Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Modelado y evaluación de la tecnología Sigfox para NS3

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Adrián Munera Sánchez

Tutor(es): Alberto Miguel Bonastre Pina

José Carlos Campelo Rivadulla

Julio de 2018

Resumen

Este trabajo de final de grado presenta la tecnología Sigfox dentro del ámbito del Internet de las Cosas, desarrolla un modelo de simulación de la tecnología y valida su correcto funcionamiento. En primer lugar, en una pequeña introducción general al Internet de las Cosas se explican sus principales características y los diferentes campos de aplicación, y más adelante se centra en su uso más concreto en redes de área extensa y bajo consumo, las cuales son el interés de este trabajo. Seguidamente se desarrollan las principales tecnologías dentro de este ámbito, donde se observa que Sigfox destaca actualmente gracias a sus prestaciones.

Por ello, este trabajo se centra en desarrollar un modelo de simulación práctico que permita crear topologías y evaluar el rendimiento de esta tecnología. Tras analizar algunos de los simuladores de redes actuales se decide optar por el simulador NS-3. Finalmente se realizan pruebas reales de la tecnología Sigfox para confirmar el correcto funcionamiento del modelo de simulación de acuerdo con los datos obtenidos.

Palabras clave: Sigfox, Internet de las Cosas, Redes de área extensa, NS3, modelo de simulación.

Abstract

This final degree project presents Sigfox technology within the scope of the Internet of Things, a simulation model of technology is developed and its correct functioning is validated. First, in a brief general introduction to the Internet of Things, its main characteristics and different fields of application are explained, and then it focuses on its more concrete use in wide area and low consumption networks, which are of interest of this work. The main technologies within this field are presented, where it is observed that Sigfox stands out thanks to its benefits.

Therefore, this work focuses on developing a simulation model that allows creating topologies and evaluating the performance of this technology. After analysing some of the current network simulators it is decided to opt for the NS-3 simulator. Finally, real tests of the Sigfox technology are carried out to confirm the correct functioning of the simulation model according to the obtained data.

Keywords: Sigfox, Internet of Things, Wide Area Networks, NS3, simulation model.

Tabla de contenidos

Tabla de contenido

Índice de figuras

1. Introducción

Recientemente el concepto de Internet de las Cosas (IoT, "Internet of Things") se ha vuelto popular gracias a la evolución tecnológica en la que estamos inmersos, donde cada día más dispositivos están conectados a la red y ofrecen la posibilidad de comunicación remota. Esta evolución en las comunicaciones es interesante tanto en entornos sencillos (como es el caso de la domótica) como en los más complejos, donde existen grandes cantidades de dispositivos a controlar. Las tecnologías adaptadas a entornos de áreas extensas para internet de las cosas son recientes y llevan muy poco tiempo en uso, por lo que aún muchas de ellas son grandes desconocidas pese a su gran potencial.

En este trabajo se presentan varias de las tecnologías de red más utilizadas (Bluetooth, Zigbee, etc), donde se destacan las tecnologías pensadas para el ámbito de redes de área extensa, es decir, para redes que cubren grandes distancias, y que a menudo van ligadas con la necesidad un bajo coste energético. Sigfox [1] es una de estas tecnologías, y es de las más prometedoras como se verá más adelante. Por eso es la elegida como objetivo principal de este trabajo académico, en el que se detallará la implementación de un modelo de simulación y su posterior validación.

Para el desarrollo del modelo de simulación se ha partido de una versión inicial creada por el alumno del máster de ingeniería de computadores y redes Pablo Pardal Garcés. Esta versión fue presentada en su trabajo final de máster y únicamente modela el nivel físico de la tecnología. Mediante el presente trabajo de final de grado se ha completado todo el nivel lógico y se han hecho correcciones a su trabajo, las cuales se explicarán en detalle en el apartado del modelo de simulación. Posteriormente a estas correcciones e implementaciones se ha podido comprobar mediante pruebas de uso reales que el funcionamiento del modelo de simulación ahora es completamente funcional.

Tal como menciona él mismo en el trabajo futuro de su proyecto de máster [2]:

"*Asimismo, abre la puerta para introducir nuevas mejoras y parámetros de comunicación en la tecnología Sigfox. […]Se contempla, además, la validación de todos los modelos de simulación anteriores mediante medidas obtenidas con implementaciones reales de estas tecnologías en entornos reales…*"

Se ha modificado su modelo en gran medida y se ha ampliado para que sea funcional, siendo ahora su parte únicamente un 25% del modelo diseñado. Finalmente se ha validado el funcionamiento con pruebas experimentales reales.

Este trabajo se ha desarrollado bajo el marco de una beca de colaboración con el Departamento de Informática de Sistemas y Computadores (DISCA), ofertada para el curso académico 2017-2018 por el Ministerio de Educación. Esta beca tiene el fin de promover la iniciación en tareas de investigación de los estudiantes universitarios que vayan a finalizar los estudios del grado. En este sentido, este Trabajo Fin de Grado se ha presentado dentro del proyecto de investigación DPI2016-80303-C2-1-P titulado "Hacia el Hospital Inteligente: Investigación en el diseño de una plataforma basada en Internet de las Cosas y su aplicación en la mejora del cumplimiento de higiene de manos", del grupo de Redes Inalámbricas de Sensores del Instituto de Tecnologías de la Información y Comunicaciones (ITACA) de la Universitat Politècnica de València.

1.1 Motivación

Las tecnologías de área extensa y bajo consumo, como se ha mencionado en la introducción, son muy novedosas en el sentido de que llevan poco tiempo en el mercado y en evaluación. Su uso y utilidad ha aumentado en gran medida debido a la evolución tecnología y el auge de Internet de las Cosas, de esta forma se han convertido en unas de las tecnologías con más interés actual. Es por esto que es totalmente necesario realizar un estudio en profundidad de su funcionamiento y de sus características. Pese a que es una de las tecnologías con más potencial, actualmente no existe ningún modelo de simulación, al menos documentado públicamente, de la tecnología Sigfox, que permita modelar instalaciones reales y obtener unos resultados que puedan predecir de forma ajustada la realidad.

La creación de este modelo de simulación permitirá estudiar las futuras instalaciones y evaluar las prestaciones de la red antes de una posible implementación real. Prácticamente todas las tecnologías de red importantes hoy en día disponen de al menos un modelo de simulación, por lo que se considera necesario que esta tecnología también disponga de esta ventaja. Los modelos de simulación, además de permitir evaluar situaciones reales, también permiten comparar resultados de distintas tecnologías para un mismo escenario.

1.2 Objetivos

El objetivo principal es la creación de un modelo de simulación para la tecnología Sigfox y la validación de su correcto funcionamiento. Para ello será necesario definir la estructura del modelo de simulación, implementar y ensamblar las diferentes partes de esta estructura, y depurar los errores derivados de la versión inicial. Una vez implementado, el modelo se validará mediante pruebas experimentales comprobando que los resultados obtenidos en la simulación concuerdan con datos medidos in situ, considerados reales. Para ello se realizarán distintas mediciones sobre una antena real de Sigfox, enviando mensajes desde distintas partes de la provincia de Valencia con el fin de emular entornos rurales, suburbanos y urbanos, y se obtendrán los valores de la potencia de la señal recibida por la antena. La comparación de estos valores con los obtenidos mediante simulación para estos entornos permitirá validar el modelo.

Como objetivo secundario se considera ofrecer una visión general sobre el concepto de Internet de las Cosas. Para ello se explicarán las ideas generales que lo definen. También se estudiará su aplicación en el campo de redes de área extensa y bajo consumo, así como las tecnologías predominantes a día de hoy, en especial la tecnología Sigfox.

1.3 Estructura del trabajo

El trabajo se divide principalmente en los siguientes puntos:

- **Internet de las cosas:** En este apartado se explica qué es este concepto, los elementos por los que está formado, así como una explicación detallada de cada uno y su interés actual.
- **Sigfox:** En este apartado se desarrolla en detalle cómo funciona esta tecnología y cuáles son sus características. Se entrará en profundidad en cada uno de los aspectos más importantes de la tecnología debido a que es uno de los objetivos principales del trabajo.
- **Simuladores de redes:** En este apartado se exponen algunos de los simuladores de redes más famosos actualmente y se comenta brevemente qué ofrece cada uno. Además, se desarrolla con profundidad el funcionamiento del simulador NS-3 [3].
- **Modelo de simulación:** Aquí se entra en detalle de cómo es el modelo de simulación implementado, incluyendo un diagrama de su estructura, así como una explicación de

cada una de sus partes más relevantes. Se explicará adecuadamente qué se ha aportado al trabajo inicial y qué facilidades aporta NS-3.

- **Validación del modelo:** En este último apartado se describe la campaña de medidas realizada con el fin de validar el modelo, indicando qué medidas se han tomado y describiendo detalladamente el proceso de obtención de la mismas sobre la tecnología Sigfox. Finalmente, se presenta un estudio comparativo entre los resultados simulados y los datos reales que permite validar el modelo.

1.4 Planificación del trabajo

En la figura 1 se representa mediante un diagrama de Gantt la planificación seguida desde el inicio del proyecto. Los colores indican una temática, con el fin de separar fases dentro del desarrollo. El tiempo de desarrollo coindice con la duración de la beca de colaboración mencionada anteriormente.

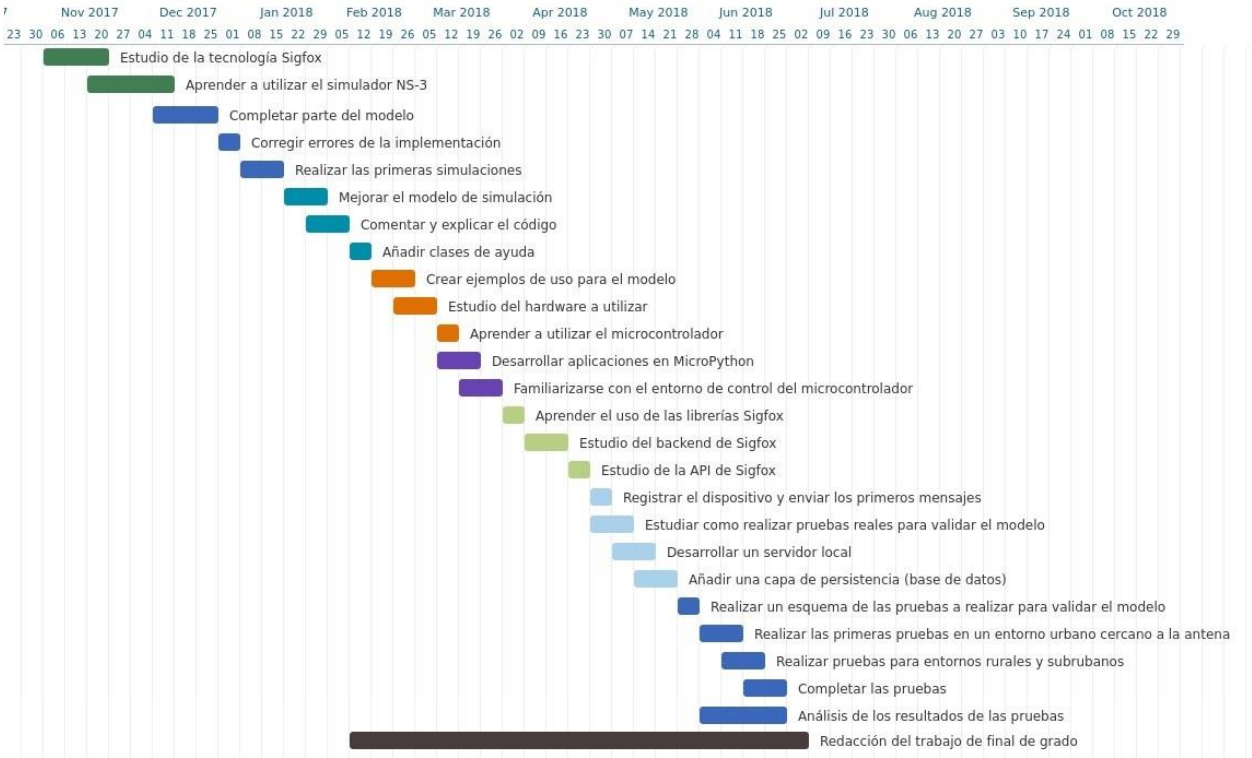

Figura 1 – Diagrama de Gantt, planificación del trabajo.

2. Situación actual de la tecnología y crítica

Actualmente, como se menciona en la motivación del trabajo y pese a que posee un gran potencial gracias a sus prestaciones, no existe ningún modelo de simulación de la tecnología Sigfox. Esto es debido a que es una tecnología nueva y que apenas lleva tiempo en uso.

Aun así, el alumno del máster de ingeniería de computadores y redes de la UPV Pablo Pardal Garcés realizó una primera aproximación a lo que sería un modelo que simula la tecnología de Sigfox [2]. Para ello se centró en modelar el nivel físico, basándose en otros modelos ya implementados de tecnologías similares. Por desgracia por aquel entonces no fue posible validar su trabajo de forma real. Por lo tanto, lo que hizo en su caso fue comparar los resultados con las especificaciones que daba la propia compañía de Sigfox, y obtener unos resultados similares, lo que derivó a algunos errores en el modelo que no se pudieron detectar.

Es por esto por lo que el objetivo de este trabajo de fin de grado es conseguir un modelo totalmente funcional de la tecnología, partiendo de la base del nivel físico diseñado por este alumno y validar finalmente su correcto funcionamiento. Se realizan múltiples correcciones a su trabajo debido a que no se ajustaba bien a la realidad en varios aspectos, y además se completa el nivel lógico del modelo de forma que sea posible definir escenarios de uso con varios dispositivos y poder obtener resultados verídicos. Los cambios que se han realizado a su modelo se explican en detalle en el apartado del modelo de simulación.

También mencionar que existen otras tecnologías del mismo campo de aplicación que Sigfox (redes de área extensa y bajo consumo), como por ejemplo LoRa [4], que sí disponen de un modelo de simulación creado por otros investigadores de universidades europeas, y hay artículos que evalúan las prestaciones de esta tecnología, como por ejemplo el artículo “LoRaWAN indoor performance analysis” [5] por el departamento de ingeniería de computadores de la universidad de Istanbul, o el artículo “Performance evaluation of LoRa networks in a Smart city scenario” [6] en la universidad de Padova, Italia.

3. Análisis del problema y solución propuesta

Una vez definida la situación actual de la tecnología se puede empezar a abordar el problema. Para ello se parte del objetivo de crear un modelo de simulación funcional para Sigfox. Existen dos formas de obtener un modelo de simulación de una tecnología, crear un modelo desde cero, o por el contrario basarse en el modelo de alguna tecnología similar o de algún trabajo anterior.

En este trabajo se ha optado por partir del modelo creado como trabajo de final de máster de un alumno. De esta forma no se ha tenido que empezar totalmente desde el principio, sino que parte de la base estructural del modelo ya se había realizado. Por tanto, el objetivo es ampliar en gran medida este modelo y realizar las correcciones oportunas sobre la parte ya implementada para que se ajuste a la realidad de las pruebas de validación.

Otro aspecto a tener en cuenta es el simulador que se va a utilizar. En este caso se ha elegido NS-3 por varias razones. Según los estudios realizados y como se refleja en numerosos artículos científicos, es el simulador con más prestaciones. Además, el modelo inicial del alumno de máster está realizado en NS-3 también, lo que ayuda a no tener que reproducir esa parte para otro simulador. NS-3 es ampliamente utilizado en muchos campos de investigación en los que se requieren realizar simulaciones de redes.

Una solución alternativa es empezar de cero el modelo de simulación. Sin embargo, ya que se dispone de una base, es más sensato utilizarla y aprovechar más el tiempo para llegar en conjunto a un modelo más completo, que comenzar desde cero y quizás no poder completar todo el modelo con la misma calidad.

Para validar el modelo no hay muchas posibilidades, ya que no se comercializan, y por tanto no es posible adquirir, una estación base de Sigfox. Las estaciones base son antenas conectadas a la infraestructura de una empresa (en concreto para Sigfox en España están instaladas y gestionadas por CELLNEX [7]), y son las encargadas de recoger los mensajes que envían los usuarios y de procesarlos. Todos los mensajes que un dispositivo envía mediante la tecnología de Sigfox son recogidos por las estaciones base, y en caso de que el dispositivo esté registrado en la infraestructura sus mensajes son procesados.

Por tanto, hubo que contactar con la empresa CELLNEX para que nos facilitara la posición de una de sus antenas que hacen de estación base para Sigfox. Gracias a esto se puede conocer la distancia concreta que recorre cada señal, desde la posición en la que se emite hasta la antena. Sabiendo esta distancia, y dado que en la web de Sigfox los usuarios pueden consultar los mensajes enviados por sus dispositivos, se puede hacer una relación entre la distancia y la potencia de la señal recibida por la estación base. Esta parte se explicará detalladamente en el apartado 9 que trata de la validación experimental realizada.

4. Internet de las cosas

Internet de las cosas o *internet of things* es un concepto que relaciona dos ideas: la existencia de objetos que son capaces de interactuar con el entorno y la habilidad de comunicarse de forma remota. Dotar de conectividad a los objetos que nos rodean nos permite intercambiar información valiosa y útil, que puede utilizarse para lograr un objetivo definido. Los objetos, al poseer de comunicación, son capaces también de colaborar entre ellos, por ejemplo, para dividirse el trabajo, o incluso para establecer una cadena de producción, donde cada uno realiza su parte individual y se complementan para realizar un objetivo, aplicación o servicio común.

De esta forma internet de las cosas, en un mundo en el que hoy en día los objetos con capacidad de procesamiento e interacción con el ambiente son cada vez más comunes, ofrece la posibilidad de recopilar toda esta información que nos proporcionan y utilizarla para crear en conjunto un entorno inteligente. El objetivo principal es poder crear espacios y sistemas inteligentes gracias a los cuales se pueda mejorar la calidad de vida de las personas, incrementar la eficiencia de las instalaciones, y facilitar el trabajo.

Internet de las cosas está muy relacionada con las tecnologías de *Big Data* y computación en la nube. De hecho, se puede considerar parte de su base, ya que es de ahí de donde se obtiene toda la información que más adelante se almacena, procesa, y filtra en servidores en la nube con alta capacidad de procesamiento. Si se suma la capacidad de obtención de información aportada por los objetos con conectividad, la posibilidad de recopilar toda esta información, y el procesamiento junto con técnicas de inteligencia artificial, se pueden conseguir entornos inteligentes con capacidad de ser independientes. Estos entornos aportan todo tipo de servicios relevantes a las personas, mejorando su calidad de vida en general.

Un punto muy importante a tener en cuenta también es la seguridad; todos los objetos son en esencia aparatos electrónicos, que hacen uso de algún tipo de comunicación conocida, y por lo tanto están expuestos a ser víctimas de ataques. En internet de las cosas la seguridad es algo fundamental, ya que cualquier vulnerabilidad podría provocar graves consecuencias. Se estudiará con más profundidad la seguridad en la sección 4.7.

4.1 Historia

Internet de las cosas es un concepto nuevo que surgió aproximadamente en el año 1999; sin embargo, muchas de sus bases ya fueron planteadas desde principios del siglo XIX, en el que la idea de máquinas comunicándose entre sí ya se había presentado, sobre todo con la invención del telégrafo y de las comunicaciones de voz por radio a partir del año 1900. El primer objeto dotado de conectividad que se conoce fue un dispositivo de información meteorológica en la cima del Mont Blanc por científicos franceses en 1874, que se comunicaba mediante ondas de radio con una central en París.

La utilización de internet tal y como hoy la conocemos comenzó en 1980, cuando la llamada red ARPANET creada por el departamento de defensa de Estados Unidos se expandió hacia un ámbito público. Ésta fue la base principal sobre la que se desarrolló internet. En 1990 el ingeniero John Ronkey creó el primer objeto conectado a internet: una tostadora capaz de encenderse de forma remota utilizando el protocolo TCP/IP.

Junto con el desarrollo de internet surgieron las redes de sensores inalámbricos; internet de las cosas se basa en este tipo de redes. Están formadas por un conjunto de sensores autónomos que monitorizan condiciones físicas o ambientales, y comparten los datos que obtienen datos a través de la red. Las redes de sensores inalámbricos fundaron las bases que actualmente se utilizan en internet de las cosas, donde varios objetos están conectados entre sí y se comunican

cooperativamente. En el siguiente apartado se explicará más en detalle cómo funcionan estas redes.

El concepto de internet de las cosas empezó a utilizarse de forma pública por primera vez en 2009 por el profesor Kevin Ashton en el *RFID journal* (Ashton, 2009); sin embargo, la idea no destacó entonces debido al nivel tecnológico del momento. En los últimos años mediante la popularización de las redes inalámbricas, este concepto ha crecido de forma exponencial, siendo usado a día de hoy en numerosos artículos científicos, campos de investigación y productos comerciales.

4.2 Redes de sensores inalámbricos

Las redes de sensores inalámbricos o WSN [8] (*Wireless sensor networks*) tienen su origen en iniciativas militares sobre el año 1980. Son redes de ordenadores pequeños llamados nodos inalámbricos, que poseen sensores con el fin de colaborar en una tarea común. No poseen ninguna infraestructura física y no son centralizadas, es decir, no necesitan ser administradas desde un nodo concreto. Básicamente permiten obtener información del entorno y ofrecerla de forma remota y continuada, esto abre un sinfín de posibilidades tanto comerciales como de investigación.

Los nodos pueden ser móviles y permiten ser desplegados en distintos lugares, debido a que pueden comunicarse sin necesidad de cableado. La distancia se limita por la tecnología de red utilizada.

Las redes de sensores se utilizan en numerosos campos industriales y de consumo, como el control de procesos industriales, monitorización de la salud de un paciente, domótica, control ambiental, etc. También es importante destacar que los nodos al muchas veces no estar físicamente conectados a una red eléctrica carecen de una fuente de alimentación constante, lo que hace que el consumo energético sea otro de los inconvenientes que tienen que resolver. En la figura 2 se representa una red de sensores.

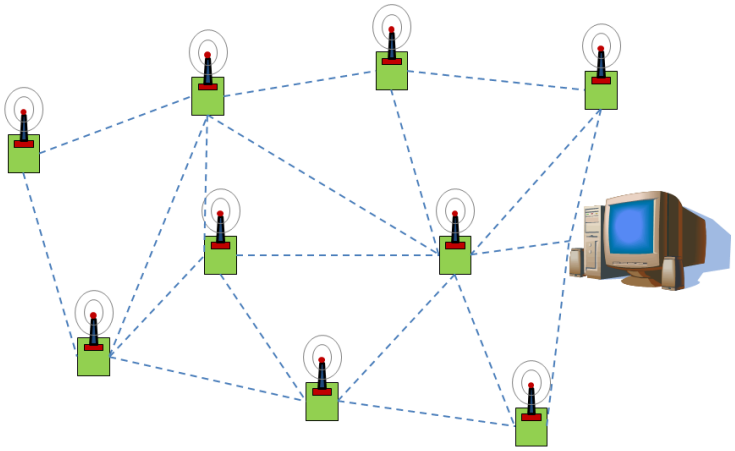

Figura 2 - Red de sensores inalámbricos.

En el ejemplo de la figura se observa como los nodos forman entre sí una serie de enlaces. Estos enlaces determinan el orden en el que se comunican los nodos, se establecen rutas con el fin de que los mensajes lleguen al destino de la forma más adecuada posible.

Pese a que la idea de cómo funcionan estas redes ya estuviera establecida desde hace mucho tiempo, actualmente siguen siendo un campo de investigación muy activo, y han surgido aplicaciones comerciales al respecto. Es más, este trabajo se centra sobre Sigfox, la cual es una tecnología de red pensada para un caso concreto de este tipo de redes.

4.3 Conceptos generales

Para entender mejor cómo funciona internet de las cosas y en que se basa, en este apartado se describen algunos de los conceptos generales que lo definen. Aunque la idea en sí es bastante simple (objetos dotados de conectividad) cabe destacar algunos aspectos que son fundamentales:

- **Dispositivos y *Hardware*:**

  El *hardware* es uno de los componentes principales. Éste es el que se encarga de relacionarse con el mundo físico, por ejemplo, mediante algún tipo de sensor: sensor de luz, de distancia, de humedad, acelerómetros, etc. O mediante el uso de actuadores que modifiquen su entorno, siendo el ejemplo más común los motores.

  El *hardware* está incorporado en el objeto en sí, y debe poseer de conectividad para ser capaz de comunicarse y de transferir la información que recopila. Esto quiere decir también que debe tener la habilidad de procesar datos, utilizando por ejemplo algún tipo de microcontrolador o algún circuito integrado, ya que en definitiva los objetos en su mayoría son sistemas embebidos y deben tratarse como tal.

- **Procesamiento de información:**

  Debe existir algún tipo de procesamiento de datos e información, el cual depende totalmente del sistema a implementar. Por ejemplo, para el uso de aplicaciones o servicios pequeños como una casa domótica la tarea de manejar la información y de tratarla puede hacerla directamente el propio objeto. Esto es posible ya que no suelen ser cargas de trabajo elevadas y el microcontrolador puede realizarlas sin problemas.

  Sin embargo, cuando se habla de sistemas grandes en los que existen miles de objetos conectados es deseable mantener un sistema más centralizado. En este caso el sistema se tiene que encargar de recopilar esa información, dejando a los objetos en un segundo plano simplemente como proveedores de información o como actuadores. Aquí intervienen en gran medida las tecnologías de *Big Data* y de computación en la nube, que en estos casos son la solución más apropiada actualmente. Existen numerosas plataformas que ofrecen estos servicios.

- **Conectividad:**

  La conectividad es un aspecto fundamental ya que es la que define la forma en la que se comunican los objetos. Al igual que el procesamiento de datos, la conectividad depende en gran medida del tipo de sistema que se quiera implementar. Hay muchos tipos de comunicación, cada uno de ellos tiene sus ventajas y sus inconvenientes, por lo que elegir cual se debe utilizar depende de las características de la aplicación y de los costes y riesgos que se quieran asumir.

  Por ejemplo, para el caso de una casa domótica se podría escoger la tecnología WiFi si no importa el coste energético y solo se requiere comodidad de uso; sin embargo, si se van a usar dispositivos con una energía limitada seguramente sea mejor opción utilizar

tecnologías pensadas para estos casos, como puede ser Zigbee. En el otro extremo, si la aplicación requiere miles de dispositivos independientes, lejanos entre sí, y con energía muy limitada, lo ideal sería utilizar alguna de las tecnologías de larga distancia y bajo consumo, como LoRa o Sigfox.

- **Seguridad:**

  La seguridad es un punto también sumamente importante en internet de las cosas, ya que tanto los objetos como las comunicaciones son susceptibles de ser atacados. Los sistemas grandes de objetos son propensos a poseer varias vulnerabilidades y derivar en inconvenientes. Por ejemplo, en el caso de una ciudad inteligente, podría provocar accidentes en las calles debido al mal funcionamiento del sistema, o se podría alterar el funcionamiento para favorecer a algún fin malintencionado.

  Es por ello por lo que a la hora de diseñar el sistema se ha de establecer una capa de seguridad y se debe tener en cuenta, desde el primer momento, que ésta es una de las prioridades. Alguna de las medidas necesarias en prácticamente cualquier sistema es que los datos deben ser totalmente inaccesibles e inalterables para personas ajenas, y no se deben poder modificar los objetos, o al menos, no permitir que se puedan manipular a voluntad.

Todos estos conceptos generales se irán desarrollando en detalle en los siguientes apartados.

4.5 Tecnologías de comunicación

Las redes de comunicación han ido evolucionando hacia el sector de internet de las cosas en los últimos años, y han surgido varias redes que intentan satisfacer esta demanda, mientras que otras se han adaptado. Teniendo en cuenta que el término internet de las cosas engloba a cualquier par de objetos que estén conectados a la red y no estén próximos entre sí, este sector es realmente amplio.

Las redes se pueden clasificar según su alcance de la siguiente forma (Figura 3):

- **PAN:** *Personal Area Network*, son redes que abarcan un rango de apenas unos metros. Sirven para conectar varios dispositivos típicamente en la misma habitación. Ejemplo: Bluetooth.

- **LAN:** *Local Area Network*, son redes que abarcan un rango de unas pocas decenas de metros. Sirven para conectar dispositivos dentro del espacio de un edificio pequeño. Ejemplo: Wifi.

- **MAN:** *Metropolitan Area Network*, son redes que abarcan el tamaño de una ciudad, aproximadamente 10 kilómetros. Sirven para dotar de conectividad a áreas residenciales. Ejemplo: WiMAX.

- **WAN:** *Wide Area Network*, son las redes más extensas, cubren países o continentes enteros. Sirven para comunicar entre sí varias LAN. Ejemplo: 3G

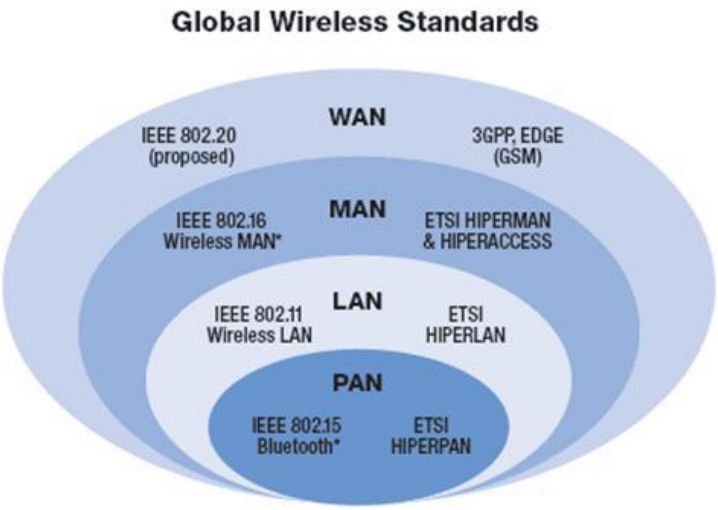

Figura 3 - Clasificación de redes según el alcance [9]

Fuera de esta clasificación estaría LPWAN [10] (*Low Power Wide Area Network*), es decir, redes de área extensa y bajo consumo. Es una de las nuevas redes surgidas en la última década debido al auge de las redes de sensores.

LPWAN es un tipo de red inalámbrica diseñada para cubrir grandes distancias y permitir transmitir mensajes de unos pocos bytes. La idea es enviar paquetes pequeños de información para que se consiga que el dispositivo tenga el mínimo consumo energético posible. De esta forma los dispositivos pueden mantenerse autónomos por un largo periodo de tiempo gracias a su propia fuente de alimentación, típicamente una batería.

Estas redes son especialmente útiles para tareas que requieren de sensores que monitoricen el estado de un entorno. Permiten colocar un gran número de dispositivos a grandes distancias y no requieren de estar conectados a una red eléctrica. Un ejemplo de tecnologías de red que entran dentro de este tipo son Sigfox y LoRa.

A continuación, se verá en detalle cuales son las tecnologías de red más utilizadas hoy en día en el internet de las cosas:

4.5.1 M2M (*Machine to Machine*) GSM/GPRS

Las redes M2M son muy apoyadas por las grandes empresas de telecomunicaciones, Están basadas en el modelo de negocio del GPRS [11] (*General Packet Radio Service*), pago por Mbyte transmitido, de igual forma que las tecnologías 3G/4G.

Sin embargo, en un sistema en el que hay miles de dispositivos conectados esta tecnología tiene las desventajas de que es difícilmente escalable, tiene una cobertura asociada a un operador, y presenta un coste por datos transmitidos. Además, las transmisiones de datos suponen un coste energético elevado, a menudo inasumible por la mayoría de los dispositivos pensados para internet de las cosas.

Las redes M2M son las más comercializadas en el mundo de internet de las cosas, pero para un futuro próximo se predice un declive en su uso frente a las tecnologías con modulación *NarrowBand* (es decir, que utilizan canales con rangos de frecuencias muy pequeños).

4.5.2 Sigfox

Sigfox es la red de comunicaciones LPWAN para internet de las cosas más extendida a nivel mundial, cubriendo un 98% del territorio europeo y americano. Utiliza modulación UNB (*Ultra Narrow Band*) que le permite mitigar mejor el ruido en la banda de 868Mhz para Europa y 902Mhz en Estados Unidos, con un alcance de varios kilómetros.

Una de las ventajas que ofrece Sigfox además de su gran rango de cobertura es la posibilidad de almacenar toda la información en sus servidores y la infraestructura que ofrece a sus usuarios, lo que permite que sea fácilmente accesible, además del bajo precio de sus dispositivos. También dispone de la posibilidad de usar tecnologías de la nube de forma nativa, tales como *Amazon Web Services* o *Microsoft Azure*. El coste energético de las transmisiones y recepciones mediante esta tecnología es muy bajo, lo cual es ideal para internet de las cosas.

La tecnología Sigfox permite enviar unos pocos mensajes al día, hasta 140, y su tamaño es de apenas una docena de bytes como máximo. Por último, hay que mencionar que las descargas de información son más lentas que las subidas hacia estaciones base, por lo que tiene una velocidad asimétrica. En el apartado 5 se describe en detalle esta tecnología.

4.5.3 LoRa

LoRa también es una red LPWAN, con un modelo de negocio muy similar a Sigfox, pese a que la tecnología que utiliza es algo diferente. En vez de *NarrowBand* emplea *Spread Spectrum*, que consiste en transmitir el mensaje a lo largo de una banda muy ancha de frecuencias. Destaca frente a Sigfox en que es más apropiada para comunicaciones que requieran intercambiar información constante de forma bidireccional. Otra ventaja es que presenta unas especificaciones más abiertas que Sigfox.

LoRa puede transmitir en diferentes frecuencias pertenecientes a las bandas de 109 Mhz, 433 Mhz, 866 Mhz y 915 Mhz, y ofrecen una funcionalidad bidireccional, donde la subida y la descarga de información se realizan a la misma velocidad.

Uno de los principales inconvenientes de LoRa es que no proporciona una infraestructura de red a sus clientes, es decir, únicamente desarrolla el estándar y se basa en vender transceptores. Esto quiere decir que cualquier usuario que desee emplear LoRa debe de gestionar completamente su propia infraestructura. El coste de inversión y la experiencia previa necesaria es algo importante a asumir para poder utilizar esta tecnología.

4.5.4 BLE (*Bluetooth Low Energy*)

El Bluetooth de baja energía, conocido también como *Bluetooth Ultra Low Power* o *Bluetooth Smart* [12], es una tecnología inalámbrica pensada para conectar pequeños dispositivos, dentro de un área de unos pocos metros, y con bajos volúmenes de datos.

Esta tecnología destaca, al igual que las tecnologías de red de LWPAN, en su consumo energético reducido para aumentar la autonomía de los dispositivos. Esto lo consigue minimizando la potencia de transmisión de las señales de radio y reduciendo el radio de cobertura. Tiene la característica de que las distancias que cubre son muy pequeñas. Opera en la banda de 2.4 GHz ya que es una banda ISM [40] de uso libre.

BLE pretende ser utilizado en pequeñas redes domésticas, en las que por ejemplo los electrodomésticos o pequeños aparatos puedan ser capaces de comunicarse, mientras que tiene una utilidad reducida en entornos industriales y redes de sensores.

4.5.5 Zigbee

Zigbee [13] es una tecnología de comunicación inalámbrica ampliamente utilizada en redes de sensores y aplicaciones de domótica e industriales. Posee un alcance de centenares de metros, y está pensado para volúmenes bajos de carga de información. También opera en la banda de 2.4 Ghz, y es capaz de emplear una topología de red en malla.

Zigbee permite tres topologías de red: en estrella, en árbol y en malla. La topología en malla es una de las causas por la que puede triunfar esta tecnología, ya que si en un momento dado un nodo falla se pueden rehacer las conexiones gracias a un nodo que haga la función de coordinador.

Al igual que las anteriores tecnologías tiene un consumo energético reducido, pero Zigbee destaca por una mayor seguridad en las comunicaciones, alta escalabilidad, y capacidad de soportar grandes cantidades de dispositivos en una misma red. Sin embargo, tiene como principal desventaja de que su velocidad es bastante menor que la que ofrecen otras tecnologías como Bluetooth.

Esta tecnología lleva varios años siendo utilizada, y es muy popular en su ámbito debido a sus prestaciones.

4.5.6 Comparación

En la figura 4 se muestra una clasificación de las tecnologías según su facilidad de uso y su evolución en el mercado:

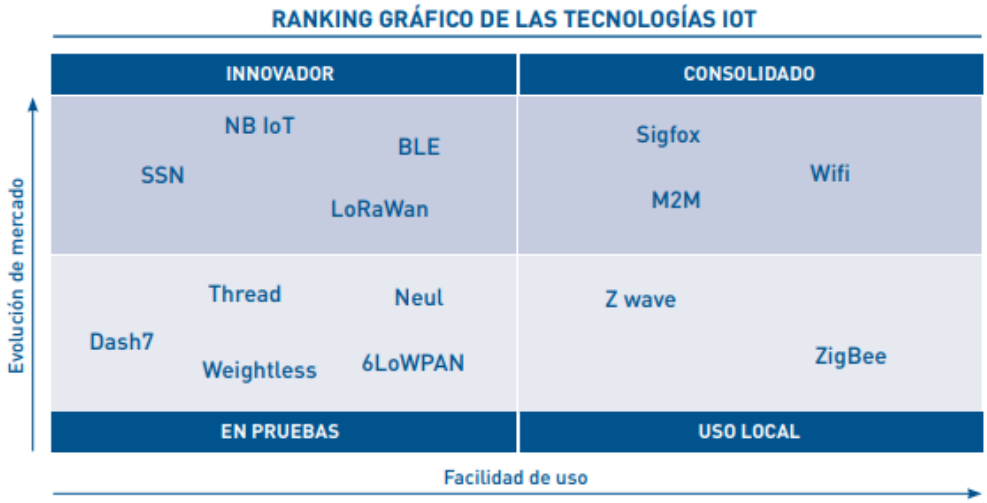

Figura 4 - Comparación de las tecnologías de red según su evolución de mercado y facilidad de uso. Fuente Efor [14]

La figura 5 muestra una comparación de las tecnologías que se han desarrollado en los apartados anteriores frente a otras, atendiendo a factores como: consumo, alcance, madurez, disponibilidad, seguridad, usabilidad y tasa de datos.

TECNOLOGÍA	CONSUMO	ALCANCE	MADUREZ	DISPONIBILIDAD	SEGURIDAD	USABILIDAD	TASA DE DATOS
GSM/GPRS	Muy alto	Alto	Muy Alto	Muy alto	Alta	Alta	Alta
SigFox	Bajo	Medio	Alto	Medio	Media	Alta	Muy baja
LoRa	Bajo	Medio	Bajo	Muy bajo (ad hoc)	N A	Baja	Muy baja
WiFi	Alto	Bajo	Muy alto	Alto	Baja	Alta	Muy alta
BLE	Muy bajo	Muy bajo	Alto	Bajo	Baja	Media	Baja
ZigBee	Medio	Bajo	Medio	Muy bajo	Alta	Baja	Baja

Figura 5 - Comparación de las prestaciones de diversas tecnologías de red. Fuente Efor [14]

No existe una tecnología mejor que otra sobre el papel, sino que cada una tiene sus ventajas e inconvenientes y se debe balancear esta información para determinar cuál es más oportuna para cada caso de uso.

Las tecnologías LPWAN, como son LoRa y Sigfox, pese a que ambas tienen el mismo campo de aplicación y unas prestaciones similares, son muy diferentes entre sí. La principal ventaja que tiene Sigfox frente a LoRa es que dispone de toda una infraestructura ya desplegada por gran parte del mundo, y su uso es bastante más sencillo y económico que LoRa, que requiere del despliegue y administración de la red por parte del propio usuario o de terceras compañías. Sin embargo, esto también implica que los usuarios de LoRa tienen más capacidad de actuación sobre sus propias redes, pudiendo modificar casi cualquier aspecto de la infraestructura, mientras que Sigfox es más cerrado en este aspecto.

Este trabajo académico se centra en las tecnologías LPWAN, en concreto Sigfox, debido a que es una de las tecnologías más prometedoras y de la cual no existe ningún modelo de simulación actual.

4.5 Dispositivos

Los dispositivos hacen referencia a los objetos dentro del mundo de internet de las cosas. Son los que poseen el *hardware* encargado de comunicarse con el entorno mediante sensores y/o actuadores y permiten obtener y recibir información mediante algún tipo de comunicación.

Aunque los dispositivos pueden ser muy diversos dependiendo del tipo de aplicación, sistema, o servicio que se esté desarrollando, todos tienen algunos puntos en común.

4.5.1 Unidad de procesamiento

Independientemente de la aplicación todos los dispositivos poseen alguna unidad de procesamiento más o menos potente, para poder realizar las labores de obtención de información o de actuación. Lo más común es utilizar microcontroladores o circuitos integrados que implementan junto a la unidad de procesamiento las interfaces de comunicación *hardware* más utilizadas hoy en día: I2C, SPI, UART, y USB. Se deben balancear parámetros como velocidad de procesamiento, memoria, tamaño, precio, conectividad, o consumo energético.

Uno de los requisitos de internet de las cosas es que los dispositivos deben ser pequeños, por lo que los procesadores corrientes que se han estado utilizando en ordenadores personales no son compatibles con estas nuevas necesidades. Los puntos clave son el tamaño y el consumo energético, mientras que la velocidad y la potencia quedan en un segundo plano.

A continuación, se presentan las empresas más destacadas dedicadas al diseño de microprocesadores y/o microcontroladores para aplicaciones de internet de las cosas.

ARM

ARM [15] es uno de los diseñadores más famosos e importantes de procesadores actualmente. Posee un enorme mercado en *smartphones* o teléfonos inteligentes, y sus soluciones cumplen perfectamente con las necesidades de tamaño y consumo para este tipo de dispositivos.

Dispone de un catálogo muy amplio de familias de microprocesadores, entre las que destacan Cortex-A, Cortex-R y Cortex-M. Todos los microprocesadores de ARM implementan RISC (*Reducted instruction set computing*, es decir, un conjunto de instrucciones reducido) de 32 bits. Como ocurre normalmente, ARM se encarga del diseño mientras que son terceras compañías las que los fabrican.

El ámbito de uso de cada una de estas familias es el siguiente:

- Cortex-A (de *Application*): Está pensada para aplicaciones que requieren un alto rendimiento, y es capaz de soportar un sistema operativo completo. Por ejemplo, para tabletas digitales, móviles, televisiones inteligentes, etc.
- Cortex-R (de *Real-Time*): Está pensada para sistemas críticos que requieren respuestas en tiempo real mientras mantienen un buen rendimiento. Por ejemplo, para control industrial, robótica, etc.
- Cortex-M (de *Microcontroller*): Está pensada para aplicaciones genéricas de microcontroladores que no requieren de grandes capacidades. Por ejemplo, electrodomésticos, mandos, etc.

Dentro de estas familias la que mejor se adapta a el campo de aplicación de internet de las cosas es la Cortex-M. Esto es debido a que poseen una capacidad de procesamiento más que suficiente para los pequeños dispositivos que se utilizan en internet de las cosas, y además son la familia más pequeña de ARM en cuanto a tamaño de sus procesadores.

Dentro de Cortex-M a su vez existen varias subfamilias: Cortex-M0, Cortex-M3, Cortex-M4 y Cortex-M7. Cada subfamilia es más potente que la anterior y tiene unas características que se adaptan a distintos ámbitos de uso. En la figura 6 se puede observar cual sería la aplicación de éstas y algunas de sus propiedades.

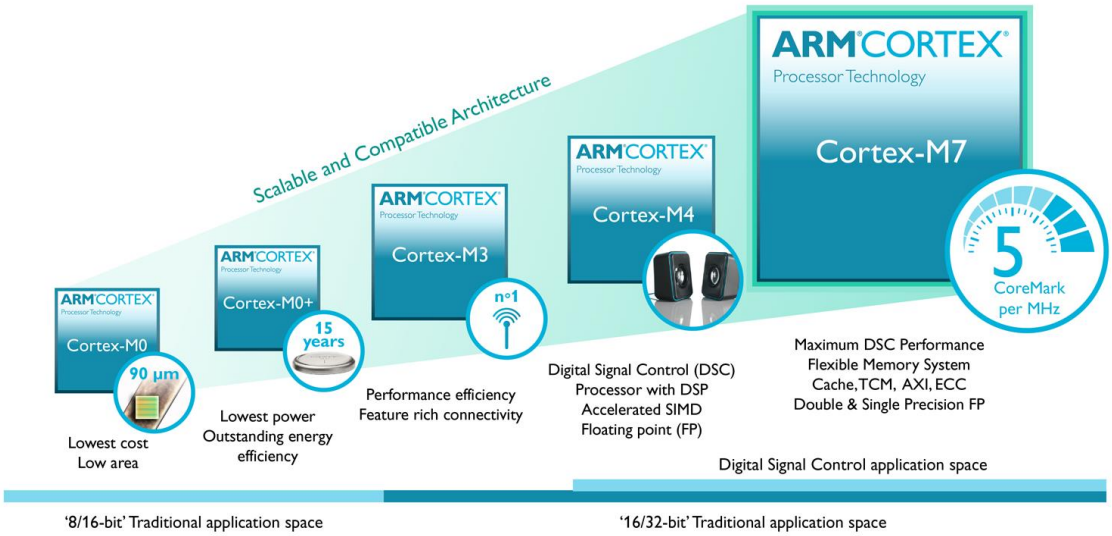

Figura 6 – Familia Cortex-M de ARM. Fuente ARM [15].

INTEL

Intel [16] es sin duda uno de los grandes diseñadores de procesadores, tiene un largo recorrido en la industria y ha ganado una gran popularidad. Sin embargo, sus principales familias de productos están orientadas a los ordenadores personales y a grandes equipos, por lo que sus procesadores destacan más en su rendimiento que en el tamaño o el consumo.

Es por esto por lo que Intel lanzó en 2013 una nueva familia de microcontroladores denominada Quark, formada por una serie procesadores de 32 bits con un tamaño reducido, y que priman el bajo consumo energético frente al rendimiento. Intel se adentró en este nuevo mercado (ya liderado por ARM) por primera vez con estos procesadores, dando el salto hacia el ámbito de internet de las cosas.

Intel no lo ha tenido fácil en este mercado ya que su principal competidor, ARM, le lleva muchos años de ventaja y está muy consolidado en este terreno. Por esto no ha tenido tanto éxito en internet de las cosas comparado con el que tiene con sus procesadores de uso general para ordenadores.

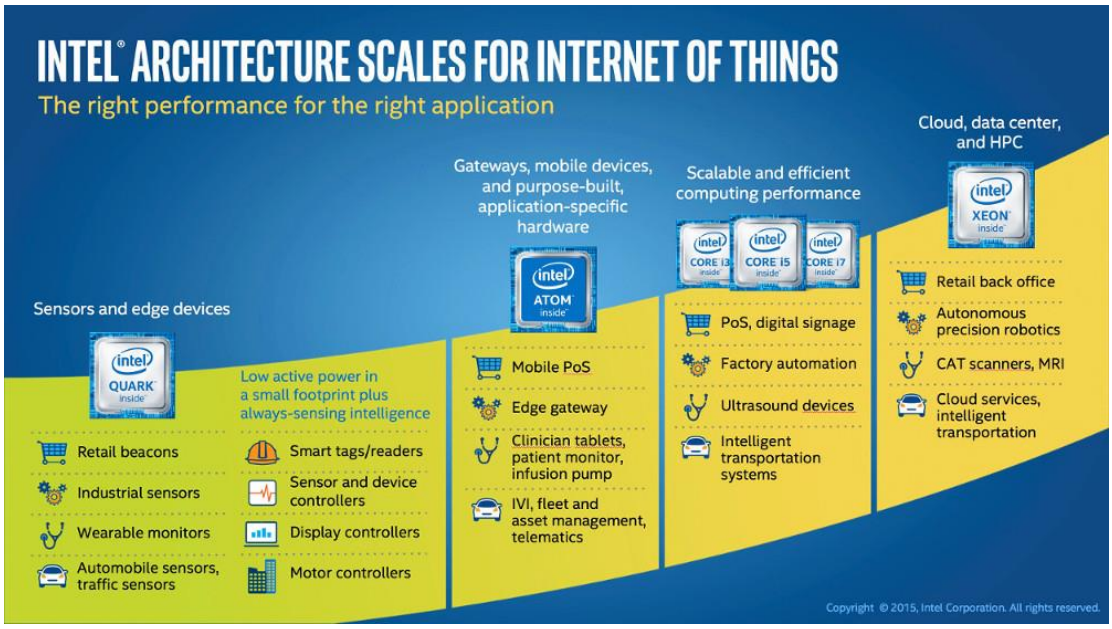

Figura 7 – Procesadores Intel en internet de las cosas. Fuente Intel [16]

En la figura 7 se puede ver lo que Intel ofrece en el ámbito de internet de las cosas. La parte de dispositivos finales u objetos está cubierta con la familia Quark que se había comentado en el párrafo anterior, mientras que para los centros de gestión de datos (se desarrollarán en la sección 4.6) ofrecen otras de las familias más orientadas a un uso general y en el que Intel ya tiene más experiencia, como por ejemplo su famosa familia de procesadores Intel Core.

ATMEL

Junto a grandes compañías como ARM o Intel también destaca Atmel [17] con su famosa familia de microcontroladores AVR. Su diseño simple y las facilidades que aportan han conseguido que estos microcontroladores sean de los más populares entre los usuarios.

Los microcontroladores AVR son también RISC, lo que permite que casi cualquier persona con unos conocimientos básicos de electrónica y programación pueda ser capaz de utilizarlos. El producto estrella que implementa los microcontroladores de AVR es Arduino [18]: se trata de una gama de placas de desarrollo que ha tenido un enorme éxito, atrayendo incluso a personas que nunca habían tenido contacto con ningún microcontrolador.

La principal ventaja de AVR, y por lo que ha tenido tanto éxito, es por su facilidad de uso gracias a librerías simples ya implementadas que permiten abstraerse completamente del *hardware*. Si bien esto no es algo interesante para usuarios expertos, permite que nuevos usuarios se introduzcan en el desarrollo de proyectos con microcontroladores sin necesidad prácticamente de conocimientos previos. Gracias a esto se ha creado una gran comunidad entorno a las placas Arduino, por lo que se ha expandido la cantidad de recursos disponible en forma de librerías, foros de ayuda, proyectos públicos, etc.

También es cierto que los microcontroladores de AVR están más orientados hacia un uso personal, más que para integrarse en grandes productos comerciales como ocurre con ARM en teléfonos inteligentes o Intel en ordenadores personales. Pese a esto tienen cabida en internet de las cosas gracias a que los microcontroladores de AVR no son conocidos por su gran rendimiento y velocidad, pero sí que poseen un tamaño reducido y un consumo energético bajo.

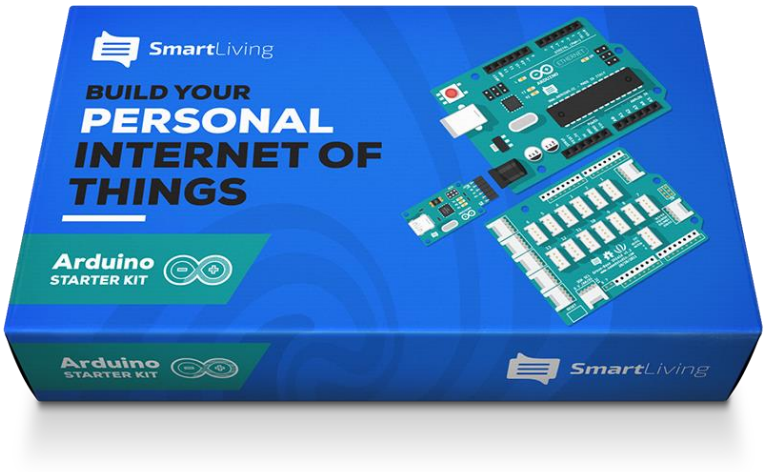

Figura 8 – Kit de desarrollo Arduino para internet de las cosas. Fuente SmartLiving.

Con el aumento de popularidad de las placas Arduino han surgido varios paquetes de desarrollo como el que se muestra en la figura 8. Estos productos tienen como objetivo en la mayoría de las ocasiones llamar la atención del usuario y enseñarle a realizar proyectos sencillos pero interesantes. De esta forma se introduce al usuario en el uso de microcontroladores para pequeñas aplicaciones.

4.5.2 Comunicación

En cuanto a la comunicación, algunos microcontroladores ya incorporan de serie alguna interfaz de comunicación. Sin embargo, los que no las poseen como puede ser el caso de algunas placas Arduino o de placas de desarrollo con procesadores ARM, se les puede añadir módulos adicionales que realicen esta función, los más comunes son:

Módulo *Bluetooth*: Se trata de un módulo muy simple que proporciona comunicación bluetooth a cualquier microcontrolador. El más conocido es el HC-05 (Figura 9), que se controla vía serie:

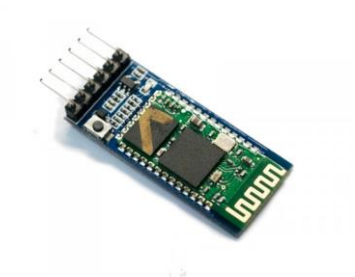

Figura 9 - Módulo HC-05

Módulo *Ethernet*: Este módulo dota de comunicación ethernet al microcontrolador y se controla mediante la interfaz SPI. El módulo mostrado es el ENC28J60 (Figura 10).

Figura 10 - Módulo ENC28J60

Módulo *Wifi:* El módulo Wifi más común es el ESP8266 (Figura 11), que es ampliamente utilizado en microcontroladores. Al igual que el módulo Bluetooth se controla mediante una interfaz serie:

Figura 11 - Módulo ESP8266

Además de estas tecnologías de comunicación ampliamente conocidas, están las LPWAN que se describieron en el apartado anterior. Proporcionan grandes rangos de coberturas y un consumo mínimo, gracias a que requieren poca potencia de uso.

Hay varias placas de desarrollo que integran estas tecnologías, por ejemplo, el "STM32L0 Discovery kit LoRa, Sigfox, low-power wireless" [19] o el Arduino MKR FOX 1200 [20], ambos permiten utilizar la tecnología Sigfox, que se explicará en detalle en el apartado 5. Para el caso de microcontroladores genéricos que no lo integran de serie, existen módulos que actúan como transceptores Sigfox, y se comunican con el microcontrolador a través de interfaces típicas como SPI o serie.

Estos módulos, al igual que las placas de desarrollo mencionadas, vienen de serie con licencias Sigfox para integrarse en la infraestructura, por lo que están listos para ser utilizados. Permiten incorporar la tecnología de Sigfox a prácticamente cualquier microcontrolador, independientemente de sus características, siempre y cuando posea algún tipo de interfaz de comunicación típica con periféricos. Un ejemplo es el módulo "IOTEAM's Single Chip SIGFOX" [21], que puede ser utilizado junto con un microcontrolador que se comunique mediante SPI.

4.5.3 Sensores y actuadores

Finalmente, lo que permite obtener información a cualquier dispositivo es la presencia de sensores y/o actuadores que ayudan al microcontrolador a interactuar con lo que le rodea. Para ello existen varios sensores que se pueden instalar de una manera sencilla, y que utilizan alguna de las interfaces de comunicación estándar con periféricos.

Algunos de los más comunes son: sensores de temperatura, sensores de humedad, sensores de distancia, sensores de luz, acelerómetros, giroscopios, barómetros, etc. Y en cuanto a los actuadores, los más comunes son los motores eléctricos de corriente continua que permiten modificar el entorno, como por ejemplo abrir una puerta, cerrar un cajón, o mover algún objeto.

Algunos ejemplos de sensores y motores típicos son los siguientes:

Figura 12 - Sensor HC-SR04

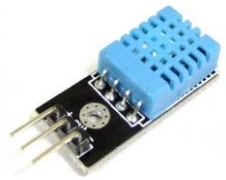

Figura 13 - Sensor DHT11

Figura 14 – Motor DC de 9V

En la figura 12 se muestra un sensor de distancia por ultrasonidos, en la figura 13 un sensor de humedad digital, y finalmente en la figura 14 un motor de corriente continua.

4.6 Gestión de datos

Hoy en día la mayoría de las infraestructuras para internet de las cosas requieren de un conjunto de servicios, ya sea espacio de almacenamiento, disponibilidad o capacidad de procesamiento. Para cubrir estas necesidades en infraestructuras de cierto tamaño es recomendable utilizar lo que se definen como plataformas IoT (*Internet of Things*), que se encargan de aportar todo esto, en adición a los objetos y las comunicaciones.

Una plataforma IoT es la base para que la información de los dispositivos pueda ser tratada de manera correcta y se genere un ecosistema propio. Son plataformas web integradas al internet de las cosas, que manejan grandes cantidades de información. Una plataforma aporta suficientes recursos de cómputo, de almacenamiento etc. Y ofrece un modelo de pago por uso, que se adapta perfectamente a las necesidades de las infraestructuras.

Servicios en la nube

Internet de las cosas genera escenarios totalmente nuevos dentro del entorno de las tecnologías de información, en los que se requiere de altas prestaciones que por sí solos los objetos no pueden mantener dada sus limitaciones. En cambio, si contamos con la computación en la nube disponemos de una tecnología que maneja cantidades prácticamente ilimitadas en capacidad de cómputo y almacenamiento. Por lo tanto, estas dos tecnologías son complementarias entre sí, y se espera que su fusión marque un antes y después en el futuro de Internet.

Este nuevo paradigma, que algunos han denominado *CloudIoT* creará aún mejores escenarios para la industria e investigación de los que actualmente pueden generar por separado. A diferencia de la computación en la nube, IoT tiene contacto directo con el mundo real, y toda la información generada por éste puede ser gestionada y procesada de manera eficiente en la nube. Esto permitirá implementar nuevas soluciones y servicios aplicables a multitud de situaciones en la vida real.

A continuación, se detallarán las plataformas IoT más conocidas.

4.6.1 Sofia 2

Sofia 2 [22] surge de un proyecto europeo de investigación y desarrollo denominado SOFIA que es el acrónimo de *SMART OBJECTS FOR INTELLIGENT APPLICATIONS*. Una vez finalizada la investigación se tomó la plataforma SOFIA original y se desarrolló bajo enfoque empresarial dando como resultado Sofia2**.** Entre sus principales características destaca que es *open-source*, multiplataforma, multilenguaje e independiente de las comunicaciones.

Sofia2 consiste en la unión de un middleware y un repositorio capaz de procesar miles de eventos por segundo, con almacenamiento en tiempo real y *Big Data*, análisis, modelado visual y reglas integradas, todo operable desde una consola Web.

Dispone de un conjunto de documentación, tutoriales, *API's*, etc. donde los desarrolladores pueden obtener información acerca de cómo incluirlo en sus propias aplicaciones.

Entre las principales características de Sofia2 se destacan:

- **Seguridad Integrada:** en todos los niveles y elementos de la arquitectura.
- **Interoperabilidad:** capacidad de operar con diversas soluciones
- **Enfoque *Big Data*:** análisis y procesamiento *Big Data*.
- **Personalizable y extensible:** adaptable a las necesidades.

4.6.2 Fiware

Fiware [23] nace como iniciativa de la Unión Europea a través de la FI-PPP (*Future Internet Public Private Partnership*) que tiene como objetivo avanzar en la competitividad de Europa en las tecnologías de Internet del Futuro.

Esta plataforma maneja el concepto de GE (*Generic Enabler*), que es una librería de servicios de propósito general que cubren funcionalidades comunes en campos como seguridad, almacenamiento, *cloud*, *data context* e IoT. Cada GE puede ser considerado un bloque que contiene un conjunto de API's para la construcción de aplicaciones inteligentes.

Fiware, como plataforma libre, provee varios recursos que van desde documentación académica, tutoriales, API's, y foros donde los desarrolladores puedan entrar en profundidad en los GE.

4.6.3 Microsoft Azure IoT Suite

Azure IoT Suite [24] es un conjunto de servicios y soluciones preconfiguradas en la nube perteneciente a la familia Azure de la multinacional Microsoft. Al igual que la mayoría de los servicios IoT en la nube, dispone de una extensa variedad de servicios de comunicación para dispositivos, así como herramientas de almacenamiento y análisis de la información generada por los mismos.

Algunas de sus características son:

- **Autenticación y Seguridad:** cada dispositivo tiene asociado un identificador y clave que le permite autenticarse con la plataforma.
- **Enrutamiento de mensajes:** Azure IoT permite definir reglas con rutas de mensajes para administrar hacia dónde y desde dónde se están enviando los mensajes.
- **Supervisión de Dispositivos:** opción que permite tener métricas y registros de las actividades y estado de los dispositivos como eventos de conectividad, identidad, *throughput* de mensajes, etc.
- **Bibliotecas para dispositivos:** disposición de SDK's y API's para la mayoría de los sistemas embebidos y lenguajes.

- **Módulos para los diferentes protocolos IoT:** Azure IoT dispone de bibliotecas donde se pueden implementar soluciones en sistemas embebidos usando protocolos como MQTT, HTPP AMQP, etc. de forma nativa.
- **Escalabilidad:** escalabilidad de hasta millones de dispositivos con sus respectivos eventos y propiedades.

4.6.4 Samsung Artik Cloud

Samsung Artik [25] es un proyecto de la multinacional surcoreana Samsung Electronics. Es una plataforma abierta de intercambio de datos para internet de las cosas (IoT), que permite conectar dispositivos y habilitar nuevos servicios y aplicaciones.

Las características fundamentales de la plataforma Artik se describen a continuación:

- **Módulos IoT:** Samsung Artik no solo entrega soluciones en *software* sino también soluciones en *hardware* que pueden ser desplegadas como dispositivos.
- **Interoperabilidad de dispositivos y datos:** Samsung Artik permite que dispositivos puedan comunicarse con otros dispositivos o servicios en la nube. Además, Artik provee las herramientas para poder actuar sobre los dispositivos y ejecutar determinadas acciones, generar alertas o correos.
- **Seguridad y Privacidad:** tanto los módulos ARTIK, dispositivos de terceros y las aplicaciones pueden usar mecanismos de seguridad extremo a extremo. Cada uno de ellos se conecta de manera segura usando el protocolo TLS, elementos seguros y certificados de seguridad.
- **Administración de Dispositivos:** Artik ofrece herramientas web para la monitorización y administración de dispositivos. Estas herramientas permiten observar las propiedades de los dispositivos, sus servicios, ejecución de tareas, etc.
- **Herramientas para el desarrollador:** Artik cuenta con todo un conjunto de API's para recolectar, almacenar y consultar información de los dispositivos. Los desarrolladores pueden construir sus aplicaciones usando los diferentes SDK's que Artik ofrece.
- **Escalabilidad y Latencia:** La plataforma Artik está diseñada para administrar billones de dispositivos sin incrementar la latencia y manteniendo altos estándares de disponibilidad.

4.7 Seguridad

Internet de las cosas se está convirtiendo en un elemento clave para el futuro de internet. Y siendo cada vez más ampliamente utilizado en infraestructuras con tamaño en aumento, está expuesto a todo tipo de ataques. Debe de ser lo suficientemente fiable, con el fin de que emplear esta tecnología no sea un riesgo que correr.

Algunos de los puntos para tener en cuenta son los siguientes:

- **Protección ante ataques DoS/DDOS:** este un tipo de ataque que consiste en denegar el servicio, es decir, impedir que el sistema funcione. Es un ataque muy común en el ámbito de internet y también es aplicable a las infraestructuras del internet de las cosas, por lo que éstas deben de poseer de alguna técnica de defensa que impida que el sistema falle y deje de estar en funcionamiento.

- **Monitorización del sistema:** las infraestructuras tienen que poseer alguna forma de monitorización de su estado, para ser capaces de detectar si están sufriendo ataques, en cuyo caso, tomar las medidas de protección correspondientes.

- **Distintos tipos robustos de control de acceso:** para evitar usuarios no autorizados, y disponer a la vez de un sistema de permisos. Controlar quién puede acceder a qué tipo de información y cuándo lo hace, permite recopilar datos útiles a la hora de actuar ante ataques externos.

- **Independencia de control humano:** el sistema debe auto mantenerse y ser regulado por él mismo, sin necesidad de una gran ayuda externa. Para lograr esto se pueden emplear técnicas de inteligencia artificial como *machine learning*.

Dentro de una infraestructura se distinguen dos fuentes altamente susceptibles a ser atacadas: el *hardware* y las comunicaciones. En cuanto a el *hardware*, éste debe mantenerse fuera del alcance de usuarios ajenos, y además debe estar correctamente configurado para evitar que sea reprogramable o accesible de alguna forma maliciosa.

Y respecto a las comunicaciones, aunque en un principio dependen casi totalmente del protocolo que se utilice, siempre se pueden tomar medidas adicionales, como encriptar manualmente la información que se envía por los canales de comunicación, más allá de la propia seguridad del protocolo.

5. Sigfox

En este apartado se explicará en detalle la tecnología de LPWAN Sigfox y sus características, el cual es motivo de estudio de este trabajo de final de grado. Sigfox es una compañía francesa fundada en 2009 que da nombre a su tecnología de red área extensa y bajo consumo. Es una tecnología pensada para comunicaciones de dispositivos que requieren de un gran alcance (hasta 50 kilómetros en los casos más favorables), a la vez que no están conectados a la red eléctrica y, por lo tanto, necesitan de un tiempo de autonomía muy elevado a cambio de requerir enviar poca información y en intervalos amplios. Sigfox proporciona una red ideal para estos dispositivos, a los cuales las tecnologías de telefonía móvil no se adaptan y resultan inviables.

5.1 Arquitectura

La red de Sigfox se divide principalmente en dos capas (Representadas en la figura 15):

- **Equipamiento, dispositivos y antenas:** en esta capa se encuentran todos los dispositivos encargados de transmitir los mensajes, junto con las estaciones base que los reciben.

- **Sistema de soporte:** es en esta capa donde los mensajes son procesados y se generan los avisos finales a los clientes. También ofrece las funcionalidades para los usuarios (APIs) que permiten a éstos generar acciones de respuesta frente a las llegadas de mensajes, controlar y monitorizar sus dispositivos, herramientas de análisis, etc. Es la capa menos visible pero la que más complejidad engloba.

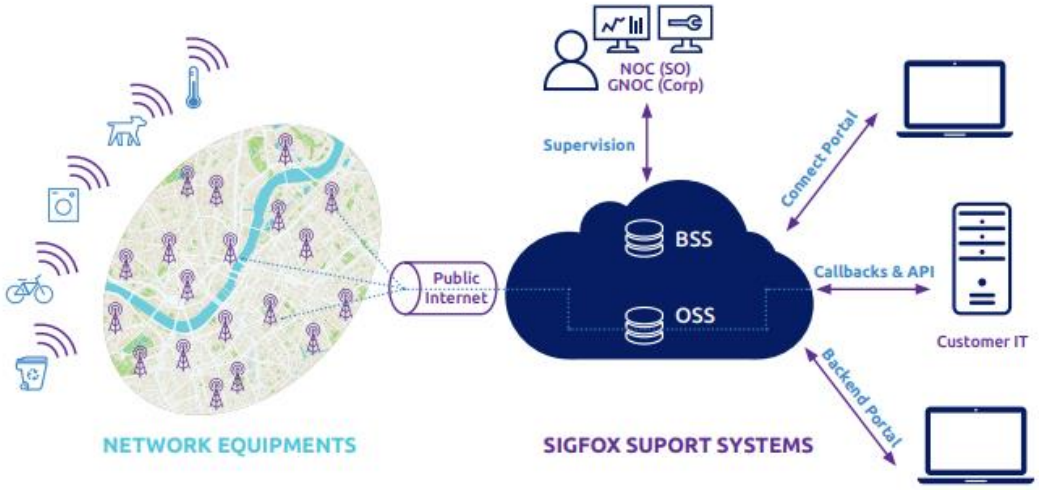

Figura 15 - Arquitectura de Sigfox. Fuente Sigfox [1]

La información es enviada desde los dispositivos hasta las estaciones base de forma inalámbrica. Una vez es recibida las estaciones se encargan de comunicarse con los servidores de Sigfox, generalmente lo hacen usando DSL o 3G/4G [26] como segunda opción.

El *back-end* o servidor se encarga de manejar los mensajes. Es muy probable que el mismo mensaje se reciba varias veces debido a la recepción cooperativa que se explicará más adelante, por lo que el servidor debe asegurarse de únicamente almacenar el mensaje una vez. Además del

mensaje en sí también se almacenan datos adicionales, o *metadata*, asociada al mismo, de forma que los usuarios puedan recuperarla. Esto incluye atributos como la intensidad de la señal recibida, el ruido, etc.

Por último, los usuarios pueden acceder a los mensajes a través de la interfaz web y la API, e incluso definir *callbacks* HTTP frente a los distintos eventos, como por ejemplo la llegada de un mensaje. Esto es útil para usarlo en conjunto de una base de datos propia o de un servicio de *cloud* adicional.

5.*2 Ultra Narrow Band*

El propio nombre de esta tecnología es muy descriptivo. Se basa en emplear canales muy estrechos del espectro electromagnético, del orden de menos de 1 KHz. Transmitir datos en estos canales tiene la ventaja de que apenas se necesita energía y las señales alcanzan grandes distancias en el aire. En la figura 16 se observa una comparación:

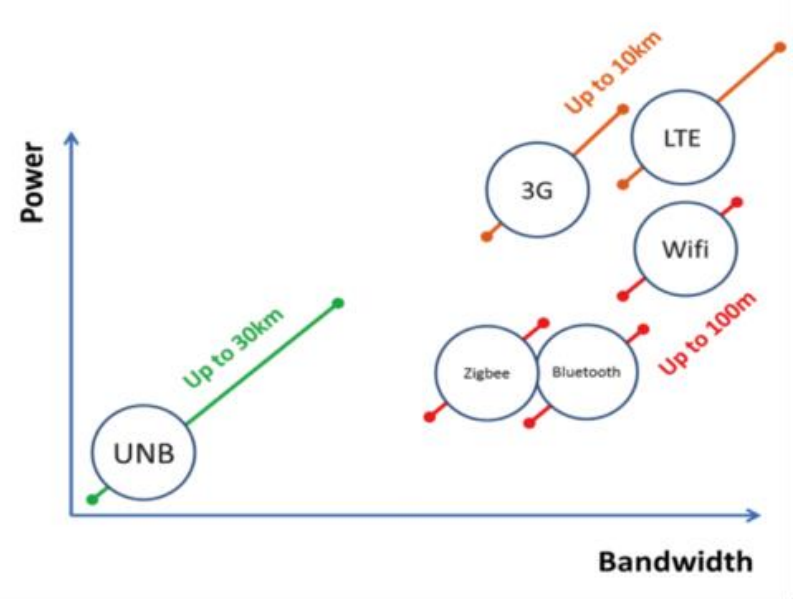

Figura 16 - Comparación Ultra Narrow Band. Fuente Sigfox[1]

Esta tecnología se suele utilizar en comunicaciones de un solo sentido, en el que unos sensores aportan cierta información a unos receptores, comúnmente llamados estaciones base, y no esperan ninguna respuesta. Sin embargo, también es posible implementar una comunicación en ambos sentidos. Debido a que se quiere que el consumo energético recaiga lo menor posible en los dispositivos finales la comunicación es asimétrica, por lo que no se transmite y recibe a la misma velocidad.

Además, otra ventaja que proporciona *Ultra Narrow Band* es que no interfiere con comunicaciones de banda ancha (que utilizan canales más amplios), y es muy resistente al ruido, por lo que es perfecta para obtener bajas tasas de errores de comunicación y pérdida de paquetes. Esto es observable en las siguientes figuras 17 y 18.

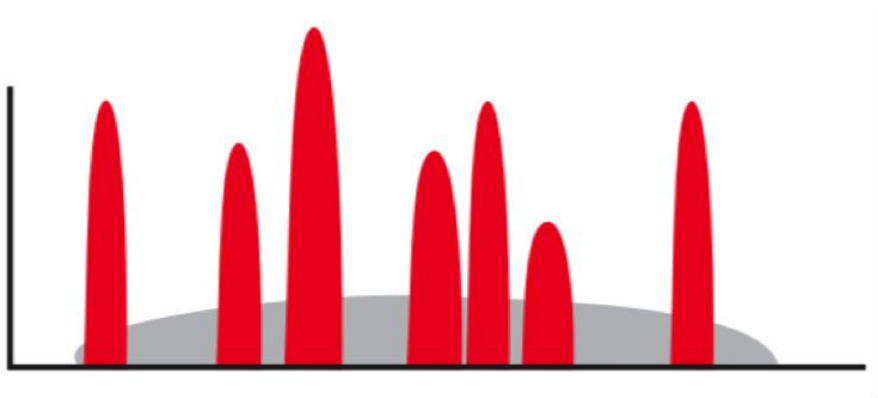

Figura 17 - Señal Ultra Narrow Band. Fuente Sigfox [1]

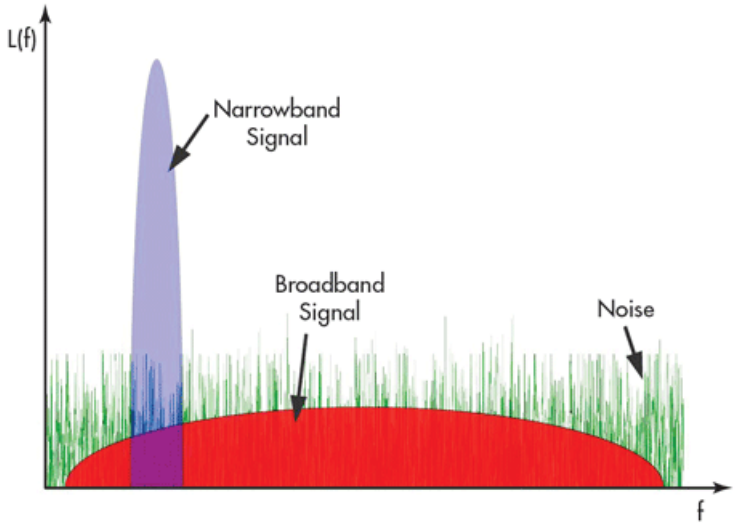

Figura 18 - Comparación de la señal de Ultra Narrow Band. Fuente Sigfox[1]

Sigfox trabaja en el rango de frecuencias de 868 - 868.6 Mhz para Europa y utiliza canales de un ancho de 100 Hz.

5.*3* Acceso aleatorio: protocolo RFTDMA

El acceso aleatorio es una de las claves que hacen que Sigfox tenga una tasa de colisiones tan baja en las comunicaciones. A la hora de enviar información cada dispositivo elige aleatoriamente uno de los canales disponibles, y los datos se envían sin ningún tipo de sincronización entre el emisor y el receptor. Además, también es capaz de enviar hasta dos copias del mismo mensaje en distintos canales, para una mayor robustez. A esto se le conoce como protocolo RFTDMA (*Random Frequency and Time Division Multiple. Access*).

Las estaciones base son las que se encargan de muestrear todo el rango del espectro (868- 868.6 MHz) en busca de señales UNB. Dado que existen del orden de 60000 canales disponibles y cada uno de 100 Hz, la posibilidad de que dos dispositivos cercanos escojan el mismo canal en el mismo instante de tiempo es muy pequeña, por lo que apenas se producen colisiones.

En la figura 19 se muestra un ejemplo de simulación para un número aún más reducido de canales:

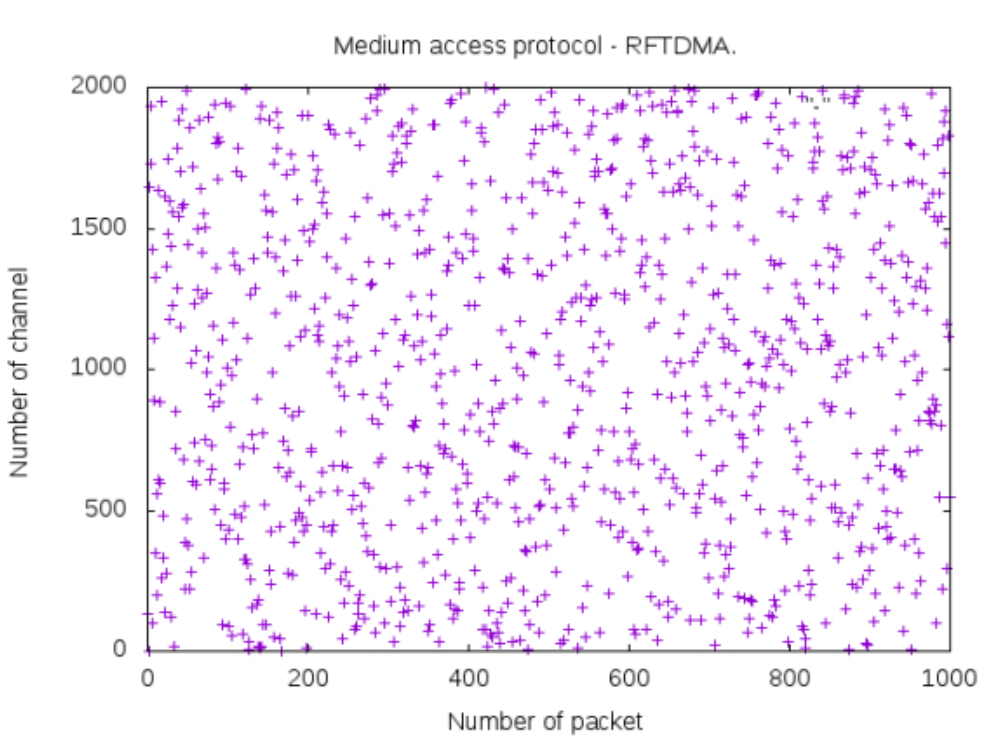

Figura 19 - Simulación del protocolo RFTDMA. Fuente [2]

5.4 Mensajes

El protocolo de Sigfox utiliza mensajes de tamaño reducido, cuya parte de datos va desde 1 hasta 12 bytes. Este espacio es suficiente para almacenar cualquier tipo de valor obtenido de un sensor, de alertas, de GPS, o de otro tipo de información similar, por lo que no se requiere de mensajes de un mayor tamaño. Unos ejemplos son:

- **Coordenadas GPS:** 6 bytes
- **Temperatura:** 2 bytes
- **Estado del dispositivo:** 1 byte

Figura 20 - Formato de los mensajes de Sigfox. Fuente Sigfox [1]

En la figura 20 se observa una representación gráfica de los mensajes en Sigfox. La regulación europea indica que solo se puede utilizar esta banda pública de frecuencias (868 MHz) un 1% del tiempo total. Esto quiere decir que se pueden enviar como máximo 140 mensajes de 12 bytes al

día, y aunque la legislación no es idéntica respecto a esto en todos los lugares, Sigfox ofrece como máximo esta cantidad de mensajes por el momento en todos los países por igual.

5.5 Recepción cooperativa

Los dispositivos no están asociados a una estación base en concreto, al igual que los protocolos de redes móviles, sino que a la hora de transmitir todas las estaciones base dentro del radio de alcance reciben el mensaje. Esto proporciona mucha robustez al sistema, ya que, aunque una estación base falle por cualquier motivo, el resto seguirán siendo capaces de recoger el mensaje y ofrecerlo en el *back-end.*

El término *back-end* hace referencia a la infraestructura que hay detrás del sistema, que incluye la lógica de tratamiento, almacenamiento y procesamiento de la información. En el caso de Sigfox, éste ofrece a través de un portal Web, tras la autenticación adecuada, el acceso a la información que poseen sus servidores, donde muestra a los usuarios información relevante asociada a sus dispositivos. Se desarrollará el *back-end* de Sigfox en el apartado 9.

En la figura 21 se representa gráficamente el concepto de recepción cooperativa.

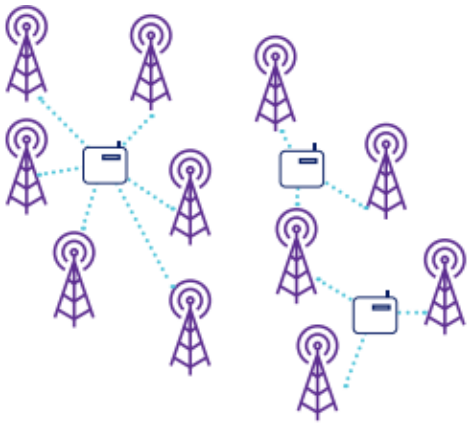

Figura 21 - Recepción cooperativa. Fuente Sigfox[1]

5.6 Modulación

Sigfox utiliza modulación por desplazamiento de fase (PSK) [27]. Esta modulación se caracteriza porque la fase de la señal representa cada dígito de información.

En concreto se emplea la modulación BPSK, la cual es óptima para transmisores de bajo coste que no requieren altas velocidades. Posee dos fases de salida para una sola frecuencia, cada una de 180 grados:

- **Fase 1:** “1” Lógico
- **Fase 2:** “0” Lógico

Un ejemplo de modulación para una entrada digital sería el de la figura 22.

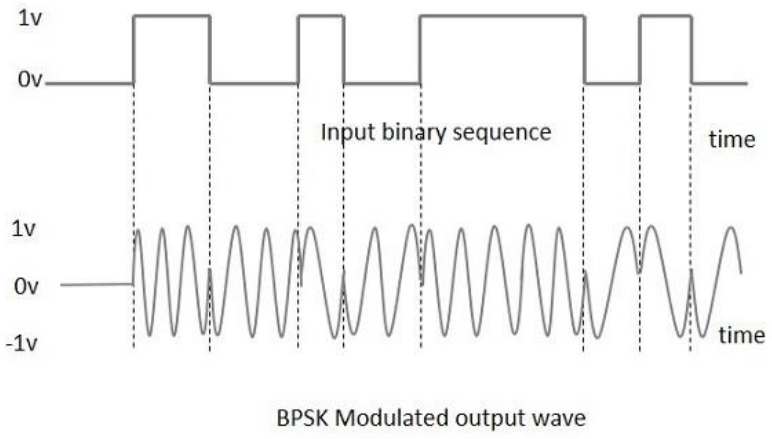

Figura 22 - Modulación BPSK

5.7 Seguridad

Sigfox es una tecnología segura, que emplea mecanismos de seguridad a varios niveles, y que en conjunto convierten a la red Sigfox en una opción confiable para los usuarios.

5.7.1 Seguridad en el procesamiento de mensajes

A la hora de procesar los mensajes Sigfox realiza una serie de comprobaciones previas:

- **Número de secuencia:** este número es una ayuda para evitar duplicación de mensajes. Es simplemente un número de 0 a 255 (un *byte* sin signo) que comprueba el *back-end* de Sigfox cuando se valida un mensaje. De esta forma si se capta un mensaje y se duplica malintencionadamente, éste será automáticamente descartado, ya que el número de secuencia no habrá aumentado y pasará a ser un mensaje no válido.

- **Verificación de MAC (*Message Authentication Code*):** cada dispositivo dispone de una llave simétrica de autentificación que se le otorga a la hora de fabricar el dispositivo, y cada mensaje que se envía contiene una parte cifrada basada en esa clave. De esta forma el *back-end* conociendo el identificador del dispositivo es capaz de descifrar esa parte utilizando la clave simétrica, y así comprobar tanto la autenticidad del dispositivo que ha emitido la señal como la integridad del mensaje en sí. El proceso se representa en la figura 23.

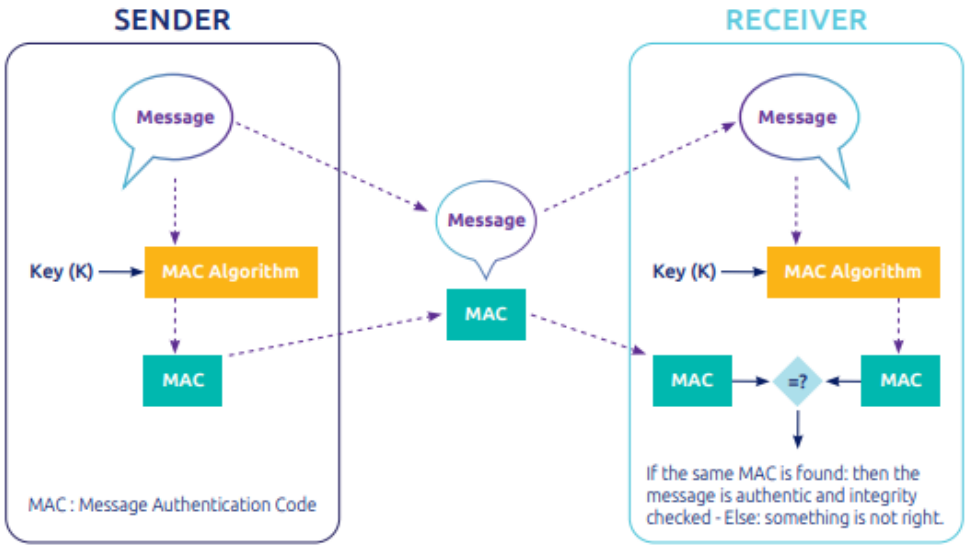

Figura 23 – Verificación de MAC en los mensajes de Sigfox. Fuente Sigfox [1]

- **Cifrado de los mensajes:** por defecto la información se envía sin cifrar. Sin embargo, en algunos casos es necesario que esta información sea totalmente inaccesible y no se puede confiar exclusivamente en la seguridad de la red. Por ello Sigfox ofrece la posibilidad de utilizar su propia función de encriptación, creada en colaboración con CEA-LETI [28], un instituto de investigación francés, que está pensada para mensajes especialmente pequeños, y se deriva de la llave simétrica del dispositivo.

5.7.2 Seguridad en las estaciones base

Por seguridad se han implementado varios mecanismos en las estaciones base con el fin de proteger la integridad del sistema:

- Nadie puede robar el *software* de las antenas.
- Nadie puede alterar el sistema operativo de las antenas, ya que en la secuencia de arranque se comprueba su integridad, y además se realiza un chequeo periódico en tiempo de ejecución.
- Hay una relación entre el sistema operativo y el *hardware*; el sistema solo puede arrancar en una estación base construida por Sigfox.

Adicionalmente las comunicaciones entre las antenas y el *back-end* se realizan a través de una red privada virtual (VPN) que impide el acceso de intrusos.

5.7.3 Seguridad en la creación y reparto de claves

El proceso de creación de claves sólo puede llevarse a cabo si el fabricante posee de los certificados requeridos por Sigfox para comunicarse con la red, y existen diferentes niveles de certificaciones. El proceso es el siguiente:

1. El fabricante envía un email con la petición de una cantidad de identificadores de dispositivos a Sigfox, junto con los certificados.
2. Una vez se ha validado la petición, Sigfox le ofrece un rango de identificadores que pueden utilizar. La *Central Registration Authority* (CRA) crea una *Network*

Authentication Key (NAK) y un *Porting Authorization Code* (PAC). Este último se utilizará más adelante para activar el dispositivo.

3. El fabricante obtiene los ficheros necesarios del CRA: un fichero binario que contiene parejas (identificador de dispositivo, clave) cifrados con AES-ECB [29], y un archivo de texto que contiene parejas (identificador de dispositivo, PAC).
4. El fabricante utiliza las aplicaciones ofrecidas por Sigfox para descifrar y cargar las claves en los dispositivos.
5. El usuario final recibe el dispositivo junto con el identificador del dispositivo y el PAC.
6. El usuario final accede al portal web de Sigfox y registra el dispositivo con estos datos.
7. El *back-end* comprueba estos datos y registra el dispositivo, junto a algunas operaciones adicionales internas.

5.7.4 Seguridad en los centros de datos

Los servidores de Sigfox están protegidos físicamente con protección biométrica para el acceso físico. Además, cada centro está asociado a distintos proveedores de internet. Sigfox posee de una arquitectura balanceada en cuanto a cargas, y todas sus capas están completamente monitorizadas.

Adicionalmente Sigfox utiliza una solución propia para defenderse de ataques tipo DDoS, explicados anteriormente, que detecta estos tipos de ataques y los mitiga. En la figura 24 se muestra una visión general de la seguridad de Sigfox .

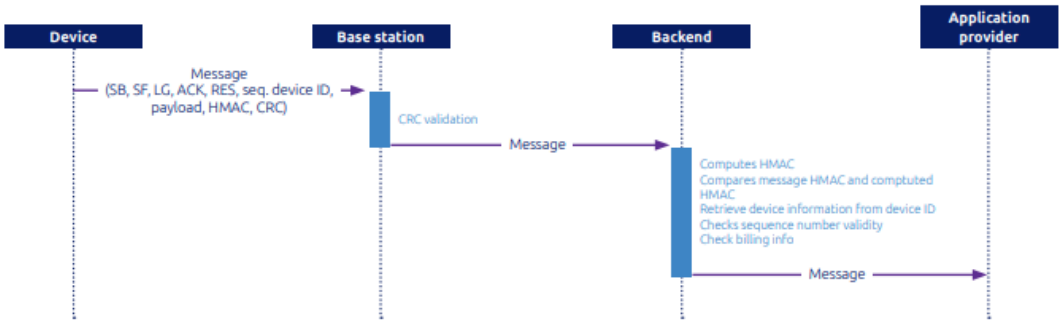

Figura 24 - Esquema de la seguridad de Sigfox. Fuente Sigfox[1]

5.8 Características

5.8.1 Escalabilidad

Sigfox es una red altamente escalable, a la que se pueden conectar millones de dispositivos. Esto es debido gracias a la suma de tres factores que hemos explicado en los anteriores apartados, también representados en la figura 25:

- **UNB, Ultra *Narrow Band:*** permite una gran resistencia frente a las interferencias además del alcance elevado.
- **Acceso aleatorio:** el hecho de que los canales se elijan aleatoriamente ayuda en gran medida a evitar colisiones.

- **Reparto espacial**: los dispositivos están colocados en lugares separados entre sí, donde las antenas que pueden recibir los mensajes son muchas veces distintas. Esto aporta mucha robustez al sistema.

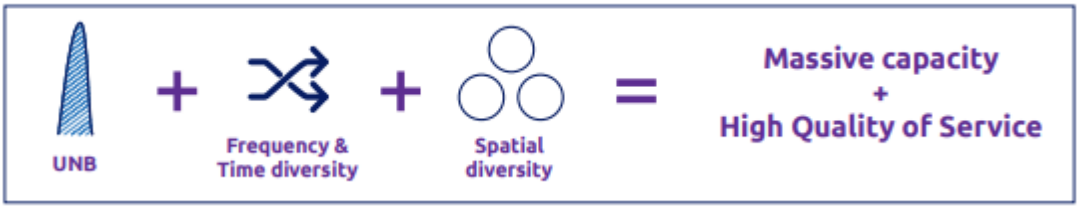

Figura 25 - Escalabilidad de Sigfox. Fuente Sigfox[1]

5.8.2 Alta eficiencia energética

El coste energético, además de ser bajo gracias al UNB también depende del chip empleado. Este coste suele rondar entre 10 a 50 mA en transmisión, y entre 5 y 20 mA en la recepción, teniendo en cuenta que en Europa se permite una potencia máxima de transmisión de 14 dB, mientras que en Estados Unidos es de 22 dB. El tiempo medio en el aire de un mensaje es de 6 segundos.

La mayoría del tiempo los dispositivos se mantienen en estado *idle* o de descanso, en los que no consumen apenas energía, y únicamente se despiertan para realizar transmisiones o para captar información de sus sensores. El coste energético típico se representa en la figura 26.

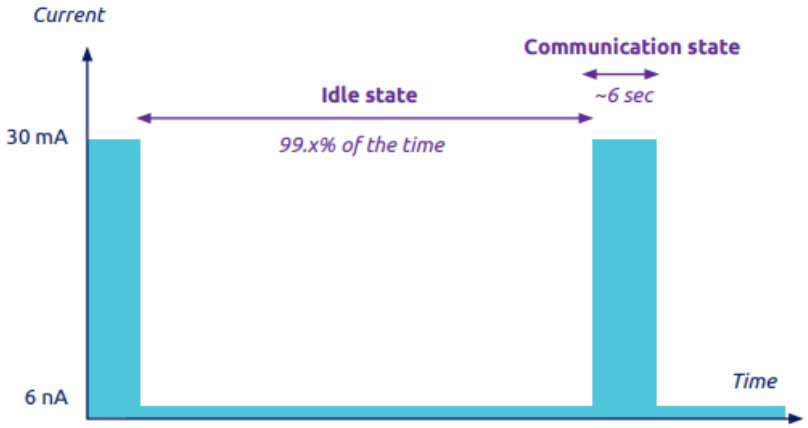

Figura 26 - Consumo energético típico de Sigfox. Fuente Sigfox[1]

5.8.3 Alcance

Esta es una de las mejores ventajas que ofrece Sigfox. La distancia máxima usando esta tecnología se define por varios parámetros:

- La potencia de envío (máximo 14 dB en Europa)
- La sensibilidad de recepción de las antenas
- La topografía del lugar
- Si los dispositivos están en interiores o al aire libre. Este punto es el que menos afecta debido a que usa frecuencias por debajo del GHz, y estas se ven menos afectadas para los casos de interior.

De por sí, Sigfox tiene un buen alcance gracias a UNB y a la baja tasa de transferencia, que es generalmente unos 100 bits por segundo. Posteriormente se presentan los estudios de alcance realizados y su comparación con los valores teóricos.

5.8.4 Cobertura

Actualmente Sigfox es la tecnología orientada hacia el internet de las cosas con la infraestructura que más territorio cubre en el mundo. Es posible consultar su cobertura en todo momento a través de su web [30].

El mapa de cobertura, a fecha de junio de 2018, se muestra en la figura 27, donde el color azul claro representa un lugar donde ya se han desplegado completamente las antenas de Sigfox y el color morado representa que el lugar que todavía está siendo cubierto, pero ya es funcional.

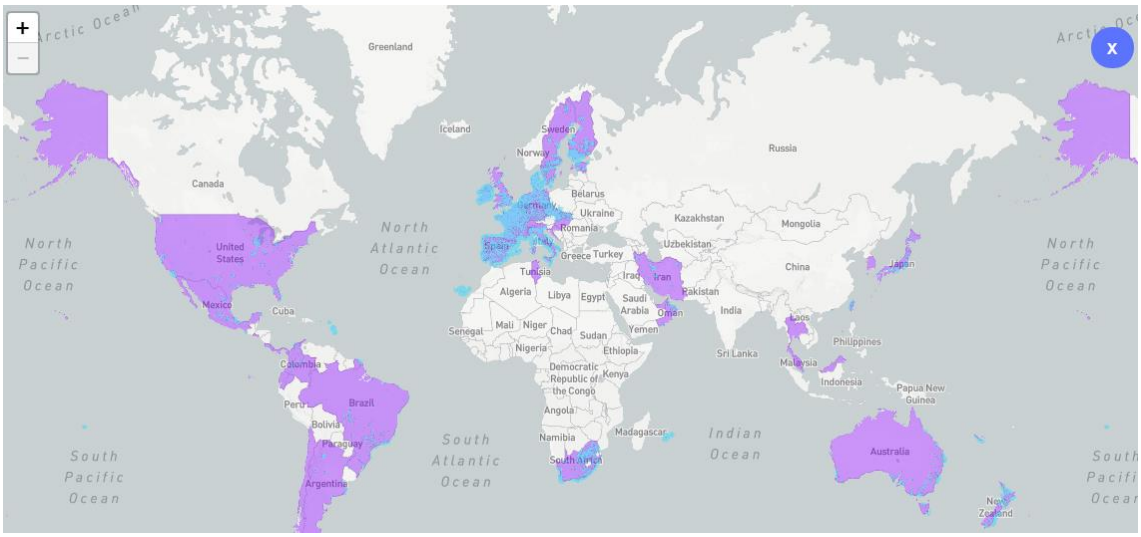

Figura 27 - Cobertura actual de Sigfox. Fuente [30]

5.9 API

El término API hace referencia a *Application Programming Interface*, y se define como un conjunto de funciones y procedimientos que ofrece una biblioteca para ayudar a el usuario final.

En el caso de Sigfox, la empresa ofrece una API que permite que los usuarios, mediante una serie de consultas definidas (a las que se pueden proporcionar parámetros) puedan obtener o editar información con un formato concreto. El formato se refiere al estilo de las respuestas que ofrece el servidor. Algo común es utilizar formatos JSON o XML. Un ejemplo sería una consulta para obtener información sobre grupos, *callbacks* o datos acerca de los dispositivos asociados. Esta API es accesible a través de peticiones HTTP del tipo GET dirigidas a un determinado URL, por lo que se puede hacer directamente desde un navegador web convencional.

Con el fin de que solo los usuarios registrados en Sigfox puedan acceder a estas consultas uno de los parámetros de las mismas debe ser un identificador del usuario, lo que se conoce como un API Key. En Sigfox las API Key tienen asociadas un nombre de usuario, una contraseña y unos permisos. Tanto el nombre como la contraseña son aleatorios, y solo se puede modificar los permisos.

Para obtener una API Key primero se deben seleccionar los permisos que ésta debe tener. Los permisos permiten crear *keys* que solo tengan acceso a una parte de la información y por lo tanto no puedan acceder a todas las consultas. Se pueden elegir los siguientes permisos:

- **DEVICES_MESSAGES:** Permite leer mensajes.
- **LIMITED_ADMIN:** Permisos de administración.
- **ONLINE_HELP:** Permite pedir soporte online.
- **OPT_API_CREATION:** Permite a su vez crear claves API.
- **OPT_DEVICETYPE_READ:** Permite leer dispositivos.
- **OPT_SERVICE_MAP:** Permite acceder al servicio de mapas.

Por ejemplo, para crear una clave API con permisos de administración se seguirían los pasos que se muestran en la figura 28.

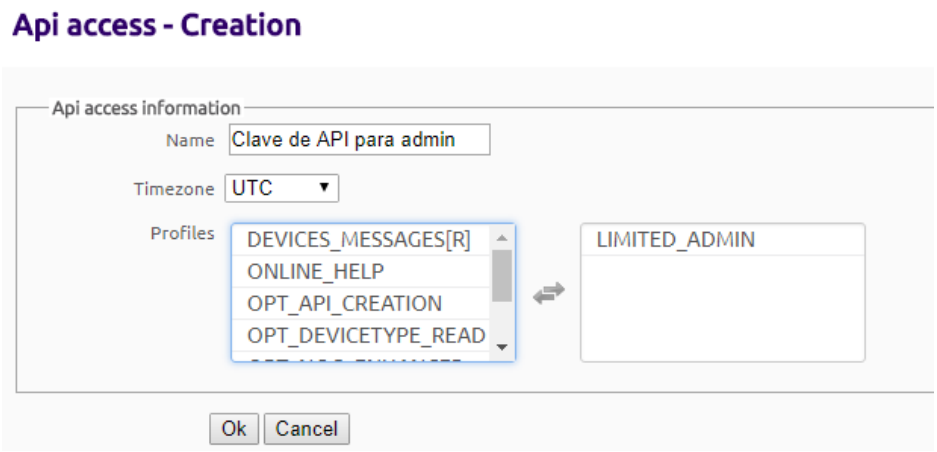

Figura 28 - Creación de una clave para usar el API de Sigfox.

A continuación, se desarrollan cuáles son las consultas que ofrece la API de Sigfox sobre los diferentes puntos:

- **Grupos:** Permite enumerar los distintos grupos disponibles y mostrar la información de un grupo.
- **Dispositivos:** Permite enumerar los dispositivos, mostrar la información de un dispositivo y editar su información, listar dispositivos por tipo, obtener los mensajes enviados por un dispositivo, obtener la localización de un dispositivo, etc.
- ***Callbacks:*** Permite enumerar las *callbacks*, eliminar, deshabilitar/habilitar un *callback.*
- **Cobertura:** Permite obtener información de cobertura sobre un área especificando sus coordenadas (latitud y longitud) con precisión de un grado y obtener información sobre antenas cercanas.

Se ha omitido la sintaxis de las consultas ya que no resulta de interés. Sin embargo, un ejemplo para una consulta simple que solo devuelve el listado de dispositivos activos sería el siguiente:

1. Se accede a la dirección web

   https://backend.sigfox.com/api/devicetypes

2. Se introduce la API Key como se observa en la figura 29.

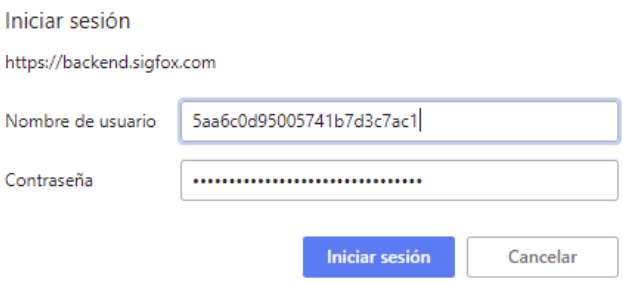

Figura 29 – Solicitud de la API Key

3. Se obtiene la respuesta por parte del servidor, en este caso:

```
{"data":[{"id":"5aa049463c878932a912fe1f","name":"Pycom Upv
kit","group":"5aa049463c878932a912fe1d","description":"Auto created
device type for EVK user : Adrian
Munera","payloadType":"None","contract":"5a706c203c878968fedcd0ff","kee
pAlive":0}]}
```

6. Simuladores de redes de comunicaciones

Cuando se va a diseñar una infraestructura de una red siempre es buena idea averiguar cuál será su comportamiento aproximado antes de desplegarla para verificar que se ajusta a los requisitos. Por ello se han desarrollado simuladores de redes de comunicaciones.

Un simulador es básicamente *software* que emula o predice el comportamiento de un sistema mediante cálculos matemáticos, por lo que apenas requiere de coste más allá del *hardware* de procesamiento. Hoy en día la mayoría de los simuladores de redes soportan las tecnologías más populares, como son: WiFi, Ethernet, LTE, etc. Sin embargo, las tecnologías más recientes aún carecen de un modelo de simulación pensado para ellos, como es el caso de Sigfox. Por esto el objetivo principal de este trabajo de fin de grado es el de aportar un modelo de simulación para la tecnología Sigfox, que permita evaluar sus prestaciones en distintos escenarios.

Por norma general los simuladores de redes permiten a los usuarios definir los dispositivos típicos de redes de comunicaciones, tales como *routers*, antenas, emisores, repetidores, etc. Además, se definen las distancias entre ellos u otros parámetros. Es decir, permiten al usuario describir el escenario concreto que desea simular para una determinada tecnología.

Una vez simulado el escenario se puede ver cuál ha sido el comportamiento de la red. Esto es observable gracias a los datos que nos aporta el simulador sobre una gran cantidad de variables, como: pérdida de paquetes, velocidad de transmisión, fallos, retardos, volumen de datos, etc. El uso de simuladores está muy generalizado en el ámbito de la investigación, ya que aportan información fiable a un coste muy bajo.

Los simuladores de redes se pueden dividir en dos tipos según el método de simulación que utilicen:

- **Simuladores de eventos discretos:** estos simuladores funcionan definiendo una secuencia de eventos en distintos instantes de tiempo. Están pensados para simulaciones de sistemas basados en colas o acciones secuenciales.
- **Simuladores de tiempo continuo:** estos simuladores requieren de complejos modelos matemáticos para describir el comportamiento continuo de un sistema, y deben describir las interacciones entre los componentes con gran detalle. Están pensados para modelar sistemas que cambian muy levemente con el tiempo.

Los simuladores más comunes y utilizados son los que funcionan por eventos, por lo que a continuación se comentará algunos de los más relevantes y sus características.

6.1 NS-2

NS-2 [31] es un simulador de eventos discretos, de código abierto, y ampliamente utilizado en trabajos de investigación. Se utiliza para simular distintas tecnologías de red, tanto inalámbricas como cableadas, permitiendo modificar la topología del sistema. Está programado en C++ y nos ofrece una interfaz de simulación creada en OTcl, un dialecto orientado a objetos de Tcl (Lenguaje de programación de scripts, similar a Python o PERL).

Los usuarios describen topologías escribiendo scripts OTcl, y el programa principal de NS se encarga de simularlas. También se puede visualizar las simulaciones gráficamente mediante un *network animator* NAM, que ofrece acciones a los usuarios tales como detener, reanudar, o avanzar una simulación.

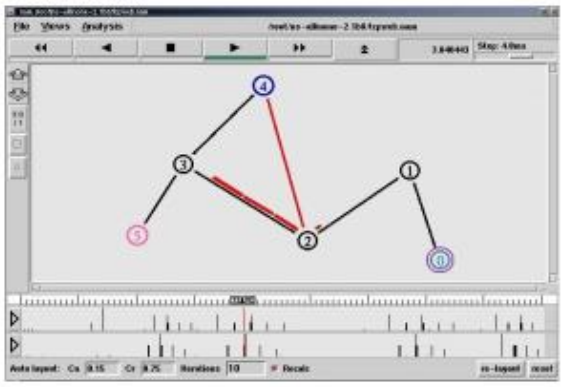

Figura 30 - Network Animator para NS2.

En NS-2 todos lo que ocurre en la red se describe en forma de evento. Estos eventos son almacenados en colas secuenciales y procesados en orden. El tiempo en sí indica únicamente el orden establecido, por lo que la simulación no es en tiempo real, sino que es una simulación virtual.

6.2 NS-3

El proyecto de NS-3 comenzó en 2006 y al igual que NS-2 es de código abierto y está basado en eventos. Sin embargo, NS-3 es mucho más que una actualización de NS-2, es un simulador nuevo, que ya no hace uso de OTcl, y está programado en C++ junto con Python.

Este simulador permite ejecutar varias simulaciones simultáneas, además de tener más características mejoradas. Los *scripts* de simulación se pueden escribir enteramente en C++, e incluso algunas tecnologías se pueden modelar en Python.

Aunque por defecto NS-3 carece de interfaz gráfica se le puede añadir una a la simulación de igual manera que con NS-2:

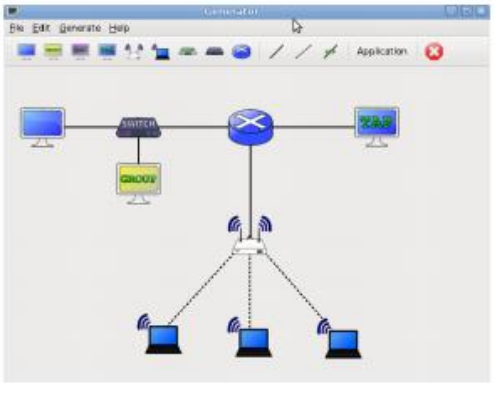

Figura 31 - Interfaz gráfica de la simulación en NS-3.

NS-3 permite generar ficheros PCAP (Interfaz de aplicación para captura de paquetes) que son útiles a la hora de depurar simulaciones, y crea ficheros que son fácilmente leídos e interpretados por otras herramientas como Wireshark [39].

Tanto NS-2 como NS-3 son compatibles con plataformas UNIX y Windows.

6.3 OMNET++

Este simulador de uso libre está disponible desde 1997, y tiene actualmente una gran base de usuarios. A diferencia de NS-2 y NS-3 además de las funcionalidades de simulación de redes también es capaz de modelar multiprocesadores, *hardware* distribuido y sistemas complejos de *software*.

OMNET++ [32] se basa en eventos discretos y tiene una interfaz gráfica completa tanto para modelar las topologías y simular como para visualizar y analizar los resultados. Está pensado para ser usado en un ámbito académico y de investigación, a la vez que intenta ser lo más potente posible. Actualmente está disponible para sistemas UNIX y Windows

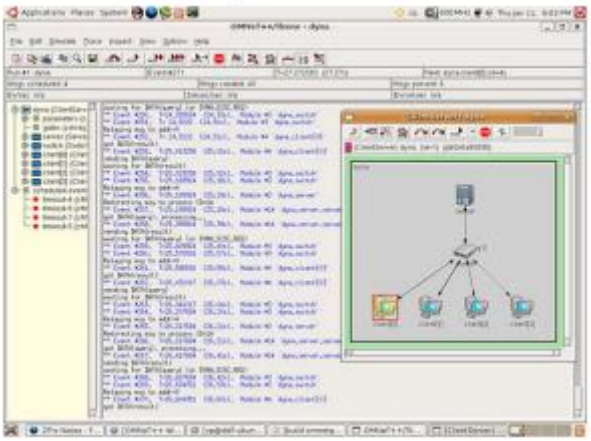

Figura 32 - Interfaz gráfica de OMNET++.

6.4 GloMoSiM

Global Mobile Information System Simulator (GloMoSiM) [33] es un simulador de redes inalámbricas de gran tamaño programado en C. Utiliza al igual que los simuladores anteriores un motor basado en eventos, además de un compilador adicional llamado Parsec para compilar la simulación de los protocolos.

Su principal ventaja es que es capaz de simular topologías grandes con un número de dispositivos del orden de miles, incluyendo dispositivos heterogéneos y con comunicación simétrica. Está diseñado especialmente para redes inalámbricas, y soporta la mayoría de los protocolos actuales utilizados en internet. Sin embargo, tiene como inconveniente que su velocidad de cómputo es algo menor comparada con el resto de los simuladores.

Está disponible en plataformas UNIX y Windows.

6.4 Comparación

Entre los simuladores presentados, el que más destaca actualmente gracias a sus prestaciones es el simulador NS-3. Éste es el simulador más utilizado en trabajos de investigación, debido a que posee un alto rendimiento, un manejo de la memoria eficaz y una facilidad de uso considerable.

Además de sus prestaciones, uno de los motivos principales por los que este simulador es tan utilizado es gracias a que es de código abierto, y permite a los usuarios modificarlo y ampliarlo añadiéndole módulos y funcionalidades extra. Esto es de gran importancia también en los campos de investigación, ya que permite crear modelos de tecnologías de forma sencilla, gracias a las facilidades que aporta el simulador.

Otra de las ventajas es las herramientas y facilidades que incluye el simulador, tales como colas, generación de números pseudoaleatorios, funciones hash, etc. En el siguiente apartado se verán con detalle.

Existen varios artículos de investigación acerca de los distintos simuladores de redes disponibles a día de hoy, y en la mayoría se destaca el simulador NS-3 frente al resto. Un ejemplo es el artículo "*A Performance Comparison of Network Simulators for Wireless Networks*" [41].

7. Simulador NS-3

Como se ha explicado en el apartado anterior, NS-3 es un *software* de simulación de redes para las plataformas Unix y Windows, programado en C++, que está pensado para ser modificado y expandido por los propios usuarios.

NS-3 permite crear redes virtuales mediante nodos, canales, y aplicaciones. Facilita el uso de colas de eventos, generadores de topologías, contadores, variables aleatorias, etc. Todas estas herramientas son muy útiles a la hora de realizar una simulación, y al estar ya implementadas en el *software* los usuarios pueden utilizarlas directamente para crear sus propios modelos.

Tiene una estructura modular, donde cada nivel es independiente del resto y su organización es la mostrada:

<table>
<tr><td colspan="4">ANÁLISIS</td></tr>
<tr><td colspan="4">AUXILIAR</td></tr>
<tr><td>ENRUTAMIENTO</td><td>INTERNET-PILA</td><td>DISPOSITIVOS</td><td>APLICACIONES</td></tr>
<tr><td colspan="2">NODO</td><td colspan="2">MOVILIDAD</td></tr>
<tr><td>GENERAL</td><td colspan="3">SIMULACIÓN</td></tr>
<tr><td colspan="4">NÚCLEO</td></tr>
</table>

Cuando se desarrolla un modelo en NS-3 para una tecnología se han de diseñar algunos de estos niveles, como son el de dispositivos, internet, o auxiliar. Otros niveles son comunes y pertenecen al simulador, por ejemplo, el núcleo, la simulación, o la movilidad.

7.1 Orientación a objetos

Al estar programado en C++ NS-3 es un simulador orientado a objetos, que además utiliza polimorfismo, interfaces, punteros inteligentes, y técnicas de agregación de objetos.

Es por esto por lo que NS-3 es tan eficiente en cuanto a memoria, ya que hace uso de estos punteros, que simulan el comportamiento de un puntero normal de C y además incluyen comprobaciones extras para liberar recursos en memoria cuando no están siendo utilizados. Un ejemplo de creación de un puntero en NS-3 es:

```
Ptr<Node> nodo = CreateObject<Node> ();
```

Esto crea un puntero a un objeto de tipo Nodo, inicialmente vacío, al que posteriormente se le pueden agregar partes como la movilidad, un dispositivo de alguna tecnología, la aplicación que utiliza, etc.

7.2 Núcleo

El núcleo es el nivel más importante del simulador, ya que es aquí donde reside todo el sistema de eventos, de colas, y de tiempo. El sistema de eventos y colas está pensado para asociarlo a *callbacks* o llamadas de retorno, que se ejecutan cuando un evento es activado.

El tiempo de simulación se almacena en una variable entera de 64 bits, en unidades de nanosegundos. El usuario no accede directamente a esta variable, sino que debe utilizar las funciones de la clase simulador, que permite fácilmente trabajar con el tiempo y realizar operaciones aritméticas con él. También incluye funciones para realizar conversiones entre unidades de tiempo y definir rangos.

Para poner en cola un evento se ha de utilizar el método *Schedule* de la clase simulador, al cual se le pasan como parámetros el tiempo en el que se ejecutará el evento, un puntero a la función que debe procesar, y argumentos para ésta.

Un ejemplo sería:

```
EventId id = Simulator::Schedule (Seconds (10.0), &MiEvento, 3.1415);
```

Esta línea de código define y encola un evento en el simulador, en el instante de tiempo de 10 segundos, que debe ejecutar la función MiEvento y se le debe pasar como argumento el número PI.

Este evento también se puede cancelar mediante el método *Remove*:

```
Simulator::Remove (id);
```

Una vez definidos los eventos únicamente hay que llamar a la función *Run*, que se encarga de comenzar la simulación:

```
Simulator::Run ();
```

El aspecto que tendría la cola de eventos sería algo parecido a lo que representa esta figura, donde se han definido distintos eventos en varios instantes de tiempo:

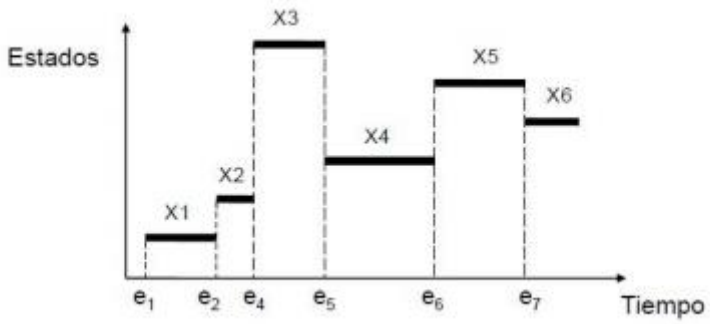

Figura 33 - Eventos en NS-3

A continuación, se mencionan otras de las funcionalidades que posee el núcleo del simulador NS-3.

7.2.1 Variables aleatorias

Las simulaciones utilizan una gran cantidad de números aleatorios. Un estudio demostró que la CPU gasta la mitad del tiempo en una simulación generando estos números, y la calidad de la aleatorización influye directamente en la calidad de los resultados.

NS-3 dispone de un generador de números pseudo-aleatorios. Esto es de vital importancia porque para que un simulador funcione correctamente debe ser capaz de emular la aleatoriedad de determinados eventos y condiciones. Por defecto NS-3 utiliza una secuencia de números aleatorios generados, que están relacionados con un valor fijo o lo que se conoce como *seed* o semilla. Para un mismo valor las semillas siempre producen la misma secuencia de números. Sin embargo, en el simulador NS-3 se puede elegir una semilla distinta por ejecución.

Todos los números aleatorios provienen de una instancia del objeto de ns3 denominado *RandomVariableStream (ns3::RandomVariableStream).* Y el objeto que se encarga de escoger la semilla a utilizar para generar los números aleatorios es *RngSeedManager (ns3::RngSeedManager).*

Para obtener números aleatorios dentro de NS-3 se ha de crear el objeto *UniformRandomVariable:*

```
Ptr<UniformRandomVariable> x =CreateObject<UniformRandomVariable> ();
```

Entonces se puede acceder a la secuencia de números, por ejemplo:

```
myRandomNo = x->GetInteger ();
```

7.2.2 Funciones *hash*

El simulador NS-3 dispone de una serie de funciones *hash* de uso general. Las funciones hash se encargan de resumir un objeto cualquiera en una serie de caracteres de tamaño menor, de forma que sea imposible obtener el propio objeto partiendo del resumen. A la vez tiene que garantizar que objetos distintos generen resúmenes distintos.

Para ello NS-3 generalmente utiliza resúmenes de 32 o 64 bits representados mediante caracteres. Obtener un hash de un objeto es tan simple como emplear las funciones para ello incluidas en el simulador, tales como:

Hash32 (buffer, buffer_size) o *Hash64(buffer, buffer_size)*

7.2.3 Logging

El simulador NS-3 es capaz de monitorizar todo lo que ocurre en el transcurso de una simulación, generando un fichero a modo de registro legible (o simplemente mostrándolo por pantalla) que permita saber qué ha ocurrido tras completar la simulación. Esta opción se puede activar/desactivar simplemente mediante la variable de entorno *NS_LOG*.

A la hora de crear un modelo de simulación siempre resulta conveniente añadir ciertos mensajes de *log* que el usuario pueda utilizar para detectar qué ocurre internamente dentro del modelo. Por

ejemplo, para avisar que cierto evento se ha lanzado, que un dispositivo ha rechazado un mensaje, que ha habido un cambio de estado no esperado, etc.

Además, NS-3 define distintos tipos de mensajes dentro del *log*. Para filtrar los mensajes según su significado la variable *NS_LOG* puede tomar los siguientes valores:

- ***LOG_NONE:*** la simulación no muestra ningún mensaje de *log*.
- ***LOG_ERROR:*** se muestran mensajes de errores.
- ***LOG_WARN:*** se muestran mensajes de advertencia.
- ***LOG_DEBUG:*** se muestran mensajes de *debug*.
- ***LOG_INFO:*** se muestran mensajes informativos.
- ***LOG_FUNCTION:*** se muestran mensajes de seguimiento de funciones.
- ***LOG_LOGIC:*** se muestran mensajes del flujo de la lógica.

Estos valores son progresivos, es decir, cada posible valor de *NS_LOG* genera también mensajes correspondientes a niveles inferiores de depuración. Es decir, si se asigna a la variable *NS_LOG* el valor *LOG_INFO*, también se mostrarán los mensajes de *debug*, de advertencia, y los de errores. Es responsabilidad del creador de un módulo definir bien todos los mensajes de *log* que se usen en su modelo, así como su tipo de mensaje, si es el caso de querer incluir esta información.

Finalmente, existen otras formas en las que un usuario puede indicar el nivel de *log* que quiere usar, además de la variable de entorno. Por ejemplo, puede hacerlo mediante las funciones *LogComponentEnable (LOG_LEVEL)* dentro del propio *script* de simulación que se haya definido.

7.2.4 *Helpers* o auxilares

Los *helpers* son unos de los objetos más útiles dentro de NS-3 para los usuarios. Básicamente se encargan de automatizar el proceso de crear *scripts* de simulación, de forma que si el usuario no está familiarizado con el modelo que va a utilizar (por ejemplo, el que se crea en este trabajo) le ayude a crear los dispositivos correctos, a definir las variables, los canales, las direcciones, etc.

Los investigadores que crean un módulo para NS-3 generalmente definen un objeto *helper* que ayuda a crear los dispositivos de esa tecnología, estos *helpers* suelen tener funciones de nombre común que al usuario le resulten conocidas, como la creación de nodos. De esta forma, aunque dentro de esa función cada modelo realice distintas tareas, el usuario puede simplemente no preocuparse por entender que hay por debajo y usar el *helper* como una interfaz común a todos los módulos. Por eso es interesante que todos los *helpers* de las tecnologías se asemejen.

7.3 Características generales

Algunas de las características que definen al simulador NS-3 son:

- **Extensibilidad:** NS-3 es un simulador de código abierto, que permite a los usuarios mejorarlo añadiendo módulos o mejorando los ya existentes. Gracias a su modelo orientado a objetos es sencillo programar nuevas funcionalidades para el simulador.

- **Reusabilidad del código:** gran parte del código de NS-2 es adaptable fácilmente a este nuevo simulador, gracias a las técnicas de herencia y de módulos extensibles.

- **Configuración en tiempo de ejecución:** permite a los usuarios redefinir valores y tipos de clases sin necesidad de recompilar el simulador, siendo posible modificarlas desde línea de órdenes.

- **Seguimiento:** gracias a los *callbacks* o llamadas de retorno se puede conocer el estado en todo momento de cada componente de la simulación, además de poder realizar un post-procesamiento de las trazas de ejecución de las simulaciones

- **Escalabilidad:** NS-3 posee una alta escalabilidad permitiendo trabajar con simulaciones de grandes dimensiones

7.4 Ejemplo de *script* en NS-3

El objetivo de este apartado es mostrar cómo se crea un *script* simple en NS-3. Para ello simularemos una red Ethernet. Aunque se podría emplear el modelo creado como objetivo de este trabajo para el ejemplo, su complejidad no lo hace recomendable para ilustrar de forma general cómo funciona el simulador con un *script* sencillo.

En primer lugar, todo *script* debe declarar el espacio de nombres de NS-3.

```
using namespace ns3;
```

Después, dentro de la función principal *main*, se declaran los mensajes de *logs.*

```
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);

LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

Y se crean los dispositivos o nodos, mediante el *helper* llamado *NodeContainer*.

```
NodeContainer nodes;

nodes.Create (2);
```

Posteriormente se define el tipo de canal. En este caso se utilizará un canal punto a punto.

```
PointToPointHelper pointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate",
StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay",
StringValue ("2ms"));
```

A continuación, se crean los dispositivos de red en los que se instalan los nodos creados anteriormente, y se les asigna unas direcciones IP. También se instala el protocolo de red en los nodos.

```
NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);

InternetStackHelper stack;

stack.Install (nodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

El último paso es crear las aplicaciones, colocarlas en los dispositivos y asociarles una función. En este caso se creará un nodo servidor *echo* (que devuelve los mensajes) en el puerto 7 y un nodo cliente, que envía mensajes de 1024 bits con intervalos de un segundo. Las aplicaciones se almacenan en el *helper* llamado *AplicationContainer*. La aplicación del nodo cliente comenzará a enviar mensajes en el segundo 2 de la simulación y parará en el segundo 10. Estos tiempos corresponden a la escala de la simulación, no al tiempo real de ejecución de la misma.

```
UdpEchoServerHelper echoServer (7);

ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 7);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));
```

Finalmente, para iniciar la simulación se ha de llamar al método *Run* del simulador.

```
Simulator::Run ();
```

8. Modelo NS-3 de Sigfox

Con el fin de evaluar las características de Sigfox se ha desarrollado un modelo en NS-3 capaz de simular la tecnología y obtener información respecto al consumo de potencia, alcance o la tasa de errores. El uso de un modelo de simulación permite el análisis de situaciones reales, pudiendo así conocer en detalle el comportamiento real del protocolo, y valorar los resultados obtenidos en la simulación.

Este apartado se centra, en primer lugar, en explicar la estructura general del modelo de simulación realizado, seguido del desarrollo de todos los módulos que la componen. Cada apartado se centra en un módulo en concreto, en el que se realiza una breve explicación de la funcionalidad de éste, así como detalles de la implementación.

Como se ha mencionado anteriormente parte de este modelo surge de un modelo anterior más básico desarrollado por un alumno de máster [2]. El módulo se centraba en la capa física, dentro de cada sección se describe la ampliación y mejoras, además de las nuevas funcionalidades implementadas.A continuación, se presenta la estructura general del modelo.

8.1 Estructura

El modelo implementado, de acuerdo con la norma general del simulador NS-3, se divide en una serie de módulos que desarrollan cada una de las funcionalidades de las diversas partes independientes del modelo. Un modelo de simulación debe definir cómo se comporta a diferentes niveles, por ejemplo, al nivel físico o al nivel lógico. Estas partes son independientes entre ellas pese a que deben comunicarse, lo que permite dividir la implementación en varias capas. En este apartado se describe la estructura elegida.

La estructura se divide principalmente en dos partes, la primera, dedicada a la capa física se encarga de describir los detalles de bajo nivel del protocolo. Aquí se incluyen puntos como la decodificación y modulación de las señales, el espectro electromagnético, el cálculo de la tasa de errores, o el consumo de potencia.

La segunda parte define la capa MAC (*Medium Access Control*). Se encarga de manejar (como su propio nombre indica) el acceso al medio, es decir, al canal de comunicación. Para ello define una cola de envío de mensajes, así como los detalles de creación de un paquete de red, en el que se añaden además de los datos, unos campos adicionales, conocidos como cabecera y cola. A continuación, en el diagrama de la figura 34 se pueden observar detalladamente los distintos módulos definidos en el modelo de simulación y su relación entre ellos:

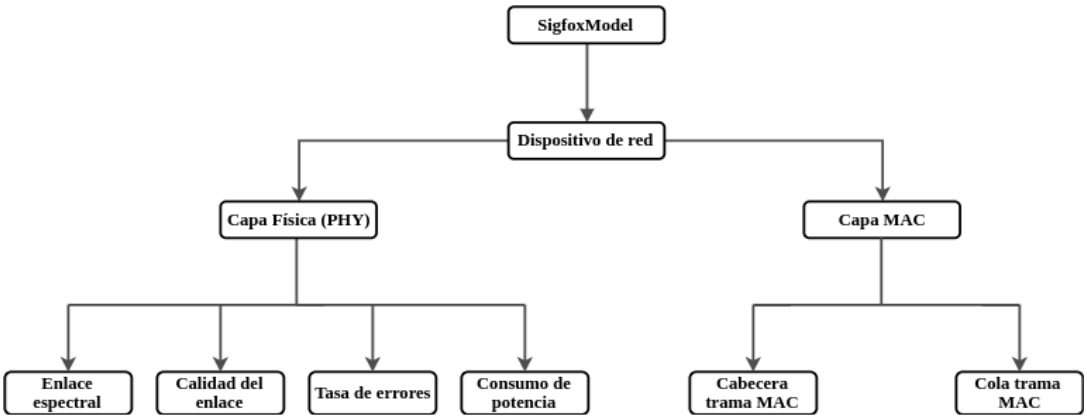

Figura 34 - Estructura del modelo de simulación.

Los módulos definidos son los siguientes:

- **Capa Física (PHY):** Modela el funcionamiento de la capa física. Se divide en dos partes, capa física de los dispositivos y capa física de las estaciones base.
- **Enlace espectral:** Modela las características del enlace de comunicación inalámbrico.
- **Calidad del enlace:** Evalúa la calidad del enlace de comunicación.
- **Tasa de errores:** Evalúa la tasa de errores en la comunicación.
- **Consumo de potencia:** Evalúa el consumo de potencia del dispositivo de red.
- **Capa MAC:** Modela el funcionamiento de la capa MAC (*Medium Access Control*). Se divide en dos partes, capa MAC de los dispositivos y capa MAC de las estaciones base.
- **Cabecera de la trama MAC:** Permite crear la cabecera de los paquetes de comunicación.
- **Cola de la trama MAC:** Permite crear la cola de los paquetes de comunicación.
- **Dispositivo de red:** Permite la implementación de dispositivos Sigfox.

Todos estos módulos se implementan en los archivos que se muestran en la figura 35. En comparación con el modelo inicial, el número de ficheros se ha duplicado, , al incluir en el nuevo modelo las estaciones base y no solo los nodos finales como ocurría de forma incompleta inicialmente. Cada fichero ".c" tiene su correspondiente archivo de cabecera ".h", como es habitual en el lenguaje de programación C y C++.

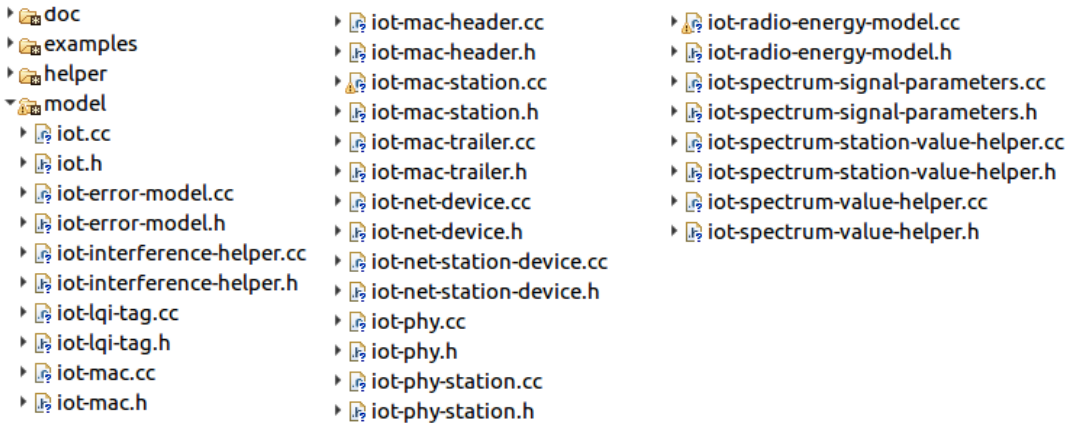

Figura 35 – Ficheros del modelo de simulación.

8.2 Modelo de propagación Okumura-Hata

Con el fin de simular el desgaste de la señal por el aire se usa el modelo de propagación de Okumura-Hata [34], el cual ya está implementado por el propio NS-3. Este modelo es uno de los más empleados y contrastados actualmente, y es válido en el rango de frecuencias de 150Mhz a 1500Mhz. Es capaz de simular diferentes entornos, principalmente los entornos rurales, suburbanos, y de ciudad.

La ecuación de pérdidas para un entorno urbano es la siguiente:

$$L_U = 69.55 + 26.16 \log_{10} f - 13.82 \log_{10} h_B - C_H + [44.9 - 6.55 \log_{10} h_B] \log_{10} d$$

Figura 36 – Ecuación de pérdidas Okumura-Hata para un entorno urbano.

Donde:

- **f:** Representa la frecuencia de la señal en Mhz.
- **hB:** Representa la altura de la antena receptora.
- **hM:** Representa la altura de antena transmisora.
- **Ch:** Representa el factor de corrección de la altura de la antena receptora dependiendo del tipo de entorno.
- **d:** Representa la distancia entre el emisor y el receptor en Km.

Para los entornos urbanos con ciudades pequeñas Ch toma el valor de:

$$C_H = 0.8 + (1.1 \log_{10} f - 0.7) h_M - 1.56 \log_{10} f$$

Figura 37 – Factor de corrección Okumura-Hata para ciudades pequeñas.

Y para ciudades grandes:

$$C_H = \begin{cases} 8.29 (\log_{10}(1.54 h_M))^2 - 1.1 & \text{, if } 150 \le f \le 200 \\ 3.2 (\log_{10}(11.75 h_M))^2 - 4.97 & \text{, if } 200 < f \le 1500 \end{cases}$$

Figura 38 - Factor de corrección Okumura-Hata para ciudades grandes.

Las ecuaciones de pérdidas de las áreas suburbanas y las áreas rurales dependen directamente de la ecuación del entorno urbano (Lu). Para un entorno suburbano:

$$L_{SU} = L_U - 2\left(\log_{10} \frac{f}{28}\right)^2 - 5.4$$

Figura 39 – Ecuación de pérdidas Okumura-Hata para un entorno suburbano.

Lsu: Índice de pérdidas en entornos suburbanos.

Y finalmente para los espacios rurales:

$$L_O = L_U - 4.78 (\log_{10} f)^2 + 18.33 \log_{10} f - 40.94$$

Figura 40 – Ecuación de pérdidas Okumura-Hata para un entorno rural.

Lo: Índice de pérdidas en entornos rurales.

8.2 Capa física

La capa física es uno de los módulos más importantes y que más define a una tecnología. En este caso es de las partes más complejas del modelo de simulación, ya que necesita tener en cuenta el estado actual de los dispositivos, además de recibir y enviar información por el canal de comunicación y aportar información sobre el consumo energético.

El módulo de la capa física se encarga de codificar las señales, permitiendo de esta forma que se puedan entender las tramas enviadas por el canal de comunicación. En el modelo implementado la capa física es la que recibe y emite las señales por el canal conectado al dispositivo físico. Para ello utiliza el modelo de propagación presentado en el apartado anterior, que permite simular la atenuación de la señal a través del espacio y producir pérdidas de paquetes.

En el caso de recepción, la capa física decodifica la señal entrante, y avisa a la capa MAC del resultado, ya sea transfiriendo el paquete o comunicando su pérdida. En la emisión es la capa MAC la que empieza la rutina, indicando a la capa física que envíe un paquete. En ese caso la capa física añade una serie de parámetros a la señal, junto con el paquete a enviar, y lo redirige al canal de comunicación.

En el modelo inicial del cual parte este trabajo esta capa estaba orientada a comunicaciones entre dos únicos dispositivos. Solo eran capaces de recibir una señal a la vez, por lo que no tenían en cuenta ningún tipo de interferencia con otras señales, más allá del ruido. Esta simplificación no es aceptable en casos reales, ya que normalmente casi todos los escenarios presentan más de un dispositivo. Además, también existen estaciones base aparte de los dispositivos, que tienen diferente comportamiento a este nivel físico. Por ello ha sido necesario crear de cero la capa física de las estaciones base de Sigfox.

- **Capa física de dispositivos finales o *end points*:** En este caso la capa física se ha diseñado como una máquina de estados, que permite conocer el estado actual de esta capa y manejar los eventos de envío y recepción. Se distinguen los siguientes estados:

    - ***Sleep*:** En este estado el dispositivo se encuentra totalmente inactivo, siendo incapaz de enviar o recibir información por el canal. Es el estado por defecto y el que más tiempo ocupa.
    - **Transmisión (TX_ON):** Se establece al enviar información por el canal, indicando que el dispositivo está ocupado en esta tarea. Es el estado con más consumo energético.
    - **Recepción (RX_ON):** Se establece tras finalizar el envío de información por el canal, es decir, al finalizar el estado de transmisión. El estado de recepción se mantiene únicamente durante un breve tiempo determinado, unos seis segundos, tras los cuales si no se ha recibido nada se evalúa la cola de envíos sin esperar más tiempo, y en caso de que este vacía se entra en el estado *sleep*. Si no, se volvería nuevamente al estado de transmisión.

    Cabe destacar que la transición de estos estados no es instantánea, sino que requiere de un pequeño tiempo para ello. El diagrama de flujo se representa en la figura 41.

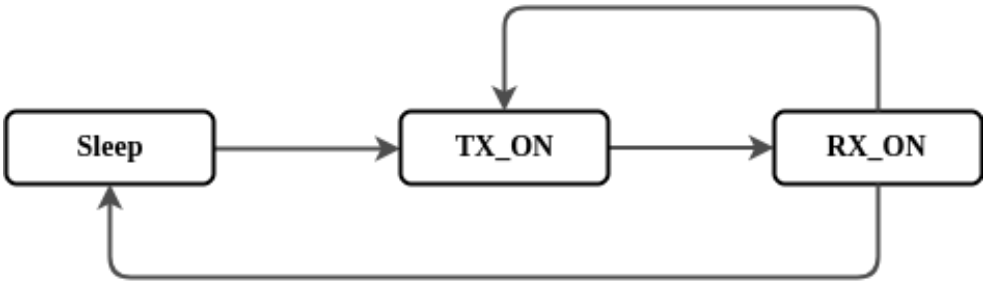

Figura 41 - Diagrama de flujo de la capa física.

- **Capa física de las estaciones:** La capa física para simular antenas funciona de una forma totalmente distinta. No dispone de ningún estado que la defina, sino que las antenas pueden enviar y recibir información en cualquier momento, de forma concurrente e ilimitada. Se ha adoptado este enfoque con el objetivo de asemejarse lo máximo posible al funcionamiento normal de una antena Sigfox real.

El diagrama de estados que existía en el modelo inicial era completamente erróneo, ya que el dispositivo únicamente realizaba un cambio de estado a la hora de enviar o recibir un paquete. Esto provocaba que si el dispositivo recibía un mensaje mientras estaba enviando otro toda la simulación fallase. Como se ha mencionado antes, el modelo estaba diseñado únicamente para dos dispositivos, un emisor y un receptor, que interactuaban en un entorno ideal.

El funcionamiento de la capa física en general es el siguiente:

En el caso de recepción se calcula si la señal procede de una antena o de otro dispositivo (esto es posible debido a que las antenas utilizan una frecuencia distinta, existiendo las frecuencias de *uplink* y de *downlink*). Además, se calcula el canal y se suma la señal al conjunto de señales actuales. Si el dispositivo se encuentra en estado de recepción la señal se marca como válida. Seguidamente se encola el evento de finalización de recepción, que tarda aproximadamente seis segundos en activarse para un mensaje de 12 bytes (el tiempo depende directamente del tamaño del mensaje), debido al tipo de señal.

Cuando se activa este evento se comprueba si la señal ha sido valida. En caso afirmativo se pasa a calcular las interferencias de ésta, es decir, se comprueba si ha habido interferencias entre el conjunto de señales activas, con el fin de saber si el mensaje se ha podido recuperar o no. Esto se hace mediante un modelo probabilístico que depende de la intensidad con la que se ha recibido la señal, siendo más probable que se recupere el mensaje cuanto mayor sea la potencia de la señal recibida. Si el mensaje se ha podido recibir correctamente se avisa a la capa MAC, y en caso contrario simplemente se añade el error al sistema de *log* y se descarta la señal.

Todo este proceso también tiene en cuenta cuando el dispositivo debe terminar el modo de recepción y ponerse a descansar para evitar que el hecho de recibir señales obligue al dispositivo a mantenerse activo.

Finalmente, en el caso de emisión del mensaje el proceso es más simple, únicamente se genera la señal a partir del mensaje que le envía la capa MAC, para esto se elige un canal por el que transmitirla, añadiéndole la densidad espectral correspondiente. Cuando la señal está completamente creada la capa física se encarga de enviarla por el canal de comunicación.

8.3 Enlace espectral

Todo este módulo se ha tenido que rehacer desde cero sobre el modelo original. El módulo del enlace espectral es utilizado para modelar el espectro. En él se define el rango de frecuencias que utiliza Sigfox, así como los canales que se pueden definir dentro de este rango. Este modelo espectral se crea mediante funciones de NS-3 ya diseñadas para este fin (ns3::SpectrumModel). En concreto NS-3 aporta una función para modelar un campo espectral a partir de definiciones de bandas, por lo que para crear esta parte se ha tenido que definir cómo son los canales de Sigfox.

Concretamente, Sigfox opera en el rango de frecuencias 868,0 - 868,60 MHz para *uplink*, es decir, para comunicaciones desde los nodos finales hacia las antenas, y en el rango de 869,40-869,65 MHz para *downlink*, para comunicaciones desde la antena hacia los nodos finales.

Se ha implementado el enlace espectral de forma que abarque todo este rango, de 868,0 - 869,65 MHz, utilizándose un total de 5175 canales de 100 Hz de ancho de banda cada uno. Sin embargo, de todos estos canales Sigfox utiliza únicamente 2000 para *uplink* y 830 para *downlink*. En la figura 42 se representa el espectro electromagnético.

En este módulo también se definen otras funciones relativas al espectro que no vienen incluidas por defecto en NS-3. Algunas de ellas son: la creación de un modelo de representación de densidad espectral sobre un canal (útil para generar una señal sabiendo el canal por el que se va a enviar), funciones que generan ruido sobre un canal, que se utiliza como ruido blanco a la hora de calcular las interferencias, o funciones para obtener la potencia media de un canal, que sirve para calcular la potencia de la señal recibida.

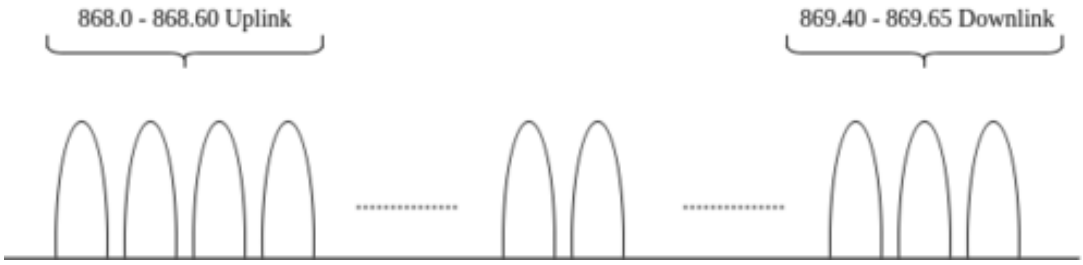

Figura 42 - Espectro electromagnético del modelo de simulación.

8.4 Calidad del enlace y tasas de errores

El módulo de calidad del enlace se encarga de comprobar la calidad del canal basándose en el último paquete recibido. Para ello evalúa la intensidad con la que se ha recibido y el número de errores que se han producido. Evaluar la calidad del enlace permite conocer el estado del canal de comunicación entre el emisor y el receptor, el cual se ve influido sobre todo por el modelo de propagación, y aunque por lo general esto no afecta en gran medida a la hora de realizar las simulaciones es de interés conocer esta información.

El módulo de tasa de errores trabaja a partir de la potencia de la señal recibida que le comunica la capa física. Realiza una valoración de si el mensaje se ha perdido o no, mediante un modelo probabilístico. Este módulo es clave dentro de la simulación, ya que es el que decide si un mensaje llega o si se ha perdido debido a la mala calidad de la señal que se ha recibido.

Ambos módulos están bastante relacionados ya que la calidad del enlace depende directamente de la tasa de errores, y ninguno de estos está previamente definido en NS-3.

Con relación al modelo original, el módulo de la calidad del enlace se ha mantenido respecto al mismo. Sin embargo, el módulo correspondiente a la evaluación de la tasa de errores ha sido rehecho, puesto que se ha detectado un error significativo en el modelo probabilístico que podía causar que las señales siguieran siendo válidas con potencias demasiado bajas.

8.5 Consumo de potencia

Este módulo se encarga de calcular el consumo de potencia de los dispositivos. Está directamente relacionado con el estado actual de la capa física. Cada vez que se realiza un cambio de estado de la capa física se notifica al módulo de consumo de potencia para realizar los cálculos de consumo.

Dichos cálculos se realizan siguiendo las características eléctricas del nodo a simular, es decir, utilizando la información de tensión y la corriente. En el caso de este trabajo, se ha emulado el comportamiento del microcontrolador *lopy4* [35], el cual se utiliza en la evaluación del modelo y presenta las siguientes características eléctricas:

- **Tensión de alimentación:** 3.3 – 5.5 V.
- **Consumo en la recepción:** 15mA.
- **Consumo en transmisión:** 42mA.
- **Consumo en *sleep*:** 0.5uA.

El modelo permite adaptarse a los costes energéticos del nodo que se vaya a utilizar. Únicamente hay que modificar las variables de consumo dentro del código del módulo, que es precisamente lo que se ha hecho sobre el modelo inicial para ajustarlo a las pruebas experimentales.

8.6 Capa MAC

El módulo de la capa MAC (Medium Access Control) se encarga de manejar y gestionar las comunicaciones, intercambiando información con la capa física para transmitir y recibir información.

En transmisión, la principal función de la capa MAC es definir una cola de mensajes, en la que se almacenan los paquetes a transmitir. Además de gestionar esta cola, se encarga de montar los paquetes añadiéndoles las colas y cabeceras correspondientes. Una vez la capa MAC está lista para enviar un paquete se lo comunica a la capa física, que se encarga del resto.

La capa MAC también recibe información procedente de la capa física cuando ésta decodifica un paquete con éxito. En este caso se encarga de eliminar las cabeceras y las colas y comprobar la información que almacena el paquete, como por ejemplo la dirección MAC destino.

Esta capa puede actuar tanto en modo promiscuo, es decir, aceptar todos los paquetes recibidos, o en modo dirgido, obligando así que se descarten los paquetes cuyo destino no sea el dispositivo receptor.. En modo normal, el procedimiento es el siguiente:

1. Se comprueba si la dirección de destino del paquete coincide con la del dispositivo.
2. Se realiza una comprobación de errores de la cabecera y datos del paquete.

Para controlar el acceso al canal se utiliza el protocolo RFTDMA que se ha explicado anteriormente en el apartado 5 sobre Sigfox.

En cuanto al modelo original, ocurría lo mismo que con la capa física, es decir, estaba pensado únicamente para dos dispositivos, y muchos de los estados posibles no se habían tenido en cuenta.

Tampoco se había realizado ninguna comprobación de la dirección *hardware* asociada a los mensajes, funcionando todos los dispositivos únicamente en modo promiscuo por defecto. Además, no se diferenciaba a los dispositivos de las estaciones base de Sigfox, cuando ambas son totalmente diferentes.

De igual manera que en la capa física, se han implementado dos tipos de capas MAC, una para los dispositivos o nodos finales y otra para las antenas:

- **Capa MAC de los dispositivos o nodos finales:** Se ha definido una máquina de estados que permite conocer el estado actual de la capa y definir qué acción está realizando en cada momento, siguiendo una serie de transiciones. Los posibles estados son los siguientes:

  - **MAC_IDLE:** indica que la capa MAC está descansando, sin realizar ninguna acción en concreto.
  - **MAC_SENDING:** indica que la capa MAC está ocupada enviando un paquete por la capa física.
  - **MAC_FAILURE:** indica que ha surgido algún problema en la capa, debido al montaje de tramas o a fallos de recepción.

  El grafo que representa la transición entre los estados se puede observar en la figura 43.

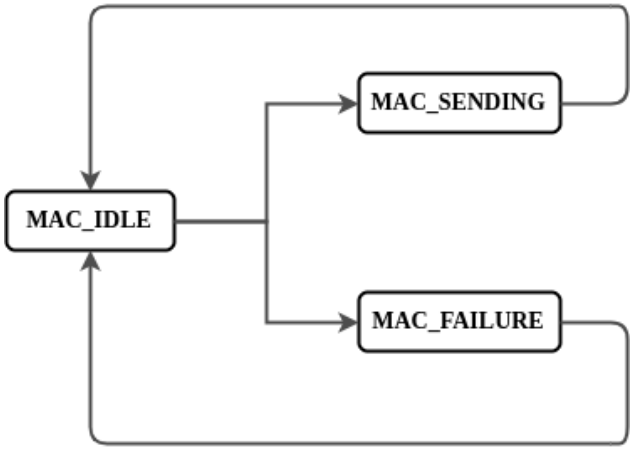

Figura 43 - Máquina de estados de la capa MAC

- **Capa MAC de las estaciones:** Al igual que con la capa física de las antenas, la capa MAC tampoco dispone de una máquina de estados, por lo que puede enviar simultáneamente varios paquetes, y no es necesario implementar una cola de mensajes.

El funcionamiento interno de la capa MAC es el siguiente:

En recepción, esta capa es notificada por la capa física, y su primera función es retirar tanto las cabeceras como las colas del mensaje y procesarlas. Se comprueba si el mensaje iba dirigido a este dispositivo; en caso afirmativo (o si está en modo promiscuo) se invoca al *callback* correspondiente definido por el usuario para el caso de recepción si lo hubiera. Podría ser, por ejemplo, mostrar el mensaje por pantalla o actualizar el contador de mensajes recibidos. Si el mensaje no tenía como destino este dispositivo se notifica mediante un mensaje de *log* de advertencia.

Para enviar un mensaje el proceso es más complejo. En primer lugar, se generan tanto las tramas de cabecera como las colas, y se crea un paquete con todo el contenido a enviar. Una vez hecho

esto el paquete se encola. Cada vez que el dispositivo alcanza el estado MAC_IDLE, se notifica a la capa física para que envíe el mensaje más antiguo de la cola, y mientras tanto el dispositivo se cambia al estado MAC_SENDING. Cuando la capa física termina de enviar el mensaje se notifica de nuevo a la capa de MAC, que ahora realiza una comprobación de su cola de envíos. Si no está vacía entonces se repite el proceso y el dispositivo continua en el estado de MAC_SENDING; en caso contrario el dispositivo se colocaría de nuevo en el estado de MAC_IDLE. Si ocurre una excepción en el proceso de enviar el paquete la capa cambiaría al estado MAC_FAILURE.

Toda la parte de envíos se ha tenido que rehacer sobre el modelo original, de forma que se han corregido numerosos errores de estados incoherentes que antes se podían producir, y, además, se ha mejorado la interacción con la capa física.

8.7 Ejemplo de uso del modelo

Con el fin de mostrar cómo se utiliza el modelo desarrollado se explicarán los pasos para definir un escenario concreto y obtener las mediciones de tasa de errores. Se va a definir un escenario urbano en el que existen dos nodos finales y una estación base, estos nodos enviarán un total de 5 paquetes cada uno. El primero de los nodos finales va a estar en continuo desplazamiento alejándose cada vez más de la estación base, mientras que el otro se va a mantener fijo. En la figura 44 se representa aproximadamente este escenario.

Figura 44 – Escenario de la simulación.

En primer lugar, se definen las variables finales de tamaño de los paquetes, número, potencia de transmisión e incremento en el desplazamiento.

```
int increment = 100;
int maxPackets = 100;
int packetSize = 12;
double txPower = 14; //dbm 23mw = 13.61 dbm, 25 mW = 14 dBm
```

Se declaran los nodos a utilizar, así como el canal de comunicación y el modelo de propagación asociado a éste.

```
Ptr<Node> n1 = CreateObject <Node> ();
Ptr<Node> n2 = CreateObject <Node> ();
Ptr<Node> baseStation = CreateObject <Node> ();

Ptr<MultiModelSpectrumChannel> channel =
CreateObject<MultiModelSpectrumChannel> ();

Ptr<OkumuraHataPropagationLossModel> model =
CreateObject<OkumuraHataPropagationLossModel> ();

model->SetAttribute("Environment",EnumValue (UrbanEnvironment));
channel->AddPropagationLossModel (model);
```

Una vez los nodos han sido generados hay que crear los dispositivos Sigfox (tanto los nodos finales como las antenas), e integrarlos en los nodos. Se les asocia una dirección y se conectan al canal de comunicación.

```
Ptr<iotNetDevice> dev1 = CreateObject<iotNetDevice> ();
   dev0->SetChannel(channel);
   dev0->SetAddress(Mac16Address ("00:00"));
   n1->AddDevice(dev1);

Ptr<iotNetDevice> dev2 = CreateObject<iotNetDevice> ();
   dev1->SetChannel(channel);
   dev1->SetAddress(Mac16Address ("00:01"));
   n2->AddDevice(dev2);

Ptr<iotNetStationDevice> dev3 =  CreateObject<iotNetStationDevice>
();
   dev2->SetChannel(channel);
   dev2->SetAddress(Mac16Address ("00:02"));
   baseStation->AddDevice(dev3);
```

Ahora hay que asignarles un modelo de movilidad para definir sus respectivas posiciones. También se les asigna la potencia de transmisión a los dispositivos.

```
Ptr<ConstantPositionMobilityModel>
mob1=CreateObject<ConstantPositionMobilityModel> ();
  mob1 ->SetPosition(Vector (-100,100,1));
  dev1->GetPhy ()->SetMobility (mob1);
  dev1->GetPhy ()->SetTXPower (txPower);

Ptr<ConstantPositionMobilityModel>
mob2=CreateObject<ConstantPositionMobilityModel> ();
  mob2 ->SetPosition(Vector (100,100,1));
  dev2->GetPhy ()->SetMobility (mob2);
  dev2->GetPhy ()->SetTXPower (txPower);

Ptr<ConstantPositionMobilityModel>
mob3=CreateObject<ConstantPositionMobilityModel> ();
  mob3 ->SetPosition(Vector (0,0,1));
  dev3->GetPhy ()->SetMobility (mob3);
```

El propósito de esta simulación es llevar a cabo la cuenta de los paquetes que llegan a la estación base. Para ello se debe declarar una función que incremente en uno el valor del contador correspondiente cada vez que la estación base reciba un paquete.

```
static void
iotErrorDistanceCallback (McpsDataIndicationParams params,
Ptr<Packet> p)
{
  station_received++;
}
```

Y asociarla a la estación base.

```
McpsDataIndicationCallback cb0;
  cb0 = MakeCallback (&iotErrorDistanceCallback);
  dev3->GetMac ()->SetMcpsDataIndicationCallback (cb0);
```

Finalmente, ya solo queda definir los parámetros del paquete a enviar y comenzar con el bucle de envíos mientras se va modificando la posición del nodo 1.

```
McpsDataRequestParams params;
  params.m_dstAddress = Mac16Address ("00:03");
  params.m_msduHandle = 0;


for (int j =0; j <= 15000;)
    {
      for (int i = 0; i < 5; i++)
        {
          p = Create<Packet> (packetSize);

         Simulator::Schedule (Seconds (i),
                              &iotMac::McpsDataRequest,
                              dev1->GetMac (), params, p);

          p = Create<Packet> (packetSize);

         Simulator::Schedule (Seconds (i),
                              &iotMac::McpsDataRequest,
                              dev2->GetMac (), params, p);


        }

     Simulator::Run ();

     std::cout<< station_received << " " << j << std::endl;
     station_received=0;
     j += increment;

     mob1->SetPosition (Vector (j,100,1));
    }
```

Con esto el ejemplo está completo y se puede ejecutar con el simulador NS-3. La salida que proporciona este código es la siguiente:

```
10 0
10 100
10 200
10 300
10 400
…
8 4800
6 4900
5 5000
```

Donde la primera columna corresponde al número de mensajes recibidos por la estación base y la segunda a la distancia aproximada a la que se encuentra el nodo 1 de la estación. Con esta información y algún programa externo que nos permita graficar estos datos se puede obtener la representación de la figura 45.

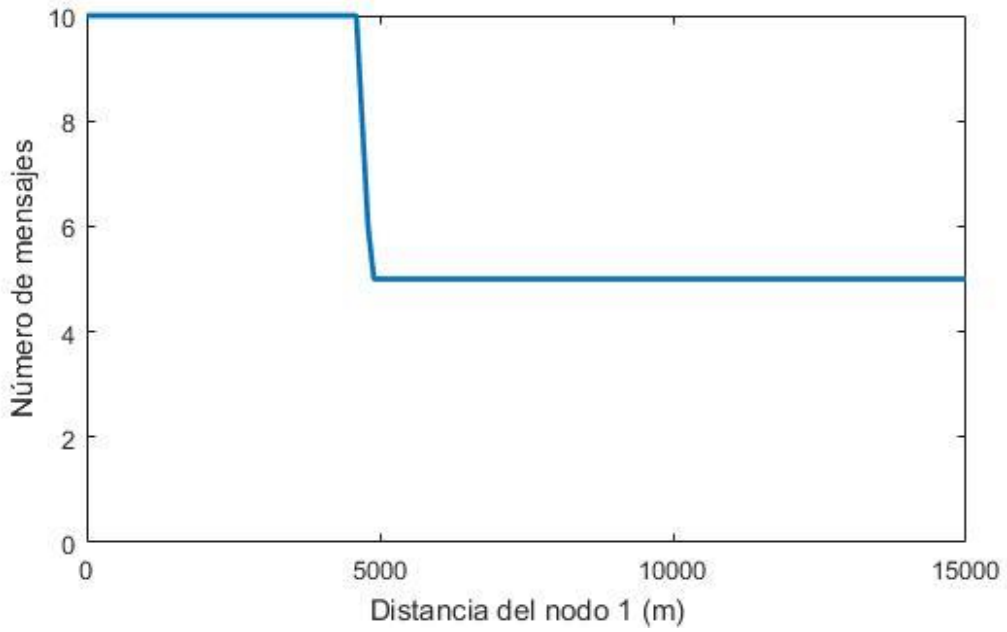

Figura 45 – Resultados de la simulación.

Como se observa en la figura 45, cuando el nodo uno se encuentra próximo a la estación base, ésta es capaz de recibir los 10 paquetes que envían los dos nodos en conjunto. Sin embargo, al alejar cada vez más el nodo uno lo que ocurre es que la estación deja de recibir sus mensajes a partir de aproximadamente 5 km (Es conveniente recordar que se está simulando un entorno urbano). A partir de esta distancia, la estación base solo recibe los 5 mensajes del nodo 2 que se mantiene estático.

9. Validación experimental

Con el fin de validar el modelo de simulación creado para la tecnología Sigfox en NS-3 se han realizado diversas pruebas reales. A continuación, se detallarán las pruebas realizadas, así como el *hardware* y el *software* empleado. Finalmente se mostrará una comparativa de los resultados de la simulación con los obtenidos experimentalmente.

9.1 Hardware empleado

Como dispositivo emisor se ha utilizado el microcontrolador *lopy4* producido por Pycom [35], que puede verse en la figura 46. Este microcontrolador se controla mediante MicroPython, un subconjunto de Python, y posee compatibilidad con las tecnologías Sigfox, LoRa, WiFi, y Bluetooth. Es una placa de desarrollo pensada para el internet de las cosas que tiene un bajo consumo energético pese a su considerable potencia de procesamiento.

El *lopy4* posee dos procesadores y un sistema de radio. El primer procesador está dedicado exclusivamente a los programas de usuario, mientras que el segundo es un coprocesador para monitorizar los pines de entrada y salida, junto con las interfaces de comunicación. Además, dispone de un acelerador de instrucciones de coma flotante para realizar grandes operaciones de forma rápida, interfaces UART, SPI, e I2C, cuatro temporizadores, y 24 pines de uso genérico.

En cuanto a memoria, dispone de una memoria RAM de 4MB, y de una memoria flash externa de 8MB, más que suficiente para aplicaciones típicas de internet de las cosas.

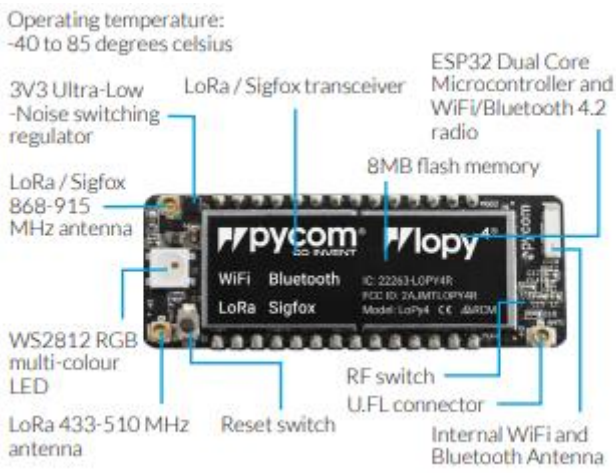

Figura 46 - Esquema del lopy4.

Se alimenta mediante una entrada de 3.3 a 5.5 V, y sus consumos eléctricos dependen de qué interfaz de comunicaciones se emplee:

- **WiFi:** 12mA en modo activo, 5uA en reposo
- **LoRa:** 15mA en modo activo, 1uA en reposo
- **Sigfox(Europa):** 12 mA en modo de recepción, 42mA en modo de transmisión y 0.5uA en reposo.

- **Sigfox(Australia,Nueva Zelanda y América del Sur):** 12mA en modo de recepción, 120mA en modo de transmisión y 0.5uA en reposo.

Para poder dotar al microcontrolador de interfaz USB y conectarlo así a un ordenador personal, se ha de utilizar una de las placas adicionales ofrecidas por Pycom:

- **"Expansion Board" (Figura 47)**: esta placa es la más básica de las disponibles. Sin embargo, además de ofrecer una interfaz USB para alimentar y comunicarse con el *lopy* también dispone de otras funcionalidades, como un puerto para baterías LiPo (baterías de polímero de litio), espacio para colocar una tarjeta microSD, leds y botones de usuario, circuitos de protección ante posibles voltajes inversos, etc.

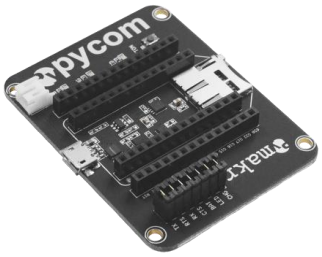

Figura 47 - Expansion Board de Pycom.

- **"PySense" (Figura 48):** esta placa, como su nombre indica, se caracteriza por incorporar varios sensores útiles para obtener información del entorno y no depender de *hardware* adicional. Entre ellos dispone de un sensor de luz ambiental, un barómetro usado comúnmente para medir la altura, un sensor de humedad, un sensor de temperatura, y un acelerómetro de tres ejes y 12 bits. También posee de la interfaz USB, del puerto para baterías LiPo y del espacio para tarjetas microSD.

Figura 48 - PySense de Pycom.

- **"PyTrack" (Figura 49):** esta placa posee, al igual que las anteriores, la interfaz USB, el puerto para baterías LiPo y el espacio para tarjetas microSD- Lo que le diferencia de las anteriores es que posee un sistema de localización GPS integrado, muy útil para obtener las coordenadas del dispositivo en infraestructuras grandes basadas en el internet de las cosas.

Figura 49 - PyTrack de Pycom.

El *lopy4* viene con una licencia de Sigfox con validez de dos años. Lo único que hay que hacer es registrar el dispositivo en la web de Sigfox con su identificador. Este es uno de los motivos por los que se ha optado por este dispositivo.

Es importante mencionar que el *lopy4* requiere de una antena externa adicional para utilizar Sigfox o LoRa, ya que el sistema de radio que tiene incorporado no es suficiente para este caso. Hay que utilizar una antena diseñada para la banda de frecuencias de 868Mhz.

9.2 Software

Para programar las pruebas se ha utilizado el entorno de desarrollo Atom [36]. Éste es un editor de texto multiplataforma, con funcionalidades típicas de autocompletado, visualización intuitiva de ficheros y corrección de sintaxis para varios lenguajes de programación.

Figura 50 - Editor Atom.

Una de sus principales ventajas es que es extensible mediante paquetes, por lo que cada desarrollador puede crear un paquete para agregar funcionalidades extra al entorno de desarrollo. En este caso se usa el paquete "Pymakr" que permite conectarse directamente al *lopy4* mediante un puerto serie virtual, que además permite subir ficheros de código directamente al dispositivo y ejecutarlos directamente.

En cuanto al código, para utilizar Sigfox se emplea una librería para el *lopy4* desarrollada en MicroPython, que permite de forma fácil utilizar la tecnología Sigfox para comunicarse remotamente con las estaciones base u otros dispositivos. Un ejemplo de uso sería el siguiente:

Primero hay que crear un objeto Sigfox. A este objeto se le indica el protocolo a usar y la zona en la que se encuentra el dispositivo, requisito necesario para ajustar las frecuencias de los canales a la legislación local.

```
sigfox.init(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1, * , frequency=None)
```

- ***mode*:** identifica el protocolo y la modulación a usar; existe el modo Sigfox y el modo FSK, el cual sirve para comunicar dos dispositivos utilizando esta modulación.
- **rcz:** indica la zona del dispositivo
  - RCZ1: Europa, Omán y África del sur.
  - RCZ2: Estados Unidos, Méjico y Brasil.
  - RCZ3: Japón.
  - RCZ4: Australia, Nueva Zelanda, Singapur, Taiwán, Hong Kong, Colombia y Argentina.
- ***frequency*:** utilizado para el modo FSK.

El siguiente paso es crear el *socket* de Sigfox:

```
s = socket.socket(socket.SOL_SIGFOX, socket.SOCK_RAW)
```

Y configurarlo:

socket.setsockopt(level,optname,value)

- ***level*:** Indica el tipo de tecnología, se usará siempre *socket.SOL_SIGFOX*.
- ***optname*:** Indica el atributo que se quiere cambiar:
  - SO_RX: define si es *uplink* y *downlink* o únicamente *uplink*.
  - SO_OOB: define si se pueden enviar mensajes fuera de banda, es decir, en rangos de frecuencia fuera de los convencionales para la tecnología.
  - SO_BIT: define si seleccionar el bit de valor cuando únicamente se envía un bit.
- ***value*:** Indica el valor del atributo anterior, puede ser *True* o *False.*

Finalmente se envían los bytes que se deseen:

```
s.send(bytes([1,...]))
```

Adicionalmente si hemos configurado el *socket* como *downlink* se puede utilizar el método *recv*() para leer la respuesta de la estación base.

Este mensaje llegará a las estaciones base que haya al alcance del dispositivo. Otros métodos interesantes sobre el objeto Sigfox de python son también:

- **mac():** devuelve un *byte* con la dirección MAC del dispositivo.
- **id():** devuelve el identificador de Sigfox del dispositivo (este es el identificador que se utiliza para registrarlo).
- **rssi():** devuelve un entero indicando el nivel de señal del último paquete recibido.
- **pac():** devuelve un byte con el Sigfox PAC (también utilizado para registrar el dispositivo).

Un *script* completo para enviar 12 bytes mediante Sigfox utilizando esta librería podría ser el siguiente:

```
from network import Sigfox
import socket

sigfox = Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ1)


s = socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)


s.setblocking(True)


s.setsockopt(socket.SOL_SIGFOX, socket.SO_RX, False)

# send some bytes
s.send(bytes([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]))
```

9.2.1 Servidor local

Con el fin de almacenar los resultados obtenidos en las pruebas de validación se ha implementado un servidor local que actúa como objetivo de las *callback* o llamadas de retorno del *back-end* de Sigfox. Este servidor se ha desarrollado utilizando *servlets* de Java. Éstos permiten crear aplicaciones web dinámicas, lo que quiere decir que permiten realizar consultas, insertar y eliminar datos que modifiquen la página web.

Los *servlets* son pequeños programas escritos en Java que admiten peticiones a través del protocolo HTTP. Reciben peticiones desde un navegador web o desde otro servidor, las procesan y devuelven una respuesta. Necesitan almacenarse dentro de lo que se conoce como un contenedor de aplicaciones web.

Los contenedores son en esencia servidores web que reciben las peticiones HTTP y se encargan de redireccionarlas a las distintas aplicaciones que tiene en su interior. En este caso se ha utilizado Apache Tomcat [37] como contenedor de aplicaciones *servlets.*

Para almacenar los mensajes se ha empleado JDBC (Java *Database Connectivity)*, la cual es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de

programación Java, independientemente del sistema operativo o de la base de datos a la cual se accede. Y como base de datos se ha utilizado SQLite [38], ya que es una base de datos muy ligera y que no requiere de un proceso cliente-servidor como es común en las bases de datos, sino que se enlaza directamente con el programa principal. El conjunto de la base de datos es almacenado en un único fichero en la máquina local.

Se ha definido un objeto mensaje, que será el que se almacene en la base de datos a modo de tabla. Tiene los siguientes atributos:

- **Device:** número de serie del dispositivo emisor.
- **Time:** tiempo actual de recepción del mensaje.
- **SNR (*Signal to noise ratio)*:** nivel de ruido en la recepción.
- **RSSI (*Received Signal Strength*):** potencia de la señal recibida.
- **Seq:** número de secuencia del mensaje.
- **LQI (*link Quality Indicator)*:** indicador de la calidad del canal.
- **Station:** identificador de la estación base.
- **Data:** texto del mensaje recibido (representado en hexadecimal).

El servidor que se ha implementado se divide en dos partes:

- **Respuesta a peticiones POST:** dentro del *back-end* de Sigfox se ha configurado las distintas llamadas de retorno como peticiones POST, que devuelven los parámetros de interés de los mensajes, enumerados anteriormente. Este servidor se encarga de obtener los parámetros de la petición y de crear el objeto mensaje. Estos mensajes los almacena dentro de la base de datos local.
- **Respuesta a peticiones GET:** las peticiones GET las solicita el navegador web cuando accede al servidor, así que en este caso lo que realiza es acceder a la base de datos y obtener una lista de todos los mensajes almacenados. Una vez obtenida la lista se devuelve como respuesta el texto conjunto de los mensajes.

La respuesta del servidor a peticiones GET por parte del navegador tiene el siguiente aspecto:

```
Device: &4d2e78 Time: 1519912861.0 SNR: 50.0 RSSI: -
23.520000457763672 Seq: 26 LQI: GOOD Station: 0x0708 Data:
736563757269746173

Device: &4d2e78 Time: 1519912706.0 SNR: 50.0 RSSI: -
22.81999969482422 Seq: 25 LQI: GOOD Station: 0x0708 Data: 6164696f73
…
```

De esta forma se han almacenado los mensajes de Sigfox dentro del servidor local, pudiéndose visualizarse su contenido en cualquier momento a través de un navegador web.

9.3 *Back-end* de Sigfox

Una vez ya está configurado el *hardware* y el *software* se puede empezar a enviar mensajes que serán recogidos por las estaciones base de Sigfox al alcance. Para visualizarlos se puede o utilizar la API HTTP explicada en la sección 5.9 de Sigfox, acceder al portal web o crear las *callbacks* hacia el servidor local.

Dentro del portal web se puede monitorizar a todos los dispositivos asociados a la cuenta, y visualizar los mensajes junto con su *metadata* asociada. También es posible configurar *callbacks* o llamadas de retorno HTTP a una dirección. Éstos son activados ante determinados eventos, como por ejemplo la llegada de un mensaje. En las llamadas de retorno se configura la información que se desea devolver, pudiéndose incluir desde información física de la recepción (potencia de la señal, ruido, etc.) hasta el propio mensaje completo.

En las pruebas realizadas se ha definido un *callback* al servidor local. Este *callback* se activa cuando se detecta un nuevo mensaje, y devuelve el conjunto de parámetros de éste mediante una petición POST: *Device*, *time*, SNR, RSSI, Seq, *Station* y *data*. De esta forma la información es recogida y almacenada por el servidor implementado. En la figura 51 se observa el aspecto que tiene un *callback* dentro del *back-end* de Sigfox.

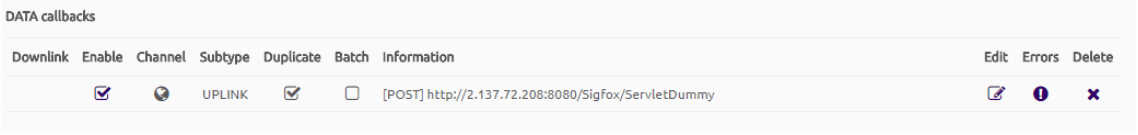

Figura 51 - Callback definido en el back-end de Sigfox.

Un parámetro de las *callbacks*, tal y como se muestra en la figura, es la opción *duplicate*. Esta opción permite que cada estación base que reciba el mensaje genere un *callback*, en vez de generar únicamente uno por mensaje independientemente del número de estaciones que lo reciban. Si se activa esta opción es posible conocer la respuesta de cada estación base a un mismo mensaje, ya que cada una tiene un identificador distinto definido en el parámetro *Station* de los mensajes, anteriormente nombrado.

También es posible visualizar directamente los mensajes desde el *back-end* sin emplear llamadas de retorno. Sin embargo, de esta forma solo se puede visualizar parte de la información del mensaje y no la totalidad de sus parámetros. Además, tener los mensajes dentro de una base de datos local permite realizar búsquedas de forma sencilla y rápida, que son imposibles de hacer directamente desde el *back-end*.

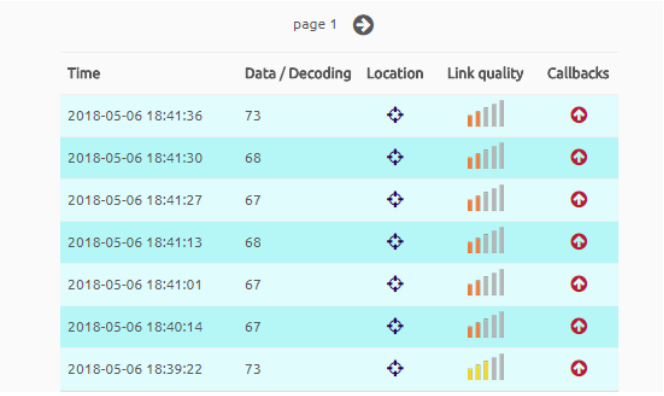

Figura 52 - Visualización de los mensajes en el back-end de Sigfox.

Hay que mencionar que, aunque Sigfox nos proporciona la ubicación aproximada del dispositivo emisor, esta información es inútil en este caso, ya que proporciona resoluciones de un grado en latitud y longitud.

9.4 Pruebas experimentales

En primer lugar, se ha utilizado el *lopy4* junto con una antena externa para enviar los mensajes, y se ha programado un pequeño *script* como el mostrado en el apartado anterior que se encarga de enviar bytes mediante el protocolo Sigfox. En el *back-end* se ha configurado el *callback* mencionado anteriormente con duplicados activos, que genera una llamada POST cuando aparece un mensaje nuevo. Esta información es recogida por el servidor local, y se almacena en la base de datos. En la figura 53 se muestra un esquema.

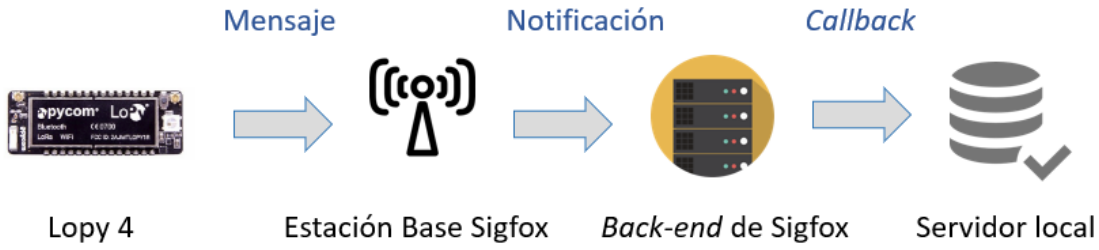

Figura 53 – Diagrama de las pruebas experimentales.

El objetivo es validar el modelo de simulación realizando una comparación de la potencia de la señal recibida en función de la distancia entre el dispositivo emisor y la estación base. Cabe mencionar que para realizar el estudio se debe conocer dónde se encuentra al menos una de las estaciones, y así poder comparar la potencia de las señales respecto de la distancia.

En este caso únicamente se ha podido obtener la posición de una estación base situada en el centro de la ciudad de Valencia, ya que esta información es confidencial y la empresa CELLNEX (encargada en España de las antenas Sigfox) solo ha proporcionado la posición de dicha antena.

El proceso es simple:

1. Se coloca el dispositivo *lopy* en una ubicación en concreto.
2. Éste envía una serie de mensajes.
3. El *back-end* de Sigfox ejecuta las *callbacks* definidas.
4. Se accede al servidor local y a la potencia de la señal recibida por la antena (RSSI) que conocemos su posición (para esto es importante activar los duplicados en los *callback*, para que respondan todas las antenas)
5. Se hace una media de los valores de potencia de cada mensaje, y finalmente se representa la relación RSSI – distancia obtenida.

La idea de la validación es tomar una gran cantidad de muestras por la ciudad de Valencia, tanto en entornos rurales como urbanos. Con estas muestras se puede relacionar la potencia de la señal recibida por una estación base con la distancia recorrida por ésta dentro en un entorno determinado. Comparar esta información real con la generada por el modelo permite validar el modelo de simulación en cuanto a emulación de las señales electromagnéticas.

Las estaciones de Sigfox tienen una sensibilidad aproximada de -140db, lo que quiere decir que son capaces de captar señales de hasta -140db de potencia. Normalmente la potencia máxima de las señales recibidas está alrededor de -70db en el caso más favorable, cuando el dispositivo

emisor se encuentra prácticamente al lado de la estación base. A continuación, se presenta la comparación entre los resultados reales y los simulados, para cada entorno.

9.4.1 Entorno urbano

Para modelar el entorno urbano se han realizado numerosas pruebas alrededor de la antena o estación base disponible. Al estar rodeada de edificios es bastante sencillo colocar el dispositivo en ubicaciones que se puedan clasificar como urbanas. Por cada ubicación se ha tomado una media de aproximadamente 10 medidas del RSSI. Mediante las pruebas se observa que el alcance de los mensajes es de 5 km aproximadamente. A partir de entonces la estación base no es capaz de recibirlos. Este radio se representa en la figura 54.

Figura 54 – Radio de cobertura de Sigfox para entornos urbanos.

Las ubicaciones se han escogido de forma aleatoria dentro del radio marcado en la figura. La estación base de la cual se nos ha proporcionado información se encuentra dentro de este radio, sin ser el centro la ubicación exacta. Los resultados son los siguientes:

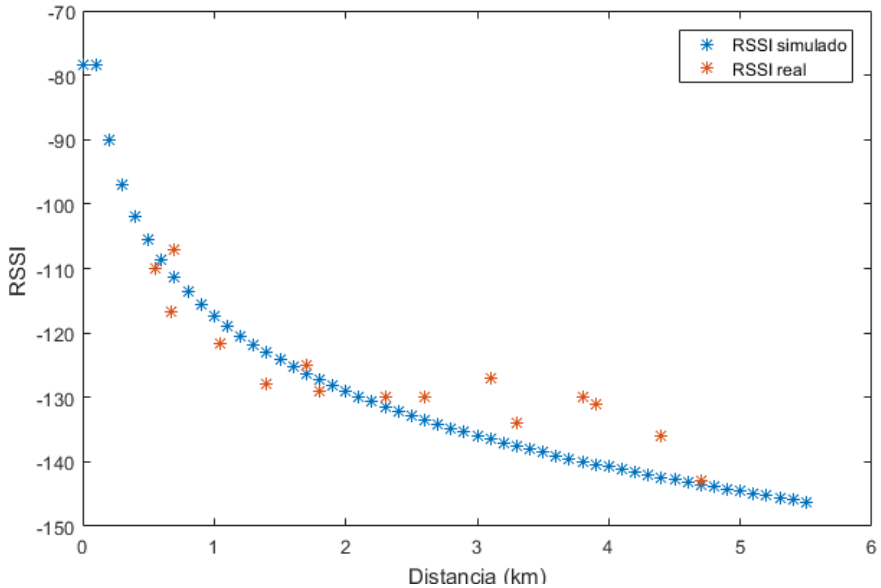

Figura 55 - Comparación de resultados en entorno urbano.

En la figura 55 se observa cual ha sido el RSSI obtenido por el modelo de simulación para un entorno urbano en comparación con las mediciones reales. Se puede ver que los resultados son muy similares. Sin embargo, el entorno urbano es el más inexacto de todos, ya que depende en gran medida de los edificios circundantes. El hecho de realizar las mediciones debajo de un edificio o ligeramente a campo abierto afecta en gran medida a la calidad de la señal, por lo que es difícil de modelar exactamente la potencia recibida por la estación.

9.4.2 Entorno suburbano

En cuanto al entorno suburbano se han realizado pruebas intentado evitar grandes edificios de por medio, y se ha podido observar que en este caso la señal es capaz de llegar hasta aproximadamente 6.5 km. Este radio se representa en la figura 56.

Figura 56 – Radio de cobertura de Sigfox para entornos suburbanos.

Nuevamente las pruebas se han realizado aleatoriamente dentro de este radio, en ubicaciones lo más suburbanas posibles, es decir, con pocos edificios de por medio. El centro no representa la ubicación exacta la antena o estación base Sigfox, sino una aproximación por normas de confidencialidad con CELLNEX.

En la gráfica de la figura 57 se muestra cómo en este caso los resultados son aún más exactos que en el entorno urbano, y el modelo simulado se adapta bien a la realidad.

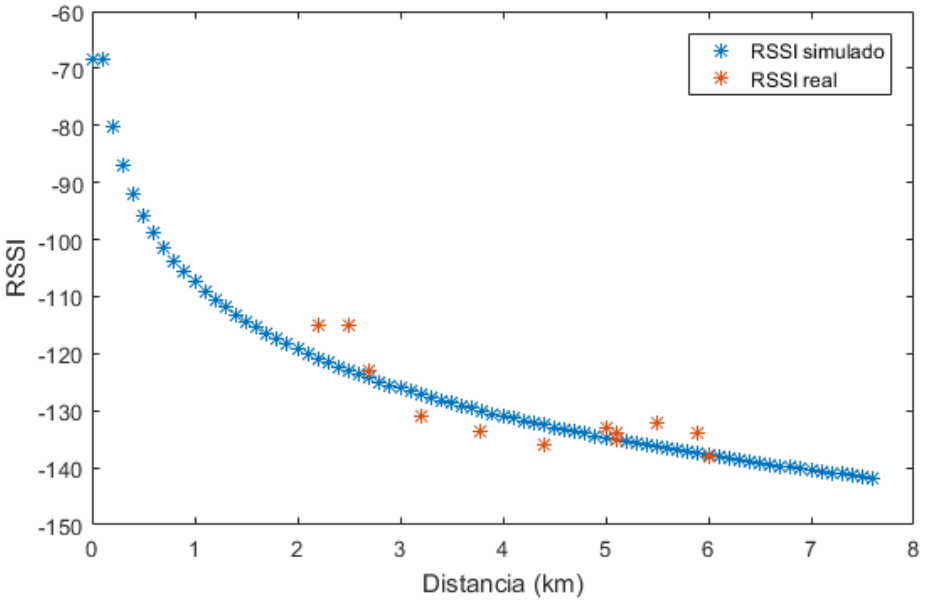

Figura 57 - Comparación de resultados en entorno suburbano.

9.4.1 Entorno rural

Finalmente, para modelar el entorno rural es necesario alejarse la antena unos 5 km. Esto es debido a que está situada en una zona céntrica de la ciudad y es imposible obtener medidas totalmente rurales a menos de esta distancia, ya que se clasificarían en todo caso como suburbanas. Para emular un entorno rural se han realizado pruebas por la costa de Valencia, para evitar el mayor número de obstáculos posible entre el emisor y la antena. En este caso se comprueba que el alcance llega hasta 25 km, por lo que se puede deducir que la presencia de obstaculos daña en gran medida a las señales de esta tecnología. El radio se representa en la figura 58.

Figura 58 – Radio de cobertura de Sigfox para entornos rurales.

De igual manera que en los anteriores entornos las pruebas se han realizado aleatoriamente dentro de este radio, a partir de los 5 km. La mayoría de las muestras se han obtenido en la costa, ya que ofrece un entorno rural bastante apropiado. Los resultados de las pruebas son los siguientes:

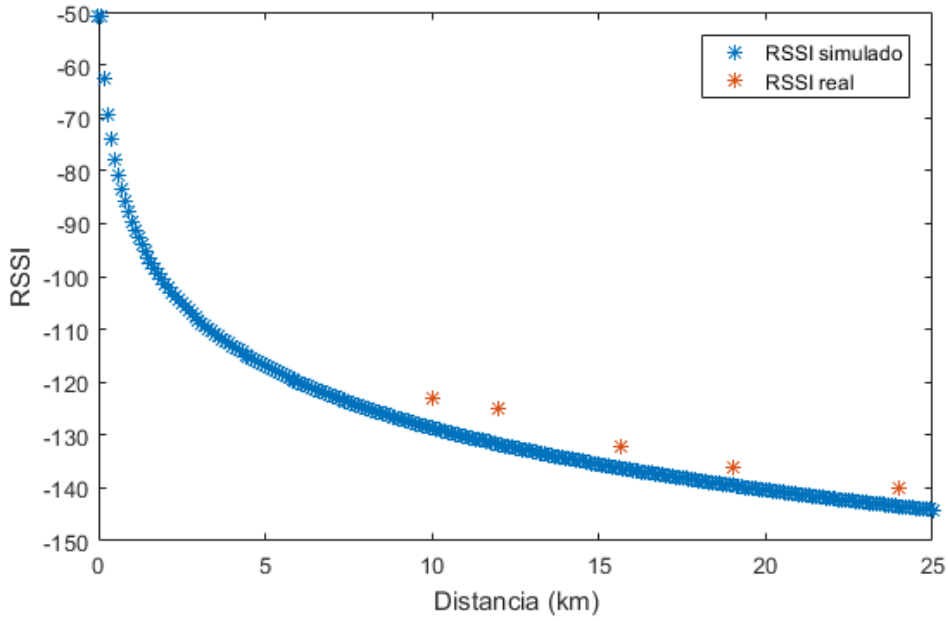

Figura 59 - Comparación de resultados en entorno rural.

En la gráfica de la figura 59 se observa que las mediciones obtenidas se ajustan bien de acuerdo con el modelo de simulación. En este caso se han podido realizar menor cantidad de pruebas debido a que al cubrir un radio tan amplio se ha priorizado tomar las pruebas siguiendo un orden de pueblos, desde valencia hasta las afueras. Uno de los posibles trabajos futuros es ampliar estas pruebas añadiendo mayor número de muestras.

10. Conclusiones y trabajo futuro

En este trabajo académico de final de grado se ha presentado en detalle en que consiste el mundo del Internet de las Cosas, y se ha hecho una comparativa entre las distintas tecnologías de red actuales. El objetivo principal que se planteó, es decir, crear un modelo de simulación de la tecnología Sigfox y validarlo mediante pruebas reales ha concluido con un resultado positivo.

Pese a que se ha partido de un modelo ya empezado (sobre todo al nivel de la capa física) se ha tenido que realizar mucho trabajo para poder obtener el modelo completo. Se puede decir que dentro del resultado final aproximadamente la parte ya implementada inicialmente ocupa únicamente un 25%, considerando que se ha tenido que modificar ésta también en varios puntos.

También hay que comentar que este trabajo ha sido realizado únicamente por un alumno, y que sobre todo para la validación del modelo de simulación se han tenido que realizar numerosas pruebas por gran parte de la provincia de Valencia. Los recursos han sido limitados tanto a nivel de *hardware* (ya que únicamente se disponía de un dispositivo) como a nivel de tiempo disponible.

El trabajo futuro pasaría por realizar una evaluación más exhaustiva del modelo, tanto a nivel lógico como a nivel físico. La idea sería forzar el modelo de simulación con escenarios complejos para corroborar que sigue siendo eficaz y no se produce ningún fallo en la simulación, o en este caso corregir los posibles errores menores que puedan surgir debido a escenarios inicialmente no contemplados. Y en cuanto a nivel físico se podría reafirmar la validación de la potencia de la señal reflejada en la simulación con más pruebas reales. Para esto se debería conocer la posición de alguna antena más que permita realizar en conjunto un estudio más completo. Dado que la antena de la cual se nos ha proporcionado información está situada en el centro de la ciudad ha sido imposible validar la simulación en entornos rurales de 1 a 5 km, ya que está totalmente rodeada de edificios a esa distancia.

El trabajo realizado ha sido satisfactorio personalmente, ya que he podido poner en práctica y ampliar mis conocimientos sobre gran cantidad de tecnologías. He trabajado directamente con *hardware* real, diseñado un modelo de simulación en lenguaje C++, entrado en temas sobre computación en la nube, implementado un servidor completo junto con una base de datos, conocido simuladores de red, estudiado una tecnología completa como es Sigfox, etc. Por esto puedo decir que ha sido un trabajo de final de grado bastante completo en mi opinión.

Por último, agradecer a la empresa CELLNEX por ofrecerse a ayudar en este trabajo al proporcionarnos la información de una de sus antenas que actúan como estación base de Sigfox.

11. Relación del trabajo con los estudios cursados

Gran parte del trabajo ha sido posible gracias a los conocimientos adquiridos en las distintas asignaturas impartidas en el grado, empezando por las primeras asignaturas de programación y de redes las cuales son un punto clave del grado de informática, hasta asignaturas más avanzadas sobre sistemas distribuidos o sistemas empotrados.

El objetivo principal como se ha destacado en varias ocasiones ha sido el de implementar un modelo de simulación. Este modelo se ha programado en C++, que pese a que no es un lenguaje en el que se entre en profundidad en el grado se basa en varios conceptos genéricos sobre lenguajes de programación, y hace uso de estructuras de datos complejas que sí se enseñan en la carrera. Además, al trabajar con *hardware* para realizar las pruebas de evaluación han sido de utilidad las asignaturas de la rama de ingeniería de computadores que tratan sobre la arquitectura de procesadores y sobre microcontroladores.

Obviamente al tratarse de un trabajo sobre una tecnología de red ha sido muy útil conocer cómo funcionan las redes más populares hoy en día como WiFi o Ethernet, que se explican durante el grado.

Finalmente concluir que el conocimiento que se enseña en el grado de informática es un conocimiento bastante polivalente. Esto quiere decir que, aunque las tecnologías que se enseñan no sean exactamente las que un alumno vaya a utilizar en su carrera profesional sí que asientan unas bases muy similares que permiten al alumno adaptarse a nuevas tecnologías de forma rápida.

12. Bibliografía

Últimos accesos a las direcciones web en junio de 2018.

[1] Tecnología Sigfox. (2017). “Sigfox technical overview”.
Disponible en:
https://www.disk91.com/wpcontent/uploads/2017/05/4967675830228422064.pdf

[2] Pablo Pardal Garcés. (2017). Redes de Área Extensa para aplicaciones de IoT: modelado de comunicaciones Sigfox. Máster en Ingeniería de Computadores y Redes DISCA-UPV.

[3] Simulador NS-3. https://www.nsnam.org/

[4] Tecnología LoRa. https://lora-alliance.org/

[5] Mehmet Ali Ertük. (2017). “LoRaWAN indoor performance analysis”.

[6] Davide Magrin, Marco Centenaro, y Lorenzo Vangelista. (2017). “Performance evaluation of LoRa networks in a smart city scenario”.

[7] Empresa de telefonía CELLNEX TELECOM. https://www.cellnextelecom.com/

[8] Red de sensores inalámbricos. http://www.ni.com/white-paper/7142/es/

[9] *Wireless Global Standards.*
http://www.rfidc.com/docs/introductiontowireless_standards.htm

[10] *Low-power wide-area network*. https://www.i-scoop.eu/internet-of-things-guide/lpwan/

[11] Tecnología GPRS. https://www.pctechguide.com/mobile-communications/gprs-technology

[12] Bluetooth Smart. http://www.radio-electronics.com/info/wireless/bluetooth/what-is-bluetooth-smart-low-energy-ble.php

[13] Tecnología Zigbee. http://www.zigbee.org/

[14] Empresa Efor. (2017). Tecnologías de comunicación para IoT.
Disponible en: https://www.efor.es/sites/default/files/tecnologias-de-comunicacioon-para-iot.pdf

[15] Empresa ARM. https://www.arm.com/

[16] Empresa Intel. https://www.intel.es

[17] Empresa Atmel. https://start.atmel.com/

[18] Empresa Arduino. https://www.arduino.cc/

[19] Microcontrolador STM32L0 Discovery. https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html

[20] Microcontrolador Arduino MKR FOX 1200. https://store.arduino.cc/arduino-mkrfox1200

[21] Módulo Sigfox IOTEAM's Single Chip. https://partners.sigfox.com/products/sigfox-module

[22] Plataforma en la nube Sofia 2. http://sofia2.com/

[23] Plataforma en la nube Fiware. https://www.fiware.org/

[24] Plataforma en la nube Azure. https://azure.microsoft.com/en-us/

[25] Plataforma en la nube Samsung Artik. https://www.artik.io/

[26] Tecnologías de comunicación móvil. https://www.universidadviu.es/evolucion-la-red-comunicacion-movil-del-1g-al-5g/

[27] Modulación PSK. https://www.tutorialspoint.com/digital_communication/digital_communication_phase_shift_keying.htm

[28] Centro de investigación CEA-LETI. http://www.leti-cea.fr/cea-tech/leti/Pages/Accueil.aspx

[29] Encriptación AES-ECB. http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fecb.html

[30] Cobertura de Sigfox. https://www.sigfox.com/en/coverage

[31] Simulador NS-2. https://www.isi.edu/nsnam/ns/

[32] Simulador OMNET++. https://omnetpp.org/

[33] Simulador GloMoSiM. http://phdprojects.org/glomosim-simulator/

[34] Modelo de propagación Okumura-Hata. https://www.xirio-online.com/help/es/okumura-hata.htm

[35] Empresa Pycom. https://pycom.io/

[36] Editor Atom. https://atom.io/

[37] Contenedor de aplicaciones Apache Tomcat. http://tomcat.apache.org/

[38] Base de datos SQLite. https://www.sqlite.org/

[39] Analizador de paquetes WireShark. https://www.wireshark.org/

[40] Banda ISM. https://www.pcmag.com/encyclopedia/term/45467/ism-band

[41] Atta ur Rehman Khan, Sardar M. Bilal y Mazliza Othman. University of Malaya & University Carlos III de Madrid. “A Performance Comparison of Network Simulators for Wireless Networks”.