Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de tracking mediante LoRaWAN para embarcaciones de vela ligera

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Mario Cordero Sánchez

Tutor: José Vicente Soler Bayona

2018-2019

Resumen

Este TFG, describe la experiencia obtenida al realizar un sistema de tracking en tiempo real para embarcaciones de recreo. Este sistema hace uso de una nueva tecnología, de nombre LoRaWan, es una especificación para redes de baja potencia y área amplia, LPWAN (en inglés, Low Power Wide Area Network), diseñada específicamente para dispositivos de bajo consumo de alimentación, que operan en redes de tipo de alcance local, hasta global. Este estándar de red está llamando a ser la base para todo tipo de proyectos que vayan a hacer uso de *Internet de las Cosas*. Gracias a este nuevo tipo de protocolo y a los requerimientos del IC, se abre un nuevo mundo para el seguimiento de embarcaciones ligeras.

Palabras clave: LoRaWan, Internet de las Cosas, Seguimiento en tiempo real.

Abstract

This TFG, describe the experience obtained after doing the live-tracking design for a leisure crafts. This system uses a new technology, with name LoRaWan, is a type of wireless telecommunication wide area network designed to allow long range communications at a low bit rate among things.This standard is called to be the base system to create a controlled environment for Internet of Things. Thanks to this new protocol and IoT requirements, can be found a new environment projects for tracking system-based.

Keywords: LoRaWan, Internet of Things, live-tracking.

Tabla de contenidos

Figuras, Ecuaciones, Tablas y Apéndice

Prefacio

Motivación

Dado el auge en los últimos años del Internet de las Cosas o el Big Data, nos resultaba bastante interesante abordar un proyecto, que abarcase, aunque fuese superficialmente alguno de estos dos temas. Además, nuestro idea de proyecto está intrínsecamente ligada al mar, al cual tengo cierta afición, dado que mi familia suele realizar algún que otro viaje en él, pues cuando pasa esto, siempre quiero saber en que momento se encuentran, por si hay algún percance. Por este motivo, y salvando grandes distancias, diseñar un sistema de tracking marino, aunque sea precario, es un gran aliciente para la concepción de este trabajo.

En otro orden de cosas, para realizar este sistema de monitorización o tracking, podríamos haber empleado muchas tecnologías ya existentes, como podría ser la *GSM*, pero dado que se empezaba a resonar, se decantó la balanza por LoRaWAN, y quien sabe si la elección de esta tecnología nos pueda ayudar en el futuro laboral.

Objetivos

El objetivo principal de este trabajo es desarrollar y que sea funcional, un sistema de localización o tracking, enfocándose en el ámbito marino. Esta necesidad surge de la idea de tener ubicados todas las naves de una escuela náutica, realizar un seguimiento, ver el histórico de cada embarcación de recreo…

Otro de los principales requisitos sería además que esta información sobre su posición fuese en tiempo real, que inicialmente nos parece razonable, pues existen otros métodos de tracking de vehículos, como puede ser para naves o aviones, pero en un entorno comercial.

Por último, nos interesaría, que nuestro sistema tuviese un gran alcance, pues para hacer un seguimiento de una regata, las embarcaciones de vela ligera no pasan precisamente cerca de las boyas que marcan el recorrido, sino que lo hacen a una relativa distancia, de la misma forma, que la idea inicial, no consiste en colocar un servidor de localización, más adelante nombrado como *gateway*, sino colocarlo en ciertas boyas estratégicas que abarquen la totalidad del recorrido. Por otro lado, y dada la idea inicial, si se pretende colocar un sensor geolocalizador, o *nodo*, en cada embarcación de vela.

Introducción

El Internet de las Cosas, o en inglés Internet of Things (Iot), surge de las nuevas y prometedoras características de un amplio abanico dispositivos, que permiten interconectarse, y a su vez crear una red capaz de gestionar a éstos. Gracias a las posibilidades de este concepto, es posible trabajar con una infinidad de datos de diferentes tipos de tecnologías, tanto a nivel físico como de enlace.

En este entorno, encontramos las Low-Power Wide Area Networks (LPWANs), que actualmente se encuentran en proceso de emerger, y por este motivo, apunta, no sólo en el ámbito personal sino también en el empresarial, como un posible estándard para el futuro. Pese a que estas redes las LPWAN, presentan algunas limitaciones, como son el tamaño de las tramas, el ancho de banda o la tasa de mensajes, que pese a ser significativas, ofrece grandes ventajas, como su bajo coste, conectividad inalámbrica y baja potencia.

Las LPWAN se pueden usar para crear una red privada de sensores inalámbricos, pero también pueden ser un servicio o infraestructura ofrecida por un tercero, lo que permite a los propietarios de sensores implementarlos en el campo sin invertir en tecnología de puerta de enlace. Por este motivo, es fácil pensar que este tipo de entornos, las de máquina a máquina (p2p en inglés) serán dominadas por las LPWAN, en declive de las redes móviles, que en comparación, su coste es mayor y de menor alcance y autonomía.

Aunque hay otros estándares como LTE-Cat M, IEEE P802.11ah (WiFi de baja potencia) y Dash7 Alliance Protocol, en la siguiente sección analizaremos brevemente cada una de las tecnologías y enumerarán los pros y los contras de los métodos más implementados:

	Ventajas	Inconvenientes
sigfox	✓ *El de mayor tracción* ✓ *Excelente relación con proveedores* ✓ *Sin circuitos RX =>Ahorro Energía* ✓ *Ideal para monitoreo y medición*	× *No es un protocol abierto* × *Mínima seguridad incorporada* × *Casos de Uso limitados* × *Niveles altos de Interferencia RF* × *Reglamentacción FCC según Región*
LoRa	✓ *Miembros de reconocido prestigio (Cisco, IBM..)* ✓ *Mayor Seguridad* ✓ *Tamaño del paquete definido por el Usuario* ✓ *Implementación más popular junto con Sigfox*	× *No está destinado a redes privadas* × *Capacidad de enlace limitada* × *Limitado a vendedores aprobados por Semtech* × *Protocolo tipo ALOHA todavía complicado* × *Reglamentacción FCC según Región*
LinkLabs	✓ *Alta sensibilidad* ✓ *Límite de frecuencia flexible (150 Mhz a 1 Ghz* ✓ *Capaz de operar sin servidor de red*	× *Requiere software Sympony Link* × *Comunidad de Usuarios Escasa*
INGENU	✓ *Buenatecnología apilada* ✓ *Alta cobertura y robustez* ✓ *Emergiendopese a su entrada tardía al Mercado*	× *Interferencia tanto Wifi como Bluetooth* × *Dificultad de penetración structural* × *Criterios de potencia pueden no ser los óptimos energéticamente*.
WEIGHTLESS™	***N*** ✓ *Ideal para redes de sensors* ✓ *Buen rango urbano* ✓ *Estándar Abierto* ***P*** ✓ *Comunicación Bidireccional* ✓ *Velocidad datos adaptable* ✓ *Estaándar abierto*	× *Capacidad de enlace descendiente* × *Muy lento (100bps)* × *Requiere oscilador de cristal con compensación de temperature* × *Disposición de hardware limitada* × *Menor escalabilidad que la N* × *Rango de comunicaciones limitado*

Tabla 1 Comparativa Tecnologías LPWAN

Vistos los pros y contras anteriores para las más conocidas tecnologías LPWAN, y buscando que el despliegue sea lo más económico posible, nos encontramos con un escenario ideal para LoRa/LoRaWAN. Con el fin que tanto su despliegue y configuración sean los idóneos, necesitaremos conocer tanto aspecto como configuración de dicho paradigma.

Estructura

A lo largo de este documento dividido en cuatro capítulos, se analizarán las diferentes características que ofrece la tecnología LoRaWAN con tal de evaluar su uso en una red de tracking para embarcaciones de vela ligera, mostrando a su vez pruebas en un escenario real.

En el primer capítulo, se introducirá la tecnología LoRa y profundizará a su vez en el protocolo LoRaWAN, donde se nos mostrarán matices clave para comprender el resto de los capítulos.

Ya en el segundo capítulo, veremos las especificaciones técnicas de nuestro entorno de trabajo, así como una visión preliminar del software utilizado para implementar nuestra red LoRaWAN.

Más adelante, en el tercer capítulo, ahondaremos en el diseño de nuestra solución, acomodando el escenario con los datos necesarios para comprender este documento.

Posteriormente, en el cuarto capítulo, presentamos la implementación a través de los 3 entornos abordados en el proyecto, el gateway, el IDE de Arduino, y la aplicación Android.

En el quinto capítulo, ve presentarán los resultados obtenidos en las pruebas realizadas, tanto en la universidad, como en el escenario real. La descripción de los pasos llevados a cabo para la implementación y configuración de los nodos LoRaWAN, así como el Gateway, se traslada a los anexos como ocurría en el capítulo anterior.

A continuación del quinto capítulo, se expondrán las conclusiones extraídas de este proyecto para la concepción de una red de tracking para pequeñas embarcaciones, así como la relación con los estudios cursados en el grado.

Para finalizar, tras las conclusiones y la relación con los estudios cursados en el grado, se presentarán los anexos anteriormente citados, en los que podremos consultar los pasos llevados a cabo desde la preparación hasta la configuración de los diferentes elementos de nuestro sistema de tracking LoRaWAN. Dichos anexos irán acompañados de explicaciones para facilitar la reproducción y creación de este sistema por parte del lector.

Capítulo 1: Tecnología LoRa

En este primer capítulo, dotaremos de perspectiva a la tecnología Lora, dando a conocer el principal objetivo de este Proyecto, la especificación LoRaWAN. Además, servirá para dar a conocer los conocimientos previos que faciliten la comprensión de los capítulos posteriores. Dentro de las Low Power Wide Area Network, la tecnología LoRa, ha conseguido posicionarse como una de las principales, y esto es gracias a:
-Servicios de Comunicación bidireccionales
-Facilidad de Interoperabilidad entre sensores sin necesidad de instalaciones complejas.
-Facilidad de despliege tanto para empresas como a desarrolladores en la IoT.
La Tecnología LoRa define una capa física patentada por Cycleo en el 2008, aunque posteriormente fue adquirida por Semtech en 2012, y emplea la radio frecuencia sub-gigahercio sin licencia, como son 169, 433 y 868 MHz para Europa y la 915 MHz para Norte América. Por otro lado, la especificación LoRaWAN es una propuesta que nace de la LoRa Alliance en Junio del 2015, y ofrece una capa MAC basada en la modulación LoRa.

1.1. LoRa

LoRa es una técnica de modulación, cuyo nombre nace de la abreviatura de Long Range, que está basada en técnicas de espectro ensanchado, a.k.a Chirp Spread Spectrum (CSS), y es una variación que modula los datos sobre diferentes canales y velocidades, con corrección de errores integrada, a.k.a Forward Error Correction (FEC). Lora utiliza toda la anchura de banda del canal, generando una mejora significativa en la sensibilidad del receptor, como ocurre en otros tipos de CSS, pero a diferencia de éstas, compensa el ruido generado en las frecuencias del canal por el uso de cristales de bajo coste. LoRa, a diferencia de la mayoría sistemas de desplazamiento de frecuencia, a.k.a Frequency Shift Keying (FSK), que para demodular adecuadamente la señal requiere de entre 8 y 10 dB por encima del nivel de ruido, ésta puede trabajar con señales de 19.5dB por debajo del nivel de ruido.
LoRa posee las mismas características de las modulaciones FSK, pero se fundamentada en una modulación CSS, pero ganando ampliamente en el alcance de la comunicación… Todo esto, alcanzando velocidades de transmisión desde los 0.3 kbps hasta los 38.4 kbps, gracias a la sensibilidad del receptor por la disposición de codificación. Por este motivo, podemos establecer que este sistema es adecuado para comunicaciones directas entre nodos, a.k.a Peer to Peer (P2P).

1.1.1 Red LoRa

En el modo P2P, los nodos pueden conectarse directamente entre ellos y enviar mensajes directamente sin ningún costo (ya que no utiliza la red de LoRaWAN, sino que sólo dirigen comunicación por radio). Esta tipología de red es útil ya que podemos crear redes secundarias en cualquier momento, sin por ello cambiar el firmware, únicamente hemos de utilizar comandos AT específicos en la biblioteca actual. Este modo funciona sin la necesidad de una estación Base o una cuenta Cloud en caso de que no se desee comprar

licencia alguna, o renovar la licencia después del período inicial, y podrá seguir utilizando los módulos de esta manera.

Fig. 1 Tipología Peer to Peer redes LoRa

1.1.2 Bandas de Frecuencia y Canales

LoRa emplea diferentes bandas de frecuencia según la región en la que nos encontremos, como pueden ser Estados Unidos, Europa o China. Hay que tomar en cuenta que tanto el Gateway como el dispositivo destino pueden usar la misma frecuencia para la transmisión, pero deben ocupar distintos time slots. Este concepto se conoce como duplexación por división de tiempo, a.k.a. Time-Division Duplexing (TDD). Dicho de otro modo, existen limitaciones que establecen el tiempo máximo que puede estar el transmisor active o transmitiendo. La red LoRaWAN impone una limitación del ciclo de trabajo por cada sub-banda en la que una trama se transmite en una sub-banda dada, donde el tiempo de la emisión y la duración en el aire de la trama se registran para esta sub-banda. La misma sub-banda no puede ser utilizada de nuevo durante los próximos Toff segundos, donde:

Ecuación 1

$$T_{off} = \frac{TimeOnAir}{DutyCycle_{subband}} - TimeOnAir$$

Region	Banda Frecuencia LoRa	Canal Frecuencia LoRa
UE	863 a 870 MHz	○ 868.10 MHz (utilizado por Gateway para escuchar) ○ 868.30 MHz (utilizado por Gateway para escuchar) ○ 868.50 MHz (utilizado por Gateway para escuchar) ○ 864.10 MHz (utilizado por el dispositivo final para transmitir solicitud de unión) ○ 864.30 MHz (utilizado por el dispositivo final para transmitir solicitud de unión) ○ 864.50 MHz (utilizado por el dispositivo final para transmitir solicitud de unión) ○ 868.10 MHz (utilizado por el dispositivo final para transmitir solicitud de unión) ○ 868.30 MHz (utilizado por el dispositivo final para transmitir solicitud de unión) ○ 868.50 MHz (utilizado por el dispositivo final para transmitir solicitud de unión)
USA	902 a 928 MHz	○ 902.3 MHz a 914.9 MHz espaciados a 200KHz (canales Upstream-64) ○ 903 MHz a 914.2 MHz espaciados a 1.6 MHz de separación (Upstream- 8 canales) ○ 923.3 MHz a 927.5 MHz espaciados a 600KHz de separación (Downstream- 8 canales)
CN	770 a 787 MHz	○ 779.5 MHz (canal predeterminado) ○ 779.7 MHz (canal predeterminado) ○ 779.9 MHz (canal predeterminado) ○ 779.5 MHz (Usado por ED para transmitir solicitud de unión) ○ 779.7 MHz (Usado por ED para transmitir solicitud de unión) ○ 779.9 MHz (Usado por ED para transmitir solicitud de unión) ○ 780.5 MHz (Utilizado por ED para transmitir solicitud de unión) ○ 780.7 MHz (Usado por ED para transmitir solicitud de unión) ○ 780.9 MHz (Utilizado por ED para transmitir solicitud de unión)

Tabla 2 Frecuencias por Región para tecnologías LoRa

ETSI ha definido una banda de frecuencia de 433 a 434 MHz para la aplicación LoRa.
Utiliza los canales de frecuencia 433.175 MHz, 433.375 MHz y 433.575 MHz.
Los dispositivos Clase B utiliza canal de frecuencia 869.525 MHz en la banda de la UE.

1.1.2.1 Canal en la Unión Europea

La banda se centra en las frecuencias desde 863 a 870 MHz. En la table siguiente podremos observar la relación existente entre el número del canal y la frecuencia canal respectivamente. Estos 8 canales que la conforman se encuentran separados con un margen de 0.3 MHz con sus canales adyacentes.

Channel Number	LoRa Center Frequency
CH_10_868	865.20 MHz
CH_11_868	865.50 MHz
CH_12_868	865.80 MHz
CH_13_868	866.10 MHz
CH_14_868	866.40 MHz
CH_15_868	866.70 MHz
CH_16_868	867 MHz
CH_17_868	868 MHz

Tabla 3 Relación Canal - Frecuencia LoRa para Europa

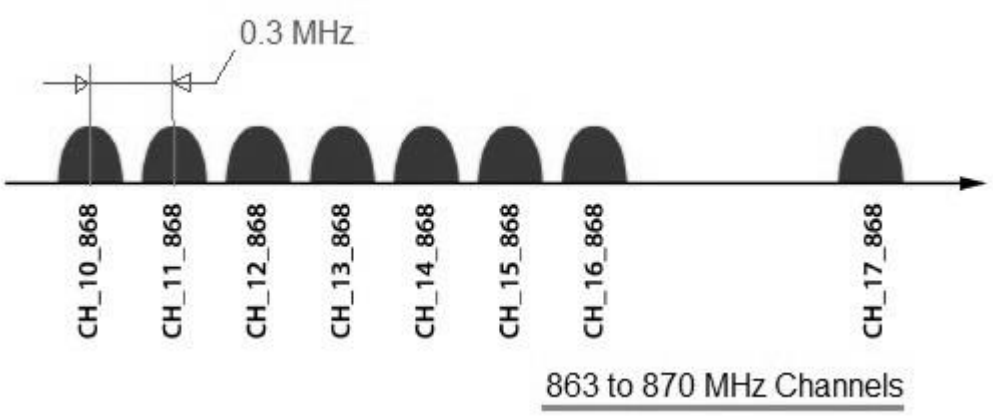


Fig. 2 Representación Canales Lora Europa

1.2. LoRaWAN

A diferencia de LoRA que ofrece una modulación cuya funcionalidad es de capa física (PHY), LoRaWAN es un protocolo MAC estandarizado para las LPWAN por parte de la LoRA Alliance, y que sus principales características son su gran capacidad y su largo alcance, así como su red en estrella. LoRaWAN define su propio protocolo MAC y arquitectura del Sistema de red, determinando así de forma decisiva la vida útil de la batería de un nodo, la calidad del servicio ofrecido, la naturaleza de sus aplicaciones en dicha red, y su seguridad. Además, puede emplear tanto una modulación FSK a nivel físico como emplear LoRa. Por otro lado, LoRa "sólo" permite el enlace de comunicación de largo alcance a nivel físico, lo que hace que no disponga de normas o reglas más allá de la propia naturaleza de su capa física o de las librerías utilizadas.

Dado que LoRaWAN optimiza la duración de la batería en sensores de bajo coste además de mejorar la compensación entre latencia de red y vida útil de la batería en los diferentes tipos de nodos. Desde el punto de vista de la arquitectura, observamos totalmente bidireccional, además de fiable y segura. Además, está diseñada tanto para rastrear como localizar objetos móviles, por este motivo, está siendo desplegado por diversos operadores de telecomunicaciones para sus redes móviles, y gracias a la LoRa Alliance, se nos asegura que dichas redes, son interoperables.

1.2.1 Red LoRaWAN

Por lo general, la disposición de las redes LoRaWAN es de topología Estrella-Estrella o dicho de otro modo, la arquitectura típica es una red de Redes en Estrella, de forma que la primera estrella está formada por los dispositivos finales y las puertas de enlace, y la segunda estrella está formada por las puertas de enlace y un servidor de red central. En este caso las puertas de enlaces son un puente transparente entre los dispositivos finales y el servidor de red central. La comunicación entre los componentes (nodos y gateways) puede ser unidireccional o bidireccional; por otro lado, los gateways se conectan al servidor empleando una conexión IP estándard, mientras que los nodos y el gateway lo hacen utilizando un enlace directo mediante LoRa o FSK a nivel físico.
Gracias a esto, los dispositivos LoRaWAN se pueden manipular de dos formas: en modo P2P, como en las redes LoRa, o en modo híbrido. La principal ventaja del modo híbrido es que maneja una combinación de modos P2P y LoRaWAN que permite enviar mensajes utilizando redes LoRaWAN. El mayor inconveniente, es que este tipo de red requiere de licencia LoRaWAN. En este modo, el nodo central, que actúa como gateway que se mantiene a la escucha y actúa de enlace encaminando la información, empleando la tecnología LoRaWAN, y el resto de nodos, trabajan en modo P2P dispuestos en estrella.

1.2.2 Modos de Conexión

Existen 2 maneras de unirse a una red LoRaWAN:

- **Over-the-Air Activation (OTAA):** Es el método más seguro para conectarse a una red, además de ser el más utilizado, como por ejemplo en The Things Network. Los dispositivos realizan un procedimiento "join" o unión con la red, en el que se asigna de forma dinámica un DevAddr y las claves de seguridad se negocian con el dispositivo.

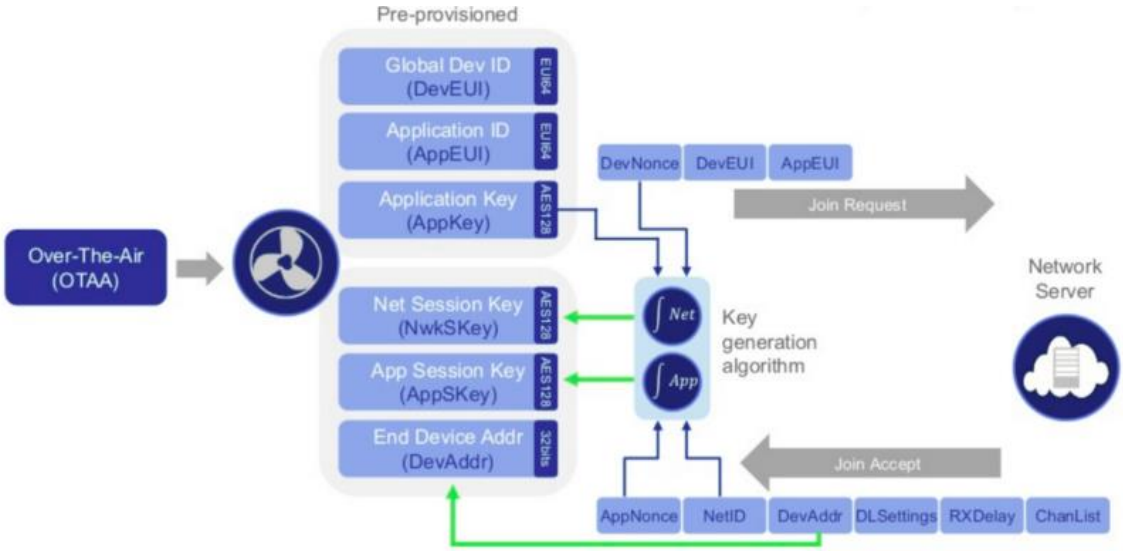


Fig. 3 Esquema Over-the-Air Activation

- **Activation by Personalization (ABP):** Existen casos en los que se necesita codificar el DevAddr, así como las claves de seguridad del dispositivo. Esto significa activar un dispositivo por personalización. Esta estrategia a priori puede parecer más simple al omitir el procedimiento de "join", pero tiene diversas desventajas relacionadas con la seguridad.

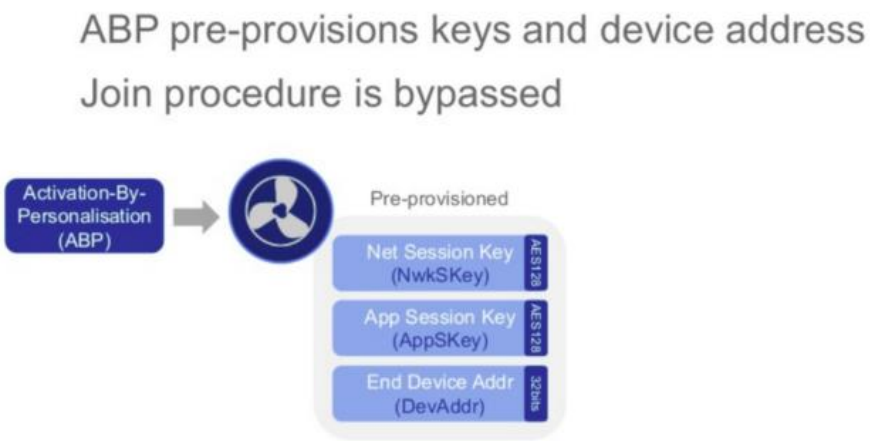


Fig. 4 Esquema Activation-By-Personalisation

1.2.3 Clases LoRaWAN

Encontramos 3 clases de dispositivos LoRaWAN, de la A a la C. Todos los dispositivos implementan, al menos, la funcionalidad de clase A, pero además, pueden implementar las opciones de clase B o C.

- **Clase A:**

  La más soportada en casi todos los dispositivos, este tipo de clase ofrece el mayor ahorro de energía debido a que solo entra en modo escucha (llamado ventana RX) después de enviar un dato hacia el gateway, por eso es ideal para dispositivos que usan una batería. Permiten una comunicación bidireccional, con la limitación de que sólo pueden recibir datos (canal *downlink*) si han enviado antes un paquete (canal *uplink*).

- **Clase B:**

  Este tipo de dispositivos tiene las ventanas de recepción con base a tiempos predeterminados con el gateway, este tipo de nodos puede usar una batería o una fuente externa dependiendo de los tiempos asignados de escucha. Esto consigue mediante el envío periódico de *beacons* por parte de la puerta de enlace. Estos beacons permiten a los dispositivos estar sincronizados con el gateway, y de esta forma pueden negociar tiempos de recepción de paquetes desde la puerta de enlace al dispositivo (downlink). Esta clase de dispositivos tendrán un consumo mayor de energía que los de clase A debido a la recepción periódica de los beacons desde el gateway.

- **Clase C:**

  Este tipo de clase ofrece el menor ahorro de energía debido a que siempre está en modo escucha, y sólo cuando es necesario en modo transmitir, consiguiendo con ello mejores tiempos de respuesta y capacidad de envío desde el servidor a los dispositivos. La recomendación es usarlo en dispositivos que cuentan con una fuente externa de alimentación.

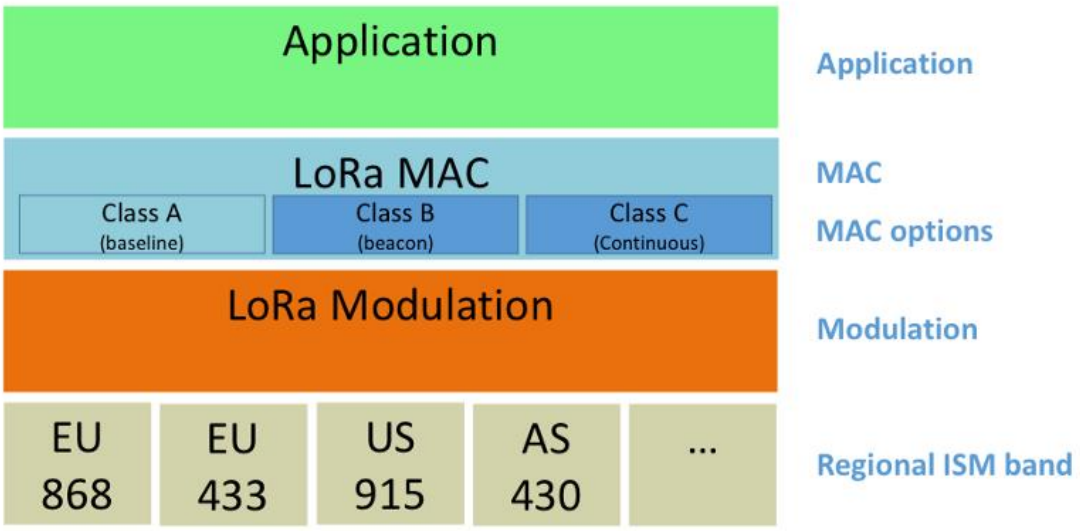


Fig. 5 Pila Protocolo LoRaWAN

1.2.4 Seguridad

Dado que la información que puede contener los paquetes requiere ser protegida, es indispensable, a la par que necesario, tartar el tema de la seguridad en las redes LoRaWAN. Por esto, LoRaWAN hace uso del algoritmo de cifrado AES128 en sus diversas capas, con sus correspondientes claves.

Nombre	Codificación Clave	Seguridad	Uso en Conexiones
Network Session Key	128 bits	A nivel de Red	OTAA / ABP
Application Session Key	128 bits	De extremo a extremo a nivel de Aplicación	OTAA / ABP
Application Key	128 bits	De extremo a extremo a nivel de Aplicación	OTAA

Tabla 4 Claves de Seguridad LoRaWAN

1.3. Características Red LoRaWAN

Por tratarse de un protocolo estandarizado, LoRaWAN dispone de características propias de su especificación, permitiendo interoperabilidad entre los propios dispositivos que la implementen, sin importar que hagan uso de librerías externas.

En nuestro Proyecto utilizamos el Kit Seeduino LoRaWAN[i], que describiremos más adelante, pero a modo de ejemplo, describiremos en función del módulo RHF0M301-868, que es quién conforma nuestro gateway.

1.3.1 Potencia Tx

Podemos configurar la potencia a través de la especificación LoRaWAN usando comandos de la MAC. Estos sería para la banda EU868.

Potencia Tx	Salida (dBm)
0	20
1	14
2	11
3	8
4	5
5	2
6-15	RFU

Tabla 5 Relación Potencia – Salida

1.3.2 Limitación Ciclo de Trabajo

Únicamente la banda EU868 necesita activar la limitación del ciclo de trabajo para cumplir con la norma standard ETSI [EN300.220] La banda y su limitación tal como sigue:

Índice Banda	Frecuencias (MHz)	Potencia Máxima	Ciclo Trabajo	Amplitud Banda
g	865.00 ~ 868.00	14 dBm	1%	3MHz
g1	868.00 ~ 868.60	14 dBm	1%	600 KHz
g2	868.70 ~ 869.20	14 dBm	0.1%	500 KHz
g3	869.40 ~ 869.65	27 dBm	10%	250 KHz
g4	869.70 ~ 867.00	14 dBm	1%	300 KHz

Tabla 6 Banda Limitación para Europa

1.3.3 Estructura de Paquetes

En LoRaWAN se encuentran dos tipos de mensaje de downlink y uplink:

- Los mensajes de *uplink* los envían los dispositivos finales al servidor de red, utilizando uno o varios gateways como intermediarios, estos mensajes utilizan el modo explícito de paquetes de radio.

- Los mensajes de *downlink* son enviados por el servidor de red a un solo dispositivo final pasando por un solo gateway. Estos mensajes utilizan el modo explícito de paquete de radio, en el que se incluye el encabezado físico LoRa

Todos los mensajes LoRaWAN formados con un campo *preamble* de 8 bytes de longitud, una cabecera (PHDR) y el *Payload*, los dos últimos con su CRC (PHDR_CRC y CRC), eliminando el este último CRC en el caso de los downlinks:

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

Fig. 6 Formato Frame LoRaWAN

El Payload de capa física cuenta con una cabecera MAC, el *MAC Payload* y un Message Integrity Code, un código de cuatro bytes que se calcula a partir de la *Network session key (NwkSKey).*

MHDR	MACPayload	MIC
1 byte	1-M bytes	4 bytes

Fig. 7 Estructura Payload

La cabecera MAC especifica el tipo de mensaje y la versión del formato de la trama de la especificación de la capa LoRaWAN con la que ha sido codificada. Existen seis tipos de mensajes MAC.

MType	**Descripción**
000	Join Request
001	Join Accept
010	Unconfirmed Data Up
011	Unconfirmed Data Down
100	Confirmed Data Up
101	Confirmed Data Down
110	RFU7
111	Proprietary

Tabla 7 Tipo Mensajes MAC

El *MACPayload* incluye una cabecera de trama, un campo de puerto opcional y un campo de *Payload* de trama opcional.

FHDR	FPort	FRMPayload
7-23 bytes	0-1 bytes	0-N bytes

Tabla 8 Formato Frame MACPayload

La cabecera de trama contiene la dirección con la que se identifica el dispositivo dentro de la red, un campo *FCtrl* para habilitar el *Adaptive data rate*, un contador de tramas y un campo *FOpts* en el caso de que se deseé transmitir un comando MAC. El campo *FPort* sirve para determinar si el campo *FRMPayload* contiene comandos MAC o datos de la aplicación.

Los siete bits más significativos del campo *DevAddr* se utilizan para el identificador de red (*NwkID*) mientras que los veinticinco restantes corresponden a la dirección de red (*NwkAddr*), la cual puede ser asignada por el administrador de la red.

DevAddr	FCtrl	FCnt	FOpts
4 bytes	1 byte	2 bytes	0-15 bytes

Tabla 9 Formato Frame Cabecera

El tamaño máximo del campo *MACPayload* varía según la banda de frecuencia sobre la que se trabaja, el data rate y la ausencia del campo de control (*FOpts*), en el caso de Europa (frecuencia de 868 MHz) sería la siguiente.

DataRate	**M (bytes)**
0	59
1	59
2	59
3	123
4	230
5	230
6	230

Tabla 10 Tamaño Payload Europa

1.3.4 Comunicación entre nodos

La comunicación en LoRaWAN se efectúa entonces de nodos a gateway y de gateway al servidor de red. El gateway es el encargado de convertir los paquetes LoRaWAN en paquetes UDP y viceversa. El nodo puede trasmitir tantos mensajes seguidos como números de canales tenga habilitados. Cuando el nodo transmite un mensaje con confirmación, este llega al gateway, que convierte el mensaje en un paquete UDP y lo reenvía al servidor de red. Aunque el mensaje no requiera confirmación, el gateway sí recibe respuesta del servidor de red. Una vez ha transmitido por el último canal

disponible, el nodo no podrá volver a transmitir hasta pasado el ciclo de trabajo de cada canal.

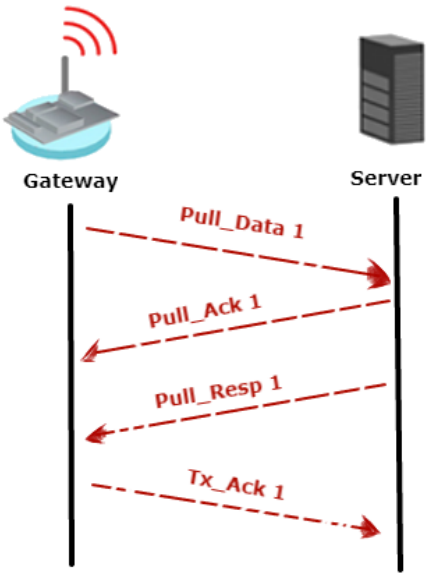


Fig. 9 Comunicación Downstream Gateway/Server

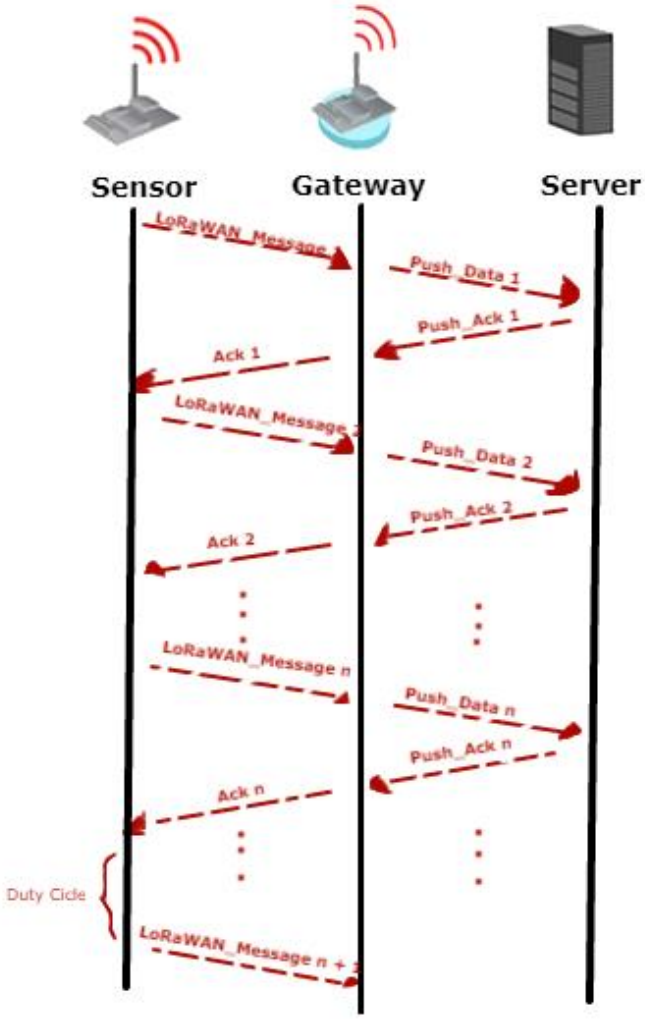


Fig. 8 Comunicación Upstream Sensor/Gateway/Server con confirmación

Capítulo 2: Entorno de Trabajo

En este capítulo se describe el material empleado que hace uso de las tecnologías expuestas en el capítulo anterior, además se detalla las características relevantes del dispositivo final (nodo/sensor) utilizado y el proceso de establecer la comunicación entre los elementos de nuestra red, con el fin de pormenorizar el escenario de estudio que recogerá este documento.

2.1. Componentes LoRa LoRaWAN Gateway - 868MHz Kit con Raspberry Pi 3 de SeeedStudio

Para construir tu propia red LoRaWAN hay 3 cosas necesarias: Un Gateway, almenos 1 nodo, y un servidor con lo que poder monitorizar nuestros dispositivos. Este Kit de SeeedStudio nos provee de todos los componentes necesarios, desde una Raspberry Pi 3 hasta cables Ethernet.

Fig. 10 SeeedStudio LoRa LoRaWAN Gateway - 868MHz Kit con Raspberry Pi 3

2.1.1 Raspberry Pi 3

Raspberry Pi es un computador de placa reducida, computador de placa única o computador de placa simple (SBC) de bajo costo desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas.
Hemos optado por la revisión o modelo 3, que vió la luz en el año 2016, cuyas características de hardware se enuncian a continuación:

- **Sistema en Chip**: Broadcom BCM2837 (CPU + GPU + DSP + SDRAM + Puerto USB)
- **CPU**: 1.2GHz 64-bit quad-core ARMv8
- **Juego de Instrucciones**: RISC de 64 bits
- **GPU**: Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC3
- **Memoria (SDRAM)**: 1 GB (compartidos con la GPU)
- **Puertos USB 2**.0: 4
- **Entradas de Vídeo**: Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la RPF
- **Salidas de Vídeo**: Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD
- **Salidas de Audio**: Conector de 3.5 mm, HDMI
- **Almacenamiento Integrado**: MicroSD
- **Conectividad de red**: 10/100 Ethernet (RJ-45) vía hub USB, Wifi 802.11n, Bluetooth 4.1
- **Periféricos de Bajo Nivel**: 17 x GPIO y un bus HAT ID
- **Reloj en Tiempo Real:** Ninguno
- **Consumo energético**: 800 mA, (4.0 W)
- **Fuente de Alimentación**: 5 V vía Micro USB o GPIO header
- **Dimensiones**: 85.60mm × 53.98mm (3.370 × 2.125 inch)
- **Sistemas Opertativos Soportados**: GNU/Linux: Debian (Raspbian), Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE Linux Enterprise Server for ARM. RISC OS2.

En este caso en particular, hemos optado por una MicroSD de 8GB Micro SD Card – Class 10, con el Sistema Operativo Raspbian.

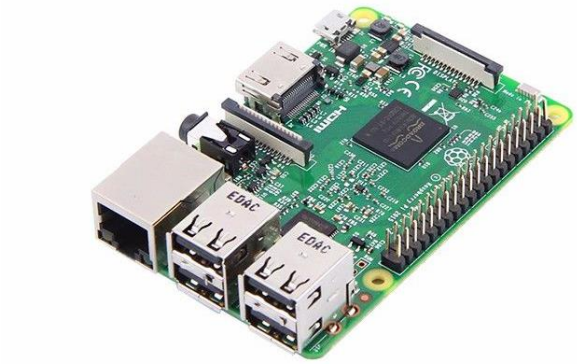


Fig. 11 Raspberry Pi 3

2.1.2 PRI 2 bridge RHF4T002

Este componente tiene la finalidad de ser el punto de unión entre la Raspberry Pi y el módulo "Gateway" LoRa RHF0M301, consiguiendo así un LoRaWAN Gateway multicanal.

Fig. 12 PRI 2 Bridge RHF4T002

2.1.3 Gateway module RHF0M301 – 868

Este módulo Gateway es un concentrador de 10 canales. (8x Multi-SF + 1 s Standard LoRa + 1 FSK) El módulo está integrado en un hearder DIP de 24 pins, que facilitando la integración del propio módulo con nuestra plataforma personalizada.

Fig. 13 Módulo RHF0M301

Voltaje:	menos 6V
Temperatura de Funcionamiento:	-40°C a +85°C
Dimensiones:	40 x 63 mm
Potencia entrada RF:	less than -13dBm
Interfaz:	SPI

Tabla 11 Características RHF0M301

2.1.4 0dBi Rubber Duck Antenna

La antena utilizada en el módulo LoRa es un modelo especial de 0 dBi de ganancia.

Fig. 14 Antena Rubber Duck

2.1.5 Seeeduino LoRaWAN con GPS

Esta placa de desarrollo se basa en el módulo de comunicación RHF76-052AM, y es compatible con LoRaWAN de Clase A / C, además de compatible con unas diversas frecuencias de comunicación.

Los 4 conectores standard Grove aseguran una amplia compatibilidad entre diferentes módulos. Asimismo, la placa está dotada de un chip integrado de gestión de batería de litio que permite su carga a través de una interfaz USB. En el modo de bajo consumo, una batería de litio cargada puede alimentar la placa durante varios meses.

Procesador Arduino

- ATSAMD21G18 @ 48MHz con 3.3V
- Compatible con Arduino (basado en Arduino Zero bootloader)
- Con chip integrado para gestionar batería de litio y un led indicador de estado
- 20 GPIOs
- 4 conectores Grove
- 18 x PWM pins
- 6 x entradas analógicas
- 1 x salida analógica (A0)
- Regulador de 3.3V con salida 200mA
- Botón Reset.

Fig. 15 Seeeduino LoRaWAN con GPS

LoRaWAN/RHF76-052

- 1.45uA corriente en reposo o WOR mode (por el módulo, no por la placa)
- Estimado Alto enlace 160dB. -140dBm sensibilidad y 19dBm potencia de salida.
- Dual band, 434/470MHz y 868/915MHz
  - 19dBm@434MHz/470MHz
  - 14dBm@868MHz/915MHz
- Soporte protocolo LoRaWAN, Clases A/C
- Comunicación de largo alcance
- Consumo de energía ultrabajo
- Firmware actualizable
- Tamaño reducido: 23mm X 28mm con paquete de SMT de 33 pines.

2.2. Software

Es necesario disponer del software necesario tanto para programar los dispositivos (módulos LoRa/LoRaWAN, gateways y servidores) como para diseñar la red, y así conseguir el correcto funcionamiento de los módulos, programando e integrando así las librerías que requieran.

2.2.1 Arduino IDE

El sensor, o nodo, requiere de programación para ser configurado. En este punto, hemos optado por la última versión de Arduino IDE.

Para que funcione correctamente hemos de descargar el *driver* que nos facilita SeeedStudio para hacerlo funcionar en sistemas Windows.

Arduino IDE posee una interfaz amigable y sencilla, que nos ayuda a la programación de estos dispositivos, así como de su configuración del sensor, como son los citados parámetros anteriormente. (DevAdd, el modo de conexión, etc)

2.2.2 Raspbian

El propio Kit viene con una distribución de una distribución Linux optimizada para la Raspberry Pi, como es Raspbian, que se basa en una optimización de Debian para este tipo de placas. Inicialmente iba a ser nuestra elección para el Gateway LoRaWAN, pues puesto que con él podríamos incluir nuevas funcionalidades, o incluso personalizar más la configuración de éstas.

Capítulo 3: Diseño

Para agilizar el desarrollo e implementación, en vistas a desarrollar un entorno más ágil para la visualización de los datos, dimos con LoRaServer.io, un proyecto open-source para el despliegue de redes LoRaWAN. Dicha distribución se componía de un sistema Linux embebido podría ser eh utilizado en múltiples modelos LoRa entre éstos, se encuentra nuestra Raspberry Pi[ii]. Esta no iba a ser nuestra idea inicialmente, pues como hemos comentado en el apartado anterior, tanto la sd, como la wiki de rishing nos ofrecían un entorno en el que desplegar nuestro Kit, la red de The Things Networks, la cual llegamos a probar, pero descartamos por el modelo que vemos a continuación.

3.1. LoRa Gateway Os

En nuestro caso, optamos por la versión all-in-one, que comenzó su desarrollo en diciembre del 2018, que se componía de dos tipos:

- **Base**: Incluye los componentes mínimos para envío-recepción de paquetes, (el packet-forwarder de Semtech y Gateway-bridge) así como una utilidad CLI para la configuración de la puerta de enlace.
- **Full**: Además de las características ya proporcionas por la versión Base, no proporciona un entorno más completo al añadir LoRaServer y LoRaAppServer, que se ejecutan en el mismo entorno del Gateway.

La configuración inicial del sistema (en entorno Terminal de Linux), se podría realizar tanto directamente, (por medio de un teclado USB y un monitor) así como por consola a través de SSH. Para establecer el Hardware del cual se compone nuestra puerta de enlace (RisingHF - RHF0M301 LoRaWAN IoT Discovery Kit).

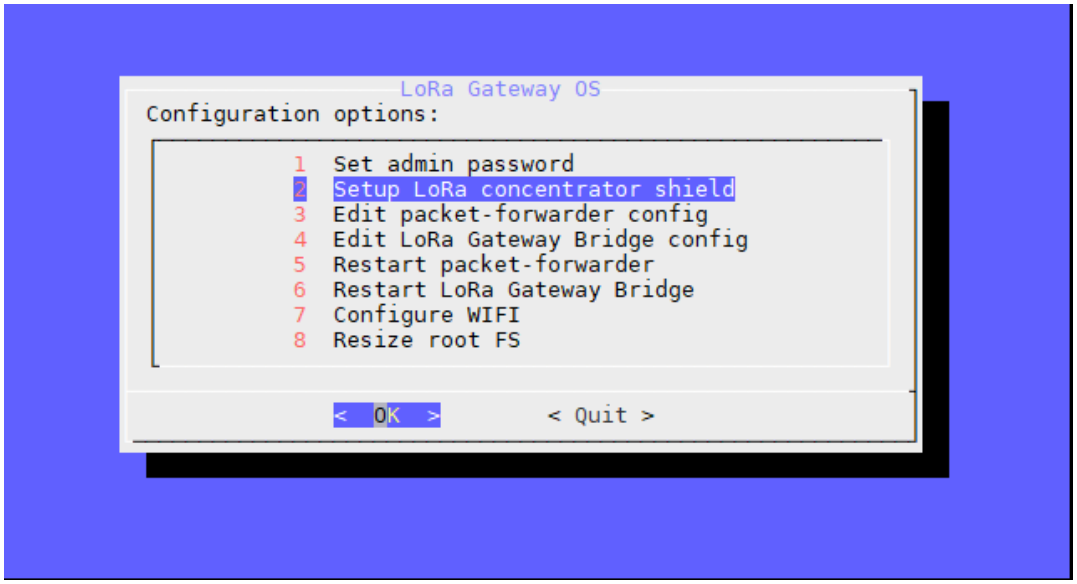


Fig. 16 Opción para configurar el concentrador en LoRa Gateway OS

1	Conectarnos al Gateway
2	Ejecutar el comando “gateway-config”
3	Seleccionamos nuestro concentrador (RHF0M301)
4	Elegimos el canal que usará (EU 868).

Tabla 12 Pasos a seguir para configurar Concentrador

Si todo ha ido correctamente, nos habrá saltado un *prompt* de que se ha cambiado la ID de nuestro Gateway (añade FFFE a mitad de la mac para completar 8 pares, es decir b827eb57278, pasa a ser b827ebFFFE57278, así como que se ha reiniciado el packet-forwarder.) Para comprobar que ha sido satisfactorio, tecleamos en la consola “monit status” y podremos verificar el estado de cada componente de la puerta de enlace.

```
Process 'loraserver'
  status                            OK
  monitoring status                 Monitored
  monitoring mode                   active
  on reboot                         start
  pid                               305
  parent pid                        1
  uid                               0
  effective uid                     0
  gid                               0
  uptime                            9d 1h 26m
  threads                           17
  children                          0
  cpu                               0.2%
  cpu total                         0.2%
  memory                            1.7% [15.6 MB]
  memory total                      1.7% [15.6 MB]
  security attribute                -
  disk read                         0 B/s [13.0 MB total]
  data collected                    Wed, 29 May 2019 15:21:40

Process 'lora-packet-forwarder'
  status                            OK
  monitoring status                 Monitored
  monitoring mode                   active
  on reboot                         start
  pid                               4755
  parent pid                        1
  uid                               0
  effective uid                     0
  gid                               0
  uptime                            1m
  threads                           5
  children                          0
  cpu                               0.2%
  cpu total                         0.2%
  memory                            0.1% [1.3 MB]
  memory total                      0.1% [1.3 MB]
  security attribute                -
  data collected                    Wed, 29 May 2019 15:21:40

Process 'lora-gateway-bridge'
  status                            OK
  monitoring status                 Monitored
  monitoring mode                   active
  on reboot                         start
  pid                               279
  parent pid                        1
  uid                               0
  effective uid                     0
  gid                               0
  uptime                            9d 1h 26m
```

```
  threads                     14
  children                    0
  cpu                         0.0%
  cpu total                   0.0%
  memory                      0.9% [7.9 MB]
  memory total                0.9% [7.9 MB]
  security attribute          -
  disk read                   0 B/s [7.4 MB total]
  data collected              Wed, 29 May 2019 15:21:40

Process 'lora-app-server'
  status                      OK
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  pid                         273
  parent pid                  1
  uid                         0
  effective uid               0
  gid                         0
  uptime                      9d 1h 26m
  threads                     16
  children                    0
  cpu                         0.4%
  cpu total                   0.4%
  memory                      2.4% [22.0 MB]
  memory total                2.4% [22.0 MB]
  security attribute          -
  disk read                   0 B/s [17.0 MB total]
  data collected              Wed, 29 May 2019 15:21:40

System 'raspberrypi3'
  status                      OK
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  load average                [0.37] [0.17] [0.11]
  cpu                         1.3%us 1.5%sy 1.2%wa
  memory usage                54.2 MB [5.8%]
  swap usage                  0 B [0.0%]
  uptime                      9d 1h 26m
  boot time                   Mon, 20 May 2019 13:55:34
  data collected              Wed, 29 May 2019 15:21:40
```

Tabla 13 Salida comando monit status

Con ello, además podemos observar los elementos que conforman el Gateway:

3.1.1 LoRaServer

Es el responsable de la deduplicación y manejo de las tramas uplink recibidas por el gateway, administración de la capa mac de LoRaWAN, y programar las transmisiones downlink.

3.1.1.1 LoRa Gateway Bridge

Es el servicio que se encarga de convertir los protocolos del LoRa packet-forwarder a protocolos entendibles por el servicio LoRa Server (JSON y Protobuf)

3.1.1.1.a Backends

- **Basic Station packet-forwarder**:
  Implementa el protocolo LNS. Expone un controlador de websocket al que se pueden conectar los gateways gracias a él.
  Soporta los siguientes tipos de autentificación (Sin Autentificación /Autentificación TLS Servidor/ Autentificación Cliente y Servidor TLS)
  Asigna un perfil Gateway a nuestro Gateway dentro de LoRa App Server. Es un requerimiento, y se encarga de realizar el handshake, enviando la configuración del plan de canales.

- **Semtech UDP packet-forwarder:**
  Se encarga de abstraer el protocol UDP, siendo compatible tanto con las Versiones 1 y 2, además de una modificación para Kerlink iBTS.
  Puede desplegarse tanto en el mismo gateway como "en la nube". En el primer caso, el beneficio radica en la capa de autentificación que dota MQTT y el TLS. En la nube, varios Gateways pueden conectarse a la misma instancia del Lora Gateway Bridge.

3.1.1.1.b Integraciones

- **Broker Genérico MQTT**
- **GCP Cloud IoT Core MQTT Bridge**

3.1.1.2 LoRaAppServer

Administra la parte de "inventario" del dispositivo en la infraestructura LoRaWAN, así como el manejo y cifrado tanto de las solicitudes de *join* como de los *payloads*.

Ofrece una interfaz web simple y ágil, donde podemos manipular desde usuarios hasta dispositivos, pasando por organizaciones y aplicaciones. Este servicio ofrece API RESTful y gRPC para la integración con servicios externos.

Los datos son escritos directamente en InfluxDB, y pueden ser tanto enviados como recibidos a través de MQTT y HTTP.

3.1.1.3 Raspberry Pi 3

Es la parte central del Gateway, la cual vertebra cada uno de los componentes de nuestro kit, ya sea a nivel de software como de hardware, como ya pudimos observar en el apítulo.

Los requisitos para el correcto funcionamiento de Gateway OS son:

- **MQTT broker**: publicar y recibir payloads de la aplicación.
- **PostgreSQL**: almacenamiento persistente de la Gateway.
- **Base Datos Redis**: el datastore de los datos no persistentes.

Para establecer la conexión a internet, llamaremos a la utilidad de configuración del Gateway desde el terminal, "gateway-config", y seleccionaremos la opción 6, que es configurar el WiFi. Esto nos redirigirá a "connmanctl", y únicamente hemos de seguir los pasos que nos indica:

1. Activar el uso de la red inalámbrica
2. Escanear redes wifi disponibles
3. Mostar el resultado
4. Activar el agente
5. Conectar a la red
6. Salir

Tabla 14 Pasos Configurar Wifi

Fig. 17 Pasos Configurar Wifi

3.2. Desarrollo

Una vez ya tenemos operativa nuestra Raspberry con LoRaGatewayOS plenamente operativo y conectado a internet, podemos abordar a las partes relativas al diseño, tanto en el entorno web, móvil, como api.

3.2.1 Entorno Web

Para el buen funcionamiento de nuestra red LoRaWAN, hay que configurar diversos parámetros a través de LoRa App Server. Para ello, haremos uso de su sitio web, que bastará con introducir la IP de nuestra Raspberry Pi (que previamente habíamos conectado a internet) con el puerto 8080.

Una vez logueados, por defecto tanto usuario como *password* es *admin*, procederemos a introducir la configuración de los perfiles (dispositivo, gateway y servicio) así como otros datos de interés (aplicación, organización, usuarios y lo relacionado con los perfiles anteriores)

3.2.2 Entorno móvil

Decidimos desarrollar aplicación para Android, ya que habíamos trabajado anteriormente con esta plataforma, tanto en el Grado cómo con las prácticas curriculares.

El entorno de trabajo para éste, escogimos medios open-source, cómo son GitHub para el repositorio y Atom para programar.

3.2.3 API

El propio entorno que crea LoRaServer nos brinda una serie de endpoints, (*ApplicationService, DeviceProfileService, DeviceQueueService, DeviceService, GatewayProfileService, GatewayService, InternalService, MulticastGroupService, NetworkServerService, OrganizationService, ServiceProfileService, UserServiceUserService*) útiles para la mayoría de los casos. Es accesible a través de la IP de la Raspberry, el puerto 8080 y la ruta "/api", y veremos dicha consola API, basada en Swagger UI. Hay que tener en cuenta que la mayoría de *endpoints* es necesario proporcionar un token JWT[iii] válido. El principal problema que presentan algunos *endpoints*, es que están destinados para el "*debugging*" y, por tanto, no son "utilizables" para nuestra aplicación móvil, por lo que tuvimos que hacer uso de la integración http y volcar los datos en otro sitio web que nos brindase la probabilidad de ser accesible externamente además de poder crear *endpoints*.

El otro entorno para las Apis se llama getsandbox[iv]. De hecho, es extremadamente fácil de manejar, dejando, entre muchas otras opciones, manipular su Base de Datos al vuelo o importar API ya creadas por otros medios (como SWAGGER) y así modificarlas o ampliarlas.

Para el uso que le vamos a dar, empezamos de cero, creando una serie de endpoints, en función de que los vayamos requiriendo. El código interno que utiliza es JavaScript, con un conjunto de librerías por defecto, así como importar otras que creemos conveniente.

Las librerías soportadas actualmente, en su Version 3 (2019+) - ECMAScript 2019, son:

- lo-dash v4.17.11 (Utilidad general)[v]
- momentjs v2.24.0 (Fechas y tiempo)[vi]
- faker 4.1.0 (Generador de Datos)[vii]
- ajv 6.10.0 (Validador de Esquemas JSON)[viii]
- validator 10.11.0 (Asistente de Validación)[ix]

Además, internamente, posee una consola, **Console**, para que veamos cómo se han procesado las peticiones, 2 vistas de código (**Overview**, que es la más simple, y dónde creamos inicialmente las rutas, y **Code**, dónde se presenta todo el código en sí), y la pestaña **State**, dónde sería las entradas de nuestra Base de Datos, presentada como su fueran Objetos.

3.3. Arquitectura

La arquitectura de nuestro entorno debería ser la siguiente:

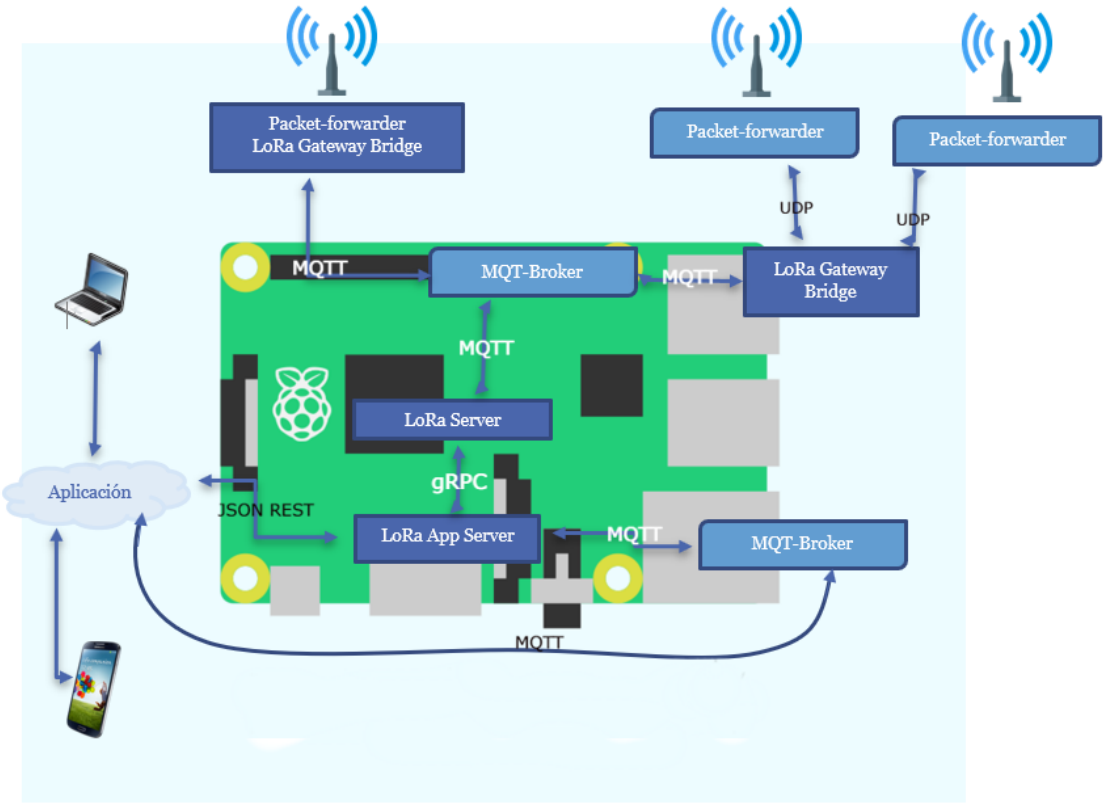


Fig. 18 Arquitectura del Sistema

Recibimos los datos de los nodos a través de Packet-forwarder, y el MQTT-Broker lo gestiona para que se reciba en el LoRa Server. Este último, ya sea directamente o a través nuevamente por el MQTT, solicita nuevos datos.

Consideramos aplicación tanto el entorno web, como la app para Android, pese a que su funcionamiento es bastante diferente. Las peticiones de éstos se hacen a través de las APIs, la interna del Servidor, así como otra realizada expresamente para recuperar los datos, que no son accesibles de otro modo.

3.4. Diseño Detallado

El funcionamiento del entorno parte de la premisa que toda configuración la vamos a realizar a través del entorno web, el cual veremos con más detalle en el capítulo 4, que es lo que conocemos como LoRaApp Server, y engloba tanto el api, como el entorno web.

La idea consiste en que, con el gateway en funcionamiento, nuestro sensor se comunique con éste, enviando los datos que hemos configurado. (posición, altura, velocidad, satélites…) El sistema es escalable, tanto en número de gateways, como de sensores. La comunicación de los dispositivos es gracias al packet-forwarder, que por medio del LoRaGateway Bridge se comunica con el Brocker-MQTT, para ser recogido por el LoRaServer. Debido a que *mosquitto* no se almacena de manera permanente, los datos a través de la implementación de la aplicación del server son recogidos en otro servidor para poder ser consultados a posteriori.

Luego, a través de la aplicación móvil, podemos visualizar los datos pertinentes a lo recogido mediante nuestro servidor, y operaciones de tipo CRUD[1] de usuario. Desde Android, empleamos tanto el api del LoRaApp Server, como la creada por nosotros.

En el Backend de la aplicación móvil, hemos diseñado un servicio con tal de gestionar lasa peticiones http que realizamos, se podrá consultar dicho fichero en el anexo, donde además veremos las rutas que contienen ambos entornos. Las peticiones son del tipo:

<table>
<tr><th>Tipo</th><th>Entorno</th><th>Parámetro</th><th>Headers</th></tr>
<tr><td>Delete</td><td rowspan="4">LoRaApp/Sandbox</td><td rowspan="2">null</td><td rowspan="4">Autorización JWT, tipo de contenido, Acepta</td></tr>
<tr><td>Get</td></tr>
<tr><td>Post</td><td rowspan="2">Objeto JSON</td></tr>
<tr><td>Put</td></tr>
</table>

Tabla 15 Peticiones HTTP

3.5. Tecnología Utilizada

Como hemos mencionado anteriormente, para el servidor hemos utilizado LoRa Gateway OS, que se trata de una recopilación de servicios utilizando sistema Linux embebido.

Para el desarrollo de la aplicación móvil, nos decantamos por el sistema Android, y para diseñar el backend, empleamos el framework Ionic, con Cordova. Nos decantamos por este, ya que si en un futuro quisiéramos migrar a otro entorno (Windows, iOS) podríamos realizarlo con los mínimos cambios. Además, se basa en Angular, que hace uso de TypeScript, que es una versión de JavaScript destinada a mejorar el trabajo por parte de los desarrolladores.

[1] *Create, Read, Update and Delete:* Crear, Leer, Actualizar y eliminar.

Finalmente, para la configuración del sensor, utilizaremos Arduino.

Todos estos apartados, serán pormenorizados en la implementación.

Capítulo 4: Implementación

Cómo hemos visto en el capítulo anterior, el proyecto se podría dividir en 2 partes, el Hardware (Gateway y nodo) y software (Entorno Web y Aplicación Web), pero nos vamos a centrar únicamente en la parte de Configuración del Software, ya que no tiene ningún misterio montar el Gateway.

4.1. Configuración Gateway

Obviamente después de Autentificarnos, vamos a cambiar la contraseña por motivos de seguridad. Lo primero sería establecer el Network Server, que hará posible que los datos recibidos por el Gateway sean visibles por por la web. Dicho de otro modo, lo que conseguimos con esto, es interrelacionar LoRa Server con LoRa App Server.

Damos nombre al Network Server y proporcionamos, su ip y puerto. Dado que está todo en la Raspberry Pi usamos localhost, y el puerto 8000 (que es el puerto por defecto, pero podría cambiarse modificando el fichero loraserver.toml.

Cabe destacar, que es posible conectar LoRa App Server a múltiples instancias de LoRa Server, como podría ser para crear una instancia distinta por cada región.

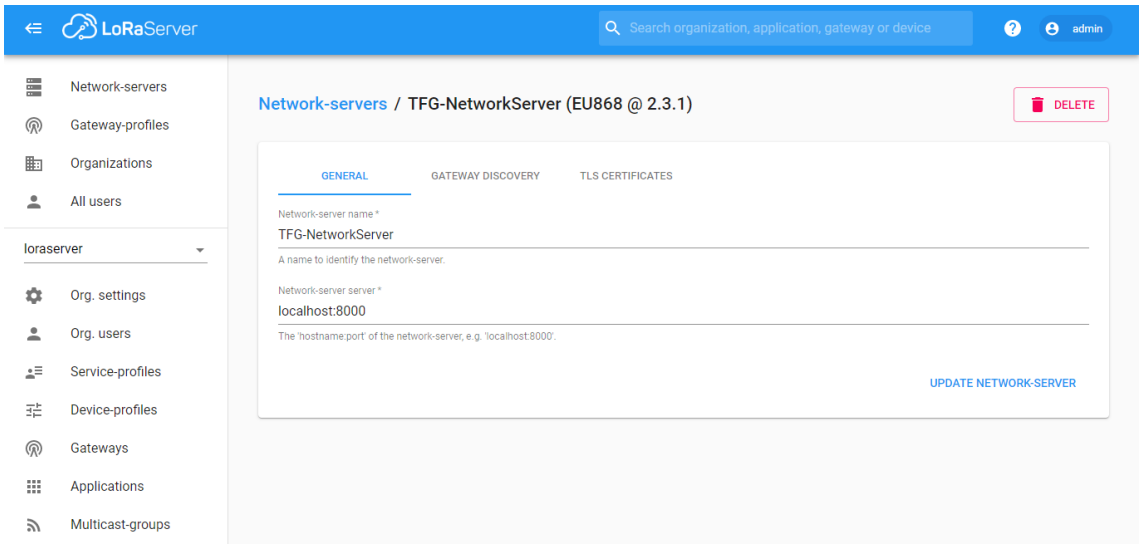


Fig. 19 Pantalla Network-servers

Una vez creada la red, creamos el *Profile* para el Gateway. Esta parte es bastante sencilla, pues únicamente nos limitamos a introducir el nombre, y los canales que queremos que esté a la escucha, pero además podemos crear canales extra, a través del tipo de Modulación (LoRa o FSK), ancho de banda en kHz (125 , 250, 500), la frecuencia en Hz y los Factores de Propagación.

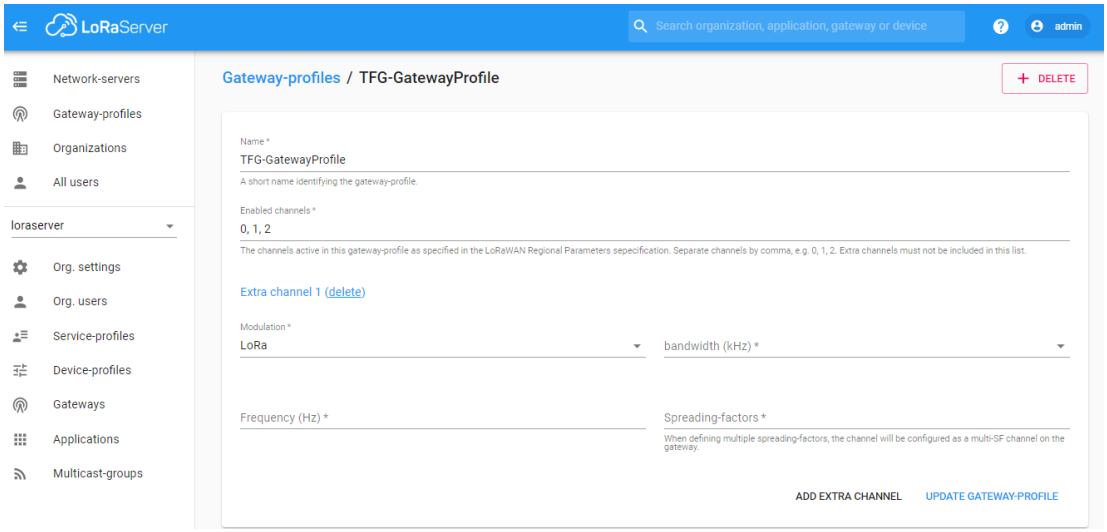


Fig. 20 Pantalla creación Gateway-Profile

Ya que hemos creado su profile, vamos a añadir nuestro *Gateway*, para ello, después de introducir su nombre y descripción, le daremos el perfil que creamos antes, y si queremos, incluiremos sus coordenadas GPS. La Raspberry no está dotada de dicha capacidad, sinó, en el momento que la red recibiese las estadísticas del gateway, se rellenaría tanto la latitud como la longitud, así como la altura.

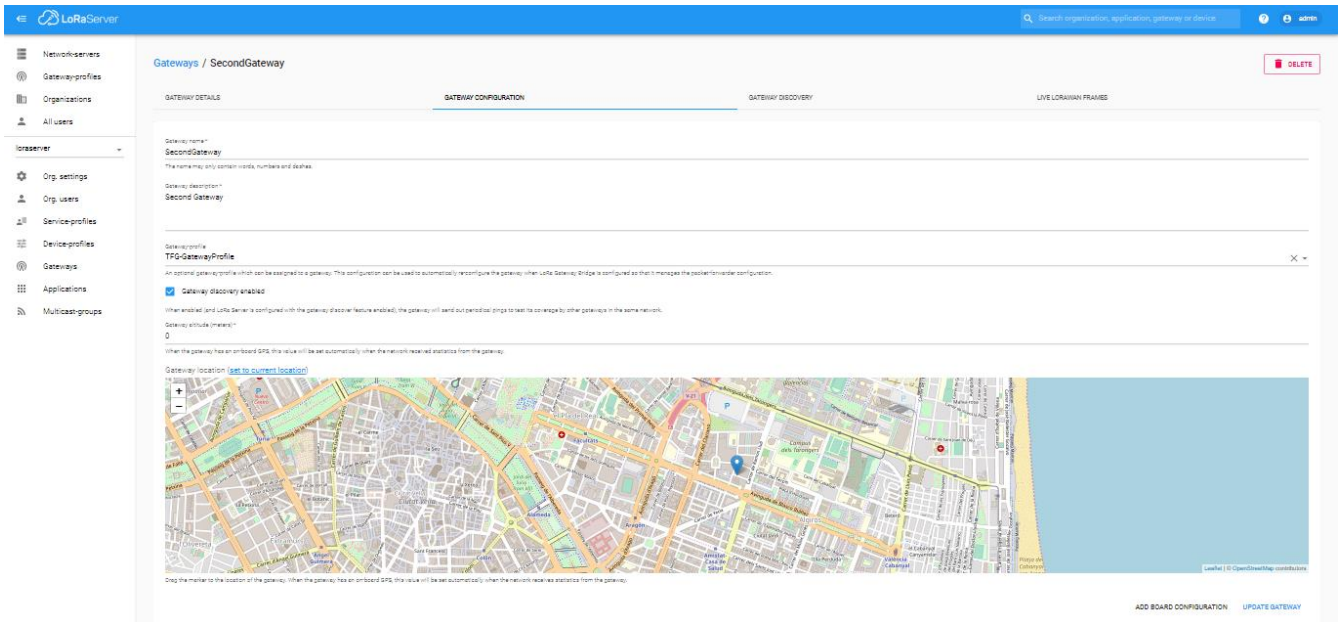


Fig. 21 Detalle Localización Gateway

Seguidamente, formaremos el *Service-Profile*, que define las características que puede utilizar una organización. Además, esto servirá para asociar la Organización con la instancia del Network Server.

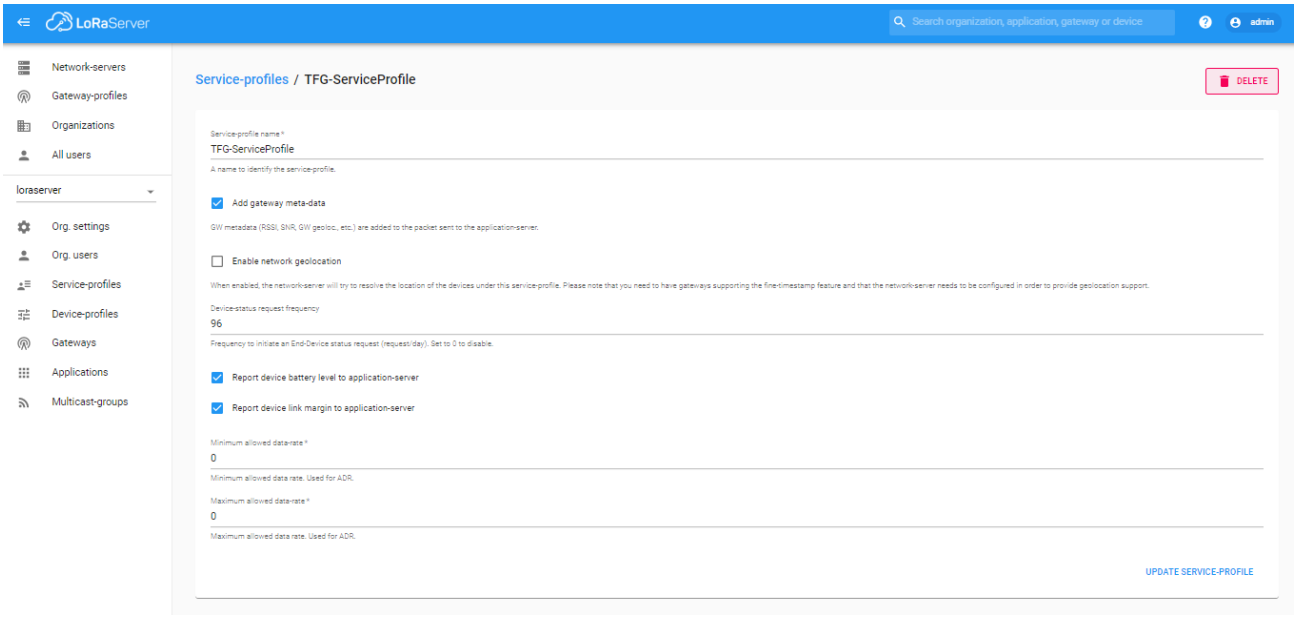


Fig. 22Configuración Service-Profile

Ahora, lo que hemos de definir son las propiedades de nuestro nodo, para ello, nos dirigimos a *Device-Profile*. Con ello, conseguiremos fijar al dispositivo una serie de propiedades, como serían el tipo de activación, (OTAA vs. ABP) la versión de LoRaWAN, la clase…

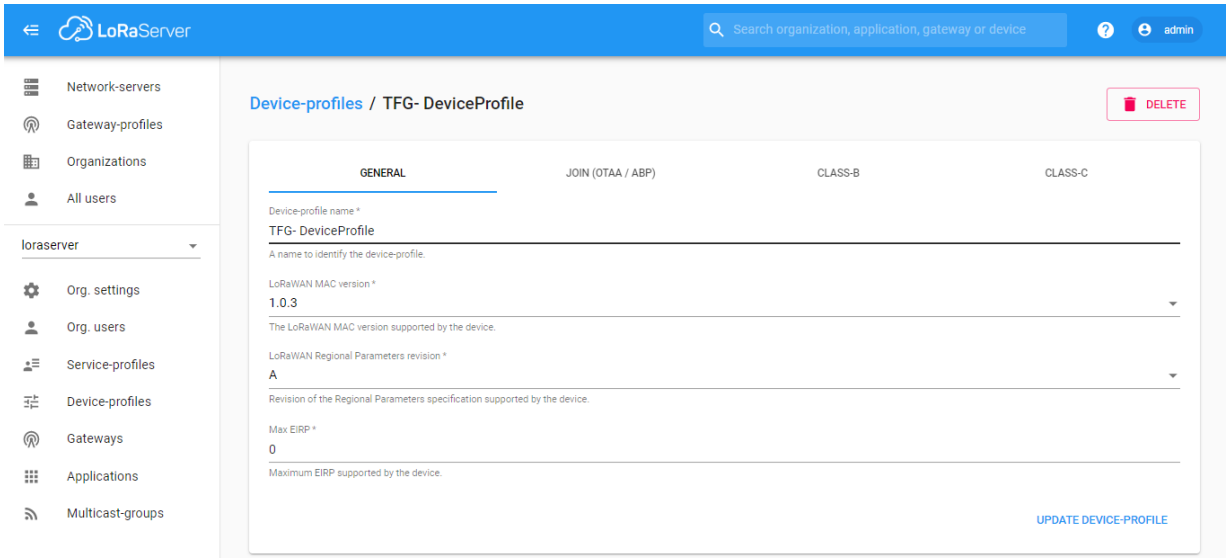


Fig. 23 Vista opciones de Device-Profile

Una vez hecho esto, tendremos enlazados LoRa App Server con nuestra instancia de LoRa Server, nuestra Organización tiene su *Service-Profile* y *Device-Profile*, es el momento de añadir nuestra "aplicación".

En ella, al hacer click en crear, nos pedirá que proporcionemos tanto nombre como descripción de ésta, así como un *Service-Profile* y códec de Payload (Cayenne LPP, JavaScript personalizado o ninguno). Y una vez creada, podemos establecer a su vez, el tipo de Integración (InfluxDB o HTTP)

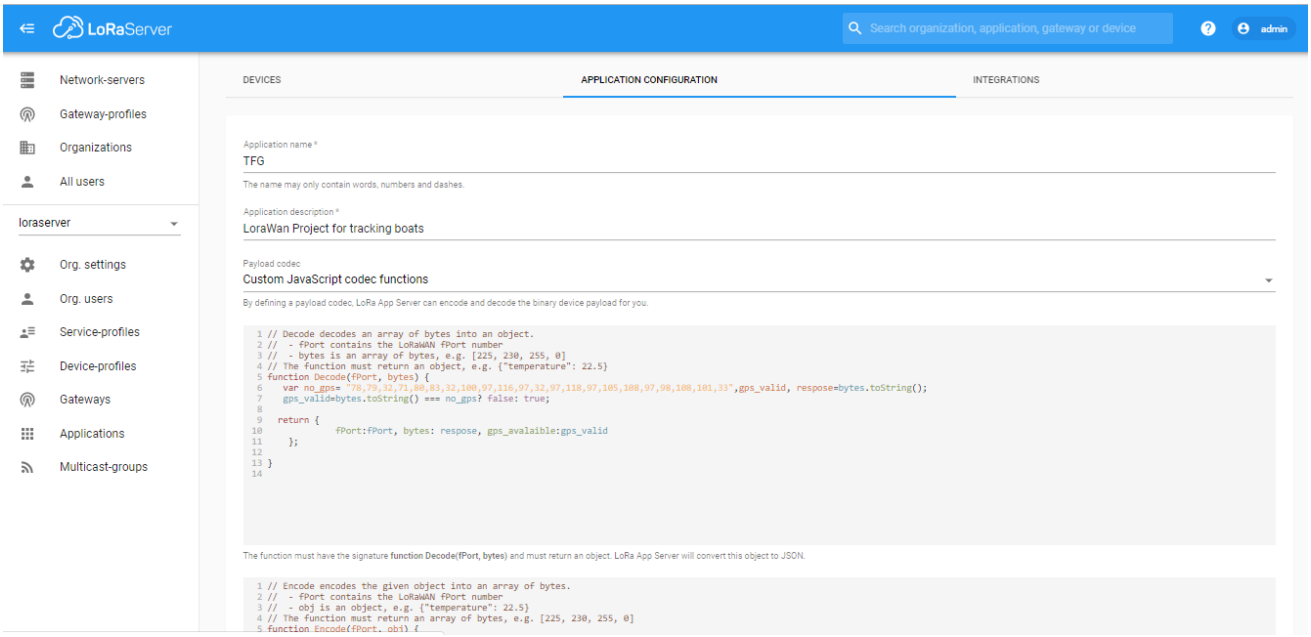


Fig. 24 Detalle configuración de aplicación

Dentro de *Applications*, si añadimos un nuevo dispositivo nos daremos cuenta de que en función del tipo de *join* que le determinemos (**ABP** u **OTAA**) seremos direccionados a su página de configuración correspondiente; en el primer caso, nos pedirá introducir las claves de sesión (**NwkSKey**, **AppSKey** y **AppKey**), mientras que, en el segundo, una única clave (**AppKey**).

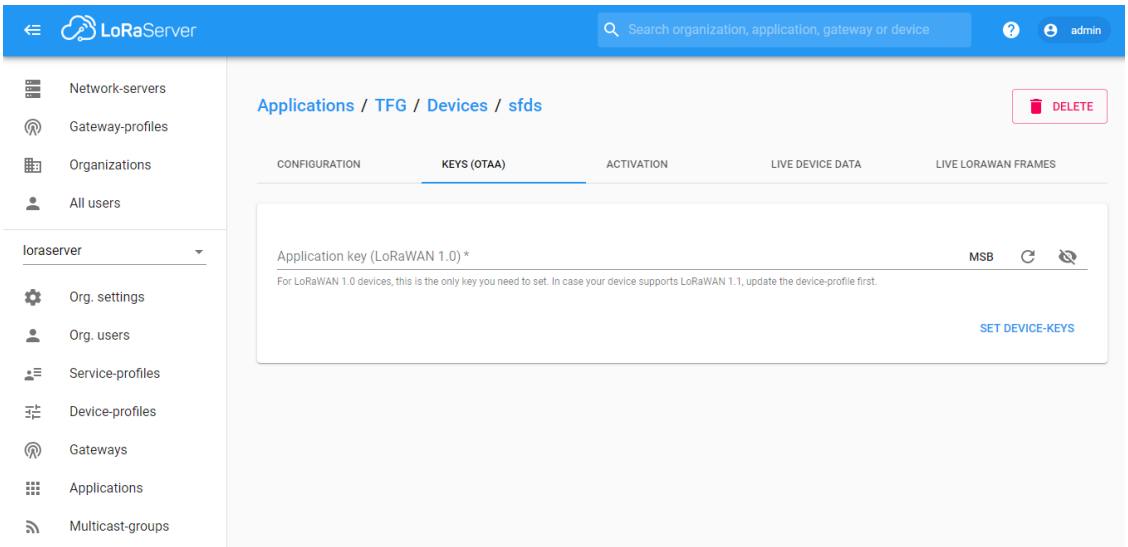


Fig. 25 Vista Opciones dentro de Dispositivos

Si observamos el apartado general del *Gateway* y el *Device*, observaremos que nos ofrecen un *recap*.

En el caso del *Gateway* vemos que nos muestra una gráfico con su actividad en los últimos 30 días. Y una vez dentro, es cuando veríamos en detalle la cantidad de Frames recibidos y transmitidos.

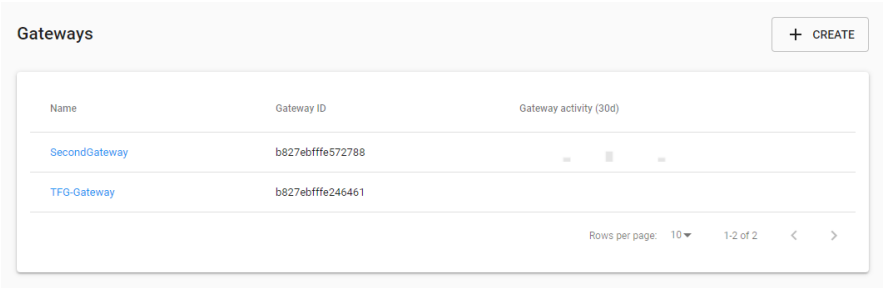


Fig. 26 Lista de Gateways registradas

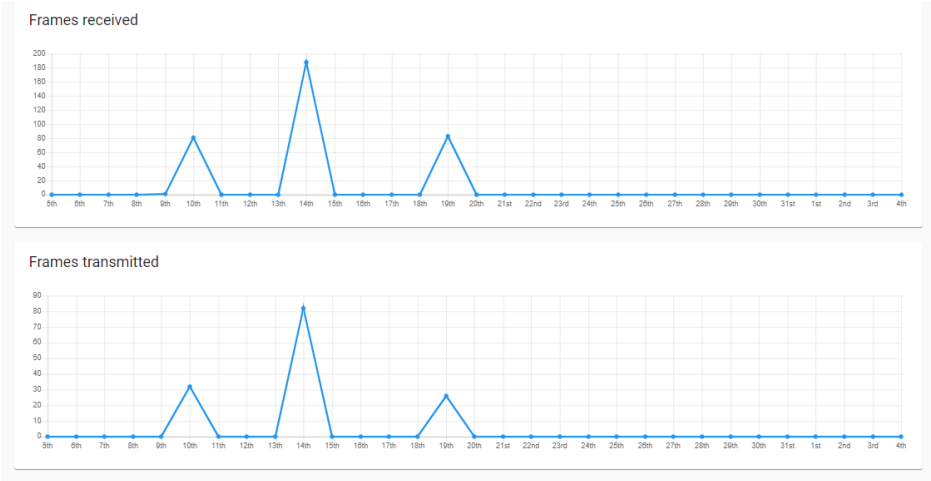


Fig. 27 Detalle Frames recibidos/Transmitidos

Para los Dispositivos, veríamos, los días desde la última vez que fuesen visto en nuestra network, el margen de enlace, (en dB, que cómo ya sabemos, es la diferencia entre la sensibilidad del receptor y la potencia mínima recibida esperada) y el nivel de batería (siendo *n/a* para si no está presente).

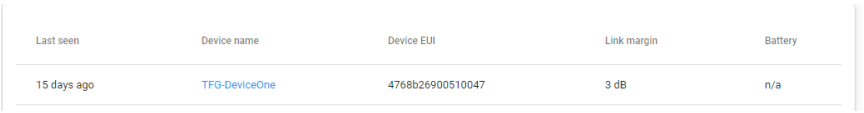


Fig. 28 Detalles Dispositivo

Ambas vistas, Gateway y Device, una vez dentro de éstas, nos muestran una visión *Live* de los Frames LoRaWAN, con la excepción de que el dispositivo, además, nos enseñaría el Data que transmitiría dicho nodo. Estas vistas páginas permiten resumir, pausar y descargar dicha trama en un fichero tipo json.

Finalmente, en el apartado de Integraciones, dentro de Aplicaciones que mencionamos anteriormente, vamos a exponer el uso que hemos dado a la plataforma Sandbox, siendo esta de tipo HTTP.

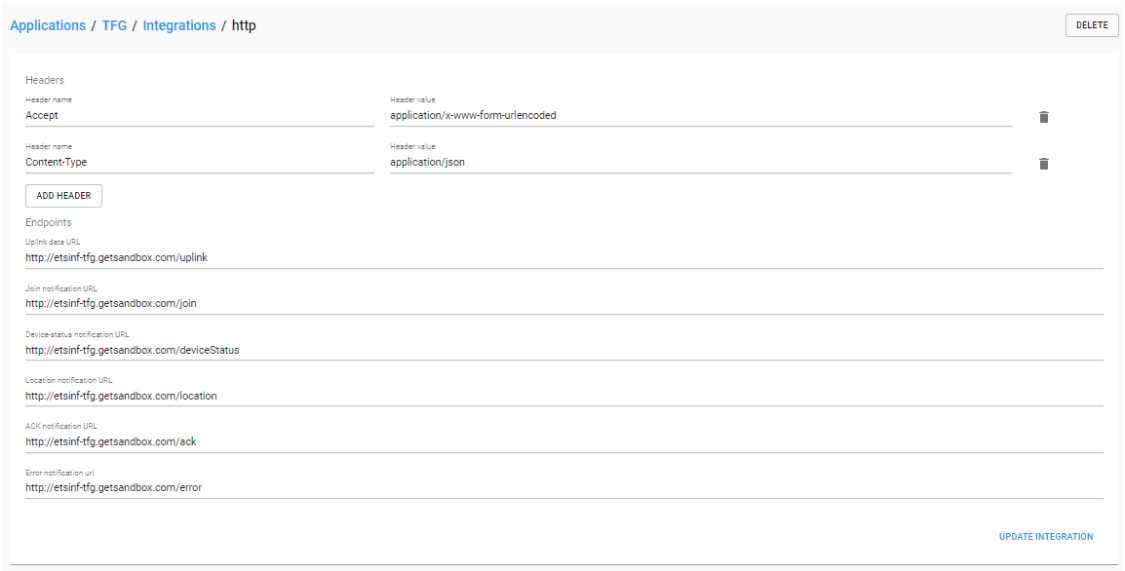


Fig. 29 Apartados integración http

Notamos que está dividido en 2 partes, lo que serían las cabeceras http, y las rutas. Podemos añadir cuantas cabeceras (nombre, valor) queramos. Por el contrario, las rutas, ya están predefinidas (ack, device status, error, join, location, uplink) y únicamente tenemos que asignar la ruta a donde queremos que los datos o notificaciones sean enviados.

4.2. Configuración Arduino IDE

Obviamente, lo primero que hemos de hacer sería instalar el programa. Nosotros lo hemos probado en un Sony Vaio, modelo SVE1512R1EW con Windows 10. Personalmente, optamos por la descargable a través de su sitio web[x] , y no por el store (disponible en win 8.1 y 10) que nos ha dado ciertos problemas a la hora de usarlo.

Después de esto procederemos a añadir Seeeduino LoRa al Arduino, pero para ello, tenemos que descargar un *driver*, pues inicialmente, cuando conectamos el nodo por USB, no es reconocido. Dicho controlador, se puede conseguir a través del repositorio de GitHub de Seeed [xi], que podemos acceder desde la documentación, y es probable que se requiera Deshabilitar el uso obligatorio de controladores firmados para instalarlo. Hecho esto será reconocido por el sistema:

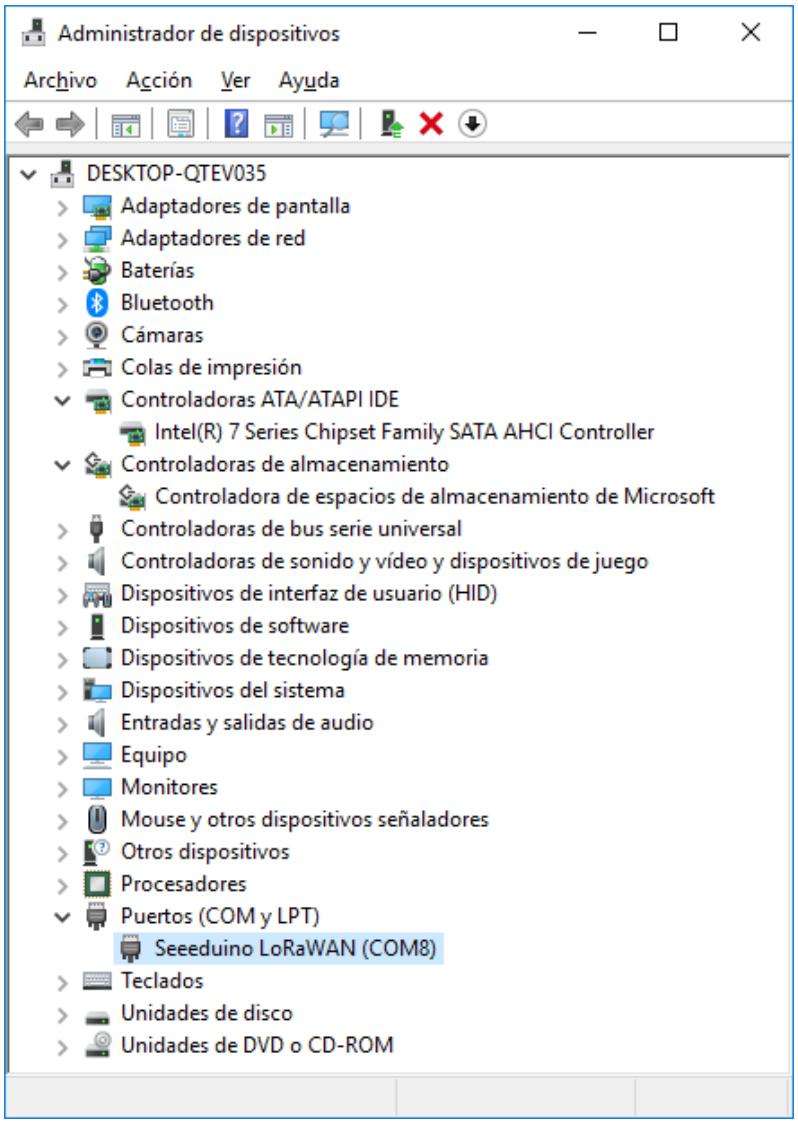


Fig. 30 Detección Seeeduino LoRaWAN en Windows 10

Con el driver ya operativo, sólo falta añadir el conjunto de placas a Arduino IDE, para ello tenemos que efectuar unos pequeños cambios. Bastará con dirigirnos a la ventana de preferencias ("Control" + ",") dentro de Archivo, y una vez allí, añadir la dirección del archivo de *boards* de Seeed al Gestor de URLs adicionales de tarjetas, que es el siguiente:

https://raw.githubusercontent.com/Seeed-tudio/Seeed_Platform/master/package_seeeduino_boards_index.json.

Después de esto, nos dirigimos a Herramientas>Gestor de Tarjetas, y en él, buscamos *Seeeduino LoRaWAN/GPS,* y hacemos "click" en Instalar.

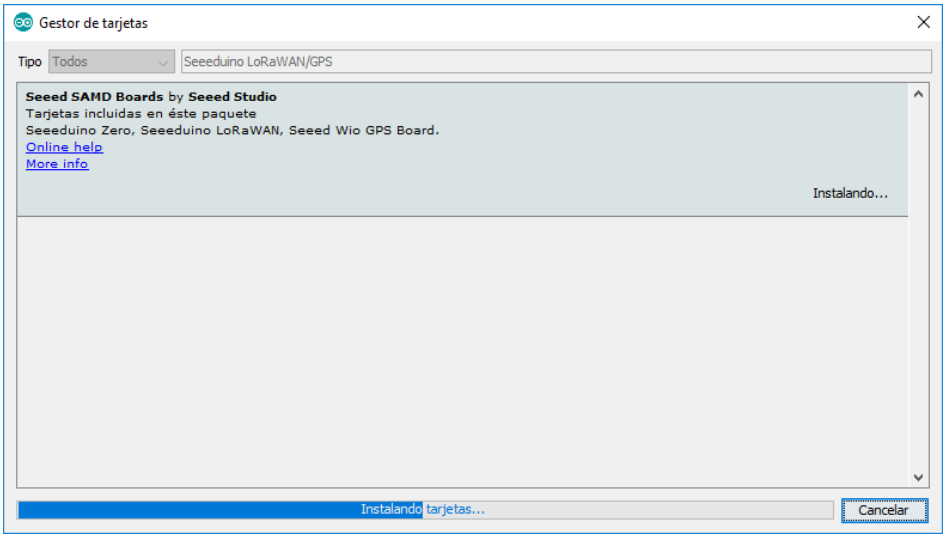


Fig. 31 Instalación Dispositivo en Arduino

A partir de ese momento, estará disponible para su uso la placa, como vemos a continuación:

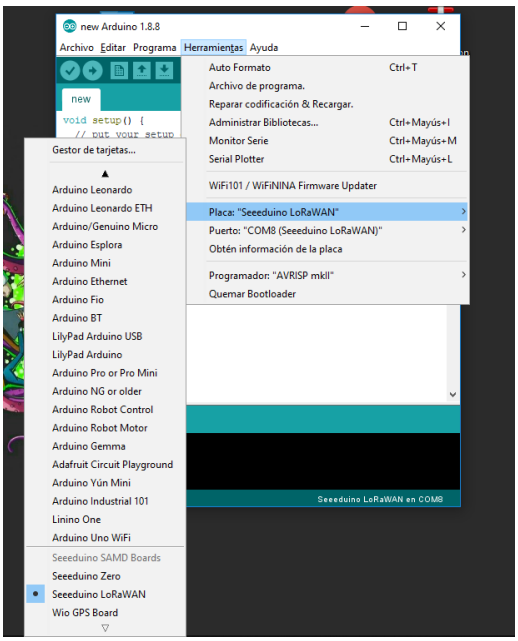


Fig. 32 Ubicación Tarjeta dentro de Arduino

Una vez ya reconocida la *board* por Arduino, vamos a hacer uso de una serie de librerías para poder enviar de forma adecuada los datos recogidos por el chip GPS.

El primero de ellos, TinyGPS[xii], disponible a través de su proyecto GitHub. Con éste, podremos ser capaces de obtener los datos decodificados de la Latitud, Longitud, número de satélites… Y no los datos en bruto, como sería en el caso de NMEA (National Marine Electronics Association).

El otro caso, sería el de CayenneLPP [xiii](Cayenne Low Power Payload), cuyo uso podría ser obviado, pero lo utilizamos, para limitar el tamaño del Búffer. Inicialmente su uso se debía para utilizar la infraestructura de The Things Network.

Finalmente, la última librería sería la de LoRaWAN, se añade por defecto, si partimos de los ejemplos que se instalan al añadir al catálogo de tarjetas *Seeeduino LoRaWAN/GPS,* escogiendo alguno de dicho tipo. Con ella, vamos a poder gestionar todo aquello relativo al uso y manejo de la red LoRaWAN: escoger canal, proporcionar las claves de una red, unirnos a esta…

Seguidamente, y basándonos en el esquema típico de un programa Arduino, en las variables globales, definiríamos las claves de red, la instancia de las librerías que vamos a usar, así como otros datos relativos a la red. Después, la función setup(), inicializaríamos las librerías y procederíamos a activar el dispositivo, que en nuestro caso sería OTAA. Por último, en loop(), recogeríamos los datos del GPS, y los enviaríamos, conformando un paquete.

4.3. Configuración Entorno Aplicación Móvil

Por su simplicidad, además de haber trabajado con anterioridad en dicho entorno, escogimos Atom. Por otro lado, debido a la potencia y simplicidad en diseño, escogemos trabajar con Angular, Ionic y Cordova para diseñar toda la aplicación. Ambos frameworks, Angular e Ionic, se encontraban en fase desarrollo (beta) en el momento de la creación de la *aplicación móvil híbrida,* y para la parte *backend,* utilizaremos TypeScript.

La arquitectura de la *app*, que mejor se podría adaptar a lo que buscamos es SOA, (Arquitectura Orientada a Servicios) pues se trata de un estilo de arquitectura propio de las TI[2], que se apoya en la orientación a servicios. Nuestra Orientación a Servicios se ha debido, a crear cada *endpoint* para el recurso que requeríamos, y finalmente obtener su resultado en el móvil. En otro orden de ideas, y debido también a la concepción propia de Angular e Ionic, que se basa en componentes, contaríamos con la arquitectura MVC (Modelo – Vista – Controlador) aunque no clásico, pues nuestro modelo tiene mucha relación con la vista (los datos se conforman bilateralmente, definiéndose casi como un MV. Dónde nuestro Modelo sería el *controller*, que incluiría llamadas al api, detección de eventos, así como el resto de código que determinaría lo que veríamos en la Vista.

Como puntualización, cabe destacar que hemos hecho uso de un plugin de Cordova para poder mostrar el mapa de Google [xiv].

4.3.1 Aplicación Móvil

Se ha intentado diseñar una app lo más sencilla y amigable posible. Tal vez se podría haber simplificado mucho más su desarrollo, pero optamos por diferenciarla en secciones.

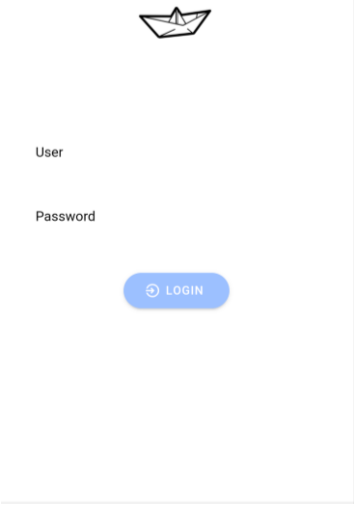


Fig. 33 Pantalla Login

4.3.1.1 Pantalla de Inicio - Login

Esta vista, nos pide la introducción de los datos (nombre de usuario y contraseña), con tal de autentificarnos. Se trata de los mismos datos que usaríamos para acceder al entorno web. Una vez realizado, la clave JWT, será enviada/actualizada, con todos los datos pertinentes de usuario en el api del getsandbox.

[2] Tecnologías de la Información

4.3.1.2 Acceso Directo

Se trata de un menú desplegable, accesible desde la mayoría de las vistas, en las que vamos a poder acceder a las diferentes pantallas de nuestra aplicación, así como la consabida opción de cerrar sesión o *log-out*.

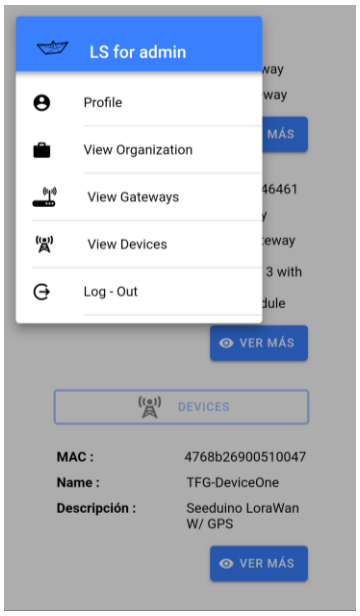


Fig. 34 Detalle Menú

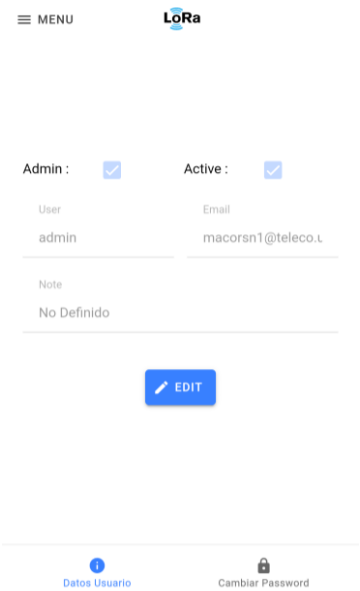
Fig. 35 Vista General

4.3.1.3 Visión General

En esta vista, veremos de forma resumida los datos relativos a nuestros dispositivos disponibles dentro de la red, ya sean Gateways o Nodos, pudiendo acceder en profundidad a cada uno de ellos.

4.3.1.4 Perfil

Nos muestra, los datos pertinentes a un usuario (si es administrador, si está activo, nombre de usuario, mail, y alguna nota/recordatorio), permitiendo además editar éstos si se quiere.



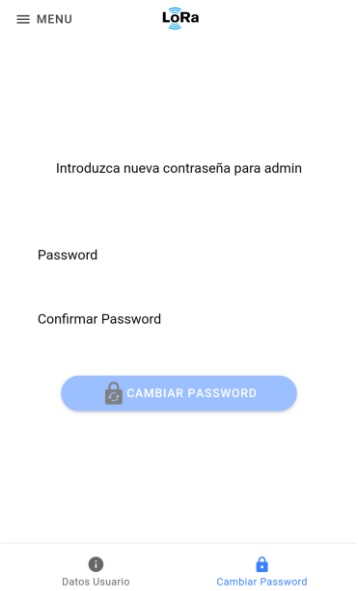


Fig. 36 Perfil

4.3.1.4.1 Password

Esta ventana es un añadido de la anterior, y su finalidad es actualizar/cambiar la contraseña del usuario actual.

Fig. 37 Password

4.3.1.5 Organización

En ella, vemos la visión general de la Organización, además de la posibilidad de ver un listado de los Usuarios, así como crear uno nuevo. Observamos si dispone de una Puerta de Enlace, cuando fue creada y actualizada, y el nombre dado.

Fig. 38 Organización

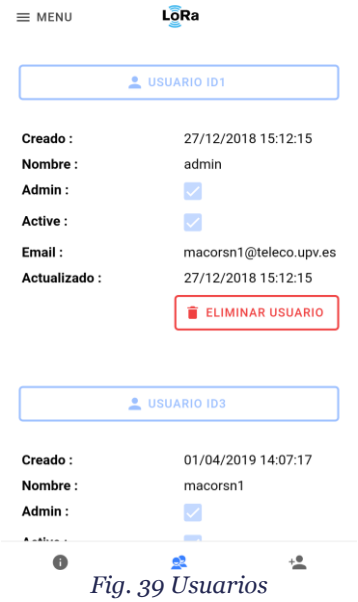


Fig. 39 Usuarios

4.3.1.5.1 Usuarios

En esta pantalla, veríamos todos los usuarios registrados en nuestra organización, con sus respectivos datos de creación. También es posible eliminar un usuario en concreto.

4.3.1.5.2 Creación Usuario

A través de un formulario, podemos añadir un usuario a la organización. Es recomendable anotar la contraseña, pues esta se genera de manera aleatoria. Recordemos, que una vez autentificado, podremos cambiarla si se desea.

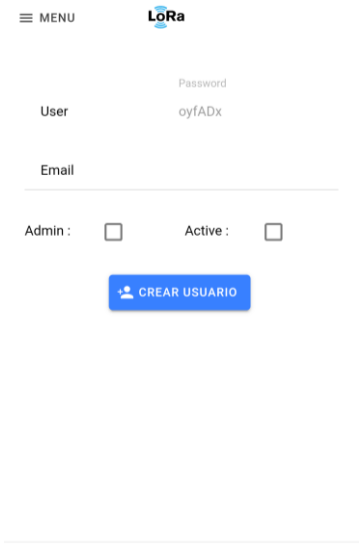


Fig. 40 Creación Usuario

Fig. 41 Gateway

4.3.1.6 Gateway

Esta vista, unifica todos los dispositivos que son del tipo Puerta de enlace, mostrando estadísticas, localización, y otros datos de interés (creación, actualización, primera y última vez vista). Si se accede desde la visión general, mostrará únicamente el Gateway seleccionado, sino mostraría todos los disponibles.

4.3.1.7 Nodos

Al igual que en la pantalla anterior, dependiendo del origen, mostrará todos los disponibles, o el seleccionado en concreto (recordemos que la vista general de los dispositivos o nodos es accesible desde el acceso directo). En los nodos, es visible los datos de Seguridad y Batería, además de los datos más generales, algunos ya vistos en la Visión General.

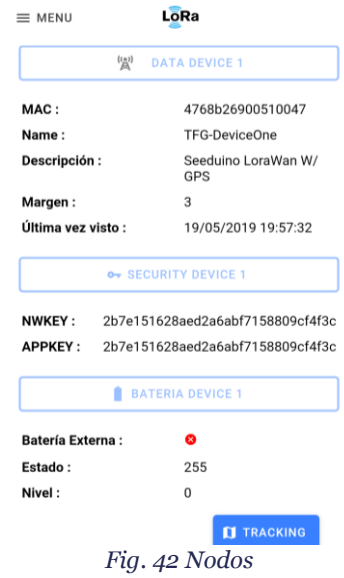


Fig. 42 Nodos

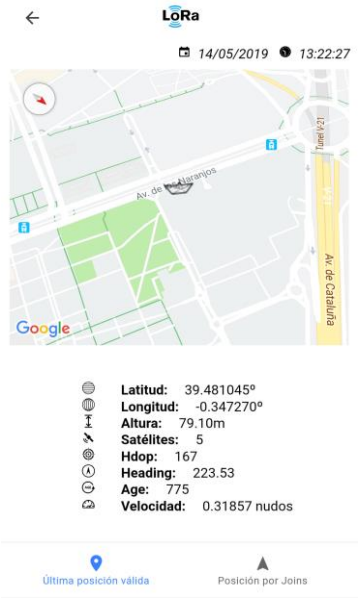


Fig. 43 Última Posición

4.3.1.7.1 Última posición

Accesible a través del nodo. En ésta, vemos la última posición reconocida en la red, con los datos relativos a él, con su disposición en el mapa representado por el logo de la aplicación.

4.3.1.7.2 Histórico de Posiciones

Podemos ver un histórico de los *joins* realizados por el dispositivo, y seleccionando uno de éstos, nos dibujaría una traza con las posiciones registradas. Presenta los mismos datos que la última posición, con la salvedad que, en ésta, vemos la dirección del dispositivo (que la otorga la red una vez unido) y hasta cuando es esa dirección (generalmente hasta que se produce otra unión del dispositivo con la red). Podemos desplazarnos entre los *uplinks* del *join* seleccionado, gracias a las flechas disponibles a ambos lados de los datos presentados.

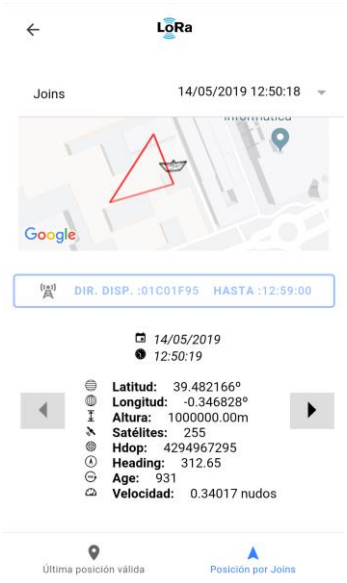


Fig. 44 Histórico de Posiciones

Capítulo 5: Prueba de Campo y Problemas conocidos

Inicialmente el Kit venía con una tarjeta SD de 8 GB, pero, dado que, con las múltiples pruebas, y que cambiamos la concepción con la que inicialmente se presentaba el kit, cambiamos a 32 GB, liberando el espacio, y porque no, mejorando un poco el rendimiento del sistema. Por otro lado, en los primeros días de la implantación del sistema, y de dejar este expuesto a la red, sufrimos un ataque por medios rusos, que dieron al traste el trabajo realizado, obligándonos a partir de cero, pues corrompieron casi la totalidad del sistema.

Las pruebas de campo han sido de manera experimental, es decir, no hemos desarrollado en la totalidad una infraestructura que sirviese para geoposicionar una pequeña flota de naves de recreo, como pudiese ser las de una escuela náutica. Y es en estas pruebas empíricas dónde vemos que los resultados, simplemente no son como esperábamos en un principio. De hecho, no algo tan sonado como el caso práctico llevado a cabo por The Things Network, que consiguieron transmitir un paquete a más de 700 km de distancia[xv], pero si esperábamos algo más. Puede deberse a que inicialmente, nuestro pack, es simplemente un kit de iniciación a esta tecnología, es decir, realizar pequeñas pruebas, como podría ser montar un servidor y controlar, eso sí, en distancias cortas, un objeto a partir del dispositivo.

Hemos hecho sendas pruebas, una de ellas, bastante cercana a la realidad, puesto que nos desplazamos al puerto, colocando nuestro gateway lo más cerca del mar, y llevando el dispositivo en una embarcación. Para recoger y tener acceso a los datos, conectamos la puerta de enlace un *MiFi*, (router portátil) y una batería portátil de gran capacidad. Pudiera ser también que debería estar en una zona bastante elevada y al descubierto, como pudiese ser un faro, y no, como se realizó, escondido en el espigón.

Otro de los casos clave fue dentro de la ciudad de Valencia, en la zona de Blasco Ibáñez, mientras uno se desplazaba con el nodo, otra persona se quedaba con la puerta de enlace. En esta ocasión, el principal *hándicap* residía en la existencia de obstáculos arquitectónicos, a diferencia del mar abierto.

En estas pruebas, se obtuvo que el alcance máximo se situaba cercano a los 300 metros, esto no venía ligado a condiciones climáticas adversas que empobreciesen la calidad del enlace, pues el día era soleado. Lo buscado, en nuestro caso sería que, como mínimo, esa distancia fuera de 1200 metros más. El otro inconveniente reside en una de las propiedades del LoRaWAN, el *payload*, su *uplink,* y el *duty cycle*. El problema reside, en el tiempo que existe de envío entre paquete y paquete ronda la media de 30 segundos, así que el hecho de ser en tiempo real no es tan preciso como se quisiera, que, por otro lado, también hay que tener en cuenta, que cuan más se aleja de la base más va a tardar en transmitir el mensaje, además del hecho que cuanto más frecuente sea, más alto coste energético va a tener. El tema del alcance, como dijimos inicialmente, debería poderse solucionar cambiando la antena del gateway, pues añadimos una antena uFL directamente en el dispositivo, y no apreciamos cambios significativos en el alcance de

la señal. Por otro lado, estudiando chips similares al empleado, que dicen alcanzar los 3 km de distancia, eso sí con una antena de 2 m de alto, se comenta que tanto la humedad como la temperatura, así como la proximidad al mar (debido a su capacidad para absorber señales inalámbricas) empobrecerían y reducirían el rango de alcance.

Como mejoras adicionales, pensamos que, si queremos realizar toda la infraestructura para el seguimiento de una flota, requeriríamos quizá de unos 5 gateways, formando un plano, que sería la superficie por donde se moverían las embarcaciones, con uno el centro por posibles problemas de alcance. También sería, obviamente, al menos un dispositivo por embarcación. Además, gracias a los pinouts que disponen las placas de los nodos, sería posible de dotarlas de más dispositivos Arduino, con los que recoger, por ejemplo, la fuerza del viento, o la humidad relativa.

Finalmente, y debido a que ambos aparatos van a estar expuestos a condiciones climáticas complicadas, sería interesante el hecho de fabricar para ambas cajas estancas, en la que principalmente a la intemperie estarían las antenas, pero con tomas para puertos USB ocultas y aisladas, con tal de poder recargar los nodos.

Conclusiones

Después de la concepción de este proyecto, nos ha sido posible apreciar el alcance y potencial que tiene esta tecnología, que es prácticamente una desconocida por mucha gente. Si bien el coste del kit de iniciación a esta tecnología es contenido, si queremos realizar algo de gran calibre, lo más probable es que requiera un gran desembolso, y que quizás, por este motivo, se quiera optar por redes anteriores y ya creadas.

Gracias al marco teórico de nuestro proyecto hemos podido comprobar la complejidad y funcionamiento de una red emergente, que además gracias a las tecnologías y sistemas con las que opera, (Linux, Arduino, Bases de Datos…) nos ha ayudado tanto a profundizar como a una toma de contacto para redescubrir porque nos gusta la Informática. No sin haber tenido que pasar por un arduo período de adaptación, estudio y aprendizaje de la tecnología LoRaWAN, pero que como una pequeña introducción al mundo IoT, nos ha resultado muy gratificante cuando observábamos resultados, pese a no ser tal como los esperábamos.

Respecto a lo que sería la temática del proyecto, un sistema de tracking para embarcaciones de vela ligera, como comentamos en el capítulo anterior, la tecnología LoRaWAN tal vez no sea la más idónea, pues a nuestro parecer estaría más enfocado para la domótica o la inmótica (como pudiera ser sistemas del cuidado de la salud, sistemas de eficiencia energética, etc.).

Con este proyecto, además, hemos aprendido la importancia de la planificación y organización de tareas, pues, con el paso del tiempo, nos hemos dado cuenta de que, en el mundo real, la planificación y desarrollo de un proyecto de software conlleva también ciertos contratiempos que pueden llegar a ser grandes errores si no se ha optimizado y tenido en cuenta en la gestión de tareas que pueden surgir imprevistos. Y en mi opinión, en esta idea o proyecto, se ha querido abarcar y hacer demasiadas tareas que al final a penalizado en la concepción de este, llevando en muchos casos a ir deprisa y corriendo, e incluso, a contracorriente.

Como epílogo para nuestro proyecto podemos concluir que, pese a que la tecnología LoRaWAN ha sufrido un auge en los últimos años, no acaba de dar un repunte que la sitúe en las primeras tecnologías como método de comunicación inalámbrica, y que esto se puede deber a que posé limitaciones, (como pudiera ser complejidad o coste) pese a que organizaciones como The Things Network o la LoRa Alliance intenten cambiarlo.

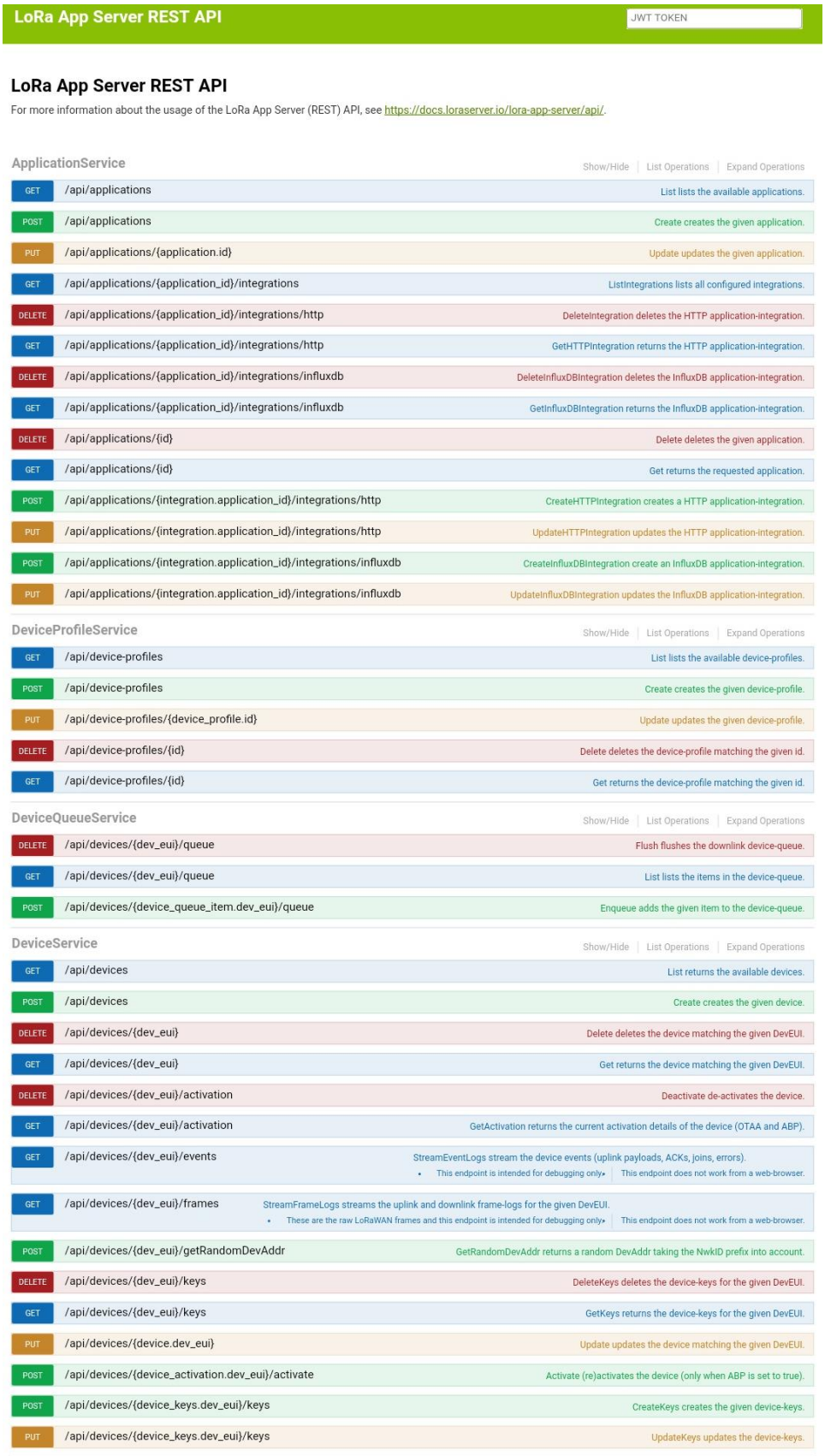
Relación del trabajo desarrollado con los estudios cursados

En general todas las asignaturas han aportado su granito de arena en la concepción de este proyecto, pues hemos tratado desde Bases de Datos hasta el manejo de un terminal Linux, pasando por la distribución de tareas, manejo de GitHub, pero quizás las más importantes serían Ingeniería de Software y Interfaz Persona Computador, pues mientras que una nos enseña como vertebrar una aplicación, la otra nos permitiría como mostrarlo de la forma más simple y amigable posible.

En lo que respecta a las competencias transversales que han sido de gran ayuda para la realización del proyecto serían:

- Aprendizaje permanente
- Aplicación y pensamiento práctico
- Innovación, creatividad y emprendimiento

Anexo



LoRa App Server REST API

JWT TOKEN

LoRa App Server REST API

For more information about the usage of the LoRa App Server (REST) API, see https://docs.loraserver.io/lora-app-server/api/.

ApplicationService

Show/Hide | List Operations | Expand Operations

GET	/api/applications	List lists the available applications.
POST	/api/applications	Create creates the given application.
PUT	/api/applications/{application.id}	Update updates the given application.
GET	/api/applications/{application_id}/integrations	ListIntegrations lists all configured integrations.
DELETE	/api/applications/{application_id}/integrations/http	DeleteIntegration deletes the HTTP application-integration.
GET	/api/applications/{application_id}/integrations/http	GetHTTPIntegration returns the HTTP application-integration.
DELETE	/api/applications/{application_id}/integrations/influxdb	DeleteInfluxDBIntegration deletes the InfluxDB application-integration.
GET	/api/applications/{application_id}/integrations/influxdb	GetInfluxDBIntegration returns the InfluxDB application-integration.
DELETE	/api/applications/{id}	Delete deletes the given application.
GET	/api/applications/{id}	Get returns the requested application.
POST	/api/applications/{integration.application_id}/integrations/http	CreateHTTPIntegration creates a HTTP application-integration.
PUT	/api/applications/{integration.application_id}/integrations/http	UpdateHTTPIntegration updates the HTTP application-integration.
POST	/api/applications/{integration.application_id}/integrations/influxdb	CreateInfluxDBIntegration create an InfluxDB application-integration.
PUT	/api/applications/{integration.application_id}/integrations/influxdb	UpdateInfluxDBIntegration updates the InfluxDB application-integration.

DeviceProfileService

Show/Hide | List Operations | Expand Operations

GET	/api/device-profiles	List lists the available device-profiles.
POST	/api/device-profiles	Create creates the given device-profile.
PUT	/api/device-profiles/{device_profile.id}	Update updates the given device-profile.
DELETE	/api/device-profiles/{id}	Delete deletes the device-profile matching the given id.
GET	/api/device-profiles/{id}	Get returns the device-profile matching the given id.

DeviceQueueService

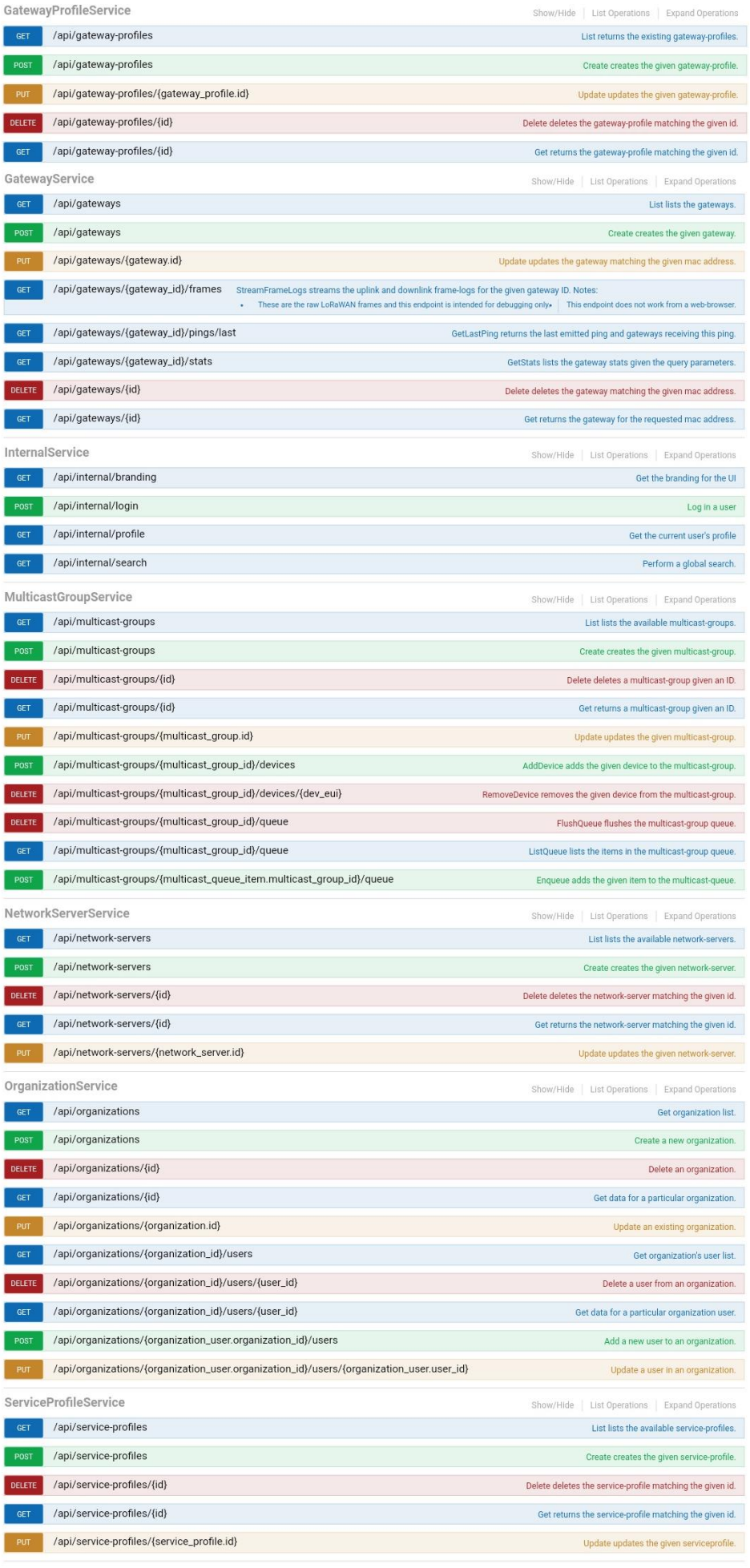
Show/Hide | List Operations | Expand Operations

DELETE	/api/devices/{dev_eui}/queue	Flush flushes the downlink device-queue.
GET	/api/devices/{dev_eui}/queue	List lists the items in the device-queue.
POST	/api/devices/{device_queue_item.dev_eui}/queue	Enqueue adds the given item to the device-queue.

DeviceService

Show/Hide | List Operations | Expand Operations

GET	/api/devices	List returns the available devices.
POST	/api/devices	Create creates the given device.
DELETE	/api/devices/{dev_eui}	Delete deletes the device matching the given DevEUI.
GET	/api/devices/{dev_eui}	Get returns the device matching the given DevEUI.
DELETE	/api/devices/{dev_eui}/activation	Deactivate de-activates the device.
GET	/api/devices/{dev_eui}/activation	GetActivation returns the current activation details of the device (OTAA and ABP).
GET	/api/devices/{dev_eui}/events	StreamEventLogs stream the device events (uplink payloads, ACKs, joins, errors). • This endpoint is intended for debugging only. This endpoint does not work from a web-browser.
GET	/api/devices/{dev_eui}/frames	StreamFrameLogs streams the uplink and downlink frame-logs for the given DevEUI. • These are the raw LoRaWAN frames and this endpoint is intended for debugging only. This endpoint does not work from a web-browser.
POST	/api/devices/{dev_eui}/getRandomDevAddr	GetRandomDevAddr returns a random DevAddr taking the NwkID prefix into account.
DELETE	/api/devices/{dev_eui}/keys	DeleteKeys deletes the device-keys for the given DevEUI.
GET	/api/devices/{dev_eui}/keys	GetKeys returns the device-keys for the given DevEUI.
PUT	/api/devices/{device.dev_eui}	Update updates the device matching the given DevEUI.
POST	/api/devices/{device_activation.dev_eui}/activate	Activate (re)activates the device (only when ABP is set to true).
POST	/api/devices/{device_keys.dev_eui}/keys	CreateKeys creates the given device-keys.
PUT	/api/devices/{device_keys.dev_eui}/keys	UpdateKeys updates the device-keys.



GatewayProfileService — Show/Hide | List Operations | Expand Operations

GET	/api/gateway-profiles	List returns the existing gateway-profiles.
POST	/api/gateway-profiles	Create creates the given gateway-profile.
PUT	/api/gateway-profiles/{gateway_profile.id}	Update updates the given gateway-profile.
DELETE	/api/gateway-profiles/{id}	Delete deletes the gateway-profile matching the given id.
GET	/api/gateway-profiles/{id}	Get returns the gateway-profile matching the given id.

GatewayService — Show/Hide | List Operations | Expand Operations

GET	/api/gateways	List lists the gateways.
POST	/api/gateways	Create creates the given gateway.
PUT	/api/gateways/{gateway.id}	Update updates the gateway matching the given mac address.
GET	/api/gateways/{gateway_id}/frames	StreamFrameLogs streams the uplink and downlink frame-logs for the given gateway ID. Notes: • These are the raw LoRaWAN frames and this endpoint is intended for debugging only• This endpoint does not work from a web-browser.
GET	/api/gateways/{gateway_id}/pings/last	GetLastPing returns the last emitted ping and gateways receiving this ping.
GET	/api/gateways/{gateway_id}/stats	GetStats lists the gateway stats given the query parameters.
DELETE	/api/gateways/{id}	Delete deletes the gateway matching the given mac address.
GET	/api/gateways/{id}	Get returns the gateway for the requested mac address.

InternalService — Show/Hide | List Operations | Expand Operations

GET	/api/internal/branding	Get the branding for the UI
POST	/api/internal/login	Log in a user
GET	/api/internal/profile	Get the current user's profile
GET	/api/internal/search	Perform a global search.

MulticastGroupService — Show/Hide | List Operations | Expand Operations

GET	/api/multicast-groups	List lists the available multicast-groups.
POST	/api/multicast-groups	Create creates the given multicast-group.
DELETE	/api/multicast-groups/{id}	Delete deletes a multicast-group given an ID.
GET	/api/multicast-groups/{id}	Get returns a multicast-group given an ID.
PUT	/api/multicast-groups/{multicast_group.id}	Update updates the given multicast-group.
POST	/api/multicast-groups/{multicast_group_id}/devices	AddDevice adds the given device to the multicast-group.
DELETE	/api/multicast-groups/{multicast_group_id}/devices/{dev_eui}	RemoveDevice removes the given device from the multicast-group.
DELETE	/api/multicast-groups/{multicast_group_id}/queue	FlushQueue flushes the multicast-group queue.
GET	/api/multicast-groups/{multicast_group_id}/queue	ListQueue lists the items in the multicast-group queue.
POST	/api/multicast-groups/{multicast_queue_item.multicast_group_id}/queue	Enqueue adds the given item to the multicast-queue.

NetworkServerService — Show/Hide | List Operations | Expand Operations

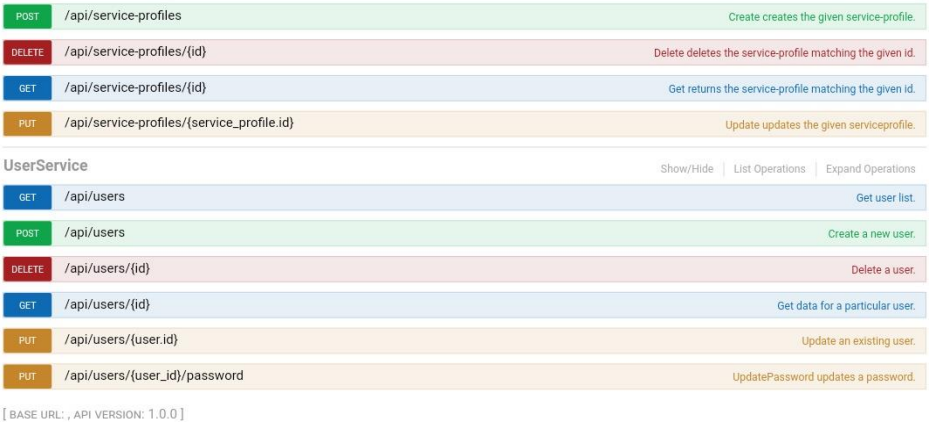
GET	/api/network-servers	List lists the available network-servers.
POST	/api/network-servers	Create creates the given network-server.
DELETE	/api/network-servers/{id}	Delete deletes the network-server matching the given id.
GET	/api/network-servers/{id}	Get returns the network-server matching the given id.
PUT	/api/network-servers/{network_server.id}	Update updates the given network-server.

OrganizationService — Show/Hide | List Operations | Expand Operations

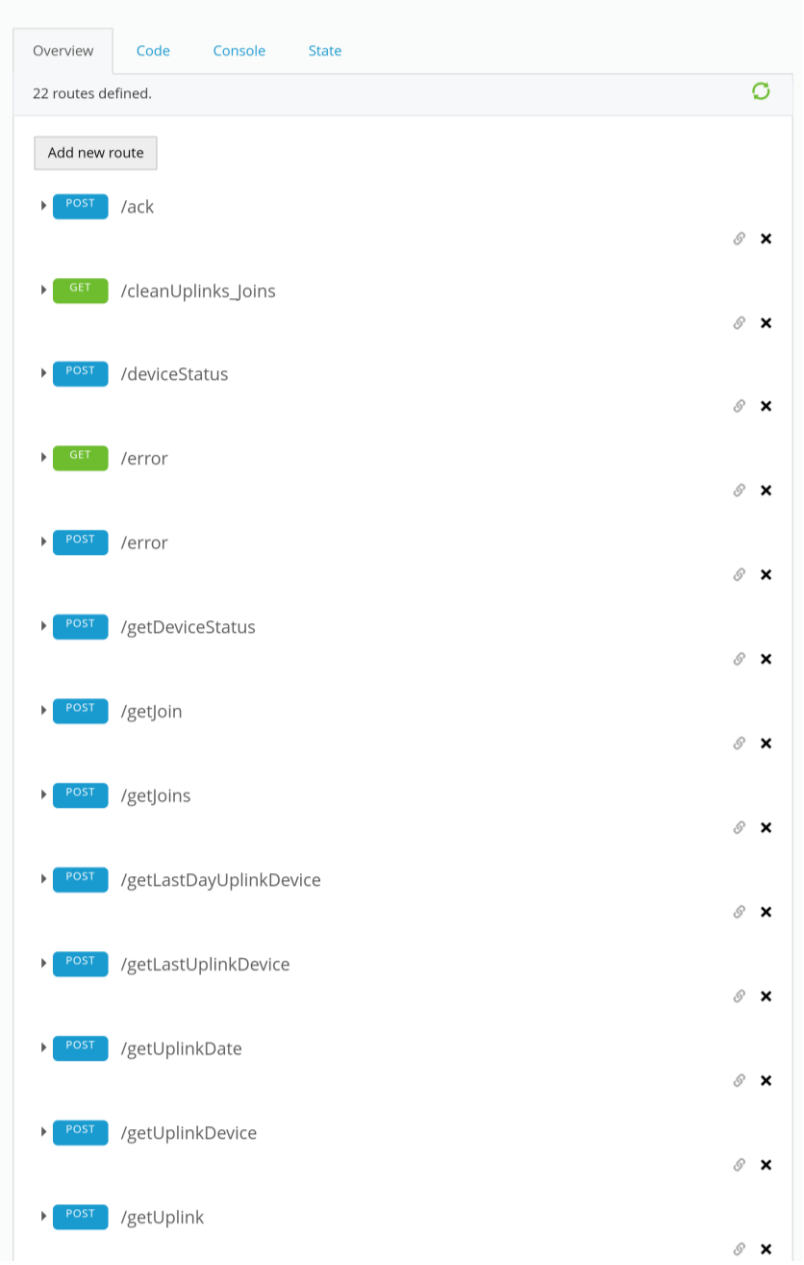
GET	/api/organizations	Get organization list.
POST	/api/organizations	Create a new organization.
DELETE	/api/organizations/{id}	Delete an organization.
GET	/api/organizations/{id}	Get data for a particular organization.
PUT	/api/organizations/{organization.id}	Update an existing organization.
GET	/api/organizations/{organization_id}/users	Get organization's user list.
DELETE	/api/organizations/{organization_id}/users/{user_id}	Delete a user from an organization.
GET	/api/organizations/{organization_id}/users/{user_id}	Get data for a particular organization user.
POST	/api/organizations/{organization_user.organization_id}/users	Add a new user to an organization.
PUT	/api/organizations/{organization_user.organization_id}/users/{organization_user.user_id}	Update a user in an organization.

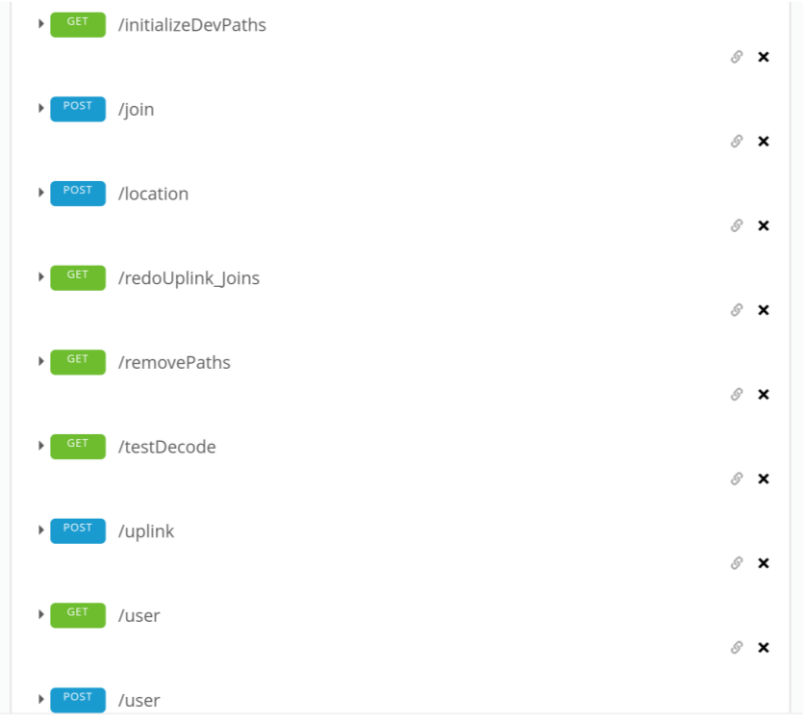
ServiceProfileService — Show/Hide | List Operations | Expand Operations

GET	/api/service-profiles	List lists the available service-profiles.
POST	/api/service-profiles	Create creates the given service-profile.
DELETE	/api/service-profiles/{id}	Delete deletes the service-profile matching the given id.
GET	/api/service-profiles/{id}	Get returns the service-profile matching the given id.
PUT	/api/service-profiles/{service_profile.id}	Update updates the given serviceprofile.



Apéndice 1 Detalle API LoRaServer





Apéndice 2 Detalle API creada getSandBox

```
import { Component, Injectable, Inject }    from '@angular/core';
import { HTTP } from '@ionic-native/http/ngx';

@Injectable()
export class HttpService {

  serverAPI = "http://192.168.8.100:8080/api/";
  serverSandBox="http://etsinf-tfg.getsandbox.com/";


constructor(
              private http: HTTP
              )
    {
      this.http.setDataSerializer('json');
    }


    method(method,api,sandbox, params,jwt){
      let response;

      let headers;

      if(jwt){

           headers ={
              'Content-Type': 'application/json' ,
              'Accept'      : 'application/json',
              'Grpc-Metadata-Authorization': 'Bearer '+ jwt
            }

          }else if(jwt == null){

            headers ={
               'Content-Type': 'application/json' ,
               'Accept'      : 'application/json'
             }

      }

      if(!sandbox){

        switch(method){

          case "delete":  response = this.http
                                     .delete(this.serverAPI+api,
params, headers);
                          break;

          case "get" :  response = this.http
                                       .get(this.serverAPI+api,{},
headers);

                         break;

          case "post":  response = this.http
                                     .post(this.serverAPI+api, params,
headers);
                          break;

         case "put":  response = this.http
```

```
                                        .put(this.serverAPI+api, params,
headers);
                            break;
          }


       }else

            switch(method){

              case "delete" :  response = this.http
.delete(this.serverSandBox+api,{}, headers);
                              break;

              case "get" :  response = this.http
.get(this.serverSandBox+api,{}, headers);
                              break;

              case "post":  response = this.http
                                        .post(this.serverSandBox+api,
params, headers);
                               break;

             case "put":  response = this.http
                                        .put(this.serverSandBox+api,
params, headers);
                              break;
            }


       return response;
     }


}
```

Apéndice 3 Servicio Peticiones HTTP

```
#include <TinyGPS.h>
#include <LoRaWan.h>
#include <CayenneLPP.h>

TinyGPS gps;
CayenneLPP lpp(26);
char memBuffer[256];
char packetBuffer[26];
char DevEUI[] = "47:68:B2:69:00:51:00:47";
char AppEUI[] = "70:B3:D5:7E:F0:00:68:82";
char devAddr[] = "00:F9:19:C0";
char nwkSKey[] = "2B7E151628AED2A6ABF7158809CF4F3C";
char appSKey[] = "2B7E151628AED2A6ABF7158809CF4F3C";
char AppKey[] = "2B7E151628AED2A6ABF7158809CF4F3C";

const int pin_battery_status  = A5;
const int pin_battery_voltage = A4;

  //Setting gateway possition
  //Grao_LAT = 38.991570758942146, Grao_LON = -0.1646232604980469;
static const double GTW_LAT = 38.991570758942146, GTW_LON = -
0.1646232604980469;



#define LoraDeviceMode LWOTAA//LWABP // LWOTA or LWABP
#define LoraDataRate DR0
// datarate SF/BW bits/s
// DR0 SF12/125KHz 250
// DR1 SF11/125KHz 440
// DR2 SF10/125KHz 980
// DR3 SF9/125KHz 1760
// DR4 SF8/125KHz 3125
// DR5 SF7/125KHz 5470
// DR6 SF7/250KHz 11000
// DR7 FSK:50kbps 50000
#define LoraPower 14
#define LoraPort 1
#define LoraADR 1
#define LoraNET 0//1: Public net
#define DEBUG 0


void setup(void)
{
  if (DEBUG) {
    SerialUSB.begin(115200);
     while(!SerialUSB);
  }
  Serial.begin(9600);
  lora.init();
  pinMode(pin_battery_status, INPUT);

  if (DEBUG) {
    memset(memBuffer, 0, 256);
    lora.getVersion(memBuffer, 256, 1);
    SerialUSB.print(memBuffer);
  }
```

```
  // Disable UART timeout
  memset(memBuffer, 0, 256);
  SerialLoRa.print("AT+UART=TIMEOUT, 0\r\n");

  if (DEBUG) {
    SerialUSB.print(memBuffer);
  }
  // Set public network key
//  lora.setPubNetwKey(LoraNET);
  // NwkSKey, AppSKey, AppKey
  lora.setKey(nwkSKey, appSKey, AppKey);
  lora.setId(devAddr, DevEUI, AppEUI);

  if (DEBUG) {
    SerialUSB.println();
  }

  lora.setDutyCycle(false);
  lora.setJoinDutyCycle(false);
  lora.setChannel(0, 868.1, DR0, DR7);
  lora.setChannel(1, 868.3, DR0, DR7);
  lora.setChannel(2, 868.5, DR0, DR7);
  lora.setReceiceWindowFirst(0,868.1);
 // lora.setReceiceWindowSecond(869.525, DR3);
  // Set RXwin2
  memset(memBuffer, 0, 256);
  SerialLoRa.print("AT+RXWIN2=869.525,DR3\r\n");

  if (DEBUG) {
    SerialUSB.print(memBuffer);
  }
  lora.setPower(LoraPower);
  lora.setDeciveMode(LoraDeviceMode);

  //lora.setChannel(1, 0);
  //lora.setChannel(2, 0);
  lora.setDataRate(LoraDataRate, EU868);
  lora.setPort(LoraPort);

  if (DEBUG) {
    SerialUSB.println("Setup completed");
  }

  while(!lora.setOTAAJoin(JOIN));
}


void loop(void)
{


  String packetString = "";
  packetString = get_gpsdata();

  if (DEBUG) {
    SerialUSB.println(packetString);
  }

  int strLength = packetString.length() + 1;
  packetString.toCharArray(packetBuffer, strLength);
```

```
  if (DEBUG) {
    SerialUSB.println("Start transmission");
  }
  bool result = false;
  result = lora.transferPacket(packetBuffer, strLength);
  lora.transferPacket(lpp.getBuffer(), lpp.getSize());
  if (result)
  {
    short length;
    short rssi;
    memset(memBuffer, 0, 256);
    length = lora.receivePacket(memBuffer, 256, &rssi);

    if (DEBUG) {
      if (length)
      {
        SerialUSB.print("Length is: ");
        SerialUSB.println(length);
        SerialUSB.print("RSSI is: ");
        SerialUSB.println(rssi);
        SerialUSB.print("Data is: ");
        for (unsigned char i = 0; i < length; i ++)
        {
          SerialUSB.print("0x");
          SerialUSB.print(memBuffer[i], HEX);
          SerialUSB.print(" ");
        }
        SerialUSB.println();
      }
    }
  }
  //delay(10 * 10);
  //delay(1);
}
String get_gpsdata() {
  bool newData = false;

  lpp.reset();
  String returnString = "";
  float flat, flon;
  unsigned long chars;
  unsigned short sentences, failed;
  unsigned long age;
  // For one second we parse GPS data and report some key values
  for (unsigned long start = millis(); millis() - start < 1000;)
  {
    while (Serial.available())
    {
      char c = Serial.read();

     Serial.write(c); // uncomment this line if you want to see the
GPS data flowing

     if (gps.encode(c)) // Did a new valid sentence come in?
        newData = true;
    }
  }
  if (newData)
  {
    gps.f_get_position(&flat, &flon, &age);
    gps.stats(&chars, &sentences, &failed);
```

```
    lpp.reset();
    //lpp.addAnalogInput(4, gps.f_speed_kmph());
    lpp.addGPS(1, flat, flon,gps.f_altitude());
//    lpp.addAnalogInput(5, gps.satellites());
//    lpp.addAnalogInput(3, gps.f_course());
//    lpp.addAnalogInput(6, gps.hdop());
//    lpp.addAnalogInput(7,gps.f_speed_kmph());
//    lpp.addAnalogInput(4,age);
//    lpp.addAnalogInput(2, gps.course_to(GTW_LAT, GTW_LON, flat,
flon));
/// For uplink object
   //1
        lpp.addAnalogInput(1, gps.satellites());
   //2
        lpp.addAnalogInput(2, gps.hdop());
   //3
        lpp.addAnalogInput(3, gps.f_course());
   //4
        lpp.addAnalogInput(4,gps.f_speed_kmph());


   // returnString += ";LAT=";
    returnString += String(flat, 6);
   // returnString += ";LNG=";
    returnString +=' '+ String(flon, 6);
    //returnString += ";ST=";
    returnString +=' '+ String(gps.satellites());
    //returnString += ";PRC=";
    returnString +=' '+ String(gps.hdop());
    //returnString += ";A=";
    returnString +=' '+ String(age);
    //returnString += ";ALT=";
    returnString +=' '+ String(gps.f_altitude(),2);
    //returnString += ";SPD=";
    returnString +=' '+ String(gps.f_speed_kmph(),2);
    //returnString += ";COURSE=";
    returnString +=' '+ String(gps.f_course(), 2);

    Serial.print(gps.course_to(GTW_LAT, GTW_LON, flat, flon));
  } else {
    returnString = "NO GPS data available!";
  }
  return returnString;

}
```

Apéndice 4 Código Nodo entorno Arduino

Bibliografía

Fuentes de Información Online:

Seeed, Wiki Seeeduino LoRaWAN [fecha de consulta: 15 de mayo]:
http://wiki.seeedstudio.com/Seeeduino_LoRAWAN/

LoRa-Alliance, “What is LoRaWAN?” [fecha de consulta: 12 de diciembre]:
https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf

O. Brocaar, LoRaServer [fecha de consulta: 25 de marzo]:
https://www.loraserver.io/

Rishing, Wiki-Rishing [fecha de consulta: 5 de diciembre]:
http://wiki.risinghf.com/doku.php?id=start

TTN, The Things Network [fecha de consulta: 3 de febrero]:
https://www.thethingsnetwork.org/docs/lorawan/

Cayenne- TTN, API Reference Cayenne LPP [fecha de consulta: 10 de Enero]:
https://www.thethingsnetwork.org/docs/devices/arduino/api/cayennelpp.html

Sandbox, Sandbox Documentation [fecha de consulta: 15 de enero]:
https://getsandbox.com/docs/http-request-handling

Base64 Decode and Encode [fecha de consulta : 2 de Enero]
https://www.base64decode.org/

Referencias

[i] https://www.seeedstudio.com/LoRa-LoRaWAN-Gateway-868MHz-Kit-with-Raspberry-Pi-3-p-2823.html
[ii] https://www.loraserver.io/lora-gateway-os/overview/
[iii] https://jwt.io/
[iv] https://getsandbox.com/
[v] https://lodash.com/docs/4.17.11
[vi] https://momentjs.com/docs/
[vii] https://github.com/marak/Faker.js/
[viii] https://ajv.js.org/
[ix] https://github.com/validatorjs/validator.js
[x] https://www.arduino.cc/en/Main/Software
[xi] https://github.com/SeeedDocument/Seeeduino_LoRa/raw/master/res/driver.zip
[xii] https://github.com/mikalhart/TinyGPS
[xiii] https://github.com/sabas1080/CayenneLPP
[xiv] https://github.com/mapsplugin/cordova-plugin-googlemaps-doc
[xv] https://www.thethingsnetwork.org/article/ground-breaking-world-record-lorawan-packet-received-at-702-km-436-miles-distance