



Universidad Nacional de Rosario  
Facultad de Ciencias Exactas, Ingeniería y Agrimensura  
Escuela de Ingeniería Electrónica

## Proyecto de Ingeniería

# Implementación red LoRaWAN

Autores:

Echavarria, Mariano (E-1039/1)  
Pellegrino, Adrián (P-4464/4)

Director:

Dr. José Coronel

# 1 RESUMEN

LoRaWAN es un protocolo de red que usa la tecnología LoRa para comunicar y administrar dispositivos con una fuerte aplicación a Internet de las Cosas (IoT). Una red LoRaWAN se compone de tres partes fundamentales: nodos, gateways y servidores. Los nodos son los que envían pequeños paquetes de datos al gateway mediante radiofrecuencia y modulación LoRa. El gateway toma el paquete y lo envía al servidor correspondiente (por Ethernet, 4G, WiFi, etc.). Se entiende por servidor al conjunto de servidores que pueden o no funcionar en el mismo hardware, cada uno tiene una tarea específica según la función que cumpla dentro de la infraestructura completa de la red. El servidor administra los nodos, gateways y paquetes que recibe y se encarga de entregar la información de la aplicación al usuario final. Esta tecnología es bidireccional, por lo que se puede enviar paquetes Uplink (desde el nodo al servidor) y Downlink (desde el servidor al nodo), siempre que se respete las temporizaciones de transmisión.

En este informe se presenta inicialmente el estudio de la tecnología LoRaWAN, su elección frente a Sifox y la selección de los parámetros a utilizar, debido a que nos encontramos en un contexto donde aún no se ha regularizado completamente la tecnología.

Posteriormente se muestra la implementación de la red LoRaWAN. Esto incluye el desarrollo del nodo, con la placa de desarrollo FRDM KL46Z y el transeiver LoRa RFM95W, el programa ejecutado por la placa es de Semtech y StackForce, adaptado especialmente para poder ejecutarse en dicha placa. El gateway se implementó con una placa Raspberry Pi 3 con el concentrador LoRa de la marca RAK (modelo RAK831). Finalmente se dejó operativo el servidor “loraserver.io” en una PC con sistema operativo Ubuntu, también es posible correr todo lo necesario para el funcionamiento del servidor dentro de la misma Raspberry Pi 3.

Una vez operativa la red se realizaron los ensayos necesarios para validar el protocolo LoRaWAN y corroborar todas las características que ofrece, como ser: activación personalizada o por aire, data rate adaptativo y uplink confirmado.

Finalmente se hace un pequeño ensayo del alcance de la señal por dos áreas en la ciudad de Rosario.

1	RESUMEN .....	2
2	INTRODUCCION .....	5
3	OBJETIVOS .....	6
3.1	Objetivos generales .....	6
3.2	Objetivos específicos.....	7
3.3	Ensayos pretendidos .....	7
4	MARCO TEORICO .....	9
4.1	Tecnología LoRaWAN y aspectos técnicos.....	9
4.2	Aplicaciones .....	10
4.3	Regulaciones (uso del espectro y potencia) .....	11
4.3.1	Componentes de una red LoRaWAN .....	12
4.4	Selección de los componentes de la red .....	14
4.4.1	Nodo .....	14
4.4.2	Gateway .....	23
4.4.3	Servers .....	28
4.5	Topología final de la red .....	32
5	DESARROLLO.....	32
5.1	Ensayos preliminares.....	32
5.2	Estudio y desarrollo del GATEWAY.....	39
5.2.1	Protocolo de comunicación del Gateway .....	39
5.2.1.1	Protocolo de subida .....	39
5.2.1.2	Protocolo de bajada .....	44
5.3	Estudio y desarrollo del SERVIDOR .....	50
5.3.1	Componentes del Servidor.....	50
5.3.1.1	LoRa Gateway Bridge.....	51
5.3.1.2	LoRa Server .....	51
5.3.1.3	Servidor de aplicación LoRa .....	52
5.4	Estudio y desarrollo del NODO .....	62
5.4.1	Plan de frecuencia y parámetros .....	62
5.4.2	Protocolo nodo clase A (All) .....	64
5.4.3	Protocolo nodo clase B (Beacon) .....	65
5.4.4	Protocolo nodo clase C (Continuously Listening).....	67
5.4.5	Flujo de datos en la red LORAWAN .....	68

5.4.6	Activación del dispositivo final (Join) .....	68
5.4.7	Máscara de canal .....	69
5.4.8	Mensajes confirmados .....	69
5.4.9	Mensaje Uplink confirmado .....	69
5.4.10	Mensaje downlink confirmado.....	70
5.4.11	ADR .....	70
5.4.12	Hardware .....	71
5.4.13	Aplicación del nodo .....	73
5.4.14	Código .....	73
5.4.14.1	Características del código .....	73
5.4.14.2	Estados del sistema LoRa.....	75
5.4.14.3	Variables de utilidad .....	77
5.4.14.4	Funciones de utilidad .....	79
5.5	Ensayo de la RED LORAWAN .....	80
5.5.1	Ensayo Clase A.....	82
5.5.2	Ensayo clase B .....	87
5.5.3	Ensayo clase C .....	94
5.5.4	Ensayo activación ABP .....	97
5.5.5	Ensayo activación OTAA .....	98
5.5.6	Ensayo Uplink no confirmado.....	102
5.5.7	Ensayo de Adatative Data Rate (ADR). ....	103
5.5.8	Ensayo de paquetes duplicados .....	107
5.5.9	Ensayo de Alcance.....	111
5.5.9.1	Ubicación 1 (Tucumán 1346).....	113
5.5.9.2	Ubicación 2 (9 de Julio 3940) .....	113
6	CONCLUSIONES.....	116
7	BIBLIOGRAFÍA .....	118
8	GLOSARIO .....	119

## 2 INTRODUCCION

El término Internet de las cosas o del inglés Internet of Thing (IoT) se encuentra cada dia más presente en el área de las telecomunicaciones. Ello nos motiva al estudio e implementación de una tecnología que aplique esta nueva dinámica de comunicación. En este contexto las redes Low-Power Wide-Area Network (LPWAN) respalda el esquema de IoT, conectando objetos/sensores de forma inalámbrica con una baja tasa de bit y un bajo consumo de energía dando como resultado una mayor durabilidad de la batería del transmisor. Además al ser una tecnología de largo alcance permite hacer una transmisión de kilómetros dando paso a nuevas aplicaciones.

Entre las redes LPWAN más importantes se encuentran LoRa y Sigfox. Ambas tienen muchas similitudes y su principal objetivo es lograr el mayor alcance con la menor potencia. Sigfox se caracteriza por utilizar una banda espectral ultra-estrecha y cuenta con modulación Gaussian Frequency-Shift Keying (GFSK) mientras que LoRa utiliza mayor ancho espectral y modulación Chirp Spread Spectrum (CSS), a pesar de estas diferencias ambos obtienen una excelente relación señal a ruido aún en largas distancias.

Frente a estas alternativas se optó por el estudio e implementación de una red LoRaWAN porque a comparación de Sigfox presenta una tecnología flexible, versátil, y con menos restricciones para realizar la comunicación. A su vez LoRa permite crear, configurar y administrar una red propia mientras Sigfox es propietaria y requiere de licencia. Los nodos finales de LoRa quedan a desarrollo del cliente (cumpliendo con algunos requisitos posteriormente se pueden certificar). Por otro lado, las transmisiones de datos para Sigfox son limitadas a una cierta cantidad de mensajes por día a diferencia de LoRa que si bien presenta limitaciones estas se basan en el tipo de transmisión y tamaño del paquete. Por estas razones LoRa es probablemente la mejor opción para enlaces bidireccionales debido al enlace simétrico, dando así mejor funcionalidad de comando y control. A continuación, se muestra una tabla comparativa de ambas tecnologías.

Características	LoRaWAN	Sigfox
Frecuencia	433/868/780/915MHz (ISM)	868/902 MHz (ISM)
Tamaño de paquete	Definido por el usuario	12 bytes
Tasa de bits	Max. 30Kbps	Max. 100bps
Modulación	CSS	UNB/GFSK/BPSK
Inmunidad a las interferencias	Muy alta	Baja
Costo del modulo	Bajo	Bajo
Ancho del canal	125-500 KHz	100Hz
Licencia	Propietarios de la capa física pero MAC libre	Se debe usar su red
Alcance	2-5 Km	3-10 Km

Tabla 1: Comparación entre LoRaWAN y Sigfox

Actualmente no se encuentran aplicaciones IoT o son muy pocas las del mercado nacional. Además esta nueva tecnología promete mejorar la calidad de vidas de las personas, los procesos industriales, el control y cuidado del medio ambiente, la seguridad, entre otros. Por estas razones resulta estratégico investigar e implementar posibles soluciones empleando esta nueva tecnología.

## 3 OBJETIVOS

### 3.1 Objetivos generales

Estudiar la tecnología LoRa y LoRaWAN para construir un soporte IoT a través de una infraestructura LoRaWAN. Teniendo como fin la comunicación (mediante LoRa) de los dispositivos finales hasta el servidor de aplicación, el cual interactuara con el usuario final.

Utilizando transmisores LoRa en los dispositivos finales, gateway de recepción y un servidor para procesar los paquetes, se pretende hacer la base para una red LoRaWAN que quedara disponible tanto para posteriores desarrollos de aplicaciones IoT, como también para mejorar, expandir o escalar la red con el fin de tener mayor alcance y/o mayor admisión de dispositivos.

Se realizarán ensayos de campo de la red, con el fin de obtener información para realizar futuras mejoras y saber cuál es el límite de la red. Entre otras cosas esto permitiría conocer, el alcance que tendrán las aplicaciones a realizar e implementarse y si los requisitos necesarios para determinadas aplicaciones son mayores que los de la red, esta se pueda ampliar sin ninguna dificultad.

## 3.2 Objetivos específicos

- Entender la tecnología LoRa y cuáles son sus principales parámetros a ajustar para adaptar la red a una determinada aplicación.
- Conocer las limitaciones de LoRa.
- Construir el hardware necesario para desplegar la red. Esto incluye a los dispositivos finales, gateways y servidores.
- Lograr la comunicación LoRa del hardware mencionado en el ítem anterior.
- Ensayar la red. Alcances máximos de la señal, comunicación de las diferentes clases de dispositivos finales, impacto de las antenas a la recepción de señal y desempeño del servidor.
- Analizar y concluir la información obtenida del ensayo del ítem anterior.

## 3.3 Ensayos pretendidos

### **Alcance máximo de la comunicación**

Para este ensayo se pretende determinar la máxima distancia a la cual es posible realizar una transferencia de datos bidireccional. Se tomarán distintos puntos (para el nodo) de la ciudad para determinar también la influencia de la edificación en el link-budget. También tomaremos 2 puntos de referencia para el Gateway, uno será en la ciudad universitaria, más precisamente en el CIFASIS, y el otro será un edificio del centro de la ciudad (Pellegrini 652, 9no piso) que tiene poca interferencia alrededor.

A su vez, por cuestiones de disponibilidad y tiempo de entrega del hardware seleccionado, tenemos la posibilidad de realizar el ensayo con 2 gateways distintos, uno será provisorio y de un solo canal, el sistema de host será una Raspberry Pi 3 y el hardware front-end LoRa será el shield LoRa que tiene integrado el transceiver RFM95W. Vale aclarar en éste punto que al momento de realizar el ensayo con este Gateway se utilizará como back-end el servidor público de The Things Network (TTN), aunque el servidor utilizado no influye en el alcance máximo que se puede lograr. El otro estará formado por el mismo sistema de host pero el front-end será el concentrador desarrollado por la empresa RAK (modelo RAK831), éste será multicanal y nuestro gateway definitivo, tendrá como back-end el servidor privado que pondremos en funcionamiento.

Aquí veremos cómo influyen los parámetros Spread Factor y Data Rate propios de LoRa en el alcance, especialmente esperamos establecer qué valores optimizan el alcance al ser configurados en el nodo y cómo influye esto en sobre los otros parámetros importantes de la comunicación como la relación señal ruido.

## **Antena**

Muchos desarrolladores que trabajan con LoRa optimizan la antena utilizada o directamente construyen una propia para llevar la eficiencia del gateway al máximo, ya que si la antena no está optimizada para trabajar en la frecuencia seleccionada se pierde potencia en la salida de la señal a la antena, afectando directamente el link-budget y por lo tanto el alcance máximo que se puede lograr. Pretendemos analizar distintas antenas para esto y de ser posible implementar algún diseño de los que hemos visto. A estas antenas se pretende ensayarlas para medir el ROE (relación de onda estacionaria) de esta forma sabremos cuán optimizadas están a la frecuencia de trabajo (915MHz), y que nos dará un indicador del máximo alcance de la comunicación.

## **Respuesta del server a paquetes repetidos**

Como dispondremos de dos gateways, a pesar de que uno será de canal simple y provisorio, podremos ensayar el comportamiento del servidor ante la llegada de 2 paquetes idénticos provenientes de distintos gateways pero del mismo nodo. En este punto ya tendremos el servidor en funcionamiento y lo pondremos como back-end del gateway provvisorio.

## **Configuración del nodo**

Con éste ensayo pretendemos ver las diferencias, ventajas y desventajas de las dos posibilidades de configuración que existen del nodo. En la configuración OTA (Over the Air) el servidor manda la configuración necesaria a través del gateway hacia el nodo. Por otro lado en la configuración por personalización, se configuran manualmente todos los parámetros necesarios directamente en el nodo.

## **Clases de nodo**

Se pretende evaluar el funcionamiento de los tres tipos de clases disponibles para los nodos. El tipo de ensayo dependerá de la clase a ensayar y se definirá más cerca del momento de ensayo, en principio sería evaluar el correcto funcionamiento de las tres clases (A, B y C) y corroborar que las transmisiones respetan los tiempos de espera que le corresponden a cada clase.

## **Conectividad**

En este ensayo se pretende analizar la influencia sobre los parámetros generales del sistema de la conectividad seleccionada para el gateway (3G, LTE, WiFi, ethernet). Se realizarán varias transferencias de datos con las distintas conectividades configuradas y se analizarán los parámetros más importantes de la comunicación para detectar, si es que existe, algún tipo de influencia.

## 4 MARCO TEORICO

### 4.1 Tecnología LoRaWAN y aspectos técnicos

LoRaWAN [1] es una especificación de redes LPWAN. En relación al modelo OSI, LoRaWAN se posiciona en la capa 2 (enlace), es lo que se conoce como MAC (Media Access Control). Es decir que LoRaWAN se encarga de unir diferentes dispositivos LoRa gestionando sus canales y parámetros de conexión: canal, ancho de banda, cifrado de datos, etc.

En la capa 1 del modelo OSI (capa física) encontramos la tecnología LoRa de comunicación. Esta tecnología permite el envío y recepción de información punto-a-punto. Debido a la quasi-ortogonalidad de modulación y el uso de espectro ensanchado se puede decodificar múltiples señales a pesar de usar la misma frecuencia. Esta característica dota a LoRa de "canales virtuales".

El espectro de frecuencias que emplea LoRa están ubicados en las bandas ISM (Industrial, Scientific and Medical), aunque la tecnología puede operar en cualquier frecuencia por debajo de 1 GHz. En esta banda cualquier persona o empresa puede hacer uso de ella sin necesidad de licencia. Así pues, LoRa suele operar en las bandas 433 MHz, 868 MHz y 915 MHz según el país.

Los parámetros más utilizados de la comunicación LoRa son:

- **Canales:** son las bandas de frecuencia por donde se transmite la modulación LoRa, puede tener ancho de banda de 150, 250 y 500khz. Cada canal se identifica con la frecuencia central.
- **Spreading Factor (SF):** define el número de bits usados para codificar un símbolo. A mayor SF se tendrá menor velocidad de transferencia pero mayor inmunidad a interferencias llevando un mayor alcance. El SF se puede configurar de SF7 a SF12 como máximo.
- **Coding Rate (CR):** indica la forma de codificar para corrección de errores. Es decir, según la técnica especificada, añade símbolos de control para saber si los datos son correctos o no e incluso poder determinar los valores correctos.
- **BandWidth (BW):** indica la longitud (medida en Hz) del conjunto de frecuencia en el espectro que se va a utilizar, cuyos valores para Lora pueden ser 125, 250 o 500 KHz.
- **DataRate (DR):** mientras LoRa permite configurar manualmente los parámetros de transmisión, como la potencia de transmisión y el SF, LoRaWAN utiliza una abstracción sobre esos parámetros llamados "datarate". La velocidad de datos o DR es un valor entero entre 0 y 15 y se denotan de DR0 a DR15. Este parámetro es el resultado de la combinación del SF y el ancho de banda que se utiliza. Si

bien hay 16 DR su configuración está ligada a la banda ISM que se utiliza, además no todos los DR están definidos ya que están reservados para futuros usos.

- **Time on Air (ToA):** Una consecuencia importante del SF para LoRa es el tiempo en el aire. El radio-modulador LoRa necesita más tiempo para enviar la misma cantidad de datos mientras mayor sea el SF.
- **Duty Cycle (DC):** El máximo ciclo de trabajo es el máximo porcentaje de tiempo durante el cual un dispositivo final (nodo) puede ocupar un canal, esta es una restricción clave para operar en las bandas sin licencias. El tiempo fuera de trabajo se calcula como  $T_{off} = \frac{Time\ on\ air}{Duty\ cycle} - Time\ on\ air$  [2].

## 4.2 Aplicaciones

Como se mencionó anteriormente LoRaWAN presenta grandes ventajas para las soluciones IoT, no siendo recomendada para soluciones con requerimientos de comunicación instantánea o que necesiten transmitir gran volumen de datos.

A continuación se presentan unas de las tantas aplicaciones [3] que fueron pensadas para IoT y con posibilidad de aplicarse con LoRaWAN:

**Agricultura:** Los sistemas IoT para el procesamiento agrícola ayudan a tener un seguimiento de los productos del campo (cultivos, ganado, etc.) a través del proceso de producción para garantizar un producto de alta calidad, fresco y seguro. Mediante sensores que analizan el suelo y el aire se puede determinar los tiempos de cosecha. También se puede lograr un rastreo y control de animales.

**Control ambiental:** Monitoreo general de la contaminación del aire que cubre todas las áreas principales de una ciudad, incluyendo: calles, rutas y autopistas - escuelas, edificios, centros del centro - Áreas Industriales - Parques, piscinas y otras áreas de recreación.

**Salud para la tercera edad:** Reducir el impacto y las consecuencias de accidentes domésticos para la tercera edad, mediante detección y aviso de su ocurrencia, como también llamado inmediato a la central médica para asistir la situación.

**Detección de anormalidades domésticas:** Los sensores colocados en un edificio comercial pueden detectar signos de un incendio como ser calor, humo, gas o llamas, y dar una respuesta más rápida para la seguridad de los inquilinos y reducir el daño a la propiedad.

**Rastreo:** Localización de maquinarias de construcción y agrícola para una mejor logística a la hora de producir.

**Seguimiento:** Hacer un seguimiento de los vehículos de la flota de transporte permitirá gestionar mejor su operación, seguridad y mejorar la rentabilidad.

**Seguridad para el hogar:** Mediante sensores con baterías se puede incrementar su número para mejorar la seguridad en los lugares del hogar, como ser para dar aviso inmediato al propietario o al personal de seguridad.

**Estacionamiento:** Un sistema de monitoreo que proporciona el estado de ocupación en un pseudo tiempo real a los conductores, garajes y sistemas de coordinación de tráfico en las ciudades.

**Detección de falla industrial:** Los sensores colocados en una instalación industrial pueden monitorear el funcionamiento de los equipos para avisar a los administradores de las instalaciones el estado de las mismas, para que puedan identificar los problemas antes de que ocurra un accidente, de esta forma se reduce el tiempo de inactividad y los costos.

### 4.3 Regulaciones (uso del espectro y potencia)

Al día de la fecha no existen regulaciones concretas con respecto a las nuevas tecnologías LPWAN en Argentina, razón por la cual se toman regulaciones de otros países (como Australia, Nueva Zelanda, Países Bajos) con sistemas de similar carácter ya regularizados o por medidas tomadas por las empresas o entidades que utilizan LoRa actualmente en el país.

En cuanto al uso del espectro se adoptó la banda australiana 915MHz (AU915) que comprende las frecuencias de 915MHz a 928MHz cuya banda pertenece a la banda ISM de 915MHz que va de 902MHz a 928MHz. Misma banda adoptada por varias empresas argentinas que utilizan o venden tecnología LoRa (como Semak o YEAP).

La banda AU915 usada por LoRa está separada en canales de la siguiente forma [4]:

- Upstream – 64 canales numerados del 0 al 63 utilizando un ancho de banda de 125 KHz con posibilidad de variar el data rate de DR0 a DR5, con una taza de codificación 4/5, comenzando en 915.2 MHz e incrementando linealmente de a 200 KHz hasta 927.8 MHz.
- Upstream – 8 canales numerados de 64 a 71 utilizando un ancho de banda de 500 KHz con DR6, comenzando en 915.9 MHz e incrementando linealmente de a 1.6 MHz hasta 927.1 MHz.
- Downstream – 8 canales numerados del 0 al 7 utilizando un ancho de banda de 500 KHz con posibilidad de variar el data rate de DR8 a DR13, comenzando en 923.3 MHz e incrementando linealmente de a 600 KHz hasta 927.5 MHz.

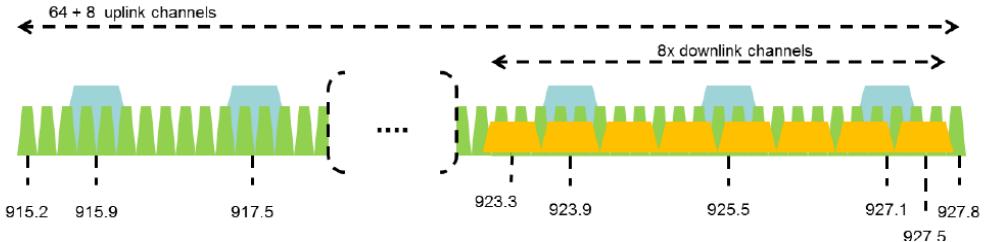


Figura 1: Canales de frecuencias AU915-928

Si se utiliza un server privado se puede hacer uso de cualquiera de estos canales pero en el caso de uno público como el de The Things Network los canales se limitan a:

Uplink: 916.8/917.0/917.2/917.4/917.6/917.8/918.0/918.2/917.5 MHz

Downlink: 923.3/923.9/924.5/925.1/925.7/926.3/926.9/927.5 MHz

Por parte de la potencia transmitida, aún no ha sido regularizado, por eso guiándonos por la Resolución 302/98 (Boletín Oficial Nº 28.834, 11/2/98) [5] encontramos que para transmisiones por espectro expandido dependiendo de la cantidad de frecuencias involucradas en la técnica de modulación por salto de frecuencia, la potencia máxima varía entre 0.25W (para sistemas con 25 a 49 frecuencias de salto) a 1W (para sistemas con 50 o más frecuencias de salto), como los moduladores no superan los 24dBm, esto se comprende dentro del peor caso, por lo cual no hay problemas de incumplimiento con [5], en dicha resolución también se aclara que para la banda de 902 MHz a 928 MHz la ganancia de la antena será tal que la potencia aparente radiada máxima de cresta no debe superar los 6 dBW.

LoRa no solo presenta limitación en su baja tasa de bits que no supera los 30kbps, esta limitación fue hecha a conciencia para tener bajo consumo de potencia y mayor alcance. Pero otra limitación a tener en cuenta es el ciclo de trabajo que es impuesto por el servidor dando un tiempo muerto (Toff) donde el nodo no puede transmitir, esto quiere decir que dependiendo de los parámetros utilizados (payload, spreading factor, bandwidth) que se contemplan en el Time-on-air, el nodo no podrá volver a transmitir hasta pasar el Toff. En Argentina el ciclo de trabajo no está definido, tomando como referencia TTN [6] usamos un ciclo de trabajo del 1% que es el límite para lugares donde aún no se han establecido limitaciones.

#### 4.3.1 Componentes de una red LoRaWAN

LoRaWAN™ define el protocolo de comunicación y la arquitectura del sistema para la red, mientras que la capa física LoRa® permite el enlace de comunicación de largo alcance. El protocolo y la arquitectura de la red tienen la mayor relevancia a la hora de determinar la vida de la batería del nodo, la capacidad de la red, la calidad del servicio, la seguridad y la variedad de aplicaciones provistas por la red.

Una red LoRa típica consiste de 4 partes: dispositivos finales (nodos), gateways, servidores (de red y aplicación) y una aplicación para el usuario final.

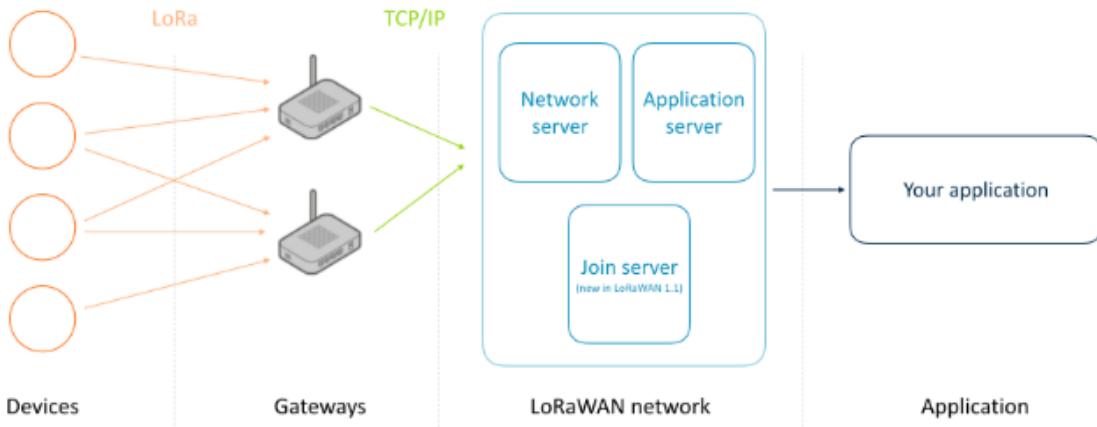


Figura 2: Topología de una red LoRa

Los hardwares que se requieren para la red son: los dispositivos y gateways. Los gateways son los encargados de analizar el espectro y capturar los paquetes LoRa, los nodos no se asocian a ningún gateway, con lo cual todos los gateways dentro del rango de alcance de un dispositivo reciben la señal y redireccionan los datos a un servidor de red que se encarga de manejar los paquetes.

El servidor de red elimina los paquetes repetidos cuando múltiples gateways reciben el mismo paquete, descriptan el mensaje (todo está encriptado, de principio a fin) y maneja las características LoRa, como por ejemplo el ADR (Adaptive Data Rate). Luego reenvía los datos a la aplicación.

A continuación, se detallan cada uno de los elementos involucrados en la red:

1. **Dispositivo final:** es aquel módulo que transmite la información censada mediante un radio LoRa para poder ser alcanzada por el gateway. En nuestro caso nuestro prototipo será una placa de desarrollo que maneje el sensor y un shield LoRa que se encargue de modular y generar la comunicación.
2. **Gateway o Concentrador:** se necesita un gateway o concentrador que “hable” LoRa con los dispositivos finales y que tenga conectividad IP con la nube (mediante Wi-Fi, Ethernet o 3G/4G). Mientras un lado del Gateway “habla” LoRa, el otro lado se conecta a internet (mediante TCP/IP o UDP/IP). El software que hace de interfaz con el backend se lo denomina típicamente Packet Forwarder, y la empresa propietaria de la modulación LoRa, Semtech, ha abierto al público el código fuente del concentrador que ellos han desarrollado.
3. **LoRaWAN Network Server o Backend:** es quien mediante LoRaWAN se comunica con los gateways y una vez procesado los paquetes, envía los datos al server de aplicación y viceversa (la aplicación manda los datos al server de red

para que finalmente y mediante un gateway se envíen los datos al nodo final). Se puede optar por una opción opensource y tener tu propio servidor, o usar algún proveedor de servicios, público o privado, que ya tenga implementado un servidor al que se pueda tener acceso. En nuestro caso vamos a poner en funcionamiento un servidor privado haciendo uso de software opensource.

4. **Aplication server o Aplicación Backend:** la aplicación IoT propiamente dicha, que utiliza el dispositivo o nodo corre en un servidor privado o público, este se comunica con el server de red LoRaWAN para obtener los datos traídos de los nodos y realiza el procesamiento de los datos. El proveedor del servidor establece la interfaz de comunicación entre el backend LoRaWAN y el servidor de aplicación. Luego esta información es dada al usuario final mediante una interfaz gráfica (un software en una computadora o una aplicación para smartphone).

## 4.4 Selección de los componentes de la red

### 4.4.1 Nodo

Para desarrollar el nodo se plantea trabajar con las placas de desarrollo que disponemos actualmente, algunas de las cuales se encuentran en el CIFASIS y nos son facilitadas por ellos: *FRDM-KL46Z*, *FRDM-K64F*, *Raspberry Pi 3*. A dichas placas las utilizaremos en conjunto con un módulo transceptor que trabaje como modem LoRa, en nuestro caso será el *RFM95W* por cuestiones de disponibilidad en el mercado y precio. Existen otros módulos disponibles en el mercado internacional pero resultan difíciles de conseguir en nuestro país y superan en costo la opción propuesta. A continuación se presentan algunas características de este módulo.

#### Descripción general

El RFM95 incorpora un modem de espectro expandido LoRa™, el cual es capaz de lograr un alcance significativamente mayor que los sistemas existentes basados en modulación FSK o OOK (On-Off Keying). Adicionalmente, con este nuevo esquema de modulación se pueden alcanzar sensibilidades mayores que con la modulación FSK. A su vez, esto logra aumentar la relación señal ruido, y por ende, también logra un mayor alcance y robustez. LoRa™ también provee ventajas significativas en la selectividad y el bloqueo respecto de otras técnicas de modulación convencionales, mejorando enormemente la confiabilidad de la comunicación.

Para máxima flexibilidad, el usuario puede decidir sobre el ancho de banda de la modulación de espectro expandido, SF y tasa de corrección de errores. Otro beneficio de este tipo de modulación es que cada SF es ortogonal, de este modo múltiples señales transmitidas pueden ocupar el mismo canal sin producir interferencia. Esto también

permite una simple coexistencia con los sistemas existentes basados en modulación FSK. Las modulaciones estándar GFSK, FSK, OOK y GMSK (Gaussian Minimum Shift Keying) también son provistas para brindar compatibilidad con los sistemas o estándares existentes como MBUS (Meter Bus) inalámbrico y IEEE 802.15.4g.

## Características

El RFM95W es un transceptor half-duplex y de baja frecuencia. La señal de radiofrecuencia recibida es amplificada primero por el LNA (Low Noise Amplifier). Las entradas del LNA están referidas a masa para minimizar el número de componentes usados en el circuito impreso y por la facilidad del diseño. Luego de convertir las señales analógicas en digitales, son procesadas por una máquina de estados digital que controla, entre otras cosas, la corrección automática de frecuencia, el indicador de intensidad de la señal recibida y el control de ganancia automático.

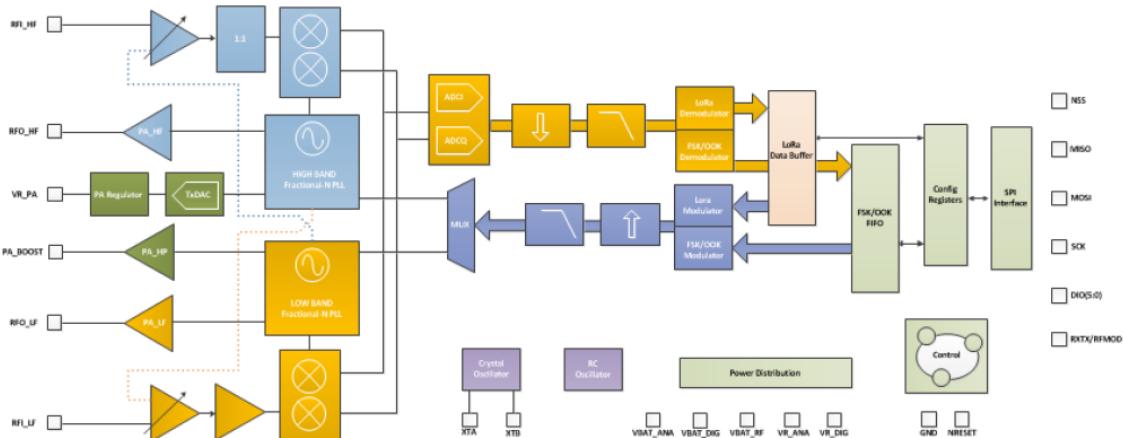


Figura 3: Diagrama de bloques simplificado

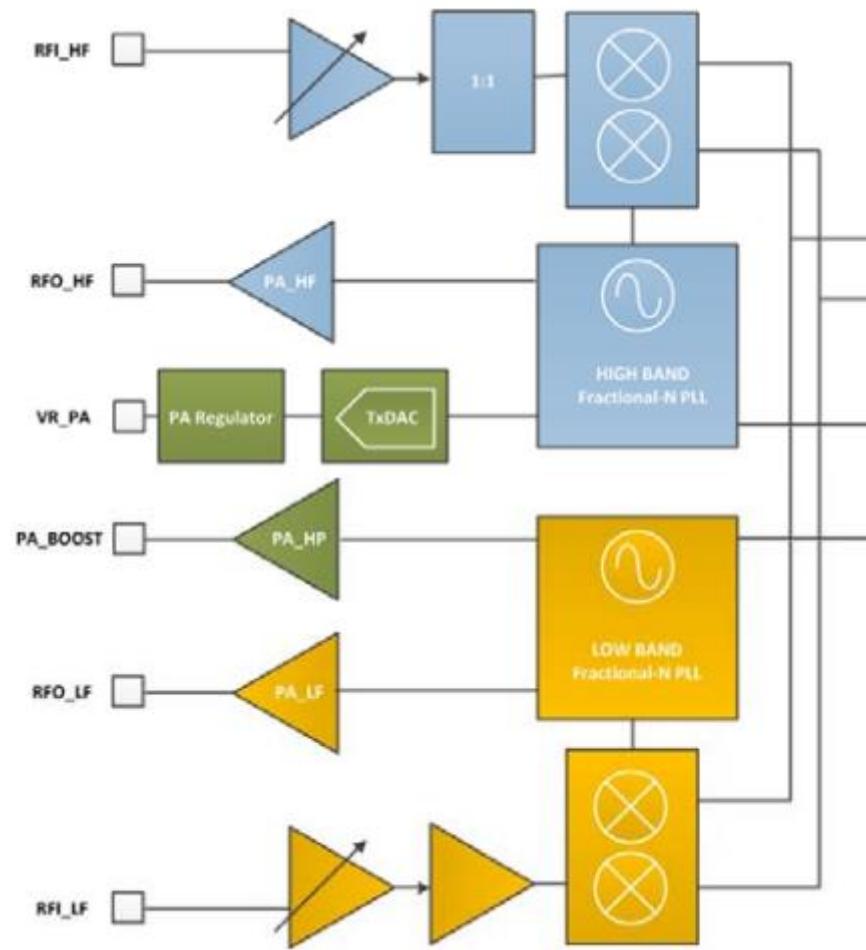


Figura 4: Detalle del diagrama de bloques (A)

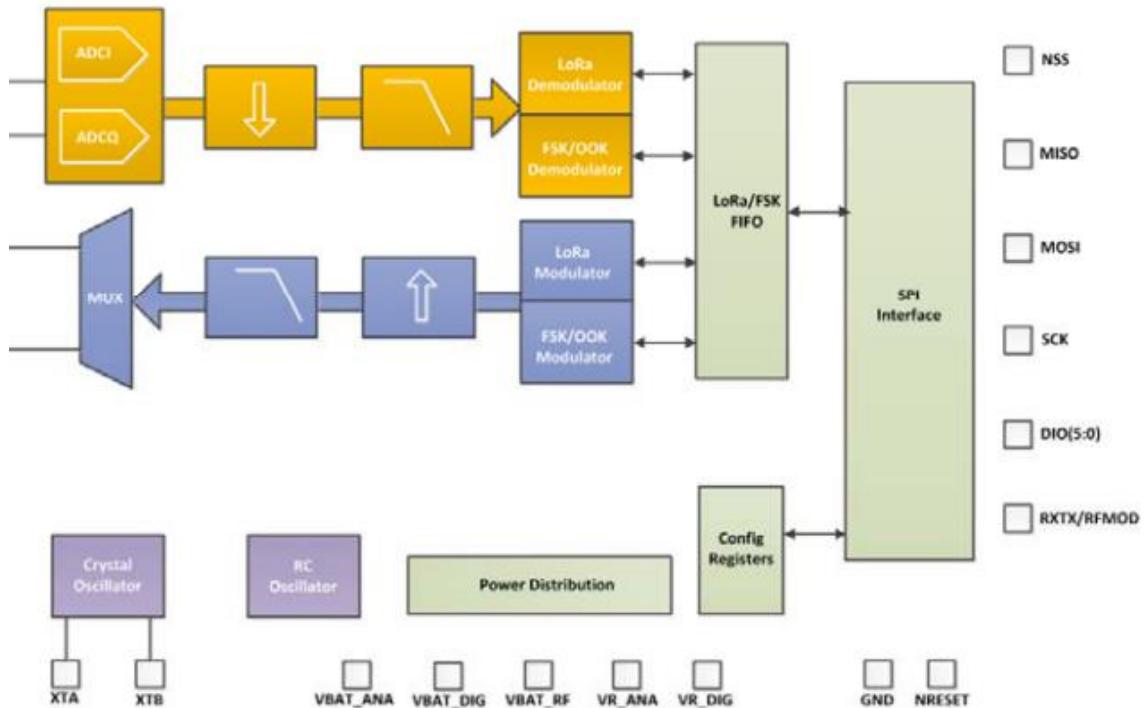


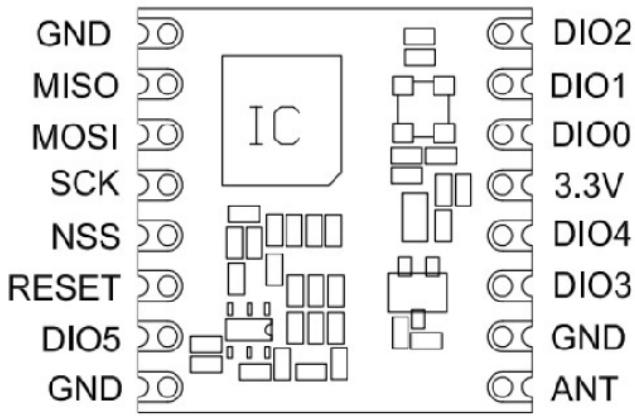
Figura 5: Detalle del diagrama de bloques (B)

El RFM95 tiene también 3 amplificadores de potencia de radiofrecuencia distintos. Dos de ellos pueden entregar hasta +14 dBm, su eficiencia no está regulada para alta potencia y se pueden conectar directamente a sus respectivos receptores a través de un par de componentes pasivos desde un solo puerto transceptor de una antena. El tercer amplificador puede entregar hasta +20 dBm a través de una red dedicada. A diferencia de los amplificadores altamente eficientes, este amplificador altamente escalable cubre todas las bandas de frecuencia que el sintetizador de frecuencia utiliza. También incluye dos referencias de clock, un oscilador RC y un cristal de 32MHz.

Todos los parámetros se pueden configurar completamente a través de una interfaz SPI que brinda acceso a los registros de configuración del transceptor, los pines usados para hacer la conexión SPI se observan en la Figura 6. Este módulo está equipado con dos módems, el estándar FSK y el espectro expandido de largo alcance LoRa™.

El módem FSK es especialmente adecuado para comunicaciones de banda angosta, soporta técnicas de modulación estándares como OOK, FSK, GFSK, MSK y GMSK.

El módem LoRa™ usa una técnica propietaria de modulación de espectro expandido. Esta modulación da un incremento en la inmunidad a las interferencias. Al mismo tiempo, los requerimientos de tolerancia de frecuencia del cristal oscilador de referencia disminuyen, permitiendo un incremento en desempeño y una reducción en el costo del sistema.



GND	Ground
MISO	SPI Data output
MOSI	SPI Data input
SCK	SPI Clock input
NSS	SPI Chip select input
RESET	Reset trigger input
DIO5	Digital I/O, software configured
GND	Ground
ANT	RF signal output/input.
GND	Ground
DIO3	Digital I/O, software configured
DIO4	Digital I/O, software configured
3.3V	Supply voltage
DIO0	Digital I/O, software configured
DIO1	Digital I/O, software configured
DIO2	Digital I/O, software configured

Figura 6: PinOut RFM95W

Para nuestro proyecto utilizaremos el Shield Emging Lora 915mhz, que está basado en el RFM95W y además presenta una fácil conexión con las placas de desarrollos que poseemos (FRDM-KL46Z y FRDM-K64F).



Figura 7: Shield Emging Lora 915mhz

## **Placas de desarrollo**

Al comienzo de la sección se mencionaron las placas de desarrollo, estas tienen las siguientes funciones:

- La placa FRDM-KL46Z manejará el shield del RFM95W y de esta forma crear el nodo.
- La FRDM-K64F también se utilizará para armar un nodo con el shield RFM95W pero además existe la opción de armar un gateway monocanal, debido a que esta placa cuenta con un puerto ethernet.
- Una Raspberry Pi 3 conectada al shield RFM95W cumplirá la función de gateway monocanal.
- Otra Raspberry Pi 3 será la encargada de gestionar el RAK831 para así obtener el gateway multicanal de alta performance.

A continuación se mencionan las características de las placas de desarrollo de las que disponemos:

*FRDM-KL46Z* es una placa de desarrollo perteneciente a la familia Kinetis L series KL4x MCU y construida con el procesador ARM® Cortex-M0+. Dentro de sus principales características se encuentra, el fácil acceso a los pines del MCU, bajo consumo de energía, fácil inserción de tarjetas para expandir las funcionalidades de la placa y una interfaz de depuración incorporada para programación flash y control de ejecución. Además es soportada por un amplio rango de software de desarrollo de NXP o de terceros.

### **Características del MCU:**

- Núcleo de alta performance ARM® Cortex™-M0+
- 48MHz, 32KB RAM, 256KB FLASH
- USB (Host/Device)
- SPI (2)
- I2C (2)
- I2S (1)
- UART (3)
- PWM (6)
- ADC (6)
- DAC (1x 6bit, 1x 12bit)
- Sensor táctil
- GPIO (84)
- Controlador LCD

### Características de la placa:

- Acelerómetro MMA8451Q-3-ejes
- Sensor Capacitive touch
- Magnetómetro MAG3110
- LCD-S401M16KR - 4-digital, 4x8 segmentos LCD
- Sensor Visible light sensor - ALS-PT19-315C/L177/TR8

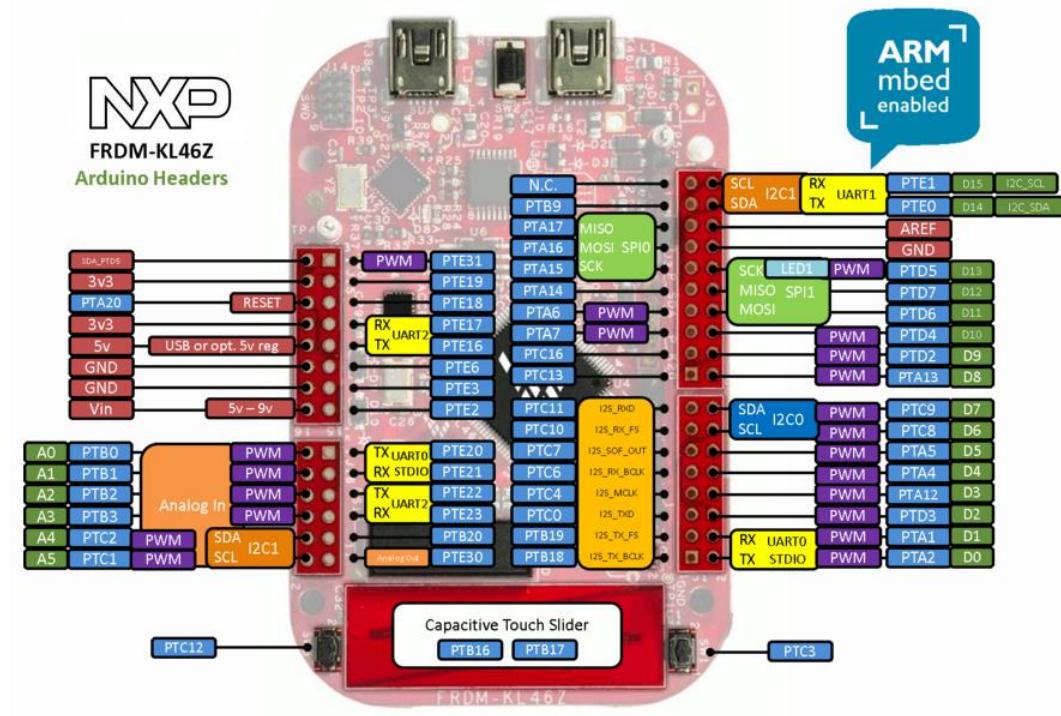


Figura 8: Mapa de pines, FRDM-KL46Z

*FRDM-K64F* es una plataforma de desarrollo perteneciente a la familia Kinetis® K64, K63 y K24 MCUs y construida con el procesador ARM® Cortex-M4. Su diseño de pines es compatible con Arduino R3 y su programación se puede hacer mediante OpenSDAv2, el hardware embebido ofrece la opción de comunicación serial, programación flash y un debug de ejecución.

### Características del MCU:

- Mucleo de alta performance ARM® Cortex™-M4
- 120 MHz, 256 KB RAM, 1 MB flash
- Canales de controladores de acceso directo a memoria (16)
- PLL and FL
- 16-bit ADCs (2)
- 12-bit DACs (2)
- Comparadores analógicos (3)
- 1x 10/100 Mbit/s Ethernet MAC controller with MII/RMII interface IEEE1588

- USB 2.0 (1)
- CAN (1)
- SPI modules (3)
- I2C (3)
- UART (6)
- Secure Digital Host Controller (SDHC) (1)
- I2S (1)

Características de la placa:

- Acelerometro FXOS8700CQ - 6-axis (1)
- Magnetometro (1)
- User push-buttons (2)
- RGB LED (1)
- Micro-USB (1)
- Ethernet RJ45 (1)
- Extensiones
  - Micro SD-Card Socket
  - Bluetooth
  - Shield compatibles con Arduino R3 (32-pins)

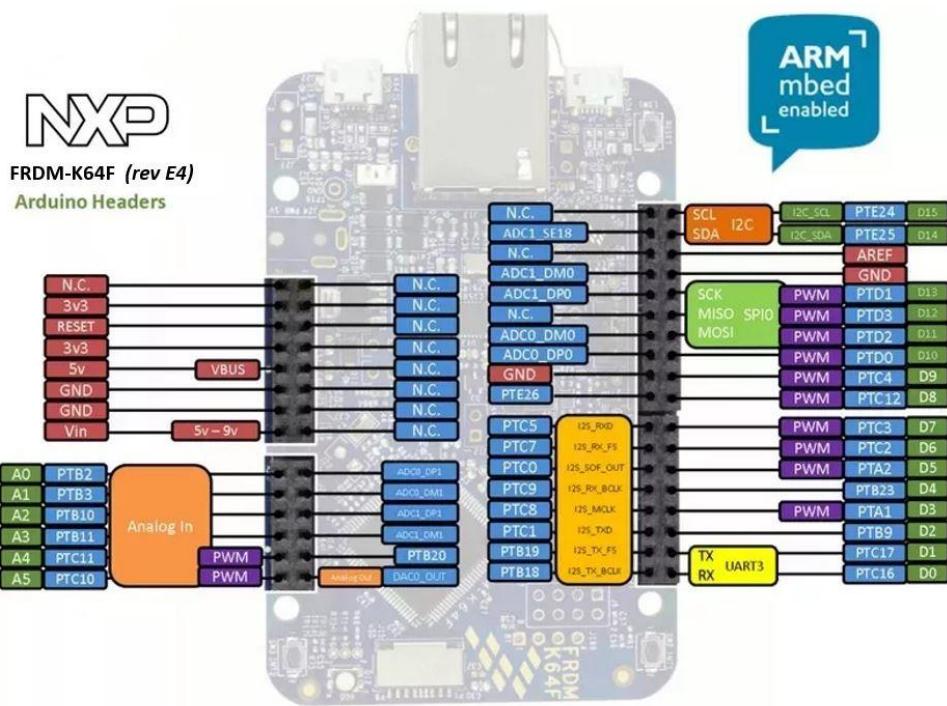


Figura 9: Mapas de pines, FRDM-K64F

Raspberry Pi 3 es una computadora de placa única con conectividad LAN y bluetooth, perteneciente a la tercera generación de Raspberry.

### Características generales:

- CPU Quad Core 1.2GHz Broadcom BCM2837 64bit
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet (1)
- GPIO (40)
- Puertos USB 2 (4)
- Salida de estéreo polar y video compuesto (4)
- HDMI
- Puerto CSI camera para conectar una Raspberry Pi camera
- Puerto DSI display para conectar un Raspberry Pi touchscreen display
- Puerto Micro SD para cargar el sistema operativo y almacenamiento de datos

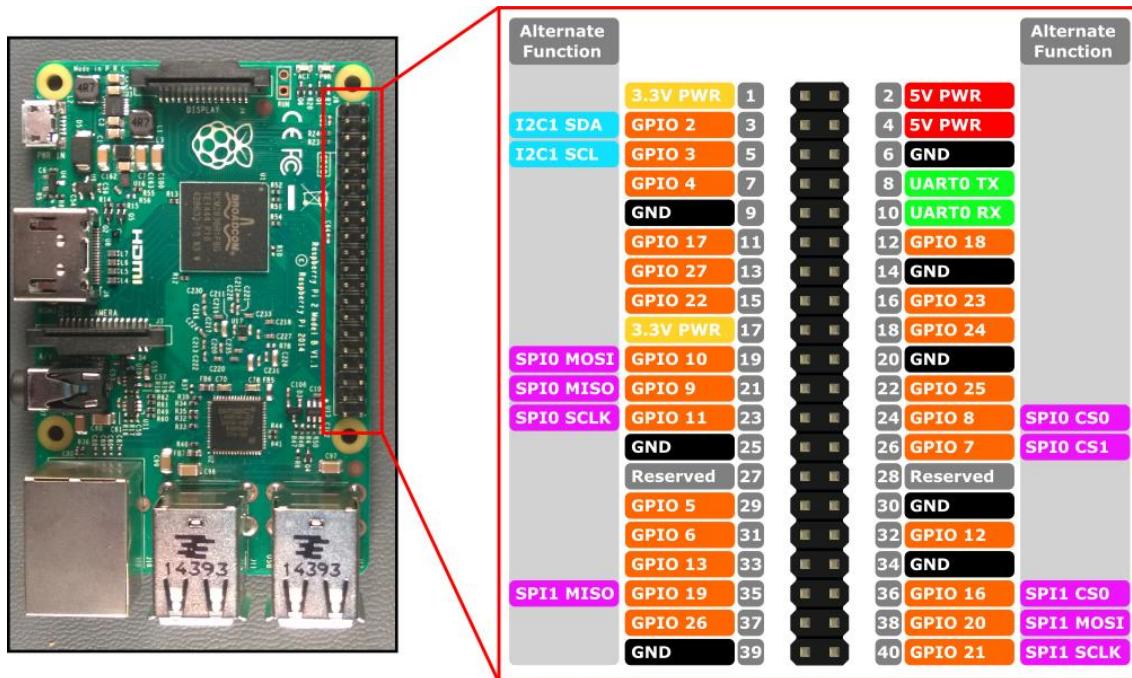


Figura 10: Mapas de pines, Raspberry Pi 3

#### 4.4.2 Gateway

En la búsqueda de distintas opciones nos encontramos con 3 posibilidades.

- **MultiConnect®ConduitTM:** de la empresa Multitech, su principal ventaja es su versatilidad en el tipo de conectividad que se puede utilizar para realizar la conexión a internet, también ofrece distintas plataformas de desarrollo. Además cuenta con el servidor integrado dentro del Gateway. Dependiendo de los módulos que se agreguen el precio oscila aproximadamente los €500.

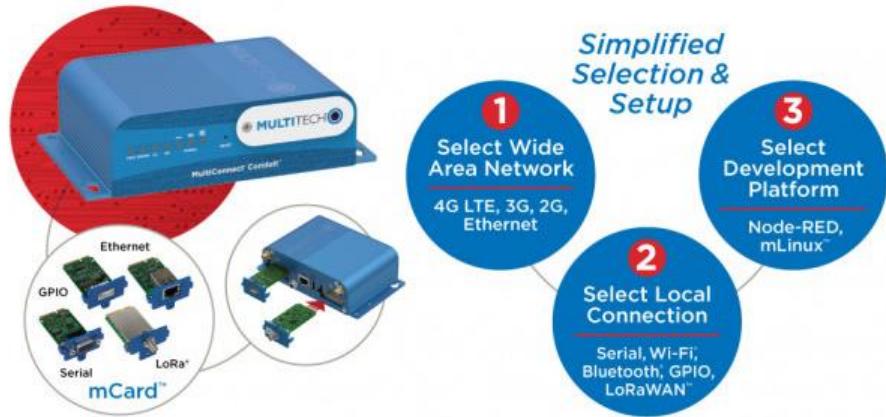


Figura 11: Gateway MULTITECH

- **The Things Gateway:** de la organización The Things Network (TTN), posibilita una implementación rápida y sencilla, ofrece conectividad WiFi o Ethernet y se integra fácilmente con la nube. Si bien no tiene el servidor integrado, TTN tiene servidores de red y de aplicación públicos a los que se puede conectar sin problema. Este Gateway ronda los €300.



Figura 12: Gateway THE THINGS NETWORK

En este punto nos dimos cuenta que si bien hay soluciones muy prácticas que permiten una implementación rápida y directa, nos quita libertad de acción sobre la configuración del servidor, que es la parte más importante de la red ya que comanda todo el flujo de datos. Otro punto en contra de este tipo de dispositivos, que no está relacionado con su funcionalidad pero que influye en nuestra elección, es el presupuesto con el que contamos, y es que al ser soluciones “llave en mano” el precio final es superior al que logramos desarrollando individualmente por cuenta propia cada componente por separado y que además es uno de nuestros objetivos en este proyecto.

Finalmente encontramos un módulo concentrador que podemos utilizar para el Gateway y que se encuentra dentro del presupuesto disponible, además nos permite desarrollar funcionalidades conforme nuestras necesidades.

- **RAK831 módulo gateway LoRaWAN:** un módulo concentrador basado en el chip SX1301 de Semtech. Permite recibir señales en diferentes canales de frecuencia al mismo tiempo y puede demodular la señal LoRa sin saber el SF (spreading Factor) del nodo transmisor. Necesita de un microcontrolador con un sistema que lo controle a través del protocolo SPI (Serial Peripheral Interface). [7]

Veamos entonces algunas de las características más destacables de este módulo, analizando las limitaciones que pueden conllevar.



Figura 13: Gateway RAK

## Diagrama de aplicación

Aquí vemos el papel del módulo dentro de la topología de la red.

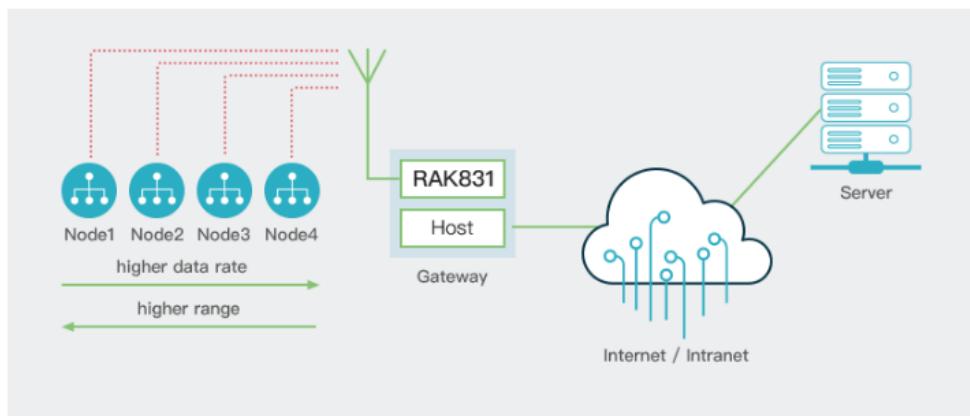


Figura 14: Topología de una red LoRaWan

## Características

- Sensibilidad hasta -142.5 dBm
- Interfaz USB y SPI
- Potencia de salida hasta 23 dBm con un rango de hasta 15km (línea directa de visión)
- Interfaz de radiofrecuencia optimizada para 50 ohm, permitiendo una integración simple del sistema.
- Soporta 10 canales (8 uplink – 1 downlink – 1 FSK)
- Rango de temperatura: -40°C a 85°C

## Sx1301 chip

El RAK831 incluye un chip digital de banda base de Semtech (SX1301) que tiene un gran procesador de señales digitales especialmente diseñado para ofrecer características de Gateway innovadoras en las bandas ISM alrededor del mundo. Tiene integrada la propiedad intelectual del concentrador LoRa.

En la parte receptora, recibe una tira de bits I/Q digitalizados por uno o dos receptores (SX1257), demodula la señal usando varios demoduladores adaptando su configuración a la señal recibida y almacena los paquetes recibidos en una memoria con metodología FIFO (First Input, First Output) para luego ser extraídos desde el sistema que controla el concentrador (PC o microcontrolador). En la parte transmisora, los paquetes son modulados usando un modulador programable (G)FSK/LoRa y son enviados al transmisor SX1257. Los paquetes recibidos pueden ser marcados con un time stamp usando un GPS PPS (Pulse per Second).

Adicionalmente, tiene un bloque de control interno que recibe micro código desde la PC o el microcontrolador. El microcódigo es provisto por Semtech en forma de un archivo

binario que se carga en el chip cuando se enciende. Esta función del bloque de control interno se hace usando Hardware Abstraction Layer (HAL). El código fuente de esta abstracción de hardware es también provisto por Semtech y puede ser adaptado por los desarrolladores del sistema que controla el concentrador.

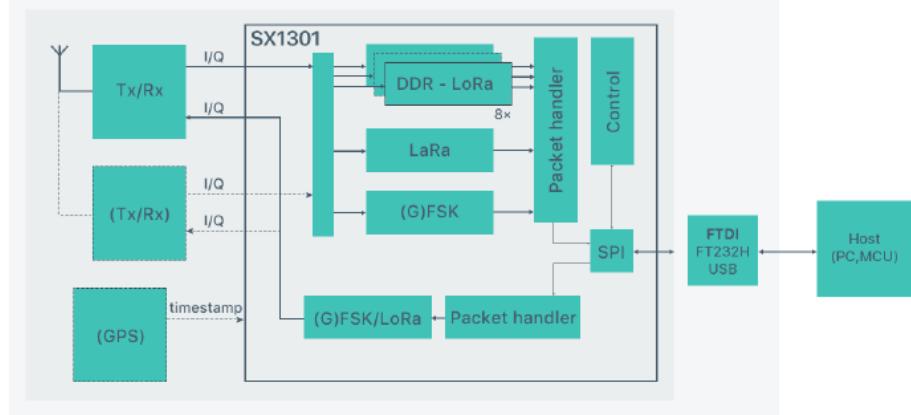


Figura 15: Chip SX1301

### Diagrama de Bloques

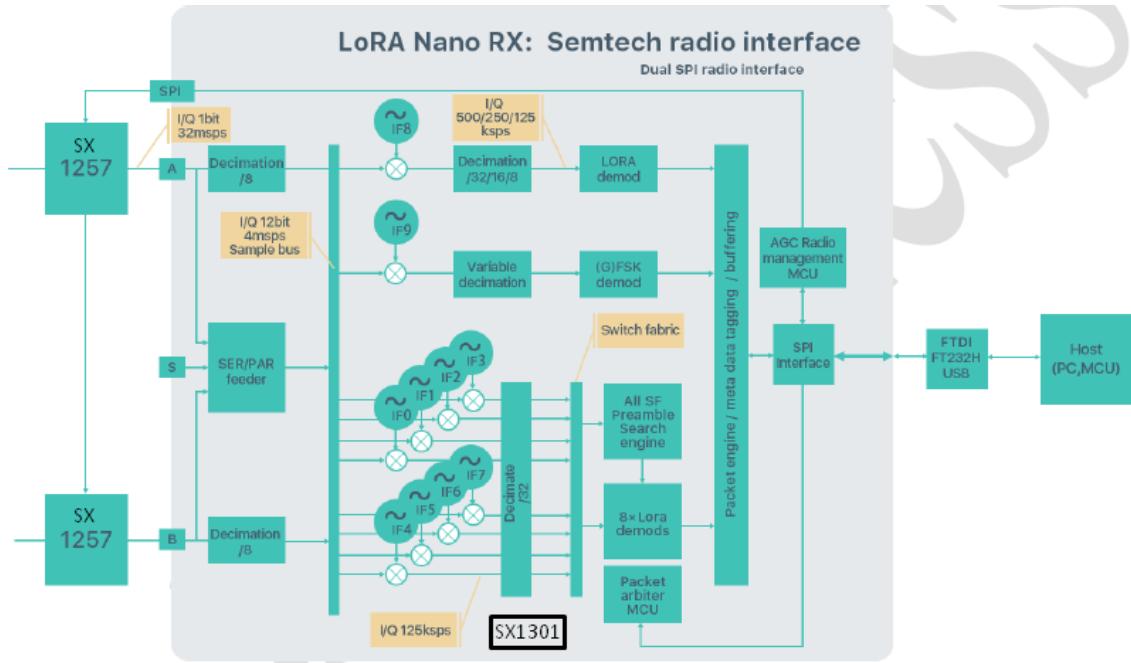


Figura 16: Diagrama de bloques RAK831

El chip digital de banda base SX1301 tiene 10 caminos de recepción programables. Estos caminos tienen distintos niveles de programabilidad y permiten diferentes casos de uso. Es importante entender las diferencias entre esos caminos de demodulación para utilizar de la mejor manera posible el sistema.

### **Canal LoRa IF8**

Este canal está conectado a uno de los SX1257, usando cualquier frecuencia arbitraria dentro del rango permitido. Este canal es solo LoRa. El ancho de banda de demodulación puede ser configurado como 125, 250 o 500 kHz. La tasa de transferencia puede ser configurada como cualquier tasa disponible dentro de LoRa (SF7 a SF12) pero, contrariamente a los canales IF0 a IF7, solo la tasa de datos configurada será demodulada. Este canal tiene el propósito de servir como un enlace de alta velocidad con otros gateways o equipos dentro de la infraestructura de la red. Este canal es compatible con la señal transmitida por la familia de chip SX1272 y SX1276.

### **Canal GFSK IF9**

El canal IF9 está conectado a un demodulador GFSK. El ancho de banda y la tasa de datos del canal pueden ser modificados. Este demodulador ofrece un alto nivel de configurabilidad sobre el cual no vamos a entrar en detalles ya que no va a ser utilizado, queda reservado para futuras aplicaciones. Este canal de demodulación puede demodular cualquier señal con formato FSK o GFSK.

### **Canales LoRa IF0 a IF7**

Estos canales están conectados al otro chip SX1257. El ancho de banda del canal es de 125 kHz y no puede ser configurado o modificado. Cada frecuencia de canal IF puede ser configurada individualmente. En cada uno de estos canales se puede recibir cualquier tasa de datos sin configuración previa.

Muchos paquetes con diferente tasa de datos (diferentes SF) pueden ser demodulados simultáneamente incluso en el mismo canal. Estos canales tienen el propósito de ser usados por un masivo número de nodos (10.000) en una red asincrónica en estrella. Cada nodo puede usar un canal aleatorio (entre IF0 y IF7) y una tasa de datos distinta para cualquier transmisión.

Los sensores ubicados cerca del Gateway usarán típicamente la mayor tasa de datos posible en el ancho de banda fijo del canal de 125 kHz (por ej. 6 kbit/s) mientras los sensores ubicados lo más lejos posible del Gateway usarán la menor tasa de datos posible, de hasta 300 bit/s (menor tasa de datos LoRa en un canal de 125 kHz).

El chip SX1301 escanea los 8 canales (IF0 a IF7) en busca de preámbulos de todas las tasas de datos, todo el tiempo. Es capaz de demodular simultáneamente hasta 8 paquetes. Cualquier combinación de SF y frecuencia intermedia de hasta 8 paquetes es posible (por ej. un paquete SF7 en IF0, uno SF12 en IF7 y un SF9 en IF1 simultáneamente). Es decir, puede detectar simultáneamente preámbulos correspondientes a todos los SF en todos los canales (IF0 a IF7). Sin embargo, no puede demodular más de 8 paquetes a la vez. Esto se debe a que la arquitectura del SX1301 separa la detección del preámbulo y la adquisición de la señal del proceso de demodulación.

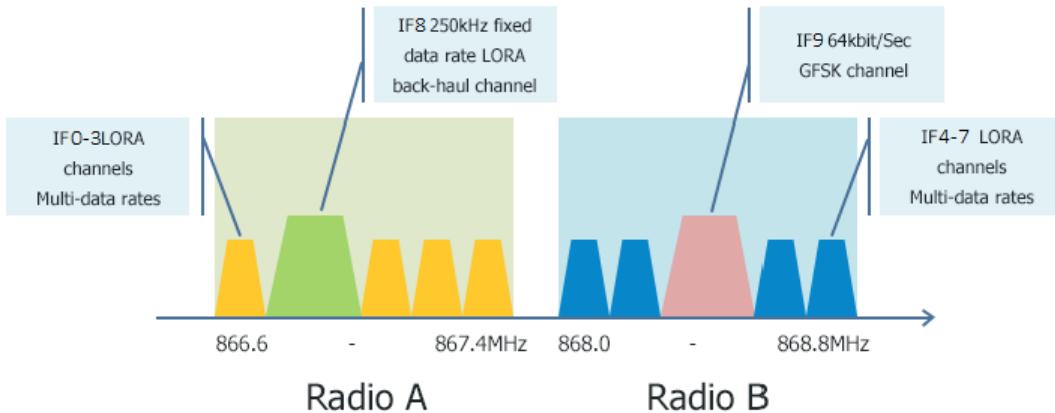


Figura 17: Espectro de los canales del RAK831

### PinOut

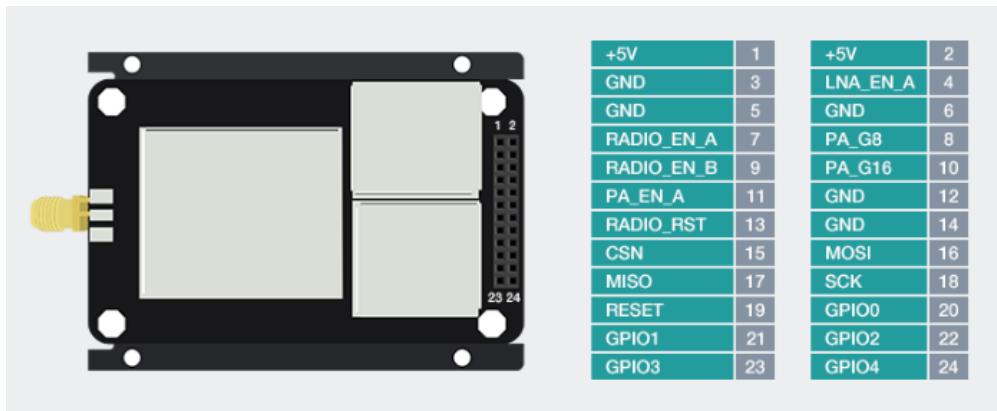


Figura 18: PinOut del concentrador RAK831

### 4.4.3 Servers

#### Introducción

Para realizar el servidor utilizaremos el “LoRa Server Project, licensed under the MIT license” el cual es open-source [8].

El server se encuentra entre los Gateway que reciben los mensajes de los dispositivos finales (nodos) y la aplicación que recibirán los datos, es el responsable de gestionar la red, es el que conoce que nodo/sesión está abierta o cuando un nuevo nodo se une a la red. En esta instancia el server le pregunta al server de aplicación si el nodo está permitido en la red, si es así, se configura todo para empezar a recibir y/o enviar mensajes al nodo.

Para las secciones activas de los nodos, el server es el encargado de des-duplicar la información recibida (la cual es producida por la recepción del mismo mensaje a través de varios gateways), también autentica esta información (esto hace más segura a la red

frente ataques de REPLAY), por último el server reenvía esta información (encriptada) al server de aplicación y luego le preguntara si tiene algún mensaje para el nodo.

El server también es el responsable de gestionar el flujo de datos mediante los llamados MAC-comandos (por ejemplo, el cambio de DR, el canal, etc.), información que se encuentra en la capa MAC de LoRaWan.

Por otro lado, y no menos importante, se encuentra el Server de Aplicación LoRa el cual es un componente compatible con el Server LoRa. Este ofrece una administración a los nodos y gateways mediante la aplicación. También ofrece administración de usuarios y la posibilidad de asignar usuarios a organizaciones y/o aplicaciones. La comunicación con la aplicación usa JSON sobre MQTT y APIs.

El server de aplicación LoRa ofrece además una interfaz web que puede ser usada para la administración de los nodos y gateways.

### **Características del server**

**Bandas ISM:** Lora Server implementa todas las bandas regionales que son definidas en las bandas ISM especificadas por la Alianza LoRaWan. Cada región tiene diferentes frecuencias y regulaciones locales.

**Clases de los dispositivos:** Lora Server implementa LoRaWan Clase-A, Clase-B y Clase-C, haciendo esto ideal para diferentes propósitos.

- Clase A: Los dispositivos finales de clase A permiten comunicaciones bidireccionales por lo cual cada transmisión uplink es seguida por dos ventanas cortas de downlink.

La ranura de transmisión programada por el dispositivo final está basada en la necesidad propia de transmisión, haciéndolo con una pequeña variación de un tiempo aleatorio (ALOHA es el protocolo utilizado para la transmisión).

- Clase B: Los dispositivos finales de clase B permiten recibir más cantidad de ranuras de recepción. Además de las ventanas de recepción aleatorias clase A, los dispositivos clase B tienen ventanas de recepción extras en tiempos programados. Para lograr la correcta recepción de dichas ventanas en un tiempo programado, el nodo recibe una señalización “Beacon” para lograr la sincronización, dicho Beacon es transmitido por el gateway.
- Clase C: Los dispositivos finales clase C tienen una ventana de recepción abierta continuamente que únicamente se cierra cuando el nodo transmite. Los nodos clase C consumen más energía que los nodos clases A o B, pero estos ofrecen una baja latencia entre el server y el nodo.

**Adaptive data-rate (ADR):** Lora Server implementa una velocidad de datos adaptativa. Los dispositivos que soportan ADR podrán operar con el mejor DR posible que se adapte a la aplicación y dando un menor consumo de energía. Esto no solo prolonga la carga de la batería sino que también permite un mejor uso de la red y del espectro de frecuencia.

**Canales reconfigurables:** Lora Server soporta la reconfiguración de canales, esto se hace automáticamente y se podrán deshabilitar los nodos que no se utilizan en la red o moverlos a otro canal en el caso de interferencia en el canal.

**Interfaz Web:** El server ofrece una interfaz web para administrar los nodos mediante la aplicación. También está disponible para administrar o asignar nuevos usuarios de organizaciones y/o aplicaciones. El proyecto LoRa Server es ideal para configurar múltiples equipos o múltiples organizaciones.

**API:** Ambos servidores, Lora Server (el servidor de la red LoRa) y LoRa App Server (el servidor de aplicación) proveen APIs para hacerlos integrables a las infraestructuras propias del usuario. Al usar la API de LoRa Server, se podría implementar un sistema de administración de nodos propio y reemplazar completamente el Servidor de aplicaciones LoRa si fuera necesario.

**Administración de Gateway:** este server ofrece administración de gateways, para administrar sus estados y su ubicación mediante GPS. A su vez se muestran estadísticas para seguir su desempeño.

## **Requisitos**

### **Hardware**

Para realizar la instalación e implementación del server se requiere un hardware que incorpore el sistema operativo Ubuntu o Debian.

Los requisitos mínimos para ejecutar Ubuntu 18.04 LTS son: procesador 700 MHz de 64 bits, 1 GB de memoria RAM, 10 GB de disco duro, lectora de DVD o puerto USB para la instalación., lector de DVD o puerto USB y conexión a internet.

Los requisitos ideales son: procesador 1 GHz x64 en adelante, 2GB de memoria RAM en adelante, 20 GB de disco duro, lectora de DVD o puerto USB para la instalación.

En base a estos requisitos consideramos que es necesario correr el servidor en una PC por cuestiones de conveniencia y costos, es decir, es mucho más accesible una PC con ese tipo de características que una Raspberry Pi por ejemplo.

## **MQTT Broker**

El servidor LoRa utiliza MQTT (Message Queuing Telemetry Transport) para el envío y la recepción de los payloads de la aplicación. Es un protocolo de mensajes estándar ISO que trabaja encima de la capa TCP/IP. Está diseñado para conexiones con ubicaciones remotas donde el ancho de banda es limitado, lo que lo hace ideal para IoT.

Para utilizar este protocolo es necesario un broker, el cual es responsable por el manejo de los mensajes de todos los clientes. Este server opensource puede trabajar con cualquier broker MQTT. Analicemos las opciones disponibles. En la siguiente figura vemos una comparación entre las diferentes implementaciones de MQTT.

Name	Developed by	Language	Type	First release date	Last release	Last release date	License
Adafruit IO	Adafruit	Ruby on Rails, Node.js <sup>[31]</sup>	Client	?	2.0.0 <sup>[32]</sup>	?	?
M2Mqtt	eclipse	C#	Client	2017-05-20	4.3.0.0 <sup>[33]</sup>	2017-05-20	Eclipse Public License 1.0
Machine Head	ClojureWerx Team	Clojure	Client	2013-11-03	1.0.0 <sup>[34]</sup>	2017-03-05	Creative Commons Attribution 3.0 Unported License
moquette	Selva, Andrea	Java	Broker	2015-07-08	0.10 <sup>[35]</sup>	2017-06-30	Apache License 2.0
Mosquitto	eclipse	C, Python	Broker and client	2009-12-03	1.4.15 <sup>[36]</sup>	2018-02-27	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD)
Paho MQTT	eclipse	C, C++, Java, Javascript, Python, Go	Client	2014-05-02	1.3.0 <sup>[37]</sup>	2017-06-28	Eclipse Public License 1.0, Eclipse Distribution License 1.0 (BSD) <sup>[38]</sup>
SharkMQTT	Real Time Logic	C	Client	2015-11-06	1.5 <sup>[39]</sup>	2017-10-08	Proprietary License
VerneMQ	VerneMQ	Erlang/OTP	Broker		1.3.2 <sup>[40]</sup>	2018-05-11	Apache License 2.0
wolfMQTT	wolfSSL	C	Client	2015-11-06	0.14 <sup>[41]</sup>	2017-11-22	GNU Public License, version 2
MQTTRoute	Bevywise Networks	C, Python	Broker	2017-04-25	1.0 <sup>[42]</sup>	2017-12-19	Proprietary License <sup>[43]</sup>
HiveMQ	dc-square GmbH	Java	Broker	2013-03-26	3.4.0 <sup>[44]</sup>	2017-05-09	Proprietary License
SwiftMQ	IIT Software GmbH	Java	Broker	2000-07-01	11.1.0 <sup>[45]</sup>	2018-06-06	Proprietary License

Figura 19: Comparación de implementaciones de MQTT

Si bien existen varias opciones, se adopta trabajar con Mosquito debido a que es opensource y es muy popular, esto significa que existe una gran comunidad detrás que puede brindar soporte en el caso de una eventualidad.

## **Base de datos**

Este servidor distingue entre dos tipos de datos para los cuales utiliza dos bases de datos diferentes.

Para los datos persistentes como las organizaciones, las aplicaciones, los nodos o los gateways, utiliza PostgreSQL 9.5+ y para el almacenamiento de datos no persistentes como por ejemplo el estado de los dispositivos que están activos se utiliza Redis 2.6+.

## 4.5 Topología final de la red

La Figura 20 muestra la topología final de nuestra red LoRa con el hardware y server que hemos seleccionado. Una vez obtenido esto la incorporación de nuevos nodos o gateways será solo cuestión de duplicar lo realizado y hacer modificaciones mínimas.

Aclaración: solo contamos con una FRDM-K64F por lo cual podrá funcionar como nodo o como gateway monocanal.

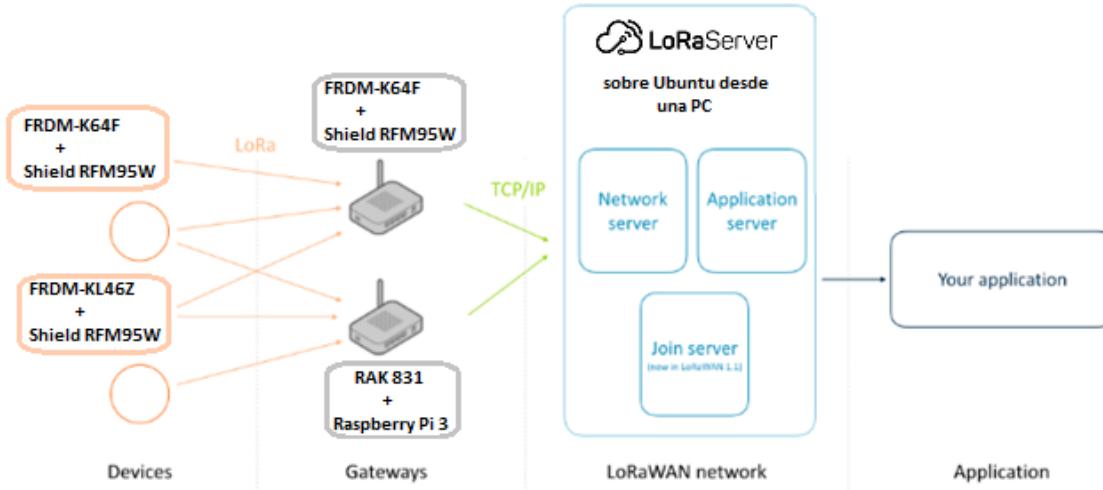


Figura 20: Topología de nuestra red LoRa

## 5 DESARROLLO

### 5.1 Ensayos preliminares

Antes de hacer le desarrollo de la red como se planteó inicialmente se hicieron pruebas para corroborar el funcionamiento de los transeiver RFM95W adquirido.

Esta red preliminar se hizo con:

Nodo: Arduino Uno + RFM95W

Gateway monocanal: Raspberry Pi 3 + RFM95W

Servidor: The Thing Network (TTN), servidor público de LoRaWAN

## Nodo:



Figura 21: Nodo preliminar

La librería implementada en Arduino Uno es LMIC (LoraMAC-in-C) de IBM, dicha librería efectúa la Clase A y tiene activación por ABP y OTAA.

En el programa principal se debe hacer los ajustes de mapeo de pines de acuerdo a la versión del Shield, en nuestro caso es la V1.0.

```
// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
```

Una verificación que se realiza en todos los códigos utilizados es probar el correcto funcionamiento de la comunicación SPI con el módulo LoRa, una manera simple de hacer esto es leyendo el ID del chip y si no se puede conectar, da error de conexión. Este error se suele dar por un mal reinicio del módulo, este caso nos sucedió, el error estaba en la forma que la librería realizaba el reinicio. Los módulos RFM95W se reinician con el pin de Reset en “0”, y luego de un tiempo (5ms aproximadamente) se debe volver a “1”, en cambio los RFM69HCW se reinician de manera inversa, primero se pone en ‘1’ y luego de un tiempo se vuelve a ‘0’. Esto se puede configurar en la función radio\_init () del archivo radio.c de la librería.

Se deben configurar los canales de frecuencia acorde al plan en el archivo principal (.ino).

Para poder recibir el Downlink se debe incorporar en el archivo .ino la función LMIC\_setClockError (MAX\_CLOCK\_ERROR \* 1/100) que ayuda a corregir los desfasaje del clock que pudieran existir entre el servidor y el nodo.

Por ultimo para unir el nodo a la red se deben completar las claves LoRaWAN. Estan deben ser concordantes entre el nodo y el servidor, también deben respetar si son MSB (Most Significant Bit) y LSB (Least Significant Bit).

En cuanto al hardware, no se requiere ninguna conexión en especial ya que éste “Shield” fue diseñado para Arduino y puede funcionar directamente acoplándolos, con el cuidado de ubicar correctamente los jumpers que distinguen entre un Arduino Uno y un Arduino Mega. Además tiene que existir algún tipo de conexión galvánica en los pines destinados idealmente para colocar un jumper con la descripción DIO 1, 2 y 5.

A continuación, en la Figura 22 se aprecia que configuramos el paquete a enviar como un String “Hello TTN”. Además podemos ver la dirección “0x260612EE” que tendrá nuestro nodo que posteriormente observaremos del lado del servidor.

```

Dragino_LoRaShield_AU915 $ 

static const u4_t DEVADDR = 0x260612EE; // <-- Change this address for every node!

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello TTN";
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 30; // [seg]

// Pin mapping
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    ...
};

El Sketch usa 23500 bytes (72%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 887 bytes (43%) de la memoria dinámica, dejando 1161 bytes para las variables locales. El máximo es 2048 bytes.

63                                         Arduino/Genuino Uno en COM18

```

Figura 22: Código Arduino (definición de DEVADDR y paquete a enviar)

### Gateway mono-canal:

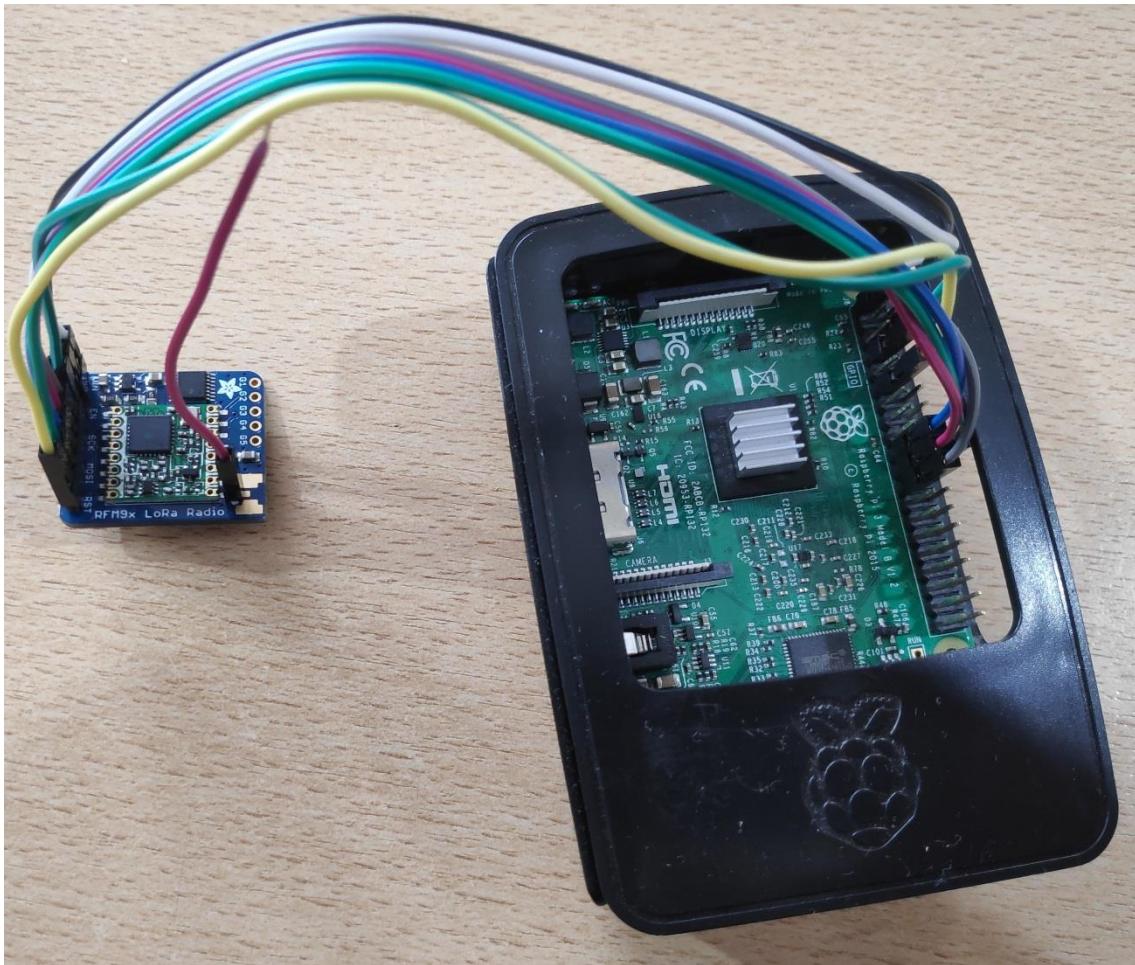


Figura 23: Gateway monocanal

En la Raspberry Pi se clono el gateway single\_chan\_pkt\_fwd de GitHub y se le configuro la frecuencia acorde al plan AU915, la dirección IP del server, el EUI del gateway para que TTN lo identifique y también se modificó el código de Reset igual que en Arduino.

El packet forward depende de la librería WiringPi la cual también se debe instalar.

### Servidor público TTN:

Para acceder al Server únicamente se debe crear una cuenta, luego registrar el Gateway (indicando la EUI del mismo), y finalmente crear una aplicación en la cual luego se agrega el dispositivo o todos los que se deseen crear. De esta forma en la consola del Gateway aparecerán todos los paquetes que llegan al Gateway y en la consola de aplicación aparecen los paquetes de esa aplicación específica con toda la información del paquete y se visualiza el payload indicando a su vez la dirección del dispositivo de dónde provino.

LoRaWAN implementa el Frame Counter Checks que sirve para comparar el contador de paquetes que recibe del gateway y el contador de paquetes que viene indicado por el nodo y comprueba que ambos valores sean iguales de lo contrario se entiende que algún paquete se perdió. Para hacer ensayos hemos desactivado este contador porque si los contadores no son iguales los paquetes llegan a la consola del Gateway pero no a la de Aplicación.

### Prueba de conexión:

Una vez configurada nuestra red procedemos a enviar el paquete y registrar todos sus movimientos. En primer lugar vemos su aparición en la consola del Gateway (Figura 24) donde ya podemos ver que la información se encuentra cifrada.

```
Packet RSSI: -40, RSSI: -97, SNR: 9, Length: 22 Message:'@.&..[D.U.<.&...5//...'  
rxpk update: {"rxpk": [{"tmst":1388533470,"freq":916.8,"chan":0,"rfch":0,"stat":1  
, "modu":"LORA","datr":"SF7BW125","codr":"4/5","rss":-157,"lsnr":9.0,"size":22,  
"data":"QAYmAAFBRKhV7iwAJqABtjUvL4ACAO=="}]}
```

Figura 24: Consola gateway

Siguiendo el camino de paso del paquete, vemos la recepción del mismo en el Gateway (Figura 25), pero ahora visualizado desde el servidor. Vemos que la dirección del dispositivo se condice con la que programamos en el nodo, y además los datos siguen cifrados. Más adelante veremos cómo esto es sumamente importante ya que de parte del Gateway se puede capturar cualquier paquete que esté dentro de su zona de cobertura.

```

 1 {
 2   "pk_id": "eui-b827ebffffed242",
 3   "payload": "Q045B1aaaaAAB1A55jV2/xQeV/n5oAHQ==",
 4   "rssi": -33,
 5   "spreading_factor": 7,
 6   "bandwidth": 125,
 7   "air_time": 56576000
 8 },
 9   "coding_rate": "4/5",
10   "timestamp": "2019-05-26T20:44:26.599Z",
11   "rssi": -33,
12 }
```

Figura 25: Recepción en el servidor (Gateway)

Finalmente nuestro paquete llega al dispositivo final registrado (Figura 26). Vemos que ahora nuestro paquete ya no está cifrado, el servidor nos muestra el payload descifrada y en hexadecimal. Esto es posible gracias a las claves de red y aplicación configuradas tanto en el nodo como en el servidor, posibilitando un cifrado de extremo a extremo.

The screenshot shows the The Things Network Console interface. In the top navigation bar, there are links for Applications, Gateways, Support, and a user profile. The main area displays a list of received data. One entry is highlighted with a red box, showing the following details:

- time:** 17:44:26
- counter:** 0
- port:** 1
- retry:** retry
- payload:** 48 65 6C 6C 6F 20 54 54 4E

Below the payload, under the **Uplink** section, there is a **Payload** field containing the same hex values (48 65 6C 6C 6F 20 54 54 4E), also enclosed in a red box. Further down, there are sections for **Fields** (no fields) and **Metadata**, which contains a JSON object representing the received message's metadata.

Figura 26: Recepción en el servidor (Nodo)

En la Figura 27 vemos qué significa ese payload en hexadecimal, convirtámoslo a String. Efectivamente es el mensaje original programado en el nodo.

This screenshot shows a web-based application for decoding hex payloads. At the top, the hex string **48656C6C6F2054544E** is displayed. Below it, there is a dropdown menu labeled **Character encoding:** with **ASCII** selected. Underneath the encoding dropdown are three buttons: **Convert**, **Reset**, and **Swap**. A green text area at the bottom displays the decoded string: **Hello TTN**.

Figura 27: Traducción del payload (hexadecimal a String)

Como nota adicional tenemos que destacar nuevamente la importancia del cifrado punto a punto debido a la naturaleza de la comunicación entre el nodo y el Gateway, es decir, cualquier receptor que esté sintonizado a la frecuencia de envío del paquete puede recibirla como se indica en la Figura 28. Podemos ver cómo esta información se encuentra totalmente cifrada y no podemos hacer uso de ella.

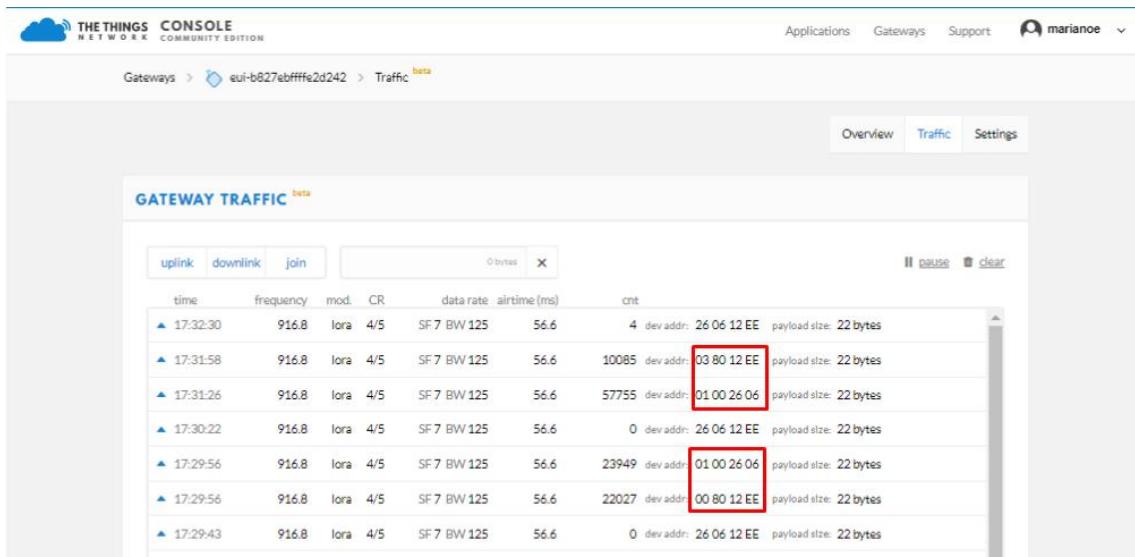


Figura 28: Recepción de paquetes ajenos

En la Figura 29 vemos efectivamente que si bien recibimos un paquete “ajeno”, no podemos ver el contenido de dicho paquete, con lo cual se refuerza el concepto de seguridad de toda esta infraestructura.

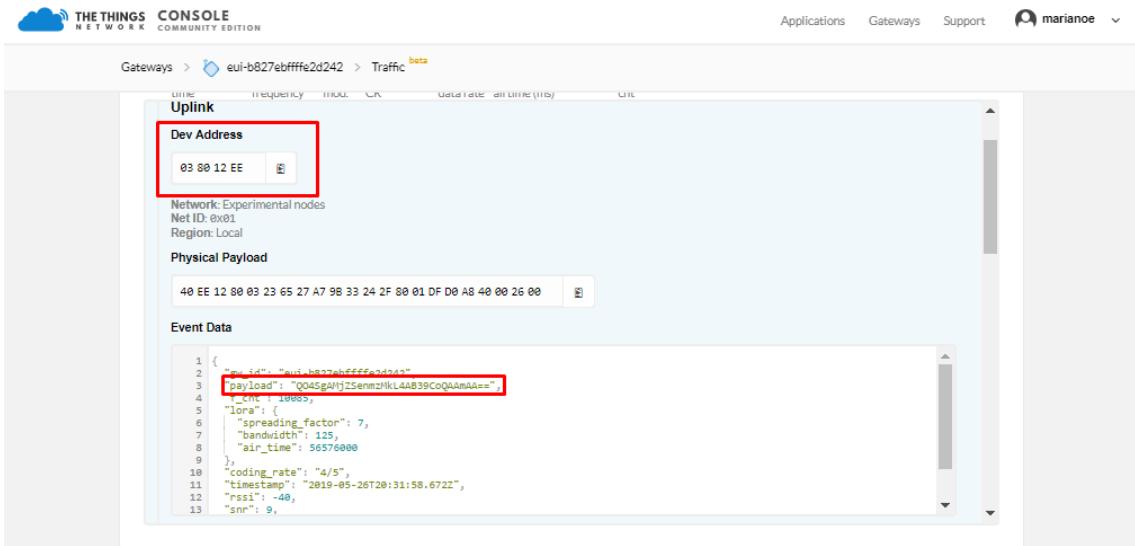


Figura 29: Detalle de un paquete ajeno

## 5.2 Estudio y desarrollo del GATEWAY

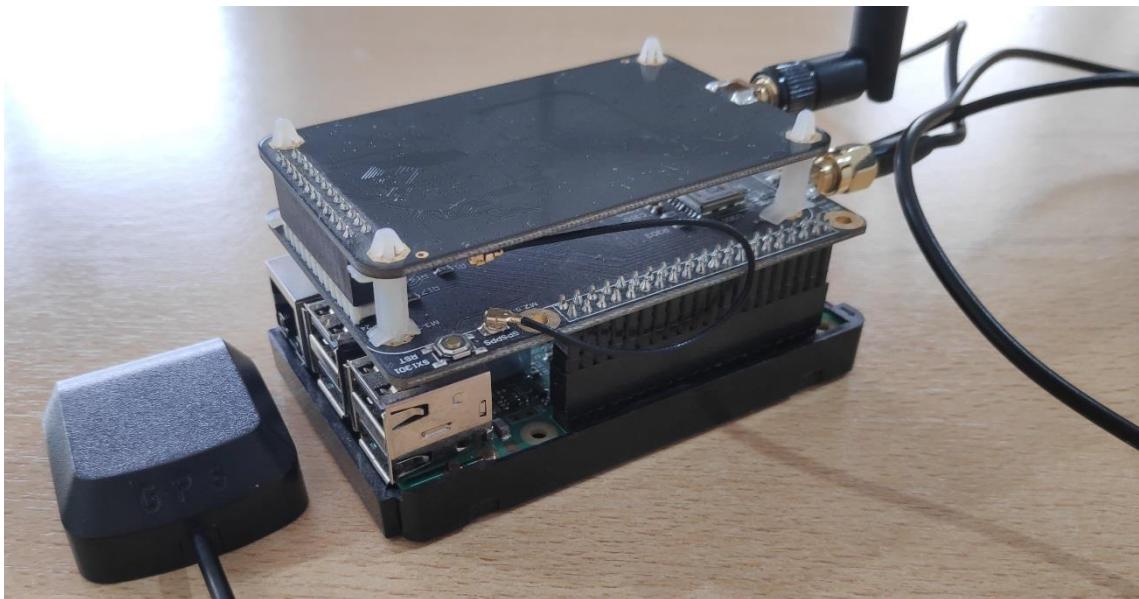
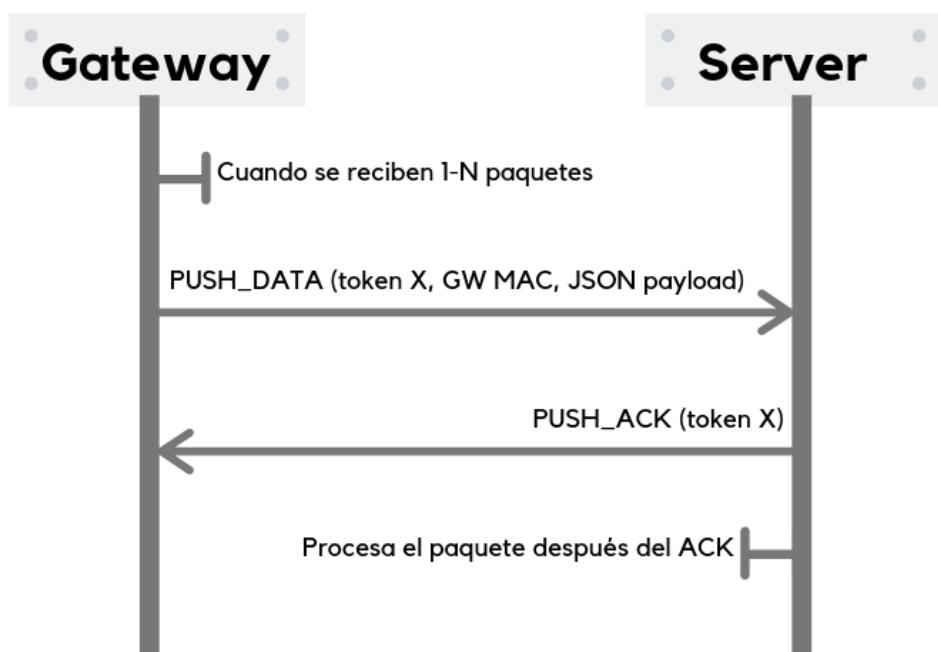


Figura 30: Gateway

### 5.2.1 Protocolo de comunicación del Gateway

#### 5.2.1.1 Protocolo de subida

Diagrama de secuencia



### Paquete PUSH\_DATA

Este paquete (Tabla 2) es usado por el Gateway principalmente para reenviar los paquetes recibidos por radiofrecuencia y sus datos asociados al server.

Bytes	Función
0	Versión del protocolo = 2
1-2	Token aleatorio
3	0x00 Identificador de PUSH_DATA
4-11	Identificador único del Gateway (MAC)
12-Final	Objeto JSON empezando con {, y terminando con }

Tabla 2: Contenido del paquete PUSH\_DATA

### Paquete PUSH\_ACK

Este paquete (Tabla 3) se usa por el server para confirmar inmediatamente todos los paquetes PUSH\_DATA recibidos.

Bytes	Función
0	Versión del protocolo = 2
1-2	Mismo Token que el paquete PUSH_DATA a confirmar
3	0x01 Identificador de PUSH_ACK

Tabla 3: Contenido del paquete PUSH\_ACK

### Estructura de datos de subida JSON

El objeto principal puede contener un array llamado "rxpk":

```

    `` ` json
    {
        "rxpk": [ { ... }, ... ]
    }
    `` .
  
```

Este array (Tabla 4) contiene al menos un objeto JSON, cada objeto contiene un paquete de radiofrecuencia y sus datos asociados según los siguientes campos:

Nombre	Tipo	Función
time	String	Tiempo UTC en que se recibió el paquete
tmms	Number	Tiempo GPS en que se recibió el paquete
tmst	Number	Tiempo interno de un evento finalizado de recepción (32b unsigned)
freq	Number	Frecuencia central de recepción en MHz (unsigned float, precisión en hz)

chan	Number	Canal IF usado por el concentrador en la recepción (unsigned integer)
rfch	Number	Canal RF usado por el concentrador en la recepción (unsigned integer)
stat	Number	Estado del CRC: 1=OK   -1=Fail   0=No CRC
modu	String	Identificador de modulación “LORA” o “FSK”
datr	String	Identificador de Datarate LoRa (ej: SF12BW500)
datr	Number	FSK Datarate (unsigned, en bits por segundo)
codr	String	Identificador de taza de codificación, LoRa ECC
rssi	Number	RSSI in dBm (signed float, 0.1 dB de precisión)
lsnr	Number	Relación señal ruido SNR (signed float, 0.1dB de precisión)
size	Number	Tamaño del payload del paquete RF en bytes (unsigned integer)
Data	String	Payload del paquete RF codificado en Base64

Tabla 4: Contenido del objeto rxpk

Ejemplo (los espacios agregados son para facilitar la lectura):

```

``` json
{
  "rxpk": [
    {
      "time": "2013-03-31T16:21:17.528002Z",
      "tmst": 3512348611,
      "chan": 2,
      "rfch": 0,
      "freq": 866.349812,
      "stat": 1,
      "modu": "LORA",
      "datr": "SF7BW125",
      "codr": "4/6",
      "rss1": -35,
      "lsnr": 5.1,
      "size": 32,
      "data": "-DS4CGaDCdG+48eJNM3Vai-zDpsR71Pn9CPA9uCON84"
    },
    {
      "time": "2013-03-31T16:21:17.530974Z",
      "tmst": 3512348514,
      "chan": 9,
      "rfch": 1,
      "freq": 869.1,
      "stat": 1,
      "modu": "FSK",
      "datr": 50000,
      "rss1": -75,
      "size": 16,
      "data": "VEVTVF9QQUNLRVRfMTIzNA=="
    },
    {
      "time": "2013-03-31T16:21:17.532038Z",
      "tmst": 3316387610,
      "chan": 0,
      "rfch": 0,
      "freq": 863.00981,
      "stat": 1,
      "modu": "LORA",
      "datr": "SF10BW125",
      "codr": "4/7",
      "rss1": -38,
      "lsnr": 5.5,
      "size": 32,
      "data": "ysgRl452xNLep9S1NTIg2lomKDxUgn3DJ7DE+b00Ass"
    }
  ]
}
```

```

El objeto principal puede contener también un objeto llamado "stat":

```
```json
{
    "rxpk": [...],
    "stat":{...}
}
```

Es posible que el paquete no contenga un "rxpk" pero si un objeto "stat" (Tabla 5).

```
```json
{
    "stat":{...}
}
```
```

El objeto que contiene el estado del Gateway puede tener los siguientes campos:

| Nombre | Tipo   | Función                                                              |
|--------|--------|----------------------------------------------------------------------|
| time   | String | Tiempo UTC del sistema que funciona como Gateway                     |
| lati   | Number | Latitud GPS del Gateway en grados (float, Norte es positivo)         |
| long   | Number | Longitud GPS del Gateway en grados (float, Este es positivo)         |
| alti   | Number | Altitud GPS del Gateway en metros (integer)                          |
| rxnb   | Number | Número de paquetes recibidos por radio (unsigned integer)            |
| rxok   | Number | Número de paquetes recibidos por radio con CRC válido                |
| rfw    | Number | Número de paquetes recibidos por radio reenviados (unsigned integer) |
| ackr   | Number | Porcentaje de datagramas de subida confirmados                       |
| dwnb   | Number | Número de datagramas de bajada recibidos(unsigned integer)           |
| txnb   | Number | Número de paquetes emitidos                                          |

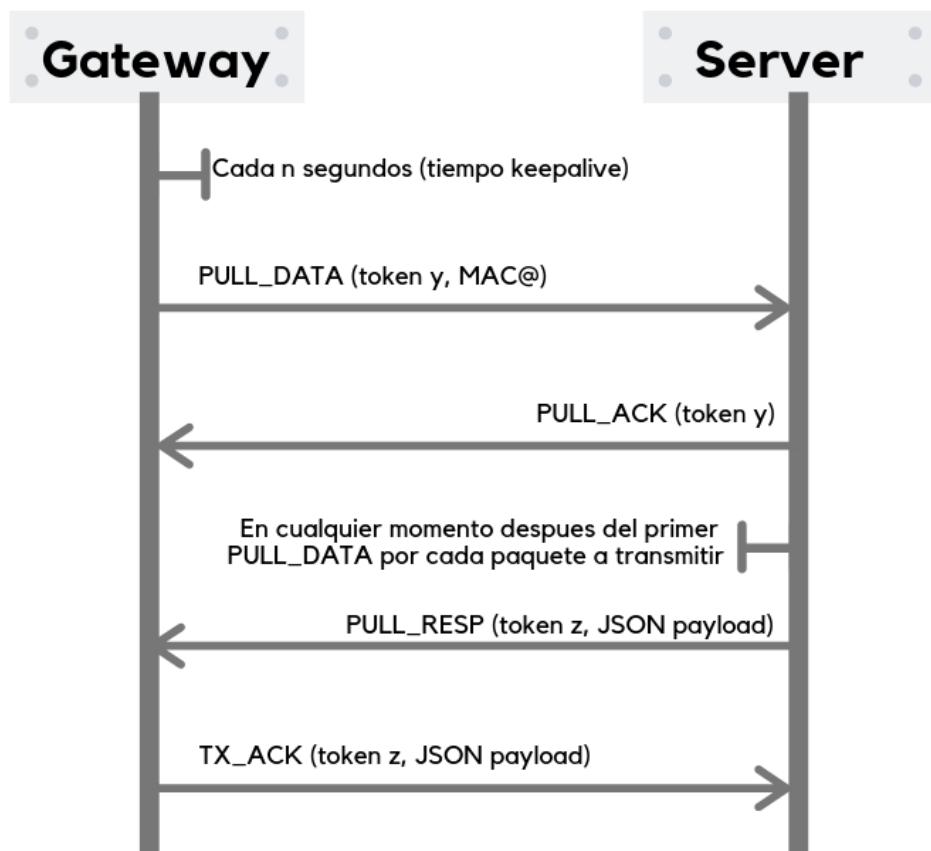
Tabla 5: Contenido del objeto stat

Ejemplo (los espacios agregados son para facilitar la lectura):

```
```json
{
    "stat": {
        "time": "2014-01-12 08:59:28 GMT",
        "lati": 46.24000,
        "long": 3.25230,
        "alti": 145,
        "rxnb": 2,
        "rxok": 2,
        "rfw": 2,
        "ackr": 100.0,
        "dwnb": 2,
        "txnb": 2
    }
}
````
```

### 5.2.1.2 Protocolo de bajada

Diagrama de secuencia



### Paquete PULL DATA

Este paquete (Tabla 6) se usa por el Gateway para obtener datos desde el servidor. El intercambio de datos es inicializado por el Gateway porque puede darse el caso en que el Gateway esté detrás de un NAT, en cuyo caso le es imposible al server comunicarse con el Gateway.

Cuando el Gateway comienza el intercambio, la ruta de red hacia el server se abrirá y permitirá el flujo de paquetes en ambas direcciones. El Gateway debe enviar periódicamente paquetes PULL\_DATA para asegurarse que la ruta permanezca abierta para que el server la pueda usar en cualquier momento dado.

| Bytes | Función                               |
|-------|---------------------------------------|
| 0     | Versión del protocolo = 2             |
| 1-2   | Token aleatorio                       |
| 3     | 0x02 Identificador de PULL_DATA       |
| 4-11  | Identificador único del Gateway (MAC) |

Tabla 6: Contenido del paquete PULL\_DATA

### Paquete PULL ACK

Este paquete (Tabla 7) se usa por el server para confirmar que la ruta de red está abierta y que el server puede enviar paquetes de PULL RESP en cualquier momento.

| Bytes | Función                                          |
|-------|--------------------------------------------------|
| 0     | Versión del protocolo = 2                        |
| 1-2   | Mismo Token que el paquete PULL_DATA a confirmar |
| 3     | 0x02 Identificador de PUSH_ACK                   |

Tabla 7: Contenido del paquete PULL ACK

### Paquete PULL RESP

Este paquete (Tabla 8) se usa por el server para enviar paquetes de radiofrecuencia y datos asociados al paquete que deberán ser reenviados en su correspondiente modulación por el Gateway.

| Bytes   | Función                                         |
|---------|-------------------------------------------------|
| 0       | Versión del protocolo = 2                       |
| 1-2     | Token aleatorio                                 |
| 3       | 0x03 Identificador de PULL_RESP                 |
| 4-Final | Objeto JSON empezando con {, y terminando con } |

Tabla 8: Contenido del paquete PULL\_RESP

### Paquete TX\_ACK

Este paquete (Tabla 9) se usa por el Gateway para enviar una confirmación al server que la petición de bajada fue aceptada o rechazada por el Gateway. El datagrama puede contener adicionalmente un objeto JSON de tipo string para dar más detalles al respecto. Si no se presenta un objeto JSON (string vacío) significa que no ocurrió error alguno.

| Bytes    | Función                                                   |
|----------|-----------------------------------------------------------|
| 0        | Versión del protocolo = 2                                 |
| 1-2      | Mismo token que el paquete PULL_RESP a confirmar          |
| 3        | 0x05 Identificador de TX_ACK                              |
| 4-11     | Identificador único del Gateway (MAC)                     |
| 12-Final | Objeto JSON opcional, empezando con {, y terminando con } |

Tabla 9: Contenido del paquete TX\_ACK

### Estructura de datos JSON de bajada

El objeto principal del paquete PULL\_RESP debe contener un objeto llamado "txpk" (Tabla 10):

```
```json
{
    "txpk": {...}
}
```

Este objeto contiene un paquete RF (de radiofrecuencia) a ser emitido por el Gateway, y sus correspondientes datos asociados, con los siguientes campos:

Nombre	Tipo	Función
imme	Bool	Enviar paquete inmediatamente (ignorará tmms y tmst)
tmms	Number	Enviar paquete en un determinado tiempo de GPS (se requiere sincronización de GPS)
tmst	Number	Enviar paquete en un determinado timestamp (ignora el tiempo)
freq	Number	Frecuencia central de transmisión en MHz (unsigned float, precisión en hz)
rfch	Number	Canal RF usador por el concentrador en la recepción (unsigned integer)
powe	Number	Potencia de transmisión Tx en dBm (unsigned integer, precisión en dBm)
modu	String	Identificador de modulación “LORA” o “FSK”
datr	String	Identificador de Datarate LoRa (ej: SF12BW500)
datr	Number	FSK Datarate (unsigned, en bits por segundo)
fdev	Bool	Desviación de frecuencia FSK (unsigned integer, en hz)
ipol	Number	Inversión de polaridad de la modulación LoRa
prea	Number	Tamaño del preámbulo del paquete RF (unsigned integer)
size	Number	Tamaño del payload del paquete RF en bytes (unsigned integer)
data	String	Payload del paquete RF codificado en Base64
ncrc	bool	True: deshabilita el CRC de la capa física (opcional)

Tabla 10: Contenido del objeto txpk

La mayoría de los campos son opcionales. Si un campo es omitido, tomará un valor por defecto.

Ejemplo (los espacios agregados son para facilitar la lectura):

```
```json
{
    "txpk": {
        "imme": true,
        "freq": 864.123456,
        "rfch": 0,
        "powe": 14,
        "modu": "LORA",
        "datr": "SF11BW125",
        "codr": "4/6",
        "ipol": false,
        "size": 32,
        "data": "H3P3N2i9qc4yt7rK7ldqoeCVJGBybzPY5h1Dd7P7p8v"
    }
}
```

```

```
json
{
    "txpk": {
        "imme": true,
        "freq": 861.3,
        "rfch": 0,
        "powe": 12,
        "modu": "FSK",
        "datr": 50000,
        "fdev": 3000,
        "size": 32,
        "data": "H3P3N2i9qc4yt7rK7ldqoeCVJGByzPY5h1Dd7P7p8v"
    }
}

```

El objeto principal del paquete TX\_ACK debe contener un objeto llamado "txpk\_ack" (Tabla 11):

```

```
json
{
    "txpk_ack": {...}
}
```

```

Este objeto contiene la información asociada al estado del paquete PULL\_RESP.

| Nombre | Tipo   | Función   |
|--------|--------|---|
| error  | String | Indicador del éxito o tipo de error producido por el pedido de bajada |

Tabla 11: Contenido del objeto txpk\_ack

Los posibles valores del campo “error” son:

| Valor            | Definición  |
|------------------|---|
| NONE             | El paquete se programó para la bajada                                     |
| TOO_LATE         | Rechazado porque era muy tarde para programar la bajada                   |
| TOO_EARLY        | Rechazado porque el timestamp del paquete está muy adelantado             |
| COLLISION_PACKET | Rechazado porque ya había un paquete programado en ese tiempo             |
| COLLISION_BEACON | Rechazado porque ya había un beacon planeado en ese tiempo                |
| TX_FREQ          | Rechazado porque la frecuencia elegida no es soportada                    |
| TC_POWER         | Rechazado porque la potencia elegida no es soportada                      |
| GPS_UNLOCKED     | Rechazado porque el GPS esta desbloqueado y su timestamp no se puede usar |

Tabla 12: Posibles valores del error en el paquete txpk\_ack

Ejemplo (los espacios agregados son para facilitar la lectura):

```
```json
{
    "txpk_ack": {
        "error": "COLLISION_PACKET"
    }
}
````
```

## 5.3 Estudio y desarrollo del SERVIDOR

### 5.3.1 Componentes del Servidor

Analicemos con más detalle la composición interna del servidor utilizado. En el siguiente diagrama se visualiza la estructura interna del mismo:

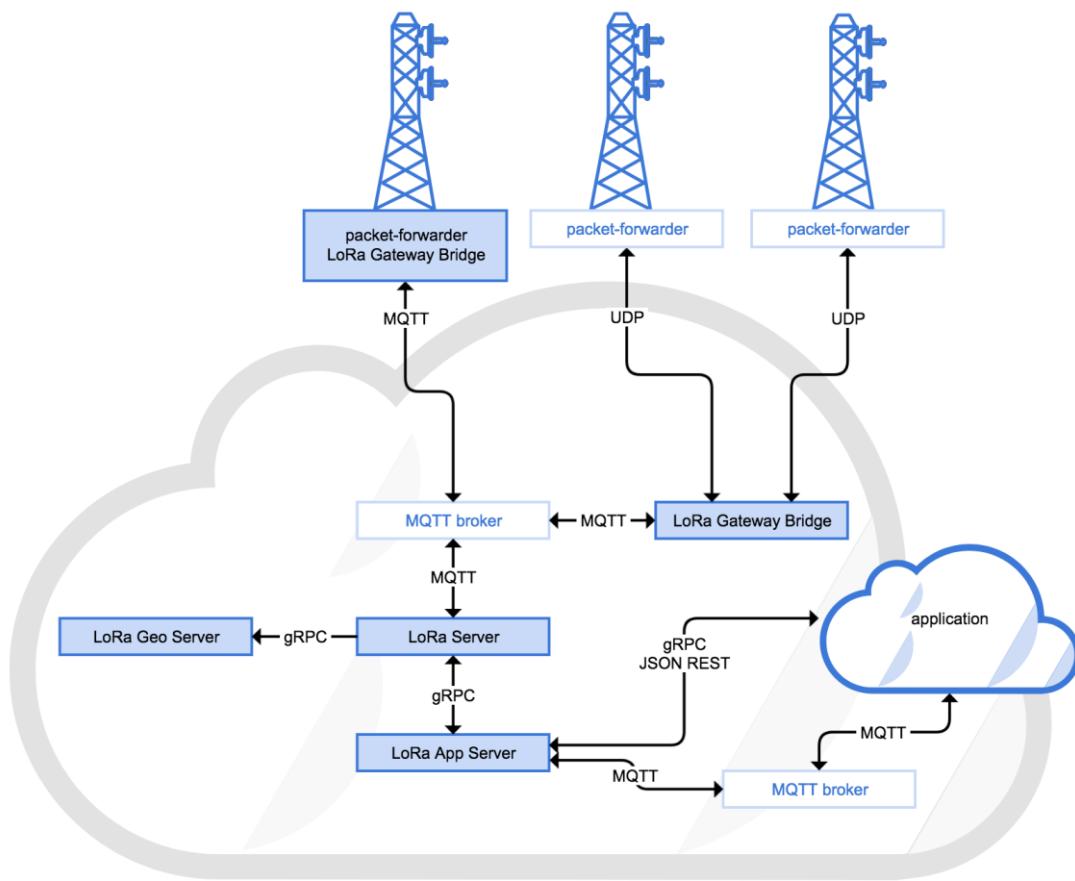


Figura 31: Arquitectura de la red LoRaWAN

En la Figura 31 observar los componentes que posibilitan la comunicación entre el Gateway y la aplicación que hace uso de los datos, mientras que entre las flechas se encuentran los protocolos y modos de comunicación que existe entre las distintas partes. Veamos que función cumple cada una de ellas.

### 5.3.1.1 LoRa Gateway Bridge

Es un servicio que posibilita la abstracción del protocolo UDP utilizado por el Gateway convirtiéndolo en JSON sobre MQTT.

Existen varias maneras de incluir el bridge dentro de la arquitectura del sistema:

- Una sola instancia: la más básica de ellas es conectar todos los gateways a una sola instancia del bridge. Esta es la opción más fácil, ya que instalarlo en cada Gateway puede requerir configuraciones adicionales. Sin embargo, hay que tener en cuenta que desde el punto de vista de la seguridad, no es la mejor opción. El protocolo UDP que implementan la mayoría de los gateways no soporta ningún tipo de autentificación, y tampoco maneja confirmación de transferencias, por lo que en un entorno donde las pérdidas de paquetes son habituales, este tipo de configuración es poco fiable. Pero es sin embargo, la forma más fácil de comenzar.
- Múltiples instancias: por cuestiones de rendimiento y disponibilidad del bridge se lo puede situar en múltiples servidores, pero todos conectados al mismo bróker MQTT. En caso de poner un balanceador de cargas frente a todos los bridge instalados hay que asegurarse que cada conexión de cada Gateway esté siempre ruteado a la misma instancia.
- En cada Gateway: dependiendo de las capacidades de los Gateway se puede instalar el bridge en cada uno de ellos, que si bien esto trae aparejado configuraciones adicionales nos brinda una serie de ventajas muy interesantes.

Al usar MQTT (que a su vez utiliza TCP) sobre UDP, se soluciona el problema de la baja fiabilidad ante la pérdida de paquetes, todo esto gracias a la confirmación de transferencias que utiliza TCP.

Es posible también configurar credenciales para cada Gateway permitiendo conocer desde donde llegan los datos y brindando una forma de autentificación.

Además, el protocolo MQTT soporta SSL/TLS, permitiendo establecer una conexión segura entre los gateways y el bróker MQTT, posibilitando que los datos no puedan ser interceptados, y en el caso de que sean interceptados, que no puedan ser interpretados gracias al cifrado.

### 5.3.1.2 LoRa Server

Cumple la función de procesar los paquetes de subida enviados por el Gateway y resolver el caso en que lleguen duplicados (que el mismo paquete llegue a dos gateways diferentes), manejar la capa LoRaWAN y programar las transmisiones de datos de bajada.

Alguna de las funciones extras que tiene este server en particular son:

- Activación: Soporta tanto ABP y OTTA.
- Datarate adaptativo: reduce el tiempo de aire (tiempo que tarda la onda en viajar desde el nodo hasta el gateway) y el consumo de energía.

- Reconfiguración de canales: Los dispositivos se reconfiguran automáticamente para usar el plan de frecuencias configurado.
- Clases de nodos: soporta todos los dispositivos especificados por LoRaWAN (A, B y C).
- Perfil de nodo: define las capacidades del dispositivo y parámetros de inicio que necesita el servidor para conectarse con el dispositivo.
- Estado del nodo: pedidos y reportes periódicos del estado de la batería e intensidad de la conexión.
- Tiempo del nodo: sincroniza el reloj interno del dispositivo con el de la red.
- Gestión de Gateway: estadísticas, ubicación y reconfiguración del plan de frecuencia (canales).
- Perfil de Gateway: Configura el plan de frecuencias.
- Geolocalización: desencripta la marca de tiempo de los Gateway con GPS y resuelve la ubicación del dispositivo usando un servidor de geolocalización.
- Regiones LoRaWAN: soporta todas las regiones especificadas por los parámetros regionales de LoRaWAN.
- Versiones LoRaWAN: el servidor soporta simultáneamente dispositivos con versiones del protocolo LoRaWAN 1.0 y 1.1.
- Multicast: también es posible enviar tramas de bajada a un grupo de dispositivos a la vez.
- Reconfiguración de parámetros de recepción: reconfigura automáticamente los dispositivos para que usen los parámetros de recepción configurados.
- Configuración de reconexión: pedido periódico de reconexión para dispositivos LoRaWAN 1.1 para actualizar el contexto de seguridad.
- Perfil de rutas: define a qué servidor de aplicaciones se deben reenviar los datos de los nodos.
- Perfil de servicio: define qué características del servidor de red están habilitadas para cada dispositivo.

### 5.3.1.3 Servidor de aplicación LoRa

Dentro de la infraestructura LoRaWAN es el responsable del “inventario” de dispositivos, manejo de los pedidos de conexión y manejo y encriptación de los datos de aplicación.

Ofrece una interfaz web donde se pueden gestionar los dispositivos, los usuarios, las aplicaciones y las organizaciones.

Para integrarse con el exterior ofrece APIs de RESTful y gRPC. Además puede recibir o enviar datos por medio de MQTT o HTTP y escribirlos directamente en InfluxDB.

## Usando el servidor de aplicaciones LoRa

### **Gestión de aplicaciones**

Una aplicación es una colección de dispositivos con el mismo objetivo o del mismo tipo. Pensemos por ejemplo en una estación del clima que obtiene información desde distintos lugares.

Cuando creamos una aplicación tenemos que seleccionar el perfil de servicio que utilizaran los dispositivos creados para dicha aplicación. Tener en cuenta que una vez seleccionado el perfil de servicio que vamos a utilizar no se puede cambiar.

Podemos configurar una aplicación para que interprete los datos recibidos del gateway y los decodifique en datos significativamente útiles (de bytes a un objeto de datos) o que codifique los datos de bajada para el Gateway (de un objeto de datos a bytes). Los datos en “crudo” en base64 siempre estarán disponibles, tanto si existiera una aplicación configurada para realizar tales tareas o no.

Existen dos posibles aplicaciones a utilizar para la codificación y decodificación de los datos.

#### **Función códec JavaScript**

Cuando seleccionamos esta opción nos permite escribir nuestras propias funciones para codifica y decodificar los datos de un array de bytes a un objeto JavaScript.

### **Ejemplo decodificador**

```
// Decodifica un array de bytes en un objeto.  
// - fPort número de puerto LoRaWAN  
// - bytes array de bytes a decodificar  
// La función debe devolver un objeto, por ej:  
{"temperatura": 22.5}  
  
function Decode(fPort, bytes) {  
  
    return {};  
}
```

## Ejemplo codificador

```
// Codifica un objeto en un array de bytes  
// - fPort número de puerto LoRaWAN  
- obj el objeto a codificar, por ej: {"temperature":  
22.5}  
  
// La función debe devolver un array de bytes  
function Encode(fPort, obj) {  
  
    return [];  
}
```

## **Cayenne LPP**

Se puede seleccionar el códec de Cayenne LPP para que el servidor codifique y decodifique usando la especificación de *Cayenne Low Power Payload*, la cual provee una manera fácil y conveniente de enviar datos a través de redes LPWAN tales como LoRaWAN, además cumple con las restricciones de tamaño de tramas y puede inclusive reducirse hasta 11 bytes, permitiendo al nodo enviar datos de múltiples sensores simultáneamente en un mismo paquete.

Adicionalmente, *Cayenne LPP* permite al nodo enviar diferentes datos de diferentes sensores en distintos paquetes, para que esto sea posible cada dato del sensor debe ser prefijado con dos bytes:

- Canal de datos: identifica únicamente a cada sensor en el dispositivo. Por ej.: sensor interior.
- Tipo de datos: identifica el tipo de dato en el paquete. Pro ej.: temperatura.

### Estructura de datos de subida

| 1 Byte         | 1 Byte        | N Bytes | 1 Byte         | 1 Byte        | M Bytes | ... |
|----------------|---------------|---------|----------------|---------------|---------|-----|
| Data1<br>Canal | Data1<br>Tipo | Data1   | Data2<br>Canal | Data2<br>Tipo | Data2   | ... |

Tabla 13: Descripción del paquete de subida

### Estructura de datos de bajada

Un mensaje de bajada (Tabla 14) puede enviarse desde el dashboard de Cayenne hacia el dispositivo, con el propósito de que realice alguna acción. Antes de que el usuario pueda mandar un comando de bajada, primero debe informar a Cayenne el estado actual

del actuador, esto lo debe hacer enviando un mensaje de subida usando los tipos de dato de salida *Analógico* o *Digital*. Luego de recibir el mensaje de subida, Cayenne mostrará un actuador (botón o slider) dependiendo del tipo de dato (digital o analógico). Cuando el usuario envíe el mensaje tocando el botón o modificando el slider Cayenne codificará el mensaje usando la siguiente estructura:

| <b>1 Byte</b> | <b>2 Byte</b>           | <b>1 Bytes</b> |
|---------------|-------------------------|----------------|
| Data<br>Canal | Data (precisión<br>0.1) | 0xff           |

Tabla 14: Descripción del paquete de bajada

El mensaje se enrutará luego hacia el servidor de red donde quedará encolado hasta el siguiente mensaje de subida (solo en el nodo clase A). Una vez extraído de la cola, el dispositivo aceptará el mensaje y ejecutará la acción correspondiente. En este momento el dispositivo deberá enviar otro mensaje de subida actualizando el valor actual del actuador.

### Tipos de datos

Los tipos de datos están definidos según la *IPSO Alliance Smart Objects Guidelines*, las cuales identifican cada tipo de dato con una ID de objeto, sin embargo según se muestra a continuación, se realiza una conversión para poder adaptar el ID a un solo byte.

LPP\_DATA\_TYPE=IPSO\_OBJECT\_ID-3200

Según la Tabla 15, cada tipo de datos puede usar uno o más bytes para enviar datos:

| <b>Tipo</b>           | <b>IPSO</b> | <b>LPP</b> | <b>HEX</b> | <b>Tamaño</b> | <b>Resolución por bit</b>  |
|-----------------------|-------------|------------|------------|---------------|----------------------------|
| Digital Input         | 3200        | 0          | 0          | 1             | 1                          |
| Digital Output        | 3201        | 1          | 1          | 1             | 1                          |
| Analog Input          | 3202        | 2          | 2          | 2             | 0.01 signed                |
| Analog Output         | 3203        | 3          | 3          | 2             | 0.01 signed                |
| Sensor de luz         | 3301        | 101        | 65         | 2             | 1 Lux unsigned MSB         |
| Sensor de proximidad  | 3302        | 102        | 66         | 1             | 1                          |
| Sensor de temperatura | 3303        | 103        | 67         | 2             | 0.1 °C Signed MSB          |
| Sensor de humedad     | 3304        | 104        | 68         | 1             | 0.5% signed                |
| Acelerómetro          | 3313        | 113        | 71         | 6             | 0.001 G signed MSB por eje |
| Barómetro             | 3315        | 115        | 73         | 2             | 0.1 Pha unsigned MSB       |

|                 |      |     |    |   |   |
|-----------------|------|-----|----|---|---|
| Girómetro       | 3334 | 134 | 86 | 6 | 0.01 °/s signed MSB por eje                                     |
| GPS (ubicación) | 3336 | 136 | 88 | 9 | Longitud: 0.0001 ° signed MSB<br>Latitud: 0.01 metro signed MSB |

Tabla 15: Tipos de datos soportados por el protocolo Cayenne

## **Gestión de perfiles de dispositivos**

Un perfil de dispositivo define las capacidades y parámetros de inicio del dispositivo que se necesitan para que el servidor de red tenga acceso al servicio de radio. Esta información debe ser provista por el fabricante del dispositivo.

Cuando se crea un perfil de dispositivo, el servidor de aplicaciones LoRa creará el perfil definitivo en el servidor de red, y mantendrá un registro de referencia para saber a qué organización pertenece.

Las siguientes opciones son implementadas por el servidor de aplicaciones y están descriptas en las especificaciones de interfaces de backend LoRaWAN (<https://lora-alliance.org/lorawan-for-developers>):

- **SupportsClassB** Dispositivo final soporta Clase B
- **ClassBTimeout** Máximo retardo para que el dispositivo conteste una petición MAC o confirme un paquete (obligatorio si soporta Clase B)
- **PingSlotPeriod** obligatorio si soporta Clase B
- **PingSlotDR** obligatorio si soporta Clase B
- **PingSlotFreq** obligatorio si soporta Clase B
- **SupportsClassC** Dispositivo final soporta Clase C
- **ClassCTimeout** Máximo retardo para que el dispositivo conteste una petición MAC o confirme un paquete (obligatorio si soporta Clase C)
- **MACVersion** Versión de LoRaWAN soportada por el dispositivo
- **RegParamsRevision** Revisión de los Parámetros Regionales soportada por el dispositivo
- **SupportsJoin** dispositivo final soporta conexión OTAA o no ABP
- **RXDelay1** Retardo RX1 Clase A (obligatorio para ABP)
- **RXDROffset1** RX1 data rate offset (obligatorio para ABP)
- **RXDataRate2** RX2 data rate (obligatorio para ABP)
- **RXFreq2** Canal de frecuencia RX2 (obligatorio para ABP)
- **FactoryPresetFreqs** Lista de frecuencias soportadas por defecto (obligatorio para ABP)
- **MaxEIRP** Máximos EIRP soportados por el dispositivo
- **MaxDutyCycle** Máximo ciclo de trabajo soportado por el dispositivo

- **RFRegion** Nombre de región RF (establecido automáticamente por el servidor LoRa)
- **Supports32bitFCnt** El dispositivo usa 32bit FCnt (obligatorio para LoRaWAN 1.) (siempre en true)

## **Gestión de dispositivos**

Un dispositivo es el nodo final conectado y comunicándose a través de la red LoRaWAN. El servidor de aplicaciones soporta conexiones OTAA y APB, las cuales se configuran seleccionando el perfil de dispositivo adecuado.

Para conexiones OTAA, luego de crear el dispositivo se pueden configurar las claves de aplicación (y de red para dispositivos LoRaWAN 1.1) debajo de la pestaña *Keys* (OTTA). Debajo de *Activation* se verá la activación del dispositivo actual (si fue exitosa).

Para conexiones APB, luego de crear el dispositivo, se lo puede activar por medio de ABT bajo la pestaña *Activation*. Se puede ingresar la dirección del dispositivo, clave de red y aplicación, o generarlos.

## **Log de eventos**

El servidor de aplicaciones posibilita tener un registro de los eventos de envío al bróker MQTT o alguna otra integración configurada. Para usar esta característica se debe ir a la página de detalles del dispositivo y abrir la **Live event logs**. Inmediatamente después de abrir esta página el servidor se suscribirá a los eventos del dispositivo seleccionado. Una vez que se recibe un evento será mostrado sin la necesidad de actualizar la página.

Hay que tener en cuenta que esta característica es solamente para debugging y no debe usarse junto a la aplicación.

## **Log de tramas**

Este servidor posibilita también el registro de tramas enviadas y recibidas por el Gateway o el dispositivo en tiempo real. Para usar esta característica se debe ir a la página de detalle de Gateway o dispositivo y abrir la pestaña **Live LoRaWAN frame logs**. Inmediatamente después de abrir esta página el servidor se suscribirá a los eventos del dispositivo seleccionado. Una vez que se recibe un evento será mostrado sin la necesidad de actualizar la página.

Hay que tener en cuenta que esta característica es solamente para debugging y no debe usarse junto a la aplicación.

## **Gestión de Gateway**

Una organización puede gestionar su propio grupo de gateways. Hay que tener en cuenta que esta característica puede no estar disponible cuando la organización se configura sin soporte de gateway.

Que un Gateway pertenece a una organización determinada no significa que el uso del Gateway esté limitado solamente a la organización. Cada nodo presente en la red será capaz de comunicarse usando dicho Gateway. La organización será responsable sin embargo de manejar los detalles del Gateway (nombre, ubicación) y podrá ver las estadísticas del mismo, características que están basadas en el valor agregado que pueda enviar cada tipo de concentrador en particular. Si las características no son visibles podría ser una señal que el Gateway está mal configurado.

## **Perfil del Gateway**

Cuando se asigna un perfil de Gateway a un concentrador propio que opera como nuestro Gateway el servidor le enviará una actualización de configuración cuando su configuración esté desactualizada. Estos perfiles se pueden configurar entrando en *Network Servers*. Tener en cuenta que cuando un perfil es asignado a un Gateway se debe configurar el *LoRa Gateway Bridge configuration* para que las actualizaciones puedan ser procesadas adecuadamente.

## **Configuración de la placa del Gateway**

Para gateways que implementan la versión 2 del diseño de referencia que es compatible con geolocalización, es posible configurar una o varias placas. Esto permite configurar la ID de la FPGA y clave de descripción AES para cada placa individualmente. Cuando la clave AES se configura, el servidor desencriptará automáticamente el tiempo del paquete una vez que reciba un mensaje de subida desde el Gateway.

## **Grupos de multicast**

Al crear grupos de multicast se puede enviar una sola trama de bajada a un grupo de dispositivos dentro de este grupo de multicast. Todos estos dispositivos comparten la misma dirección de multicas, clave de sesión y contador de paquete. Luego de crear el grupo se pueden asignar dispositivos al mismo, es decir, el dispositivo ya debe estar creado para ser agregado a un grupo. Solo dispositivos que comparten el mismo perfil de servicio que el grupo de multicast pueden ser agregados a éste.

El envío de datos al grupo ocurre usando gRPC y RESTful JSON APIs.

## **Gestión de servidores de red**

El servidor de aplicaciones puede conectarse a una o múltiples instancias de un servidor de red. Los usuarios globales con permisos de administrador pueden agregar

nuevos servidores de red al servidor de aplicaciones. Una vez que un servidor de red es asignado a un perfil de servicios, el servidor de red no se puede borrar sin antes eliminar estas entidades, devolverá un error.

Cuando se crea un nuevo servidor de red, el servidor de aplicaciones creará un perfil de ruta en el servidor de red conteniendo el *hostname:ip* del servidor de aplicaciones. En caso que no se pueda conectar en el *localhost* hay que asegurarse que el *hostname:ip* esté configurado correctamente en la configuración. Este perfil de ruta es actualizado cada vez que se actualiza el servidor de red.

## **Certificados TLS**

Dependiendo de la configuración tanto del servidor de red como el de aplicación hay que ingresar certificados de cliente y CA para permitir la conexión entre ambos servidores.

Por razones de seguridad, el contenido de la clave LS no se muestra cuando se edita o configura el servidor de red.

## **Profiles de Gateway**

Una vez que se crea el servidor de red, es posible configurar en el mismo uno o más perfiles de Gateway (disponible en la pestaña *Gateway-profiles*). Cuando se agrega un Gateway es posible entonces seleccionar uno de estos perfiles para asegurarse que la configuración del Gateway sea compatible con los canales usados por la red. Esto también debe hacerse en la configuración del bridge.

Es importante tener en cuenta que si se cambia el plan de canales del Gateway, no cambiarán así los canales usados por los dispositivos. Para una configuración correcta, los perfiles de Gateway deben soportar al menos los canales usados por los dispositivos. Estos canales deben configurarse en *LoRa Server configuration*.

## **Canales**

En canales habilitados se puede agregar una lista de canales que nos gustaría que use nuestro Gateway, siempre y cuando estén especificados en los parámetros regiones de LoRaWAN.

En canales extra se pueden agregar aquellos canales que están permitidos en nuestra región pero no están especificados por LoRaWAN.

## **Limitaciones de hardware**

Esta característica está limitada a un Gateway de 8 canales actualmente, y asume que los canales pueden distribuirse en 2 radios. Cuando se define el plan de canales hay que tener en mente que los canales entran en el ancho de banda de los dos radios.

El ancho de banda de cada radio depende el ancho de banda asignado a los canales:

- Canal de 500kHz = Ancho de banda del radio 1.1MHz
- Canal de 250kHz = Ancho de banda del radio 1Mhz
- Canal de 125kHz = Ancho de banda del radio 0.925MHz

## **Gestión de organizaciones**

Una organización puede ser usada para permitir a organizaciones o equipos gestionar sus propias aplicaciones y opcionalmente sus propios gateways..

Una organización puede tener

- Perfiles de servicio
- Perfiles de dispositivos
- Gateways cuando se lo permiten
- Aplicaciones
- Usuarios

### **Perfiles de servicio**

Usuarios globales con permisos de administrador pueden gestionar los perfiles de servicios para determinadas organizaciones. Estos perfiles pueden usarse por usuarios administradores de la organización para crear aplicaciones.

### **Perfiles de dispositivos**

Los perfiles de dispositivos pueden crearse por usuarios administrados de las organizaciones y pueden ser asignados cuando se crea un dispositivo.

### **Gateways**

Una organización puede gestionar su propio grupo de gateways. Tener en cuenta que cuando una organización se crea por medio de un administrador global, puede decidir que la misma no posea ningún Gateway. En este caso, la opción relacionada con el Gateway no estará disponible para esa organización.

Como se mencionó anteriormente, que una organización tenga permitido gestionar su propio grupo de Gateway no significa que el acceso esté limitado para este grupo. La conectividad de los gateways será compartida a través de toda la red.

### **Aplicaciones**

Las aplicaciones pueden ser creadas por usuarios administradores de una organización y definen un grupo de dispositivos con el mismo propósito.

## **Usuarios**

Los usuarios pueden asignarse a una organización para otorgarles acceso a la misma. Dentro del contexto de dicha asignación, un usuario puede ser un usuario regular o un administrador.

Un usuario regular puede ver todos los datos, pero no puede realizar ninguna modificación.

Un usuario administrador está autorizado a gestionar los usuarios asignados a la organización y también gestionar los gateways, aplicaciones y nodos de los gateways.

## **Gestión de perfiles de servicio**

El perfil de servicio puede pensarse como un “contrato” entre el usuario y la red. Describe las características que están habilitadas para el usuario de dicho perfil, y la cantidad de mensajes que éste puede enviar a través de la red.

Los siguientes campos descriptos en el *LoRaWAN Backend Interfaces specification* se implementan en el servidor de aplicaciones:

- **ULRate** taza de token bucket filling, incluyendo ACKs (packet/h)
- **ULBucketSize** tamaño de token bucket burst
- **ULRatePolicy** descartar o marcar cuando se excede el ULRate
- **DLRate** taza de token bucket filling, incluyendo ACKs (packet/h)
- **DLBucketSize** tamaño de token bucket burst size
- **DLRatePolicy** descartar o marcar cuando se excede el DLRate
- **AddGWMetadata** GW metadata (RSSI, SNR, GW geoloc., etc.) se añaden al paquete que se envía al servidor de aplicación
- **DevStatusReqFreq** Frecuencia para iniciar una petición de status (petición/día)
- **ReportDevStatusBattery** reportar el estado de la batería del nodo al servidor de aplicación
- **ReportDevStatusmargin** reportar el margen del nodo al servidor de
- **DRMin** Mínima taza de datos permitida. Usada por ADR.
- **DRmax** Máxima taza de datos permitida. Usada por ADR.
- **ChannelMask** máscara de canal. sNS no debe obedecer (i.e., informativo).
- **PRAccounted** Roaming pasivo permitido
- **HRAccounted** Handover pasivo permitido
- **RAAllowed** activación de roaming permitida
- **NwkGeoLoc** Habilitar servicio de geolocalización en la
- **TargetPER** Taza de error del paquete de destinoT
- **MinGWDiversity** Mínimo número de gateways (informativo)

## 5.4 Estudio y desarrollo del NODO

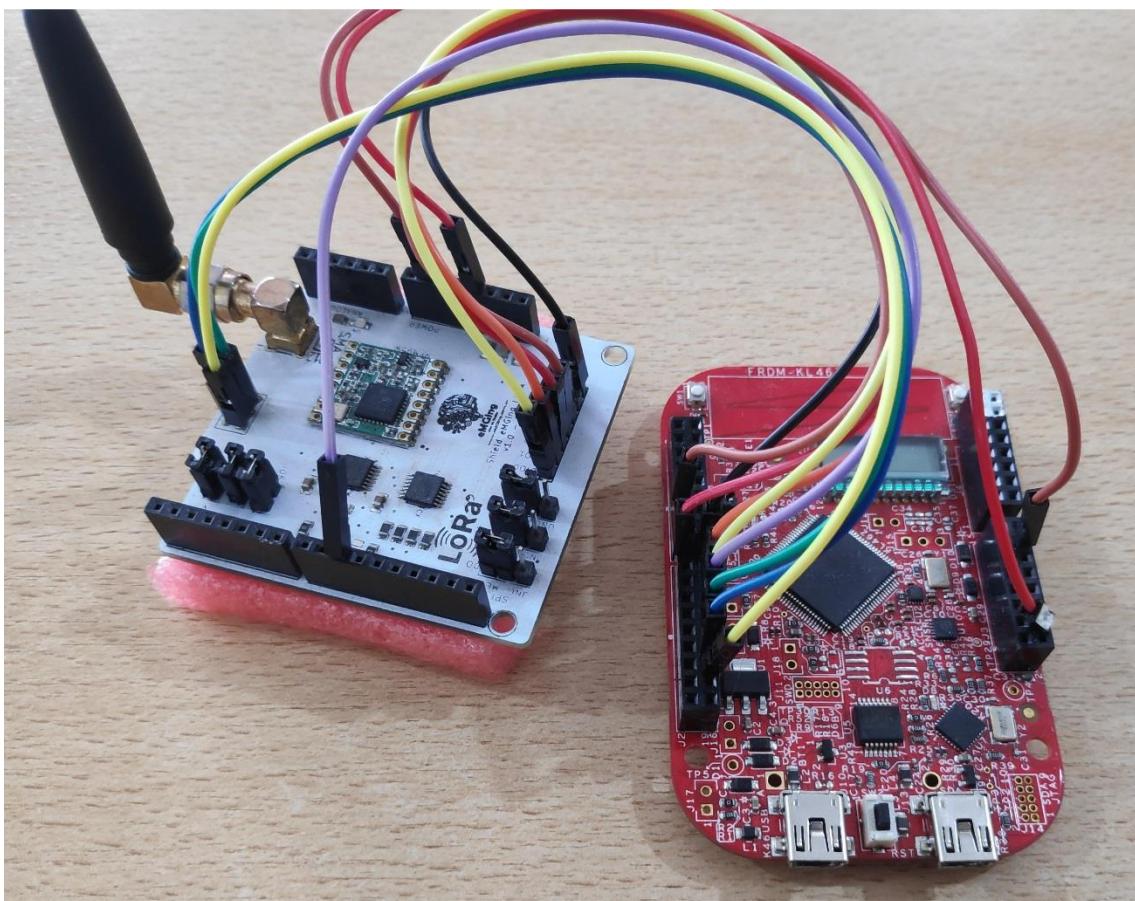


Figura 32: Nodo

**Nota:** Se entiende por Uplink a un mensaje enviado desde el nodo al gateway y un Downlink a un mensaje enviado desde el gateway al nodo.

### 5.4.1 Plan de frecuencia y parámetros

Como se mencionó y justifico en la sección 4.3, el plan utilizado es el AU915-928MHz.

Para este plan, LoRa Alliance™ ha definido los siguientes parámetros regionales de Data Rate.

| DataRate | Configuration        | Indicative physical bit rate [bit/sec] |
|----------|----------------------|--|
| 0        | LoRa: SF12 / 125 kHz | 250                                    |
| 1        | LoRa: SF11 / 125 kHz | 440                                    |
| 2        | LoRa: SF10 / 125 kHz | 980                                    |
| 3        | LoRa: SF9 / 125 kHz  | 1760                                   |
| 4        | LoRa: SF8 / 125 kHz  | 3125                                   |
| 5        | LoRa: SF7 / 125 kHz  | 5470                                   |
| 6        | LoRa: SF8 / 500 kHz  | 12500                                  |
| 7        | RFU                  |  |
| 8        | LoRa: SF12 / 500 kHz | 980                                    |
| 9        | LoRa: SF11 / 500 kHz | 1760                                   |
| 10       | LoRa: SF10 / 500 kHz | 3900                                   |
| 11       | LoRa: SF9 / 500 kHz  | 7000                                   |
| 12       | LoRa: SF8 / 500 kHz  | 12500                                  |
| 13       | LoRa: SF7 / 500 kHz  | 21900                                  |
| 14       | RFU                  |  |
| 15       | Defined in LoRaWAN   |  |

Tabla 16: DataRate para AU915-928MHz

El Data Rate es el parámetro que indica con que Spread Factor y Ancho de Banda se va a realizar la transmisión o recepción.

Cada vez que se envía un paquete Uplink se hace con un Data Rate del 0 al 6 (Tabla 16), dependiendo la velocidad de transmisión que se elija. A su vez el DR0 es el que más alcance tiene. A medida que aumenta el DR va disminuyendo el alcance.

Los DR para usar en el Downlink son del 8 al 13 (Tabla 16) y se elige como una función del Uplink. Esto es por correspondencia, por ejemplo, si un Uplink se realiza con DR0 su Downlink se hará con DR8 o si el Uplink se hizo con DR4 su Downlink se hará con DR12, y así análogamente con todos.

También se hizo mención en la sección 4.3, los canales a utilizar para la comunicación. El plan de frecuencia adoptado tiene 72 canales para el Uplink y 8 de Downlink. En este proyecto se utilizamos únicamente 8 canales de Uplink (916.0 / 917.0 / 917.2 / 917.4 / 917.6 / 917.8 / 918.0 / 918.2 MHz) y 8 de Downlink (923.3 / 923.9 / 924.5 / 925.1 / 925.7 / 926.3 / 926.9 / 927.5 MHz). Cada conjunto de 8 canales se denomina (según la empresa Multitech) Sub-Banda por consiguiente el Uplink se realiza en la 2da Sub-Banda (canales del 8 al 15).

| TXPower | Configuration (EIRP) |
|---------|----------------------|
| 0       | Max EIRP             |
| 1:14    | Max EIRP – 2*TXPower |
| 15      | Defined in LoRaWAN   |

Tabla 17: Tx power AU915

En la Tabla 17 se observa cual es la potencia a utilizar. EIRP (Effective Isotropic Radiated Power) es la potencia equivalente radiada por una antena isotrópica, el máximo EIRP es 30dBm. A medida que se aumenta el valor de TXPower disminuye la potencia utilizada para la transmisión.

#### 5.4.2 Protocolo nodo clase A (All)

Esta clase es la que mejor eficiencia de energía tiene, pero se experimenta una comunicación de Downlink muy limitada. En la Figura 33 se puede observar como es la temporización entre transmisión y recepción.

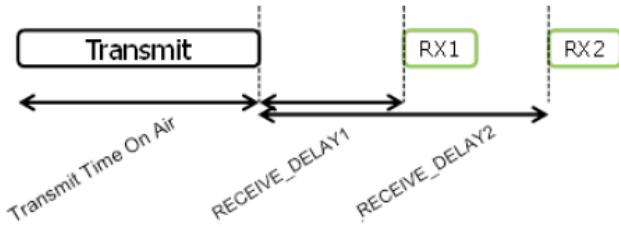


Figura 33: Diagrama de tiempo, TX y RX (Clase A)

Esta clase de nodos trabaja enviando un mensaje Uplink, concluido el Tx se espera un tiempo RECEIVE\_DELAY1, que para nuestro plan AU915 es de 1 segundo. Pasado este tiempo el radio escucha si hay un mensaje Downlink, la duración de esta ventana (Rx1) es equivalente al tiempo que le toma detectar la cantidad de símbolos que hay en el preámbulo de la trama Downlink, longitud de preámbulo, que son 8 símbolos. Si se detectó el preámbulo se procede a la escucha y desmodulación del paquete, de lo contrario la ventana se cierra inmediatamente. Si no se detectó ningún Downlink se espera RECEIVE\_DELAY2 de 2 segundos desde que finalizo Tx. Cuando se abre esta segunda ventana (Rx2) se procede de igual manera que para Rx1.

La elección de los Data Rate y Frecuencias se realizan de la siguiente forma:

Al nodo se le configura 8 canales de frecuencia Uplink, por lo tanto cuando deseé hacer la transmisión elegirá un canal al azar y hará la transmisión por ese canal, si se desea se puede configurar para que siempre envié en el mismo canal. En cuanto al Data Rate es configurado manualmente o si se cuenta con ADR se configura automáticamente mediante el servidor, el servidor también puede indicarle que canal de transmisión utilizar. Para la primera recepción Rx1 la frecuencia es seleccionada únicamente. Esto quiere decir que si se usó el canal 2 del subconjunto de canales para Uplink el Downlink se hará con el canal 2 del subconjunto de canales para Downlink. Pero para la recepción en la segunda ventana Rx2 se escoge el Data Rate de la misma forma que para Rx1, pero la frecuencia es fija y es el primer canal.

Sintetizando, esta clase tiene dos oportunidades de recibir el Downlink pero solo cuando anteriormente envió un Uplink. Esto quiere decir que si el servidor de aplicación quiere enviarle un mensaje al nodo, solo se efectuara cuando el nodo realice un Uplink.

### 5.4.3 Protocolo nodo clase B (Beacon)

Esta clase está optimizada para dispositivos con baterías que podrían ser tanto móviles o con posición fija.

La Clase B es lograda teniendo un gateway o varios, que envían un beacon regularmente para sincronizar todos los nodos de la red. De esta forma el nodo esta sincronizado con la red y puede abrir varias ventanas de recepción extra a un tiempo previsible durante un tiempo de slot periódico.

Para que una red soporte estos nodos, todos los gateways deben hacer broadcast simultáneamente de un beacon, dando una referencia temporal a los nodos. Estos pueden abrir ventanas de recepción periódicamente, a partir de ahora llamadas ping slots, las cuales pueden utilizarse por la infraestructura de red para iniciar una comunicación Downlink. Un Downlink iniciado por la red utilizando unos de esos pings slots se denomina “Ping”.

Todos los nodos comienzan en la red y se unen a ella como dispositivos de Clase A. La aplicación del nodo puede luego decidir hacer un cambio a Clase B. Esto se hace mediante el siguiente procedimiento:

- La aplicación del dispositivo final solicita a la capa LoRaWAN que cambie a Clase B. La capa LoRaWAN en el dispositivo final busca un beacon y devuelve un servicio primitivo BEACON\_LOCKED a la aplicación si es que se ha encontrado y bloqueado uno, o de lo contrario un servicio primitivo BEACON\_NOT\_FOUND.
- Basado en la SNR (Signal to Noise) del beacon y en las restricciones de duración de la batería, la aplicación del dispositivo final selecciona el DR y periodicidad de ping slot.
- Una vez en el modo Clase B el nodo avisa al servidor que el dispositivo cambió a Clase B. La capa MAC programará autónomamente un slot de recepción para cada beacon y para cada ping slot. Cuando la recepción de beacon es exitosa, la capa LoRaWAN del dispositivo final envía el contenido beacon a la aplicación en conjunto con la SNR de la señal de radio medida. Cuando un Downlink es desmodulado correctamente durante un ping slot se procesa similarmente a un Downlink como los descriptos en la Clase A.
- Si ningún beacon ha sido recibido para un dado período, la sincronización con la red se pierde y el nodo vuelve a ser Clase A. La aplicación del dispositivo final puede intentar cambiar nuevamente a Clase B periódicamente. Esto reiniciará este proceso, comenzando con una nueva búsqueda de beacon.

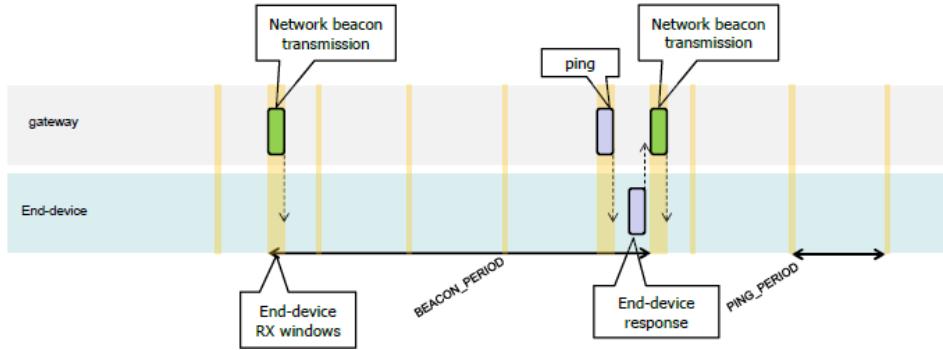


Figura 34: Ranuras de recepción de beacons y ping slots

En la Figura 34 anterior se puede observar la transmisión de dos beacon que realizo el gateway. Para nuestro plan de frecuencia el período de beacon es de 128 segundos, el dispositivo final también abre un ping slot de recepción cada cierto tiempo. La periodicidad del ping slot la decide el nodo, tanto si es fija o la calcula para optimizar el consumo de energía. La periodicidad puede ser de 1 a 128 segundos y se calcula como  $2^k$  con  $k = 0, 1, \dots, 7$ . La mayor parte del tiempo el ping slot no está en uso por el servidor y por lo tanto la ventana de recepción se cierra tan pronto como el transceiver de radio haya decidido que no hay ningún preámbulo presente en el canal de escucha. Si se detecta un preámbulo el transceiver permanecerá encendido hasta que la trama Downlink sea desmodulada.

Para que todos los gateway envíen un beacon al mismo tiempo, los mismos deben tener un GPS de donde obtienen la referencia temporal. El beacon es enviado cada 128 segundos comenzando a las 00:00:00, Domingo 5 – Lunes 6 de Enero de 1980 (comienzo del GPS epoch). El GPS epoch esta expresado en segundos y luego se puede convertir a formato calendario. Por lo tanto si el GPS epoch es divisible por 128 el gateway envía un beacon. De este modo todos los beacon de la red están sincronizados gracias a la referencia temporal del gateway.

La frecuencia de los beacons dependiendo del plan, puede ser fija o variable, en la Tabla 18 se observan los 8 canales de transmisión del beacon y tanto el nodo como el gateway saben que canal se debe usar gracias a la siguiente ecuación que es función del GPS epoch.

$$\text{canal} = \left[ \text{floor} \left( \frac{\text{GPS epoch}}{\text{Periodo del beacon}} \right) \right] \text{modulo } 8$$

| Beacon channel nb | Frequency [MHz] |
|-------------------|-----------------|
| 0                 | 923.3           |
| 1                 | 923.9           |
| 2                 | 924.5           |
| 3                 | 925.1           |
| 4                 | 925.7           |
| 5                 | 926.3           |
| 6                 | 926.9           |
| 7                 | 927.5           |

Tabla 18: Frecuencia del beacon

Así el beacon se transmite por 8 canales, comenzando por el canal 0 hasta el 7 y luego vuelve a comenzar. El beacon utiliza el DR por defecto del plan de frecuencia, en este caso es DR8.

Los pings slot utilizan la misma frecuencia y DR que el ultimo beacon recibido y luego de 128 segundos con la llegada de un nuevo beacon, los pings slot que le preceden tendrán la misma frecuencia y DR que este último.

Este es el mecanismo pseudoaleatorio por defecto, pero luego el servidor mediante un comando MAC puede decirle al nodo en que frecuencia y DR se van a enviar los beacon y pings slot, una vez que el nodo le responde con un ackowlege empieza a trabajar con frecuencia y DR fijo hasta que el servidor le indique algún cambio.

#### 5.4.4 Protocolo nodo clase C (Continuously Listening)

Los dispositivos de Clase C implementan las mismas dos ventanas de recepción que los nodos de Clase A, pero no cierran la ventana RX2 hasta que necesiten enviar nuevamente. Por lo tanto, pueden recibir un Downlink en la ventana RX2 casi en cualquier momento, incluyendo los Downlinks enviados como propósito de comando MAC o transmisión de ACK. También se abre una ventana de recepción corta a frecuencia y DR de RX2, entre el final de la transmisión y el comienzo de la ventana de recepción RX1.

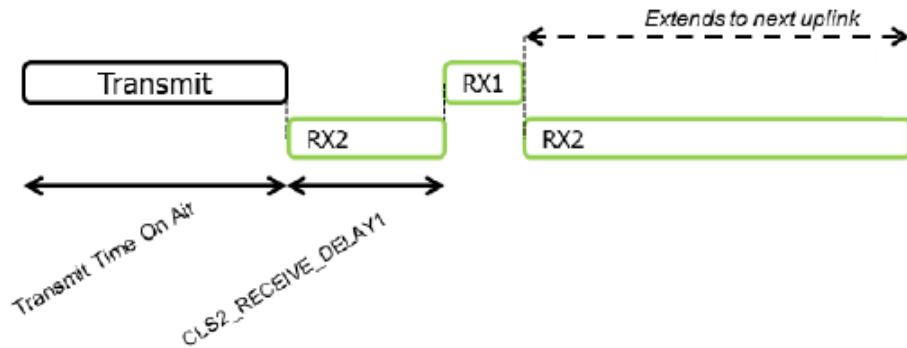


Figura 35: diagrama temporal de Tx y Rx, clase C

#### 5.4.5 Flujo de datos en la red LoRaWAN

La integridad de los datos en la red es asegurada por las llaves: Network Session Key (NwkSKey) y Application Session Key (AppSKey). La NwkSKey es usada para encriptar y desencriptar los datos de la capa MAC, mientras que la AppSKey es usada para encriptar y desencriptar los datos de aplicación. Esto se ve claramente en la Tabla 19.

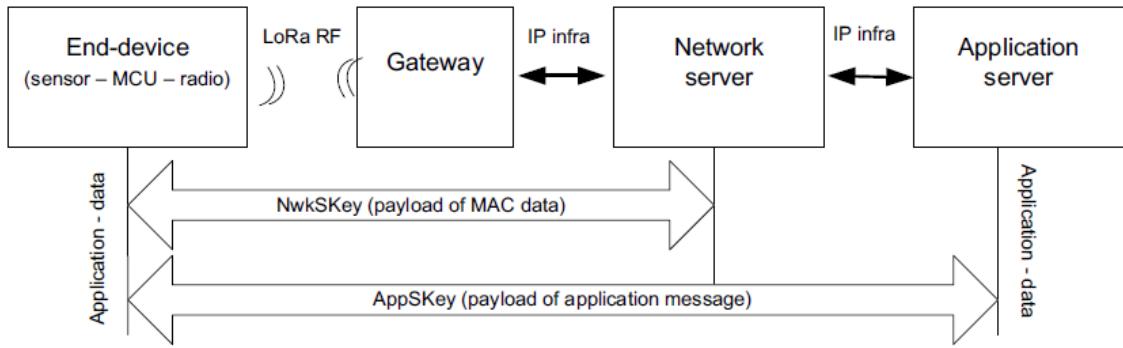


Figura 36: Flujo de dato en una red LoRaWAN

La NwkSKey es compartida entre el nodo y el servidor LoRa mientras que la AppSKey es compartida entre el nodo y el servidor de Aplicación. Esto se hace para incorporar seguridad a la comunicación.

#### 5.4.6 Activación del dispositivo final (Join)

Para que efectivamente un dispositivo final esté unido a la red LoRaWAN, y así llegue la información al servidor de red y luego al de aplicación, primero se debe hacer la activación del nodo (unión a la red). Para este propósito hay dos mecanismos.

- Activation by personalization (ABP):

En este caso la NwkSKey y AppSKey son programadas en el dispositivo final entonces al comenzar el programa el nodo se une inmediatamente a la red y puede comenzar a enviarle datos.

- Over-the-air activation (OTAA):

En este otro caso existe una interacción entre el nodo y el servidor de aplicación. Ambos comparten la misma clave, conocida como AppKey. El nodo comienza el pedido enviando un mensaje de Join Request y el servidor para unirlo a la red le envía un mensaje de Join Accept. Durante el proceso de unión el servidor de aplicación le asigna al nodo la NwkSKey y AppSKey, una vez obtenido esto, el nodo ya es parte de la red y puede comenzar a enviarle datos.

#### 5.4.7 Máscara de canal

La máscara de canal (ChMask) codifica los canales usables para acceso uplink.

Un bit en el campo ChMask seteado en 1 significa que el canal correspondiente puede usarse para una transmisión uplink si el canal permite el DR actualmente utilizado por el dispositivo final. Si, por el contrario, el bit se setea en 0, significa que el canal correspondiente debe ser evitado.

#### 5.4.8 Mensajes confirmados

Para ciertas aplicaciones puede ser necesario tener la garantía de que el mensaje llegó a destino. Es por ello que tanto un mensaje Uplink como Downlink pueden pedir un ACK para confirmar el arribo del mensaje.

#### 5.4.9 Mensaje Uplink confirmado

El diagrama de la Figura 37 ilustra los pasos seguidos por un dispositivo final tratando de transmitir dos tramas de datos confirmados (Data0 y Data1):

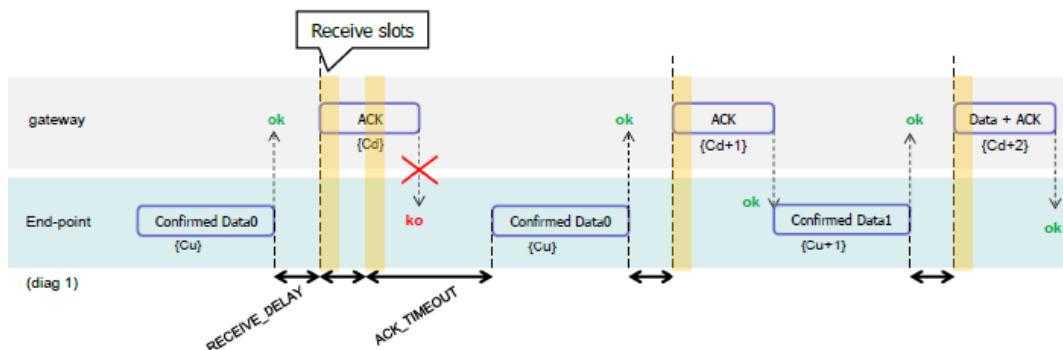


Figura 37: Diagrama temporal de uplinks confirmados

El nodo primero transmite una trama de datos confirmados contenido el payload de Data0. El contador de tramas Cu se deriva simplemente agregando 1 al contador de tramas Uplink previo. La red recibe la trama y genera una trama Downlink con el bit ACK seteado exactamente RECEIVE\_DELAY1 segundos después, utilizando la primera ventana de recepción del dispositivo final. El contador de tramas uplink Cd también es derivado agregando un 1 al último Downlink de ese dispositivo final. Si no hay payload Downlink pendiente, la red debe generar una trama sin payload. En este ejemplo, la trama que lleva el bit ACK no es recibida.

Si un dispositivo final no recibe una trama con el bit ACK seteado en una de las dos ventanas de recepción inmediatamente después de la transmisión del Uplink, debe reenviar la misma trama con el mismo payload y contador de tramas nuevamente, al menos ACK\_TIMEOUT segundos después de la segunda ventana de recepción. Si esta vez el dispositivo final recibe el Downlink ACK durante su primera ventana de recepción,

tan pronto como la trama ACK sea desmodulada, el dispositivo final queda libre para transmitir una nueva trama en un nuevo canal.

La tercera trama ACK en este ejemplo también lleva un payload de aplicación. Una trama Downlink puede llevar cualquier combinación de ACK, controles MAC y payload.

#### 5.4.10 Mensaje downlink confirmado

El diagrama de la Figura 38 ilustra la secuencia básica de un Downlink “confirmado”.

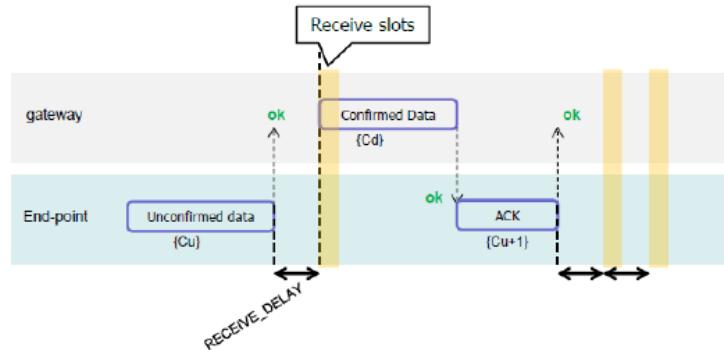


Figura 38: Diagrama temporal de downlinks confirmados

El intercambio de tramas es iniciado por el dispositivo final al transmitir un payload de aplicación “no confirmado” o cualquier otra trama en un canal. La red utiliza la ventana de recepción Downlink para transmitir una trama de datos “confirmados” hacia el dispositivo final en el mismo canal. Hasta la recepción de esta trama de datos que requiere un ACK, el dispositivo final transmite una trama con el bit ACK seteado a su propia discreción. Esta trama puede contener también datos o comandos MAC como payload. Este Uplink ACK es tratado como cualquier Uplink standard, y como tal se transmite en un canal aleatorio que podría ser distinto del canal inicial.

#### 5.4.11 ADR

El Data Rate Adaptativo es un mecanismo para optimizar el Data Rate, airtime y energía en la red. El ADR se debe activar cuando el nodo tiene condiciones de RF estables. El nodo decide si el ADR debe ser usado o no.

Para determinar el DR más óptimo [9], la red necesita algunas mediciones, las cuales toma de los mensajes Uplinks, cuando el nodo envía el bit de ADR seteado. Esta medición es la relación señal a ruido SNR, tomada del gateway que dio el mejor SNR. Luego el server calcula “el margen”, el cual es el mínimo SNR requerido para desmodular un mensaje. Este margen es usado para determinar cuánto se puede incrementar el DR o disminuir la potencia de transmisión.

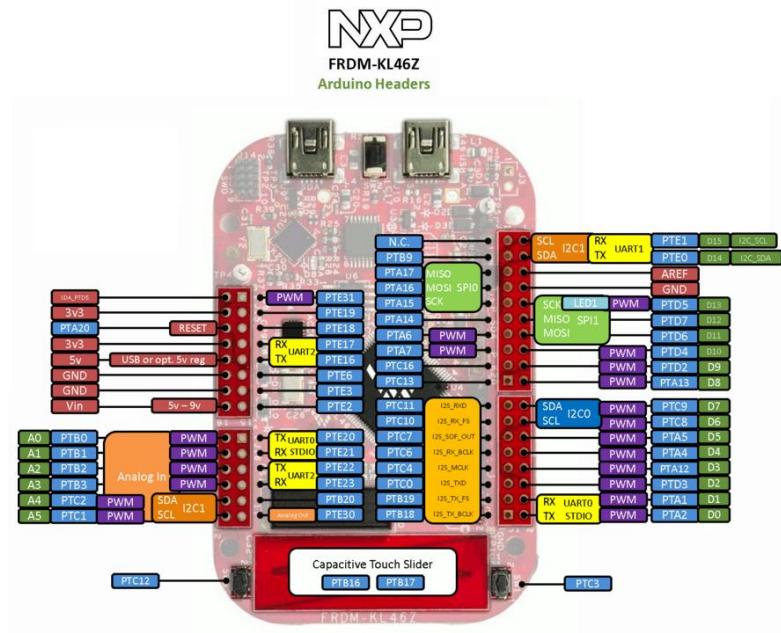
Por defecto el DR inicial para hacer la transmisión del primer Uplink es el más chico, DR0.

Hay varios momentos en los que el ADR puede ser pedido:

- Un pedido de ADR inicial es enviado inmediatamente luego de que el nodo se unió a la red. Este se hace con la medición de un solo Uplink, lo cual puede dar un DR no óptimo.
- Un pedido de ADR regular es programado cuando se miden varios SNR y se concluye que el DR no es el óptimo y se debe cambiar. Sirve para corregir el ADR inicial.
- Un pedido de ADR es enviado cuando hay suficientes mediciones y el dispositivo está usando DR0 sin necesidad.

#### 5.4.12 Hardware

El nodo está constituido por dos partes fundamentales, el *host* que en nuestro caso es la placa de desarrollo FRDM KL46Z y el *RF transceiver LoRa*, en nuestro caso es el shield RFM95W que es compatible con Arduino Uno y Mega. En la Figura 39 se observa el conexionado de ambas placas.



PinOut Shield eMGing LoRa 915Mhz

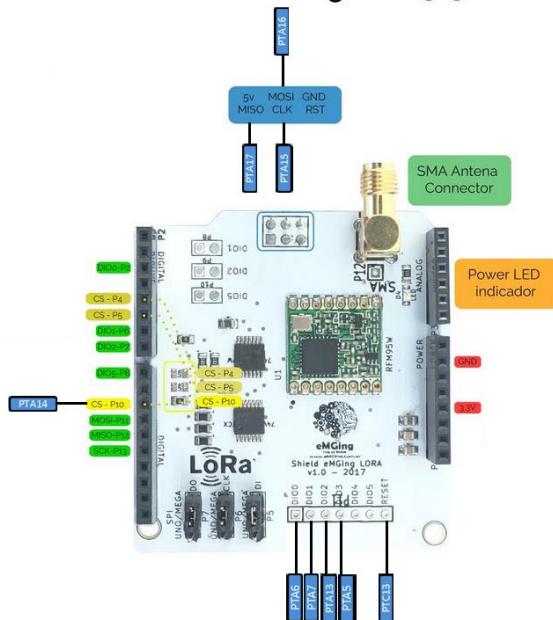


Figura 39: Conexión del nodo

Componentes relacionados al hardware:

- Reset: Es un GPIO que reinicia el transceiver cada vez que se inicializa el hardware
- SPI: Para acceder a los registros y comandos del transceiver se utiliza la comunicación SPI.

- RTC: en este caso se usa como contador que incrementa cada 1 tick de 1ms. El RTC corre en todos los modos de operación.
- Pines Input: estos pines son interrupciones para atender al transeiver.
  - ✓ DIO0: es usado para indicar que el radio LoRa realizo correctamente la tarea pedida (TxDone/RxDone).
  - ✓ DIO1: es usado para indicar que el radio fallo en la tarea pedida (RxTimeout).
  - ✓ DIO2: es usada para indicar que se detectó un preámbulo FSK (En nuestro caso no se usa).
  - ✓ DIO3/4/5: están reservados para futuros usos.
- Pines Output: son los LED de indicación.
  - ✓ LED Verde: indica que se realizó una transmisión (TxDone).
  - ✓ LED Rojo: indica que se realizó una recepción (RxDone).
- Push Boton: es un botón de reset que vuelve a inicializar el hardware.

#### 5.4.13 Aplicación del nodo

Como el objetivo de este proyecto no es hacer una aplicación sino los dispositivos de la red LoRaWAN, el nodo va a realizar la tarea de enviar cada determinado tiempo (Dutycycle) un mensaje Uplink al servidor la palabra “LoRa” a su vez si llega un mensaje Downlink debe imprimir el payload, en este caso también será la palabra “LoRa”

#### 5.4.14 Código

El programa ejecutado en el hardware para concebir el nodo es el desarrollado por Semtech y Stakforce. Esta librería tiene muchas generalidades, como por ejemplo, varios planes de frecuencia, compatibilidad con varios transceiver, etcétera, nosotros hemos utilizado únicamente lo necesario para nuestro proyecto (Plan de frecuencia AU915-928MHz y Transeiver Sx1276). Además se realizaron modificaciones en varios archivos para que el código responda a la placa de desarrollo que utilizamos.

##### 5.4.14.1 Características del código

- Concordancia con las especificaciones del protocolo de LoRa Alliance (LoRaWAN)
- Protocolos de clases A, B y C
- Banda ISM AU915-928MHz
- Activación por OTAA o ABP
- Soporta ADR

Nuestro código final incluye:

- El LoRa stack middleware
  - Capa LoRaWAN

- Utilidades LoRa como: potencia, buffer de datos, timer del sistema y de la aplicación
- Software de criptografía LoRa
- Máquina de Estado LoRa
- Soporte del transceiver Sx1276
  - Radio Semtech driver
- Main.c de Clase A, B y C
- SDK MKL46Z256VLL4 MCU, librerías de la placa de desarrollo

En la Figura 40 se observa, de una forma menos abstracta, el firmware que se ejecuta para realizar la aplicación final del usuario. En azul aparece lo que pertenece al MCU y en verde lo provisto por Semtech.

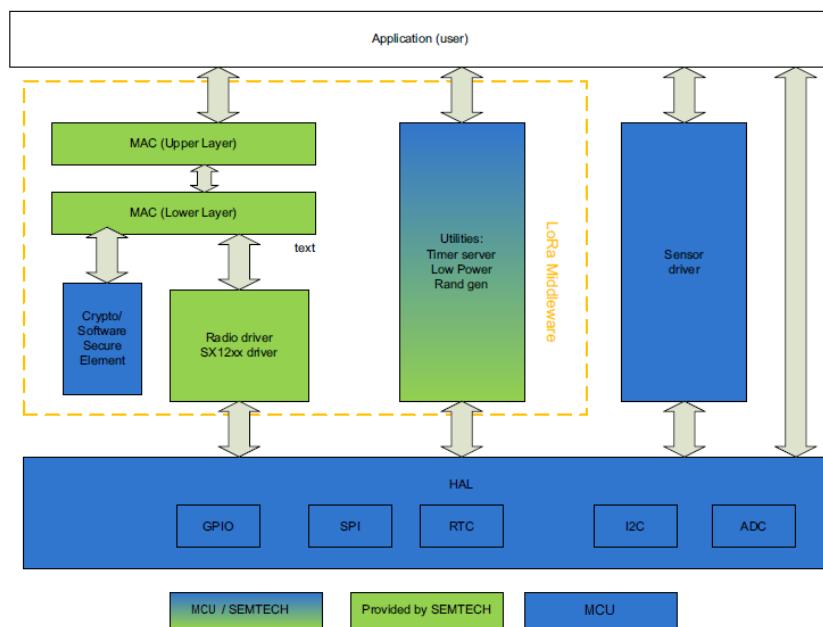


Figura 40: Arquitectura del firmware

El HAL (Hardware Abstraction Layer) está constituido por las APIs que manejan el hardware MCU requerido por la aplicación.

El RTC da una unidad de tiempo centralizada que si se desea sigue ejecutándose en modo Low-Power y así mediante la alarma del RTC se despierta al sistema. En nuestro caso la optimización de energía se obvio, por lo tanto, el nodo nunca entra en los modos de baja potencia.

El driver del radio utiliza el SPI y los GPIO. Este driver también provee APIs para ser usada en un alto nivel y controlar el radio. En nuestro caso es el drive del radio sx1276 y es proveido por Semtech.

La MAC controla el uso de la capa física (PHY). La interfaz MAC con el driver de PHY utilizan el servidor de temporización (es el que administra las temporizaciones de los eventos a ejecutar), para agregar o sacar tareas programadas. Asegurándose que todas

las temporizaciones respeten las limitaciones del Duty Cycle. La capa MAC también es conocida como LoRaMAC, esta capa ofrece los servicios de MCPS (MAC Common Part Sublayer) y sirven para transmitir o recibir los datos. También se encuentran los servicios MLME (MAC layer management entity) que administran la red LoRaWAN. Por último se encuentran los servicios MIB (MAC information base) y son los responsables de almacenar las temporizaciones y configuraciones de la capa MAC.

La máquina de estado LoRa está en un nivel intermedio entre la capa MAC y la aplicación.

La aplicación está construida en un bucle infinito y controla el Low Power (si lo hubiera), ejecuta los handlers de las interrupciones (Alarms o GPIO) y llama a la máquina de estado LoRa si hay alguna tarea para ejecutar. Esta aplicación también es la encargada de las lecturas de los sensores (si los hubiera).

#### 5.4.14.2 Estados del sistema LoRa

La Figura 41 describe los estados en los que se puede encontrar el sistema LoRa de un nodo.

- ✓ Luego de un reset o Turn On del nodo, el sistema pasa al estado **Init**. Ahí se inicializa tanto todos los componentes del Hardware como todo el stack de LoRa.
- ✓ El nodo hace un pedido de unión a la red cuando usa el método OTAA y va al estado **Mode**. En este estado puede hacer dos cosas, si el nodo se optimizo en potencia, el sistema pasa a *Sleep Mode* pero si no se optimizo se queda en *Wait Mode*.
- ✓ Cuando usa ABP, el nodo ya está unido a la red entonces el nodo salta directamente al estado **Send**. En este estado hace todo el proceso de envío y recepción de trama como se definió en la Clase A.
- ✓ Desde el estado **Mode**, si el nodo se ha unido a la red cuando un “TimerEvent” ocurrió, el nodo entra temporalmente al estado **Joined** y luego pasa al estado **Send**.
- ✓ Desde el estado **Mode**, si el nodo se ha unido a la red cuando un “OnSendEvent” ocurre, el nodo va al estado **Send**.
- ✓ Desde el estado **Send**, el nodo vuelve al estado **Mode** para esperar el “OnSendEvent” correspondiente al siguiente paquete programado para enviar.

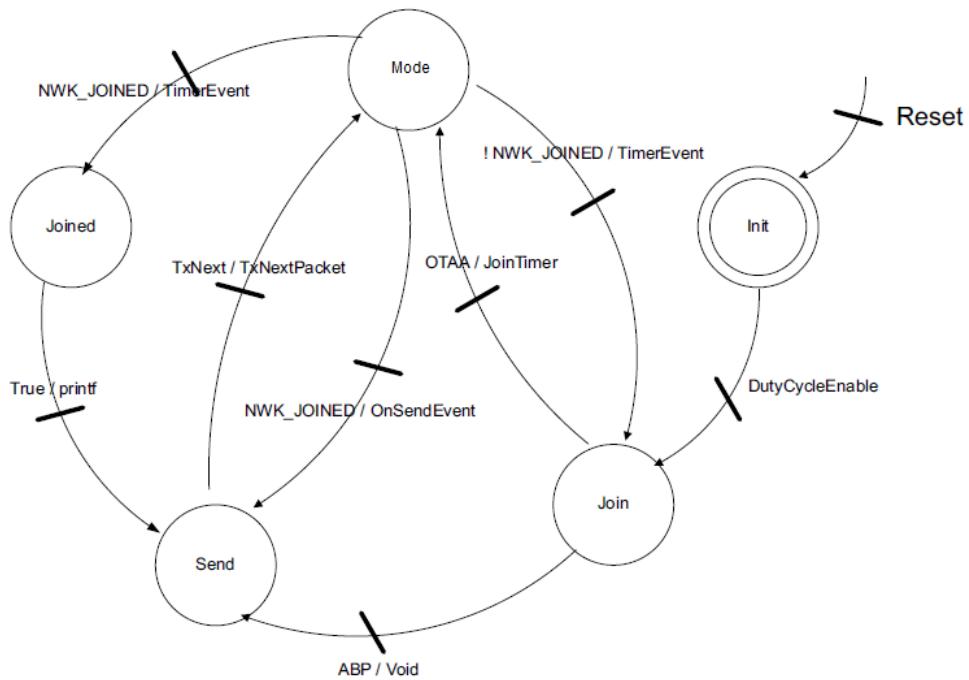


Figura 41: Estados LoRa

En la Figura 42: Estado LoRa, Clase B se observan los estados LoRa que se incorporan a la Clase B, debido al uso del beacon.

- ✓ Antes de pedir el cambio de Clase A a B el nodo debe primero estar unido a la red.
- ✓ La decisión de comutar de Clase siempre viene de la capa de aplicación del nodo. Si la decisión viene de la red, el server de aplicación debe usar el Uplink de la clase A para enviar de vuelta un Downlink para la capa de aplicación.
- ✓ Con MLME\_Beacon\_Acquisition\_req el nodo clase B va al estado **Beacon Acquistion**.
- ✓ El nodo comienza la adquisición del beacon. Cuado la capa MAC ha recibido un beacon en la cuncion RxBeacon exitosamente, el siguiente estado es **Beacon Locked**.
- ✓ Como el nodo ya ha recibido un beacon. La adquisición ya no está pedida, entonces la capa MAC entra al estado **Beacon Idle**.
- ✓ En **Beacon Idle**, la capa MAC compara el BeaconEventTime con el tiempo actual del nodo. Si el BeaconEventTime es menos que el actual, la capa MAC entra al estado **Beacon Reacquisition** de lo contrario va a **Beacon Guard**. Entonces realiza una nueva adquisición de beacon.
- ✓ Si la capa MAC no encontró un beacon, el estado de la maquina permanece en **Beacon Acquisition**. Este estado detecta que una adquisición estaba pendiente anteriormente y cambia al siguien estado **Beacon Lost**.
- ✓ Cuando la capa MAC recibe un formato de beacon malo, este debe ir a **Beacon Timeout**. En este estado se ensancha la ventana de Timeouts para incrementar la chance de recibir el siguiente beacon e ir a **Beacon Reacquisition**.

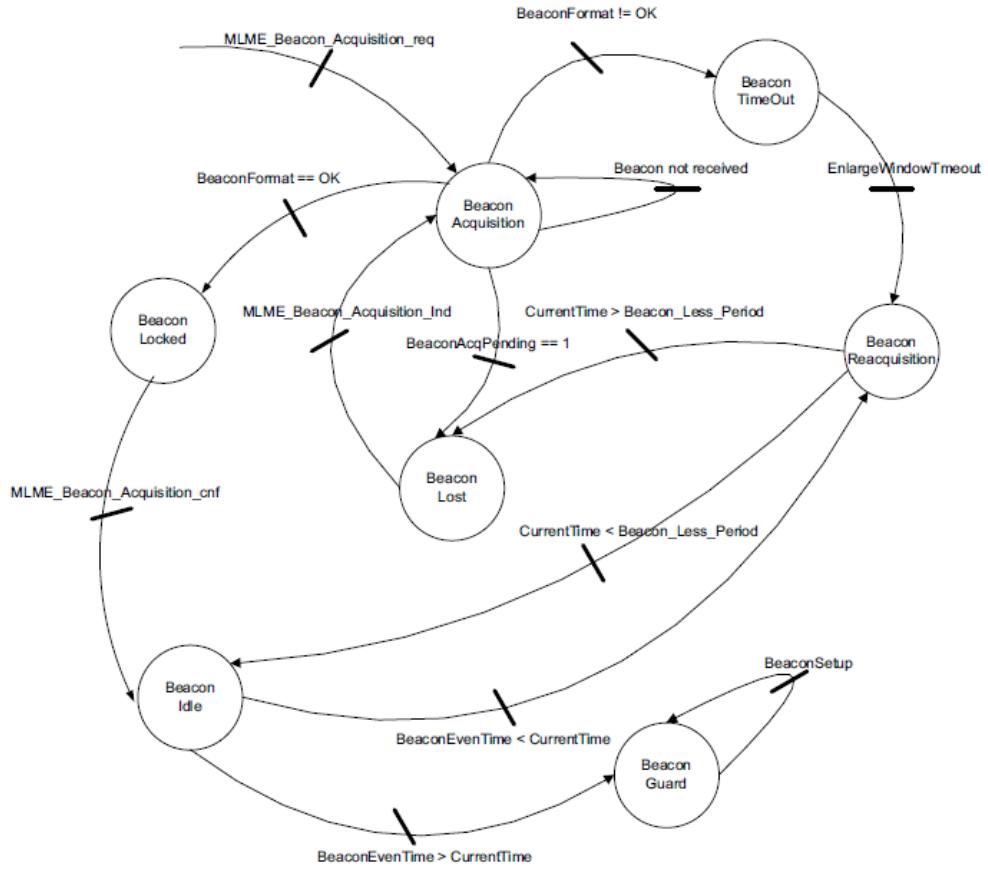


Figura 42: Estado LoRa, Clase B

#### 5.4.14.3 Variables de utilidad

En la *Tabla 19* se encuentra el listado de todas las variables que se pueden modificar para configurar el funcionamiento del sistema, a necesidad del usuario.

| Parametros a modificar   | Descripción  | Archivo |
|--------------------------|--|---------|
| USE_TTN_NETWORK          | Se configura en '1' si se usa TTN  | main.c  |
| APP_TX_DUTYCYCLE         | Se pone en ms el tiempo del ciclo de trabajo   | main.c  |
| APP_TX_DUTYCYCLE_RND     | Se configura un delay random para el duty cycle de transmision                         | main.c  |
| LORAWAN_DEFAULT_DATARATE | Es el DR que se usa en la transmisión. Si se usa el ADR este debe configurarse en DR_0 | main.c  |

|                              |  |                   |
|------------------------------|--|-------------------|
| LORAWAN_CONFIRMED_MSG_ON     | Se setea en ‘1’ para que el servidor confirme los Uplink | main.c            |
| LORAWAN_ADR_ON               | Set en 1, se activa el ADR                               | main.c            |
| LORAWAN_APP_PORT             | Se indica el puerto utilizado por la aplicación          | main.c            |
| ACTIVE_REGION                | Se debe indicar la región LoRaMAC elegida                | main.c            |
| REGION_AU915                 | Se debe definir la región a utilizar en ambos archivos.  | main.c / región.c |
| OVER_THE_AIR_ACTIVATION      | Set en 1 para OTAA o 0 para ABP                          | Commissioning.h   |
| ABP_ACTIVATION_LRWAN_VERSION | Configuracion de la versión LoRaWAN (V1.0.x o V1.1.x)    | Commissioning.h   |
| LORAWAN_PUBLIC_NETWORK       | Set 1 si el ndo se conecta a un servidor publico         | Commissioning.h   |
| LORAWAN_DEVICE_EUI           | EUI del Nodo   | Commissioning.h   |
| LORAWAN_JOIN_EUI             | EUI de la Aplicación                                     | Commissioning.h   |
| LORAWAN_APP_KEY              | Clave de aplicación (Para unirse en OTAA)                | Commissioning.h   |
| LORAWAN_NWK_KEY              | No Usado. Se configura la misma que APP_KEY              | Commissioning.h   |
| LORAWAN_DEVICE_ADDRESS       | Dirección del dispositivo                                | Commissioning.h   |
| LORAWAN_F_NWK_S_INT_KEY      | Clave de Sesión de Red                                   | Commissioning.h   |
| LORAWAN_S_NWK_S_INT_KEY      | Clave de Sesión de Red                                   | Commissioning.h   |
| LORAWAN_NWK_S_ENC_KEY        | Clave de Sesión de Red                                   | Commissioning.h   |
| LORAWAN_APP_S_KEY            | Clave de Sesión de Aplicación                            | Commissioning.h   |

Tabla 19: Opciones de cambio

#### 5.4.14.4 Funciones de utilidad

La *Tabla 20* presenta las funciones más relevantes y útiles del código, para tener un conocimiento rápido del código y si se desea modificar algo saber dónde se localiza.

| Función  | Descripción  | Archivo       |
|--|--|---------------|
| LoRaMacStatus_t<br>LoRaMacMcpsRequest<br>(McpsReq_t *mcpsRequest)  | Servicio MCPS, pedido para enviar un dato (Tx)                         | LoRaMac.c     |
| LoRaMacStatus_t<br>LoRaMacMlmeRequest<br>(MlmeReq_t *mlmeRequest)  | Servicio MLME, usada para generar un Join Request o Request link check | LoRaMac.c     |
| LoRaMacStatus_t<br>LoRaMacMibSetRequestConfirm<br>(MibRequestConfirm_t<br>*mibSet)   | Servicio MIB, set atributos de la capa LoRaMAC                         | LoRaMac.c     |
| LoRaMacStatus_t<br>LoRaMacMibGetRequestConfirm<br>(MibRequestConfirm_t<br>*mibGet)   | Servicio MIB, get atributos de la capa LoRaMAC                         | LoRaMac.c     |
| void<br>OnRxWindow1TimerEvent<br>(void)  | Setea el RxDelay1  | LoRaMac.c     |
| void<br>OnRxWindow2TimerEvent<br>(void)  | Setea el RxDelay2  | LoRaMac.c     |
| void OnTxDelayedTimerEvent<br>(void)   | Delay de la transmisión Tx   | LoRaMac.c     |
| void OnAckTimeoutTimerEvent<br>(void)  | Setea el tiempo fuera del acknowledgment                               | LoRaMac.c     |
| void LoRaMacProcess( void );   | Procesa los eventos de LoRaMAC   | LoRaMac.c     |
| LoRaMacStatus_t<br>RegionAU915NextChannel(<br>NextChanParams_t*<br>nextChanParams, uint8_t*<br>channel, TimerTime_t* time,<br>TimerTime_t*<br>aggregatedTimeOff ); | Se puede configurar los canales de frecuencia que se desean utilizar   | RegionAU915.c |
| bool RegionAU915RxConfig(<br>RxConfigParams_t* rxConfig,<br>int8_t* datarate )   | Se configuran los parametros de la ventana de recepción                | RegionAU915.c |

|  |   |        |
|--|---|--------|
| static void PrepareTxFrame(<br>uint8_t port )                        | Se agrega el puerto<br>de aplicación y<br>asignación del dato a<br>enviar   | Main.c |
| static void McpsIndication(<br>McpsIndication_t<br>*mcpsIndication ) | Se agrega el puerto<br>de aplicación y se le da<br>la tarea a realizar con<br>el dato recibido                                  | Main.c |
| int main( void )   | Como función<br>principal realiza un<br>bucle infinito en donde<br>administra los eventos<br>de LoraMAC y el<br>estado del nodo | Main.c |

Tabla 20: Funciones útiles

## 5.5 Ensayo de la RED LORAWAN

Para los ensayos de transmisión y recepción en la red LoRaWAN se dispone del **debug del nodo, el log del gateway y “live lorawan frame” de la aplicación web del servidor**, de esta manera se puede observar el tráfico de los paquetes y tener un seguimiento de los mismos a través de toda la red.

### Configuración del servidor.

En el servidor se creó 4 Device-Profile uno para cada clase (A-ABP, A-OTAA, B y C). En el Device-Profile se especifica la versión de LoRaWAN (V 1.0.3), la máxima potencia de transmisión (30dbm), si se activa por OTAA o ABP, RX1 delay (1 segundo), RX1 data rate offset (0), RX2 data rate (8), frecuencia de RX2 (923,3 MHz). Si el dispositivo soporta clase B además se le debe asignar: el timeout de espera de un paquete confirmado (20 segundos), la periodicidad, el DR y la frecuencia de los pings slot (cada 2 segundos, 8, 923,3MHz respectivamente). Si el nodo es clase C además se le debe asignar el timeout de espera de un paquete confirmado (10 segundos).

Luego en el server hay que generar una aplicación donde se configura la codificación y decodificación de los paquetes, si no se escoge ninguna configuración el payload aparecerá codificado en base64 del mismo modo para generar un Downlink el payload debe estar codificado en base64 y luego enviar. Una vez generada la aplicación se le deben crear los Devices. Para ello al crearlos se le asigna unos de los perfiles generados previamente, decir si se quiere activar la validación mediante el frame-counter (esto da más seguridad a la red y sirve para verificar que los paquetes lleguen, pero como estamos haciendo ensayos no es conveniente), por último se debe configurar el Device-address,

Net Sesion Key y App Sesion Key. Esto se debe hacer por cada nodo que se quiere incorporar a la aplicación.

Finalmente falta incorporar el gateway al servidor. Para que el servidor reconozca el gateway se debe poner el Gateway-ID que tiene cargado el gateway y también asignarle un Gateway-Profile que contiene los canales de frecuencia a utilizar.

Una vez registrado todo en el servidor, este es capaz de recibir los paquetes JSON del gateway y procesarlos.

### **Configuración Gateway.**

La configuración del gateway se hace a través del archivo global\_conf.json. En este archivo se debe configurar, los 8 canales de frecuencia, el ancho de banda y spread factor. Se le asigna un Gateway-ID (único para cada gateway de la red), IP del servidor al quien le enviara los paquetes, puerto del servidor (1700) y path del SPI donde se conecta el GPS. Por ultimo toda la configuración del beacon: periodo (128 segundos), frecuencia por defecto (923,3 MHz), cantidad de canales (8), salto entre canales (600KHz), DR (12, este es el Spread Factor que el gateway lo interpreta como DR fisico), ancho de banda (500KHz) y potencia de transmisión (14dbm).

### **Configuración del Nodo.**

La clase del nodo está definido por el main.c. Para que el nodo se incorpore a la red se le debe configurar el Commissioning.h. En este archivo se le debe indicar si se activará por OTAA o ABP, la versión LoRaWAN y las correspondientes claves de activación (Tabla 21) que depende de la versión utilizada y debe ser congruentes con las que se cargaron en el servidor, de lo contrario los paquetes llegan al gateway pero no al servidor.

| 1.0.X              | 1.1.X                   |
|--------------------|-------------------------|
| LORAWAN_DEVICE_EUI | LORAWAN_DEVICE_EUI      |
| LORAWAN_APP_EUI    | LORAWAN_JOIN_EUI        |
| N/A                | LORAWAN_APP_KEY         |
| LORAWAN_APP_KEY    | LORAWAN_NWK_KEY         |
| LORAWAN_NWK_S_KEY  | LORAWAN_F_NWK_S_INT_KEY |
| LORAWAN_NWK_S_KEY  | LORAWAN_S_NWK_S_INT_KEY |
| LORAWAN_NWK_S_KEY  | LORAWAN_NWK_S_ENC_KEY   |
| LORAWAN_APP_S_KEY  | LORAWAN_APP_S_KEY       |

*Tabla 21: Parámetros de activación*

Una vez realizada todas las configuraciones la red ya se encuentra operativa y los paquetes deben llegar sin ningún problema del nodo al server y viceversa.

### 5.5.1 Ensayo Clase A

En este ensayo se muestra el Uplink de la clase A. El nodo envía periódicamente un mensaje Uplink el cual recibe el Gateway y este lo envía al servidor. La periodicidad del mensaje está definida por el Duty Cycle en el nodo.

A continuación se observa lo que imprime el nodo en su debug, de esta forma nos provee información sobre dicho paquete enviado. Se puede ver que el contador de Uplink es 1, el nodo es clase A, el puerto por el que se envía el paquete es el 2. El Uplink es no confirmado por lo que el server no enviara un ACK cuando reciba este paquete y el dato (payload) es “LoRa”. Además muestra la configuración de la transmisión que utilizo: DR, frecuencia, potencia, y mascara de canales.

```
##### ===== UPLINK FRAME 1 ===== ##### //Contador de Uplink
CLASS      : A
TX PORT    : 2
TX DATA    : UNCONFIRMED
L o R a   //Payload

DATA RATE  : DR_5
U/L FREQ   : 916800000
TX POWER   : 5
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF
```

En la Figura 43 se ve el log del gateway, acá se aprecia en formato JSON del paquete que recibió del nodo mediante la transmisión wireless LoRa. Se indica la dirección del nodo y el contador de Uplink. Luego el JSON muestra toda la información de la transmisión y el payload codificado.

```
src/jitqueue.c:452:jit_print_queue(): INFO: [jit] queue contains 3 packets:
src/jitqueue.c:453:jit_print_queue(): INFO: [jit] queue contains 3 beacons:
src/jitqueue.c:459:jit_print_queue(): - node[0]: count_us=1010193842 - type=3
src/jitqueue.c:459:jit_print_queue(): - node[1]: count_us=1138194767 - type=3
src/jitqueue.c:459:jit_print_queue(): - node[2]: count_us=1266194691 - type=3
### [GPS] ###
# Valid time reference (age: 0 sec)
# GPS coordinates: latitude -33.02259, longitude -60.61694, altitude 22 m
##### END #####
JSON up: {"stat":{"time":"2019-04-24 01:23:44 GMT", "lati":-33.02259, "long":-60.61694, "alti":22, "rxnb":3, "rxok":1, "rxfw":1, "ackr":50.0, "dwnb":0, "txnb":0}}
INFO: [up] PUSH_ACK received in 45 ms
INFO: [down] PULL_ACK received in 3 ms
Dirección
INFO: Received pkt from mote: 2600299D (fcnt=1) Contador de Uplink
Uplink en JSON
JSON up: {"rxpk":[{"tmst":948753611, "time":"2019-04-24T01:23:59.559285Z", "tmms":1240104258558, "chan":0, "rfch":0, "freq":916.800000, "stat":1, "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "lsnr":9.5, "rss": -64, "size":17, "data": "QJ0pACYAAQACBxt3Yymdx6w="}]}
INFO: [up] PUSH_ACK received in 17 ms
INFO: [down] PULL_ACK received in 5 ms
```

Figura 43: Consola del gateway. Ensayo Uplink clase A

En las Figura 44 y Figura 45 se muestra desde el servidor el Uplink. Esta vez de forma descodificada. Aparecen los parámetro de recepción / transmisión (en la columna izquierda) y toda la información que lleva en la trama (en la columna derecha). Para ver el dato se debe ver el Uplink en el “Frame Live” donde aparece data:”TG9SYQ==” que es “LoRa” codificado en Base64. Se puede verificar que todos los parámetros son coincidentes en el nodo, gateway y servidor, por lo tanto se valida el ensayo de Uplink no confirmado.

| UPLINK   | 10:24:02 PM | UnconfirmedDataUp  | 2600299d |
|--|-------------|--|----------|
| <pre> ▼ rxInfo: [] 1 item   ▼ 0: {} 12 keys     gatewayId: "029570b6edc1ae45"     time: "2019-04-24T01:23:59.559285Z"     timeSinceGpsEpoch: "1240104258.558s"     timestamp: 948753611     rssi: -64     loraSnr: 9.5     channel: 0     rfChain: 0     board: 0     antenna: 0     ▼ location: {} 5 keys       latitude: -33.02259       longitude: -60.61694       altitude: 22       source: "UNKNOWN"       accuracy: 0       fineTimestampType: "NONE"     ▼ txInfo: {} 3 keys       frequency: 916800000       modulation: "LORA"     ▼ LoRaModulationInfo: {} 4 keys       bandwidth: 125       spreadingFactor: 7       codeRate: "4/5"       polarizationInversion: false   </pre> |             | <pre> ▼ phyPayload: {} 3 keys   ▼ mhdr: {} 2 keys     mType: "UnconfirmedDataUp"     major: "LoRaWANR1"   ▼ macPayload: {} 3 keys     ▼ fdhdr: {} 4 keys       devAddr: "2600299d"     ▼ fCtrl: {} 5 keys       adr: false       adrAckReq: false       ack: false       fPending: false       classB: false       fCnt: 1       fOpts: null       fPort: 2     ▼ frmPayload: [] 1 item       ▼ 0: {} 1 key         bytes: "BXt3Yw=="         mic: "299dc7ac"   </pre> |          |

Figura 44: Ensayo clase A. Consola del servidor. Información del Uplink

10:24:02 PM uplink

```

adr: false
applicationID: "1"
applicationName: "LED"
data: "TG9SYQ=="
devEUI: "000bacc9cba5e406"
deviceName: "Shield_A"
fCnt: 1
fPort: 2
▼ rxInfo: [] 1 item
  ▼ 0: {} 6 keys
    gatewayID: "029570b6edc1ae45"
    LoRaSNR: 9.5
    ▼ location: {} 3 keys
      altitude: 22
      latitude: -33.02259
      longitude: -60.61694
    name: "RAK831"
    rssi: -64
    time: "2019-04-24T01:23:59.559285Z"
  ▼ txInfo: {} 2 keys
    dr: 5
    frequency: 916800000

```

Figura 45: Ensayo clase A. Consola del servervidor. Trama Uplink

En este ensayo se muestra el Downlink (no confirmado) de la clase A. Desde la API del servidor se genera un Downlink (Figura 46), al presionar “Try it out!” enfila el mensaje para ser enviado al nodo cuando este envíe un Uplink al servervidor (protocolo clase A – sección 5.4.2). El dato a enviar es “LoRa”, primero se codifica en Base64 para que el server lo interprete.

The screenshot shows a Swagger UI interface for generating a Downlink message. At the top, the 'Response Content Type' is set to 'application/json'. Below this, there are sections for 'Parameters' and 'body'.

**Parameters:**

| Parameter                 | Value            | Description               | Parameter Type | Data Type |
|---------------------------|------------------|---------------------------|----------------|-----------|
| device_queue_item.dev_eui | 000bacc9cba5e406 | Device EUI (HEX encoded). | path           | string    |

**body:**

```
{
  "deviceQueueItem": {
    "confirmed": false,
    "data": "TG9SYQ==",
    "devEUI": "000bacc9cba5e406",
    "fCnt": 0,
    "fPort": 2
  }
}
```

Below the body section, there is a note: 'Parameter content type: application/json'.

At the bottom left is a blue button labeled 'Try it out!' and a link 'Hide Response'.

Figura 46: Generación del Downlink no confirmado desde el server

Desde el nodo se puede ver el Uplink que realizó y luego el Downlink, donde nos indica que llegó por la primera ventana de recepción, uso el puerto 2 y el dato es “LoRa”. Conjuntamente muestra tres parámetros de la recepción: DR, RSSI y SNR de la señal recibida.

```
##### ===== UPLINK FRAME 7 ===== #####
CLASS      : A
TX PORT    : 2
TX DATA    : UNCONFIRMED
L o R a

DATA RATE  : DR_5
U/L FREQ   : 918000000
TX POWER   : 5
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

##### ===== MCPS-Indication ===== #####
STATUS     : OK

##### ===== DOWNLINK FRAME 2 ===== ##### //Contador de Downlink
RX WINDOW  : 1 //Ventana de recepción (Rx1)
RX PORT    : 2
RX DATA    :
L o R a //Dato recibido

DATA RATE  : DR_13
RX RSSI    : -69
RX SNR     : 7
```

Desde el Log del Gateway (Figura 47) se puede observar primero el JSON Up que corresponde al Uplink recibido del nodo y el segundo JSON Down que corresponde al Downlink que envió el servidor al nodo. Este último JSON tiene los mismos parámetros que el JSON Up a diferencia que este no tiene SNR ni RSSI porque no recibió ninguna radiofrecuencia, pero si muestra la potencia con la cual envió el paquete (“powe”=27 [dBm]).

```

##### END #####
JSON up: {"stat": {"time": "2019-04-24 01:29:44 GMT", "lati": -33.02273, "long": -60.6
1695, "alti": 33, "rxnb": 0, "rxok": 0, "rxfw": 0, "ackr": 100.0, "dwnb": 0, "txnb": 1}}
INFO: [up] PUSH_ACK received in 10 ms
INFO: [down] PULL_ACK received in 7 ms
INFO: [down] PULL_ACK received in 2 ms
INFO: [down] PULL_ACK received in 18 ms

INFO: Received pkt from mote: 2600299D (fcnt=7)

JSON up: {"rxpk": [{"tmst": 1317486828, "time": "2019-04-24T01:30:08.292493Z", "tmms": 1240104627292, "chan": 6, "rfch": 1, "freq": 918.000000, "stat": 1, "modu": "LORA", "datr": "SF7BW125", "codr": "4/5", "lsnr": 10.8, "rss": -63, "size": 17, "data": "QJ0pACYABwAC058Tw9mJcf4="}]}
INFO: [up] PUSH_ACK received in 17 ms
INFO: [down] PULL_RESP received - token[102:216] :)

JSON down: {"txpk": {"imme": false, "tmst": 1318486828, "freq": 926.9, "rfch": 0, "powe": 27, "modu": "LORA", "datr": "SF7BW500", "codr": "4/5", "ipol": true, "size": 17, "data": "YJ0pACAAGACGjzmiq0lkfc=", "brd": 0, "ant": 0}}
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000)

```

Figura 47: Consola del gateway. Ensayo Downlink clase A

En la Figura 48 se observa el Uplink y luego el Downlink desde el servidor, por más que la hora que indica el servidor dice que el Uplink y Downlink sucedieron al mismo tiempo, esto no es así, ya que el encargado de respetar los tiempos de envío (sección 5.4.2) es el gateway.

|  | UPLINK   | 10:31:13 PM | UnconfirmedDataUp   | 2600299d |  |
|--|----------|-------------|---------------------|----------|--|
|  | DOWNLINK | 10:30:11 PM | UnconfirmedDataDown | 2600299d |  |
|  | UPLINK   | 10:30:11 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:29:09 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:28:08 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:27:06 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:26:05 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:24:02 PM | UnconfirmedDataUp   | 2600299d |  |
|  | UPLINK   | 10:23:29 PM | UnconfirmedDataUp   | 2600299d |  |

Figura 48: Consola server. Se observa el Uplink y Downlink

En la Figura 49 se observa la información del Downlink y todo lo que compone la trama LoRa. Como la información es coincidente en todos los dispositivos de la red y cumple con el protocolo se valida el Downlink no confirmado.

```

DOWNLINK      10:30:11 PM    UnconfirmedDataDown    2600299d

▼ txInfo: {} 10 keys
  gatewayId: "029570b6edc1ae45"
  immediately: false
  timeSinceGpsEpoch: null
  timestamp: 1318486828
  frequency: 926900000
  power: 27
  modulation: "LORA"
▼ loraModulationInfo: {} 4 keys
  bandwidth: 500
  spreadingFactor: 7
  codeRate: "4/5"
  polarizationInversion: true
  board: 0
  antenna: 0

▼ phyPayload: {} 3 keys
  mType: "UnconfirmedDataDown"
  major: "LoRaWANR1"
▼ macPayload: {} 3 keys
  fhdr: {} 4 keys
    devAddr: "2600299d"
    fCtrl: {} 5 keys
      adr: true
      adrAckReq: false
      ack: false
      fPending: false
      classB: false
      fCnt: 2
      fOpts: null
    fPort: 2
▼ frmPayload: [] 1 item
  ▼ 0: {} 1 key
    bytes: "Gjzmig=="
  mic: "a3a591f7"

```

Figura 49: Consola del server. Ensayo clase A. Información del Downlink

### 5.5.2 Ensayo clase B

El ensayo de clase B consiste en enviar Uplinks (Clase A) mientras intenta capturar un beacon, una vez obtenido se pasa a Clase B, ahora el nodo puede recibir Downlink en las ventanas Rx1, Rx2 y Ping Slot.

A continuación se observa el debug del nodo. Primero se ve la información del Uplink 1 que envía y el Downlink 27 que recibe, pero antes intento encontrar el beacon el cual no halló (Beacon not found). En este primer Uplink el nodo envía un TimeDeviceReq (Figura 52), solicitando el GPS epoch, inmediatamente en el Downlink se lo envía (Figura 53). Con esta referencia temporal el nodo tiene un tiempo aproximado del cual llegara el próximo beacon.

En el debug se ve el mensaje “Busy beacon reserved time”. Esto se debe a que el nodo envía un Uplink periódicamente pero si el nodo está utilizando el transceiver para recibir el beacon, no toma dicha solicitud y no envía el paquete. En el uso del transceiver tiene prioridad recibir un beacon que enviar un Uplink.

Luego se observa que capture el Beacon 1240197376, entonces el nodo solicita el paso de la clase A a B mediante el envío del Uplink 2 (Figura 54). Acá le envía un “PingSlotInfoReq” para informarle que periodicidad de Ping Slot va a utilizar. Una vez realizado esto el nodo cambia a Clase B.

El Uplink 3 (Figura 55) ya se envía como clase B, aunque tiene las mismas características que en Clase A.

El Downlink 30 no llegó ni por la ventana Rx1 o Rx2 sino que por un Ping Slot. Pero el Downlink 32 llegó por la ventana Rx1 del Uplink 4.

Al igual que con el Beacon, el Ping Slot tiene prioridad frente al Uplink, por eso en algunas ocasiones el nodo muestra “Busy ping-slot window time”. Esto quiere decir que el nodo no envió el Uplink programado debido a que el transceiver estaba escuchando una ventana de Ping Slot.

```
##### ===== UPLINK FRAME 1 ===== ##### //Uplink con pedido de TimeDeviceReq
CLASS : A

TX PORT : 2
TX DATA : UNCONFIRMED
00

DATA RATE : DR_2
U/L FREQ : 917200000
TX POWER : 0
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

##### ===== MLME-Confirm ===== #####
STATUS : Beacon not found //El beacon no se encontró

##### ===== DOWNLINK FRAME 27 ===== ##### //Downlink con GPS Epoch
RX WINDOW : 1
RX PORT : 0

DATA RATE : DR_10
RX RSSI : -82
RX SNR : 8

##### ===== MCPS-Request ===== #####
STATUS : Busy beacon reserved time //No se envió el Uplink ya que está esperando un beacon

##### ===== MLME-Confirm ===== #####
STATUS : OK

##### ===== BEACON 1240197376 ===== ##### //Información del beacon capturado
GW DESC : 0
GW INFO : 28 7D D2 56 55 D5

FREQ : 923300000
DATA RATE : DR_8
RX RSSI : -81
RX SNR : 7

##### ===== UPLINK FRAME 2 ===== ##### //Uplink con pedido de “PingSlotInfoReq”
CLASS : A

TX PORT : 2
TX DATA : UNCONFIRMED
00

DATA RATE : DR_2
U/L FREQ : 917200000
TX POWER : 0
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF
```

```

##### ===== Switch to Class B done. ===== ###### //Cambio a clase B

##### ===== DOWNLINK FRAME 28 ===== #####
RX WINDOW : 1
RX PORT   : 0

DATA RATE : DR_10
RX RSSI   : -72
RX SNR    : 7

##### ===== UPLINK FRAME 3 ===== #####
CLASS     : B //Ya se encuentra en la clase B

TX PORT   : 2
TX DATA   : UNCONFIRMED
00

DATA RATE : DR_2
U/L FREQ  : 917200000
TX POWER  : 0
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

##### ===== DOWNLINK FRAME 30 ===== #####
RX WINDOW : Ping-Slot //Downlink recibido por la ventana de ping slot
RX PORT   : 1
RX DATA   :
4C 6F 52 61

DATA RATE : DR_8
RX RSSI   : -79
RX SNR    : 7

##### ===== MCPS-Request ===== #####
STATUS    : Busy ping-slot window time //No se envió el Uplink ya que está escuchando en una ventana de ping
slot

##### ===== DOWNLINK FRAME 32 ===== #####
RX WINDOW : 1 //Downlink recibido por la ventana Rx1
RX PORT   : 1
RX DATA   :
4C 6F 52 61

DATA RATE : DR_10
RX RSSI   : -77
RX SNR    : 8

```

El log del Gateway muestra la información completa del beacon, ya que el Gateway es el encargado de generar y enviar uno cada 128 segundos. En la Figura 50 se ve la información del beacon enviado. La Figura 51 muestra los campos correspondiente a cada byte del beacon. Estos campos son: RFU (Reserved for Future Usage), TimeEpoch de la red, el CRC (Cyclic Redundancy Check) de esos dos campos, especificaciones del gateway (que consiste en el tipo de antena que utiliza, omnidireccional o sectorial. Latitud y Longitud de la ubicación del mismo), RFU y CRC de esos campos.

Parte de esta información se puede observar en el debug del nodo ya descodifica.

```

src/jitqueue.c:459:jit_print_queue(): - node[1]: count_us=294755338 - type=3
src/jitqueue.c:459:jit_print_queue(): - node[2]: count_us=422754874 - type=3
### [GPS] ###
# Valid time reference (age: 0 sec)
# GPS coordinates: latitude -33.02259, longitude -60.61694, altitude 26 m
##### END #####
JSON up: {"stat":{"time":"2019-04-25 03:15:34 GMT","lati":-33.02259,"long":-60.61694,"alti":26,"rxnb":1,"rxok":0,"rxfw":0,"ackr":100.0,"dwnb":0,"txnb":0}}
INFO: [up] PUSH_ACK received in 4 ms
INFO: [down] PULL_ACK received in 34 ms
INFO: [down] PULL_ACK received in 1 ms
INFO: Beacon dequeued (count_us=166755029)
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000)
INFO: Beacon queued (count_us=550754795, freq_hz=923300000, size=19):
=> 00 00 00 80 EA EB 49 6D AF 00 28 7D D2 56 55 D5 00 AB 88
INFO: [down] PULL_ACK received in 1 ms

INFO: Disabling GPS mode for concentrator's counter...
INFO: host/sx1301 time offset=(1556161991s:246258μs) - drift=-25μs
INFO: Enabling GPS mode for concentrator's counter.

```

Figura 50: Ensayo clase B. Consola del gateway. Datos del Beacon

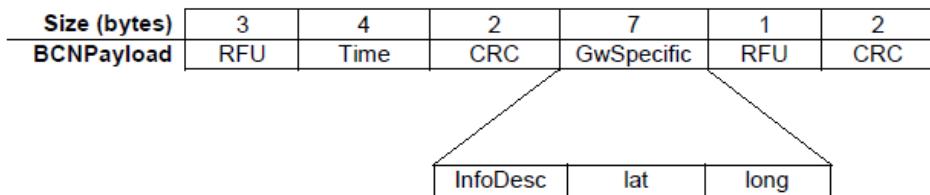


Figura 51: Campos de la trama de un beacon

|  |             |   |          |
|--|-------------|---|----------|
| DOWNLINK   | 12:14:13 AM | UnconfirmedDataDown   | 2600299e |
| UPLINK   | 12:14:13 AM | UnconfirmedDataUp   | 2600299e |
| <pre> ▼ rxInfo: [] 1 item   ▼ 0: {} 12 keys     gatewayId: "029570b6edc1ae45"     time: "2019-04-25T03:14:11.940608Z"     timeSinceGpsEpoch: "1240197270.940s"     timestamp: 61695860     rssi: -59     loraSnr: 7     channel: 2     rfChain: 0     board: 0     antenna: 0     ▼ location: {} 5 keys       latitude: -33.02258       longitude: -60.61694       altitude: 26       source: "UNKNOWN"       accuracy: 0     fineTimestampType: "NONE"     ▼ txInfo: {} 3 keys       frequency: 917200000       modulation: "LORA"     ▼ LoRaModulationInfo: {} 4 keys       bandwidth: 125       spreadingFactor: 10       codeRate: "4/5"       polarizationInversion: false   </pre> |             | <pre> ▼ phyPayload: {} 3 keys   ▼ mhdr: {} 2 keys     mType: "UnconfirmedDataUp"     major: "LoRaWANR1"   ▼ macPayload: {} 3 keys     fhdr: {} 4 keys       devAddr: "2600299e"     ▼ fCtrl: {} 5 keys       adr: false       adrAckReq: false       ack: false       fPending: false       classB: false     fCntr: 1     ▼ fOpts: [] 1 item       ▼ 0: {} 2 keys         cid: "DeviceTimeReq"         payload: null     fPort: 2   ▼ frmPayload: [] 1 item     ▼ 0: {} 1 key       bytes: "xQ=="     mic: "042984d1"   </pre> |          |

Figura 52: Ensayo clase B. Consola del servidor. Uplink “DeviceTimeReq”

|   |             |  |          |
|---|-------------|--|----------|
| DOWNLINK  | 12:14:13 AM | UnconfirmedDataDown  | 2600299e |
| <pre> ▼ bxInfo: {} 10 keys   gatewayId: "029570b6edc1ae45"   immediately: false   timeSinceGpsEpoch: null   timestamp: 62695860   frequency: 924500000   power: 27   modulation: "LORA"   ▼ loramodulationInfo: {} 4 keys     bandwidth: 500     spreadingFactor: 10     codeRate: "4/5"     polarizationInversion: true   board: 0   antenna: 0   </pre> |             | <pre> ▼ phyPayload: {} 3 keys   ▼ mhdr: {} 2 keys     mType: "UnconfirmedDataDown"     major: "LoRaWANR1"   ▼ macPayload: {} 3 keys     fhdr: {} 4 keys       devAddr: "2600299e"     ▼ fCtrl: {} 5 keys       adr: true       adrAckReq: false       ack: false       fPending: false       classB: false     fCntr: 27     ▼ fOpts: [] 2 items       ▼ 0: {} 2 keys         cid: "DeviceTimeReq"         payload: {} 1 key           timeSinceGPSEpoch: 1240197270937500000       ▼ 1: {} 2 keys         cid: "DevStatusReq"         payload: null     fPort: null     frmPayload: null     mic: "0243dbeb"   </pre> |          |
| UPLINK  | 12:14:13 AM | UnconfirmedDataUp  | 2600299e |
| DOWNLINK  | 12:06:46 AM | UnconfirmedDataDown  | 2600299e |
| DOWNLINK  | 12:04:45 AM | UnconfirmedDataDown  | 2600299e |

Figura 53: Ensayo clase B. Consola del servidor. Downlink con “TimeSinceGPSEpoch”

| UPLINK                               | 12:16:47 AM | UnconfirmedDataUp          | 2600299e |
|--------------------------------------|-------------|----------------------------|----------|
| ▼ rxInfo: [] 1 item                  |             | ▼ phyPayload: {} 3 keys    |          |
| ▼ 0: {} 12 keys                      |             | ▼ mhdr: {} 2 keys          |          |
| gatewayId: "029570b6edc1ae45"        |             | mType: "UnconfirmedDataUp" |          |
| time: "2019-04-25T03:16:46.736126Z"  |             | major: "LoRaWANR1"         |          |
| timeSinceGpsEpoch: "1240197425.736s" |             | ▼ macPayload: {} 3 keys    |          |
| timestamp: 216491380                 |             | ▼ fhdr: {} 4 keys          |          |
| rssi: -63                            |             | devAddr: "2600299e"        |          |
| loraSnr: 7.2                         |             | ▼ fCtrl: {} 5 keys         |          |
| channel: 2                           |             | adr: false                 |          |
| rfChain: 0                           |             | adrAckReq: false           |          |
| board: 0                             |             | ack: false                 |          |
| antenna: 0                           |             | fPending: false            |          |
| ▼ location: {} 5 keys                |             | classB: false              |          |
| latitude: -33.02269                  |             | fCnt: 2                    |          |
| longitude: -60.61699                 |             | ▼ fOpts: {} 3 items        |          |
| altitude: 36                         |             | ▼ 0: {} 2 keys             |          |
| source: "UNKNOWN"                    |             | cid: "DevStatusReq"        |          |
| accuracy: 0                          |             | ▼ payload: {} 2 keys       |          |
| fineTimestampType: "NONE"            |             | battery: 0                 |          |
| ▼ txInfo: {} 3 keys                  |             | margin: 8                  |          |
| frequency: 917200000                 |             | ▼ 1: {} 2 keys             |          |
| modulation: "LORA"                   |             | cid: "LinkCheckReq"        |          |
| ▼ LoRaModulationInfo: {} 4 keys      |             | payload: null              |          |
| bandwidth: 125                       |             | ▼ 2: {} 2 keys             |          |
| spreadingFactor: 10                  |             | cid: "PingSlotInfoReq"     |          |
| codeRate: "4/5"                      |             | ▼ payload: {} 1 key        |          |
| polarizationInversion: false         |             | periodicity: 1             |          |
|                                      |             | fPort: 2                   |          |
|                                      |             | ▼ frmPayload: {} 1 item    |          |
|                                      |             | ▼ 0: {} 1 key              |          |
|                                      |             | bytes: "AA=="              |          |
|                                      |             | mic: "540bc401"            |          |

Figura 54: Ensayo clase B. Consola del servidor. Uplink “PingSlotInfoReq”

|          |             |                     |  |
|----------|-------------|---------------------|--|
| UPLINK   | 12:16:51 AM | UnconfirmedDataUp   | 2600299e   |
|          |             |                     | <ul style="list-style-type: none"> <li>▼ rxInfo: [] 1 item</li> <li>  ▼ 0: {} 12 keys <ul style="list-style-type: none"> <li>gatewayId: "029570b6edc1ae45"</li> <li>time: "2019-04-25T03:16:50.339894Z"</li> <li>timeSinceGpsEpoch: "1240197429.340s"</li> <li>timestamp: 220095148</li> <li>rssi: -66</li> <li>loraSnr: 7.5</li> <li>channel: 2</li> <li>rfChain: 0</li> <li>board: 0</li> <li>antenna: 0</li> </ul> </li> <li>▼ location: {} 5 keys <ul style="list-style-type: none"> <li>latitude: -33.02269</li> <li>longitude: -60.61699</li> <li>altitude: 36</li> <li>source: "UNKNOWN"</li> <li>accuracy: 0</li> </ul> </li> <li>fineTimestampType: "NONE"</li> </ul> |
|          |             |                     | <ul style="list-style-type: none"> <li>▼ txInfo: {} 3 keys <ul style="list-style-type: none"> <li>frequency: 917200000</li> <li>modulation: "LORA"</li> </ul> </li> <li>▼ LoRaModulationInfo: {} 4 keys <ul style="list-style-type: none"> <li>bandwidth: 125</li> <li>spreadingFactor: 10</li> <li>codeRate: "4/5"</li> <li>polarizationInversion: false</li> </ul> </li> </ul>   |
| DOWNLINK | 12:16:48 AM | UnconfirmedDataDown | 2600299e   |
| UPLINK   | 12:16:47 AM | UnconfirmedDataUp   | 2600299e   |

Figura 55: Ensayo clase B. Consola del servidor. Uplink clase B

Para que se pueda ver la información del beacon el printf estaba para imprimir en Hex (hexadecimal), luego se cambió para que imprima en Chart (carácter) y aquí sí se puede ver que el dato del Downlink es “LoRa”.

```
##### ===== DOWNLINK FRAME 42 ===== #####
RX WINDOW : Ping-Slot
RX PORT   : 1
RX DATA   :
L o R a    //Dato recibido

DATA RATE : DR_8
RX RSSI   : -80
RX SNR    : 4
```

Por lo ensayado se pudo verificar que el nodo captura el primer beacon y luego no encuentra ningún otro, dando como consecuencia una desincronización del nodo con la red. También se verificó que el nodo abre ventanas de escuchas en los pings slot correspondientes. Pero esto no es suficiente para validar la clase (sección 5.4.3) ya que no respeta al protocolo. La clase B fue la última en desarrollarse y es la más compleja por lo que el código aún sigue en versión beta y con actualizaciones día a día.

### 5.5.3 Ensayo clase C

En este ensayo se va a realizar las pruebas de Downlink en una ventana continua de escucha. El nodo enviara periódicamente un Uplink como se hizo con el resto de las clases y se le enviara un Downlink en el intervalo entre dos Uplink.

En el debug del nodo se puede ver que envía un Uplink y después de que paso el tiempo de las ventanas Rx1 y Rx2 se envía un Downlink desde el server e inmediatamente el Gateway lo envía, como el nodo está en una escucha permanente (Protocolo clase C-sección 5.4.4), recibe el paquete.

```
##### ===== UPLINK FRAME 1 ===== #####
CLASS      : C

TX PORT    : 1
TX DATA    : UNCONFIRMED
L o R a

DATA RATE  : DR_4
U/L FREQ   : 917200000
TX POWER   : 0
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

##### ===== MCPS-Indication ===== #####
STATUS     : OK

##### ===== DOWNLINK FRAME 10 ===== #####
RX WINDOW  : C      //Downlink recibido en la ventana continua "C"
RX PORT    : 1
RX DATA    :
L o R a

DATA RATE  : DR_8
RX RSSI    : -74
RX SNR     : 7
```

En la Figura 56 se observa que el JSON Down no ocurre inmediatamente después del JSON Up, como si ocurre en la clase A.

```

JSON up: {"stat":{"time":"2019-04-24 02:21:44 GMT","lati":-33.02257,"long":-60.61690,"alti":24,"rxnb":1,"rxok":0,"rxfw":0,"ackr":100.0,"dwnb":0,"txnb":0}}
INFO: [up] PUSH_ACK received in 8 ms
INFO: [down] PULL_ACK received in 2 ms

INFO: Received pkt from mote: 2600299F (fcnt=1)

JSON up: {"rxpk":[{"tmst":131401084,"time":"2019-04-24T02:21:57.174010Z","tmms":1240107736173,"chan":2,"rfch":0,"freq":917.200000,"stat":1,"modu":"LORA","datr":"SF8BW125","codr":"4/5","lsnr":9.0,"rss":-55,"size":17,"data":"QJ8pACYAAQABTcqA540qqA="}]}
INFO: [up] PUSH_ACK received in 55 ms
INFO: [down] PULL_ACK received in 50 ms
INFO: [down] PULL RESP received - token[53:122] : )

JSON down: {"txpk":{"imme":true,"freq":923.3,"rfch":0,"powe":27,"modu":"LORA","datr":"SF12BW500","codr":"4/5","ipol":true,"size":17,"data":"YJ8pACaACgABH/UNtGT5LTU=","brd":0,"ant":0}}
INFO: [down] a packet will be sent in "immediate" mode
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000)

```

Figura 56: Ensayo clase C. Consola del gateway. Downlink

Desde el server (Figura 57) se puede ver que entre el Uplink y Downlink hay una diferencia de 11 segundos.

|         |             |                     |  |
|---------|-------------|---------------------|--|
| DOWLINK | 11:22:10 PM | UnconfirmedDataDown | 2600299f   |
| UPLINK  | 11:21:59 PM | UnconfirmedDataUp   | 2600299f   |
|         |             |                     | <pre> ▼ rxInfo: [] 1 item   ▼ 0: {} 12 keys     gatewayId: "029570b6edc1ae45"     time: "2019-04-24T02:21:57.174010Z"     timeSinceGpsEpoch: "1240107736.173s"     timestamp: 131401084     rssi: -55     loraSnr: 9     channel: 2     rfChain: 0     board: 0     antenna: 0     ▼ location: {} 5 keys       latitude: -33.02257       longitude: -60.6169       altitude: 24       source: "UNKNOWN"       accuracy: 0       fineTimestampType: "NONE"     ▼ txInfo: {} 3 keys       frequency: 917200000       modulation: "LORA"       ▼ LoRaModulationInfo: {} 4 keys         bandwidth: 125         spreadingFactor: 8         codeRate: "4/5"         polarizationInversion: false       phyPayload: {} 3 keys       mhdr: {} 2 keys         mType: "UnconfirmedDataUp"         major: "LoRaWANR1"       macPayload: {} 3 keys       fhdr: {} 4 keys         devAddr: "2600299f"       fCtrl: {} 5 keys         adr: false         adrAckReq: false         ack: false         fPending: false         classB: false       fCnt: 1       fOpts: null       fPort: 1       frmPayload: [] 1 item       ▼ 0: {} 1 key         bytes: "TcqgAw=="       mic: "9e0aaaa0"     </pre> |

Figura 57: Ensayo clase C. Consola del servidor. Diferencia temporal entre Uplink y Downlink

En la class C se pueden enviar Downlink consecutivos sin problema, siempre que se respete el DutyCycle.

```
##### ===== DOWNLINK FRAME 12 ===== #####
RX WINDOW : C
RX PORT   : 1
RX DATA   :
L o R a

DATA RATE : DR_8
RX RSSI   : -76
RX SNR    : 7

##### ===== MCPS-Indication ===== #####
STATUS   : OK

##### ===== DOWNLINK FRAME 13 ===== #####
RX WINDOW : C
RX PORT   : 1
RX DATA   :
L o R a

DATA RATE : DR_8
RX RSSI   : -77
RX SNR    : 7
```

| Type     | Time        | Message Type        | Device ID |
|----------|-------------|---------------------|-----------|
| DOWNLINK | 11:25:16 PM | UnconfirmedDataDown | 2600299f  |
| UPLINK   | 11:25:04 PM | UnconfirmedDataUp   | 2600299f  |
| DOWNLINK | 11:25:03 PM | UnconfirmedDataDown | 2600299f  |
| DOWNLINK | 11:25:01 PM | UnconfirmedDataDown | 2600299f  |
| UPLINK   | 11:24:02 PM | UnconfirmedDataUp   | 2600299f  |
| UPLINK   | 11:23:01 PM | UnconfirmedDataUp   | 2600299f  |
| DOWNLINK | 11:22:10 PM | UnconfirmedDataDown | 2600299f  |
| UPLINK   | 11:21:59 PM | UnconfirmedDataUp   | 2600299f  |

Figura 58: Ensayo clase C. Consola del servidor. Downlinks consecutivos

Como se observó en el ensayo la clase C queda validada dado que realiza la comunicación de acuerdo al protocolo de la clase.

### 5.5.4 Ensayo activación ABP

Para utilizar ABP se debe definir la siguiente variable en el archivo Commissioning.h del código.

```
#define OVER_THE_AIR_ACTIVATION 0
```

También se deben cargar las keys en el servidor (Figura 59), estas tienen que ser las mismas que se grabaron en el nodo, de esta manera los paquetes llegan identificados al servidor.

The screenshot shows the LoRaServer application management interface. On the left, there's a sidebar with navigation links like Network-servers, Gateway-profiles, Organizations, All users, and specific sections for loraserver (Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, Multicast-groups). The main area is titled 'Applications / LED / Devices / Shield\_A'. It has tabs for CONFIGURATION, KEYS (OTAA), ACTIVATION (which is selected), LIVE DEVICE DATA, and LIVE LORAWAN FRAMES. Under the ACTIVATION tab, there are fields for 'Device address \*' (26 00 29 9d), 'Network session key (LoRaWAN 1.0) \*' (3e 1c 9d 5a ce d7 89 52 2c e8 df 14 35 d6 b6 4b), 'Application session key (LoRaWAN 1.0) \*' (5b 8e 83 fa d1 62 ad 7a 81 d7 74 a3 9c 57 1f d9), 'Uplink frame-counter \*' (0), and 'Downlink frame-counter (network) \*' (0). At the bottom right of this section is a red '(RE)ACTIVATE DEVICE' button.

Figura 59: Activación por ABP. Configuración del servidor

Cuando el nodo se energiza lo primero que hace es la unión con la red. Este método de activación es el más rápido ya que cuando se grabó el código en el nodo, se les dio las keys y la dirección de dispositivo. De esta forma en cuestión de segundos el nodo ya puede empezar a transmitir a la red.

```
##### ===== ClassA demo application v1.0.RC1 ===== #####
DevEui    : 00-0B-AC-C9-CB-A5-E4-06
AppEui    : 70-B3-D5-7E-D0-01-10-D1
AppKey    : 3E 1C 9D 5A CE D7 89 52 2C E8 DF 14 35 D6 B6 4B

##### ===== JOINED ===== #####
ABP //Método de activación utilizado

DevAddr   : 2600299D
NwkSKey   : 3E 1C 9D 5A CE D7 89 52 2C E8 DF 14 35 D6 B6 4B
AppSKey   : 5B 8E 83 FA D1 62 AD 7A 81 D7 74 A3 9C 57 1F D9

##### ===== MCPS-Request ===== #####
STATUS    : OK

##### ===== MCPS-Confirm ===== #####

```

```

STATUS : OK

##### ===== UPLINK FRAME 1 ===== #####
CLASS : A

TX PORT : 2
TX DATA : UNCONFIRMED
L o R a

DATA RATE : DR_5
U/L FREQ : 916800000
TX POWER : 5
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

```

UPLINK            11:50:42 PM            UnconfirmedDataUp            2600299d

```

▼ rxInfo: [] 1 item
  ▼ 0: {} 12 keys
    gatewayId: "029570b6edc1ae45"
    time: "2019-04-24T02:50:39.507909Z"
    timeSinceGpsEpoch: "1240109458.507s"
    timestamp: 1853734979
    rssi: -57
    loraSnr: 9.5
    channel: 0
    rfChain: 0
    board: 0
    antenna: 0
    ▼ location: {} 5 keys
      latitude: -33.02274
      longitude: -60.61695
      altitude: 46
      source: "UNKNOWN"
      accuracy: 0
      fineTimestampType: "NONE"
    ▼ txInfo: {} 3 keys
      frequency: 916800000
      modulation: "LORA"
    ▼ LoRaModulationInfo: {} 4 keys
      bandwidth: 125
      spreadingFactor: 7
      codeRate: "4/5"
      polarizationInversion: false
    ▼ phyPayload: {} 3 keys
      mhdr: {} 2 keys
        mType: "UnconfirmedDataUp"
        major: "LoRaWANR1"
      macPayload: {} 3 keys
        fhdr: {} 4 keys
          devAddr: "2600299d"
        fCtrl: {} 5 keys
          adr: false
          adrAckReq: false
          ack: false
          fPending: false
          classB: false
        fCnt: 1
        fOpts: null
        fPort: 2
      frmPayload: [] 1 item
        ▼ 0: {} 1 key
          bytes: "BXt3Yw=="
        mic: "299dc7ac"

```

Figura 60: Activación por ABP. Consola del server. Uplink

### 5.5.5 Ensayo activación OTAA

Para usar OTAA se debe definir la siguiente variable en el Commissioning.h

```
#define OVER_THE_AIR_ACTIVATION 1
```

En esta ocasión el servidor solo necesita la Application key (Figura 59), que es la misma que se grabó en el nodo. Cuando el nodo se energiza, envía a la red un paquete “JoinRequest” (Figura 63) con el Application Key, luego el server lo identifica y le devuelve un “JoinAccept” (Figura 64), donde le indica al nodo su dirección de dispositivo y sus keys. Una vez unido a la red puede comenzar a transmitir. En la Figura 62 se aprecia que el primer Uplink no tiene Device Address por lo tanto usa la Application Key y luego del “JoinAccept” el nodo ya tiene dirección y la utiliza en sus paquetes subsiguientes.

Este método de aplicación si bien se ahorra en pre cargar las keys al nodo tiene como desventaja que tarda un cierto tiempo en hacer la unión con la red. Incluso puede haber casos en que no se una con el primer “JoinRequest”, si esto ocurre tiene que esperar un DutyCycle para poder enviar el próximo, y esto involucraría mucho tiempo.

The screenshot shows the LoRaServer web interface. The left sidebar has a tree view with nodes like 'Network-servers', 'Gateway-profiles', 'Organizations', 'All users', and a selected 'loraserver' node which further branches into 'Org. settings', 'Org. users', 'Service-profiles', 'Device-profiles', 'Gateways', 'Applications', and 'Multicast-groups'. The main content area has a breadcrumb navigation 'Applications / LED / Devices / Shield\_A'. Below it, there are tabs: 'CONFIGURATION', 'KEYS (OTAA)', 'ACTIVATION', 'LIVE DEVICE DATA', and 'LIVE LORAWAN FRAMES'. The 'KEYS (OTAA)' tab is active. It displays an 'Application key (LoRaWAN 1.0) \*' field containing the hex value '3E 1C 9D 5A CE D7 89 52 2C E8 DF 14 35 D6 B6 4B'. A note below says 'For LoRaWAN 1.0 devices, this is the only key you need to set. In case your device supports LoRaWAN 1.1, update the device-profile first.' There are buttons for 'MSB', 'SET DEVICE-KEYS', and a trash icon for 'DELETE'.

Figura 61: Activación por OTAA. Configuración del servidor, Application key

```
##### ===== ClassA demo application v1.0.RC1 ===== #####
DevEui    : 00-0B-AC-C9-CB-A5-E4-06
AppEui    : 70-B3-D5-7E-D0-01-10-D1
AppKey    : 3E 1C 9D 5A CE D7 89 52 2C E8 DF 14 35 D6 B6 4B

##### ===== MLME-Request - MLME_JOIN ===== #####
STATUS    : OK
##### ===== JOINING ===== #####
##### ===== MLME-Confirm ===== #####
STATUS    : OK
##### ===== JOINED ===== #####
OTAA //Método de activación utilizado
DevAddr   : 01282BEF

DATA RATE : DR_2

##### ===== MCPS-Request ===== #####
STATUS    : OK

##### ===== MCPS-Confirm ===== #####
STATUS    : OK
##### ===== UPLINK FRAME 1 ===== #####
CLASS     : A
TX PORT   : 2
TX DATA   : UNCONFIRMED
LoRa
```

```
DATA RATE : DR_5
U/L FREQ : 917000000
TX POWER : 5
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF
```

```
##### ===== MCPS-Indication ===== #####
STATUS : OK
```

```
##### ===== DOWNLINK FRAME 0 ===== #####
RX WINDOW : 1
RX PORT : 0
```

```
DATA RATE : DR_13
RX RSSI : -74
RX SNR : 6
```

The screenshot shows the LoRaServer web interface. On the left, there's a sidebar with navigation links for Network-servers, Gateway-profiles, Organizations, All users, and a dropdown for 'loraserver' which lists Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area has a header with a back button, a search bar, and an 'admin' link. Below the header, it says 'Applications / LED / Devices / Shield\_A'. There are tabs for CONFIGURATION, KEYS (OTAA), ACTIVATION, LIVE DEVICE DATA, and LIVE LORAWAN FRAMES, with 'LIVE LORAWAN FRAMES' being the active tab. Under this tab, there are four rows of data:

| DOWNLINK | 11:41:13 PM | UnconfirmedDataDown | 01282bef         |
|----------|-------------|---------------------|------------------|
| UPLINK   | 11:41:13 PM | UnconfirmedDataUp   | 01282bef         |
| DOWNLINK | 11:41:07 PM | JoinAccept          |                  |
| UPLINK   | 11:41:07 PM | JoinRequest         | 000bacc9cba5e406 |

Below the table are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR.

Figura 62: Comunicación de una activación OTAA

|  |             |   |                  |
|--|-------------|---|------------------|
| UPLINK   | 11:41:07 PM | <b>JoinRequest</b>  | 000bacc9cba5e406 |
| <pre> ▼ rxInfo: [] 1 item   ▼ 0: {} 12 keys     gatewayId: "029570b6edc1ae45"     time: "2019-04-24T02:41:05.070307Z"     timeSinceGpsEpoch: "1240108884.069s"     timestamp: 1279297380     rssi: -58     loraSnr: 9.5     channel: 0     rfChain: 0     board: 0     antenna: 0     ▼ location: {} 5 keys       latitude: -33.02289       longitude: -60.61687       altitude: 48       source: "UNKNOWN"       accuracy: 0       fineTimestampType: "NONE"     ▼ txInfo: {} 3 keys       frequency: 916800000       modulation: "LORA"       ▼ loraModulationInfo: {} 4 keys         bandwidth: 125         spreadingFactor: 10         codeRate: "4/5"         polarizationInversion: false   </pre> |             | <pre> ▼ phyPayload: {} 3 keys   ▼ mhdr: {} 2 keys     mType: "JoinRequest"     major: "LoRaWANR1"   ▼ macPayload: {} 3 keys     joinEUI: "70b3d57ed00110d1"     devEUI: "000bacc9cba5e406"     devNonce: 1     mic: "583b9782"   </pre> |                  |

Figura 63: Activación por OTAA. Consola del server. Uplink “JoinRequest”

|   |             |  |                  |
|---|-------------|--|------------------|
| UPLINK  | 11:42:15 PM | UnconfirmedDataUp  | 01282bef         |
| DLINK   | 11:41:13 PM | UnconfirmedDataDown  | 01282bef         |
| UPLINK  | 11:41:13 PM | UnconfirmedDataUp  | 01282bef         |
| DLINK   | 11:41:07 PM | <b>JoinAccept</b>  |                  |
| <pre> ▼ txInfo: {} 10 keys   gatewayId: "029570b6edc1ae45"   immediately: false   timeSinceGpsEpoch: null   timestamp: 1284297380   frequency: 923300000   power: 27   modulation: "LORA"   ▼ loraModulationInfo: {} 4 keys     bandwidth: 500     spreadingFactor: 10     codeRate: "4/5"     polarizationInversion: true     board: 0     antenna: 0   </pre> |             | <pre> ▼ phyPayload: {} 3 keys   ▼ mhdr: {} 2 keys     mType: "JoinAccept"     major: "LoRaWANR1"   ▼ macPayload: {} 1 key     bytes: "AX2WN7HSMK2VwSzlwWIMC1ZH0oCR8VsdfTJfbg=="     mic: "1a5f03c5"   </pre> |                  |
| UPLINK  | 11:41:07 PM | JoinRequest  | 000bacc9cba5e406 |

Figura 64: Activación por OTAA. Consola del server. Downlink “JoinAccept”

### 5.5.6 Ensayo Uplink no confirmado

Definiendo en el main.c del nodo la siguiente variable se configuran los Uplink no confirmados.

```
#define LORAWAN_CONFIRMED_MSG_ON false
```

Luego todos los mensajes son mandados al server sin esperar un ACK de confirmación. Desde el debug del nodo se puede ver esta característica del Uplink y en la Figura 65 se ve desde el servidor.

```
##### ===== UPLINK FRAME 14 ===== #####
CLASS : A
TX PORT : 2
TX DATA : UNCONFIRMED //Uplink no confirmado
L o R a

DATA RATE : DR_5
U/L FREQ : 918000000
TX POWER : 5
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF
```

UPLINK            10:37:22 PM            UnconfirmedDataUp            2600299d

```
▼ rxInfo: [] 1 item
  ▼ 0: {} 12 keys
    gatewayId: "029570b6edc1ae45"
    time: "2019-04-24T01:37:19.520849Z"
    timeSinceGpsEpoch: "1240105058.521s"
    timestamp: 1748715196
    rssi: -67
    loraSnr: 9.8
    channel: 6
    rfChain: 1
    board: 0
    antenna: 0
  ▼ location: {} 5 keys
    latitude: -33.02276
    longitude: -60.61693
    altitude: 28
    source: "UNKNOWN"
    accuracy: 0
    fineTimestampType: "NONE"
  ▼ txInfo: {} 3 keys
    frequency: 918000000
    modulation: "LORA"
  ▼ loRaModulationInfo: {} 4 keys
    bandwidth: 125
    spreadingFactor: 7
    codeRate: "4/5"
    polarizationInversion: false
  ▼ phyPayload: {} 3 keys
  ▼ mhdr: {} 2 keys
    mType: "UnconfirmedDataUp"
    major: "LoRaWANR1"
  ▼ macPayload: {} 3 keys
  ▼ fhdr: {} 4 keys
    devAddr: "2600299d"
  ▼ fCtrl: {} 5 keys
    adr: false
    adrAckReq: false
    ack: false
    fPending: false
    classB: false
    fCnt: 14
    fOpts: null
    fPort: 2
  ▼ frmPayload: [] 1 item
    ▼ 0: {} 1 key
      bytes: "Q58GLw=="
    mic: "f05f44e8"
```

Figura 65: Ensayo Uplink no confirmado

### 5.5.7 Ensayo de Adatative Data Rate (ADR).

Para el ensayo del ADR se debe definir en el archivo main.c del nodo la siguiente variable:

```
#define LORAWAN_ADR_ON 1
```

El nodo comienza enviando un Uplink con el dato que tiene que enviar (LoRa) y a su vez informa al server sobre el ADR activado (Figura 66). En el Downlink consecutivo envía toda la información (DR, Tx, Lista de canales - Figura 67), optimizada según el server. Esta información también llega por el puerto 0 (null).

Una vez que se logra configurar automáticamente el nodo, el mismo empieza a transmitir los Uplink con los nuevos parámetros (Figura 68 Figura 69).

También se puede observar el cambio de potencia en la transmisión del Uplink, ya que en el primero tiene un rssi: -58 y en el segundo un rssi: -67. Por más que la señal se haya debilitado luego de aplicar los cambios del ADR, aún sigue siendo una señal aceptable y se logra transmitir a menos potencia haciendo que el nodo consuma lo menos posible para mantener la durabilidad de la batería.

```
##### ===== UPLINK FRAME 1 ===== #####
CLASS : A

TX PORT : 2
TX DATA : UNCONFIRMED
L o R a

DATA RATE : DR_2
U/L FREQ : 916800000
TX POWER : 2
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

##### ===== MCPS-Indication ===== #####
STATUS : OK

##### ===== DOWNLINK FRAME 14 ===== ##### //Recibe los nuevos parámetros a configurar
RX WINDOW : 1
RX PORT : 0

DATA RATE : DR_10
RX RSSI : -76
RX SNR : 8

##### ===== MCPS-Request ===== #####
STATUS : OK

##### ===== MCPS-Confirm ===== #####
STATUS : OK

##### ===== UPLINK FRAME 4 ===== ##### //Uplink enviado con los nuevos parámetros
CLASS : A
```

```

TX PORT : 2
TX DATA : UNCONFIRMED
LoRa

DATA RATE : DR_6
U/L FREQ : 917400000
TX POWER : 4
CHANNEL MASK: FFFF FFFF FFFF FFFF 00FF

```

|   |            |                     |          |
|---|------------|---------------------|----------|
| DL  | 1:05:36 AM | UnconfirmedDataDown | 2600299d |
| UL  | 1:05:36 AM | UnconfirmedDataUp   | 2600299d |
| <pre> ▼ rxInfo: [] 1 item   ▼ 0: {} 12 keys     gatewayId: "029570b6edc1ae45"     time: "2019-04-25T04:05:35.443928Z"     timeSinceGpsEpoch: "1240200354.444s"     timestamp: 3145199212     rssi: -58     loraSnr: 11     channel: 0     rfChain: 0     board: 0     antenna: 0     ▼ location: {} 5 keys       latitude: -33.02232       longitude: -60.61691       altitude: 23       source: "UNKNOWN"       accuracy: 0       fineTimestampType: "NONE"     ▼ txInfo: {} 3 keys       frequency: 916800000       modulation: "LORA"     ▼ LoRaModulationInfo: {} 4 keys       bandwidth: 125       spreadingFactor: 10       codeRate: "4/5"       polarizationInversion: false     ▼ phyPayload: {} 3 keys       mhdr: {} 2 keys         mType: "UnconfirmedDataUp"         major: "LoRaWANR1"       macPayload: {} 3 keys         fhdr: {} 4 keys           devAddr: "2600299d"         fCtrl: {} 5 keys           adr: true         adrAckReq: false         ack: false         fPending: false         classB: false         fCnt: 1         fOpts: null         fPort: 2       frmPayload: [] 1 item         ▼ 0: {} 1 key           bytes: "BXt3Yw=="         mic: "552a1c61" </pre> |            |                     |          |

Figura 66: Ensayo ADR. Consola del server. ADR activado en el nodo

|  |            |                     |          |
|--|------------|---------------------|----------|
| DLINK  | 1:05:36 AM | UnconfirmedDataDown | 2600299d |
| <pre> ▼ txInfo: {} 10 keys   gatewayId: "029570b6edc1ae45"   immediately: false   timeSinceGpsEpoch: null   timestamp: 3146199212   frequency: 923300000   power: 27   modulation: "LORA" ▼ loraModulationInfo: {} 4 keys   bandwidth: 500   spreadingFactor: 10   codeRate: "4/5"   polarizationInversion: true   board: 0   antenna: 0   phyPayload: {} 3 keys     mType: "UnconfirmedDataDown"     major: "LoRaWANR1"     macPayload: {} 3 keys       fhdr: {} 4 keys         devAddr: "2600299d"         fCtrl: {} 5 keys           adr: true           adrAckReq: false           ack: false           fPending: false           classB: false           fCnt: 14       fOpts: [] 1 item         0: {} 2 keys           cid: "LinkADRReq"           payload: {} 4 keys             dataRate: 6             txPower: 4             ▶ chMask: {} 16 items       redundancy: {} 2 keys         chMaskCntr: 0         nbRep: 1       ffPort: null       frmPayload: null       mic: "6f409a6e"     </pre> |            |                     |          |
| <p>UPLINK      1:05:36 AM      UnconfirmedDataUp      2600299d</p>   |            |                     |          |

Figura 67: Ensayo ADR. Consola del server. El server le informa al nodo los parámetros a utilizar.

|        |            |                   |          |
|--------|------------|-------------------|----------|
| UPLINK | 1:07:11 AM | UnconfirmedDataUp | 2600299d |
|--------|------------|-------------------|----------|

```

▼ rxInfo: [] 1 item
  ▼ 0: {} 12 keys
    gatewayId: "029570b6edc1ae45"
    time: "2019-04-25T04:07:10.725197Z"
    timeSinceGpsEpoch: "1240200449.725s"
    timestamp: 3240480481
    rssi: -67
    loraSnr: 9.8
    channel: 8
    rfChain: 0
    board: 0
    antenna: 0
    ▼ location: {} 5 keys
      latitude: -33.02275
      longitude: -60.61695
      altitude: 26
      source: "UNKNOWN"
      accuracy: 0
      fineTimestampType: "NONE"
    ▼ txInfo: {} 3 keys
      frequency: 917500000
      modulation: "LORA"
    ▼ loRaModulationInfo: {} 4 keys
      bandwidth: 500
      spreadingFactor: 8
      codeRate: "4/5"
      polarizationInversion: false
    ▼ phyPayload: {} 3 keys
      ▼ mhdr: {} 2 keys
        mType: "UnconfirmedDataUp"
        major: "LoRaWANR1"
      ▼ macPayload: {} 3 keys
        ▼ fhdr: {} 4 keys
          devAddr: "2600299d"
        ▼ fCtrl: {} 5 keys
          adr: true
          adrAckReq: false
          ack: false
          fPending: false
          classB: false
          fCnt: 4
          fOpts: null
          fPort: 2
      ▼ frmPayload: [] 1 item
        ▼ 0: {} 1 key
          bytes: "xpvFug=="
        mic: "c133cbc3"
  
```

Figura 68: Ensayo ADR. Consola del server. Uplink enviado con los parámetros que le indica el server

1:07:11 AM                    uplink

```

adr: true
applicationID: "1"
applicationName: "LED"
data: "TG9SYQ=="
devEUI: "000bacc9cba5e406"
deviceName: "Shield_A"
fCnt: 4
fPort: 2
▼ rxInfo: [] 1 item
  ▼ 0: {} 6 keys
    gatewayID: "029570b6edc1ae45"
    loRaSNR: 9.8
    ▼ location: {} 3 keys
      altitude: 26
      latitude: -33.02275
      longitude: -60.61695
      name: "RAK831"
      rssi: -67
      time: "2019-04-25T04:07:10.725197Z"
    ▼ txInfo: {} 2 keys
      dr: 6
      frequency: 917500000
  
```

Figura 69: Ensayo ADR. Consola del server. Uplink enviado con los parámetros que le indica el server

### 5.5.8 Ensayo de paquetes duplicados

En este ensayo pretendemos observar el comportamiento del servidor en dos situaciones en particular. La primera es qué sucede cuando el paquete de un nodo registrado en la red llega a dos gateways de la red, y la segunda es cuando se programa un downlink para un nodo de la red y éste se encuentra dentro del alcance de más de un Gateway.

Ambos ensayos se realizan en conjunto ya que para probar el downlink es necesario que el nodo realice un uplink primero.

Tenemos dos gateways registrados y conectados a nuestro servidor, uno es el concentrador principal que utilizamos (Raspberry Pi 3 + RAK831) que es multicanal, y el otro es el de canal simple que utilizamos en los ensayos preliminares (Raspberry Pi 3 + RFM95W). Enviamos un paquete desde un nodo, y como ambos gateways están dentro del rango de alcance del nodo, el paquete llega a ambos.

En la Figura 70 y la Figura 71 vemos como el paquete llega a los dos gateways a través de la modulación en radiofrecuencia.

```
JSON up: {"rxpk":[{"tmst":12256267,"time":"2019-05-25T19:47:18.675652Z","tmms":1242848857676,"chan":0,"rfch":0,"freq":916.800000,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","lsnr":9.0,"rss":0,-6,"size":14,"data":"gJ0pACYAAQACSA0JjZg="}]}
INFO: [up] PUSH_ACK received in 16 ms
INFO: [down] PULL_RESP received - token[135:20] :)

JSON down: {"txpk":{"imme":false,"rfch":0,"powe":27,"ant":0,"brd":0,"tmst":13256267,"freq":923.3,"modu":"LORA","datr":"SF7BW500","codr":"4/5","ipol":true,"size":12,"data":"YJ0pACagBgDXrP5M"}}
INFO: [down] PULL_ACK received in 11 ms
INFO: tx_start_delay=1497 (1497.000000) - (1497, bw_delay=0.000000, notch_delay=0.000000)
INFO: [down] PULL_ACK received in 13 ms
```

Figura 70: Consola Gateway RAK831

```
stat update: 2019-05-25 19:47:12 GMT no packet received yet
Packet RSSI: -32, RSSI: -102, SNR: 9, Length: 14 Message:'...').&....I....'
rxpk update: {"rxpk":[{"tmst":2504253008,"freq":916.8,"chan":0,"rfch":0,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","rss":-32,"lsnr":9.0,"size":14,"data":"gJ0pACYAAQACSA0JjZg="}]}
stat update: 2019-05-25 19:47:42 GMT 1 packet received
```

Figura 71: Consola Gateway RFM95W

Este mensaje es dirigido al servidor de red primero a través del bridge LoRa y luego es reenviado al servidor web de aplicaciones donde aparece en ambos Gateways registrados, como podemos verlo en la Figura 72 y la Figura 73. Aquí también podemos ver que se envía un mensaje de downlink pero sólo a través de uno de los gateways. Más adelante veremos la razón de este funcionamiento.

The screenshot shows the LoRaServer web interface. The left sidebar is titled "loraserver" and includes options: Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main content area is titled "Gateways / RAK831". It has tabs for GATEWAY DETAILS, GATEWAY CONFIGURATION, GATEWAY DISCOVERY, and LIVE LORAWAN FRAMES (which is selected). Below these tabs are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR. A table displays two rows of received frames: one DOWNLINK entry from 4:47:20 PM labeled "UnconfirmedDataDown" with ID 2600299d, and one UPLINK entry from 4:47:19 PM labeled "ConfirmedDataUp" with ID 2600299d. Each row has a dropdown arrow icon at the end.

Figura 72: Recepción en el servidor (RAK831)

The screenshot shows the LoRaServer web interface. The left sidebar is titled "loraserver" and includes options: Network-servers, Gateway-profiles, Organizations, All users, Org. settings, Org. users, and Service-profiles. The main content area is titled "Gateways / RFM95W". It has tabs for GATEWAY DETAILS, GATEWAY CONFIGURATION, GATEWAY DISCOVERY, and LIVE LORAWAN FRAMES (which is selected). Below these tabs are buttons for HELP, PAUSE, DOWNLOAD, and CLEAR. A table displays one UPLINK entry from 4:47:19 PM labeled "ConfirmedDataUp" with ID 2600299d. The table has a dropdown arrow icon at the end.

Figura 73: Recepción en el servidor (RFM95W)

Veamos con más detalle el contenido de estos paquetes. En la Figura 74 y la Figura 75 vemos efectivamente como ambos paquetes están dirigidos al mismo nodo, sin embargo podemos ver también el valor de SNR y RSSI, donde claramente muestra que el Gateway con el RAK831 tiene mejor RSSI, que se debe a su mejor antena. Esta es la razón por la cual el downlink se programa en éste Gateway, el servidor evalúa cuál de todos los gateways que están dentro del alcance de un nodo está en mejores condiciones para enviar el paquete, y lo elige para programar los downlinks.

The screenshot shows the LoRaServer application interface. On the left, there's a sidebar with navigation links: Network-servers, Gateway-profiles, Organizations, All users, loraserver (selected), Org. settings, Org. users, Service-profiles, Device-profiles, Gateways, Applications, and Multicast-groups. The main panel displays detailed information about an uplink. The rxInfo section shows a single item with gatewayId, time, and timeSinceEpoch. The txInfo section shows frequency, modulation, bandwidth, spreadingFactor, codeRate, and polarizationInversion. The macPayload section shows phyPayload, mhdr, mType, major, macPayload, fhdr, fCtrl, fPending, classB, fCntr, fOpt, fPort, fmPayload, and mic. A red box highlights the rxInfo, txInfo, and macPayload sections.

Figura 74: Detalle uplink RAK831

This screenshot is similar to Figura 74, showing the LoRaServer interface for an uplink. The sidebar and rxInfo section are identical. The txInfo section shows frequency, modulation, bandwidth, spreadingFactor, codeRate, and polarizationInversion. The macPayload section shows phyPayload, mhdr, mType, major, macPayload, fhdr, fCtrl, fPending, classB, fCntr, fOpt, fPort, fmPayload, and mic. A red box highlights the txInfo and macPayload sections.

Figura 75: Detalle uplink RFM95W

Analicemos ahora qué sucede con el paquete que llega al nodo en el servidor de aplicación. Como se observa en la Figura 76 vemos que se recibe solo un paquete. Pero como podemos ver en la Figura 77 en el paquete se detalla que el mismo fue recibido desde dos gateways distintos.

The screenshot shows the LoRaServer web interface. The left sidebar has a tree view with nodes like 'Gateway-profiles', 'Organizations', 'All users', and 'loraserver'. Under 'loraserver', there are 'Org. settings', 'Org. users', 'Service-profiles', and 'Device-profiles'. The main area shows a breadcrumb path: Applications / Prueba / Devices / mote-kl46z. Below the path is a red 'DELETE' button. There are tabs for 'VATION', 'DEVICE DATA' (which is selected), 'LORAWAN FRAMES', and 'FIRMWARE'. Below the tabs are buttons for '? HELP', 'PAUSE', 'DOWNLOAD', and 'CLEAR'. A timestamp '4:47:20 PM' and 'uplink' status are displayed. The main content area shows a JSON-like structure of the received message.

```

{
    "data": "AA==",
    "device": "0000acc9cba5e406",
    "deviceName": "mote-kl46z",
    "fCnt": 1,
    "fPort": 2,
    "rxInfo": [
        {
            "gatewayID": "029570b6edc1ae45",
            "lorasNR": 9
        },
        {
            "gatewayID": "b827ebffff2d242",
            "lorasNR": 9
        }
    ],
    "location": {
        "altitude": 19,
        "latitude": -32.94681,
        "longitude": -60.67883
    },
    "name": "RAK831",
    "rssi": -6,
    "time": "2019-05-25T19:47:18.675652Z"
}

```

Figura 76: Recepción en el nodo (servidor)

This screenshot is similar to Figure 76 but provides a more detailed view of the received message. The left sidebar includes 'Network-servers', 'Gateway-profiles', 'Organizations', 'All users', 'Org. settings', 'Org. users', 'Service-profiles', 'Device-profiles', 'Gateways', 'Applications', and 'Multicast-groups'. The main area shows the same JSON structure as Figure 76, with specific fields like 'data', 'device', 'fCnt', 'fPort', 'rxInfo', 'location', 'name', 'rssi', and 'time' highlighted with red boxes.

```

{
    "data": "AA==",
    "device": "0000acc9cba5e406",
    "deviceName": "mote-kl46z",
    "fCnt": 1,
    "fPort": 2,
    "rxInfo": [
        {
            "gatewayID": "029570b6edc1ae45",
            "lorasNR": 9
        },
        {
            "gatewayID": "b827ebffff2d242",
            "lorasNR": 9
        }
    ],
    "location": {
        "altitude": 19,
        "latitude": -32.94681,
        "longitude": -60.67883
    },
    "name": "RAK831",
    "rssi": -6,
    "time": "2019-05-25T19:47:18.675652Z"
}

```

Figura 77: Detalle del paquete recibido en el nodo

## 5.5.9 Ensayo de Alcance

En este ensayo se pretende determinar el alcance máximo que se puede lograr, alcance que estará limitado principalmente por la modulación en radiofrecuencia. Otros factores que influyen también en dicho límite es tanto la intensidad de señal con la que llega la onda al concentrador (RSSI), y la relación señal/ruido (SNR) que la determina el ambiente en el que se realiza el ensayo. Es por esto que decidimos que sean 2 las ubicaciones en las cuales realizamos las pruebas.

En la primera de ellas tratamos de replicar las peores condiciones ambientales, es decir, una zona con muchos edificios que atenúan mucho la señal, y repleta de ruido dentro de casi todo el espectro de frecuencias cercano a donde trabaja la modulación LoRa. Este primer lugar se encuentra en el centro de Rosario (*Tucumán 1346*), un lugar precisamente con alta densidad de edificios, y cercano a muchas antenas de celulares (que operan alrededor de 1GHz), lo que produce bastante interferencia.

En la segunda tratamos de reproducir un ambiente un poco más favorable, se realizó en el barrio Echesortu de Rosario, que es una zona con bastante baja densidad de edificios, está poblado en su mayoría por casas de 1 y 2 pisos, lo disminuye bastante la atenuación de la señal. También tenemos menos antenas de celular, lo que reduce el ruido ambiente y mejora la relación señal/ruido.

### ***Modalidad de ensayo***

Para realizar el ensayo se dejó el Gateway con el concentrador RAK831 y la Raspberry Pi3 conectados al servidor público TTN, de ésta manera podíamos visualizar la llegada de los paquetes y por lo tanto, confirmar una conexión exitosa.

Luego salimos a recorrer las cercanías de la zona en la que se encontraba el Gateway con el nodo compuesto por el Shield RMF95W y la placa de desarrollo FRDM-KL46Z (Figura 78) conectado a un cargador portátil de batería y programado para enviar un paquete cada un minuto, en frecuencias aleatorias y con el ADR activado.

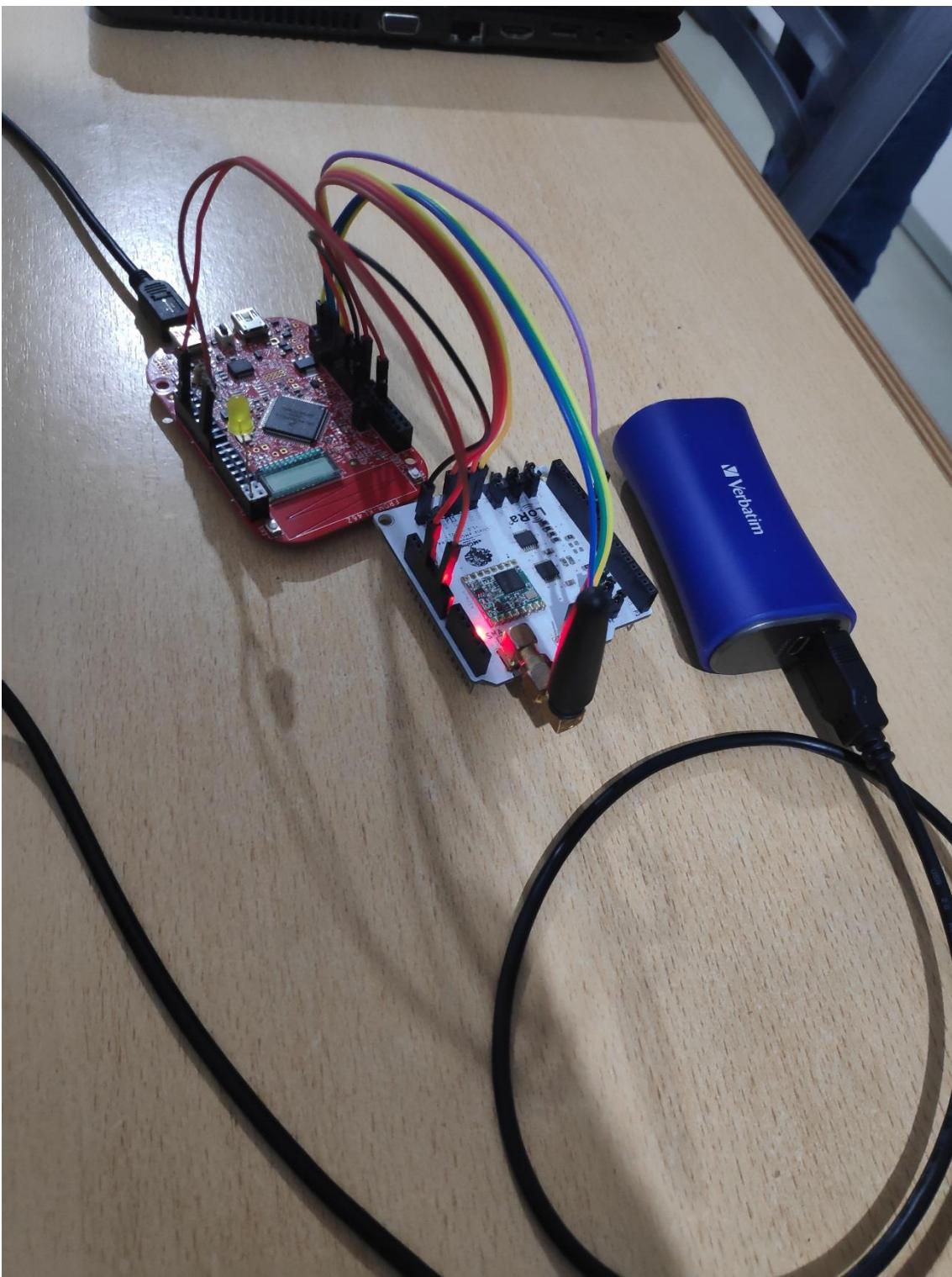


Figura 78: Nodo usado en el ensayo de alcance

Se registraron los puntos más lejanos en los cuales registramos una conexión exitosa con el Gateway así como también los valores de RSSI y SNR del paquete recibido, y se probó mandar paquetes desde distintas direcciones para probar en primera instancia la mejora del alcance mientras más directo es el camino de la onda, y en segunda instancia evaluar la direccionalidad de la antena.

### 5.5.9.1 Ubicación 1 (Tucumán 1346)

Aquí se colocó el Gateway en un cuarto piso, con vista directa a la calle (casi sobre ella). Si bien es el último piso de ese edificio, está rodeado de edificios de más de 10 pisos. En la Figura 79 se pueden ver los puntos de máximo alcance logrados en esta zona.

Tratándose de una de las peores condiciones de transmisión posibles en la ciudad podemos apreciar que el alcance promedio (unos 400 metros aproximadamente) es bastante satisfactorio por tratarse de una tecnología con muy bajo consumo de energía.

Además tenemos que tener en cuenta que el Gateway no está diseñado para aplicaciones industriales, es decir, su composición de hardware no es la ideal, y tampoco cuenta con una antena muy potente.

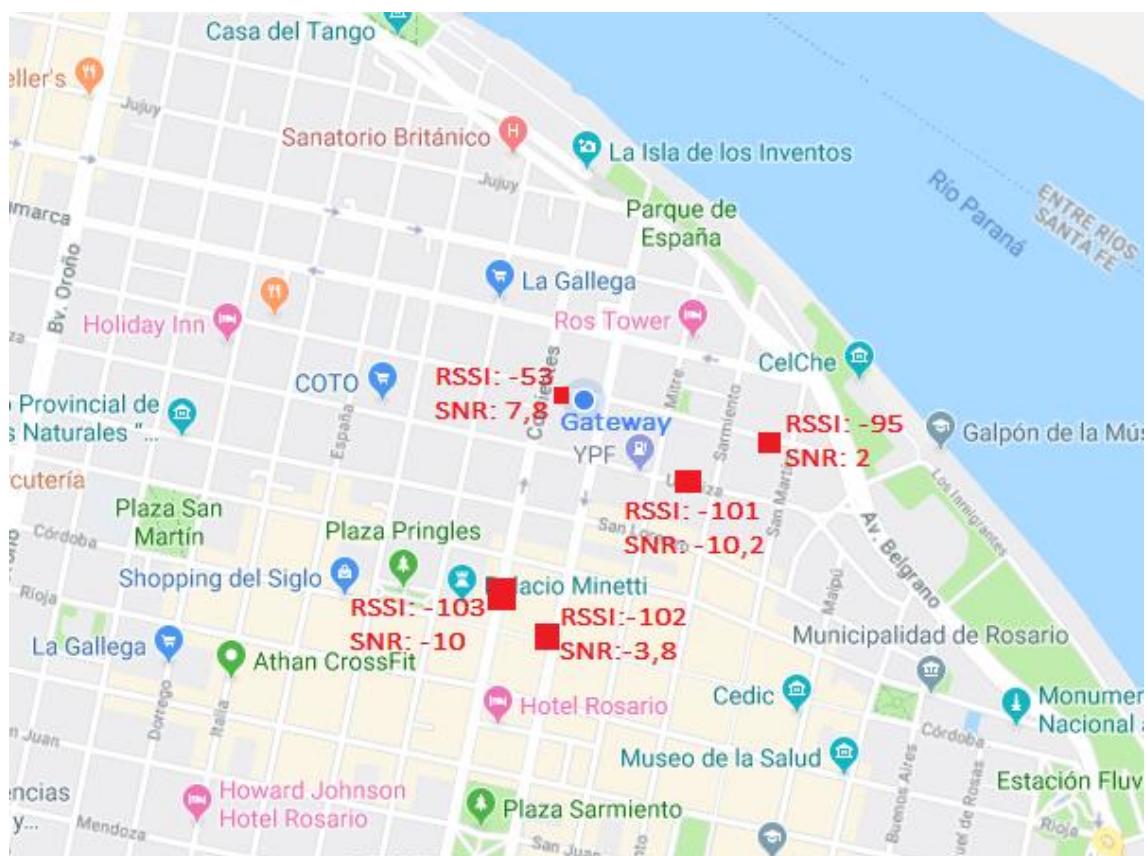


Figura 79: Mapa de alcance Tucumán 1346

### 5.5.9.2 Ubicación 2 (9 de Julio 3940)

Aquí nuevamente se colocó el Gateway en un equivalente a un cuarto piso, se lo colocó en el techo de una casa de 3 pisos. En la Figura 80 se puede ver la instalación realizada.



Figura 80: Gateway Instalado para el ensayo

Con los resultados obtenidos (Figura 81) podemos ver que efectivamente el alcance mejora en una zona con menos interferencia y atenuación. Como los datos con los que armamos el mapa de alcance se procesaron posteriormente a la toma de todos los puntos máximos de conexión exitosa, hay detalles que quedaron en evidencia una vez desmontada la instalación y por lo tanto no se pudieron volver a reproducir los ensayos.

En este caso podemos ver en el mapa que el alcance es significativamente mayor en una dirección que en la otra, y creemos que esto se debe al tanque de agua que se encontraba detrás del Gateway. Los puntos de menor alcance se encuentran justamente detrás del tanque de agua.

Este ensayo nos permite ver que con una buena disposición del Gateway se pueden lograr alcances más que satisfactorios para una ciudad como rosario, y que si se escala el sistema a una aplicación más robusta se podrían lograr alcances que no serían posibles con otras tecnologías dentro de ese consumo tan pequeño de energía.

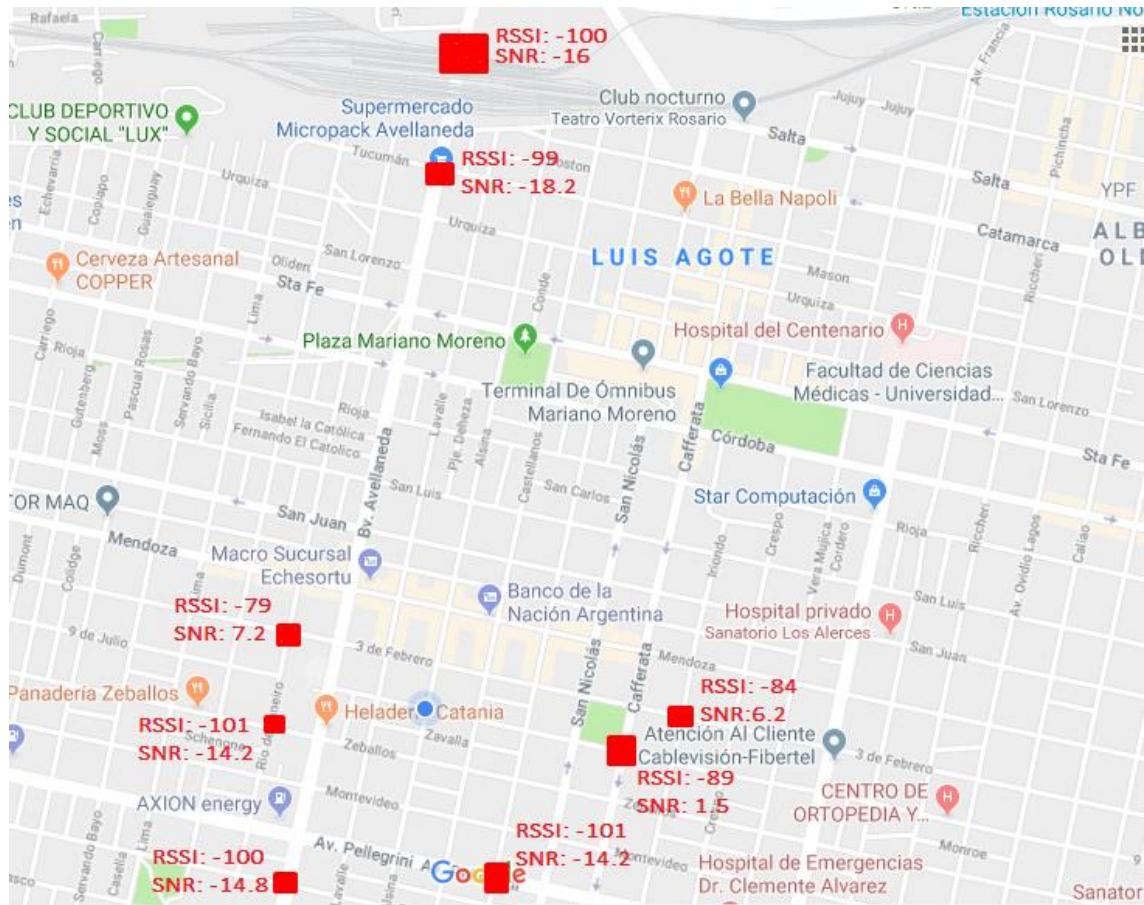


Figura 81: Mapa de alcance 9 de julio 3940

## 6 CONCLUSIONES

El objetivo principal de este proyecto es el estudio de una red LoRaWAN. Partiendo de un estudio teórico orientado a la práctica y aplicación. Pretendemos mostrar el funcionamiento específico de cada elemento de la red y a su vez su interconexión. Primeramente se montó la totalidad de la red para luego realizar los ensayos detallados en la sección 5.5, con el propósito de validar el protocolo de comunicación y determinar algunos aspectos característicos de la infraestructura. Gracias al desarrollo y ensayo de la red se pudo llegar a un entendimiento profundo del funcionamiento de las partes que componen la red. De esta manera logramos cumplir los objetivos planteados, estos son: estudio de la tecnología LoRa y LoRaWAN, implementación de la red, construcción de los nodos clase A y C, construcción del gateway multicanal, instalación del servidor LoRa y verificación de la comunicación a través de toda la red (paquete Uplink y Downlink). Quedando fuera de los objetivos: la implementación del nodo clase B y la optimización del alcance a través de comunicaciones con diferentes antenas.

Los resultados de la investigación, desarrollos y ensayos pueden ayudar (a quien decida realizar una red LoRaWAN) a interiorizarse con las especificaciones técnicas de todas las partes de dicha red, es aquí donde creemos que este proyecto toma verdadero valor, saber tomar una decisión optima a la hora de diseñar una aplicación, analizando si es conveniente o no el despliegue de una red privada u optar por una infraestructura pública. También ayuda a la mejora de decisiones al momento de diseñar una parte específica de la red, ya sea porque se quiera desarrollar un nodo puntual, desplegar gateways para capturar el tráfico LoRa u orientarse a la aplicación del usuario final partiendo de los datos del servidor.

Si bien hemos ahondado en las características y comportamiento de toda la infraestructura, existen ensayos que quedan fuera de nuestro alcance. Uno de ellos es el comportamiento de la red con un tráfico numeroso, en cuyo caso se deberá evaluar y adaptar los ensayos para determinar la configuración óptima de la red, es aquí donde se evidencia que esta tecnología es la mejor para un desarrollo futuro a gran escala, ya que permite una enorme variedad de parámetros a configurar, permitiendo un control total de todos sus componentes, control que queda limitado cuando se usa un desarrollo público o el uso de otra tecnología como SigFox por ejemplo.

Una mejora que se puede plantear en una futura implementación es el cambio de la placa de desarrollo, la utilizada no cumple con uno de los requisitos necesarios para la correcta ejecución del código, ocasionando asincronismo entre el nodo y la red. Otro punto a tener en cuenta es en la Clase B, como se mencionó anteriormente, esta clase es la más compleja y aún está en constante desarrollo. Como consecuencia al tratarse de una tecnología nueva, Argentina aún no dispuso políticas regulatorias de la tecnología por tanto todo lo que se desarrolle deberá utilizar los parámetros definidos por empresas u organismos que ya trabajan en este ámbito, pero teniendo como recaudo la posible modificación de alguno de ellos.

Los trabajos futuros que se desprenden de este proyecto pueden ser: la optimización del consumo energético del nodo, el diseño o elección de las antenas para mejorar el alcance entre el nodo y gateway, desarrollar nodos de aplicaciones IoT específicas (como SmartCity), desarrollar un prototipo comercial del nodo donde se integran todos los componentes en un mismo circuito impreso, realizar estudio del tráfico LoRa frente a una alta densidad de nodos y el comportamiento de la red. Como trabajo futuro a largo plazo se podría pensar en el despliegue de la red en campo para ejecutar una aplicación específica de IoT, que cumpla con las regulaciones y certificaciones dictadas por los entes reguladores.

## 7 BIBLIOGRAFÍA

- [1] LoRaWAN, «LoRa Alliance,» [En línea]. Available: [lora-alliance.org](http://lora-alliance.org).
- [2] Adelantado, Vilajosana, Tuset-Peiro, Martínez, Melià-Seguí, Watteyne, «Understanding the Limits of LoRaWAN,» 2017.
- [3] Semtech, «Semtech,» [En línea]. Available: <https://www.semtech.com/technology/lora/lora-applications>.
- [4] LoRa Alliance, «LoRaWAN 1.1 Regional Parameters,» 2017.
- [5] Comisión Nacional de Comunicaciones, «Resolución 302/98 (Boletín Oficial N° 28.834,11/2/98),» Buenos Aires, 1998.
- [6] W. Giezeman, «The things Network,» [En línea]. Available: [www.thethingsnetwork.org](http://www.thethingsnetwork.org).
- [7] Shenzhen Rakwireless Technology Limited Company, «RAK831 Lora Gateway Datasheet,» V1.3, 2017.
- [8] MIT, «LoRaServer,» [En línea]. Available: [www.loraserver.io](http://www.loraserver.io).
- [9] LoRa Alliance, «The Things Network,» [En línea]. Available: <https://www.thethingsnetwork.org/docs lorawan/adr.html>.
- [10] HopeRF Electronics, «RFM95(W) Low Power Low Range Transceiver Module,» V1.0.

## 8 GLOSARIO

**ACK:** Acknowledgement

**ADR:** Adaptive Data Rate

**AFC:** Automatic Frequency Correction

**AGC:** Automatic Gain Control

**BPSK:** Binary phase-shift keying

**CR:** Coding rate

**CSS:** chirp spread spectrum

**DC:** Duty cicle

**EIRG:** Effective Isotropic Radiated Power

**FIFO:** First Input, First Output

**FSK:** Frequency Shift Keying

**FSK:** Frequency Shift Keying

**GBPSK:** Gaussian phase-shift keying

**GFSK:** Gaussian Frequency Shift Keying

**GMSK:** Gaussian Minimum Shift Keying

**HAL:** Hardware Abstraction Layer

**IOT:** Internet of Things

**ISM:** Industrial, Scientific and Medical

**LNA:** Low Noise Amplifier

**LPWAN:** Low-Power Wide-Area Network

**MBUS:** Meter Bus

**OOK:** On-Off Keying

**PPS:** Pulse per Second

**ROE:** Relación de Onda Estacionaria

**RSSI:** Received Signal Strength Indicator

**SF:** Spreading Factor

**SNR:** Signal to Noise

**SPI:** Serial Peripheral Interface

**ToA:** Time on Air

**TTN:** The things network

**WAN:** Wide Area Network