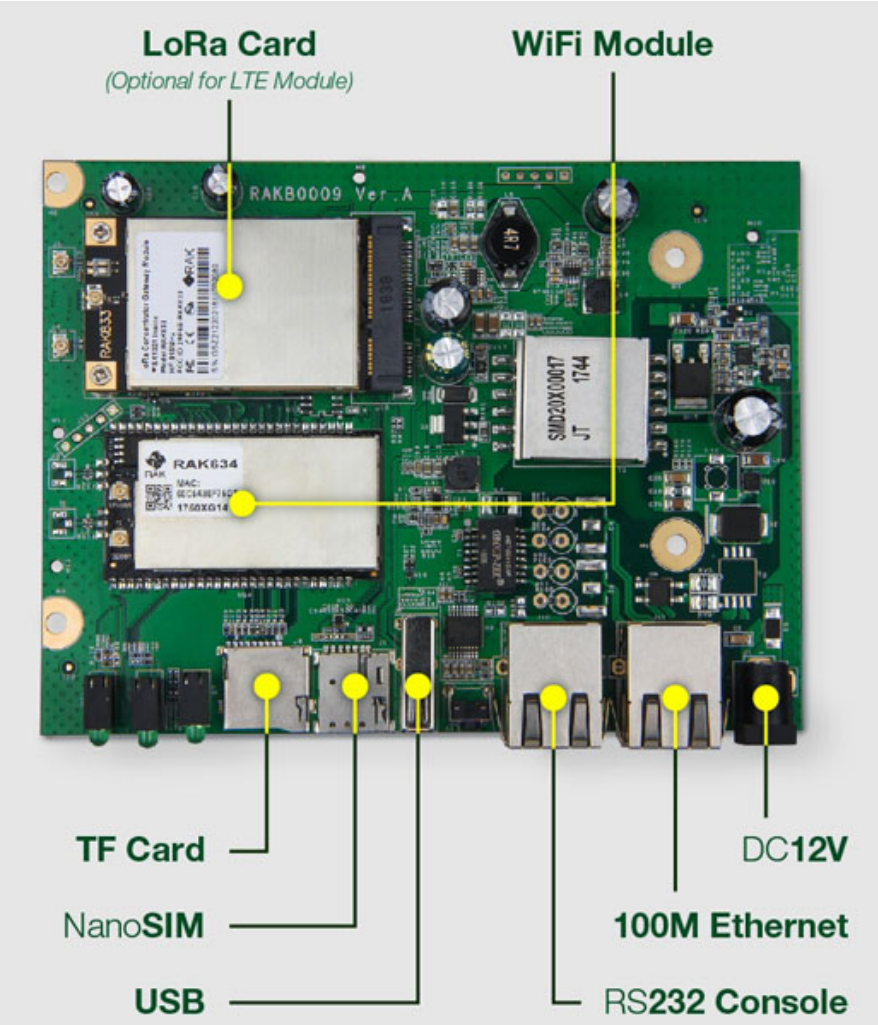


Figura 1.8: Gateway RAK7258 de la empresa RAK Wireless (vista interna)[8]

Capítulo 2

Marco Teórico y Estado del Arte

En este capítulo se presentan los conceptos fundamentales para comprender las tecnologías *LPWAN* y en particular el estándar *LoRaWAN*^{TM}. Se presenta además el funcionamiento de los protocolos para control de errores y retransmisión de mensajes o *ARQ* necesarios para modelar el funcionamiento de la fragmentación del protocolo *Static Context Header Compression (SCHC)*. *SCHC* es un estándar que permite el transporte de *IPv6* y otros protocolos sobre redes *LPWAN*. *SCHC* se verá en detalle en el capítulo 3.

En la segunda parte del capítulo se revisa el estado del arte sobre el uso del protocolo de Internet (IP) en redes *LPWAN*, ya que la transmisión de protocolos con encabezados del orden de las decenas de bytes no es soportada nativamente por redes *LPWAN*.

2.1. Marco Teórico

2.1.1. Low-Power Wide-Area Network (LPWAN)

Las comunicaciones para *Internet of Things o IoT* cada vez demandan mayor cobertura y cantidad de dispositivos por área geográfica. Las tecnologías *Bluetooth Low Energy (BLE), ZigBee y Z-Wave* no están diseñados para ello, debido a su corto alcance. Estas características permiten que otras tecnologías surjan como solución a los problemas de autonomía energética y cobertura.

LPWAN es la definición para un grupo de tecnologías inalámbricas de red de área amplia o *WAN*, por sus siglas en inglés, diseñadas para tener un baja tasa de datos que va de los 0,3 kbps a los 50 kbps, además de un bajo consumo de energía que permite a los dispositivos finales poder durar años sin cambiar sus baterías y una cobertura geográfica del orden de los kilómetros [9].

En comparación con otras tecnologías inalámbricas como *WiFi*, *LPWAN* se ubica como un grupo de tecnologías que renuncian a tener una tasa de datos alta en favor de una mayor

cobertura, permitiendo distancias del orden de los 15 kilómetros en zonas rurales.

Los tres tecnologías más representativas de la categoría *LPWAN* son: *Sigfox, NB-IoT y LoRaWAN™*. A continuación se da una breve descripción para cada una de ellas.

- **Sigfox**: posee una modulación patentada denominada *Ultra Narrow Band o UNB*, por sus siglas en inglés. *Sigfox* implementa a través de sus socios, sus propias estaciones bases en la banda ISM sub-GHz (ejemplo 868 MHz en Europa, 915 MHz en Norte América y 433 MHz en Asia) [10]. En el caso de Chile la banda utilizada es desde los 920,8 MHz a los 923,4 MHz. Los nodos finales pueden solo transmitir 140 mensajes por día y la carga útil en *uplink* es de 12 bytes. En *downlink* solo se pueden recibir 4 mensajes por día con una carga útil de 8 *bytes*, lo que hace difícil la capacidad de confirmación de mensajes.
- **NB-IoT**: coexiste con redes *LTE y GSM* bajo bandas licenciadas utilizando un *resource block* de 200 KHz de ancho de banda. A pesar de compartir la arquitectura de red con *LTE*, está modificado para disminuir el consumo energético evitando funcionalidades como la medición de calidad de radio y agregación. Utiliza modulación *QPSK* sobre *FDMA* en el enlace *uplink* y *OFDMA* en el *downlink*, con una tasa de datos máxima de 200 kbps y 20 kbps, respectivamente.
- **LoRaWAN™**: es un estándar que utiliza la modulación *LoRa* patentada por la empresa *Semtech*. Esta modulación está basada en la técnica *chirp spread spectrum o CSS*, por sus siglas en inglés y opera en la banda ISM sub-GHz (ejemplo 863-870MHz en Europa, 902-928MHz en Norte América y 779-787MHz en Asia) [10]. En el caso de Chile la banda utilizada es desde los 915 MHz a los 928 MHz. *LoRaWAN™* permite comunicación bidireccional y soporta 6 velocidades de transmisión o *spreading factors*. Los dispositivos finales pueden ser de 3 tipos o clases, donde la diferencia entre ellos es el tiempo que permanecen activos en recepción.

Terminología en común

A pesar de que estas tres tecnologías tienen diferencias en el tipo de modulación, frecuencia de operación (licenciada o no licenciada), tasa de datos máxima y si el estándar es propietario o abierto, todas ellas comparten arquitecturas similares, pero terminologías diferentes. La figura 2.1 muestra la arquitectura genérica para redes tipo *LPWAN* definido por el *IETF* en [4].

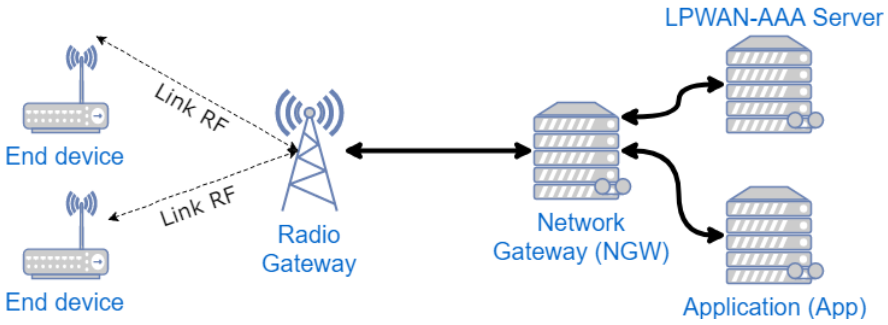


Figura 2.1: Arquitectura *LPWAN* [elaboración propia]

- ***End-device***: dispositivo final o **cosa**. Puede ser un sensor o actuador. Puede haber una gran cantidad de ellos conectados a un *Radio Gateway*
- ***Radio Gateway***: encargado de comunicarse con los *end-devices* a través de la tecnología inalámbrica (*LoRa, Sigfox y NB-IoT*) y con el Network Gateway a través de IP.
- ***Network Gateway***: es el nodo encargado de la conexión entre el *Radio Gateway* e Internet. Permite que los *end-devices* puedan comunicarse a Internet.
- ***LPWAN-AAA Server***: nodo responsable de la autorización y autenticación de los usuarios
- ***Application***: nodo encargado de la capa de aplicación que conecta con el *Network Gateway* y transmite las solicitudes al *end-device*

Los elementos de una red *LPWAN* tienen nombres diferentes según la tecnología, pero en general, los roles son muy similares. La tabla 2.1 muestra una comparación entre los distintos nombres por tecnología.

Function/Technology	**LoRaWAN**	**NB-IoT**	**Sigfox**
Sensor, Actuator, device, object	End Device	User Equipment	End Point
Transceiver Antenna	Gateway	Evolved Node B	Base Station
Server	Network Server	PDN GW/SCEF	Service Center
Security Server	Join Server	Home Suscriber Server	Registration Authority
Application	Application Server	Application Server	Network Application

Tabla 2.1: Terminologías usadas segun tecnología LPWAN [elaboración propia]

Dado que esta tesis se enfoca en *LoRaWAN*^{TM}, la siguiente sección explica en detalle esta tecnología.

2.1.2. Long Range Wide-Area Network (LoRaWAN^{TM})

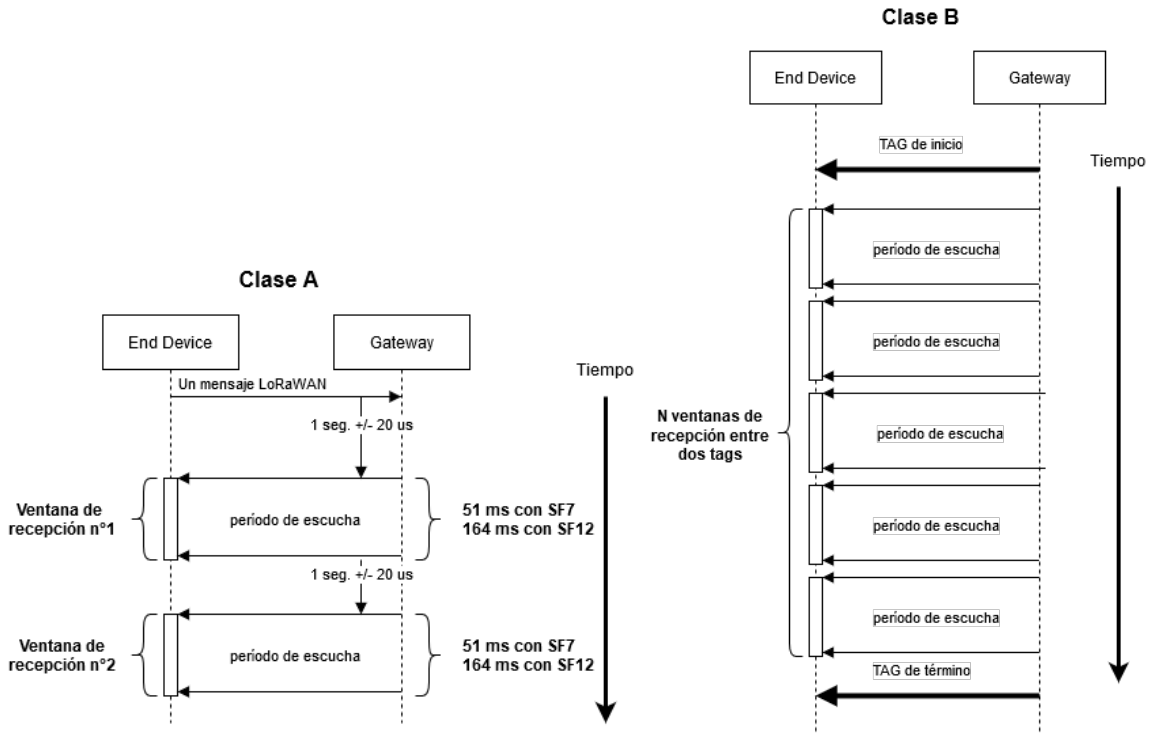
LoRaWAN^{TM} es un protocolo perteneciente a las redes de área amplia y baja energía *LPWAN* y está diseñado para satisfacer los requerimientos del *Internet of Things*. Con él es posible conectar las **cosas** a baterías y cubrir varios kilómetros usando una comunicación bidireccional asimétrica.

Arquitectura y dispositivos

La arquitectura *LoRaWAN*^{TM} es una estrella donde el centro está formado por el *Gateway* que comunica a los *End-devices* con el *Network Server* y viceversa. La comunicación entre el *Network Server* y el *Gateway* es a través de mensajes *LoRaWAN*^{TM} encapsulados sobre *UDP* e *IP*. Luego el *Gateway* desencapsula el mensaje *LoRaWAN*^{TM} y lo transmite hacia los *End-devices* usando modulación *LoRa* [4]. La figura 1.3 muestra la arquitectura de red *LoRaWAN*^{TM}.

El estándar define tres clases de *end-devices* para satisfacer las distintas necesidades de las redes *LPWAN*

- **Clase A**: Todos los *end-devices* deben soportar esta clase. La comunicación siempre es iniciada por el *end-device*. Cada transmisión en dirección *uplink* es seguida por dos ventanas para recepción de mensajes en sentido *downlink*, lo que permite una comunicación bidireccional. El acceso al medio para los dispositivos que usan esta clase es del tipo *ALOHA*.
- **Clase B**: los dispositivos pertenecientes a esta clase heredan las características de la Clase A y agregan ventanas de recepción programadas para los mensajes *downlink*. Para ello se utilizan *beacons* sincronizados por el *gateway* que permiten abrir las ventanas de recepción en el *end-device*.
- **Clase C**: los dispositivos pertenecientes a esta clase utilizan el mismo mecanismo de transmisión que los de las clases anteriores, pero mantienen abierta la ventana de recepción en todo momento mientras el *end-device* no transmita. Esto provoca un aumento en el consumo energético, pero una disminución en la latencia del *downlink*.



(a) Clase A (b) Clase B

Figura 2.2: Tipos de dispositivos $LoRaWAN^{TM}$ [elaboración propia]

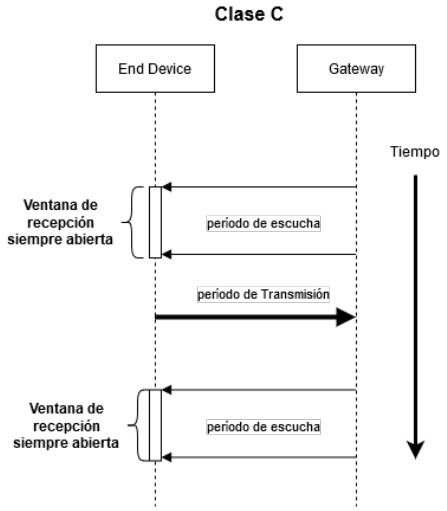


Figura 2.3: Dispositivo Clase C [elaboración propia]

Plan de Frecuencias

El estándar define 10 bandas de operación [11]. Las frecuencias usadas están en las bandas de 400 MHz, 700 MHz, 800 MHz y 900 MHz. La tabla 2.2 muestra todas las bandas actuales con su respectivas frecuencias.

Para el caso de Chile, se pueden utilizar los *Channel Plan* **AU915-928, AS923, US902-**

Channel Plan	Name
EU863-870	EU868
US902-928	US915
CN779-787	CN779
EU433	EU433
AU915-928	AU915
CN470-510	CN470
AS923	AS923
KR920-923	KR920
IN865-867	IN865
RU864-870	RU864

Tabla 2.2: Plan de bandas LoRaWAN™[11]

928 según [11]. Cada banda es dividida en canales los cuales son explicados a continuación:

- **AU915-928**: Esta banda puede ser dividida en los siguientes canales:
  - Para el enlace *UPLINK*, 64 canales de 125 KHz de ancho de banda numerados del 0 al 63, con posibilidad para usar DR0 a DR5 (el detalle de los *Data Rates o DR* son vistas en la sección 2.1.2). Se debe utilizar modulación *LoRa* y *coding rate* 4/5.
  - Para el enlace *UPLINK*, 8 canales de 500 KHz de ancho de banda numerados del 64 al 71, con posibilidad para usar solo DR6 (el detalle de los *Data Rates o DR* son vistas en la sección 2.1.2). Se debe utilizar modulación LoRa.
  - Para el enlace *DOWNLINK*, 8 canales de 500 KHz de ancho de banda numerados del 0 al 7, con posibilidad para usar DR8 a DR13 (el detalle de los *Data Rates o DR* son vistos en la sección 2.1.2). Se debe utilizar modulación *LoRa*.
- **AS923**: Esta banda va desde los 923 MHz hasta los 923,5 MHz. En ella los canales pueden ser asignados libremente por el operador de la red, pero debe al menos implementar los siguientes:
  - Dos canales con modulación *LoRa* y 125 KHz de ancho de banda. La frecuencia central de los canales es 923,20 MHz y 923,40 MHz. Se puede usar DR0 a DR5.
- **US902-928**: Esta banda puede ser dividida en los siguientes canales:
  - Para el enlace *UPLINK*, 64 canales de 125 KHz de ancho de banda numerados del 0 al 63, con posibilidad para usar DR0 a DR3 (el detalle de los *Data Rates o DR* son vistos en la sección 2.1.2), se debe utilizar modulación *LoRa* y *coding rate* 4/5.
  - Para el enlace *UPLINK*, 8 canales de 500 KHz de ancho de banda numerados del 64 al 71, con posibilidad para usar solo DR4 (el detalle de los *Data Rates o DR* son vistos en la sección 2.1.2), se debe utilizar modulación *LoRa*.
  - Para el enlace *DOWNLINK*, 8 canales de 500 KHz de ancho de banda numerados del 0 al 7, con posibilidad para usar DR8 a DR13 (el detalle de los *Data Rates o DR* son vistos en la sección 2.1.2), se debe utilizar modulación *LoRa*.

En esta tesis se utilizará la banda AU915-928. Los canales asociados a esta banda son

ilustrados en la figura 2.4. Es esta figura, los canales en color verde corresponden a canales de subida de 125 KHz. Los canales de color celeste corresponden a canales de subida de 500 KHz. Los canales de color amarillo corresponden a canales de bajada de 500 KHz de ancho de banda.

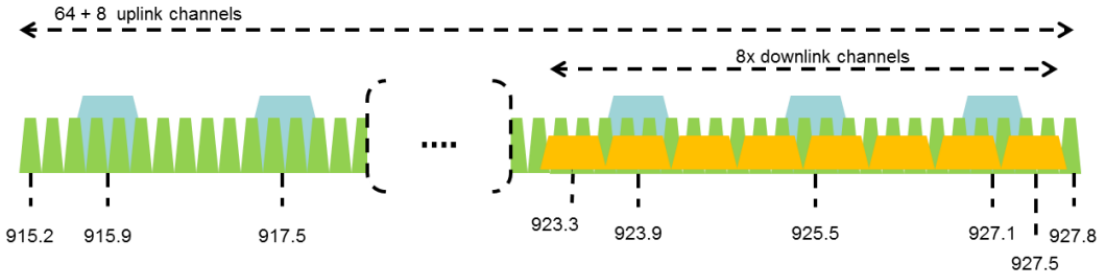


Figura 2.4: Canales $LoRaWAN^{TM}$ en la banda AU915-928 [11]

Data Rates

La tasa de datos es la velocidad de transmisión medida en *bits* por segundo (bps) que un canal *LoRaWAN*^{TM} puede soportar. En *LoRaWAN*^{TM} la tasa de datos es de carácter discreto y modificar su valor involucra un *trade-off* entre el rango de cobertura (en kilómetros) y el tiempo de transmisión del mensaje o *Time on Air*[12].

Por ejemplo, un *Data Rate 0 (DR0)* involucra un *spreading factor* 12 y su máxima tasa de datos es de 250 bps, pero con un *spreading factor* 12 es posible alcanzar una mayor distancia que con un *spreading factor* 8.

La tabla 2.3 muestra las distintos Data Rates soportadas por *LoRaWAN*^{TM}

Data Rate o DR	Configuración	Tasa de Datos (bps)
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF8 / 500 kHz	12500
7	Reserved for Future Usage	
8	LoRa: SF12 / 500 kHz	980
9	LoRa: SF11 / 500 kHz	1760
10	LoRa: SF10 / 500 kHz	3900
11	LoRa: SF9 / 500 kHz	7000
12	LoRa: SF8 / 500 kHz	12500
13	LoRa: SF7 / 500 kHz	21900
14	Reserved for Future Usage	
15	Defined in LoRaWAN^{TM}	

Tabla 2.3: Tasa de Datos para la banda AU915-928 [11]

Time on Air

El *Time on Air* es un parámetro que permite calcular el tiempo que demora el transmisor en insertar un paquete en el medio inalámbrico. El cálculo de este tiempo es útil para el diseño de un sistema LoRaWAN, ya que considera los retardos en la transmisión en función del *spreading factor*.

Para calcular el *Time on Air* es necesario definir la duración de un símbolo. Esta duración es mostrada en la ecuación 2.1. En [13] se definen los siguientes valores para LoRaWAN: $n_{preamble}$ es igual a 8 y los *spreading factors* (SF) van de 7 a 12. Para el caso del Ancho de Banda BW pueden ser canales de $125KHz$, $250KHz$ o $500KHz$.

$$T_{sym} = \frac{2^{SF}}{BW} \tag{2.1}$$

Por otro lado, la figura 2.5 muestra un paquete LoRa en capa física.

Figura 2.5: Paquete LoRa (capa física) [elaboración propia]

El tiempo de duración de este paquete está dado por la ecuación 2.2. $T_{preamble}$ es el tiempo que toma transmitir los $n_{preamble}$ símbolos del preámbulo. $T_{payload}$ es el tiempo que toma transmitir los símbolos de los campos **header, CRC, Payload y Payload CRC** mostrados en la figura 2.5.

$$T_{packet} = T_{preamble} + T_{payload} \tag{2.2}$$

Tiempo de preámbulo $T_{preamble}$ o tiempo de preámbulo, está dado por el tiempo que toma transmitir los $n_{preamble}$ símbolos. La ecuación 2.3 muestra el tiempo para transmitir estos símbolos:

$$T_{preamble} = (n_{preamble} + 4,25) \cdot T_{sym}, \tag{2.3}$$

donde $n_{preamble}$ es el número de símbolos del preámbulo y T_{sym} es calculado de acuerdo a la ecuación 2.1.

Tiempo de payload $T_{payload}$ o tiempo de carga útil, está dado por el tiempo que toma transmitir los $n_{payload}$ símbolos del paquete sin incluir el preámbulo. La ecuación 2.4 muestra el tiempo para transmitir estos símbolos:

$$T_{payload} = T_{sym}\left(8 + max\left(ceil\left(\frac{8PL - 4SF + 28 + 16CRC - 20H}{4\,(SF - 2DE)}\right) \cdot (CR + 4)\,;0\right)\right) \tag{2.4}$$

donde:

- PL: Payload en bytes

- SF: Spreading Factor con valores de 7 a 12
- CRC: Cyclic Redundancy Check mode. habilitado es $CRC = 1$, deshabilitado es $CRC = 0$ (por defecto para LoRaWAN $CRC = 1$)
- H: Header implícito $H = 1$ o Header explícito $H = 0$ (por defecto para LoRaWAN $H = 0$)
- DE: Adaptive Data Rate. habilitado es $DE = 1$, deshabilitado es $DE = 0$
- CR: Coding Rate. $CR = 1, 2, 3, 4$. (por defecto para LoRaWAN $CR = 1$)

2.2. Estado del Arte

2.2.1. Internet Protocol (IP) sobre LPWAN

Como se indicó en la sección 1.3, la hipótesis de esta tesis está asociada a la relación entre la eficiencia del protocolo SCHC y los distintos *spreading factor* del protocolo LoRa. Debido a lo nuevo del protocolo *SCHC*, esta relación no tiene estudios asociados directamente a estas variables. Por lo tanto, este capítulo se centra en revisar el estado del arte de las alternativas que existen a la fecha para transportar IPv6 en redes restringidas o *constrained networks*.

En la sección 1.1, se indicaron las ventajas de utilizar direccionamiento IP en dispositivos *IoT*. (Ayoub et al.) [14], indican que el crecimiento exponencial del numero de nodos conectados a Internet para finales del 2020, no permitirá que la tecnología IP usada sea la versión 4 debido a la cantidad de direcciones disponibles. Por lo tanto, las propuestas técnicas para soportar IP en redes *LPWA* necesariamente deben ser sobre tecnología IPv6.

(Weber et al.) [15] presentan una solución para transportar IPv6 sobre LoRaWAN basada en el estándar 6LoWPAN. La figura 2.6 muestra su propuesta llamada 6LoRaWAN la cual está basada en una capa de adaptación entre la capa de enlace de datos y la capa de red del modelo OSI/ISO.

	IEEE 802.15.4	**6LoWPAN**	**LoRaWAN™**	**6LoRaWAN**
Application layer	e.g. ZigBee	e.g. COAP	custom	e.g. COAP
Presentation layer				
Session layer				
Transport layer		e.g. UDP		e.g. UDP
Network layer	e.g. ZigBee	IPv6		IPv6
		6LoWPAN adaption		6LoRaWAN adaption
Data link layer	802.15.4 MAC	802.15.4 MAC	LoRaMAC	LoRaMAC
Physical layer	802.15.4 PHY	802.15.4 PHY	LoRaPHY	LoRaPHY

Figura 2.6: IPv6 sobre LoRaWAN (Weber, 2016) [15]

Existen dos problemas al querer transportar IPv6 sobre LoRaWAN los cuales 6LoWPAN ya los resuelve. Primero está el problema de tener un paquete IPv6 con un tamaño mayor a la MTU de LoRaWAN, en este caso 6LoWPAN lo resuelve aplicando compresión de headers. El segundo problema se relaciona con la necesidad de fragmentar un paquete, el cual es resuelto utilizando un protocolo de fragmentación incluido en 6LoWPAN.

Desgraciadamente el uso del protocolo LoRaWAN más la capa de adaptación 6LoWPAN

tiene un excesivo overhead. Tal como indica el autor, cada mensaje puede alcanzar los 60 bytes solamente de encabezados del protocolo 6LoWPAN. Considerando que LoRaWAN para el caso de la banda AU915 puede llegar a tener una MTU de 50 bytes utilizando los *spreading factors* 12, 11 y 10, esta solución se limita entonces solo al uso de los *spreading factors* 9, 8 y 7. (Thielemans et al.) [16], presentan un *test-bed* que utiliza el estándar SCHC para la transmisión de paquetes IPv6 sobre LoRaWAN en nodos instalados en vehículos. La solución considera solo la etapa de compresión de SCHC ya que utiliza tamaños de paquetes IPv6 menor a la MTU de LoRaWAN *spreading factor* 10 asegurando de esta forma la no utilización de la etapa de fragmentación. Respecto a los resultados, cabe destacar que están basados en el *Round Trip Time o RTT* y el *Packet Delivery Ratio o PDR* al transmitir mensajes ICMPv6 desde una estación base hasta la OBU instalada en el vehículo, la cual soporta LoRaWAN + SCHC. Estos resultados indican que el RTT es sensible a la velocidad del vehículo, a pesar que el fabricante del módulo LoRa (Semtech) destaca que la modulación es inmune al efecto Doppler. Desgraciadamente este trabajo de investigación no aporta a responder la hipótesis planteada en esta tesis debido a varias razones. Primero, no se utiliza la etapa de fragmentación, la cual es fundamental en este documento. Segundo, los resultados están asociados solamente a un *spreading factor*. De esta forma no se puede realizar ninguna comparación entre distintos tamaños de MTU.

Para poder comparar las ventajas y desventajas del estándar 6LoWPAN sobre el nuevo protocolo SCHC (Gomez et al.) [17] realizan un análisis basado en las capacidades de compresión de 6LoWPAN y como el overhead afecta a las tecnologías LPWAN como LoRaWAN, Sigfox y NB-IoT. Respecto a la compresión, los protocolos como *Robust Header Compression (ROHC)* necesitan una sincronización de contextos que no se pueden efectuar en redes LPWA. 6LoWPAN considera un mecanismo mejorado respecto a ROHC, pero este mecanismo no considera la compresión de protocolos de capas superiores como CoAP. Por otro lado, 6LoWPAN sigue teniendo un overhead grande para tecnologías IoT que manejan payload del tamaño del orden de los 10 bytes. Respecto a la fragmentación, 6LoWPAN considera un mecanismo de fragmentación que posee dos características que no son óptimas para redes LPWA. Primero, 6LoWPAN no tiene la posibilidad de retransmitir un solo fragmento cuando hay una pérdidas en una trama. Esto conlleva a sobre-utilizar los recursos de las redes LPWA para retransmisiones que podrían ser manejadas a nivel de fragmento perdido y no a nivel de paquete perdido. Otro punto que juega en contra de 6LoWPAN para redes LPWA es el overhead en los fragmentos usados para fragmentación. 6LoWPAN utiliza headers del orden de los 5 bytes. Esto es ineficiente para tecnologías que tienen un payload del orden de los 10 bytes.

(Ayoub et al.) [14] comparan los protocolos *LoRaWAN^{TM}, DASH7 y NB-IoT* en función de su habilidad para soportar el protocolo IP a través de diferentes soluciones estándar para operar IP sobre tecnologías de *IoT* como: *6LoWPAN y 6Lo, 6TiSCH, RObust Header Compression (ROHC), LOWPAN IPHC y LOWPAN NHC, Routing Protocol for Low-power and lossy networks (RPL), CoAP y SCHC*. A pesar de no hacer un análisis riguroso ya que no se presentan resultados, el trabajo llega a la conclusión de que el protocolo *CoAP* y el esquema *SCHC* proporcionan el mejor enfoque que se puede adoptar como solución base para las tecnologías *LPWAN*.

(Abdelfadeel et al.) [18] presentan una implementación y evaluación de SCHC a través

de una comparación con *Internet Protocol Header Compresion o IPHC*, que también apunta a redes restringidas. Además, proponen una mejora de SCHC llamada *Layered SCHC (LSCHC). LSCHC* es un contexto en capas que reduce el consumo de memoria y la complejidad del procesamiento, y agrega flexibilidad al comprimir paquetes. El resultado de esta comparación indica que LSCHC logra factores de compresión más altos en comparación con IPHC e ilustran el efecto de esto en el tiempo de transmisión y la confiabilidad sobre un enlace LoRa. LSCHC es muy efectivo si los flujos potenciales dentro de una red se conocen de antemano, sin embargo, funciona mal con flujos desconocidos de antemano.

Basándose en el uso de SCHC sobre tecnología LPWAN, en [19] se propone una técnica llamada *asignación ficticia para compresión / descompresión* o *dummy mapping* basada en un mejoramiento de la subcapa de compresión del protocolo SCHC. Esta propuesta permite mejorar la compresión de los headers en flujos que no se conocen de antemano. La asignación ficticia crea una lista de tamaño fijo de valores ficticios que se comparten entre el *Network Server* y el dispositivo *LPWAN*. El *Network Server* asigna los headers que se comprimirán a valores ficticios, enviando al dispositivo solo el índice de las asignaciones. El dispositivo luego reconstruye los valores del encabezado comprimido utilizando los valores ficticios, en lugar de los valores reales del encabezado. En respuesta, el *Network Server* vuelve a establecer los valores comprimidos en sus valores reales y completa correctamente el intercambio de mensajes. Esta técnica proporciona un comportamiento similar a las que utilizan un contexto dinámico, sin embargo, sin la compleja resincronización que requiere el contexto dinámico y que no es adecuado para las redes *LPWA*.

A nivel de movilidad IPv6 sobre redes vehiculares (Sanchez-Iborra et al.) [20] presentan un enfoque basado en *multi-access edge computing o MEC* para incrustar datagramas IPv6 dentro de redes LoRa utilizando SCHC. Ellos realizan un procedimiento de adaptación en el dispositivo final o *end-device*, mientras que su contraparte se ha desarrollado para un nodo MEC conectado a una puerta de enlace LoRa. El nodo MEC se encarga entonces de interconectar los dispositivos finales LoRa con la red IPv6 normal, por lo tanto, proporciona interconectividad con Internet. El banco de pruebas considera entornos reales de vehículos. Han obtenido pruebas de retardo de ida y vuelta de aproximadamente 900 ms, demostrando que el impacto de la velocidad en el rendimiento del canal LoRa es evidente. Esto lo consiguen observando una correlación entre la movilidad y la relación de entrega de paquetes o *packet delivery ratio*.

Maturana[21] presenta una plataforma experimental para la implementación y evaluación del estándar SCHC sobre una red LoRaWAN consistente en un nodo terminal Microchip y un *Radio Gateway (RG)* de Everynet. Se logra la transmisión exitosa de paquetes IPv6 sobre LoRaWAN utilizando SCHC como mecanismo de compresión. Se evalúan tres contextos SCHC que permiten una compresión de 66,25 %, 58,75 % y 7,5 %. A diferencia de esta tesis, su trabajo considera solo la etapa de compresión. Además, la metodología utlizada es solo empírica.

En resumen, los estudios revisados apuntan a que 6LoWPAN no es la solución indicada para la compresión y fragmentación en redes *LPWA*. La razón: el excesivo overhead tanto en el proceso de compresión como en el de fragmentación. Por otro lado, 6LoWPAN no tiene soporte de compresión para el protocolo CoAP el cual apunta a ser un protocolo de capa

de aplicación eficiente y adaptado para LPWAN. Ademas, 6LoWPAN no fue creado para comprimir protocolos de aplicación, sino para adaptar IPv6 a distintas capas inferiores cuyas capacidades no lo permitían de manera nativa, en especial por su MTU.

El estandar *SCHC* [1] permite comprimir los encabezados de los protocolos *IPv6, UDP y CoAP* para redes *LPWAN*, permitiendo comunicarse con un *end device* directamente mediante direcciones *IP*. El estándar también permite la fragmentación de los mensajes transportados sobre protocolos *LPWAN*.

El grupo *LPWAN* del *IETF*, a través de sus reuniones técnicas, ha realizado continuas *hackatones* donde se promueve el desarrollo de código abierto para *SCHC*. *OpenSCHC* es una implementación *OpenSource* de *SCHC* basada en *Python* y *micropython* que implementa los avances del Grupo de trabajo de *LPWAN* en el *IETF* [22].

Este trabajo de tesis ha reimplementado el estándar a través de un nuevo código en Python llamado PySCHC. La razón, al momento de comenzar este trabajo, OpenSCHC aun estaba en fase inicial respecto a la tecnología $LoRaWAN^{TM}$.

Capítulo 3

Static Context Header Compression (SCHC)

La propuesta del estándar SCHC entrega funcionalidades de compresión y fragmentación a tecnologías de la familia Low Power Wide Area Networks (LPWANs) [1]: Sigfox, LoRaWAN™, NB-IoT.
Este capítulo está dentro del marco teórico general de este trabajo y pretende explicar el funcionamiento del estándar SCHC.

3.1. Introducción

SCHC es considerado como una capa de adaptación entre un protocolo de capa de red como es IPv6 y una capa de enlace de datos como las tecnologías de tipo LPWAN. Esta capa de adaptación está compuesta por dos subcapas denominadas Compresión y Fragmentación, como se observa en la figura 3.1.

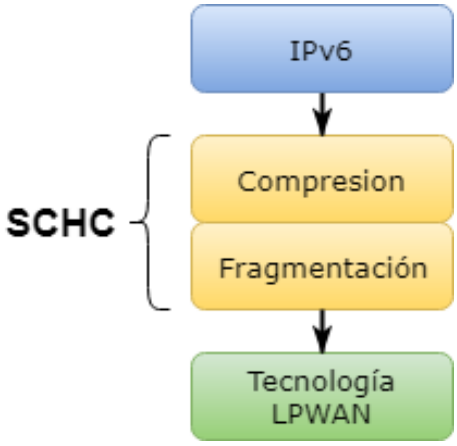


Figura 3.1: Subcapas SCHC

Antes que un paquete IPv6 sea trasmitido, este es procesado por la subcapa de compresión.

El resultado de este proceso es un *paquete SCHC* o *SCHC packet*. Si el *SCHC packet* es menor o igual que el MTU del protocolo LPWAN, este es transmitido sin fragmentación, en caso contrario, el *SCHC packet* es entregado a la subcapa de fragmentación como se verá en la sección 3.3. La figura 3.2 ilustra el proceso de transmisión de un paquete IPv6 al pasar por las subcapas de SCHC.

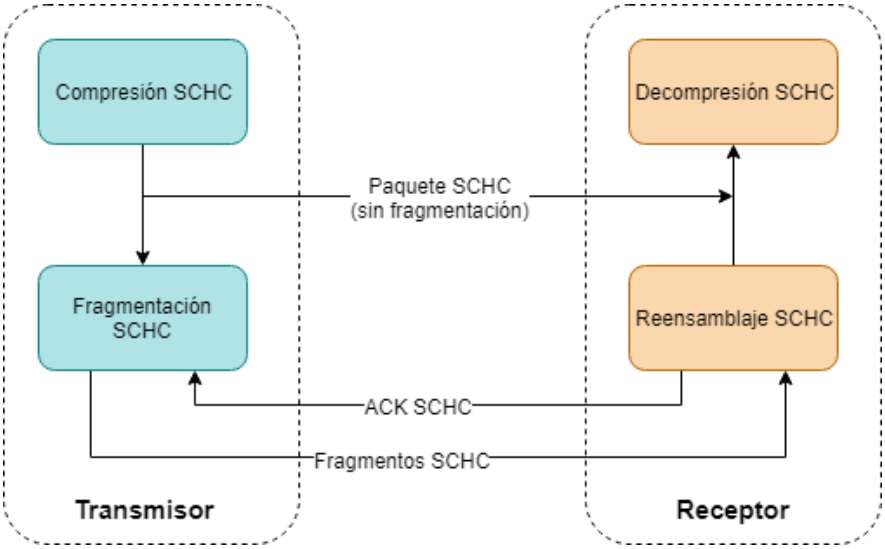


Figura 3.2: Operacion de SCHC

3.2. Compresión

Esta subcapa basa su funcionamiento en la creación de un *contexto* en el cual se almacenan *reglas*. Cada regla tiene una serie de *descriptores* o *Field Descriptions* que definen cómo se debe comprimir o descomprimir un header específico [19].

Cada descriptor está compuesto por 7 campos que se describen a continuación:

- **Field ID (FID)** define el protocolo y el *header* (ejemplo *UDP - Destination Port*) que será evaluado por el descriptor.
- **Field Length (FL)** representa el largo en bits del header.
- **Field Position (FP)** debido a que un mismo header puede estar más de una vez en un paquete, este campo define cúal header se evalúa (primera, segunda, tercera o cuarta ocurrencia). Por defecto es 1 (primera ocurrencia).
- **Direction Indicator (DI)** indica la dirección del paquete. Si este campo indica *Up*, el descriptor solo es aplicado al header cuando el paquete va en sentido uplink. Si este campo indica *Dw*, el descriptor solo es aplicado al header cuando el paquete va en sentido downlink. Si este campo indica *Bi*, el descriptor se aplica al header sin importar el sentido del paquete.
- **Target Value (TV)** es el valor con el que se compara el valor del header que viene en el paquete.

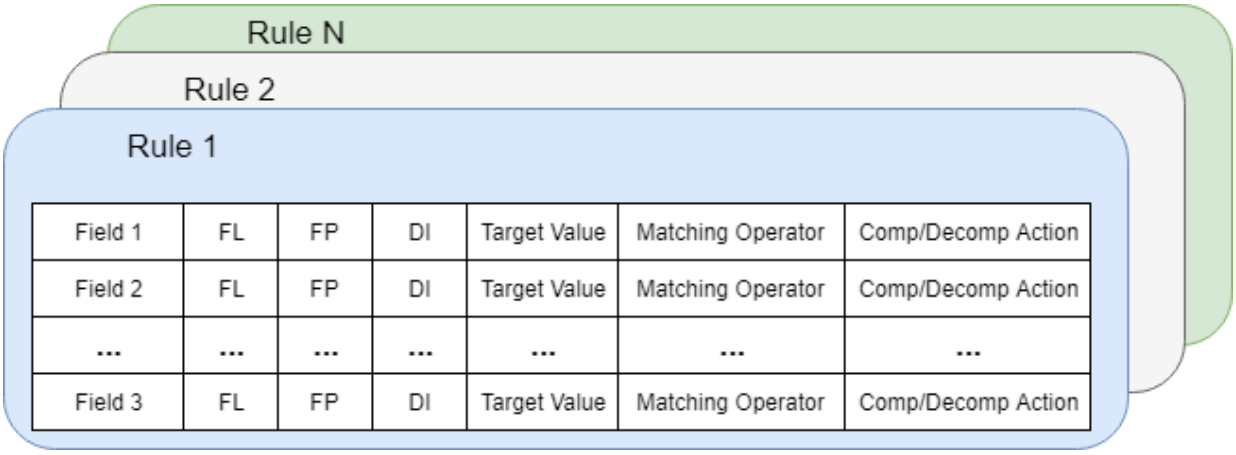


Figura 3.3: SCHC Context [elaboración propia]

- **Matching Operator (MO)** es el operador utilizado para hacer coincidir el valor del header y el Target Value (TV). MO solo se usa durante la fase de compresión.
- **Compression/Decompression Action (CDA)** describe el proceso de compresión o descompresión que debe ser efectuado después que todos los MO han sido evaluados.

La figura 3.3 muestra los conceptos de Contexto, Regla y Descriptor, junto con los parámetros de cada Field Descriptor.

Cuando se desea enviar un paquete (IPv6 + UDP) se compara cada header del paquete contra cada regla del contexto. La tabla 3.1 muestra una regla de compresión para IPv6 y UDP.

FID	**FL**	**FP**	**DI**	**TV**	**MO**	**CDA**
IPv6 Version	4	1	Bi	6	ignore	not-sent
IPv6 DiffServ	8	1	Bi	0	equal	not-sent
IPv6 Flow Label	20	1	Bi	0	equal	not-sent
IPv6 Length	16	1	Bi		ignore	compute-*
IPv6 Next Header	8	1	Bi	17	equal	not-sent
IPv6 Hop Limit	8	1	Bi	255	ignore	not-sent
IPv6 DevPrefix	64	1	Bi	FE80::/64	equal	not-sent
IPv6 DevIID	64	1	Bi		ignore	DevIID
IPv6 AppPrefix	64	1	Bi	FE80::/64	equal	not-sent
IPv6 AppIID	64	1	Bi	::1	equal	not-sent
UDP DevPort	16	1	Bi	123	equal	not-sent
UDP AppPort	16	1	Bi	124	equal	not-sent
UDP Length	16	1	Bi		ignore	compute-*
UDP checksum	16	1	Bi		ignore	compute-*

Tabla 3.1: Ejemplo de Regla de compresión [elaboración propia]

Si al comparar cada header con cada descriptor dentro de una regla todos los **MO** resultan **Verdaderos**, esa regla es escogida como regla de compresión y se aplican los **CDA** a cada header. Si la comparación al aplicar algún **MO** resultase **Falso**, la regla es descartada y se continúa con la evaluación de la siguiente regla. El proceso finaliza cuando se escoge una regla y se ejecutan todos los **CDA**. Como resultado de este proceso se crea un paquete SCHC como el de la figura 3.4.

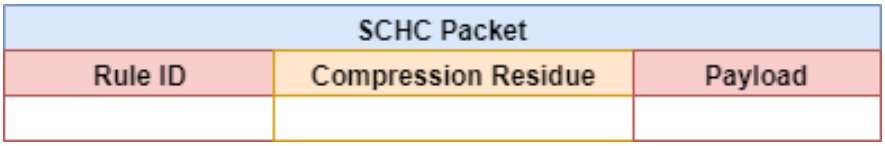


Figura 3.4: Estructura de un paquete SCHC [elaboración propia]

La *RuleID* es el identificador de la regla seleccionada, que debe ser el mismo ID que tiene el proceso de descompresión. El *compression residue* es la concatenación de todos los residuos no vacíos después de aplicar la compresión. El *payload* corresponde a la carga útil del protocolo superior. Cuando se comprime IPv6 + UDP, el payload es la carga útil de UDP. La figura 3.5 muestra el proceso de compresión paso a paso.

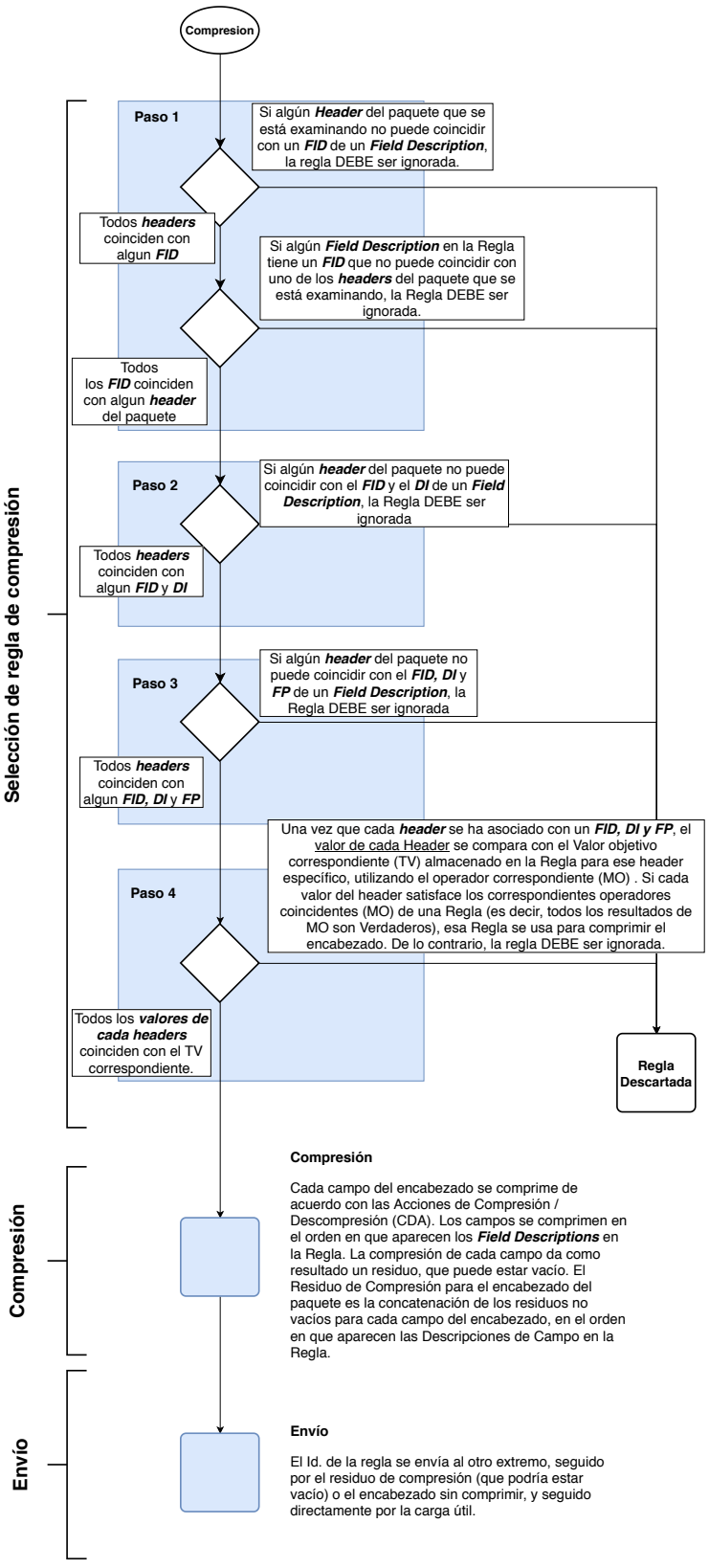


Figura 3.5: Proceso de compresión SCHC [elaboración propia]

3.3. Fragmentación

En las tecnologías LPWAN, la Unidad Máxima de Transmisión en capa de enlace o MTU es del orden de los 100 bytes. Este tamaño sumado a la falta de un mecanismo de fragmentación impide el transporte de protocolos basados en tecnología de Internet. Para cambiar esto, el estándar SCHC ha incluido el mecanismo de fragmentación permitiendo el soporte a IPv6 que requiere al menos 1280 bytes. Para ello, se incluyen varios modos de funcionamiento que son explicados en profundidad en la sección 3.3.2.

3.3.1. Aspectos Generales

Para poder soportar el mecanismo de fragmentación el estándar define algunos elementos comunes como son: *tiles*, ventanas, mensajes y campos. Un *SCHC packet* o paquete SCHC es dividido en partes llamadas *tiles*. Los grupos de *tiles* son llamados ventanas. La figura 3.6 presenta este concepto.

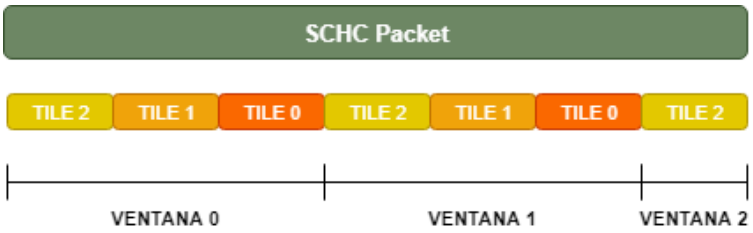


Figura 3.6: División de un *SCHC packet* en *tiles* y ventanas [elaboración propia]

Para poder transmitir los *tiles* desde el transmisor al receptor, SCHC utiliza distintos mensajes llamados *fragmentos SCHC*. Cada uno de estos mensajes tiene una función específica:

- **SCHC Fragment**: mensaje enviado desde el transmisor al receptor. Transporta partes de un paquete SCHC llamadas *tiles*.
- **SCHC ACK**: mensaje enviado desde el receptor al transmisor. Permite confirmar o no la recepción de uno o varios *tiles*.
- **SCHC ACK REQ**: mensaje enviado desde el transmisor al receptor solicitando la confirmación de uno o más *tiles* a través de un SCHC ACK.
- **SCHC Sender-Abort**: mensaje enviado desde el transmisor al receptor indicando que el transmisor ha cancelado el envío de fragmentos.
- **SCHC Receiver-Abort**: mensaje enviado desde el receptor al transmisor indicando que cancele el envío de fragmentos.

Cualquiera de los mensajes listados anteriormente puede contener los siguientes campos o *headers*.

- **Rule ID**: Este campo está presente en todos los mensajes y es usado para identificar si un mensaje es o no un fragmento SCHC.

- **Datagram Tag (DTag)**: este campo permite identificar unívocamente a dos fragmentos que pertenecen a dos *SCHC packet* distintos.
- **W**: Cuando los fragmentos SCHC enviados se agrupan en ventanas, este campo transporta el numero de la ventana al que pertenece el fragmento.
- **Fragment Compressed Number (FCN)**: campo que indica el numero de secuencia de los *tiles*.
- **Reassembly Check Sequence (RCS)**: este campo solo aparece en los mensajes *All-1 SCHC fragment*. Transporta el código de detección de errores basado en el polinomio CRC32.
- **C (integrity Check)**: Este campo tiene el tamaño de 1 bit y se utiliza en el mensaje *SCHC ACK* para informar sobre la verificación de integridad del paquete *SCHC* reensamblado. Un valor de 1 indica que la verificación de integridad se realizó y es exitosa. Un valor de 0 indica que no se realizó la verificación de integridad o que fue un error.
- **Compressed Bitmap**: Este campo aparece en el mensaje *SCHC ACK* para informar sobre el mapa de bits del receptor. El mapa de bits indica que *tiles* se recibieron de forma correcta y cuales no.

3.3.2. Modos de Fragmentación

El estándar SCHC define 3 modos de operación para la transmisión de fragmentos. Cada uno de estos modos tiene características que los hacen útiles para distintos escenarios y tecnologías LPWAN. Para el caso de LoRaWAN se utilizan solo 2 de los 3 modos: Ack-on-Error y Ack-Always. En esta tesis se utiliza solamente el modo ACK-on-Error debido a que los estudios indican que

la mayoría de las aplicaciones es el *end-device* el que envía datos desde campo hacia el *network server* en el *cloud*.

ACK-on-Error

El estándar SCHC para LoRaWAN utiliza el modo de fragmentación *ACK-on-Error* para el *uplink*, esto es, los datos enviados desde el *end-device* y recibidos por el *Gateway*. Este modo de fragmentación soporta una MTU variable y una entrega fuera de orden de los fragmentos. Además, utiliza el concepto de ventanas y *tiles* definidos en la sección 3.3.1 y mostrado gráficamente en la figura 3.6. Todos los *tiles* deben ser del mismo tamaño con excepción del último.

El mensaje utilizado para transportar los fragmentos se llama *Regular SCHC fragment*. Un *Regular SCHC fragment* puede transportar uno o más *tiles*. Puede darse el caso que un *Regular SCHC fragment* transporte varios *tiles* que se dividen en varias ventanas. O sea, si la *MTU* es lo suficientemente grande existe la posibilidad que un grupo de *tiles* pertenezcan a la ventana *i*, otro grupo de *tiles* a la ventana *i+1* y otro a la ventana *i+2*, todos ellos transportados en el mismo *Regular SCHC fragment*.

Cuando uno de los fragmentos se pierde, el receptor debe enviar un mensaje *SCHC ACK* al emisor indicando qué *tiles* llegaron correctamente y cuáles no. El mensaje *SCHC ACK* solo reporta la recepción de los *tiles* pertenecientes a una ventana. El estándar definido en la RFC8724 [1] indica que no se debe enviar un mensaje *SCHC ACK* cuando todos los *tiles* se recibieron exitosamente. En cambio, el perfil de SCHC para LoRaWAN [5] indica que sí se debe enviar un *SCHC ACK* al final de la sesión de fragmentación en la ventana de recepción RX1 o RX2 del *end-device*. Si esto no se cumple, el *end-device* debe enviar un mensaje *SCHC ACK Request* para que el mensaje *SCHC ACK* sea retransmitido por el receptor. La figura 3.7 muestra el flujo de mensajes para el modo de fragmentación ACK-on-Error con un *SCHC packet* que necesita ser divido en dos ventanas de transmisión. Los mensajes indicados en un cuadro rojo son los mensajes que se pierden. El valor $c = 0$ en el mensaje *SCHC ACK* enviado por el receptor al transmisor, indica que no se realizó la verificación de integridad o que no se recibió un *tile*. $c = 1$ indica que la verificación de integridad de la ventana se realizó y es exitosa. *End of window* indica el momento que la ventana ha finalizado, en este caso cuando se recibe el SCHC ACK correspondiente. Cabe destacar que la primera ventana es de transmisión y las siguientes son de retransmisión de los *tiles* perdidos.

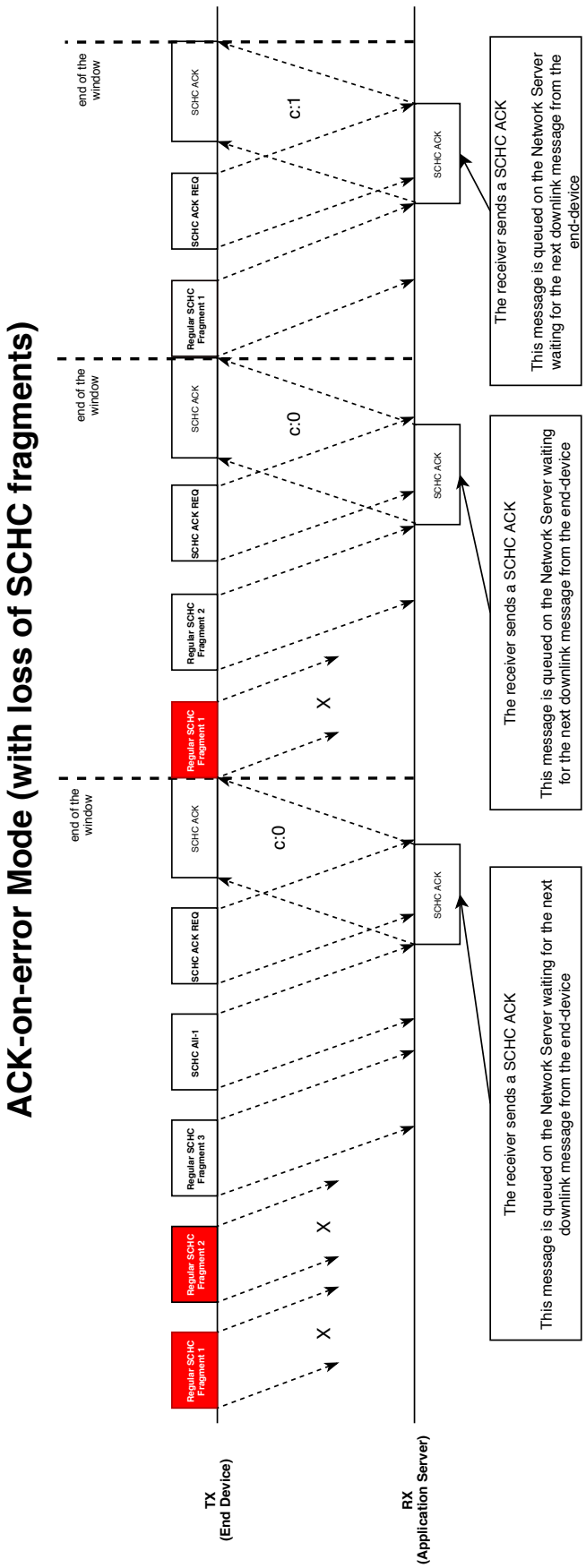


Figura 3.7: Modo Ack-on-Error [elaboración propia]

Capítulo 4

Modelado teórico de la compresión y fragmentación para LoRaWAN

Para poder responder la pregunta planteada en la hipótesis respecto a la relación entre la eficiencia *SCHC* y el *spreading factor* de *LoRa*, las siguientes secciones presentan un modelo que permite relacionar estas variables desde un punto de vista teórico. Así, este capítulo corresponde a un aporte del trabajo de tesis basándose en el actual estado del arte mostrado en la sección 2.2.

4.1. Compresión

Como se indicó en el capítulo 3, el estándar SCHC permite comprimir encabezados IPv6 y UDP para redes LPWA, habilitando el uso de comunicaciones basadas en IP directamente en el dispositivo final. La propuesta de SCHC opera sobre la base de contextos estáticos. Esto significa que los valores en los encabezados de los paquetes IPv6 y UDP que se comprimen, son conocidos o se pueden deducir de la información de otros encabezados.

Dentro de un contexto SCHC, se pueden encontrar diferentes reglas que indican el comportamiento del compresor/descompresor para cada encabezado IPv6 y UDP. La selección de estas reglas afecta directamente el número de bytes que componen el paquete comprimido, y por lo tanto, el tiempo que lleva transmitir cada paquete comprimido en el medio inalámbrico. El tiempo de transmisión se ha denominado típicamente Tiempo en el aire o *Time on Air (ToA)* [12, 23]. Este capítulo propone una metodología para comprender el comportamiento del *ToA* y caracterizar su desempeño bajo diferentes condiciones de operación, es decir, *spreading factors* variables, en el enlace LoRaWAN.

4.1.1. Caracterización de porcentajes de compresión y tiempos de transmisión

La caracterización de los porcentajes de compresión en SCHC permite responder a preguntas tales como: ¿Qué pasa con el comportamiento del *ToA* al aumentar o disminuir la relación de compresión de paquetes *SCHC*? ¿varía linealmente con el porcentaje de compresión dados los factores de propagación de $LoRa^{TM}$?. Estas preguntas ayudan a los diseñadores u operadores de red a realizar una selección informada de una regla de compresión dado que se conoce el comportamiento del *ToA*. Por ejemplo, poder seleccionar un conjunto de reglas sabiendo que el *ToA* permanece constante sin importar la regla utilizada dentro del conjunto. Existen aplicaciones IoT que pueden contener máquinas de estados basada en temporizadores, por lo que un aumento del *ToA* puede activar estados diseñados para lidiar con una pérdida de paquetes, cuando en realidad solo ha ocurrido un retraso en la transmisión. En el caso de que la regla escogida sea fija, es necesario identificar cómo la naturaleza cambiante del Spreading Factor de LoRaWAN conlleva una variación en el ToA de los paquetes comprimidos con dicha regla.

Para la evaluación de la compresión, además del ToA, en esta tesis se introducen dos nuevas métricas. Una de ellas permite determinar las reglas de compresión para las cuales la variación en el Spreading Factor conlleva una mínima variación del ToA. Esto sería beneficioso para aplicaciones que son sensibles a la variación del retardo, permitiendo identificar aquellas modificaciones en la compresión que no involucren mayores cambios en el tiempo de transmisión. La otra métrica permite conocer cual será la máxima variación del ToA, para un Spreading Factor dado, cuando se modifica el porcentaje de compresión. Esto es útil para aplicaciones que poseen múltiples flujos de datos, ya que cada flujo estará asociado a una regla diferente y por ende a un porcentaje de compresión diferente. Así, a pesar de que se mantenga el Spreading Factor, estos flujos pueden presentar distintos retardos.

Derivando los porcentajes de compresión de SCHC

Cada regla dentro de un contexto SCHC se basa en un número finito de *Field Descriptions*. En total, hay 12 descriptores para un paquete IPv6/UDP: 8 descriptores para encabezados IPv6 (suponiendo un paquete básico IPv6) y 4 descriptores más para UDP. Tomando los 12 descriptores y combinando todas las posibilidades de compresión, se puede construir una tabla con 4096 reglas posibles (2^{12} reglas). La figura 4.1 ilustra las posibles combinaciones, donde 0 indica que el encabezado correspondiente no está comprimido (es decir, CDA igual a *value-sent*) y 1 indica que el encabezado debe estar comprimido (es decir, CDA igual a *not-sent*).

Considerando los campos de encabezado de un paquete IPv6, se tiene que::

- **IPv6.version** el valor de este header siempre es 6 para IPv6, por lo que el receptor puede deducirlo al saber que se esta trabajando en una red con este tipo de protocolo.
- **IPv6.payloadLength** siempre se puede calcular en el extremo decodificador, por lo tanto no se justifica su envío.

- **UDP.length** siempre se puede calcular en el extremo decodificador, por lo tanto no se justifica su envío.
- **UDP.checksum** siempre se puede calcular en el extremo decodificador, por lo tanto no se justifica su envío.
- **IPv6.nextHeader** en un ambiente IPv6 + UDP siempre es 0x11, por lo que se puede omitir el envío de su valor.
- En sentido uplink, el parámetro **IPv6.hopLimit** puede ser considerado 0xff ya que entre SCHC C/D en el device y el SCHC C/D en el Network Side, no hay elementos de red que puedan modificar su valor.
- En sentido uplink, si el end device solo tiene asignada una dirección de red, el header **IPv6.sourceAddress** no se modifica en cada paquete enviado. Se puede considerar que es un parámetro estático.

Considerando lo anterior, el número de encabezados disminuye de 12 a 6, lo que da como resultado 64 posibles reglas (2^6 reglas). Los porcentajes de compresión para cada regla se pueden obtener evaluando cada regla con los tamaños de encabezados correspondientes. Cabe señalar que, incluso si se logra una compresión de todos los encabezados, el 100 % de compresión no es factible ya que, de acuerdo con [1], al menos se debe enviar el ID de la regla.

El tamaño final del paquete *SCHC* es por lo tanto, la suma de los bits en el Residuo de compresión: $ComRes$, los bits en el RuleID: $Rule$ y la carga útil UDP: $UDP_{payload}$, como se indica en la ecuación 4.1.

$$SCHC_{length} = RuleID + ComRes + UDP_{payload}. \tag{4.1}$$

RuleName	RuleID	IPv6.version	IPv6.trafficClass	IPv6.flowLabel	IPv6.payloadLength	IPv6.nextHeader	IPv6.hopLimit	IPv6.sourceAddress	IPv6.destinationAddress	UDP.sourcePort	UDP.destinationPort	UDP.length	UDP.checksum	
Rule 0	0	0	0	0	0	0	0	0	0	0	0	0	0	4096 rules
Rule 1	1	0	0	0	0	0	0	0	0	0	0	0	1	
Rule 2	2	0	0	0	0	0	0	0	0	0	0	1	0	
Rule 3	3	0	0	0	0	0	0	0	0	0	0	1	1	
:	:	:	:	:	:	:	:	:	:	:	:	:	:	
Rule N	N	1	1	1	1	1	1	1	1	1	1	1	1	

Figura 4.1: Agrupación de reglas según encabezado comprimido [elaboración propia]

Métricas de rendimiento propuestas

Time on Air (ToA) : se define como el tiempo que toma el transmisor en insertar un paquete en el medio inalámbrico [6]. Se calcula considerando la longitud del paquete en bits y la duración del bit de acuerdo con el *spreading factor* empleado en el momento de la transmisión. El *ToA* se calcula de la siguiente manera [23]:

$$ToA\,(ms) = T_{preamble}\,(ms) + T_{payload}\,(ms)\,, \tag{4.2}$$

donde:

$$T_{preamble}\,(ms) = (n_{preamble} + 4{,}25) \cdot T_s\,(ms)\,, \tag{4.3}$$

$$T_{payload}\,(ms) = n_{payload} \cdot T_s\,(ms)\,, \tag{4.4}$$

$$T_s\,(ms) = \frac{2^{SF}}{BW} \cdot 1000, \tag{4.5}$$

así, para LoRaWAN™[13], $n_{preamble} = 8$, SF va desde 7 a 12 y el ancho de banda: BW, puede ser 125 KHz, 250 KHz o 500 KHz.

En el escenario donde la compresión SCHC es usada, el $n_{payload}$ es calculado como sigue:

$$n_{payload} = 8 + max\left[C \cdot (CR + 4)\,; 0\right], \tag{4.6}$$

$$C = ceil\left(\frac{8PL - 4SF + 28 + 16CRC - 20H}{4\,(SF - 2DE)}\right), \tag{4.7}$$

$$PL = MHDR + FHDR + FPort + SCHC_{length} + MIC, \tag{4.8}$$

donde:

- PL: *Payload* de capa física. De acuerdo a la ecuación 4.8, el tamaño PL corresponde a la suma de *MAC header field* $MHDR$ (1 byte), el *Frame header field* $FHDR$ (7 a 22 bytes), el *Field port* $FPort$ (1 byte), el tamaño *MAC payload* corresponde al tamaño del SCHC packet $SCHC_{length}$, y *Message Integrity Code* MIC (4 bytes).
- SF: Spreading Factor con valores entre 7 y 12
- CRC: es 1 cuando el *cyclic redundancy check* está habilitado, 0 en otro caso (1 por defecto en LoRaWAN)
- H: es 1 para *Implicit Header*, 0 para *Explicit Header* (0 por defecto en LoRaWAN)
- DE: es 1 cuando el *Adaptive Data Rate* está habilitado, 0 en otro caso.
- CR: Coding Rate con valores $CR = \{1, 2, 3, 4\}$ (1 por defecto en LoRaWAN)

Time on Air variation (ΔToA) : se define como la resta entre el ToA calculado para el porcentaje más bajo de compresión (PC) y el ToA calculado para el PC más alto dado un *spreading factor* (SF). ΔToA se calcula de la siguiente manera:

$$\Delta ToA_{SF} = ToA_{PC_{min}} - ToA_{PC_{max}} \tag{4.9}$$

Time on Air range (ToA_{range}) : corresponde a la diferencia entre el ToA de los *spreading factors* SF más altos y más bajos del enlace LoRaWAN, es decir, 12 y 7, respectivamente, para un porcentaje dado de compresión (PC). ToA_{range} se calcula de la siguiente manera:

$$ToA_{range_{PC}} = ToA_{SF_{max}} - ToA_{SF_{min}} \tag{4.10}$$

4.2. Fragmentación

Para la subcapa de fragmentación, el estándar SCHC utiliza un diseño similar a los protocolos *Automatic Repeat Request o ARQ* para el control de errores y retransmisión. Existen 3 modos de funcionamiento en la subcapa de fragmentación: *No-ACK, ACK-Always y ACK-on-Error*. El modo que se utiliza en esta tesis es el definido en el estándar *SCHC* para el enlace de subida o *uplink* en LoRaWAN™, o sea, *ACK-on-Error*. El modo *ACK-on-Error* es definido en la sección 8.4.3 de la RFC8724 [1].

El modelo presentado en este capítulo, muestra los tiempos involucrados en la transmisión de un paquete SCHC mediante el proceso de fragmentación. Así, el resultado es una serie de tiempos que permiten determinar de antemano, valores como la eficiencia, calculada a partir de los tiempos de transmisión de un paquete SCHC y el tiempo de transmisión de un fragmento.

4.2.1. Modo Ack-on-Error sin pérdida de fragmentos

Tiempos y Flujo de mensajes

Para el caso de que no existan pérdidas de fragmentos, el modelo está basado en los siguientes supuestos:

- El paquete SCHC o *SCHC packet* es del tamaño suficiente para ser enviado en una ventana de fragmentación. Esto quiere decir, utilizando 63 tiles de 10 bytes cada uno, o sea, 630 bytes como máximo.
- El *spreading factor* de LoRa es el mismo para todos los *Regular SCHC fragments*.
- Debido a que cada *Regular SCHC Fragment* transporta *"n"tiles*, el último *Regular SCHC fragment* enviado podría no transportar *"n"tiles*. Por ejemplo: un *SCHC packet*

de 500 bytes es dividido en 50 tiles (tamaño de un tile es 10 bytes). Supóngase que para un *spreading factor* dado, cada *Regular SCHC fragment* puede transportar a lo más, 11 tiles. Los *Regular SCHC fragment* 1, 2, 3 y 4 habrán transportado 44 tiles en total. Dado que quedan 6 tiles pendientes de transmisión para completar los 50 tiles, el último *Regular SCHC fragment* SOLO transportará 5 tiles ya que el mensaje *All-1 SCHC Fragment* llevará el último tile. El modelo por lo tanto considera que el último *Regular SCHC fragment* puede ser de un tamaño menor o igual a los *Regular SCHC fragment* enviados previamente.

- Todos los mensajes en modo *downlink* son recibidos por el *end device* en la primera ventana de recepción LoRa, esto quiere decir, al finalizar el temporizador $RECEIVE\_DELAY\_1$. Se considera así, ya que el modo de operación más restrictivo es cuando el nodo funciona en Clase A.
- El procesamiento en el receptor de un *All-1 SCHC Fragment* toma más tiempo que la duración de espera máxima de las ventanas de recepción en el *end device* ($RECEIVE\_DELAY\_2 + RX2_{min}$). Es por esto que hay que esperar al siguiente mensaje en el enlace de subida para que se abra nuevamente la ventana y así recibir el mensaje *SCHC ACK* enviado en respuesta a un *All-1 SCHC Fragment* (ver figura 4.2).

Considerando el flujo de mensajes mostrado en la figura 4.2, es posible dividir la transmisión sin pérdidas de fragmentos en un solo bloque de mensajes.

Bloque 1 Este bloque está compuesto por todos los mensajes *Regular SCHC fragment* que componen la primera ventana de transmisión. La ventana finaliza al terminar de procesar el mensaje *SCHC ACK* en el lado del emisor. El mensaje *SCHC ACK* fue enviado por el receptor indicando qué *tiles* se recibieron y cuáles no en la última ventana. Este bloque tiene un tiempo de duración T_1 dado por la ecuación 4.11

$$T_1 = (m-1) \cdot (T_F + RD_2) + (T_{LF} + RD_2) + (T_{All-1} + RD_2) + (T_{AR} + RD_1) + TP_{ACK} \tag{4.11}$$

donde:

- T_F es el tiempo que lleva transmitir un mensaje *Regular SCHC fragment* en el medio inalámbrico. Su cálculo es explicado en la sección 4.2.1.
- m es la cantidad de *Regular SCHC fragment* que se deben enviar para completar los *tiles* del *SCHC packet* dentro de una ventana.
- $RD1$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RECEIVE\_DELAY\_1$ definido en [13].
- $RD2$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RX2_{min}$ definido en [13]. Corresponde a la suma entre el $RECEIVE\_DELAY\_2$ y $RX2_{min}$. $RECEIVE\_DELAY\_2$ es igual a 2 segundos y $RX2_{min}$ es la longitud de una ventana de recepción y debe ser al menos el tiempo requerido por el receptor para detectar efectivamente un preámbulo de enlace descendente. La ventana de recepción RX2 utiliza el *data rate 8 (DR8)* que equivale a un *spreading*

Figura 4.2: Flujo de mensajes en el modo Ack-on-Error (sin pérdidas de fragmentos) [elaboración propia]

factor 12 con 500 KHz de ancho de banda. Con estas condiciones el preambulo tiene un largo de 100,35 *ms*

- T_{LF} es el tiempo en transmitir el último *Regular SCHC fragment* dentro de la primera ventana. Recordar que este *Regular SCHC fragment* puede ser de un tamaño menor a los *Regular SCHC fragment* enviados antes, pero dentro de la misma ventana. Su cálculo es explicado en la sección 4.2.1.
- T_{All-1} es el tiempo para enviar un *All-1 SCHC Fragment*. Su cálculo es explicado en la sección 4.2.1.
- T_{AR} es el tiempo en transmitir un *SCHC ACK REQ*. Su cálculo es explicado en la sección 4.2.1.
- TP_{ACK} es el tiempo de transmisión del paquete *SCHC ACK*.

Cálculo del tiempo de frame

El tiempo de *frame* es el tiempo que toma el transmisor en enviar un *frame* en el medio inalámbrico. Su cálculo está basado en el *Time on Air* definido en la sección 2.1.2. El *Time on Air* basa su cálculo, entre otras cosas, en el tamaño del *payload* transportado en el mensaje *LoRaWAN*^{TM}. Por lo tanto, el tiempo de *frame* será distinto si se envía un *Regular SCHC fragment*, un *All-1 SCHC fragment* o un *SCHC ACK Request*.

Dado que el *payload* para un mensaje *LoRaWAN*^{TM} será un mensaje *SCHC*, el tamaño de cada mensaje *SCHC* está definido en la sección 5.6.2 del documento *Static Context Header Compression (SCHC) over LoRaWAN* [5].

4.2.2. Modo Ack-on-Error con pérdida de fragmentos

Tiempos y Flujo de mensajes

Para el caso de pérdidas de fragmentos, el modelo está basado en los siguientes supuestos:

- El paquete SCHC o *SCHC packet* es del tamaño suficiente para ser enviado en una ventana de fragmentación. Esto quiere decir, utilizando 63 tiles de 10 bytes cada uno, o sea, 630 bytes como máximo.
- El *spreading factor* de LoRa es el mismo para todos los *Regular SCHC fragments*.
- Debido a que cada *Regular SCHC Fragment* transporta $\boldsymbol{n}$ *tiles*, el último *Regular SCHC fragment* enviado podría no transportar $\boldsymbol{n}$ *tiles*. Por ejemplo: un *SCHC packet* de 500 bytes es dividido en 50 tiles (tamaño de un tile es 10 bytes). Supóngase que para un *spreading factor* dado, cada *Regular SCHC fragment* puede transportar a lo más, 11 tiles. Los *Regular SCHC fragment* 1, 2, 3 y 4 habrán transportado 44 tiles en total. Dado que quedan 6 tiles pendientes de transmisión para completar los 50 tiles, el último *Regular SCHC fragment* SOLO transportará 5 tiles ya que el mensaje *All-1 SCHC Fragment* llevará el último tile. El modelo por lo tanto considera que el último

Regular SCHC fragment puede ser de un tamaño menor o igual a los *Regular SCHC fragment* enviados previamente.

- Todos los mensajes en modo *downlink* son recibidos por el *end device* en la primera ventana de recepción LoRa, esto quiere decir, al finalizar el temporizador $RECEIVE\_DELAY\_1$. Se considera así, ya que el modo de operación más restrictivo es cuando el nodo funciona en Clase A.
- El procesamiento en el receptor de un *All-1 SCHC Fragment* toma más tiempo que la duración de espera máxima de las ventanas de recepción en el *end device* ($RECEIVE\_DELAY\_2 + RX2_{min}$). Es por esto que hay que esperar al siguiente mensaje en el enlace de subida para que se abra nuevamente la ventana y así recibir el mensaje *SCHC ACK* enviado en respuesta a un *All-1 SCHC Fragment* (ver figura 4.3).

Recordar que el mensaje *SCHC ACK* enviado por el receptor, solicita la retransmisión de los *tiles* que faltan y almacena en *buffer* los *tiles* que se han recibido correctamente esperando poder reconstruir el mensaje cuando se retransmitan los *tiles* perdidos.

Considerando el flujo de mensajes mostrado en la figura 4.3, es posible separar la transmisión con pérdidas de fragmentos en 3 bloques de mensajes.

Bloque 1 Este bloque está compuesto por todos los mensajes *Regular SCHC fragment* que componen la primera ventana de transmisión. La ventana finaliza al terminar de procesar el mensaje *SCHC ACK* en el lado del emisor. El mensaje *SCHC ACK* fue enviado por el receptor indicando qué *tiles* se recibieron y cuáles no en la última ventana. Este bloque tiene un tiempo de duración T_1 dado por la ecuación 4.12

$$T_1 = (m-1) \cdot (T_F + RD_2) + (T_{LF} + RD_2) + (T_{All-1} + RD_2) + (T_{AR} + RD_1) + TP_{ACK} \tag{4.12}$$

donde:

- T_F es el tiempo que lleva transmitir un mensaje *Regular SCHC fragment* en el medio inalámbrico. Su cálculo es explicado en la sección 4.2.1.
- m es la cantidad de *Regular SCHC fragment* que se deben enviar para completar los *tiles* del *SCHC packet* dentro de una ventana.
- $RD1$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RECEIVE\_DELAY\_1$ definido en [13].
- $RD2$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RX2$ definido en [13]. Corresponde a la suma entre el $RECEIVE\_DELAY\_2$ y $RX2$.
- T_{LF} es el tiempo en transmitir el último *Regular SCHC fragment* dentro de la primera ventana. Recordar que este *Regular SCHC fragment* puede ser de un tamaño menor a los *Regular SCHC fragment* enviados antes, pero dentro de la misma ventana. Su cálculo es explicado en la sección 4.2.1.

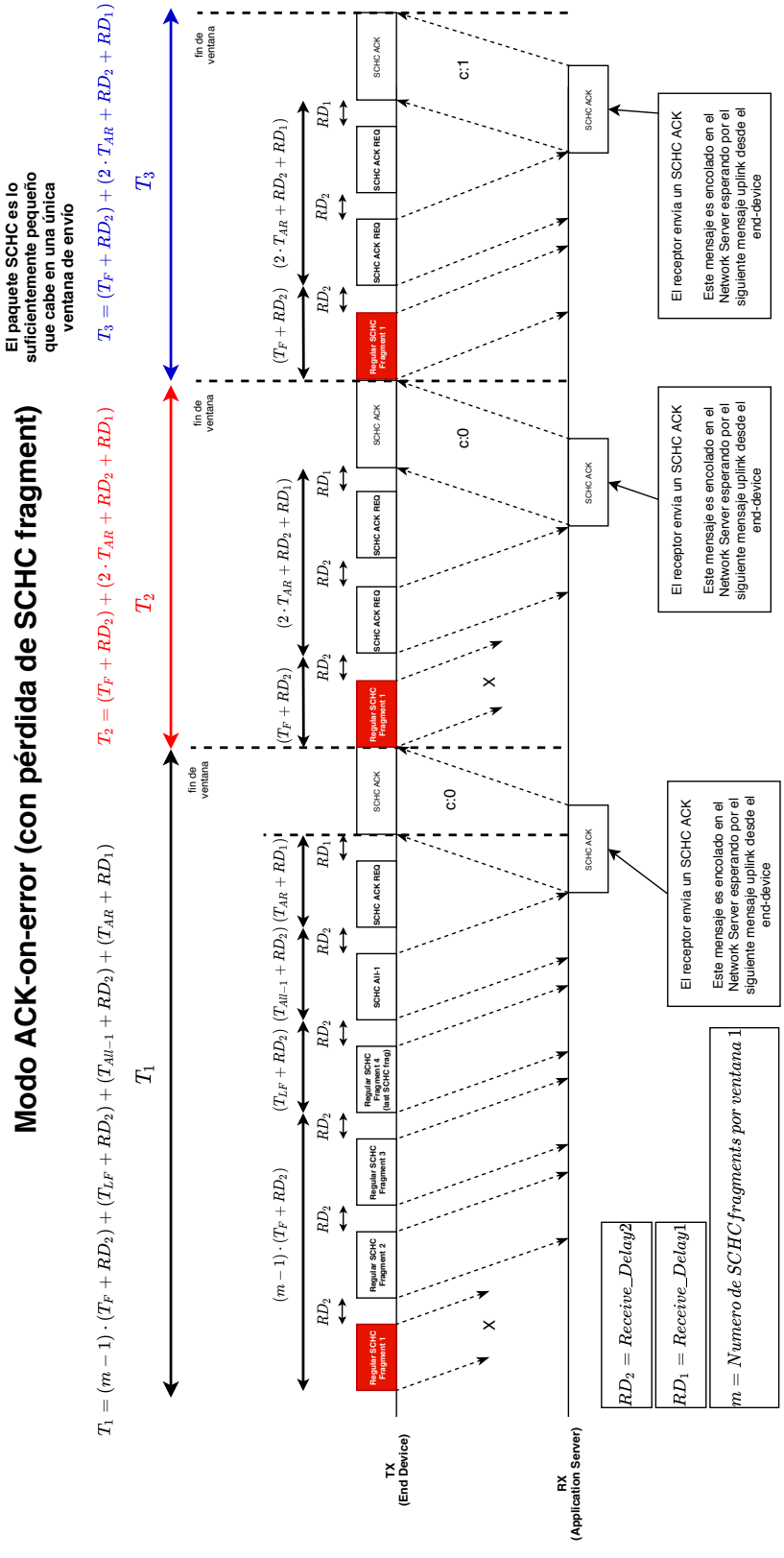


Figura 4.3: Flujo de mensajes en el modo Ack-on-Error (con pérdidas de fragmentos) [elaboración propia]

- T_{All-1} es el tiempo para enviar un *All-1 SCHC Fragment*. Su cálculo es explicado en la sección 4.2.1.
- T_{AR} es el tiempo en transmitir un *SCHC ACK REQ*. Su cálculo es explicado en la sección 4.2.1.
- TP_{ACK} es el tiempo de transmisión del paquete *SCHC ACK*.

Bloque 2 El bloque 2 esta compuesto por todas las ventanas de retransmisión. Puede haber más de una ventana de retransmisión ya que los fragmentos se pueden perder nuevamente. Cada ventana está compuesta por el mensaje *Regular SCHC fragment* que transporta los *tiles* perdidos más el envío de un mensaje *SCHC ACK REQ*. El objetivo del mensaje *SCHC ACK REQ* es abrir una nueva ventana de recepción, considerando que el mensaje *SCHC ACK* se queda encolado en el *network server* al no alcanzar a ser enviado antes de que expiren las ventanas de recepción subsiguientes al envío del último mensaje *Regular SCHC fragment*. Cada ventana de retransmisión tiene un tiempo de duración dado por la ecuación 4.13:

$$T_2 = (T_F + RD_2) + (2 \cdot T_{AR} + RD_2 + RD_1) + TP_{ACK} \tag{4.13}$$

donde:

- T_F es el tiempo que lleva transmitir un mensaje *Regular SCHC fragment* en el medio inalámbrico.
- $RD1$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RECEIVE\_DELAY\_1$ definido en [13].
- $RD2$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RX2$ definido en [13]. Corresponde a la suma entre el $RECEIVE\_DELAY\_2$ y $RX2$.
- T_{AR} es el tiempo en transmitir un *SCHC ACK REQ*.
- TP_{ACK} es el tiempo de transmisión del paquete *SCHC ACK*.

Bloque 3 El bloque 3 está formado por la ventana de retransmisión sin la pérdida del *Regular SCHC fragment*. Esta ventana está compuesta por el mensaje *Regular SCHC fragment* que transporta los *tiles* perdidos. Además incluye el mensaje *SCHC ACK REQ* enviado dos veces. El primer *SCHC ACK REQ* es necesario para que el receptor retransmita el mensaje *SCHC ACK*. El segundo *SCHC ACK REQ* es para recibir el mensaje *SCHC ACK* que quedó encolado en el *Network Server* esperando por un slot de bajada o recepción. Cada ventana de retransmisión tiene un tiempo de duración dado por la ecuación 4.14:

$$T_3 = (T_F + RD_2) + (2 \cdot T_{AR} + RD_2 + RD_1) + TP_{ACK} \tag{4.14}$$

donde:

- T_F es el tiempo que lleva transmitir un mensaje *Regular SCHC fragment* en el medio inalámbrico.
- T_{AR} es el tiempo en transmitir un *SCHC ACK REQ*.
- $RD1$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RECEIVE\_DELAY\_1$ definido en [13].
- $RD2$ es el tiempo que transcurre entre el fin de una transmisión en el enlace de subida y el término del temporizador $RX2$ definido en [13]. Corresponde a la suma entre el $RECEIVE\_DELAY\_2$ y $RX2$.
- TP_{ACK} es el tiempo de transmisión del paquete *SCHC ACK*.

Los tiempo del bloque 2 y bloque 3 son similares. El modelo se separa en 3 bloques para hacer alusión a que existe una ventana de recepción donde finalmente el *Regular SCHC fragment* perdido se recibe de forma correcta.

4.2.3. Eficiencia en el modo Ack-on-Error

Al observar la figura 4.3 y considerando los tiempos de cada bloque explicados en la sección 4.2.2, el tiempo en transmitir exitosamente un *Regular SCHC fragment* en un enlace LoRaWAN^{TM}usando el estándar SCHC con el modo *Ack-on-Error*, puede ser modelado por una variable aleatoria discreta X.

Sea Ω el espacio muestral de todos los posibles resultados para la variable aleatoria X. Ω queda expresado en la ecuación 4.15 como:

$$\Omega = \left\{ \begin{array}{c} x_0 = T_1, \\ x_1 = T_1 + T_3, \\ x_2 = T_1 + T_2 + T_3, \\ x_3 = T_1 + 2T_2 + T_3, \\ x_4 = T_1 + 3T_2 + T_3, \\ x_5 = T_1 + 4T_2 + T_3, \\ \vdots \\ \vdots \end{array} \right\} \tag{4.15}$$

Considerando que $T_2 = T_3$ la ecuación 4.15 puede ser reescrita como muestra la ecuación 4.16.

$$\Omega = \left\{ \begin{array}{c} x_0 = T_1, \\ x_1 = T_1 + T_2, \\ x_2 = T_1 + 2 \cdot T_2, \\ x_3 = T_1 + 3 \cdot T_2, \\ x_4 = T_1 + 4 \cdot T_2, \\ \vdots \\ \vdots \\ x_{\mathrm{i}} = T_1 + \mathrm{i} \cdot T_2 \end{array} \right\} \tag{4.16}$$

La función de probabilidad $P[X = x_{\mathrm{i}}]$ para el espacio muestral Ω definido en la ecuación 4.16 puede ser modelado como una distribución geométrica, siendo p la probabilidad de que un *Regular SCHC fragment* se pierda. La función de probabilidad $P[X = x_{\mathrm{i}}]$ está dada por la ecuación 4.17.

$$P\left(X = x_{\mathrm{i}}\right) = (1 - p) \cdot p^{\mathrm{i}} \tag{4.17}$$

La esperanza de la variable aleatoria X representa el tiempo esperado para que un mensaje *Regular SCHC fragment* pueda llegar con éxito al receptor. La esperanza de la variable X esta dada por la ecuación 4.18.

$$\begin{aligned} E[X] &= \sum_{\mathrm{i}=0}^{\infty} x_{\mathrm{i}} \cdot P[X = x_{\mathrm{i}}] \\ E[X] &= \sum_{\mathrm{i}=0}^{\infty} (T_1 + \mathrm{i} \cdot T_2) \cdot (1 - p) \cdot p^{\mathrm{i}} \end{aligned} \tag{4.18}$$

Al resolver la ecuación 4.18 se obtiene el resultado de la ecuación 4.19.

$$\begin{aligned} E[X] &= \sum_{\mathrm{i}=0}^{\infty} (T_1 + \mathrm{i} \cdot T_2) \cdot (1 - p) \cdot p^{\mathrm{i}} \\ E[X] &= \left(T_1 \cdot (1 - p) \cdot \sum_{\mathrm{i}=0}^{\infty} p^{\mathrm{i}} \right) + \left(T_2 \cdot (1 - p) \sum_{\mathrm{i}=0}^{\infty} \mathrm{i} \cdot p^{\mathrm{i}} \right) \\ E[X] &= \left(T_1 \cdot (1 - p) \cdot \frac{1}{(1 - p)} \right) + \left(T_2 \cdot (1 - p) \cdot \frac{p}{(1 - p)^2} \right) \\ E[X] &= T_1 + T_2 \cdot \frac{p}{(1 - p)} \end{aligned} \tag{4.19}$$

Cálculo de la eficiencia A. García en [24] (capítulo 5.2.1) presenta el cálculo de la eficiencia para el protocolo *Stop and Wait ARQ*. Esta definición puede ser extendida a otros protocolos al utilizar un concepto claro y preciso como es la tasa de transmisión efectiva.

Según el autor, la eficiencia η_0 es la **razón** entre la *tasa de transmisión efectiva* R^0_{eff} y *la tasa de transmisión del canal en bits* denominada R, tal como se indica en la siguiente ecuación:

$$\eta_0 = \frac{R^0_{eff}}{R} \tag{4.20}$$

Así, la *tasa de transmisión efectiva* R^0_{eff} es calculada como la **razón** entre la *cantidad de bits de información entregada al destinatario* y el *tiempo total requerido para transmitir dicha cantidad de bits.*

Para el caso del protocolo SCHC utilizando el modo Ack-on-Error, el *número de bits de información entregado al destinatario* es denominado n_{bits} y está dado por la ecuación 4.21:

$$n_{bits} = n_{tiles} \cdot tile_{size} \tag{4.21}$$

Donde, n_{tiles} es el número de *tiles* dentro de la primera ventana de transmisión. $tile_{size}$ es el tamaño de cada *tile* medido en bits. Por otro lado, el tiempo total requerido para transmitir dicha información en un canal con pérdidas esta dado por la esperanza definida en la ecuación 4.19.

Así, la ecuación 4.22 describe matemáticamente esta relación.

$$R^0_{eff} = \frac{\text{numero de bits de información entregada al destinatario}}{\text{tiempo total requerido para transmitir los bits de información}} = \frac{n_{bits}}{E[X]} \tag{4.22}$$

Utilizando las ecuaciones 4.19, 4.20 y 4.22, se obtiene la ecuación final para la eficiencia de *SCHC* en el modo *Ack-on-Error* dada por la ecuación 4.23.

$$\eta_0 = \frac{R^0_{eff}}{R} = \frac{n_{tiles} \cdot tile_{size}}{\left(T_1 + T_2 \cdot \frac{p}{(1-p)}\right) \cdot R} \tag{4.23}$$

Para el caso de la banda AU915 utilizada en Chile, R puede ser obtenido a partir de la tabla 34 del documento *LoRaWAN^{TM}Regional Parameters* [11].

Capítulo 5

PySCHC: Implementación experimental de SCHC

5.1. Plataforma experimental

La arquitectura experimental está basada en la topología mostrada en la figura 1.3. Cada uno de los componentes están descritos en la sección 1.5.1. El diseño considera cuatro bloques: *End-device*, *Radio Gateway*, *Network Server* y *Application Server*.

A nivel de código fuente, se ha optado por desarrollar una reimplementación del estándar basada en el documento RFC8724 [1] y en la versión 7 del draft "Static Context Header Compression (SCHC) over LoRaWAN"[5]. La razón, al momento de comenzar este trabajo, el código desarrollado en las hackatones de la IETF llamado *OpenSCHC* aun estaba en fase inicial respecto a la tecnología *LoRaWAN*^{TM}.

Además, solo se desarrolla el uso del enlace uplink, esto es, el rol de compresión/fragmentación está ubicado en el *end-device* y el rol de decompresión/desfragmentación está ubicado en el *application server*, tal como lo muestra la figura 5.1.

Se ha seleccionado a Python como lenguaje de desarrollo para implementar el estándar *SCHC* de LoRaWAN [5] por las siguientes razones:

- La placa de desarrollo que representa al *end-device* es programada a través de Micropython.

- Al mismo tiempo, el código desarrollado para el *end-device* puede ser implementado en Google Cloud Platform reutilizando gran parte del desarrollo.

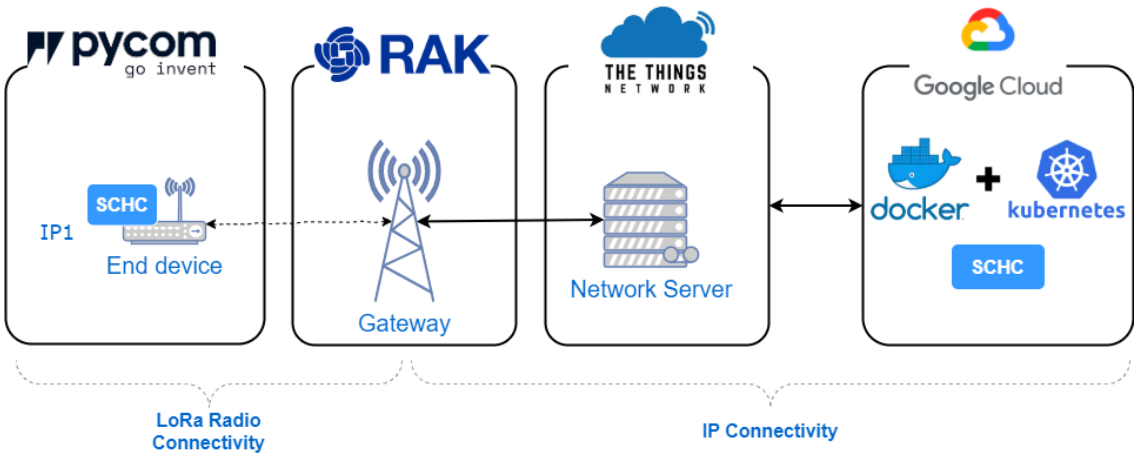


Figura 5.1: Arquitectura utilizada [elaboración propia]

5.2. Desarrollo de PySCHC

5.2.1. Arquitectura de Software

La aplicación PySCHC está dividida en 2 módulos: **PySCHC Compression** y **PySCHC Fragmentation**. A su vez el módulo **PySCHC Fragmentation** está subdividido en 2 módulos: **PySCHC Gateway** y **PySCHC Node**. A continuación el detalle de cada módulo.

- PySCHC_Fragmentation: Código que implementa la Fragmentación/Desfragmentación definida en el estándar.
  - PySCHC_Gateway: En este módulo está el código encargado de la desfragmentación en el enlace uplink. Se debe implementar en el *application server*.
  - PySCHC_Node: En este módulo está el código encargado de la fragmentación en el enlace uplink. Se debe implementar en el *end device*.
- PySCHC_Compression: Código que implementa la Compresion/Descompresión definida en el estándar.

SCHC Compression

La figura 5.2 muestra el diagrama de clases para PySCHC Compression. El detalle de cada clase es el siguiente:

- SCHC_RuleManager: clase que almacena y gestiona el acceso a la reglas.
- SCHC_Compressor: clase que realiza la compresión del *SCHC packet* con base en un paquete IPv6 y un *SCHC Rulemanager* con la lista de reglas soportadas.
- PacketGenerator: clase que genera un paquete IPv6 (usado solo para generar tráfico).
- SCHC_Parser: clase que permite convertir un paquete IPv6 en formato de bytes en un diccionario Python.

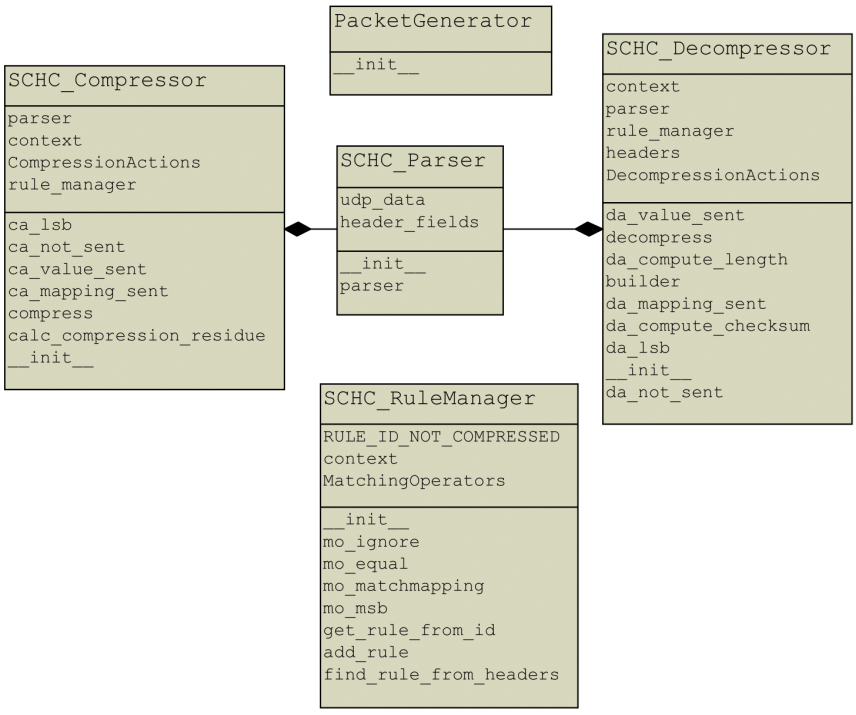


Figura 5.2: Diagrama de Clases módulo PySCHC Compression [elaboración propia]

- SCHC_Decompressor: clase que realiza la decompresión del *SCHC packet* con base en un *ruleId* y un *SCHC Rulemanager* con la lista de reglas soportadas. Entrega un nuevo paquete IPv6.

SCHC Fragmentation

Las figuras 5.3 y 5.4 muestran el diagrama de clases para PySCHC Fragmentation. El detalle de cada clase es el siguiente:

- LoRaWAN_Pycom: clase encargada de implementar los métodos de comunicación de bajo nivel. En particular implementa los métodos para utilizar la placa de desarrollo LoPy4.
- SCHC_Fragmenter_Node: clase encargada de manejar y almacenar las sesiones SCHC. Además tiene las referencias a los métodos que permiten comunicarse a través de LoRaWAN.
- SCHC_Session_Node: esta clase permite almacenar las variables asociadas a cada sesión SCHC. Tiene un puntero a la máquina de estados que implementa el protocolo SCHC.
- SCHC_Session_Gateway: esta clase permite almacenar las variables asociadas a cada sesión SCHC. Tiene un puntero a la máquina de estados que implementa el protocolo SCHC.
- LoRaWAN_ACK_On_Error_TX: clase que implementa el protocolo SCHC para un enlace ascendente (uplink) para el protocolo LoRaWAN en el lado del *end-device*. Realiza la fragmentación.

Figura 5.3: Diagrama de Clases módulo PySCHC Node [elaboración propia]

- SCHC_Message: clase encargada de la codificación y decodificación de los mensajes SCHC para fragmentación.
- LoRaWAN_ACK_On_Error_RX: clase que implementa el protocolo SCHC para un enlace ascendente (uplink) para el protocolo LoRaWAN en el lado del *application server*. Realiza la desfragmentación.

5.2.2. Problemas en el desarrollo

Esta sección muestra los problemas encontrados al momento de desarrollar el estándar SCHC usando lo definido en los documentos *draft* del *LPWAN Group de la IETF*. Cabe destacar que esta tesis fue desarrollada mientras se iban liberando distintas versiones del *draft* de SCHC. Al momento de su publicación, SCHC ya es un estándar definido en la RFC8724 [1].

LoRaWAN profile definition

En la sección 8.4.3, del documento "draft-ietf-lpwan-ipv6-static-context-hc-19", se indica que cada perfil debe definir lo siguiente:

- si el último *tile* se debe transportar en un *Regular SCHC Fragment* o en un *All-1 SCHC Fragment* (sección 8.4.3.1)

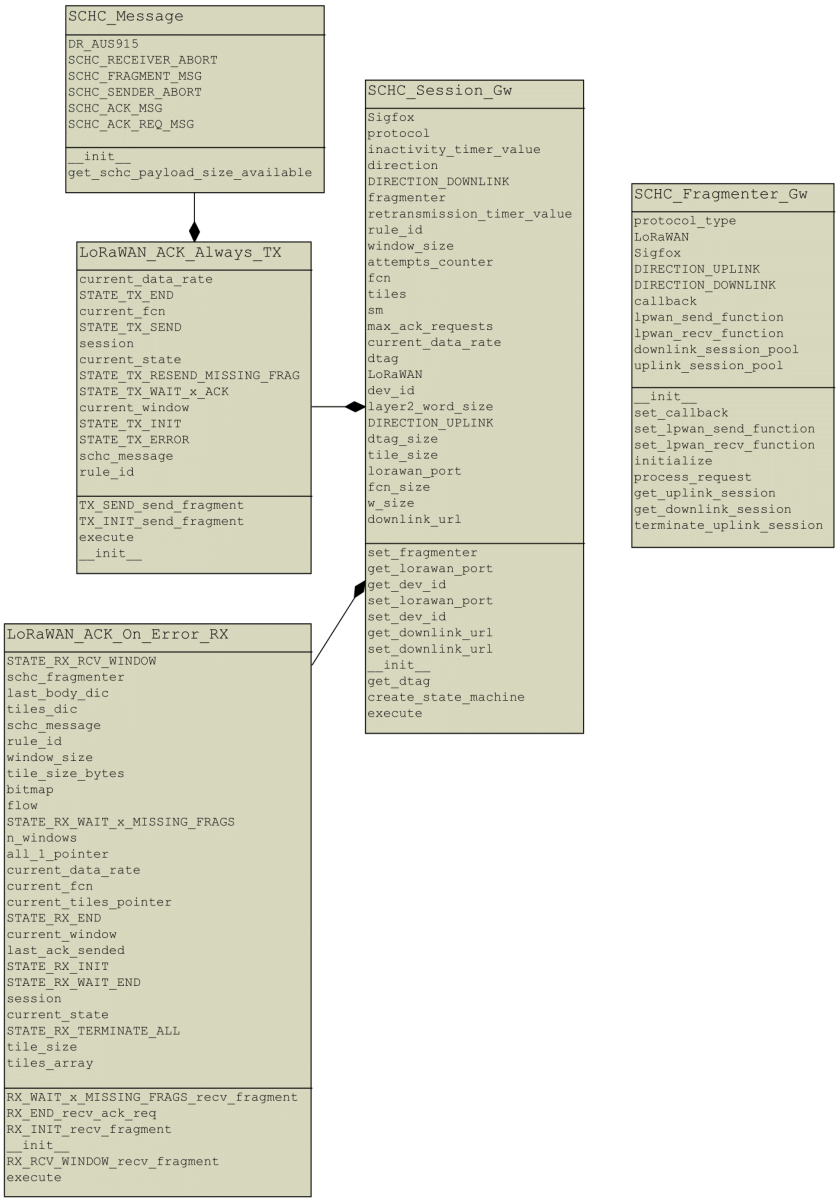


Figura 5.4: Diagrama de Clases módulo PySCHC Gateway [elaboración propia]

- si el penúltimo *tile* puede tener o no un largo de una palabra en capa 2 más pequeña que el tamaño de un *tile* normal. El largo de una palabra en capa 2 para LoRaWAN es de 1 byte. En este caso, el tamaño del *tile* regular debe ser al menos el doble del tamaño de una palabra en capa 2.

Pero al revisar la definición del *profile* para *LoRaWAN* en el borrador "draft-ietf-lpwan-schc-over-lorawan-02"no se encuentra una definición.

Finalmente, los autores del draft "draft-ietf-lpwan-schc-over-lorawan-05"para *LoRaWAN* definen lo siguiente:

- El último *tile* en la sesión de fragmentación de enlace ascendente se puede transportar

en un *All-1 SCHC fragment* usando FPortUpDefault o FPortUpShort.

- El penúltimo *tile* tiene el mismo tamaño que los otros *tiles*

Temporizadores de retransmisión e inactividad

En la sección 5.6.2, el borrador "draft-ietf-lpwan-schc-over-lorawan-07"[5] se indica:

- Temporizadores de retransmisión e inactividad: los *end-devices* en LoRaWAN no implementan un temporizador de retransmisión. Al final de una ventana o una sesión de fragmentación, los mensajes *SCHC ACK* correspondientes se transmiten por el *SCHC Gateway* (servidor de aplicaciones LoRaWAN) en la ventana de recepción RX1 o RX2 del *end-device*. Al final de sus ventanas RX, el *end-device* envía un *SCHC ACK REQ* para solicitar una retransmisión de un *SCHC ACK*, y puede implementar un "temporizador de inactividad". La duración RECOMENDADA predeterminada de este temporizador es de 12 horas, según los requisitos de la aplicación y PUEDE ser modificado por la aplicación.

El párrafo anterior no define cuanto tiempo debe esperar el *end-device* antes de enviar el *SCHC ACK REQ* para solicitar una retransmisión del *SCHC ACK* por parte del *SCHC Gateway*.

En la implementación se utilizó un tiempo de espera cero. Esto quiere decir que después de enviar el ultimo *SCHC Regular fragment* de la ventana y esperar el temporizador del $RECEIVE\_DELAY\_2 + RX2_{min}$, se envía inmediatamente un *SCHC ACK REQ* para solicitar la retransmisión del SCHC ACK. La figura 5.5 muestra el flujo de mensajes para el envío exitoso de una ventana.

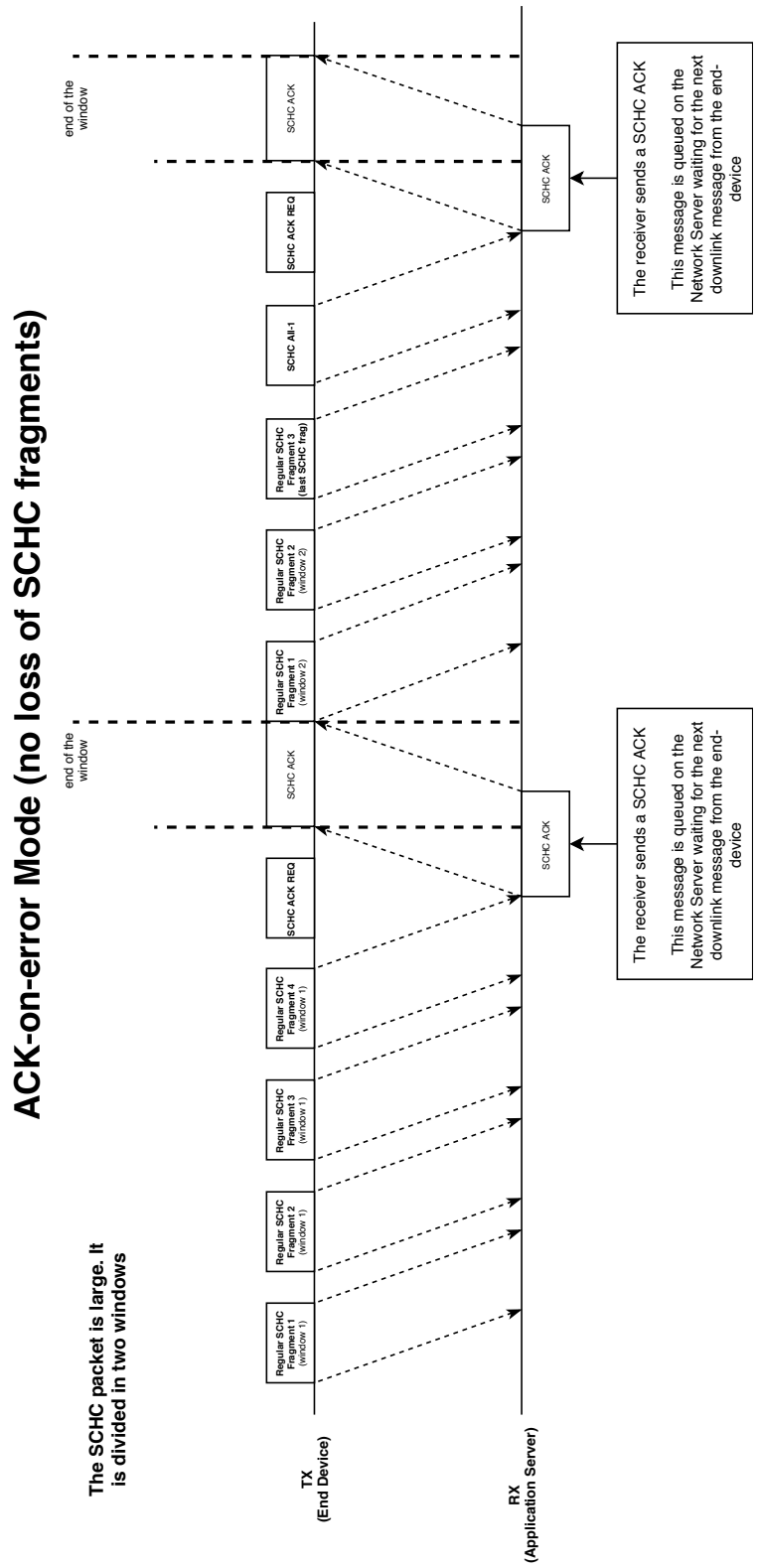


Figura 5.5: Flujo de mensajes exitosos - Modo Ack on error [elaboración propia]

5.3. Despliegue de PySCHC

5.3.1. Instalación en Google Cloud Platform

El código *PySCHC* para la desfragmentación es desplegado en un *container* utilizando *Docker*. Un *container* es un paquete cerrado que contiene todo lo necesario para que una aplicación o un servicio se pueda ejecutar de manera encapsulada dentro de una sola imagen, independiente al servidor anfitrión que lo aloje, incluyendo tanto binarios como archivos de configuración y demás archivos que necesite.

Docker, por otro lado, es un sistema que permite construir, transferir, desplegar y ejecutar los *containers* con las aplicaciones de una manera muy sencilla y confiable, garantizando un despliegue escalable de forma eficiente sin importar el sistema operativo anfitrión.

Si bien *Docker* mismo permite ejecutar los *containers* creados y las herramientas de gestión que provee pueden ser suficientes para gestionar un solo servidor, resultan muy poco prácticas de utilizar a gran escala para montar una infraestructura distribuida de múltiples *containers* corriendo en tal vez docenas o cientos de servidores al mismo tiempo.

Para esto se necesita un sistema que se encargue de gestionar todo el *cluster* de servidores, eso significa un sistema que se encargue de distribuir los *containers* a través del sistema según los recursos disponibles en el *cluster*, además de crear, ejecutar, vigilar, medir, destruir y relanzar los *containers*, debe mantener y controlar en todo momento cada aspecto relevante de los *containers* y su estado, de eso se encarga *Kubernetes*.

PySCHC es desplegado en *Google Kubernetes Engine (GKE)*. GKE es una plataforma basada en *Kubernetes*. Los pasos para instalar la aplicación PySCHC en *Kubernetes* están detallados en el Anexo F. El código de PySCHC implementado para esta tesis se encuentra disponible de forma pública para su descarga desde Github[1].

[1] https://github.com/RodrigoMunozLara/PySCHC

Capítulo 6

Análisis y Resultados

En este capítulo se presenta en detalle el diseño de los experimentos. Primero se muestra la simulación para obtener la eficiencia en función de la probabilidad de error. Segundo, se presenta el experimento para la eficiencia en función del *spreading factor* y el tamaño de *tile*. Para ambos experimentos, los modelos se presentan en el capítulo 4.

Para cada uno de estos experimentos existe una sección de resultados. En ella se revisa el comportamiento de la eficiencia en el canal de subida para $LoRaWAN^{TM}$ utilizando el estándar *SCHC*. Se discute la relación entre la eficiencia, el *spreading factor*, el tamaño de *tile* y la probabilidad de error.

6.1. Análisis de la compresión

6.1.1. Diseño de los experimentos - Time on Air vs Porcentaje de compresión

Basado en la metodología descrita en la sección 4.1.1, se calculan los porcentajes de compresión para las 64 reglas resultantes que se aplican a LoRaWAN^{TM}utilizando los valores enumerados en la Tabla 6.1. El *ToA* se calcula en función de las ecuaciones (4.2) - (4.8).

6.1.2. Resultados - Time on Air versus Porcentaje de compresión

Los resultados del *time on air* para cada *spreading factor* en función del porcentaje de compresión se ilustran en la figura 6.1. La evaluación del ΔToA y ToA_{range} se basa en los porcentajes de compresión de las 64 reglas ilustradas en la figura 6.1. Hay que tener en cuenta que hay casos en los que varias reglas alcanzan el mismo porcentaje de compresión; por lo tanto, se trazan menos de 64 puntos para las curvas de cada factor de dispersión.

Tabla 6.1: Parámetros de evaluación [elaboración propia]

Parámetro	**Valor**
Spreading Factors	SF7 a SF12
Channel Bandwidth	125 KHz
Rule ID length	1 byte
UDP Payload	10 bytes
Explicit header mode (H)	disabled (0)
Adaptive Data Rate (DE)	1 (SF>=11), 0 (SF<11))
Cyclic Redundancy Check mode (CRC)	enabled (1)
Coding Rate mode (CR)	1

La figura 6.2 muestra los valores obtenidos para el ΔToA en función del *spreading factor* cuando el mensaje comprimido utiliza los valores de la tabla 6.1. La figura 6.3 muestra el ToA_{range} en función del porcentaje de compresión para un *spreading factor* de 12.

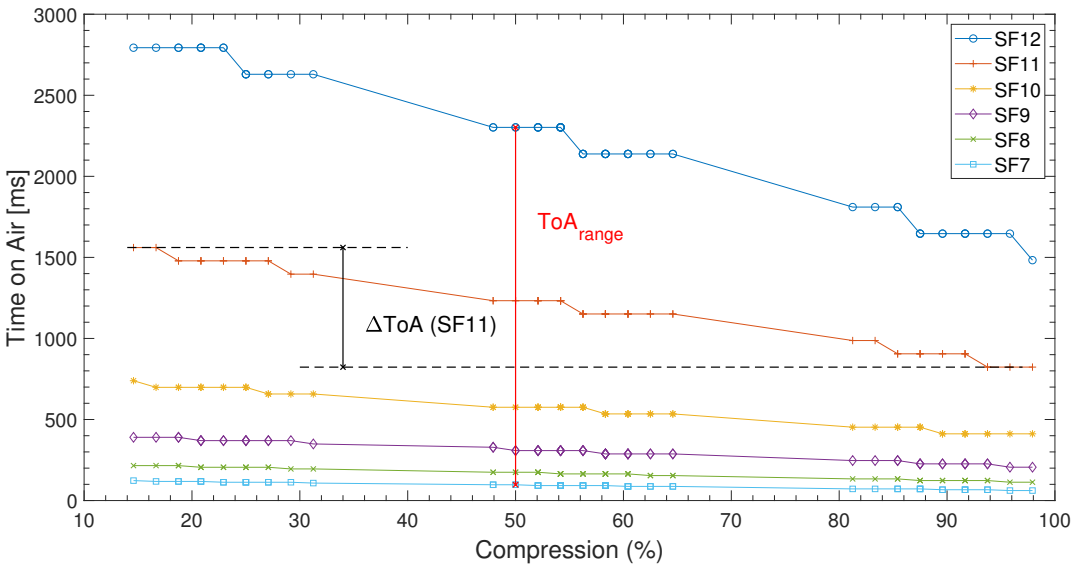


Figura 6.1: Time on Air según el porcentaje de compresión para diferentes spreading factor

6.1.3. Discusión - Time on Air versus Porcentaje de compresión

La figura 6.1 muestra que el *ToA* aumenta cuando el porcentaje de compresión disminuye. Esto es esperable, ya que la disminución porcentual da como resultado paquetes SCHC más grandes. Sin embargo, en la figura 6.1 observamos que para cada *spreading factor* hay segmentos, llamados Tiempo *Constant Time on Air Segments (CTS)*, donde el *ToA* permanece constante a pesar de las variaciones en los porcentajes de compresión. Este resultado se debe a la función de paso *ceil* en la ecuación (4.7). Con fines ilustrativos, también se proporcionan ejemplos de las dos métricas adicionales, ΔToA y ToA_{range} en la misma figura. Por ejemplo, dado un *spreading factor* 11, el ΔToA es 737,3 *ms*. considerando los valores PC_{min} y

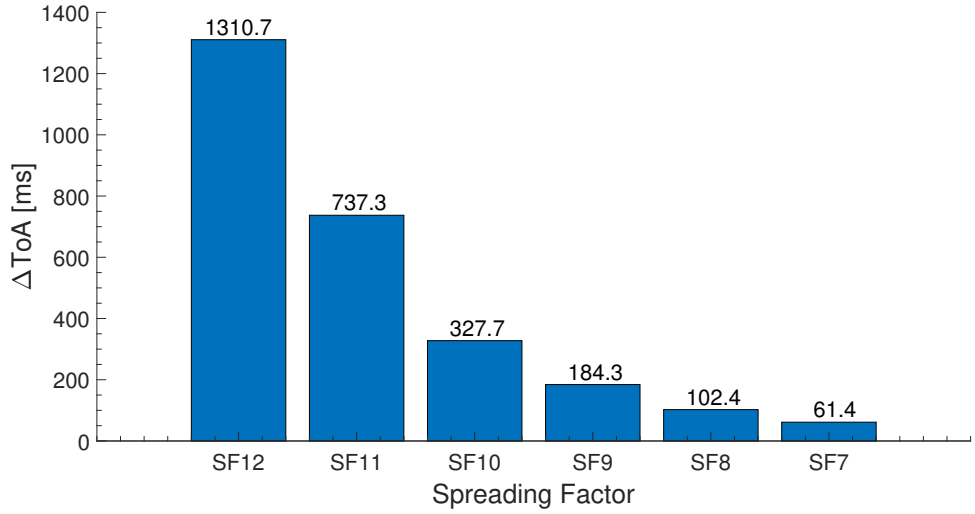


Figura 6.2: Time on Air variation vs Spead Factor

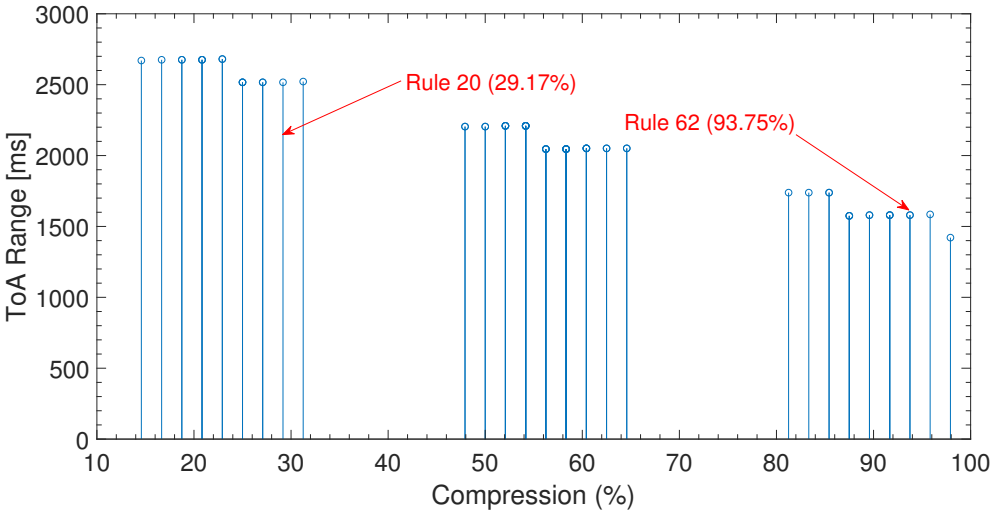


Figura 6.3: Time on Air range vs porcentaje de compresión

PC_{max}. En el caso de ToA_{range}, se muestra el valor de $PC = 58,3\,\%$, que es un porcentaje de compresión común a 5 reglas diferentes. En este caso, el ToA_{range} es igual a 2045,7 ms.

La figura 6.2 muestra los valores para el ΔToA en funcion de los diferentes *spreading factor*. Cabe señalar que estos valores no dependen del tamaño de la carga útil UDP, ya que solo consideran los porcentajes de compresión de encabezados logrados en todo el conjunto de reglas. Conocer el comportamiento del ΔToA es útil para los casos en que varios paquetes del mismo nodo requieren reglas de compresión diferentes. Si la aplicación es sensible a las variaciones de retraso, es preferible utilizar un *spreading factor* más pequeño (por ejemplo, $SF7$) en lugar de un *spreading factor* más grande, lo que implica mayores variaciones de retraso en las diferentes reglas.

La figura 6.3 muestra la variación de la variable ToA_{range} en función del porcentaje de

compresión. Se observa que hay segmentos en los que el rango no varía a pesar de la disminución o aumento de los porcentajes de compresión. Esto se debe al *CTS* identificado en la figura 6.1. Es mejor elegir el porcentaje de compresión más alto posible para evitar tener un rango muy alto en el retraso de la transmisión. Por lo tanto, si se modifican las características del enlace *LoRa*, lo que hace que los paquetes se transmitan a una velocidad menor (debido a la modificación del *spreading factor*), el rango de variación del ToA será menor, como se ha demostrado para mayores porcentajes de compresión.

Este análisis permite argumentar que la elección de la Regla de compresión en SCHC no debe basarse únicamente en el porcentaje de compresión logrado por esa regla específica. Al evaluar el porcentaje de compresión junto con el comportamiento del ToA, se ha podido determinar que hay rangos donde los retardos son insensibles a los porcentajes de compresión. Además, se ha determinado que hay reglas de compresión que son mucho más sensibles a las variaciones del *spreading factor*, lo que resulta en un aumento del ToA.

Por ejemplo, suponiendo que un dispositivo final está ejecutando una aplicación que es sensible a las variaciones de la demora (por ejemplo, que afecta a temporizadores o máquinas de estado) y el enlace *LoRa* es estable usando un *spreading factor* 10. Al examinar la figura 6.1, es posible determinar que el ToA se mantiene estable a 400 ms para reglas que tienen un porcentaje de compresión entre 99 % y 89,7 %. Por lo tanto, estas reglas son las mejores candidatas para ser empleadas por el dispositivo final.
Ahora considere un ejemplo en el que una aplicación se ha configurado para usar una regla fija con un porcentaje de compresión de 93,75 %. La figura 6.3 muestra que el ToA_{range} para este porcentaje de compresión es 1579,8 ms. Este valor significa que si la aplicación comienza a funcionar con un *spreading factor* 7, y debido a un cambio en las condiciones del enlace el dispositivo se ve obligado a usar un *spreading factor* 12, el retraso aumentará como máximo en 1579,8 ms. Ahora suponga que la misma aplicación está configurada para usar una regla con 29,17 % de compresión. Si la aplicación comienza a funcionar con *spreading factor* 7, y debido a un cambio en las condiciones del enlace, el dispositivo se ve obligado a usar un *spreading factor* 12, el retraso aumentará como máximo en 2516,7 ms. Lo anterior indica que cuando ocurre una variación del *spreading factor*, la segunda regla aumentará su retraso en casi 1 segundo más en comparación con la regla inicial. Esto puede tener un gran impacto para aplicaciones que son sensibles al retraso introducido por las variaciones del *spreading factor* tal como sistemas de advertencia en carretera en redes *Internet of Vehicle*.

6.2. Análisis de la fragmentación

6.2.1. Diseño de los experimentos: Simulación 1 - Eficiencia versus Probabilidad de Error

Esta simulación consiste en calcular la eficiencia en función de la probabilidad de pérdida del primer *SCHC fragment* en un enlace de subida LoRaWAN™utilizando el protocolo SCHC. Los resultados son calculados utilizando la ecuación 4.23 y el método de Montecarlo.

Tabla 6.2: Parámetros usados para la simulación de Eficiencia v/s Probabilidad [elaboración propia]

Parámetro	**Valor**
spreading factor	SF7 a SF12
Temporizador para activar primera ventana de recepción LoRaWAN	1 segundo
Temporizador para activar segunda ventana de recepción LoRaWAN	2 segundos
Ancho de banda LoRaWAN	125 KHz
Tamaño de paquete SCHC	400 bytes
Tamaño de *Tile*	10 bytes
Número de Simulaciones	100.000

Para la simulación se asumen los siguientes supuestos:

- El paquete SCHC o *SCHC packet* es del tamaño suficiente para ser enviado en una ventana de fragmentación. Esto quiere decir, utilizando 63 *tiles* de 10 bytes cada uno, o sea, 630 bytes como máximo. Para el caso particular de esta simulación se ha usado un *SCHC packet* de 400 bytes.

- En la transmisión de un *SCHC packet*, el *spreading factor* de LoRa es el mismo para todos los *Regular SCHC fragments*.

- Debido a que cada *Regular SCHC Fragment* transporta $\boldsymbol{n}$ *tiles*, el último *Regular SCHC fragment* enviado podría no transportar $\boldsymbol{n}$ *tiles*. Por ejemplo: un *SCHC packet* de 400 bytes es dividido en 40 *tiles* (tamaño de un *tile* es 10 bytes). Suponga que para un *spreading factor* dado, cada *Regular SCHC fragment* puede transportar a lo más, 11 *tiles*. Los *Regular SCHC fragment* 1, 2 y 3 habrán transportado 33 *tiles* en total. Dado que quedan 7 *tiles* pendientes de transmisión para completar los 40 *tiles*, el último *Regular SCHC fragment* SOLO transportará 6 *tiles* ya que el mensaje *All-1 SCHC Fragment* llevará el último *tile*. El modelo por lo tanto, considera que el último *Regular SCHC fragment* puede ser de un tamaño menor o igual a los *Regular SCHC fragment* enviados previamente.

- Debido al no procesamiento inmediato por parte del receptor de un mensaje *All-1 SCHC Fragment*, el mensaje *SCHC ACK* enviado en respuesta a un *All-1 SCHC Fragment* es encolado en el *Network Server*, esperando el mensaje *SCHC ACK Request* que es enviado por el transmisor (ver figura 4.3).

La tabla 6.2 entrega los valores utilizados en la simulación:

El experimento considera una simulación por cada *spreading factor* y por cada probabilidad de error. Existen seis *spreading factor*, del 7 al 12. El vector de probabilidad es de 10 valores, desde el 0 hasta el 0,9 en pasos de 0,1. Cada simulación esta compuesta por 100.000 iteraciones. La figura 4.3 muestra los tiempos y mensajes involucrados en cada simulación.

6.2.2. Diseño de los experimentos: Experimento 1 - Eficiencia versus spreading factor versus Tamaño de *Tile*

Este experimento se basa en la plataforma definida en la sección 5.1. El *end-device* ubicado en el nodo *Pycom*, juega el rol de **fragmentador** en el enlace de subida *LoRaWAN*^{TM} utilizando el protocolo *SCHC*. Por otro lado, el *gateway SCHC* ubicado en *Google Cloud Platform* tiene el rol de **desfragmentador**.

Para cada experimento se escoge un *spreading factor* y un tamaño de *tile*. Los *spreading factor* van desde 7 hasta 12. Los tamaños de *tile* van desde 8 bytes hasta 15 bytes. Por lo tanto, hay una combinación de 48 experimentos. Para cada uno de ellos el *end-device* envía 20 *SCHC packet*. Cada uno de estos son fragmentados y calculados sus tiempos de envío.

El experimento está basado en los siguientes supuestos:

- El paquete SCHC o *SCHC packet* es del tamaño suficiente para ser enviado en una ventana de fragmentación. Esto quiere decir, utilizando 63 *tiles* de 10 bytes cada uno, o sea, 630 bytes como máximo.
- El *spreading factor* de LoRa es el mismo para todos los *Regular SCHC fragments*.
- Debido a que cada *Regular SCHC Fragment* transporta *"n"tiles*, el último *Regular SCHC fragment* enviado podría no transportar *"n"tiles*. Por ejemplo: un *SCHC packet* de 500 bytes es dividido en 50 *tiles* (tamaño de un *tile* es 10 bytes). Suponga que para un *spreading factor* dado, cada *Regular SCHC fragment* puede transportar a lo más, 11 *tiles*. Los *Regular SCHC fragment* 1, 2, 3 y 4 habrán transportado 44 *tiles* en total. Dado que quedan 6 *tiles* pendientes de transmisión para completar los 50 *tiles*, el último *Regular SCHC fragment* SOLO transportará 5 *tiles* ya que el mensaje *All-1 SCHC Fragment* llevará el último *tile*. El modelo, por lo tanto, considera que el último *Regular SCHC fragment* puede ser de un tamaño menor o igual a los *Regular SCHC fragment* enviados previamente.
- Todos los mensajes en modo *downlink* son recibidos por el *end device* en la segunda ventana de recepción LoRa.
- Debido al no procesamiento inmediato por parte del receptor de un mensaje *All-1 SCHC Fragment*, el mensaje *SCHC ACK* enviado en respuesta a un *All-1 SCHC Fragment* es encolado en el *Network Server*, esperando el mensaje *SCHC ACK Request* que es enviado por el transmisor (ver figura 4.3).
- Este experimento es realizado en un ambiente controlado, donde el *Radio Gateway* y el *end-device* están separados por un par de metros. Por lo anterior, este experimento considera que no hay perdidas de mensajes

La tabla 6.3 entrega los valores de los parámetros utilizados y la figura 4.2 muestra los mensajes involucrados en un experimento.

Parámetros Experimento	Valor
Ancho de banda LoRaWAN	125 KHz
Banda de operación LoRaWAN	AU915
Modo de envío de mensajes LoRaWAN	sin confirmación
Metodo de autenticación LoRaWAN	ABP
spreading factor	SF7 a SF12
Tamaño de paquete SCHC	400 bytes
Tamaño de *Tile*	10 bytes
Número de Simulaciones	100.000

Tabla 6.3: Parámetros usados para el experimento de Eficiencia v/s spreading factor [elaboración propia]

6.2.3. Resultados - Eficiencia vs Probabilidad de Error

A partir de la simulación explicada en la sección 6.2.1 y utilizando el código en Matlab mostrado en el Anexo A se presenta el resultado para la eficiencia en función de la probabilidad de error. En la figura 6.4 se muestra la relación entre eficiencia promedio y probabilidad de error p para cada spreading factor a través de distintas curvas. Además, para cada valor de eficiencia promedio se muestra su desviación estándar como barras verticales en cada muestra. Se observa una disminución exponencial de la eficiencia para todos los *spreading factors* a medida que aumenta la probabilidad de error. Destaca el comportamiento para el spreading factor 10, en el cual la eficiencia decae más lentamente que para el resto de los *spreading factor*.

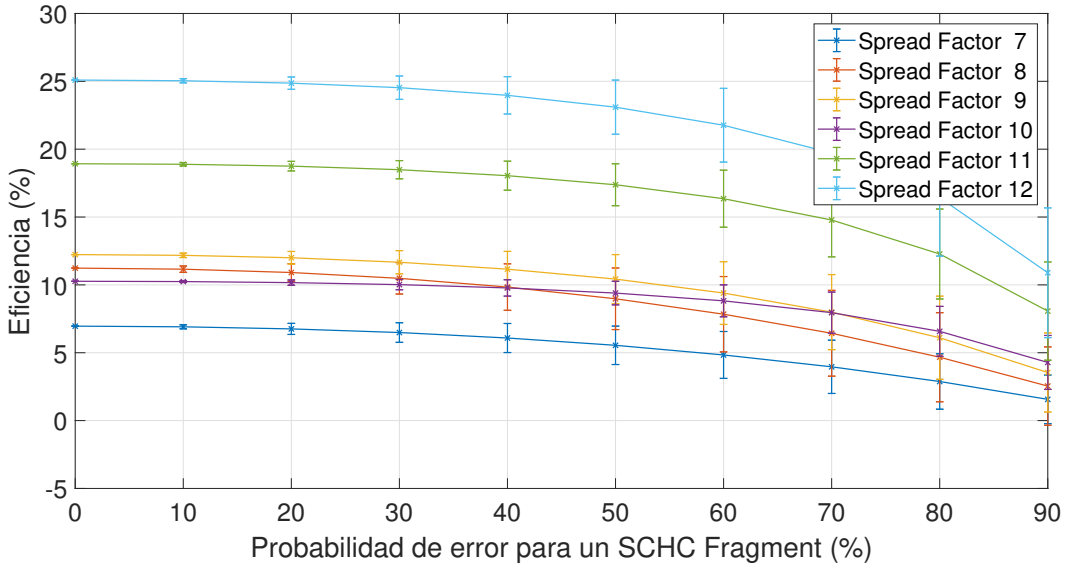


Figura 6.4: Eficiencia vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia]

Respecto a la máxima eficiencia alcanzada en función de la probabilidad, esta puede ser divida en 3 rangos. El primer rango, para una probabilidad de error entre 0 y 42, 10 % tiene al

spreading factor 12 como el factor más eficiente en función de la probabilidad. En este rango el orden decreciente de spreading factor es SF12, SF11, SF9, SF8, SF10, SF7. El segundo rango es para una probabilidad de error entre $42,10\,\%$ y $71,05\,\%$. El spreading factor 12 sigue siendo el más eficiente, pero el orden de los factores se ve modificado. En este rango el orden decreciente de spreading factor es SF12, SF11, SF9, SF10, SF8, SF7. Finalmente,en el tercer rango, para una probabilidad $71,05\,\%$ y $90\,\%$ el orden en el que los *spreading factor* están distribuidos es (en orden decreciente): SF12, SF11, SF10, SF9, SF8, SF7. En todos los casos el mejor spreading factor es el SF12 y el menos eficiente es el spreading factor 7.

La tabla 6.4 muestra los valores de eficiencia cuando la probabilidad de error es igual a cero. Estos valores son importantes ya que son las máximas eficiencias que puede alcanzar el protocolo SCHC en función de cada *spreading factor*.

spreading factor	Eficiencia Promedio
12	25,7 %
11	19,54 %
9	12,75 %
8	11,75 %
10	10,67 %
7	7,31 %

Tabla 6.4: Eficiencia vs *spreading factor* para un tamaño de *tile* de 10 bytes y una probabilidad de error igual a cero [elaboración propia]

6.2.4. Discusión - Eficiencia vs probabilidad - comportamiento *spreading factor* 10

En la figura 6.4, la curva para el *spreading factor* 10 muestra una tendencia diferente que el resto de los *spreading factors*. El numerador de la ecuación 4.23 es constante, por lo tanto, el comportamiento diferente del *spreading factor* 10 se debe al denominador en la ecuación. Al graficar el factor $T_1 + T_2 \cdot \frac{p}{(1-p)}$ del denominador en la ecuación 4.23, se obtiene la figura 6.5. En esta figura se observa que el factor $T_1 + T_2 \cdot \frac{p}{(1-p)}$ para los distintos spreading factor se mantiene creciente exponencialmente y no hay un cruce de curvas a diferencia de lo visto en la figura 6.4.

El comportamiento de R para los distintos spreading factor tiene una tendencia no lineal (ver tabla 34 en el documento *LoRaWAN^{TM}Regional Parameters* [11].). Así, solo queda revisar el comportamiento del factor $(T_1 + T_2 \cdot \frac{p}{(1-p)}) \cdot R$. La figura 6.6 muestra la relación de dicho factor en función de la probabilidad de error. La unidad de medida del factor $(T_1 + T_2 \cdot \frac{p}{(1-p)}) \cdot R$ es *bits*.

Al observar la figura 6.6 se ve claramente la tendencia para el caso del *spreading factor* 10 el cual presenta un crecimiento exponencial diferente al resto de los *spreading factors*. Esto está exclusivamente relacionado a la multiplicación entre R, T_1 y T_2. Es así como para una probabilidad de error $p = 0,421$ la eficiencia es la misma tanto para el *spreading factor* 10

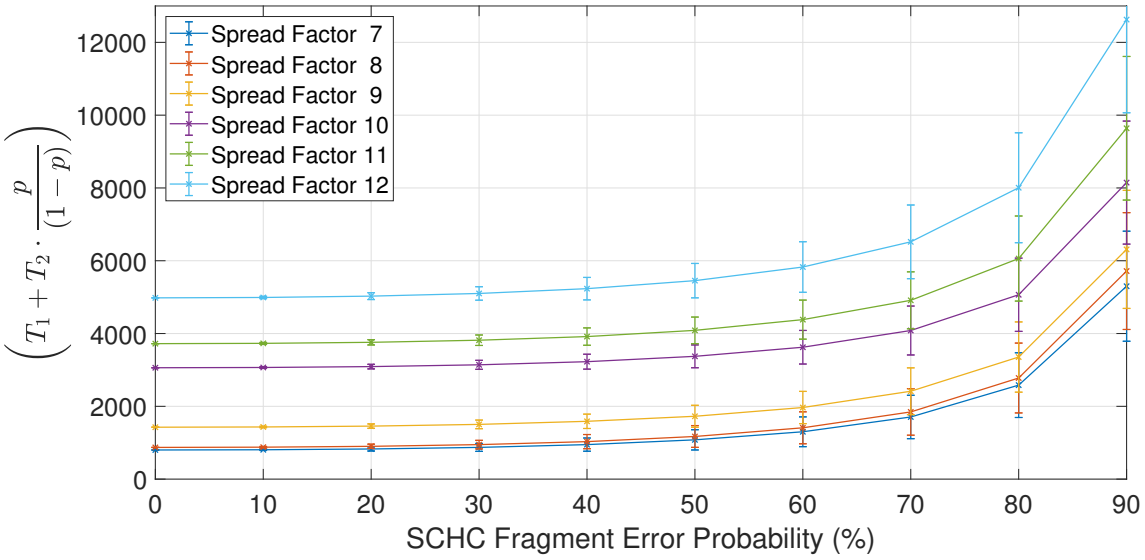


Figura 6.5: Factor $\left(T_1 + T_2 \cdot \frac{p}{(1-p)}\right)$ vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia]

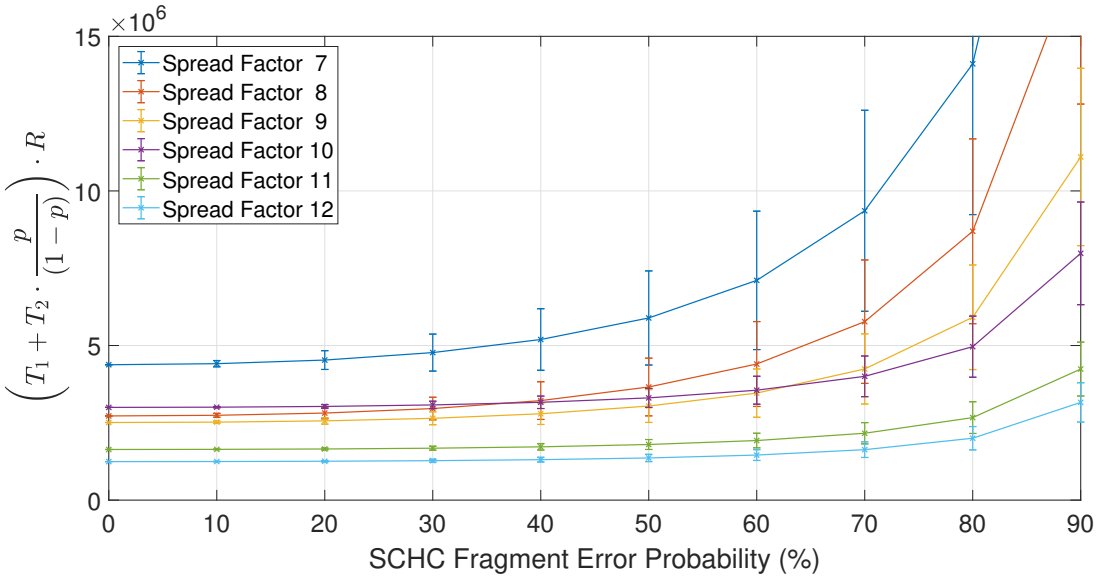


Figura 6.6: Factor $\left(T_1 + T_2 \cdot \frac{p}{(1-p)}\right) \cdot R$ vs Probabilidad de error del primer *SCHC Fragment* [elaboración propia]

como para el *spreading factor* 8.

6.2.5. Resultados - Eficiencia vs spreading factor

Cálculo Teórico

Para obtener la eficiencia en función del *spreading factor* se utiliza el modelo definido en la sección 4.2.1 que entrega los tiempos de cada grupo de mensajes o bloques, junto a la ecuación 4.23 para el cálculo de la eficiencia. Se utiliza una probabilidad $p = 0$ dado que es un modelo sin pérdidas de paquetes.

El gráfico de la figura 6.7 muestra la eficiencia en función del *spreading factor* en un enlace de subida *LoRaWAN*^{TM} utilizando el protocolo *SCHC*. Cada curva en este gráfico representa a un tamaño de *tile* distinto. Se evalúan los *tiles* de tamaños 8, 10, 12 y 14 bytes.

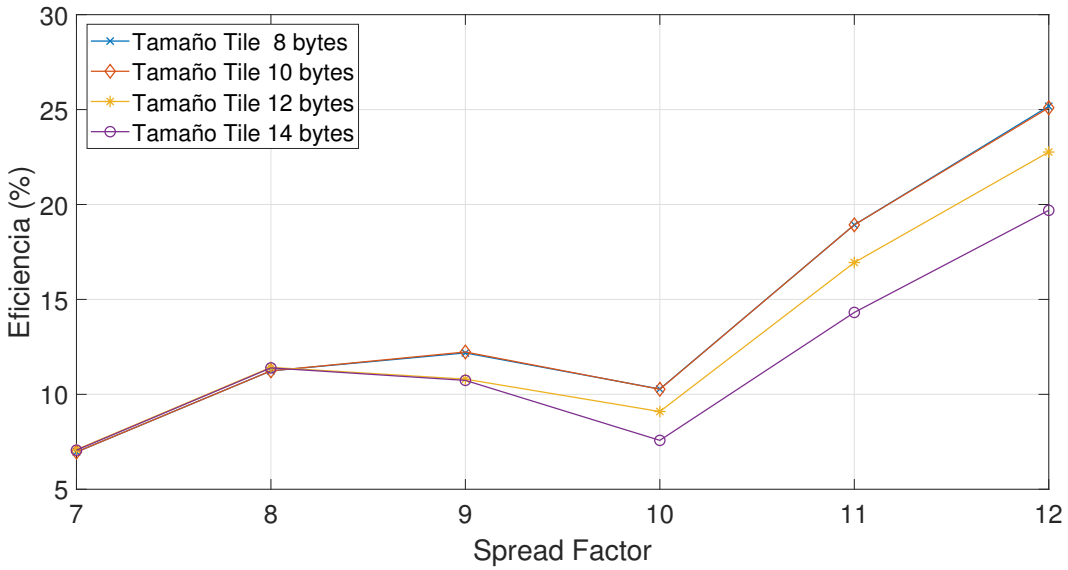


Figura 6.7: Eficiencia Teórica vs spreading factor [elaboración propia]

Dado que el estándar SCHC considera un tamaño de *tile* de 10 bytes, la tabla 6.5 muestra la eficiencia promedio para este tamaño de *tile* en función del *spreading factor*.

spreading factor	Eficiencia
7	6,96 %
8	11,24 %
9	12,23 %
10	10,27 %
11	18,93 %
12	25,09 %

Tabla 6.5: Eficiencia Teórica vs *spreading factor* para un tamaño de *tile* de 10 bytes [elaboración propia]

Para todos los tamaños de *tiles*, la mayor eficiencia es para el *spreading factor* 12 y la

menor eficiencia para el *spreading factor* 7.

Cálculo Empírico

El cálculo empírico de la eficiencia promedio en función del *spreading factor* se realiza a partir de la definición entregada en la ecuación 4.23. Los tiempos para esta ecuación son obtenidos a través del experimento explicado en la sección 6.2.2 y el código en Matlab mostrado en el Anexo C.

El resultado del experimento es ilustrado en la figura 6.8, donde se muestra la relación entre eficiencia empírica promedio y *spreading factor* para cada tamaño de *tile* a través de distintas curvas.

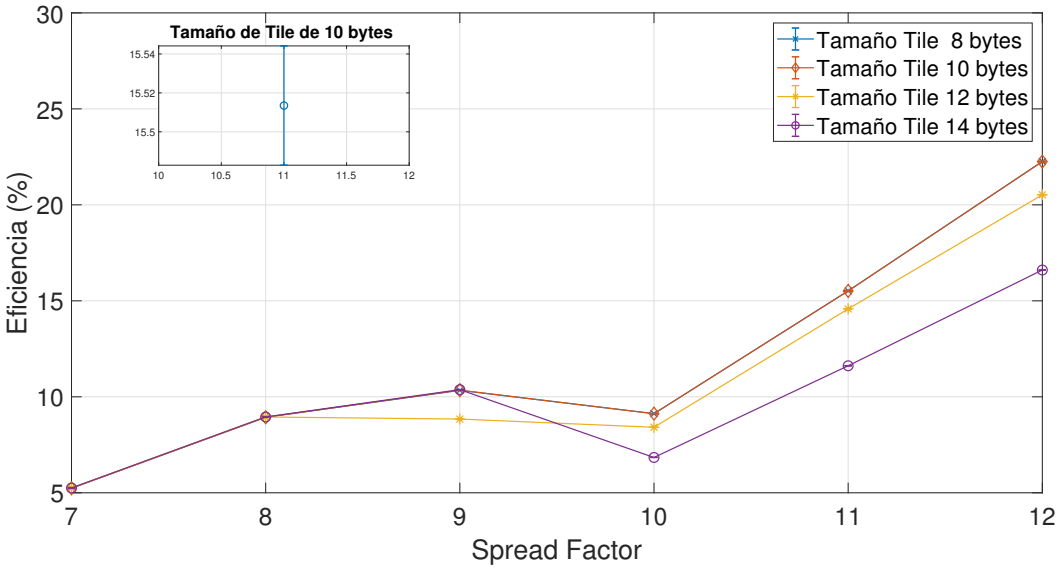


Figura 6.8: Eficiencia Empírica vs *spreading factor* [elaboración propia]

6.2.6. Discusión - Eficiencia vs spreading factor

La tabla 6.6 presenta los valores para la eficiencia empírica y la eficiencia teórica para cada *spreading factor* utilizando un tamaño de *tile* de 10 bytes. La raíz del error cuadrático medio o RMSE es de $2,34\,\%$. Los valores para la eficiencia empírica son menores que la eficiencia teórica indicando que el modelo es optimista respecto a la realidad.

spreading factor	Eficiencia Teórica	Eficiencia Empírica
12	25,09 %	22,25 %
11	18,93 %	15,51 %
10	10,27 %	9,12 %
9	12,23 %	10,35 %
8	11,24 %	8,93 %
7	6,96 %	5,24 %

Tabla 6.6: Modelo teórico y empírico para la eficiencia vs *spreading factor* para un tamaño de *tile* de 10 bytes [elaboración propia]

6.2.7. Resultados - Eficiencia vs Tamaño de Tile

Cálculo Teórico

Para obtener la eficiencia en función del tamaño de *tile* se utiliza el modelo definido en la sección 4.2.1 que entrega los tiempos de cada grupo de mensajes o bloques, junto a la ecuación 4.23 para el cálculo de la eficiencia. Se utiliza una probabilidad $p = 0$ dado que es un modelo sin pérdidas de paquetes.

El gráfico de la figura 6.9 muestra la eficiencia promedio en función del tamaño de *tile* en un enlace de subida *LoRaWAN*^{TM} utilizando el protocolo *SCHC*. Cada curva en este gráfico representa un *spreading factor* distinto. Se evalúan los *spreading factors* SF12, SF11, SF10, SF9, SF8 y SF7.

Para todos los tamaños de *tiles*, la mayor eficiencia es para el *spreading factor* 12 y la menor eficiencia para el *spreading factor* 7.

Cálculo Empírico

El cálculo empírico de la eficiencia promedio en función del tamaño de *tile* se realiza a partir de la definición entregada en la ecuación 4.23. Los tiempos para esta ecuación son obtenidos a través del experimento explicado en la sección 6.2.2 y el código en Matlab mostrado en el Anexo E.

El resultado del experimento es ilustrado en la figura 6.10, donde se muestra la relación entre eficiencia empírica promedio y tamaño de *tile* para cada *spreading factor* a través de distintas curvas.

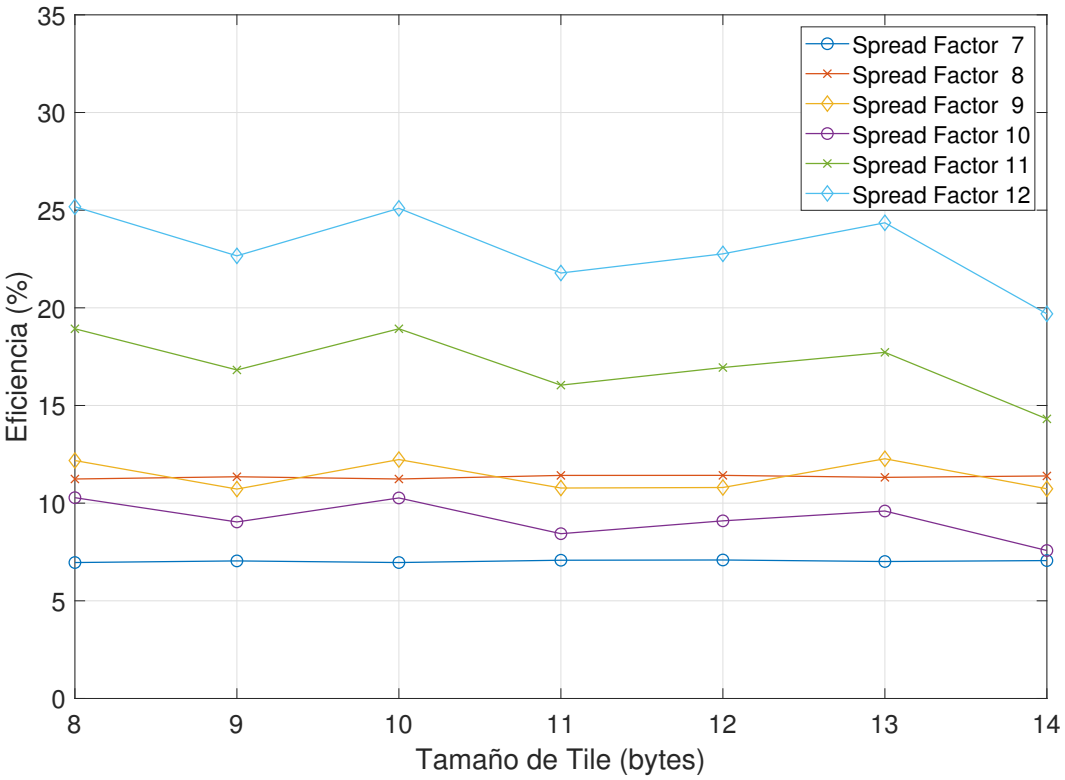


Figura 6.9: Eficiencia Teórica vs Tamaño de *Tile* [elaboración propia]

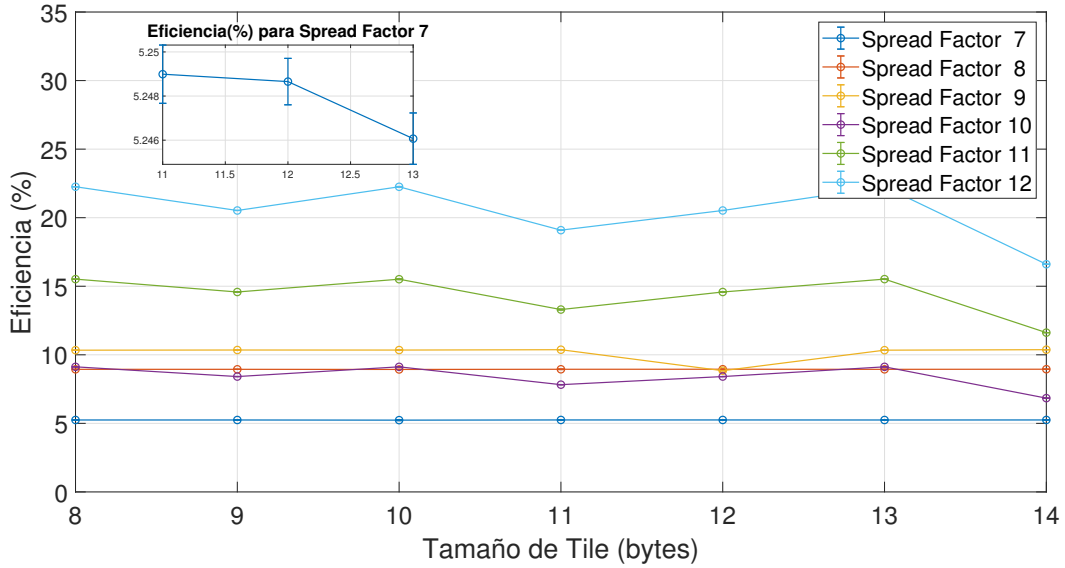


Figura 6.10: Eficiencia Empírica vs tamaño de *tile* [elaboración propia]

6.2.8. Discusión - Eficiencia vs Tamaño de *tile*

Cada curva de los gráficos de la figura 6.9 y 6.10, corresponde a un *spreading factor* distinto. Para cada uno de ellos se muestra una mínima variación de la eficiencia, enfatizada en el *spreading factor* 12 en el cual la variación es mayor que en los otros *spreading factors*. La variación de eficiencia para el *spreading factor* 12 es mostrada en la figura 6.11.

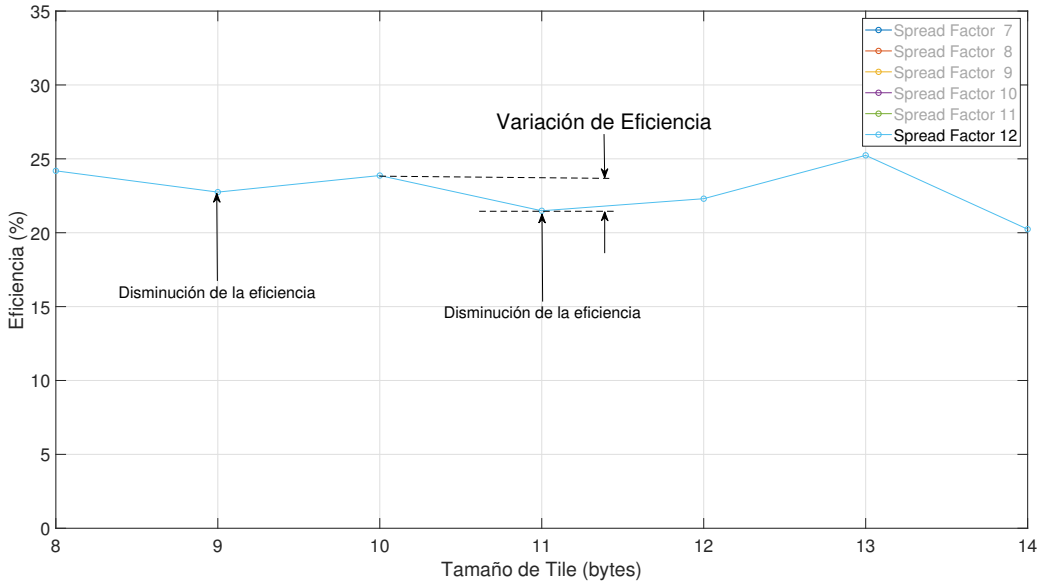


Figura 6.11: Variación en la eficiencia para el spreading factor 12 [elaboración propia]

La eficiencia tiene directa relación con el tamaño temporal de una ventana de transmisión (Ver parámetro T_1 en la ecuación 4.11). El valor de T_1 es directamente proporcional a la cantidad de mensajes *Regular SCHC fragment* enviados. La figura 6.12 muestra el número de mensajes *Regular SCHC fragment* necesarios para fragmentar un *SCHC packet* de 400 *bytes* en función del tamaño de *tile* para diferentes *spreading factor*.

Para comprender mejor lo explicado en el párrafo anterior, la tabla 6.7 muestra el número de mensajes *Regular SCHC fragment* necesarios para fragmentar un *SCHC packet* de 400 *bytes* utilizando un *spreading factor* 12.

La columna **"n° de SCHC fragments"** muestra la cantidad de mensajes necesarios para dividir un *SCHC packet*. Este valor es obtenido al dividir el **"n° de *tiles* por *SCHC packet*"** menos uno, y el n° **de *tiles* en un *SCHC fragment***. En este cálculo es necesario restar uno al n° **de *tiles* por *SCHC packet*** porque el ultimo *tile* se envía en un *All-1 SCHC Fragment* y no debe ser considerado en el cálculo de mensajes *SCHC fragment*. La ecuación 6.1 muestra este cálculo.

$$n^\circ \text{ de SCHC fragments} = \text{ceil}\left(\frac{(n^\circ \text{ de tiles por } \textit{SCHC packet}) - 1}{n^\circ \text{ de tiles en un } \textit{SCHC fragment}}\right) \tag{6.1}$$

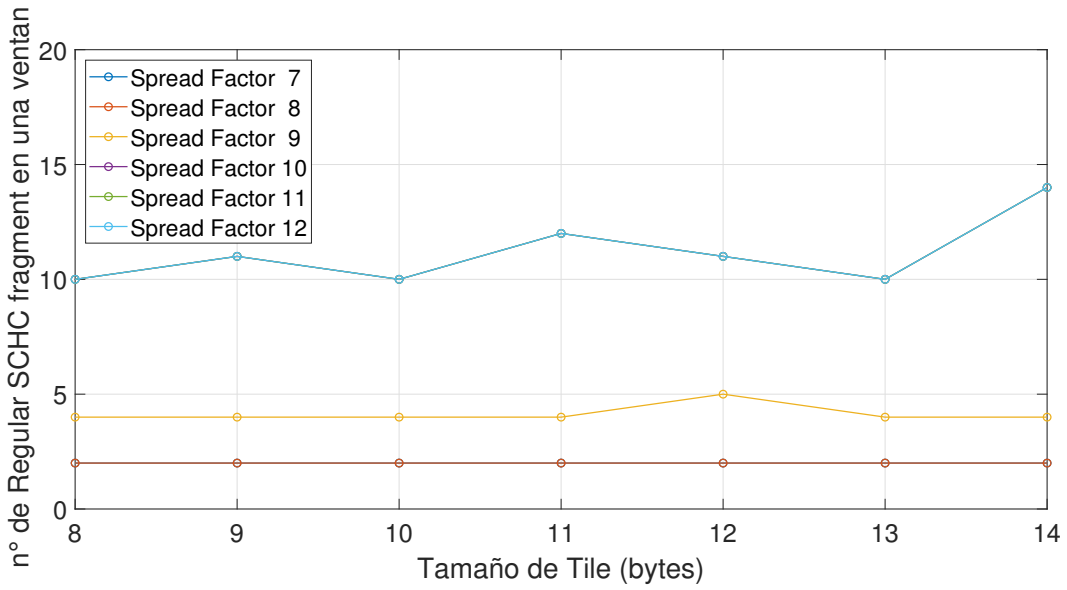


Figura 6.12: Cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia]

En la ecuación 6.1, el **"$n°$ de *tiles* por *SCHC packet*"** puede ser considerado como una función g que depende del tamaño del *tile*. Asi mismo, el **"$n°$ de *tiles* en un *SCHC fragment*"** puede ser considerado como una funcion f que también depende del tamaño del *tile*. Renombrando la variable tamaño del *tile* por x, la ecuación 6.1 puede ser reescrita como:

$$n° \text{ de SCHC fragments} = \text{ceil}\left(\frac{g(x)-1}{f(x)}\right) \tag{6.2}$$

Dado que la función *ceil* es usada solo para obtener la cantidad "entera"de *SCHC fragments*, *ceil* puede ser omitida en la ecuación 6.2 y así los cálculos para obtener el **"$n°$ de SCHC fragments"** quedan con valores decimales.

Para poder cumplir que el **"$n°$ de SCHC fragments"** sea independiente del tamaño del

Tamaño *Tile* (bytes)	**$n°$ *tiles* por SCHC packet**	**$n°$ *tiles* en un SCHC fragment**	**$n°$ de SCHC fragments**
8	50	5	10
9	45	4	11
10	40	4	10
11	37	3	12
12	34	3	11
13	31	3	10
14	29	2	14

Tabla 6.7: Cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* para un *SCHC packet* de 400 *bytes* usando el *spreading factor* 12 [elaboración propia]

tile, entonces su derivada en función del tamaño del *tile* debe ser cero. Por lo tanto, para que la eficiencia con un *spreading factor* dado sea independiente del tamaño del *tile*, se debe cumplir que:

$$\frac{\partial(n^\circ \text{ de SCHC fragments})}{\partial x} = \frac{\partial\left(\frac{g(x)-1}{f(x)}\right)}{\partial x} = 0 \tag{6.3}$$

Para evaluar la condición de la ecuación 6.3 se necesita encontrar las curvas que se ajustan a los puntos de la figura 6.12 para cada uno de los *spreading factors*.

La figura 6.13 muestra las curvas polinómicas que se ajustan a los punto de la figura 6.12. La figura 6.14 muestra la derivada de las curvas polinómicas de la figura 6.13. Se observa que la derivada para los *spreading factors* 7, 8 y 9 es cercana a cero para todos los tamaños de *tile*. Este resultado coincide con la figura 6.11 donde los *spreading factor* 7, 8 y 9 mantienen constante su eficiencia independiente del tamaño de *tile*.

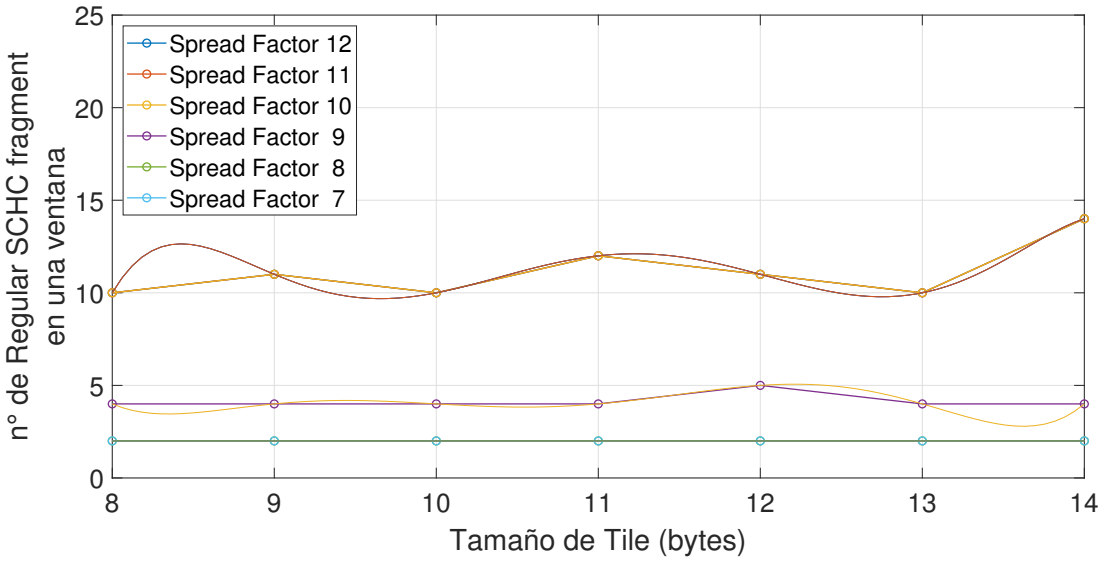


Figura 6.13: Curvas polinómicas que se ajustan a la cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia]

6.3. Resumen de resultados

A nivel de compresión del protocolo *SCHC* se demostró que la elección de la Regla *SCHC* no solo se debe basar en el porcentaje de compresión de la regla. El *Time on Air (ToA)* no tiene un comportamiento lineal respecto a las variaciones en la compresión. Se detectan 2 segmentos denominados CTS (por *Constant Time on air Segment*) y CTRS (por *Constant Time on air Range Segment*). Dentro de estos segmentos el *ToA* no varia en función del

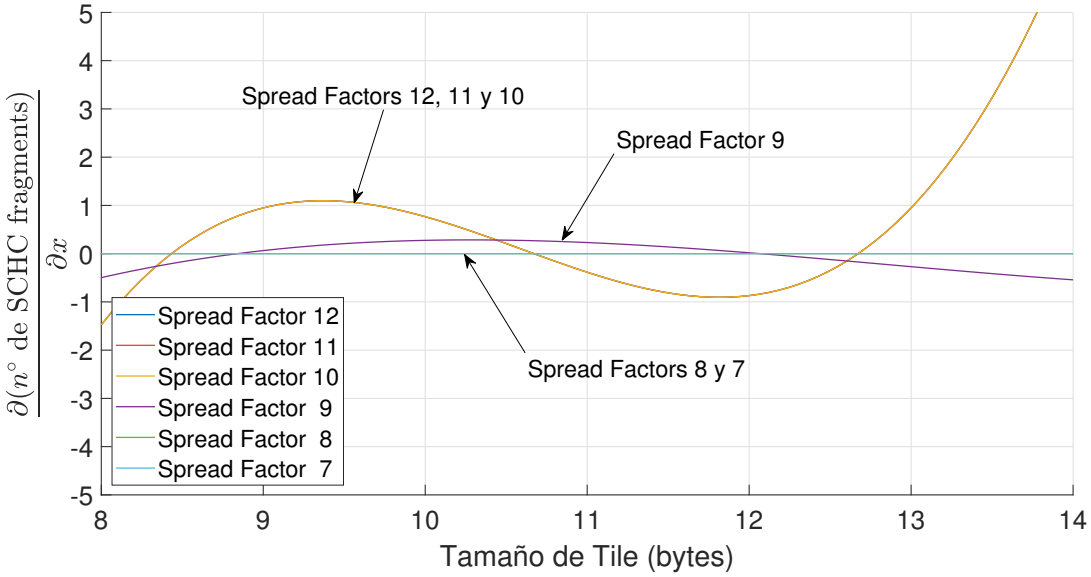


Figura 6.14: Derivada de la cantidad enviada de mensajes *SCHC fragment* en función del tamaño de *tile* [elaboración propia]

porcentaje de compresión lo que permite escoger reglas de menor compresión sin afectar al retardo introducido por el *ToA*.

Se establece también que las reglas de menor compresión son útiles cuando el paquete tiene encabezados que no se pueden comprimir debido a que sus valores son indeterminados (ejemplo, direcciones *IPv6* o puertos *UDP*) esta versatilidad se ve afectada por el aumento del *ToA*.

A nivel de fragmentación, respecto al tamaño de *tile* usado, el trabajo demostró que para *spreading factors* más bajos (cercanos a 7), la eficiencia es independiente del tamaño del *tile*. Para *spreading factors* mayores (cercanos a 12), la eficiencia varía entorno a un valor promedio a medida que se modifica el tamaño de *tile* (ver figura 6.11). Lo anterior queda demostrado tanto para la eficiencia teórica como empírica.

Para el caso de los spreading factor 12, 11 y 10, la eficiencia depende del tamaño del *tile*, como se dijo en el párrafo anterior, y esto es debido a que la derivada de la ecuación 6.1 no es igual a cero como se observa en la figura 6.14.

Capítulo 7

Conclusiones y Trabajo a futuro

7.1. Conclusiones

Esta tesis presentó un estudio cuantitativo del desempeño del protocolo *SCHC* sobre la tecnología LoRaWAN™para el canal de subida o *uplink*. Se propuso un modelo teórico para el cálculo de la eficiencia en la capa de fragmentación. Para ello se realizó un *setting* experimental que se contrastó con el modelo teórico. El *setting* incluyó el desarrollo completo del estándar SCHC para la capa de compresión y el desarrollo del estándar para el enlace de subida en el caso de la capa de fragmentación.

En cuanto a la fragmentación, se demostró que la hipótesis de una **relación lineal e inversamente proporcional entre el *spreading factor* y la eficiencia del protocolo SCHC** es falsa. De hecho, se demostró que la eficiencia y el *spreading factor* no tienen una relación lineal pero sí hay una dependencia que indica que a mayor spreading factor hay una mayor eficiencia. El caso excepcional es para el spreading factor 10 el cual no muestra la misma relación.

Desde el punto de vista de la utilidad, el trabajo a nivel de compresión permite diseñar reglas *SCHC* que estén dentro de los rangos de compresión que mantienen constante al *ToA* evitando realizar un *trade-off* entre versatilidad y aumento de la latencia (que está directamente relacionado con el *Time on Air*). A nivel de fragmentación el modelo permite predecir la eficiencia que se puede obtener cuando se modifica algún parámetro LoRaWAN como el *spreading factor* y además confirma que la elección del tamaño del *tile* no tiene mayor incidencia en el valor de la eficiencia del protocolo.

7.2. Trabajos Futuros

Esta tesis deja abierta algunas líneas de investigación relacionadas con la eficiencia de *SCHC* en ambientes LoRaWAN™.

7.2.1. Eficiencia para enlace de bajada en LoRaWAN

Esta tesis realizó el estudio sobre el canal de subida utilizando el modo Ack-On-Error del protocolo SCHC. Lo anterior deja pendiente la investigación de la eficiencia para el enlace de bajada o *downlink* utilizando el modo *Ack-Always* sobre LoRaWAN™.

Para ello se debe desarrollar el modelo como el presentado en el capítulo 4.2.2 pero considerando los mensajes involucrados en el modo Ack-Always. Posterior a la construcción del modelo, se debe desarrollar del código para la fragmentación por el lado del *SCHC Gateway* y desfragmentación del *SCHC packet* por el lado del *end-device*. Puede ser utilizada la misma plataforma donde se desplegó *PySCHC*. Además el código de *PySCHC* puede ser extendido para crear las clases y métodos necesarios que implementen la máquina de estado para el modo de operación ACK-Always.

7.2.2. Eficiencia Empírica en ambiente con pérdidas

El modelo para el cálculo de la eficiencia teórica presentado en la sección 4.2.2 permite la simulación mostrada en la sección 6.2.1 la cual considera un escenario con pérdidas de mensajes *SCHC fragment*. Así se puede evaluar teóricamente la eficiencia en función de la probabilidad de error de un SCHC fragment. Por lo tanto, queda abierta la realización del experimento para el cálculo de la eficiencia en un escenario con pérdida de *SCHC fragments* y su correspondiente comparación con el resultado obtenido del modelo teórico presentado en la sección 6.2.1. Se puede utilizar el mismo código PySCHC de esta tesis sobre un experimento con alta probabilidad de error.

7.2.3. Implementación completa del estándar SCHC

Este trabajo consideró solo el desarrollo de las máquinas de estado del estandar SCHC para el enlace de subida sobre tecnologia LoRaWAN™. Esto quiere decir que PySCHC considera la etapa de fragmentación en el *end-device* y la etapa de desfragmentación en el *Application Server*. Queda pendiente, la implementación del enlace de bajada a través del modo Ack-Always, ademas de la subcapa de compresión.

7.2.4. Estándar SCHC en modo *Fog Computing*

El estándar *SCHC* considera que el *SCHC Gateway* debe estar instalado en un *Application Server* (ver figura 2.1), esto quiere decir que debe estar posterior al *Network Server*. El *Fog Computing* es un modelo en el cual el procesamiento de los datos y las aplicaciones se concentran en los dispositivos al borde de la red, en lugar de completamente en la nube. De tal manera que los datos puedan ser procesados localmente en un dispositivo inteligente en lugar de ser enviados a la nube para ello. Así, queda pendiente presentar una mejora al modelo actual del estándar, implementando el *SCHC Gateway* en el *Gateway LoRaWAN* y no en el *Application Server*. De esta manera se fomenta el concepto *Fog Computing*.

Anexos

Apéndice A

Código para el cálculo de Eficiencia vs Probabilidad de error

```
clear variables
close all

packet_size = 400;  % bytes
sf = [7 8 9 10 11 12];
[c,d] = size(sf);
rd1 = 1000;              % miliseconds
rd2 = rd1 + 1000;        % miliseconds
 tile_size  = 10*8;      % bits
bw = 125;
w_length = 2;            % in bits (M)
fcn_length = 6;          % in bits (N)
max_ack_requests = 10;
ws = (2.^fcn_length)-1;
error_probability = 0;
error_probability = 0:0.1:0.9;    % Fragment error probability
[a,b] = size(error_probability);
n_simulations = 100000;


%% Starting Montecarlo simulation...

for z=1:d %iteracion de spread factors
    for y=1:b %iteracion de probabilidades
        for x=1:n_simulations %iteracion de simulaciones
            [n_tiles_x_normal_fragment, n_tiles_x_special_fragment , n_normal_fragments_x_window,
    n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
    get__n_fragments_x_schc_packet(sf(z), packet_size, tile_size , ws);
            [errors_x_schc_packet] = generate_random_inputs(error_probability(y));

            % Calculating the time of each block T1, T2 y T3
            tf = toa__schc_regular_fragment(n_tiles_x_normal_fragment* tile_size , sf(z), bw, w_length,
    fcn_length);
            tfe = toa__schc_regular_fragment( n_tiles_x_special_fragment * tile_size , sf(z), bw, w_length,
    fcn_length);

```

```
33              t_all_1  = toa__schc_all_1_fragment( tile_size , sf(z), bw, w_length, fcn_length);
34              t_ack_req = toa__schc_ack_request_fragment(0, sf(z), bw, w_length, fcn_length);
35
36              t1 = (n_normal_fragments_x_window)*(tf + rd2) + (tfe + rd2) + t_all_1 + rd2 + t_ack_req +
      rd1;
37
38
39              if errors_x_schc_packet == 0
40                  t2 = 0;
41                  t2_sum = 0;
42              else
43                  t2 = (tf + rd2) + (2*t_ack_req + rd2 + rd1);
44                  t2_sum = errors_x_schc_packet*t2;
45              end
46
47              t_total = t1 + t2_sum;
48              n_tiles = n_tiles_x_schc_packet;
49              p = error_probability(y);
50              R = get_physical_bit_rate(get_dr(sf(z)));
51              t1_seg = t1/1000;
52              t2_seg = t2/1000;
53              efficiency(x) = (n_tiles * tile_size )/((t1_seg + t2_seg*(p/(1-p)))*R);
54              %efficiency(x) = (t1_seg + t2_seg*(p/(1-p)))*R;
55
56              %efficiency(x) = tf/t_total;
57          end
58          efficiency_mean(y,z) = mean(efficiency);
59          efficiency_std(y,z) = std(efficiency);
60      end
61  end
62
63  for z=1:d
64      errorbar(error_probability*100, efficiency_mean(:,z)*100, efficiency_std(:,z)*100, 'x-','LineWidth'
      ,1)
65      grid on
66      xlabel('SCHC_Fragment_Error_Probability_(%)','FontSize',25)
67      ylabel('Efficiency_(%)','FontSize',25)
68      ax = gca;
69      ax.FontSize = 20;
70
71      legend_vector(z,:) = sprintf('Spread_Factor_%2d',sf(z));
72      hold on
73  end
74
75  legend(legend_vector,'Location','northeast');
76
77
78
79
80
81
82  function [errors_x_schc_packet] = generate_random_inputs(error_probability)
83
84      % Generating error for the first SCHC fragment in the first window
85      x=1;
86      while true
```

```
87          if rand() > error_probability
88              random_inputs_vector(1,x)=1;
89              break;
90          else
91              random_inputs_vector(1,x)=0;
92          end
93          x=x+1;
94      end
95
96      errors_x_schc_packet  = sum(random_inputs_vector(1,:)==0);
97
98
99      % Generating the random input vector for others windows (not the first)
100     errors_x_window(1) = sum(random_inputs_vector(1,:)==0);
101     if errors_x_window(1) ~= 0
102         n=2;
103         while true
104             for z=1:errors_x_window(n-1)
105                 if rand() > error_probability
106                     random_inputs_vector(n,z)=1;
107                 else
108                     random_inputs_vector(n,z)=0;
109                 end
110             end
111
112
113             if errors_x_window(1) == 0
114
115             end
116
117             if errors_x_window(n-1) == sum(random_inputs_vector(n,:)==1)
118                 break;
119             end
120             errors_x_window(n) = errors_x_window(n-1)-sum(random_inputs_vector(n,:)==1);
121             n=n+1;
122         end
123     end
124 end
125
126
127
128 function [n_tiles_x_normal_fragment,  n_tiles_x_special_fragment ,  n_normal_fragments_x_window,
        n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
        get__n_fragments_x_schc_packet(sf, packet_size,  tile_size ,  ws)
129     schc_regular_fragment_payload =  get_schc_payload_size_available(get_dr(sf));      % in bits (see 2.6.6
          https://lora-alliance.org/sites/default/files/2018-04/lorawantm_regional_parameters_v1.1rb_-
        _final.pdf)
130
131     n_tiles_x_schc_packet  = ceil((packet_size*8)./ tile_size );
132     if  n_tiles_x_schc_packet  <= 63
133         n_tiles_x_normal_fragment = floor(schc_regular_fragment_payload./ tile_size );     % bits/bits
134         n_normal_fragments_x_window = floor((n_tiles_x_schc_packet-1)./n_tiles_x_normal_fragment);     %
          recordar que el ultimo  tile  va en un SCHC All-1. Por eso se resta 1 tile
135          n_tiles_x_special_fragment  = mod((n_tiles_x_schc_packet-1), n_tiles_x_normal_fragment);     % bits
        /bits
136
```

```
137             if( n_tiles_x_special_fragment  == 0)
138                 n_special_fragments_x_window = 0;
139             else
140                 n_special_fragments_x_window = 1;
141             end
142
143             n_fragments_x_window = n_normal_fragments_x_window + n_special_fragments_x_window;
144
145         else
146             print("ERROR: the package does not reach inside a SCHC window")
147         end
148 end
149
150 function [t_packet] = toa__schc_regular_fragment(payload, sf, bw, w_length, fcn_length)
151         % payload in bits
152
153         schc_fragment_size = w_length + fcn_length + payload;
154
155
156         mhdr = 8;
157         devaddr = 32;
158         fctrl  = 8;
159         fcnt  = 16;
160         fopts  = 8;
161         fport  = 8;
162         frmpayload = schc_fragment_size;
163
164         lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
165         lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
166
167         n_preamble = 8;
168
169         %%
170         % $T_{sym}=\frac{2^{SF}}{BW}$
171         t_sym = (2^sf)/bw;
172
173
174         %% Preamble Time
175         % $T_{preamble} = (n_{preamble}+4,25) \cdot T_{sym}$
176         t_preamble = (n_preamble + 4.25)*t_sym;
177
178
179         %% Payload Time
180         pl  = lorawan_msg_size_bytes;
181         crc  = 1;
182         h = 0;
183         de = 1;
184         cr  = 1;
185
186         c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
187         t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
188
189         %% Packet Time
190         % $T_{packet}=T_{preamble}+T_{payload}$
191         t_packet  = t_preamble + t_payload;
192 end
```

```
193
194 function [t_packet] = toa__schc_all_1_fragment(payload, sf, bw, w_length, fcn_length)
195     % payload in bits
196
197     rcs = 32;
198     schc_all_1_size = w_length + fcn_length + rcs + payload;
199
200     mhdr = 8;
201     devaddr = 32;
202     fctrl = 8;
203     fcnt = 16;
204     fopts = 8;
205     fport = 8;
206     frmpayload = schc_all_1_size;
207
208     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
209     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
210
211
212     n_preamble = 8;
213
214     %%
215     % $T_{sym}=\frac{2^{SF}}{BW}$
216     t_sym = (2^sf)/bw;
217
218
219     %% Preamble Time
220     % $T_{preamble} = (n_{preamble}+4,25) \cdot T_{sym}$
221     t_preamble = (n_preamble + 4.25)*t_sym;
222
223
224     %% Payload Time
225     pl = lorawan_msg_size_bytes;      % Payload bytes (see the packet frame)
226     crc = 1;
227     h = 0;
228     de = 0;
229     cr = 1;
230
231     c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
232     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
233
234     %% Packet Time
235     % $T_{packet}=T_{preamble}+T_{payload}$
236     t_packet = t_preamble + t_payload;
237 end
238
239 function [t_packet] = toa__schc_ack_request_fragment(payload, sf, bw, w_length, fcn_length)
240     schc_ack_req_size = w_length + fcn_length;
241
242     mhdr = 8;
243     devaddr = 32;
244     fctrl = 8;
245     fcnt = 16;
246     fopts = 8;
247     fport = 8;
248     frmpayload = schc_ack_req_size;
```

```
249
250     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
251     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
252
253     n_preamble = 8;
254
255     %%
256     % $T_{sym}=\frac{2^{SF}}{BW}$
257     t_sym = (2^sf)/bw;
258
259
260     %% Preamble Time
261     % $T_{preamble} = (n_{preamble}+4,25) \cdot T_{sym}$
262     t_preamble = (n_preamble + 4.25)*t_sym;
263
264
265     %% Payload Time
266     pl = lorawan_msg_size_bytes;      % Payload bytes (see the packet frame)
267     crc = 1;
268     h = 0;
269     de = 0;
270     cr = 1;
271
272     c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
273     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
274
275     %% Packet Time
276     % $T_{packet}=T_{preamble}+T_{payload}$
277     t_packet = t_preamble + t_payload;
278
279
280 end
281
282 function [sf] = get_sf(dr)
283     dr_vector = [0 1 2 3 4 5];
284     sf_vector = [12 11 10 9 8 7];
285     sf = sf_vector(find(dr_vector == dr));
286 end
287
288 function [dr] = get_dr(sf)
289     dr_vector = [0 1 2 3 4 5];
290     sf_vector = [12 11 10 9 8 7];
291     dr = dr_vector(find(sf_vector == sf));
292 end
293
294 function [SCHC_payload_size_available] = get_schc_payload_size_available(dr)
295     MACPayload_size_vector = [51 51 51 115 222 222 222]*8;
296     FHDR_size = 64; % in bits
297     FPort_size = 8; % in bits
298     FRMPayload_size_vector = MACPayload_size_vector - FHDR_size - FPort_size; % bits
299     dr_vector = [0 1 2 3 4 5 6];
300     SCHC_payload_size_available = FRMPayload_size_vector(find(dr_vector == dr));    % bits
301 end
302
303 function [pbr] = get_physical_bit_rate(dr)
304     dr_vector = [0 1 2 3 4 5];
```

```
305     pbr_vector = [250 440 980 1760 3125 5470];
306     pbr = pbr_vector(find(dr_vector == dr));
307 end
```

Apéndice B

Código para el cálculo de Eficiencia Teórica vs Spread Factor

```
1
2  clear variables
3  close all
4
5  packet_size = 400;
6
7
8  % The following parameters are recommended values for the AU915-928 band.
9  rd1 = 1000;              % miliseconds
10 rd2 = rd1 + 1000;        % miliseconds
11
12
13 % The following parameters are recommended values for the SCHC Regular Fragment
14  tile_size = (8:2:14)*8; % bits
15 sf = [7 8 9 10 11 12];
16 bw = 125;
17 w_length = 2;            % in bits (M)
18 fcn_length = 6;          % in bits (N)
19 max_ack_requests = 10;
20 ws = (2.^fcn_length)-1;
21
22
23 for r=1:length(sf)
24     schc_regular_fragment_payload = get_schc_payload_size_available(get_dr(sf(r)));
25
26     for k=1:length(tile_size)
27         [n_tiles_x_normal_fragment, n_tiles_x_special_fragment , n_normal_fragments_x_window,
       n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
       get__n_fragments_x_schc_packet(sf(r), packet_size(1), tile_size (k), ws);
28
29         % Calculating the time of each block T1, T2 y T3
30         tf = toa__schc_regular_fragment(n_tiles_x_normal_fragment* tile_size (k), sf(r), bw, w_length,
       fcn_length);
31         tfe = toa__schc_regular_fragment( n_tiles_x_special_fragment * tile_size (k), sf(r), bw, w_length,
       fcn_length);
32
```

```
33            t_all_1  = toa__schc_all_1_fragment( tile_size (k), sf(r), bw, w_length, fcn_length);
34            t_ack_req = toa__schc_ack_request_fragment(0, sf(r), bw, w_length, fcn_length);
35
36            t1 = tf + rd2 + (n_normal_fragments_x_window-1)*(tf + rd2) + (tfe + rd2) + t_all_1 + rd2 +
      t_ack_req + rd1;
37
38
39            n_tiles  = n_tiles_x_schc_packet ;
40            p = 0;
41            R = get_physical_bit_rate(get_dr(sf(r)));
42            t1_seg = t1/1000;
43            t2_seg = 0;
44            efficiency (k,r) = (n_tiles * tile_size (k))/((t1_seg + t2_seg*(p/(1-p)))*R);
45        end
46    end
47
48
49    aux = zeros(length(tile_size),length(sf));
50    for i=1:length(tile_size)
51        legend_vector(i ,:) = sprintf('Tile_Size_%2d', tile_size (i)/8);
52    end
53
54
55    figure
56    hax1 = axes();
57    plot(sf, efficiency *100,'-o','LineWidth',1);
58    xticks ([7  8 9 10 11 12])
59    xlim([7  12])
60    xlabel('Spread_Factor','FontSize',25)
61    ylabel('Efficiency_(%)','FontSize',25)
62    legend(legend_vector,'Location','northwest');
63    grid on
64    ax = gca;
65    ax.FontSize = 20;
66
67
68
69    function [t_packet] = toa__schc_regular_fragment(payload, sf, bw, w_length, fcn_length)
70        % payload in bits
71
72        schc_fragment_size = w_length + fcn_length + payload;
73
74        mhdr = 8;
75        devaddr = 32;
76        fctrl = 8;
77        fcnt = 16;
78        fopts = 8;
79        fport = 8;
80        frmpayload = schc_fragment_size;
81
82        lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
83        lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
84
85        n_preamble = 8;
86
87        t_sym = (2^sf)/bw;
```

```
88
89      t_preamble = (n_preamble + 4.25)*t_sym;
90
91
92      %% Payload Time
93      pl = lorawan_msg_size_bytes;
94      crc = 1;
95      h = 0;
96      de = 1;
97      cr = 1;
98
99      c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
100     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
101
102     %% Packet Time
103     t_packet = t_preamble + t_payload;
104
105
106 end
107
108 function [t_packet] = toa__schc_all_1_fragment(payload, sf, bw, w_length, fcn_length)
109     % payload in bits
110     rcs = 32;
111     schc_all_1_size = w_length + fcn_length + rcs + payload;
112
113     mhdr = 8;
114     devaddr = 32;
115     fctrl = 8;
116     fcnt = 16;
117     fopts = 8;
118     fport = 8;
119     frmpayload = schc_all_1_size;
120
121     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
122     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
123
124     n_preamble = 8;
125
126     t_sym = (2^sf)/bw;
127
128     t_preamble = (n_preamble + 4.25)*t_sym;
129
130
131     %% Payload Time
132     pl = lorawan_msg_size_bytes;
133     crc = 1;
134     h = 0;
135     de = 0;
136     cr = 1;
137
138     c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
139     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
140
141     %% Packet Time
142     t_packet = t_preamble + t_payload;
143
```

```
144
145 end
146
147 function [t_packet] = toa__schc_ack_request_fragment(payload, sf, bw, w_length, fcn_length)
148     % payload in bits
149
150     schc_ack_req_size  = w_length + fcn_length;
151
152     mhdr = 8;
153     devaddr = 32;
154     fctrl  = 8;
155     fcnt  = 16;
156     fopts  = 8;
157     fport  = 8;
158     frmpayload = schc_ack_req_size;
159
160     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
161     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
162
163     n_preamble = 8;
164
165     t_sym = (2^sf)/bw;
166
167
168     %% Preamble Time
169     t_preamble = (n_preamble + 4.25)*t_sym;
170
171
172     %% Payload Time
173     pl  = lorawan_msg_size_bytes;
174     crc  = 1;
175     h = 0;
176     de = 0;
177     cr  = 1;
178
179     c  = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
180     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
181
182     %% Packet Time
183     t_packet  = t_preamble + t_payload;
184
185
186 end
187
188
189 function [sf]  = get_sf(dr)
190     dr_vector  = [5 4 3 2 1  0];
191     sf_vector  = [7 8 9 10 11  12];
192     sf  = sf_vector(find(dr_vector  == dr));
193 end
194
195 function [dr] = get_dr(sf)
196     dr_vector  = [5 4 3 2 1  0];
197     sf_vector  = [7 8 9 10 11  12];
198     dr = dr_vector(find(sf_vector  == sf));
199 end
```

```
200
201 function [n_tiles_x_normal_fragment,  n_tiles_x_special_fragment ,  n_normal_fragments_x_window,
      n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
      get__n_fragments_x_schc_packet(sf, packet_size ,   tile_size  , ws)
202     schc_regular_fragment_payload = get_schc_payload_size_available(get_dr(sf));      % in bits (see 2.6.6
        https://lora-alliance.org/sites/default/files/2018-04/lorawantm_regional_parameters_v1.1rb_-
        _final.pdf)
203
204     n_tiles_x_schc_packet  = ceil((packet_size*8)./ tile_size );
205     if  n_tiles_x_schc_packet  <= 63
206         n_tiles_x_normal_fragment = floor(schc_regular_fragment_payload./ tile_size );     % bits/bits
207         n_normal_fragments_x_window = floor((n_tiles_x_schc_packet-1)./n_tiles_x_normal_fragment);     %
        recordar que el ultimo  tile  va en un SCHC All-1. Por eso se resta 1 tile
208          n_tiles_x_special_fragment  = mod((n_tiles_x_schc_packet-1), n_tiles_x_normal_fragment);     % bits
      /bits
209
210         if( n_tiles_x_special_fragment  == 0)
211             n_special_fragments_x_window = 0;
212         else
213             n_special_fragments_x_window = 1;
214         end
215
216         n_fragments_x_window = n_normal_fragments_x_window + n_special_fragments_x_window;
217
218     else
219         print("ERROR: the package does not reach inside a SCHC window")
220     end
221 end
222
223 function [SCHC_payload_size_available] = get_schc_payload_size_available(dr)
224     MACPayload_size_vector = [222 222 115 51 51 51]*8;
225     FHDR_size = 64;  % in bits
226     FPort_size = 8;  % in bits
227     FRMPayload_size_vector = MACPayload_size_vector - FHDR_size - FPort_size;  % bits
228     dr_vector  = [5 4 3 2 1  0];
229     SCHC_header_size = 8;
230     SCHC_payload_size_available = FRMPayload_size_vector(find(dr_vector == dr)) - SCHC_header_size;
        % bits
231 end
232
233 function [pbr] =  get_physical_bit_rate(dr)
234     dr_vector  = [0 1 2 3 4  5];
235     pbr_vector = [250 440 980 1760 3125 5470];
236     pbr = pbr_vector(find(dr_vector == dr));
237 end
```

Apéndice C

Código para el cálculo de Eficiencia Empírica vs Spread Factor

```
clear variables
close all

 tile_size  = (8:2:14);
data_rate = [5 4 3 2 1  0];
[a,b] = size( tile_size );
[c,d] = size(data_rate);
sf = [7 8 9 10 11  12];
packet_size = 400;

m = 1;

for i = 1:length( tile_size )
    for j = 1:length(data_rate)
        str = strcat(num2str(tile_size(i)), '_bytes_x_tile_DR_', num2str(data_rate(j)));
        disp(str);
        cd(str);

        total(m).packet_size = packet_size;
        total(m).data_rate = data_rate(j);
        total(m). tile_size  =  tile_size (i);

        file_name = strcat('capture_', num2str(packet_size), '_bytes_', num2str(tile_size(i)) ,'
    _byte_x_tile_DR_', num2str(data_rate(j)), '.txt');
        str = fileread(file_name);
        values = jsondecode(str);

        total(m).mean_ttfsp = mean_total_time_for_schc_packet(values);
        total(m).mean_ttfw = mean_total_time_for_window(values);
        total(m).mean_tf = mean_time_frame(values);

        total(m).std_ttfsp = std_total_time_for_schc_packet(values);
        total(m).std_ttfw = std_total_time_for_window(values);
        total(m).std_tf = std_time_frame(values);

        total(m).mean_efficiency = mean_efficiency_vector(values, packet_size, data_rate(j));
```

```
36          total(m).std_efficiency = std_efficiency_vector(values, packet_size, data_rate(j));
37
38
39          cd('../');
40          m = m + 1;
41
42      end
43
44  end
45
46  [e,f] = size(total);
47  z = 1;
48  for w = 1:d
49      for v = 1:f
50          if total(v).packet_size == 400 && data_rate(w) == total(v).data_rate
51              %efficiency_vector(w,z) = total(v).mean_tf./total(v).mean_ttfw;
52              %efficiency_error_vector(w,z) = total(v). std_tf./total(v). std_ttfw;
53
54              efficiency_vector(z,w) = total(v).mean_efficiency;
55              efficiency_error_vector(z,w) = total(v). std_efficiency ;
56
57              latency_vector(z,w) = total(v).mean_ttfsp;
58              z = z + 1;
59          end
60      end
61      z = 1;
62  end
63
64  for i=1:length(tile_size)
65      legend_vector(i,:) = sprintf('Tile Size %2d', tile_size(i));
66  end
67
68  for t=1:b
69      errorbar(sf, efficiency_vector(t,:)*100, efficiency_error_vector(t,:)*100,'o-','LineWidth',1)
70      xticks([7 8 9 10 11 12])
71      xlim([7 12])
72      ylim([5 30])
73      xlabel('Spread Factor','FontSize',25)
74      ylabel('Efficiency (%)','FontSize',25)
75      grid on
76      hold on
77      ax = gca;
78      ax.FontSize = 20;
79  end
80
81
82  legend(legend_vector,'Location','northwest');
83
84
85
86  function mean_out = mean_total_time_for_schc_packet(val)
87      [nrow, ncol] = size(val);
88      temp = 0;
89      for k = 1:nrow
90          temp(k) = val(k).total_time_for_schc_packet;
91      end
```

```
92      mean_out = mean(temp);
93  end
94
95  function mean_out = mean_total_time_for_window(val)
96      [nrow, ncol] = size(val);
97      temp = 0;
98      for k = 1:nrow
99          temp(k) = val(k).total_time_for_window;
100     end
101     mean_out = mean(temp);
102 end
103
104 function mean_out = mean_time_frame(val)
105     [nrow, ncol] = size(val);
106     temp = 0;
107     for k = 1:nrow
108         temp(k) = val(k).tf;
109     end
110     mean_out = mean(temp);
111 end
112
113 function mean_out = mean_efficiency_vector(val, packet_size, dr)
114     [nrow, ncol] = size(val);
115     temp = 0;
116     for k = 1:nrow
117         p = 0;
118         R = get_physical_bit_rate(dr);
119         t2_seg = 0;
120         t1_seg = val(k).total_time_for_window/1000;
121         temp(k) = (packet_size*8)/((t1_seg + t2_seg*(p/(1-p)))*R);
122     end
123     mean_out = mean(temp);
124 end
125
126 function std_out = std_total_time_for_schc_packet(val)
127     [nrow, ncol] = size(val);
128     temp = 0;
129     for k = 1:nrow
130         temp(k) = val(k). total_time_for_schc_packet ;
131     end
132     std_out = std(temp);
133 end
134
135 function std_out = std_total_time_for_window(val)
136     [nrow, ncol] = size(val);
137     temp = 0;
138     for k = 1:nrow
139         temp(k) = val(k).total_time_for_window;
140     end
141     std_out = std(temp);
142 end
143
144 function std_out = std_time_frame(val)
145     [nrow, ncol] = size(val);
146     temp = 0;
147     for k = 1:nrow
```

```
148         temp(k) = val(k).tf;
149     end
150     std_out = std(temp);
151 end
152
153 function std_out = std_efficiency_vector(val, packet_size, dr)
154     [nrow, ncol] = size(val);
155     temp = 0;
156     for k = 1:nrow
157         p = 0;
158         R = get_physical_bit_rate(dr);
159         t2_seg = 0;
160         t1_seg = val(k).total_time_for_window/1000;
161         temp(k) = (packet_size*8)/((t1_seg + t2_seg*(p/(1-p)))*R);
162     end
163     std_out = std(temp);
164 end
165
166 function [pbr] = get_physical_bit_rate(dr)
167     dr_vector = [0 1 2 3 4 5];
168     pbr_vector = [250 440 980 1760 3125 5470];
169     pbr = pbr_vector(find(dr_vector == dr));
170 end
```

Apéndice D

Código para el cálculo de Eficiencia Teórica vs Tamaño de Tile

```
clear variables
close all

packet_size = 300;


% The following parameters are recommended values for the AU915-928 band.
rd1 = 1000;                % miliseconds
rd2 = rd1 + 1000;          % miliseconds


% The following parameters are recommended values for the SCHC Regular Fragment
tile_size = (8:1:16)*8;  % bits
sf = [7 8 9 10 11 12];
bw = 125;
w_length = 2;
fcn_length = 6;
max_ack_requests = 10;
ws = (2.^fcn_length)-1;


for r=1:length(sf)
    schc_regular_fragment_payload = get_schc_payload_size_available(get_dr(sf(r)));

    for k=1:length(tile_size)
        [n_tiles_x_normal_fragment, n_tiles_x_special_fragment , n_normal_fragments_x_window,
    n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
    get__n_fragments_x_schc_packet(sf(r), packet_size(1), tile_size(k), ws);

        % Calculating the time of each block T1, T2 y T3
        tf = toa__schc_regular_fragment(n_tiles_x_normal_fragment* tile_size(k), sf(r), bw, w_length,
    fcn_length);
        tfe = toa__schc_regular_fragment( n_tiles_x_special_fragment * tile_size(k), sf(r), bw, w_length,
    fcn_length);

        t_all_1 = toa__schc_all_1_fragment( tile_size(k), sf(r), bw, w_length, fcn_length);
```

```
33          t_ack_req = toa__schc_ack_request_fragment(0, sf(r), bw, w_length, fcn_length);
34
35          t1 = tf + rd2 + (n_normal_fragments_x_window-1)*(tf + rd2) + (tfe + rd2) + t_all_1 + rd2 +
        t_ack_req + rd1;
36
37
38          n_tiles = n_tiles_x_schc_packet;
39          p = 0;
40          R = get_physical_bit_rate(get_dr(sf(r)));
41          t1_seg = t1/1000;
42          t2_seg = 0;
43          efficiency (r,k) = (n_tiles * tile_size (k))/((t1_seg + t2_seg*(p/(1-p)))*R);
44      end
45  end
46
47
48  aux = zeros(length(sf),length(tile_size));
49  for i=1:length(sf)
50      legend_vector(i,:) = sprintf('Spread Factor %2d',sf(i));
51  end
52
53
54  figure
55  hax1 = axes();
56  plot( tile_size ./8, efficiency *100,'-o','LineWidth',1);
57  xlabel('Tamano_de_Tile_(bytes)','FontSize',25)
58  ylabel('Eficiencia (%)','FontSize',25)
59  legend(legend_vector,'Location','northeast');
60  grid on
61  ax = gca;
62  ax.FontSize = 20;
63  ylim([0  35])
64
65
66
67  function [t_packet] = toa__schc_regular_fragment(payload, sf, bw, w_length, fcn_length)
68      % payload in bits
69
70      schc_fragment_size = w_length + fcn_length + payload;
71
72      mhdr = 8;
73      devaddr = 32;
74      fctrl = 8;
75      fcnt = 16;
76      fopts = 8;
77      fport = 8;
78      frmpayload = schc_fragment_size;
79
80      lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
81      lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
82
83      n_preamble = 8;
84
85      t_sym = (2^sf)/bw;
86
87
```

```
88      %% Preamble Time
89      t_preamble = (n_preamble + 4.25)*t_sym;
90
91
92      %% Payload Time
93      pl = lorawan_msg_size_bytes;
94      crc = 1;
95      h = 0;
96      de = 1;
97      cr = 1;
98
99      c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
100     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
101
102     %% Packet Time
103     t_packet = t_preamble + t_payload;
104
105
106 end
107
108 function [t_packet] = toa__schc_all_1_fragment(payload, sf, bw, w_length, fcn_length)
109     % payload in bits
110
111     rcs = 32;
112     schc_all_1_size = w_length + fcn_length + rcs + payload;
113
114     mhdr = 8;
115     devaddr = 32;
116     fctrl = 8;
117     fcnt = 16;
118     fopts = 8;
119     fport = 8;
120     frmpayload = schc_all_1_size;
121
122     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
123     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
124
125
126     %% LoRa Packet (Radio PHY layer)
127     n_preamble = 8;
128
129     %%
130     t_sym = (2^sf)/bw;
131
132
133     %% Preamble Time
134     t_preamble = (n_preamble + 4.25)*t_sym;
135
136
137     %% Payload Time
138     pl = lorawan_msg_size_bytes;
139     crc = 1;
140     h = 0;
141     de = 0;
142     cr = 1;
143
```

```
144     c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
145     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
146
147     %% Packet Time
148     t_packet = t_preamble + t_payload;
149
150
151 end
152
153 function [t_packet] = toa__schc_ack_request_fragment(payload, sf, bw, w_length, fcn_length)
154     % payload in bits
155
156     schc_ack_req_size = w_length + fcn_length;
157
158     mhdr = 8;
159     devaddr = 32;
160     fctrl = 8;
161     fcnt = 16;
162     fopts = 8;
163     fport = 8;
164     frmpayload = schc_ack_req_size;
165
166     lorawan_msg_size = mhdr + devaddr + fctrl + fcnt + fopts + fport + frmpayload;
167     lorawan_msg_size_bytes = ceil(lorawan_msg_size/8);
168
169     n_preamble = 8;
170
171     %%
172     t_sym = (2^sf)/bw;
173
174
175     %% Preamble Time
176     t_preamble = (n_preamble + 4.25)*t_sym;
177
178
179     %% Payload Time
180     pl = lorawan_msg_size_bytes;
181     crc = 1;
182     h = 0;
183     de = 0;
184     cr = 1;
185
186     c = ceil((8*pl-4*sf+28+16*crc-20*h)/(4*(sf-2*de)));
187     t_payload = t_sym*(8 + max([(c*(cr+4)) 0]));
188
189     %% Packet Time
190     t_packet = t_preamble + t_payload;
191
192
193 end
194
195
196 function [sf] = get_sf(dr)
197     dr_vector = [5 4 3 2 1 0];
198     sf_vector = [7 8 9 10 11 12];
199     sf = sf_vector(find(dr_vector == dr));
```

```
end

function [dr] = get_dr(sf)
    dr_vector = [5 4 3 2 1 0];
    sf_vector = [7 8 9 10 11 12];
    dr = dr_vector(find(sf_vector == sf));
end

function [n_tiles_x_normal_fragment, n_tiles_x_special_fragment , n_normal_fragments_x_window,
    n_special_fragments_x_window, n_tiles_x_schc_packet, n_fragments_x_window] =
    get__n_fragments_x_schc_packet(sf, packet_size, tile_size , ws)
    schc_regular_fragment_payload = get_schc_payload_size_available(get_dr(sf));

    n_tiles_x_schc_packet = ceil((packet_size*8)./ tile_size );
    if n_tiles_x_schc_packet <= 63
        n_tiles_x_normal_fragment = floor(schc_regular_fragment_payload./ tile_size );
        n_normal_fragments_x_window = floor((n_tiles_x_schc_packet-1)./n_tiles_x_normal_fragment);
        n_tiles_x_special_fragment = mod((n_tiles_x_schc_packet-1), n_tiles_x_normal_fragment);

        if( n_tiles_x_special_fragment == 0)
            n_special_fragments_x_window = 0;
        else
            n_special_fragments_x_window = 1;
        end

        n_fragments_x_window = n_normal_fragments_x_window + n_special_fragments_x_window;

    else
        print("ERROR: the package does not reach inside a SCHC window")
    end
end

function [SCHC_payload_size_available] = get_schc_payload_size_available(dr)
    MACPayload_size_vector = [222 222 115 51 51 51]*8;
    FHDR_size = 64; % in bits
    FPort_size = 8; % in bits
    FRMPayload_size_vector = MACPayload_size_vector - FHDR_size - FPort_size; % bits
    dr_vector = [5 4 3 2 1 0];
    SCHC_header_size = 8;
    SCHC_payload_size_available = FRMPayload_size_vector(find(dr_vector == dr)) - SCHC_header_size;
      % bits
end


function [pbr] = get_physical_bit_rate(dr)
    dr_vector = [0 1 2 3 4 5];
    pbr_vector = [250 440 980 1760 3125 5470];
    pbr = pbr_vector(find(dr_vector == dr));
end
```

Apéndice E

Código para el cálculo de Eficiencia Empírica vs Tamaño de Tile

```
1
2  clear variables
3  close all
4
5   tile_size   = (8:1:14);
6  data_rate = [5 4 3 2 1  0];
7  [a,b]  = size( tile_size );
8  [c,d]  = size(data_rate);
9  sf  = [7 8 9 10 11  12];
10 packet_size  = 400;
11
12 m = 1;
13
14 for  i  = 1:length( tile_size )
15      for  j  = 1:length(data_rate)
16          str  = strcat(num2str(tile_size(i)),  '_bytes_x_tile_DR_',  num2str(data_rate(j)));
17          disp(str);
18          cd(str);
19
20          total(m).packet_size  = packet_size;
21          total(m).data_rate = data_rate(j);
22          total(m). tile_size   =  tile_size (i);
23
24          file_name  = strcat('capture_',  num2str(packet_size), '_bytes_',  num2str(tile_size(i))  ,'
        _byte_x_tile_DR_',  num2str(data_rate(j)), '.txt');
25          str  =  fileread(file_name);
26          values  = jsondecode(str);
27
28          total(m).mean_ttfsp = mean_total_time_for_schc_packet(values);
29          total(m).mean_ttfw = mean_total_time_for_window(values);
30          total(m).mean_tf = mean_time_frame(values);
31
32          total(m).std_ttfsp  =  std_total_time_for_schc_packet (values);
33          total(m).std_ttfw = std_total_time_for_window(values);
34          total(m).std_tf  = std_time_frame(values);
35
```

```
36          total(m).mean_efficiency = mean_efficiency_vector(values, packet_size, data_rate(j));
37          total(m). std_efficiency = std_efficiency_vector (values, packet_size, data_rate(j));
38
39
40          cd('../');
41          m = m + 1;
42
43      end
44
45  end
46
47  [e,f] = size(total);
48  z = 1;
49  for w = 1:d
50      for v = 1:f
51          if total(v). packet_size == 400 && data_rate(w) == total(v).data_rate
52
53              efficiency_vector (w,z) = total(v). mean_efficiency;
54              efficiency_error_vector (w,z) = total(v). std_efficiency ;
55
56              latency_vector(w,z) = total(v).mean_ttfsp;
57              z = z + 1;
58          end
59      end
60      z = 1;
61  end
62
63
64  for i=1:length(sf)
65      legend_vector(i ,:) = sprintf('Spread_Factor_%2d',sf(i));
66  end
67
68  for t=1:d
69      errorbar(tile_size, efficiency_vector (t ,:) *100, efficiency_error_vector (t ,:) *100,'o-','LineWidth',1)
70      xlabel('Tamano_de_Tile_(bytes)','FontSize',25)
71      ylabel('Eficiencia_(%)','FontSize',25)
72      grid on
73      hold on
74      ylim([0 35])
75      ax = gca;
76      ax.FontSize = 20;
77
78  end
79
80
81  legend(legend_vector,'Location','northeast');
82
83
84  function mean_out = mean_total_time_for_schc_packet(val)
85      [nrow, ncol] = size(val);
86      temp = 0;
87      for k = 1:nrow
88          temp(k) = val(k). total_time_for_schc_packet ;
89      end
90      mean_out = mean(temp);
91  end
```

```
92
93  function mean_out = mean_total_time_for_window(val)
94      [nrow, ncol] = size(val);
95      temp = 0;
96      for k = 1:nrow
97          temp(k) = val(k).total_time_for_window;
98      end
99      mean_out = mean(temp);
100 end
101
102 function mean_out = mean_time_frame(val)
103     [nrow, ncol] = size(val);
104     temp = 0;
105     for k = 1:nrow
106         temp(k) = val(k).tf;
107     end
108     mean_out = mean(temp);
109 end
110
111 function mean_out = mean_efficiency_vector(val, packet_size, dr)
112     [nrow, ncol] = size(val);
113     temp = 0;
114     for k = 1:nrow
115         p = 0;
116         R = get_physical_bit_rate(dr);
117         t2_seg = 0;
118         t1_seg = val(k).total_time_for_window/1000;
119         temp(k) = (packet_size*8)/((t1_seg + t2_seg*(p/(1-p)))*R);
120     end
121     mean_out = mean(temp);
122 end
123
124 function std_out = std_total_time_for_schc_packet(val)
125     [nrow, ncol] = size(val);
126     temp = 0;
127     for k = 1:nrow
128         temp(k) = val(k). total_time_for_schc_packet ;
129     end
130     std_out = std(temp);
131 end
132
133 function std_out = std_total_time_for_window(val)
134     [nrow, ncol] = size(val);
135     temp = 0;
136     for k = 1:nrow
137         temp(k) = val(k).total_time_for_window;
138     end
139     std_out = std(temp);
140 end
141
142 function std_out = std_time_frame(val)
143     [nrow, ncol] = size(val);
144     temp = 0;
145     for k = 1:nrow
146         temp(k) = val(k).tf;
147     end
```

```
148     std_out = std(temp);
149 end
150
151 function std_out = std_efficiency_vector(val, packet_size, dr)
152     [nrow, ncol] = size(val);
153     temp = 0;
154     for k = 1:nrow
155         p = 0;
156         R = get_physical_bit_rate(dr);
157         t2_seg = 0;
158         t1_seg = val(k).total_time_for_window/1000;
159         temp(k) = (packet_size*8)/((t1_seg + t2_seg*(p/(1-p)))*R);
160     end
161     std_out = std(temp);
162 end
163
164 function [pbr] = get_physical_bit_rate(dr)
165     dr_vector = [0 1 2 3 4 5];
166     pbr_vector = [250 440 980 1760 3125 5470];
167     pbr = pbr_vector(find(dr_vector == dr));
168 end
```

Apéndice F

Proceso de instalación de PySCHC

Para lograr instalar la aplicación en *Google Kubernetes Engine (GKE)* se deben seguir los siguientes pasos:

Compilación de la imagen del contenedor

Antes de compilar la aplicación en Docker, es necesario cumplir con lo siguiente:

- Tener instalado Ubuntu 16.04 TLS o superior
- Tener instalado Docker
- Tener instalado el SDK de Google Cloud (gcloud)
- Tener instalado Kubernetes (kubectl)

1.- Hacer un checkout del repositorio de GIT

```
mkdir tesis
cd tesis
git clone https://github.com/RodrigoMunozLara/Tesis.git
```

2.- Actualizar la copia local constantemente para ver los cambios reflejados

```
git pull
```

3.- (opcional) Listar todas las imagenes que hay cargadas en Docker

```
docker images ls
```

4.- (opcional) Borrar una imagen en particular

```
docker rmi [IMAGE ID]
```

5.- (opcional) Listar todos los container que hay cargados en Docker

```
docker container ls
```

6.- (opcional) Borrar un container en particular

```
docker rm []
```

7.- (opcional) Construir una imagen en Docker usando un dockerfile

```
docker image build -t [PATH]
(ejemplo: docker image build -t prueba:1.0 .)
```

8.- (opcional) Ejecutar un container en docker

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
(ejemplo: docker container run --name test prueba:1.0)
```

9.- Establecer la variable de entorno $PROJECT_ID$ con el ID del proyecto de Google Cloud. Esta variable se usará para asociar la imagen del contenedor con el Container Registry del proyecto.

```
export PROJECT\_ID=[PROJECT\_ID]
```

10.- Compilar la imagen del contenedor de la aplicación y etiquetarla para subirla. Para ello se ejecuta el siguiente comando:

```
docker build -t gcr.io/${PROJECT_ID}/prueba-app:v1 .
```

Subir la imagen del contenedor a Container Registry de Google

11.- Configurar la herramienta de línea de comandos de Docker para autenticarse en el Container Registry (debes ejecutar esto solo una vez):

```
gcloud auth configure-docker
```

12.- Ahora se puede usar la herramienta de línea de comandos de Docker para subir la imagen en el Container Registry:

```
docker push gcr.io/${PROJECT_ID}/prueba-app:v1
```

Crear un clúster de contenedores

13.- Configurar las opciones del ID del proyecto y de la zona de Compute Engine para la herramienta de gcloud:

```
gcloud config set project $PROJECT_ID
gcloud config set compute/zone [COMPUTE_ENGINE_ZONE]
```

14.- Ejecutar el siguiente comando para crear un clúster de dos nodos llamado hello-cluster:

```
gcloud container clusters create hello-cluster --num-nodes=2
```

15.- La creación del clúster puede tardar varios minutos. Una vez que se completó el comando, ejecutar el siguiente comando y observar las dos instancias de VM del clúster:

```
gcloud compute instances list
```

Implementar la aplicación

16.- Ejecutar el siguiente comando para implementar la aplicación:

```
kubectl create deployment prueba-no-web --image=gcr.io/${PROJECT_ID}/prueba-app:v1
```

17.- Para ver el pod creado por la implementación, ejecutar el siguiente comando:

```
kubectl get pods
```

18.- Exponer la aplicación a Internet

```
kubectl expose deployment hello-web --type=LoadBalancer --port 80 --target-port 8080
```

19.- Para averiguar la IP externa que GKE aprovisionó para la aplicación, ejecute el siguiente comando:

```
kubectl get service
```

20.- Ver los logs de la aplicación

```
kubectl log [pod-name]
```

21.- Para ver los pods

```
kubectl get pods
```

Bibliografía

[1] Ana Minaburo y col. *SCHC: Generic Framework for Static Context Header Compression and Fragmentation*. RFC 8724. Abr. de 2020. DOI: 10.17487/RFC8724. URL: https://rfc-editor.org/rfc/rfc8724.txt.

[2] IETF. *IPv6 over Low Power Wide-Area Networks (lpwan)*. URL: https://datatracker.ietf.org/wg/lpwan/about/.

[3] LoRa Alliance. *LoRaWAN and NB-IOT: Competitors or Complementary?* URL: https://lora-alliance.org/sites/default/files/2019-06/cr-lora-102_lorawanr_and_nb-iot.pdf.

[4] Stephen Farrell. *Low-Power Wide Area Network (LPWAN) Overview*. RFC 8376. Mayo de 2018. DOI: 10.17487/RFC8376. URL: https://rfc-editor.org/rfc/rfc8376.txt.

[5] Olivier Gimenez e Ivaylo Petrov. *Static Context Header Compression (SCHC) over LoRaWAN*. Internet-Draft draft-ietf-lpwan-schc-over-lorawan-07. Work in Progress. Internet Engineering Task Force, abr. de 2020. 25 págs. URL: https://datatracker.ietf.org/doc/html/draft-ietf-lpwan-schc-over-lorawan-07.

[6] Semtech. *SX1272/73 - 860 MHz to 1020 MHz Low Power Long Range Transceiver Datasheet*. Document. Revision 4. Semtech, ene. de 2019. URL: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1272.

[7] Pycom. *Specification Sheets LoPy4*. URL: https://pycom.io/wp-content/uploads/2018/08/lopy4-specsheet.pdf.

[8] RAK Wireless. *RAK 7258 Micro Gateway Datasheet*. URL: https://doc.rakwireless.com/datasheet/rakproducts/rak7258-micro-gateway.

[9] U. Raza, P. Kulkarni y M. Sooriyabandara. "Low Power Wide Area Networks: An Overview". En: *IEEE Communications Surveys Tutorials* 19.2 (2017), págs. 855-873. DOI: 10.1109/COMST.2017.2652320.

[10] Sigfox. *Radio Configurations*. URL: https://build.sigfox.com/sigfox-radio-configurations-rc.

[11] LoRa Alliance. *LoRaWAN Regional Parameters RP002-1.0.0*. 2019. URL: https://lora-alliance.org/sites/default/files/2020-01/rp_2-1.0.0_final_release.pdf (visitado).

[12] F. Adelantado y col. "Understanding the Limits of LoRaWAN". En: *IEEE Communications Magazine* 55.9 (sep. de 2017), págs. 34-40. ISSN: 0163-6804. DOI: 10.1109/MCOM.2017.1600613.

[13] LoRa Alliance. *LoRaWAN Specification v1.0.3*. URL: https://lora-alliance.org/resource-hub/lorawanr-specification-v103.

[14] W. Ayoub y col. "Towards IP over LPWANs technologies: LoRaWAN, DASH7, NB-IoT". En: *2018 Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC)*. Abr. de 2018, págs. 43-47. DOI: 10.1109/DINWC.2018.8356993.

[15] P. Weber y col. "IPv6 over LoRaWAN™". En: *2016 3rd International Symposium on Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*. Sep. de 2016, págs. 75-79. DOI: 10.1109/IDAACS-SWS.2016.7805790.

[16] S. Thielemans, M. Bezunartea y K. Steenhaut. "Establishing transparent IPv6 communication on LoRa based low power wide area networks (LPWANS)". En: *2017 Wireless Telecommunications Symposium (WTS)*. Abr. de 2017, págs. 1-6. DOI: 10.1109/WTS.2017.7943535.

[17] C. Gomez y col. "IPv6 over LPWANs: Connecting Low Power Wide Area Networks to the Internet (of Things)". En: *IEEE Wireless Communications* 27.1 (2020), págs. 206-213.

[18] Khaled Q. Abdelfadeel, Victor Cionca y Dirk Pesch. "LSCHC: Layered Static Context Header Compression for LPWANs". En: *Proceedings of the 12th Workshop on Challenged Networks*. CHANTS '17. Snowbird, Utah, USA: Association for Computing Machinery, 2017, págs. 13-18. ISBN: 9781450351447. DOI: 10.1145/3124087.3124092. URL: https://doi.org/10.1145/3124087.3124092.

[19] K. Q. Abdelfadeel, V. Cionca y D. Pesch. "Dynamic Context for Static Context Header compression in LPWANs". En: *2018 14th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. Jun. de 2018, págs. 35-42. DOI: 10.1109/DCOSS.2018.00013.

[20] R. Sanchez-Iborra y col. "IPv6 communications over LoRa for future IoV services". En: *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*. 2018, págs. 92-97.

[21] Nicolas Maturana Araneda. "Implementation and evaluation of static context header compression for IPv6 packets within a LoRaWAN network". Tesis de mtría. Universidad de Chile, 2019. URL: http://repositorio.uchile.cl/handle/2250/170134.

[22] IETF104 Hackathon. *OpenSCHC*. 2019. URL: https://github.com/openschc/openschc/wiki (visitado).

[23] D. Singh, O. G. Aliu y M. Kretschmer. "LoRa Wan Evaluation for IoT Communications". En: *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Sep. de 2018, págs. 163-171. DOI: 10.1109/ICACCI.2018.8554713.

[24] Alberto Leon-Garcia. *Communication Networks: Fundamental Concepts and Key Architectures*. McGraw-Hill, 2003. ISBN: 0071198482. URL: https://www.amazon.com/Communication-Networks-Fundamental-Concepts-Architectures/dp/0071198482?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0071198482.