Desarrollo de un sistema para la creación automática de mapas de cobertura LoRa

JONATHAN SÁNCHEZ PAREDES

MÁSTER EN INTERNET DE LAS COSAS, FACULTAD DE INFORMÁTICA,
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin Máster en Internet De Las Cosas

Convocatoria enero 2019
Calificación: 7

Curso 2018-2019

Director:

Francisco Igual Peña

Código de la aplicación

El código del presente Trabajo Fin de Máster: “Desarrollo de un sistema para la creación automática de mapas de cobertura LoRa”, se encuentra en un repositorio público de GitHub; https://github.com/JoniSanchez/heatmap_lora_signal, con licencia Apache License 2.0; https://github.com/JoniSanchez/heatmap_lora_signal/blob/master/LICENSE.

Resumen

En el presente trabajo se propone el diseño y desarrollo de una herramienta que facilite el estudio detallado y automático de la cobertura de red LoRa [1] en entornos reales. Para ello se ha desarrollado una aplicación implementada en dos fases: la primera consiste en el envío de paquetes LoRa que contienen las coordenadas geográficas de la posición del sensor; los paquetes son enviados a un *gateway* LoRa que reenviará los datos del paquete y la intensidad de señal recibida del paquete a un *broker* MQTT [2]. La segunda fase consiste en la suscripción del *broker* a un programa NodeJS [3], comprobando si los datos recibidos son correctos para enviarlos a una base de datos no relacional (MongoDB [4]), para su posterior lectura e impresión en un mapa de calor, mapa con visión satélite donde se visualizará la señal del sensor. El objetivo último de este proyecto es detallar gráficamente la intensidad de la señal en una red LoRa para ayudar a diseños de arquitectura, evitando tener puntos negros a la hora de instalar soluciones LoRa.

Palabras clave

LoRa
Docker
Geolocalización
Gateway
Mapas de calor
MongoDB
MQTT
Ansible
Google Heatmaps
GPS

Índice de contenidos

1. Introducción

Las comunicaciones inalámbricas entre dispositivos han experimentado una evolución drástica desde su nacimiento, siendo pioneras las comunicaciones de corto alcance (por ejemplo, Wifi). Con la aparición del paradigma IoT (*Internet of Things* – Internet de las Cosas) surge la necesidad de nuevas tecnologías para lograr un alcance mayor y un menor consumo energético, factores que posibilitarán mayores rangos de cobertura (de kilómetros o decenas de kilómetros), por ejemplo, en ciudades, campus empresariales o universitarios, y a la vez mayores despliegues de redes de sensores sencillos y de bajo consumo.

Como solución a las necesidades de IoT, han aparecido en el mercado las llamadas tecnologías LPWAN (*Low Power Wide Area Networks*, Redes de largo alcance y bajo consumo). Estas prestaciones han sido utilizadas en la industria 4.0, donde centenares o miles de sensores están distribuidos por una amplia área. Dichos sensores presentan severas limitaciones en términos de consumo energético, debido principalmente a su operación bajo baterías y la limitada autonomía de las mismas. LPWAN no es una tecnología única, sino un conjunto de tecnologías que incluye estándares patentados y abiertos, que utilizan frecuencias con o sin licencia. Dos de los ejemplos más extendidos son Sigfox [5] y LoRa.

SigFox es una compañía francesa que suministra el servicio de red LPWAN. Es una de las redes con mayor despliegue del mundo, operando en una frecuencia pública de 868 MHz en Europa. Utiliza la tecnología de modulación de radio UNB, lo que implica que solo se permite un operador sobre esta banda en cada país. Sigfox impone una serie de restricciones en los mensajes transferidos: cada dispositivo puede enviar 140 mensajes por día con un límite de 7 mensajes por hora con una longitud de 12 bytes. Para integrar Sigfox, es necesario un módulo de radio compatible y un plan de suscripción para el dispositivo. LoRaWAN es un estándar patentado que opera en frecuencia de 863 a 870 Mhz en redes inalámbricas de largo alcance para aplicaciones M2M (máquina a máquina). LoRaWAN es un protocolo abierto dando la posibilidad de uso a cualquier fabricante. A diferencia de Sigfox no es necesario una suscripción y no existe límite de numero de mensajes.

LoRa y Sigfox

Son una implementación de redes LPWAN (red de área amplia de baja potencia), en otras palabras, una red de largo alcance y bajo consumo. Las redes LoRa y Sigfox son interesantes por su bajo consumo de energía, lo que permite que los dispositivos tengan una duración de años con una sola batería, a cambio de un menor ancho de banda efectivo.

1.1 Motivación

En la industria 4.0 existe un gran número de sensores y actuadores distribuidos en áreas que comprenden centenares de kilómetros. En aras de un buen diseño de la arquitectura, resulta del todo necesario conocer la cobertura de nuestro sistema por todo el terreno. Para solucionar este problema se propone la introducción de una herramienta de monitorización automática de la señal para dar soporte a despliegues de arquitecturas IoT basados en LoRa. Con la ayuda de la herramienta será posible visualizar un mapa real de cobertura LoRa, obteniendo una ventaja doble:

1. Eliminación de puntos ciegos y señales de baja intensidad. Al poder ver gráficamente el mapa de cobertura, es posible observar fácilmente los puntos en los que la señal es débil o en los que la intensidad es escasa para una conexión de calidad.
2. Reducción de costes. En el mapa de cobertura es posible detectar el alcance de la señal. LoRa, pudiendo reducir el número de gateways y sensores, incluso mejorando arquitecturas ya diseñadas.

Actualmente en el mercado no hay aplicaciones que nos ayuden gráfica y automáticamente a diseñar arquitecturas LoRa, por lo que añadir una herramienta de monitorización de la señal en entornos reales puede resultar positiva, ayudando a mejorar los diseños de arquitectura LoRa.

Arquitectura

Se trata de la combinación de elementos software y hardware, que tiene como objetivo interconectar los componentes de una red informática.

1.2. Objetivos

El objetivo principal del proyecto es implementar una herramienta que facilite al usuario un estudio detallado y automático de la cobertura LoRa en entornos reales.

La herramienta desarrollada es la encargada de recolectar la señal de cobertura que hay en una zona determinada, y plasmarlo gráficamente en un mapa de calor. Este sistema permite localizar puntos ciegos y zonas de cobertura limitada para unos parámetros LoRa específicos, pudiendo reorganizar los elementos de la arquitectura (*gateways* y nodos sensores) para obtener una nueva arquitectura más eficiente proporcionando en todos puntos mejor cobertura y reduciendo costes asociados al despliegue.

En algunas situaciones, además, es necesario realizar diferentes estudios variando algún parámetro específico relativo a la comunicación, por ejemplo, un determinado ancho de banda o alcance. El sistema propuesto facilita las pruebas para llegar a eliminar los puntos ciegos para unos determinados parámetros de la comunicación como del terreno.

1.3. Estado del arte

En la actualidad, LoRa y Sigfox, se han ido posicionando en el mercado de una manera exponencial en respuesta la necesidad de tener dispositivos capaces de enviar datos con un coste de batería mínimo ha ido en aumento. El crecimiento sigue estando en auge, lo que concluye que es una tecnología poco madura con muchos puntos por determinar. Uno de ellos es la necesidad de una herramienta capaz de mostrar los datos de cobertura LoRa sobre un entorno real, de forma gráfica y automática. En cambio, si se compara con otras tecnologías que llevan más tiempo en el mercado, como WiFi, se pueden encontrar numerosas aplicaciones dando respuesta a esta necesidad, entre las que destacan:

- **WiFi Heatmap** [6]. Es una aplicación Android que se puede instalar en el móvil con acceso a la red Wifi. La aplicación escanea la intensidad con la que llegan los paquetes y los almacena en un base de datos. Posteriormente es posible crear un mapa de calor con los resultados obtenidos. La finalidad de esta aplicación es similar a la de este proyecto, pero con diversos puntos negativos, entre lo que destacan su falta de portabilidad a otras tecnologías de red y su disponibilidad únicamente en entornos Android. La persistencia y

accesibilidad de los datos es escasa, ya que no es accesible por otras aplicaciones. La propuesta, en cambio, permite el acceso a los datos recogidos tanto en MongoDB como en MQTT mediante suscripción a un *topic* determinado. Por último, el punto más negativo, no recoge la señal con los parámetros GPS [7] sino añadiendo la posición actual manualmente, característica sí soportada por nuestra propuesta.

- **TamoGraph** [8]. Es una aplicación para Mac y Windows de pago que genera mapas de calor con la intensidad de la red WiFi. Además, añade la posibilidad de escanear entornos *outdoor* usando GPS. En comparativa con la aplicación de este proyecto, se trata de un software cerrado, es decir, los datos que genera la aplicación solo pueden ser consultados por ella misma. En cambio, en este proyecto, el usuario puede consultar los datos en el bróker MQTT o en la base de datos MongoDB.

1.4. Plan de trabajo

La primera fase se basa en una toma de contacto e investigación de las tecnologías que se pueden utilizar para el sistema. Se desarrollará el plan de trabajo y la arquitectura de la aplicación.

El desarrollo de la herramienta se divide en cuatro fases, más una última fase reservada para imprevistos y contingencias. La primera fase consistirá en el desarrollo de un sistema envío de paquetes LoRa desde un equipo de bajo consumo (Raspberry Pi), incluyendo como parámetros las coordenadas de geolocalización en un determinado momento del sensor. La segunda fase consistirá en la recepción del mensaje, tratamiento de los datos y almacenado en un *broker*. En la tercera fase se consultarán los datos almacenados en el *broker*, serán tratados y si cumplen los requisitos serán almacenados persistentemente en una base de datos no relacional. En la última fase de desarrollo se consultarán los datos almacenados persistentemente y serán encapsulados y representados en un HeatMap o mapa de calor.

Por último, la última etapa del proyecto se basará en el desarrollo de pruebas de concepto, donde se corregirán los fallos detectados y las mejoras de diseño de la herramienta, recolectando la información para redactar la memoria.

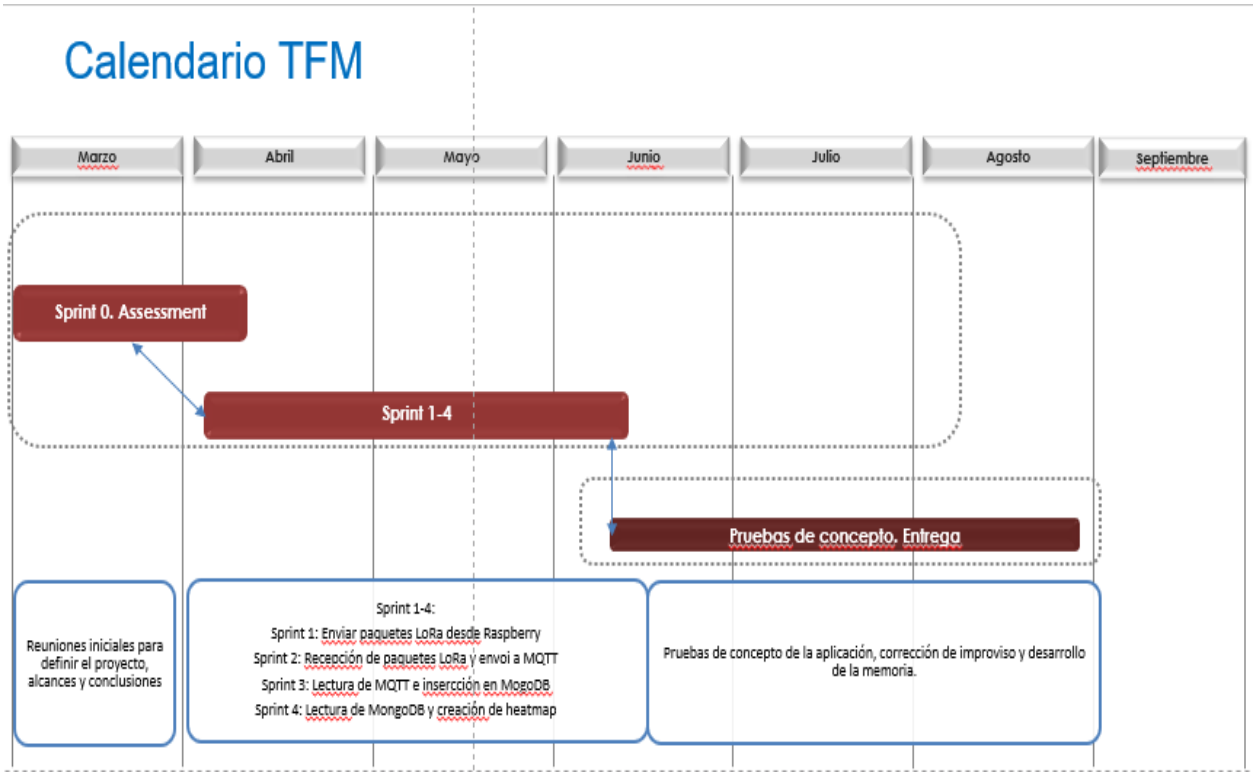

Figura 1. Calendario/cronograma del trabajo desarrollado.

2. Requisitos y consideraciones previas

Para poner en contexto qué se va a desarrollar se documenta en primer lugar la estructura y los requisitos previos para un buen funcionamiento de la aplicación y su correcta adaptación a los requisitos planteados.

2.1. Descripción general del sistema

El actor principal de la aplicación es una placa reducida de bajo coste, encargada de recolectar la posición actual de la misma y enviar mediante LoRa un paquete con su geolocalización como dato transportado por el paquete. El paquete enviado por la placa será recibido por un *gateway* LoRa que emitirá de nuevo el paquete a un *broker* remoto.

A dicho *broker* se suscribirá (utilizando el protocolo MQTT) un elemento suscriptor, leyendo constantemente cualquier nuevo mensaje recibido; una vez leído y procesado, el mensaje será almacenado de forma persistente en una base de datos no relacional. La interfaz web leerá los datos de la base de datos y los expondrá en un mapa de calor.

Por último, la interfaz web también será la encargada de definir los parámetros de comunicación LoRa que serán utilizados a la hora de enviar paquetes (por ejemplo, ancho de banda, o factor de dispersión, como se verá más adelante).

2.1.1. Casos de uso

Existen tres casos de uso de la aplicación:

1. El usuario final ejecuta el envío de paquetes desde la placa. En este caso una Raspberry PI 3 envía paquetes LoRa con la posición GPS a un bróker MQTT ejecutado en un sistema basado en Docker. El *broker* tiene un suscriptor desarrollado en NodeJS, que almacena

los mensajes recibidos en una base de datos no relacional MongoDB. Por último, los datos son leídos por la interfaz gráfica para su consulta.

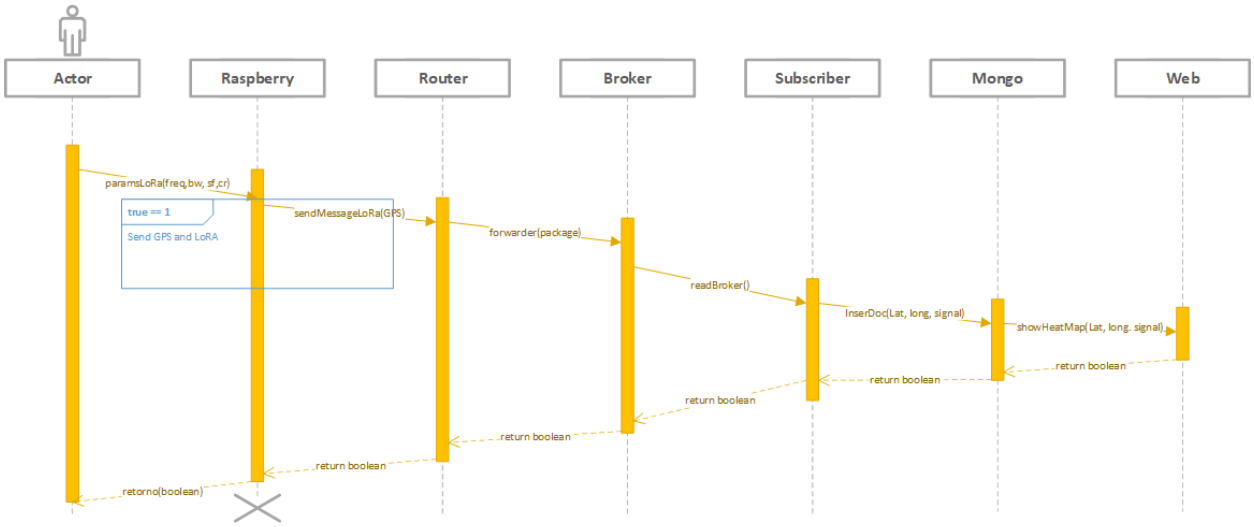

Figura 2. Caso de uso de la aplicación.

2. El usuario final accede a la interfaz gráfica para consultar los datos en la web.

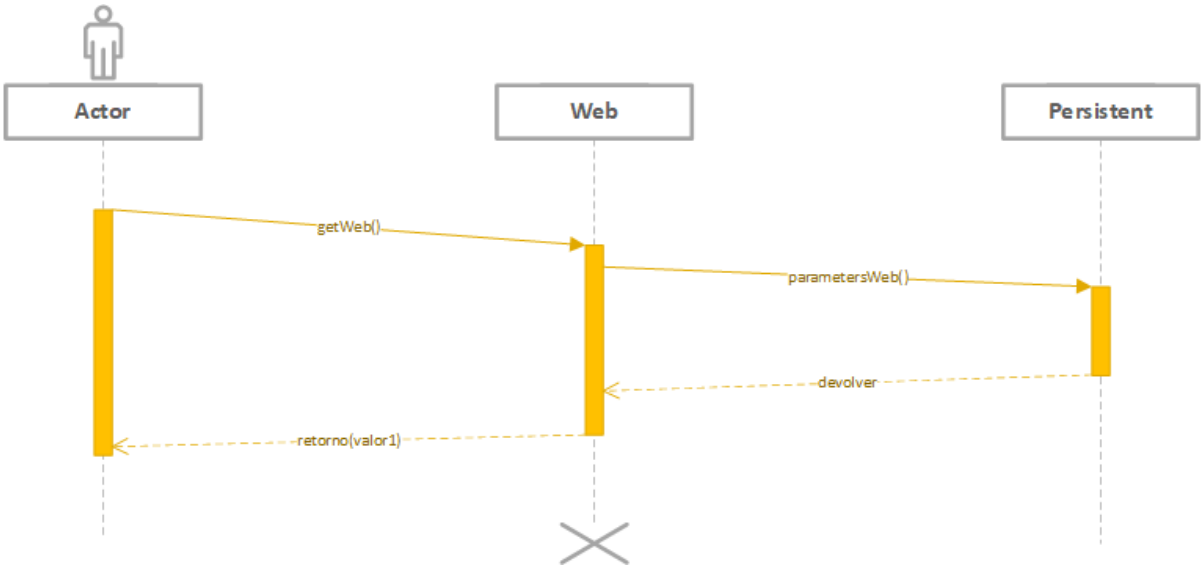

Figura 3. Caso de uso de la interfaz gráfica.

3. El usuario final modifica los parámetros LoRa para el envío de paquetes LoRa desde la interfaz gráfica ejecutando Ansible para la conexión con la Raspberry PI.

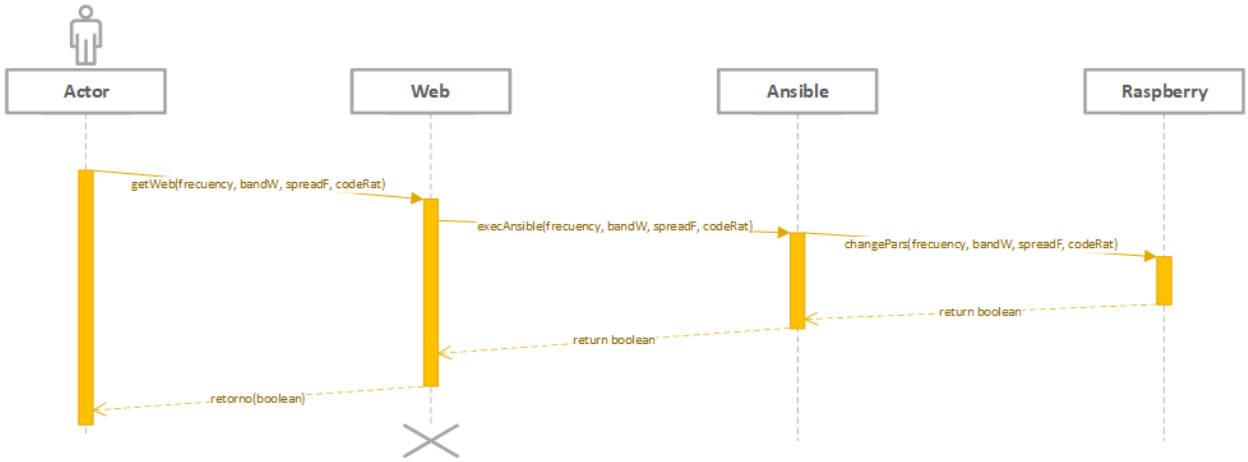

Figura 4. Caso de uso de la configuración automática de la Raspberry.

La arquitectura de la aplicación se define en tres partes, véase la Figura 5. La primera parte, en el recuadro superior izquierdo, donde Raspberry PI con un sistema operativo Raspbian envía paquetes LoRa con la posición GPS a un *gateway* LoRa. La segunda parte, en la parte inferior izquierda, donde el *gateway* LoRa recibe los paquetes LoRa y los publica en un *broker* MQTT. La tercera parte donde una aplicación NodeJS esta suscrita al bróker MQTT, lee los mensajes insertados en el mismo y los almacena en una base de datos MongoDB. Finalmente, con otra aplicación NodeJS se publican los datos en una interfaz gráfica.

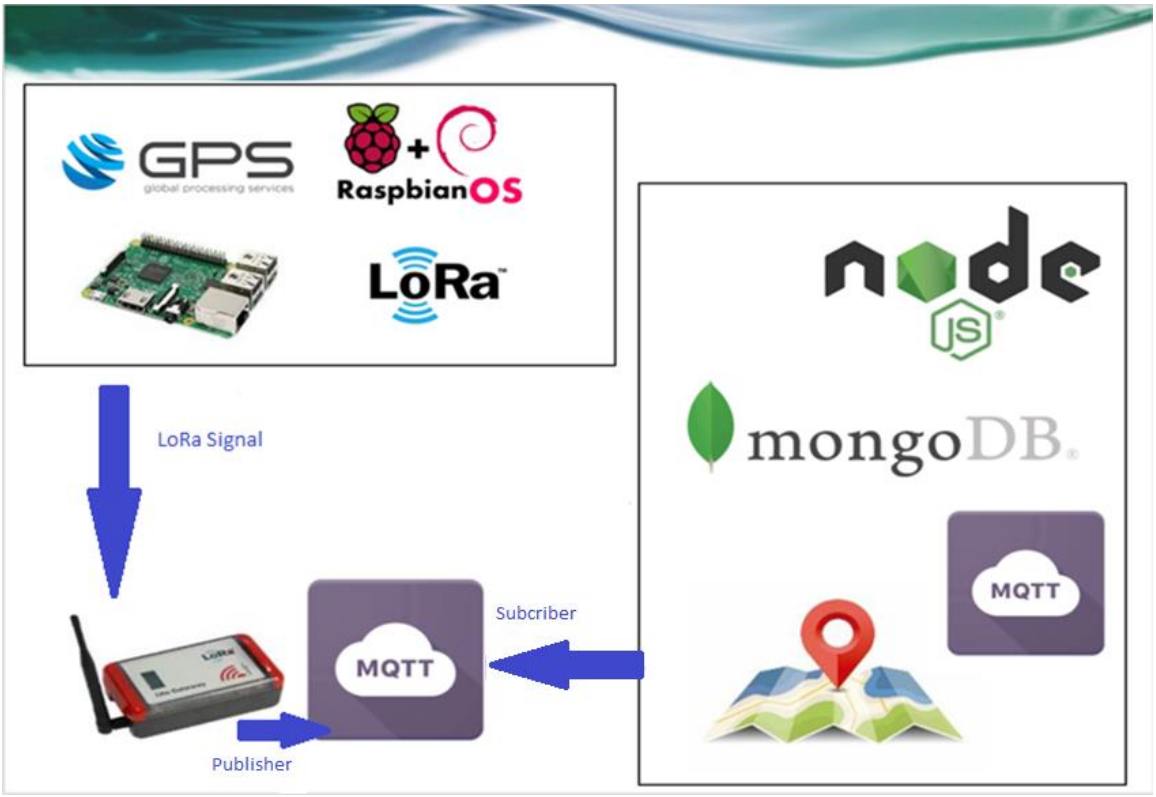

Figura 5. Arquitectura diseñada para el proyecto.

En referencia a la Figura 5, el *subscriber* es el encargado de suscribirse a un *broker* MQTT con un cierto *topic* para obtener los datos publicados. El *publisher* es el encargado de publicar mediante MQTT los datos recolectados por el sistema.

2.2 Requisitos

El sistema que se va a desarrollar trabaja mediante tecnología LoRa, por lo que es esencial disponer de un *gateway* que soporte dicha tecnología. Se trabajará sobre el alcance e intensidad de al menos un gateway, y como resultado se generará un HeatMap (mapa de calor) que detallará la intensidad de la señal LoRa alrededor del mismo.

Para el correcto funcionamiento de la aplicación tenemos que cumplir diversos requisitos tales como, un equipo (máquina virtual o física) con sistema operativo Linux, con Docker y Docker Compose instalados. Docker es el encargado de lanzar contenedores con el software necesario para

que el *susbcriber* funcione. Por otro lado, se necesitará una Raspberry PI con Raspbian instalado además de una antena LoRa con soporte GPS para geolocalizar los puntos de envío de datos.

Docker es el encargado de arrancar los contenedores que disponen del software de la aplicación y también se encargará de reiniciar el contenedor si detecta cualquier problema en su correcto funcionamiento. Docker también debe controlar el uso de recursos por parte del contenedor y realizar las conexiones entre las diferentes imágenes en ejecución.

3. Tecnologías utilizadas

En este capítulo se detallan las tecnologías empleadas para el desarrollo del sistema.

3.1. Raspberry PI y Raspbian

Por parte del *publisher*, se ha utilizado como hardware base una Raspberry PI 3 B. Se trata de un ordenador de placa reducida de coste bajo, tiene una CPU quad-core ARMv8, 1 GB de SDRAM y periféricos de bajo nivel como puede ser GPIO que usaremos posteriormente. En la Raspberry se instalará un sistema operativo exclusivo para la Raspberry -Rapsbian-. Este es un SO libre que se puede descargar de la propia web de Raspberry. Es un pequeño ordenador de bajo coste tanto económico como energético, pudiendo así transportarse durante 6 horas con una batería externa conectada mediante cable MicroUSB. Justo lo necesario para poder caminar durante 6 horas recolectando la señal LoRa que hay en cualquier posición del mapa donde esté situada.

Figura 6. Ejemplo de Raspberry PI 3.
Fuente: https://www.raspberrypi.org/downloads/raspbian/

3.2. LoRa y LoRaWAN

LoRa es una tecnología de red LPWAN (*Low Power Wide Area Network*). La patente de LoRa es de Semtech, desarrollador principal de la tecnología. Pertenece al grupo de redes de bajo consumo y gran alcance. Con cierta tolerancia ante las interferencias debido a su funcionamiento sub-Ghz, y a cambio de un ancho de banda ciertamente reducido, LoRa permite una durabilidad mayor de la batería que con otras tecnologías como Wifi, que tiene un ancho de banda mayor que permite la transferencia de más bits con un coste de batería superior. Al reducir el ancho de banda, LoRa reduce la cantidad de bits enviados por unidad de tiempo, pero también disminuye el ruido que puede aparecer en la señal. La reducción de ruido implica poder enviar una señal a una distancia superior llegando a distancias de hasta 20 km. Esta capacidad viene dada por la baja transferencia de datos, como máximo 255 bytes por mensaje haciendo a LoRa una tecnología ágil. Otro de los alicientes positivos para hacer de esta una tecnología rápida, enviando pequeños mensajes de punto a punto, aumentando positivamente su velocidad de transferencia. La frecuencia de trabajo depende en cada uno de los continentes: 915Mhz América, 868Mhz Europa, 433Mhz Asia. En nuestro caso nos focalizaremos en la frecuencia europea, utilizando valores ente 868 y 880 Mhz, esto significa que podemos variarla según las necesidades de nuestro sistema.

LoRaWAN es la red en la que opera LoRa, y pertenece a la capa MAC de la pila OSI. Principalmente, es un protocolo de nivel de red para gestionar la comunicación entre *gateways* LoRa y dispositivos sensores. Mientras LoRaWAN define la arquitectura y protocolo de comunicación, LoRa se centra en la parte física del sistema. Los dispositivos de la red emiten, de forma asíncrona, mensajes que son recibidos por uno o más *gateways* y reenviados a un servidor central. Este servidor realiza el filtrado de duplicados, chequeo de errores y gestión de red, para, finalmente, reenviar los datos a las aplicaciones pertinentes.

Existen diversos tipos de mensajes LoRaWAN:

MAC Tipo mensaje	Nombre	Descripción
000	Join Request	Estos mensajes se utilizan para establecer la conexión entre el dispositivo LoRa y *Gateway*.
001	Join Accept	Estos mensajes se utilizan para establecer la conexión entre el dispositivo final LoRa y *Gateway*.
010	Unconfirmed Data Up	Este tipo de mensaje no requiere ningún reconocimiento.
011	Unconfirmed Data Down	Este tipo de mensaje no requiere ningún reconocimiento.
100	Confirmed Data Up	Este tipo de mensaje se utiliza para incorporar funcionalidades de formato de mensaje no estándar.
101	Confirmed Data Down	Este tipo de mensaje se utiliza para incorporar funcionalidades de formato de mensaje no estándar.
110	RFU	Significa reservado para uso futuro.
111	Proprietary	Este tipo de mensaje se utiliza para incorporar funcionalidades de formato de mensaje no estándar.

A la hora de enviar mensajes LoRa existen parámetros que hay que tener en cuenta:

- **Frecuencia**: Con este parámetro podemos definir a qué frecuencia trabajará LoRa. Dependiendo el continente varía la frecuencia estándar. 915Mhz América, 868Mhz Europa, 433Mhz Asia.
- **Ancho de banda**: El ancho de banda es la tasa de bits por segundo que puede propagar como máximo de un punto a otro.
- **Tasa de codificación**: Esta técnica de agregar bits redundantes (paridad) a la transmisión para que los errores puedan recuperarse en la recepción. La velocidad de codificación se basa en la media de bits transferido que llevan información descartando mensajes de información de red y similares. La velocidad de codificación puede ser 4/5, 4/6, 4/7, 4/8. El primer valor son los datos que se envían y el segundo menos el primero son los datos de configuración que se envían. Es decir, en 4/5 se envían 4 de aplicación y 1 de configuración. A mayor corrección de errores menor velocidad de transferencia:

Coding Rate	Ciclo Coding Rate	Ratio
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

- **Factor de propagación**: Define el número de bits usados para codificar cualquier carácter de la señal. A menor factor de propagación mayor será la velocidad de transferencia, favoreciendo las interferencias en la transferencia. Dentro del ancho de banda dado, la relación entre el bit y la frecuencia de la señal para la modulación LoRa puede diferir entre el factor de expansión (SF) de 7 a 12. El dispositivo final puede transmitir en cualquier canal disponible en cualquier momento, utilizando cualquier velocidad de datos disponible:

SF	Tasa de bit a BW constante (kb/s)	Sensibilidad (dBm)
7	5.468	-123
8	3.125	-126
9	1.757	-129
10	0.976	-132
11	0.537	-134.5
12	0.293	-137

Se puede observar que un sf7 transmite a mayor velocidad, pero necesita mayor potencia, en cambio, un sf12 transmite a menor velocidad, pero con un consumo menor.

- **Modo CRC**: Se trata de un parámetro que gestiona los logs del paquete LoRa, trata los paquetes dando un error de 0 – 100%. Este parámetro es un booleano con el que se puede deshabilitar este modo.

```
// define radio settings
//////////////////////////////////////////
uint8_t power = 15;
uint32_t frequency = 868100000;
char spreading_factor[] = "sf12";
char coding_rate[] = "4/5";
uint16_t bandwidth = 125;
char crc_mode[] = "on";
```

** Parámetros utilizados por LoRaWAN.*

En conclusión, con estos parámetros se quiere llegar a adaptar la eficiencia y robustez de la comunicación a una aplicación/escenario concreto, intentando evitar que los paquetes se pierdan por una señal débil. En este punto entra un factor denominado RSSI; se trata del parámetro que mide la fuerza de la señal, difiere entre 0 y -100, cuanto mayor es su valor, mayor es la intensidad de la señal recibida. La señal se mide en dBm donde el 0 equivale a 1mW. Una señal de -50 es ideal para realizar transferencias satisfactorias, una señal de -60 sigue siendo buena para transferencias con un margen de error del 20 %. Una vez bajamos este umbral la señal comienza a ser deficiente, -70 la transferencia puede no ser satisfactoria y por debajo de -80 puede llegar a tener problemas de conexión. RSSI muestra la intensidad de la señal ideal para realizar pruebas de cobertura.

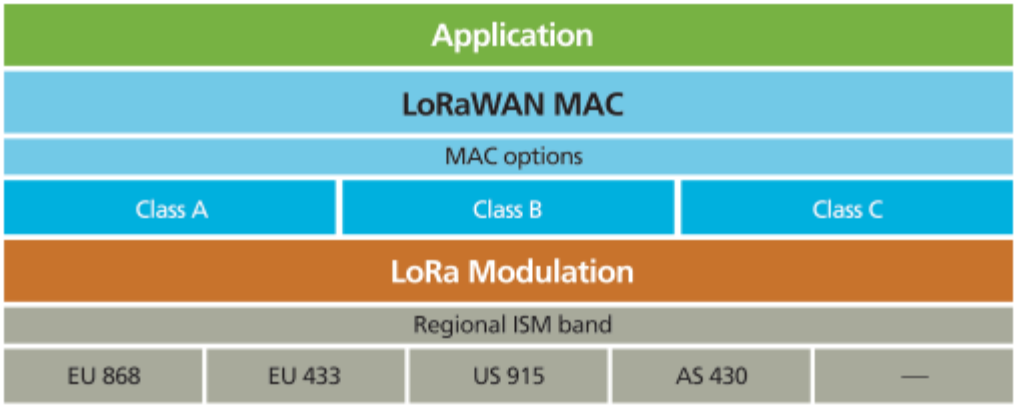

Figura 7. Ejemplo de paquete LoRaWAN.

Fuente: https://medium.com/pruebas-de-laboratorio-de-la-modulaci%C3%B3n-lora/lorawan-d00f48384160

Existen principalmente tres tipos de nodos en esta infraestructura:

- *Gateway*, encargado de recibir internet para transmitir los datos hacia fuera del sistema, y de recibir los datos de otro tipo de nodo.
- Sensores, que recolectan datos y los envían al *gateway* para su procesado.
- Actuadores, nodos que esperan una activación para realizar la tarea para la que están diseñados.

LoRaWAN sigue una topología en estrella, con una distancia máxima de transferencia de hasta 20 km; además, implementa específicamente parámetros de seguridad como encriptación nativa AES 128. Administrando los tres tipos de nodos que se han descrito previamente, y replicando las ventajas de la tecnología que trabaja por debajo, LoRa obtiene bajo consumo y largo alcance debido al corto ancho de banda y la transferencia de mensajes pequeños de 242 bytes.

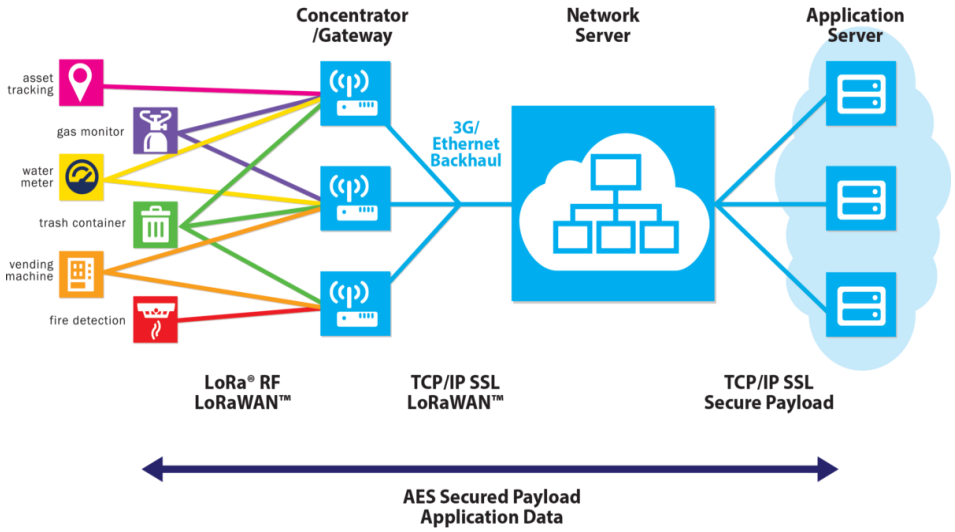

Figura 8. Arquitectura LoRa. Fuente: https://www.cooking-hacks.com/documentation/tutorials/lorawan-for-arduino-raspberry-pi-waspmote-868-900-915-433-mhz

En la Figura 8 se observa la conexión entre los diversos nodos que soporta una red LoRaWAN; en la parte izquierda están los sensores y los actuadores que se encargan de recibir y enviar datos a los *gateways*. Los *gateways* reenvían la señal a un sistema que procesará los datos capturados de los sensores y si es necesario mandará un mensaje para que los actuadores se activen, por último, los datos serán mostrados en una aplicación como servicio.

En resumen, LoRa es una tecnología flexible de largo alcance y bajo consumo. En definitiva, una buena herramienta para solucionar los problemas de la industria 4.0

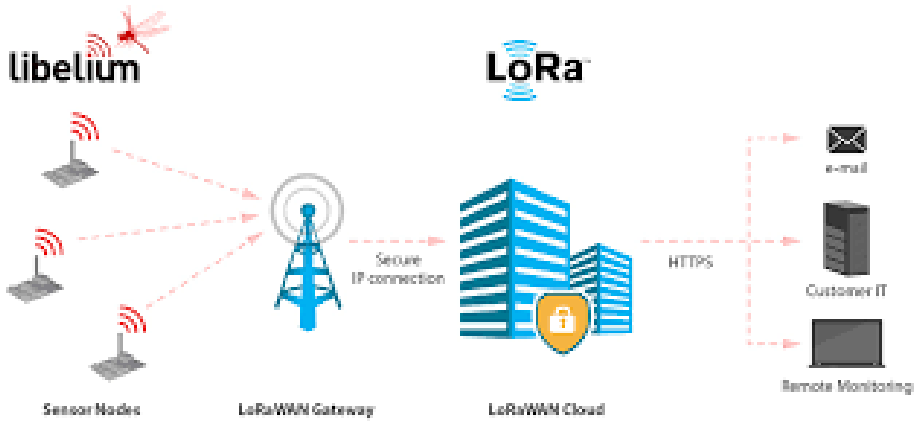

Figura 9. Ejemplo arquitectura LoRa Libelium. Fuente: https://www.cooking-hacks.com/documentation/tutorials/lorawan-for-arduino-raspberry-pi-waspmote-868-900-915-433-mhz

3.3. GPS

Se trata de un sistema de posicionamiento global (*Global Positioning System*). Es el encargado de proporcionar un sistema de navegación por satélite, soportado por una constelación de 24 satélites. Esta tecnología fue diseñada por el departamento de defensa de los Estados Unidos, en un principio colocando en órbita estos satélites con el fin militar, aunque convirtiéndose posteriormente en un sistema de navegación global sin ningún tipo de coste.

Los satélites tienen una órbita muy exacta, y transmiten información a la tierra. Los receptores de la señal son los encargados de procesar la información y triangular la posición. Este sistema se basa en calcular el tiempo que tarda la señal en llegar al receptor; procesando la información con varios satélites existe la posibilidad de geolocalizar el GPS con máxima precisión.

Las tramas de los GPS dependen del tipo de fabricante, siendo NMEA uno de los fabricantes más reconocidos por la comunidad GPS. La trama se basa en mostrar diferentes parámetros en forma de cadena de valores separados por comas. El comando que muestra la latitud, longitud y altura es GPRMC y el parámetro que muestra la cantidad de satélites de los cuales recibe señal es GPGSV.

```
     $GPGSV,2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,
45*75
     Where:
          GSV          Satellites in view
```

```
            2            Number of sentences for full data
            1            sentence 1 of 2
            08           Number of satellites in view

            01           Satellite PRN number
            40           Elevation, degrees
            083          Azimuth, degrees
            46           SNR - higher is better
                 for up to 4 satellites per sentence
            *75          the checksum data, always begins with *

      $GPRMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1
,W*6A
      Where:
           RMC          Recommended Minimum sentence C
           123519       Fix taken at 12:35:19 UTC
           A            Status A=active or V=Void.
           4807.038,N   Latitude 48 deg 07.038' N
           01131.000,E  Longitude 11 deg 31.000' E
           022.4        Speed over the ground in knots
           084.4        Track angle in degrees True
           230394       Date - 23rd of March 1994
           003.1,W      Magnetic Variation
           *6A          The checksum data, always begins with *
```

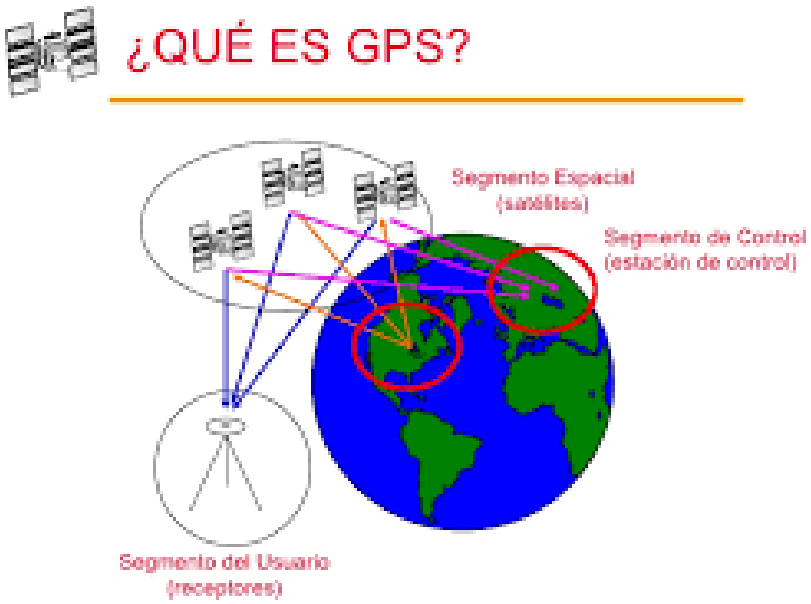

Figura 10. GPS conexión satélites. En la imagen se puede observar el funcionamiento de GPS mediante tres GPS, dando señal a una antena. Fuente: http://tecmultimed.blogspot.com/2016/08/gps.html

A la hora de calcular la posición es necesario que el receptor reciba la señal de cómo mínimo tres satélites, pudiendo así calcular una posición en 2D, es decir, se puede conocer la latitud y longitud del GPS. Si deseamos calcular otros parámetros como la altitud del receptor será necesario que otro satélite envíe información. Una vez se tiene un número superior a tres es posible calcular la posición en 3D, lo cual permite conocer más parámetros, como la altitud, velocidad, dirección…

Siendo un sistema donde la información se recibe de satélites en órbita, se puede calcular una posición en cualquier medio terrestre, ya puede ser tierra mar o aire. Como desventaja, no es posible enviar señal a interiores o zonas cerradas, como, por ejemplo, una cueva o un edificio con muros grandes. Esto hace que sea un sistema bastante usado por la mayoría de los dispositivos que utilizan la geolocalización, siendo muy potente a la hora de localizar en exteriores.

Los sistemas receptores de hoy en día son extremadamente precisos, pudiendo geolocalizar con un nivel de precisión extremo, centímetros. Actualmente se usan GPS de Garmin con capacidad WAAS (*Wide Area Augmentation System*), los receptores también pueden obtener una mejor precisión con el GPS diferencial (DGPS), que corrige las señales GPS, por último, corregir esta señal mediante una red de torres que reciben señales GPS y transmiten una señal corregida por los transmisores de baliza.

3.4. MongoDB

Se trata de una base de datos NoSQL, orientada a documentos. Estos documentos están en formato JSON, aunque internamente los guarda como BSON, es decir, JSON en formato binario. Los archivos JSON siguen una estructura de clave valor, donde realizar consultas es relativamente rápido. En el caso de este proyecto se usará para conseguir que los datos sean persistentes, aunque la aplicación deje de funcionar o se reinicie.

MongoDB es un software que puede escalar horizontalmente, es muy fluida a la hora de hacer inserciones y búsquedas simples (sin *joins*), sin soporte para *roll back*, por lo que es no es muy efectiva para realizar transacciones. En la aplicación, se usa MongoDB porque se requiere llevar a cabo inserciones y consultas fluidas; si alguna inserción o consulta falla, no es necesario volver a consultarla porque se colectan un número alto de datos en cada una de las posiciones geográficas, por lo que se le da prioridad a la fluidez.

La instalación de MongoDB será transparente para el usuario de nuestra aplicación, será instalada sobre una imagen Docker, evitando que el usuario tenga que instalar y mantener la aplicación.

Las principales características de MongoDB son:

- Consultas: MongoDB permite buscar dentro de sus documentos por campos, incluso realizar consultas más complejas para devolver un campo específico.
- Indexación: Todos los documentos almacenados en MongoDB tienen un índice para agilizar las consultas.
- Replicación: Para agilizar las consultas MongoDB permite dividir sus servidores en primarios y secundarios. El servidor primario es el encargado de recibir las escrituras y propagar los cambios al resto de servidores. Los servidores secundarios sirven para realizar consultar sobre ellos.
- Equilibrado de carga: Al poder tener servidores secundarios se puede dividir la carga de las consultas entre los servidores secundarios, evitando tener sobrecarga en el servidor principal.
- Almacenamiento de archivos: Es una base de datos no relacional, orientada a documentos, por lo tanto, es posible añadir documentos completos.
- Agregación: Existen numerosos *frameworks* para la resolución de consultas SQL.
- Ejecución de JavaScript en el servidor: Permite las consultas en JavaScript, lo cual facilita las consultas ya que las consultas con *joins* en MongoDB tienen cierta dificultad.

3.5. Docker

Docker es un software de virtualización y aislamiento de procesos a nivel de sistema operativo. Al contrario que las máquinas virtuales tradicionales, Docker no emula completamente las llamadas al sistema y por tanto no necesita de instrucciones especiales para mantener un rendimiento aceptable. Los procesos que se ejecutan bajo Docker corren directamente en el sistema operativo *host* sin software intermedio, si bien el *kernel* se encarga de aislar dichos procesos para que no interfieran con el resto del sistema.

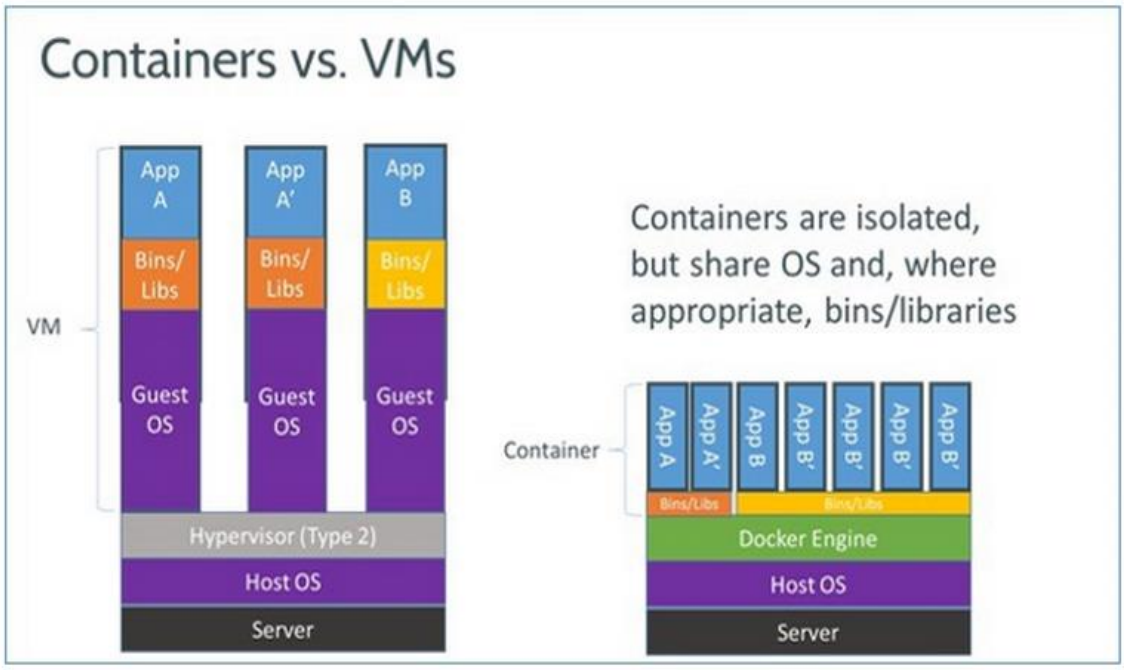

Figura 11. Diferencia entre máquinas virtuales tradicionales y Docker, como podemos ver docker funciona con el sistema operativo de la propia máquina y comparte los bins y librerías entre las aplicaciones del mismo tipo.

Fuente: http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

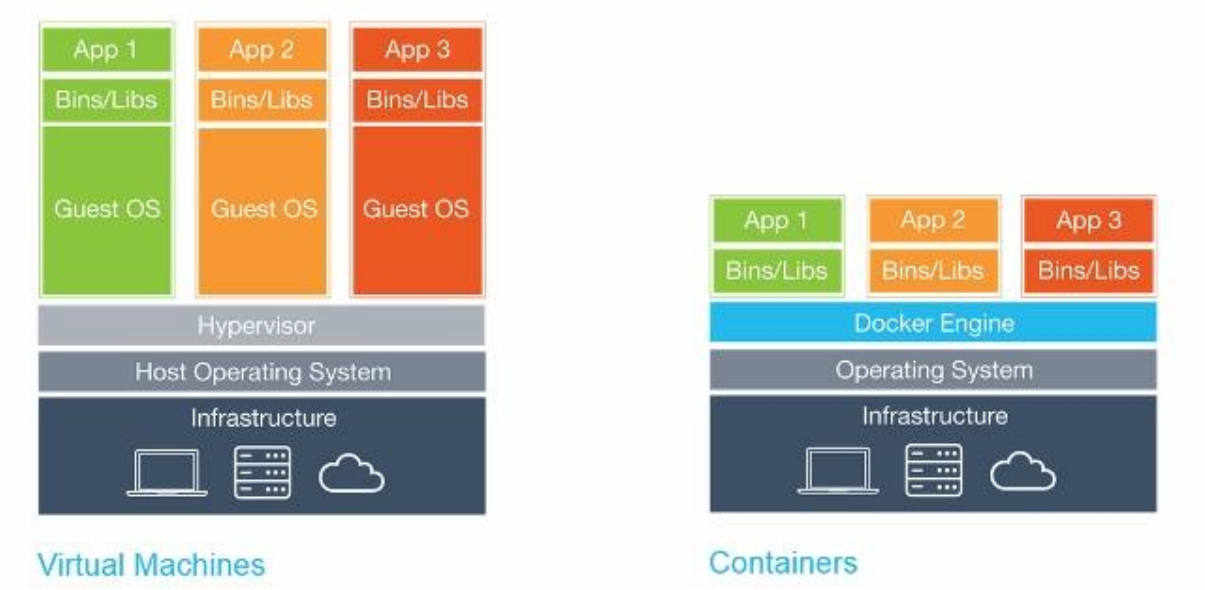

Figura 12. Diferencia entre máquinas virtuales tradicionales y Docker. En esta imagen al ser aplicaciones diferentes, la única diferencia existente es que Docker funciona con el sistema operativo de la máquina, que lo lanza. Fuente: https://www.docker.com/what-docker

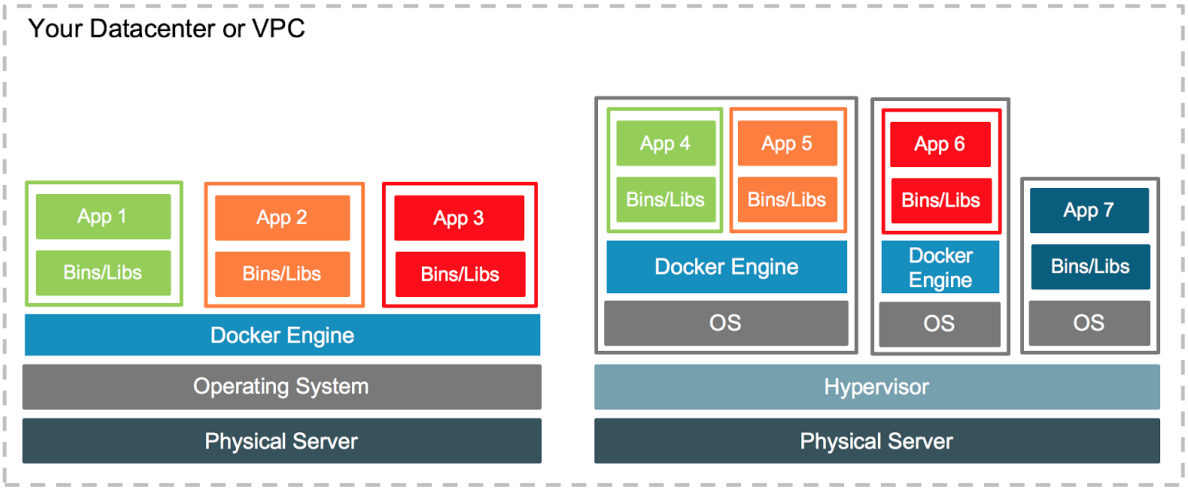

Figura 13. Uso conjunto de Docker y máquinas virtuales. En esta imagen se ve claramente la ventaja de que Docker se lance con el sistema operativo de la máquina. Fuente: https://blog.docker.com/2016/04/containers-and-vms-together/

Docker se gestiona a través de una API que puede ser invocada desde las herramientas de Docker en línea de comandos, desde bibliotecas creadas expresamente para diferentes lenguajes de programación, o bien realizando peticiones directamente al *endpoint* del servicio web que proporciona dicha API.

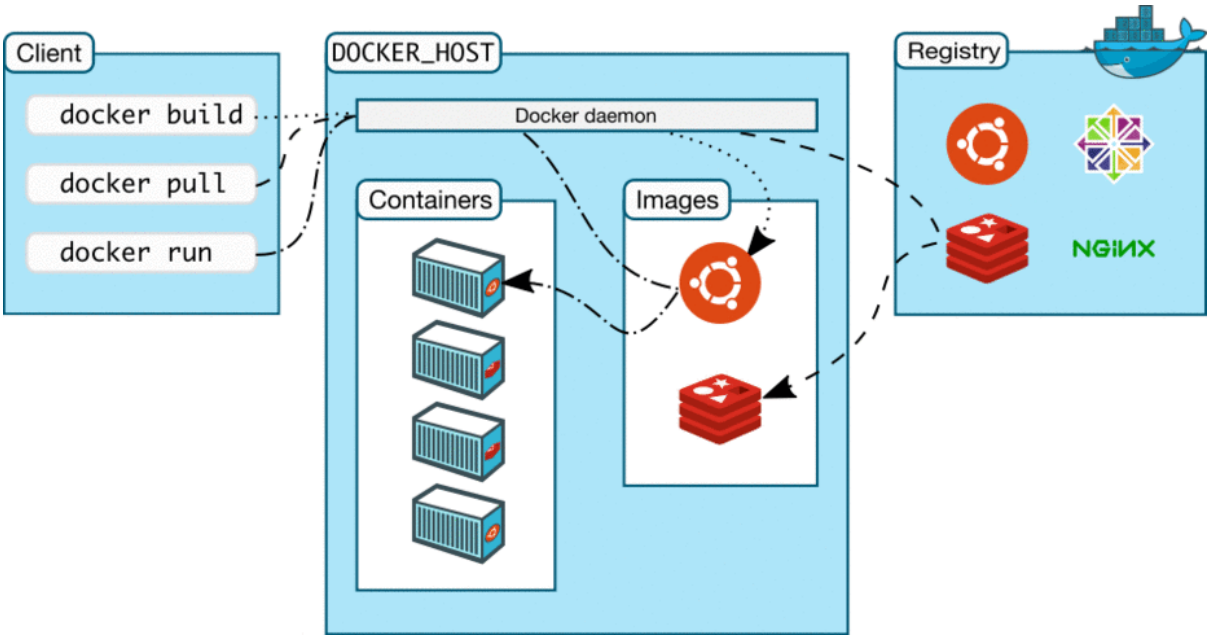

Figura 14. Arquitectura de Docker. Fuente: https://docs.docker.com/engine/docker-overview/

Docker sólo puede lanzar imágenes creadas previamente. Existe un repositorio de imágenes creadas por la comunidad (Docker Hub) y otros repositorios mantenidos por diferentes entidades, por ejemplo, Google. Cualquier usuario puede generar sus propias imágenes y subirlas a Docker Hub simplemente creando un archivo llamado Dockerfile y ejecutando unos sencillos comandos que generan a partir de este archivo la imagen en la máquina local, que luego puede subirse a un repositorio. Para crear una imagen se necesita un archivo Dockerfile. Este archivo contiene el sistema operativo y las herramientas que se indiquen. Para construir la imagen debemos hacer `docker build`. Una vez creado la imagen podemos subirla al repositorio haciendo `docker push`. Si la imagen está subida podemos descargarla con `docker pull`. Una vez tenemos la imagen se puede lanzar con `docker run` y realizar `commit` para guardar los cambios. La imagen siguiente explica estos pasos más gráficamente.

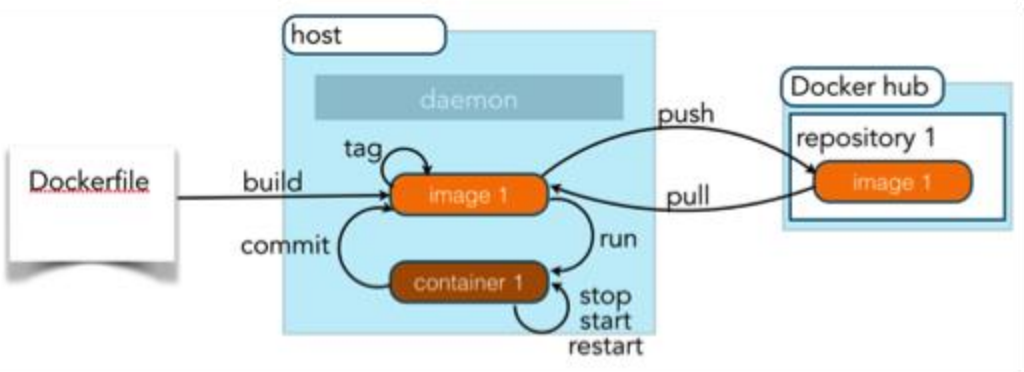

Figura 15. Uso de Docker images. Fuente: https://dzone.com/refcardz/getting-started-with-docker-1

3.6. NodeJS

NodeJS es un lenguaje que permite en tiempo de ejecución compilar JavaScript basado en eventos, por esta razón NodeJS tiene todo lo necesario para poder ejecutar JavaScript. En los principios JavaScript solamente podía ejecutar código en el navegador por lo que limitaba este lenguaje a páginas web dinámicas.

3.6.1. Arquitectura de NodeJS

Los desarrolladores de NodeJS incluyeron un compilador de JavaScript v8, haciendo mucho más ligero este lenguaje. Pasó a ser un lenguaje a bajo nivel capaz de ejecutar código sin necesidad de interpretarlo antes.

NodeJS utiliza un modelo de entrada salida no bloqueante y controlado por eventos creando un lenguaje fluido, por lo que es capaz de interactuar con el sistema de archivos, crear un servidor con una página web o realizar peticiones HTTP a una API, realizando estas operaciones de forma simultánea sin necesidad de que unas operaciones esperen a otras.

Existen diversos *frameworks* de desarrollo NodeJS, aunque nos centraremos exclusivamente en Express [9]. Express es el framework más conocido de NodeJS, siendo una ampliación del anterior *framework* "Connect" e inspirado en "Sinatra", y destacando por su sencillez y flexibilidad.

3.6.2. Framework Express

Debido a la usabilidad de este *framework*, su uso es una decisión muy habitual en el desarrollo de aplicaciones. Tiene además numerosas aplicaciones pre desarrolladas, destacando entre ellas:

1. Session Handler.
2. Grandes middlewares de terceros.
3. CookieParser y BodyParser.
4. Vhost.
5. Router.

Express presenta una sencilla estructura predefinida; dentro del proyecto se encuentra necesariamente el fichero packages.json donde indicamos las dependencias de la aplicación así como un fichero JavaScript donde se iniciará la aplicación. Varias carpetas, *public* y *controllers* donde estará el resto de las funciones de la aplicación, *routes*, donde estarán las rutas que se usarán, y *view*, que contendrá las vistas de la aplicación.

```
Nombre de la aplicación
     package.json
     server.js
     public
          javascripts
          images
      routes
          index.js
      view
          index.ejs
```

Como se ha comentado anteriormente, el archivo pakage.json es esencial debido a que contiene todas las dependencias del programa, indicando también el archivo que inicia la aplicación. Para iniciar el programa, se debe ejecutar desde la carpeta inicial donde se encuentran nuestras dependencias el comando:

```
npm install
```

Para una buena estructura interna de una aplicación Express, es importante conocer las bases del archivo principal, en nuestro caso server.js

```
var express = require("express"),
    app = express(),
    bodyParser  = require("body-parser"),
    methodOverride = require("method-override"),
    mongoose = require('mongoose');
```

En esta parte se define las librerías que se van a usar, la librería Express se asigna a una variable para poder usar sus funciones posteriormente.

3.6.3. Rutas Express

Las rutas son una parte muy importante de la aplicación, debido a que se puede definir que rutas pueden ejecutar cada uno de los módulos. Proporcionan una interfaz al usuario que consume la aplicación, y se definen como se detalla a continuación. Cuando un usuario ejecuta una operación GET en la ruta / responderá con un "Hello world".

```
//ROUTE
router.get('/', function(req, res) {
   res.send("Hello World!");
});
app.use(router);

// API routes
var datashows = express.Router();

datashows.route('/datashow').get(DataShowCtrl.findAllData)
;

app.use('/api', datashows);
```

Es aquí donde se definen aspectos muy importantes para el funcionamiento correcto de nuestra aplicación, comúnmente conocido como *middleware*. En este caso se indica que los *body* que recibiremos por peticiones POST serán tratadas como JSON, los cuales usarán codificación *urlencoded* y que podrán ser sobrescritos.

```
// Middlewares

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(methodOverride());
```

Por último, otra de las funciones de Express que resultan sencillas. Se puede comunicar por qué puerto escuchará la aplicación. A su vez se puede indicar un mensaje para comunicar por consola que el puerto está escuchando y que la aplicación se ha lanzado correctamente.

```
// Start server
app.listen(8082, function() {
  console.log("Node server running on
http://localhost:8082");
```

3.7. MQTT

MQTT (*Message Queue Telemetry Transport*) es uno de los protocolos más utilizados en Internet Of Things. Como su nombre tiene un funcionamiento muy similar a las colas. Está orientado a comunicaciones con un coste mínimo, ya que una de sus virtudes son los escasos recursos que consume, siendo un protocolo ideal para uso con sensores y actuadores, debido a que la mayoría de los sistemas empotrados tienen escasos recursos de CPU, RAM. Entre otras, sus ventajas son el consumo mínimo de ancho de banda, su rapidez y la posibilidad de procesar un alto índice de respuestas con respecto al resto de protocolos web; todo ello lleva a un consumo de batería (procesamiento) mínimo.

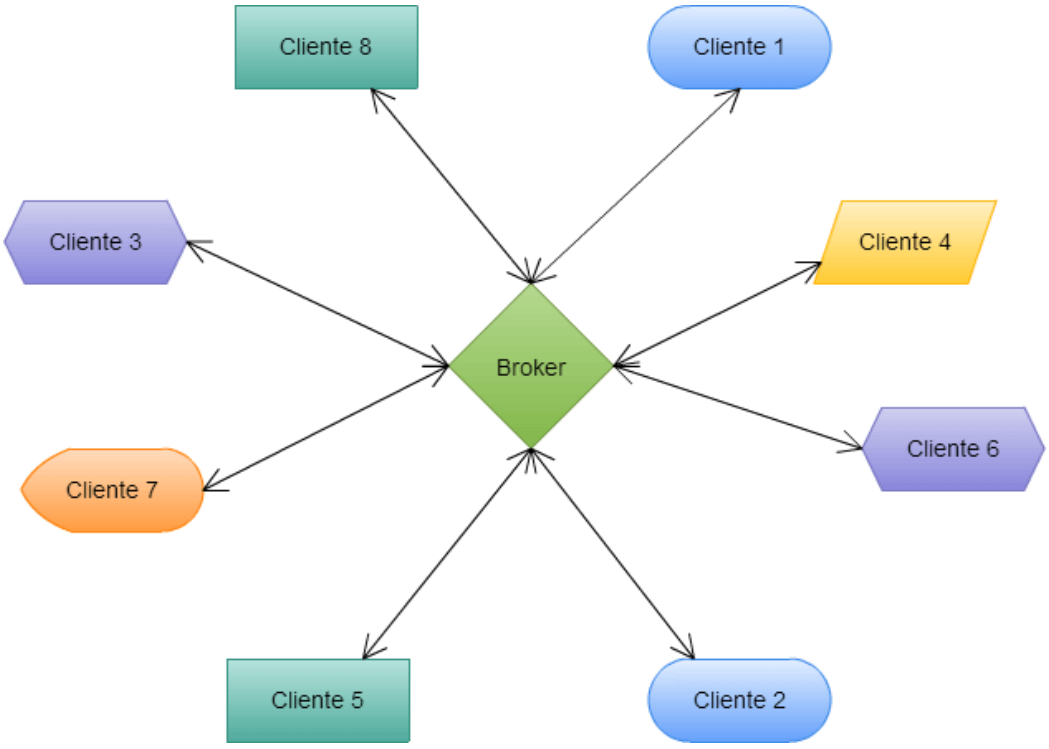

Figura 16. Ejemplo de arquitectura MQTT. Fuente: https://geekytheory.com/que-es-mqtt

Su arquitectura se basa en un *broker* o servidor, que centraliza todas las peticiones de los clientes en forma de estrella con una capacidad de miles de clientes. Este servidor es el encargado de gestionar todas las peticiones en base a *topics*, donde el cliente publica cada uno de sus temas y un *body* de como máximo de 242 Mb. Otros clientes pueden suscribirse a los *topics* y ver sus publicaciones en tiempo real. Un *topic* se representa mediante una cadena y tiene una estructura en forma de carpetas. Cada jerarquía se separa con "/" o incluso posibilidad de conectarse a todos los *topics* con "#", creando así jerarquías en forma de árbol.

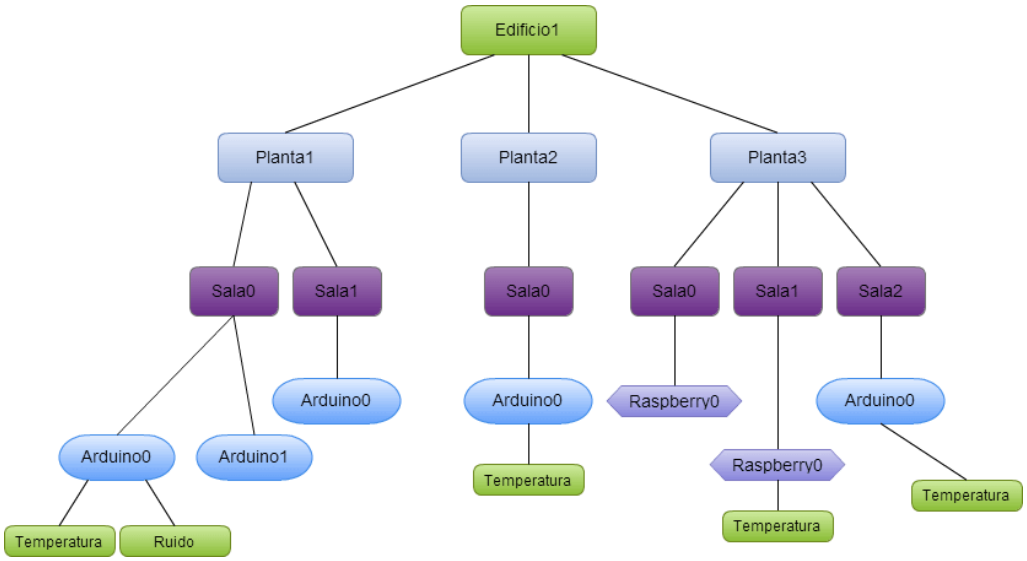

Figura 17. Ejemplo de topics MQTT, separados por una barra.

Fuente: https://geekytheory.com/que-es-mqtt

El *broker* debe tener constancia en todo momento de los clientes que tiene conectados, para que cada vez que un cliente publique todos los suscriptores sean avisados, esto lo realiza mediante "*pings*", mandando periódicamente un paquete (PINGREQ) y esperando la respuesta del *broker* (PINGRESP). La comunicación puede ser cifrada entre otras muchas opciones.

Los datos de IoT intercambiados pueden resultar muy críticos, por lo que es posible garantizar la seguridad de los intercambios en varios niveles:

- Transporte en SSL/TLS
- Autenticación mediante certificados SSL/TLS
- Autenticación mediante usuario y contraseña

El MQTT lleva integrada por defecto un protocolo de calidad QoS (*Quality of Service*, calidad de servicio). El *publisher* es el encargado de definir qué nivel de seguridad desea para la conexión. Hay tres niveles posibles:

- Un mensaje de QoS nivel 0 se entrega el paquete solo una vez y el *broker* no manda mensaje de confirmación.
- Un mensaje de QoS nivel 1 se entrega el paquete tantas veces como sea posible hasta que el *broker* responda que ha recibido el mensaje.
- Un mensaje de QoS nivel 2 se entrega tantas veces sea posible hasta recibir confirmación por parte del *broker*, añadiendo registros para evitar mensajes duplicados.

3.8. Ansible

Ansible es un software escrito en Python, cuya finalidad es el provisionamiento de máquinas y sigue una arquitectura de maestro esclavo:

- El maestro es el único que debe que tener instalado Ansible, mediante SSH se conecta con el resto de las máquinas y ejecuta los comando en Python que anteriormente ha traducido de los *yaml* de Ansible.
- El esclavo debe tener instalado Python 2.7 o superior.

Ansible se desarrolla en *yaml* y tiene tres capas para el diseño del código:

1. Hosts: Debe tener especificado las *IPs* de las máquinas donde tendrá que ejecutarse.

```
local:
  hosts:
    192.188.1.40
```

2. *Playbook*: Es el centro de la arquitectura, recoge los parámetros de la aplicación y ejecuta los roles con el usuario indicado.

```
- hosts: "local"
  vars:
    fc: "{{ fc }}"
    sf: "{{ sf }}"
    bw: "{{ bw }}"
    cr: "{{ cr }}"
  roles:
  - { role: configure-raspi }
```

3. Roles: Se trata de cada uno de los módulos que se van a ejecutar.

```
- name: "Configure Raspberry"
  become: true
  become_user: pi
  shell: "{{ command }} && {{ make }} SPREADING_FACTOR={{ sf }}
FRECUENCY={{ fc }} BANDWIDTH={{ bw }} CODING_RATE={{ cr }}"
  register: output
- name: "Print output"
  debug:
    msg: "{{ output }}"
```

3.9. Node-Red

Node-red es motor de flujos, principalmente utilizado en IoT. Permite crear flujos dentro de una interfaz gráfica. Los flujos pueden estar conectados mediante protocolos comunes como UDP, TCP, Websocket, MQTT, REST, AMQP. Además, está disponible la integración con aplicaciones de terceros, como puede ser Twitter, Mail, Slack. Es una interfaz sencilla y muy intuitiva que trabaja con NodeJS, permite la inserción de librerías de terceros, por lo que si una librería no está disponible se puede aportar. Los flujos pueden ser exportados en formato JSON,

En la parte izquierda de la aplicación aparecen los nodos instalados, que pueden usarse de entrada y salida. Se pueden descargar nuevos nodos para una mayor complejidad del flujo. Se arrastran los nodos hacía el flujo para usarlos y conectarlos mediante su punto, derecha indica nodo de envía parámetros e izquierda indica nodo que recibe parámetros.

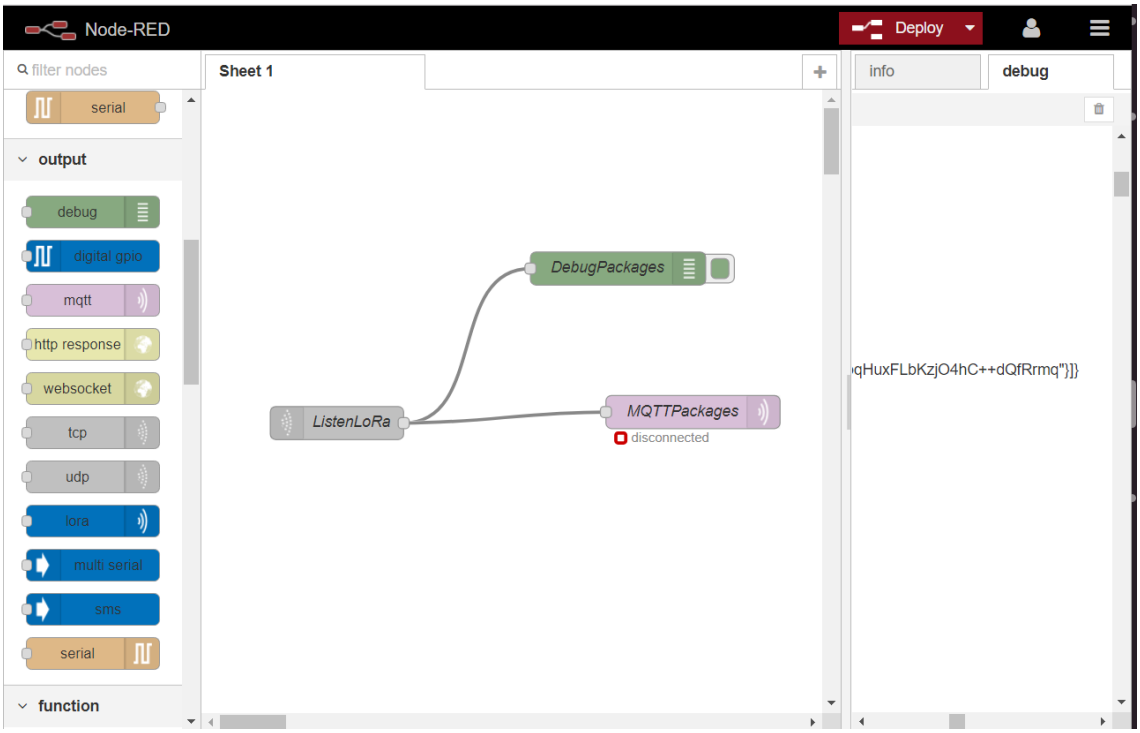

Figura 18. Ejemplo Node-red. Se puede observar un flujo de Node-Red.

4. Decisiones arquitectónicas

En este apartado se realizará una descripción de la arquitectura que se ha diseñado para este sistema:

- *Publisher*: Se trata principalmente de hardware encargado de recolectar los datos y enviarlos al *gateway* mediante LoRa.
- *Subscriber*: Se trata del apartado más software donde se suscribirá al *broker* y recolectará los datos, mostrándolos gráficamente en un mapa de calor.
- *Gateway*: Se trata de un *gateway* capaz de recibir una señal LoRa y publica los parámetros de la señal recibida a un *broker* MQTT con un *topic*.
- Configuración automática Raspberry: Se trata de ejecutar un *playbook* mediante la interfaz gráfica para la configuración de los parámetros LoRaWAN que emitirá la Raspberry.

4.1. Publisher

En este punto se describirá la parte del *Publisher*. El hardware esencial es una Rasberry PI 3, donde instalaremos como sistema operativo Raspbian. Se trata de un sistema operativo Linux Alpine (versión de Linux base con instalaciones mínimas), para el correcto funcionamiento de la aplicación deberemos deshabilitar el módulo GPS que viene por defecto en Rapsbian, la conexión del GPS con la Raspberry se hará mediante GPS. Por otro lado, mediante GPIO conectaremos una placa Cooking Hack [10] para dar soporte a una antena LoRa de Libelium.

El sistema funcionará mediante un script en que se consulta la posición actual mediante GPS y se enviará un paquete *join* LoRa, con los datos de la posición desde la que se envía dicho paquete.

Paquete *Join*

Existe 8 tipos de paquetes dentro de LoRa, se ha decidido utilizar un *join*. Envía un paquete broadcast mediante LoRa para solicitar la incorporación a una red LoRaWAN. Esto nos posibilitará la recolección de datos. En el receptor del mensaje, se puede observar diversos

parámetros como el *body* (donde se incluye la geolocalización), el ruido, el ancho de banda, la intensidad, el nombre del dispositivo, tasa de codificación y frecuencia.

4.2. Gateway

Se trata de un *gateway* encargado de recibir los paquetes *join* enviados por el *susbcriber*; este *gateway* está diseñado para que todo el tráfico de la señal LoRa que reciba lo propague mediante MQTT a un *broker* previamente diseñado. Finalmente, los datos serán consultados por el *subscriber* mediante un *topic* específico.

Topic

Dentro de la arquitectura de MQTT, es muy importante el concepto "*topic*" o "tema" ya que a través de estos "*topics*" se articula la comunicación, puesto que emisores y receptores deben estar subscritos a un "*topic*" común para poder entablar la comunicación. Este concepto es prácticamente el mismo que se emplea en colas, donde existen varios publicadores (que publican o emiten información) y unos subscritores (que reciben dicha información) siempre que ambas partes estén subscritas a la misma cola.

4.3. Subcriber

Esta parte tiene más complejidad software, y ha sido diseñada completamente sobre Docker y orquestado por Docker Compose, para facilitar al usuario el uso de esta aplicación. Los pasos principales de la parte suscriptora son:

- Se creará un *broker* MQTT para que el *gateway* pueda enviar los datos.
- Se ha diseñado una aplicación en NodeJS donde se suscribirá al *topic* que el *gateway* ha creado.
- A su vez recibidos los datos bajo suscripción, los almacenará en un base de datos NoSQL, MongoDB.

Una vez los datos son persistentes en la base de datos con NodeJS Express se crea un servidor web, donde se leerán los datos de la base de datos y serán utilizados para mostrarlos en

un HeatMap. En este caso se ha utilizado Google HeatMaps, un servicio en JavaScript, que utiliza los mapas de Google para realizar HeatMaps sobre ellos.

HeatMap

Un HeatMap (o 'mapa de calor') es un gráfico en el que se resaltan mediante un código de colores zonas concretas de una web en base a criterios como el número de clics, o las áreas por las que pasa con más frecuencia el puntero.

El objetivo de ello es obtener del comportamiento de los navegantes de nuestra web datos útiles para mejorar aspectos de esta, comprobar la visibilidad de ciertos elementos, o evaluar los mejores sitios para poner publicidad.

4.4. Configuración automática Raspberry

En esta sección se configurarán los parámetros LoRaWAN mediante una aplicación web, que ejecuta un *playbook* de Ansible, conectándose mediante SSH con la Raspberry para modificar la ejecución de la aplicación de envío de paquetes LoRa.

5. Implementación

En este apartado se defenderán las decisiones sobre la arquitectura y las tecnologías que se han usado para el desarrollo de la aplicación.

5.1. Publisher

La parte del *Publisher*, el cual actúa recolectando datos tiene como hardware principal una Raspberry PI 3, de 1 GB de SRAM, memoria con tarjeta SSD, 4 puertos USB, HDMI, conexión a internet y conexión mediante GPIO.

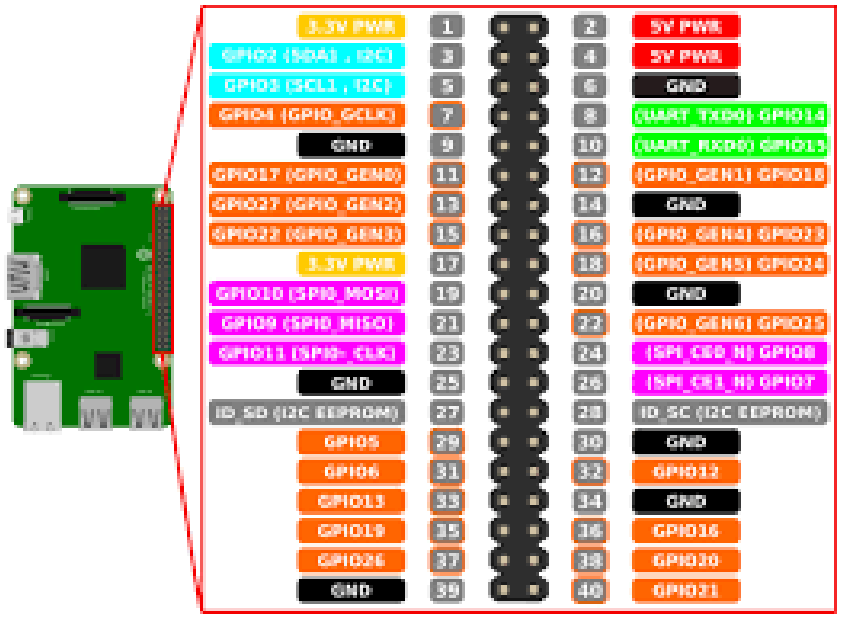

Figura 19. Puertos GPIO Raspberry PI 3

Fuente: https://docs.microsoft.com/en-us/windows/iot-core/learn-about-hardware/pinmappings/pinmappingsrpi

Posteriormente se utilizarán dos de las entradas y salidas (GPIO y USB), que dispone la Raspberry. Se conectará mediante GPIO una placa de Cooking Hack, colocando esta placa de intermediaría entre la Raspberry y la antena LoRa. Esta placa está diseñada por Libelium, una de las compañías más reconocidas de España situada en Zaragoza dedicada a productos IoT. Compatible con Raspberry y con la posterior antena que se va a utilizar, la funcionalidad de esta placa es de enlace entre las entradas salidas de la Raspberry y la antena LoRa.

La antena LoRa es de Microchip, empresa americana dedicada también a productos IoT, tiene conexión para dos antenas con distinta frecuencia, la primera de ellas trabaja a frecuencia LoRa europea y la segunda a frecuencia LoRa americana. Se puede ver su descripción en http://ww1.microchip.com/downloads/en/DeviceDoc/50002346C.pdf

La comunicación es mediante GPIO, como también tiene la Raspberry, debido a que las dimensiones de los pines son diferentes, hay que poner un bridge intermedio para que la conexión sea posible.

La arquitectura del chip de la antena LoRa es:

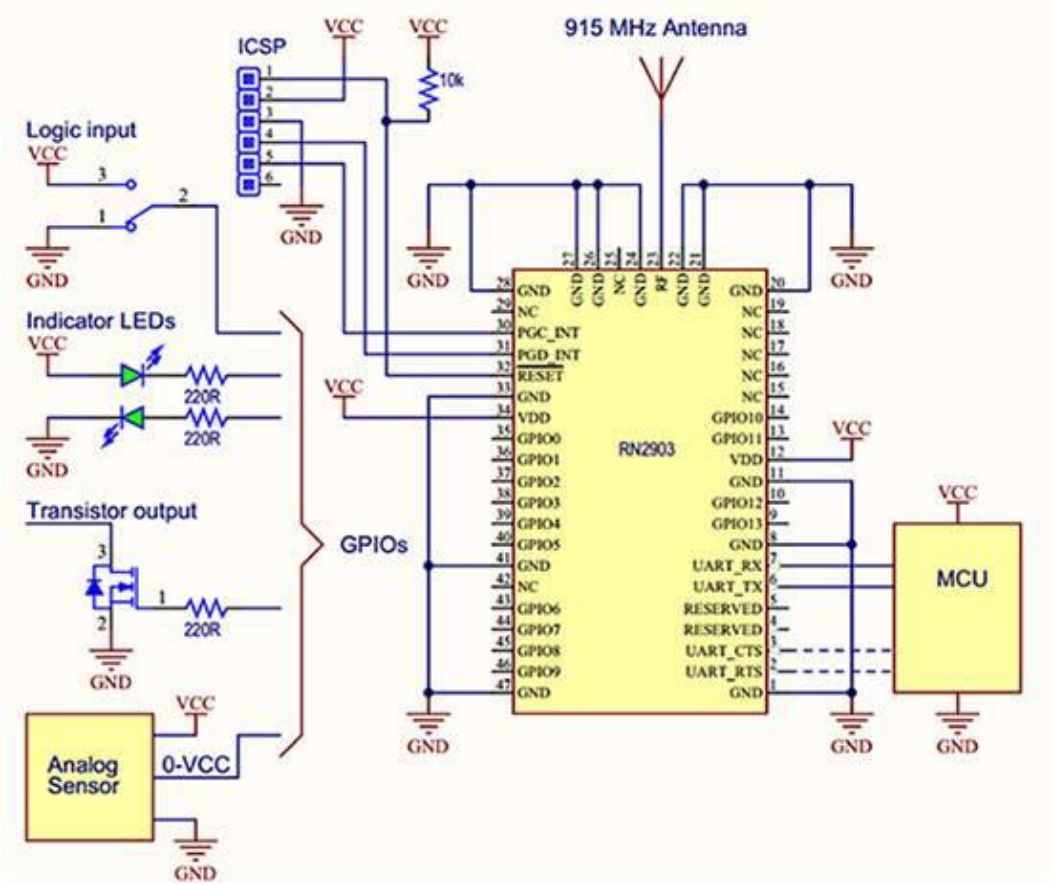

Figura 20. Arquitectura de chip LoRa.

Figura 21. Imagen del Publisher Raspberry pi 3 con antena LoRa

Fuente: https://www.cooking-hacks.com/documentation/tutorials/extreme-range-lora-sx1272-module-shield-arduino-raspberry-pi-intel-galileo

Con estas tres piezas hardware se puede disponer de un sistema que permite enviar un mensaje LoRa en cualquier medio, para que la alimentación del sistema sea dinámica se necesitará una *Power Ball* de 5V de salida de 3000 mAh.

- Para el envío de los mensajes LoRa se debe instalar sobre la Raspberry diversos softwares. Entre ellos una aplicación en C, la cual se puede consultar en https://github.com/JoniSanchez/heatmap_lora_signal/tree/master/cooking_hack
- Para poder compilar esta aplicación dentro de la Raspberry, se debe instalar un compilador C con las librerías `libgps`.

Este software se divide en dos partes, la primera de ellas donde se conocerá la posición de la Raspberry mediante GPS para lo que se debería desconectar el GPS por defecto para poder usar el nuevo GPS que tiene la antena. Para habilitar la antena mediante UART tenemos que seguir los siguientes pasos.

```
#Editar el inicio de la raspberry
sudo nano /boot/cmdline.txt
      dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
      rootfstype=ext4 elevator=deadline rootwait
#Detenemos los servicios GPS por defecto de Rasberry PI
sudo systemctl stop serial-getty@ttyAMA0.service
sudo systemctl disable serial-getty@ttyAMA0.service
sudo shutdown -r now
#Se instala GPS para poder visualizar los resultados
sudo apt-get install gpsd gpsd-clients python-gps
#Paramos los sockets de GPS ahora activos
sudo systemctl stop gpsd.socket
sudo systemctl disable gpsd.socket
#Se crea el nuevo socket GPS
sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock
#Se detiene para poder volverlo a lanzar
sudo killall gpsd
sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
#Se definen nuevos parámetros para GPS (probar por defecto anteriormente)
sudo nano /etc/default/gpsd
      START_DAEMON="true"
      GPSD_OPTIONS="/dev/ttyAMA0"
      DEVICES=""
      USBAUTO="true"
      GPSD_SOCKET="/var/run/gpsd.sock"
#Lanzamos la librería para obtener los resultados
cgps -s
```

La salida que se debe obtener tras el último comando del script y unos minutos de espera para la conexión con los satélites de GPS. Para una buena comunicación es necesario conectarse con mínimo tres satélites:

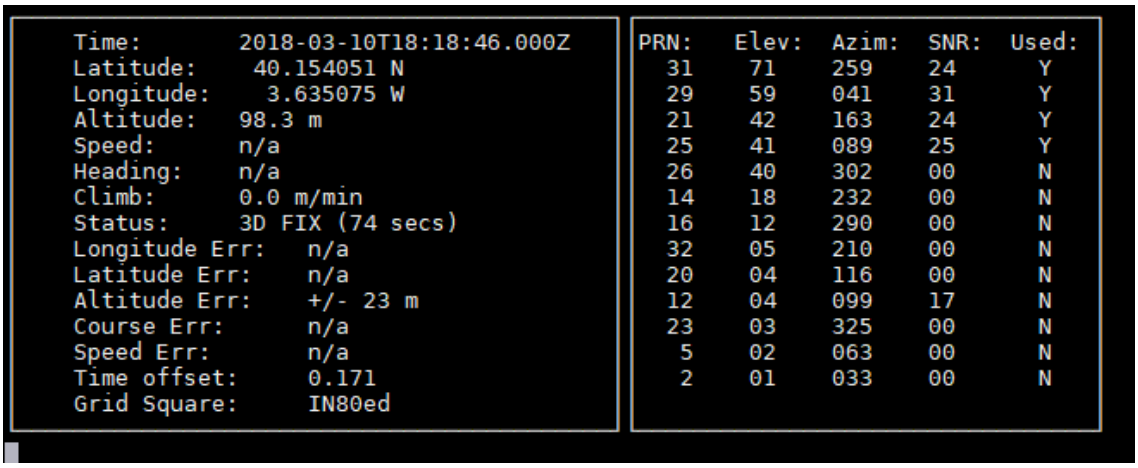

Figura 22. Salida GPS, en esta imagen se puede ver a la izquierda los parámetros base como la altitud, longitud, altura velocidad y a la derecha los parámetros de los satélites conectados.

La segunda parte será utilizar un programa en C para poder recolectar los datos que realmente se necesitan, estos son latitud, longitud para conocer la geolocalización. La librería que se ha decidido escoger para recolectar estos parámetros es `libgps`. Esta librería se instala en Raspbian con:

```
sudo apt-get install libgps-dev
```

Y para compilar el código que se nombrará más adelante se utiliza:

```
gcc -o gps libgps.c -lm -lgps
```

El programa consiste en consultar el socket de GPS cada 2 segundos, si falla 10 veces seguidas termina el programa con error y si tiene buen resultado 3 veces termina sin errores. Con esto conseguimos que el bucle no sea infinito y podamos ejecutarlo cada minuto. Si deseamos que este continuamente colectando datos simplemente tendremos que añadir un bucle infinito "`while (true)`".

Programa C:

```
#include <gps.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

int main() {
int rc;
struct timeval tv;

struct gps_data_t gps_data;
if ((rc = gps_open("localhost", "2947", &gps_data)) == -1) {
    printf("code: %d, reason: %s\n", rc, gps_errstr(rc));
    return EXIT_FAILURE;
}
gps_stream(&gps_data, WATCH_ENABLE | WATCH_JSON, NULL);

while (1) {
    if (gps_waiting (&gps_data, 2000000)) {
        /* read data */
        if ((rc = gps_read(&gps_data)) == -1) {
            printf("error occured reading gps data. code: %d, reason: %s\n",
rc, gps_errstr(rc));
        } else {.
            if ((gps_data.status == STATUS_FIX) &&
                (gps_data.fix.mode == MODE_2D || gps_data.fix.mode ==
MODE_3D) &&
                !isnan(gps_data.fix.latitude) &&
                !isnan(gps_data.fix.longitude)) {
                    //gettimeofday(&tv, NULL);
                    printf("latitude: %f, longitude: %f, altitude: %f, speed:
%f,    timestamp:    %lf\n",    gps_data.fix.latitude,    gps_data.fix.longitude,
gps_data.fix.altitude, gps_data.fix.speed, gps_data.fix.time);
            } else {
                printf("no GPS data available\n");
            }
        }
    }
```

```
        sleep(3);
    }

    /* Cerrar */
    gps_stream(&gps_data, WATCH_DISABLE, NULL);
    gps_close (&gps_data);

    return EXIT_SUCCESS;
    }
```

Se ha desarrollado un script que se ejecutará periódicamente. Para ello se ha automatizado en proceso con un cron. Para la instalación se han seguido los siguientes pasos:

```
Instalamos.
sudo apt-get install gnome-schedule
Editamos el archivo.
crontab -e
Añadimos el script dentro del archivo.
* * * * * /home/pi/gps >> /home/pi/prueba.txt
Ver que se ha creado el cron.
sudo crontab -l
```

Este cron ejecuta el programa y guarda la salida en un archivo. El archivo tiene estos datos:

latitude: 76.432423, longitude: -3.633310, altitude: 548.100000, speed: 0.436000, timestamp: 1521399842.000000

En la siguiente parte se debe crear un programa que mande mensajes mediante LoRa, para ello se utilizará la librería que proporciona Cooking Hack que ha creado un repositorio para la descarga del código donde podemos consultar las aplicaciones:

http://www.cooking-hacks.com/media/cooking/images/documentation/tutorial_kit_lorawan/arduPi_api_LoRaWAN_v1_3.zip

Se accede a `'\cooking\examples\LoRaWAN'` para poder compilar un ejemplo se debe lanzar cook.sh y el '.cpp' que se desea compilar. Por ejemplo, para enviar un mensaje en LoRa se

deberá compilar el archivo `LoRaWAN_P2P_02_send_packets.cpp` y `LoRaWAN_P2P_01_configure_module.cpp` para que el módulo Cooking se configure con la Raspberry. Al lanzar el script se enviarán paquetes mediante el protocolo LoRa.

Hay varios parámetros en la comunicación LoRa que se pueden modificar para obtener resultados. Para ello podemos cambiar el ancho de banda, la frecuencia, la codificación y el factor de propagación.

- Ancho de banda: LoRa tiene un ancho de banda alto que nos permite reducir el consumo de nuestro dispositivo y aumenta la resistencia contra la energía electromagnética.
- Frecuencia: En Europa la frecuencia que trabaja LoRa es 867-869 MHz, se trata del estándar que podemos modificar para obtener mejores resultados a la hora de enviar paquetes mediante este protocolo.
- Código de corrección: De error directo es la proporción de la secuencia de datos que es útil. Es decir, si la tasa de codificación es k / n, para cada k bits de información útil, el codificador genera un total de n bits de datos, de los cuales n-k son redundantes.
- Factor de propagación: Duración del armónico. LoRa opera con factores de propagación de 7 a 12. SF7 es el tiempo más corto en el aire, SF12 será el más largo. Cada paso en el factor de expansión duplica el tiempo en el aire para transmitir la misma cantidad de datos.

Con el mismo ancho de banda, un mayor tiempo en el aire da como resultado menos datos transmitidos por unidad de tiempo.

Para poder cambiar estos argumentos a la hora de ejecutar nuestro script de Cooking Hacks debemos modificar el programa, el cual quedaría de esta forma:

```
int main(int argc, char *argv[]){
   int rc; struct timeval tv; struct gps_data_t gps_data;  char aux[40] = "";  char params[10] = "";
   if (argc != 5){
      printf("Error en la entrada de argumentos.\n");
      exit(1);}
   frequency = atoi(argv[2]);
```

```
        strncpy(spreading_factor, argv[1], 10);
        strncpy(coding_rate, argv[4], 10);
        bandwidth = atoi(argv[3]);
        setup();
        while(1){// Catch data}
        return (0);}
```

Una vez modificados los parámetros de entrada, se ha creado un *Makefile*.

```
        ##Variables
        SPREADING_FACTOR?=sf12
        FRECUENCY?=868100000
        BANDWIDTH?=125
        CODING_RATE?=4/5
        # Frist compile to Lora
        deploy-configure-lora:  ## Frist compile to Lora
            ./cook.sh LoRaWAN_P2P_01_configure_module.cpp
            sudo   ./LoRaWAN_P2P_01_configure_module.cpp_exe   ${SPREADING_FACTOR}
${FRECUENCY} ${BANDWIDTH} ${CODING_RATE}
        # Send Packages Lora with GPS
        send-packages-lora-gps:  ## Send Packages Lora with GPS
            ./cook.sh LoRaWAN_GPS.cpp
            sudo ./LoRaWAN_GPS.cpp_
```

Por último, se ha creado un programa que puede utilizar las dos aplicaciones anteriores, por un lado, el programa en C, del cual se obtenían la localización mediante GPS. Y, por otro lado, el programa de Cooking Hack que envía paquetes mediante LoRa. Al estar los dos programas en C no se tendrá problemas de lenguaje.

Se mantendrá la estructura de Cooking Hacks, teniendo un script para ejecutar el código. Este script `cook.sh` será modificado, añadiendo a la hora de compilar la librería `libgps` para que conecte también Con GPS.

Se ha introducido en el código de Cookig Hack los comando para consultar el socket de GPS. Quedando de esta forma:

```
int main(int argc, char *argv[]){
// Anterior
    if ((rc = gps_open("localhost", "2947", &gps_data)) == -1) {
        printf("code: %d, reason: %s\n", rc, gps_errstr(rc));
        return EXIT_FAILURE;
    }
    gps_stream(&gps_data, WATCH_ENABLE | WATCH_JSON, NULL);
    setup();
    while(1){
        // Catch data
        if (gps_waiting (&gps_data, 2000000)) {
            /* read data */
            if ((rc = gps_read(&gps_data)) == -1) {
                printf("error occured reading gps data. code: %d, reason: %s\n", rc, gps_errstr(rc));
            } else {
                /* Display data from the GPS receiver. */
                if ((gps_data.status == STATUS_FIX) &&
                    (gps_data.fix.mode == MODE_2D || gps_data.fix.mode == MODE_3D) &&
                   !isnan(gps_data.fix.latitude) &&  !isnan(gps_data.fix.longitude)) {
                    sprintf(params, "%f", gps_data.fix.latitude);
                    strncat(params, ";", 20);
                    strncpy(aux, params, 20);
                    sprintf(params, "%f", gps_data.fix.longitude);
                    strncat(params, ";", 20);
                    strncat(aux, params, 20);
                    sprintf(params, "%f", gps_data.fix.altitude);
                    strncat(params, ";", 20);
```

```
                    strncat(aux, params, 20);
                    str2hex(aux, data);
                    printf("\n%s\n", data);
                      //gettimeofday(&tv, NULL); EDIT: tv.tv_sec isn't actually the timestamp!
                    printf("latitude: %f, longitude: %f, altitude: %f, speed: %f, timestamp: %lf\n", gps_data.fix.latitude, gps_data.fix.longitude, gps_data.fix.altitude, gps_data.fix.speed, gps_data.fix.time); //EDIT: Replaced tv.tv_sec with gps_data.fix.time
                } else {
                  printf("no GPS data available\n");// FIN
```

Analizando el código se puede observar que el envío de datos dentro de un paquete de LoRa se hace en hexadecimal. Se ha decidido que el formato de envío sean las coordenadas y la altura separadas por ";". Para ello se ha diseñado una librería que pase de *string* a hexadecimal. Todo este código está en GitHub, se añade el enlace al final de la memoria.

5.2. Gateway

La siguiente parte por diseñar es la configuración del *gateway* que permita conexiones LoRa. Se ha decidido utilizar el *gateway* Laird Sentrius RG1xx. Un dispositivo configurable, manejable y escalable diseñado para IoT dotado de una antena LoRa, equipado con una aplicación embebida que puede publicar con un nodo MQTT en un *broker* con un determinado *topic* "Gateway/MAC".

El *gateway* puede ser configurado como *forwarder* de mensajes LoRa, reenviando los mensajes a un determinada IP y puerto.

Para probar la conexión LoRa del *gateway* se pueden enviar mensajes a este con el programa anterior.

```
LoRaWAN_07_channels_frequency.cpp         LoRaWAN_Template.cpp_exe
pi@raspberrypi:~/Microchip/cooking/examples/LoRaWAN $ sudo ./LoRaWAN_P2P_01_configure_module.cpp_exe
Radio P2P example - Module configuration

1. Switch ON OK
-------------------------------------------------------
2. P2P mode enabled OK
-------------------------------------------------------
3.1. Set Radio Power OK
3.2. Get Radio Power OK. Power: 15
-------------------------------------------------------
4.1. Set Radio Frequency OK
4.2. Get Radio Frequency OK. Frequency: 868100000
-------------------------------------------------------
5.1. Set Radio SF OK
5.2. Get Radio SF OK. Spreading Factor: sf12
-------------------------------------------------------
6.1. Set Radio CR OK
6.2. Get Radio CR OK. Coding Rate: 4/5
-------------------------------------------------------
7.1. Set Radio BW OK
7.2. Get Radio BW OK. Bandwidth: 125
-------------------------------------------------------
8.1. Set Radio CRC mode OK
8.2. Get Radio CRC mode OK. CRC status: 1
-------------------------------------------------------
-------------------------------------------------------
Now the LoRaWAN module is ready for P2P communications.
The user must keep in mind that every time the module is
switched on, the radio settings MUST be set again.
Please check the next examples...
-------------------------------------------------------
```

Figura 23. Envío de paquete LoRa. Se puede ver en la imagen como va insertando en el paquete los diferentes parámetros insertados, como la frecuencia, la tasa de codificación.

Lanzando los paquetes con la localización:

```
--> Packet sent OK
40.153042;-3.633300;nan;
34302E3135333034323B2D332E3633333330303B6E616E3B
latitude: 40.153042, longitude: -3.633300, altitude: nan, speed: nan, timestamp: nan
--> Packet sent OK
40.153042;-3.633300;nan;
34302E3135333034323B2D332E3633333330303B6E616E3B
latitude: 40.153042, longitude: -3.633300, altitude: nan, speed: nan, timestamp: nan
--> Packet sent OK
40.153041;-3.633302;603.916000;
34302E3135333034313B2D332E3633333330323B3630332E3931363030303B
latitude: 40.153041, longitude: -3.633302, altitude: 603.916000, speed: 0.212000, timestamp: 1529964295.000000
--> Packet sent OK
40.153042;-3.633303;604.206000;
34302E3135333034323B2D332E3633333330333B3630342E3230363030303B
latitude: 40.153042, longitude: -3.633303, altitude: 604.206000, speed: 0.071000, timestamp: 1529964296.000000
--> Packet sent OK
40.153043;-3.633304;604.468000;
34302E3135333034333B2D332E3633333330343B3630342E3436383030303B
latitude: 40.153043, longitude: -3.633304, altitude: 604.468000, speed: 0.118000, timestamp: 1529964297.000000
--> Packet sent OK
40.153043;-3.633303;604.359000;
34302E3135333034333B2D332E3633333330333B3630342E3335393030303B
latitude: 40.153043, longitude: -3.633303, altitude: 604.359000, speed: 0.110000, timestamp: 1529964298.000000
--> Packet sent OK
40.153042;-3.633303;604.271000;
34302E3135333034323B2D332E3633333330333B3630342E3237313030303B
latitude: 40.153042, longitude: -3.633303, altitude: 604.271000, speed: 0.148000, timestamp: 1529964299.000000
```

Figura 24. Salida de paquete LoRa. En la imagen se puede ver el envío de varios paquetes LoRa y el contenido de su data.

Consultando el *debug* de NodeJS, se deberán observar los mensajes LoRa que se reciben en el Node-Red del *gateway*.

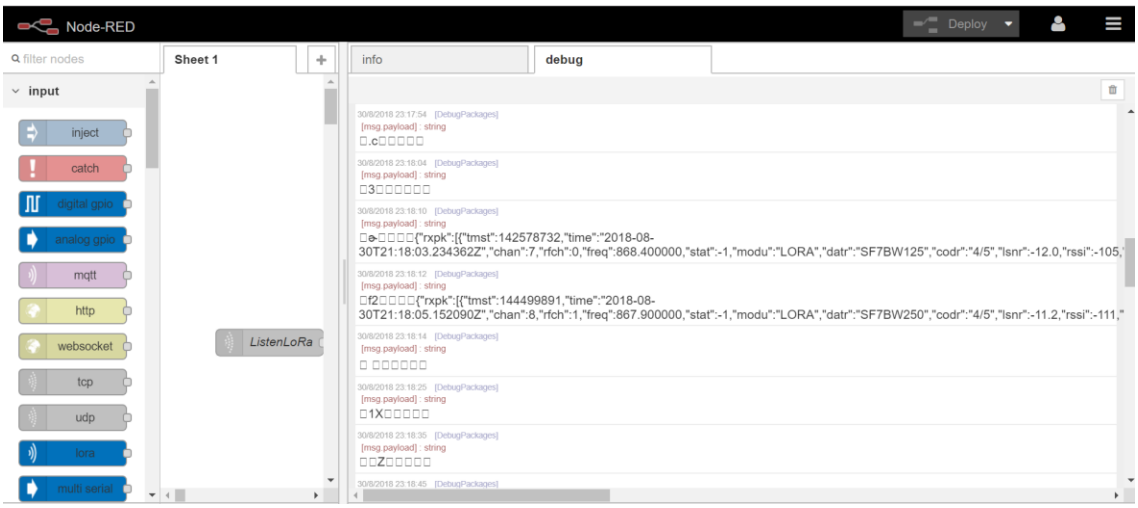

Figura 25. Salida de Node-red, en la imagen de puede apreciar la salida en la parte derecha de la imagen, se trata de un JSON con los parámetros publicados en MQTT.

Estos mensajes serán enviados a un *broker* para ser posteriormente procesados por una aplicación para poder obtener un HeatMap.

5.3. Subscriber

La última parte de este programa es el *Subscriber*, el cuál leerá de un *broker* los datos y los plasmará en una interfaz gráfica para su visibilidad.

Para el desarrollo del *publisher* se ha utilizado Docker, simplificando al usuario la usabilidad de la aplicación, ya que será el propio Docker quien instale en sus imágenes las librerías y programas necesarios, incluso encargándose de las conexiones entre los diferentes contenedores.

Las ventajas de utilizar Docker para el usuario son la compatibilidad de este sobre todas las distribuciones de Linux, evitar la conexión entre imágenes manual e instalación de software automática. El usuario solo deberá instalar Docker, Docker Compose y Git. Git es usado para clonar el repositorio de código donde se encuentra la aplicación.

Una vez descargado el repositorio, se obtendrán varias carpetas: broker-mqtt, interface-nodejs, mongo, subscriber-mqtt. La primera de ellas es broker-mqtt, donde disponemos de un

Dockerfile:

```
FROM debian:jessie

ARG BUILD_DATE
ARG VCS_REF
LABEL org.label-schema.build-date=$BUILD_DATE \
    org.label-schema.docker.dockerfile="/Dockerfile" \
    org.label-schema.license="BSD 3-Clause" \
    org.label-schema.name="docker-mosquitto" \
    org.label-schema.url="https://hub.docker.com/r/toke/mosquitto/" \
    org.label-schema.vcs-ref=$VCS_REF \
    org.label-schema.vcs-type="Git" \
    org.label-schema.vcs-url="https://github.com/toke/docker-mosquitto"

RUN apt-get update && apt-get install -y wget && \
    wget -q -O - https://repo.mosquitto.org/debian/mosquitto-repo.gpg.key |
gpg --import && \
    gpg -a --export 8277CCB49EC5B595F2D2C71361611AE430993623 | apt-key add -
&& \
    wget -q -O /etc/apt/sources.list.d/mosquitto-jessie.list
https://repo.mosquitto.org/debian/mosquitto-jessie.list && \
    apt-get update && apt-get install -y mosquitto mosquitto-clients && \
    adduser --system --disabled-password --disabled-login mosquitto

RUN mkdir -p /mqtt/config /mqtt/data /mqtt/log
COPY config /mqtt/config
RUN chown -R mosquitto:mosquitto /mqtt
VOLUME ["/mqtt/config", "/mqtt/data", "/mqtt/log"]


EXPOSE 1883 9001

ADD docker-entrypoint.sh /usr/bin/
RUN chmod 755 /usr/bin/docker-entrypoint.sh

ENTRYPOINT ["docker-entrypoint.sh"]
CMD ["/usr/sbin/mosquitto", "-c", "/mqtt/config/mosquitto.conf"]
```

En esta imagen se crea un contenedor que crea un *broker* MQTT con usuario y contraseña para conectarse. Tiene persistencia en carpetas de la máquina que lanza Docker. Y abre los puertos 1883 y 9001 para las conexiones con el *broker*.

La siguiente carpeta es subscriber-mqtt, es encargada de suscribirse al *topic* donde están los datos del *Publisher*. Esta desarrollada en NodeJS, se conecta al *topic* y va guardando los datos en tiempo real en una base de datos. El Dockerfile de esta imagen es:

```
FROM node:carbon

ADD ./ /
```

```
RUN npm install

CMD ["node", "/server.js"]
```

Lanzando la siguiente aplicación:

```
var mqtt = require('./mqtt.js')
var client = mqtt.connect('http://localhost');
var MongoClient = require('mongodb').MongoClient;
var url = "mongodb://localhost:27017/lora";

client.on('connect', function () {
  client.subscribe('#');
})

client.on('message', function (topic, message) {
  MongoClient.connect(url, function(err, db) {
  if(!err) {
    console.log("We are connected");
  }
  var data = { topic: topic.toString(), value: message.toString() };
  db.collection('documents').insertOne(data, function(err, res) {
    if (err) throw err;
    console.log("Document inserted");
  });
  db.close();
  console.log(message.toString());
  });
});

console.log('Client started...');
```

Primero se conecta con el *broker* mediante MQTT y descarga los datos dentro de un MongoDB que se desarrollara a continuación.

Para la persistencia de los datos se utiliza una base de datos no relacional como es MongoDB quien guarda los datos en forma de documentos en formato *JSON*, para facilitar la lectura por los siguientes programas.

La imagen de MongoDB se crea con el siguiente Dockerfile:

```
FROM mongo:3.6

ENV MONGODB_USER=lora
ENV MONGODB_DATABASE=lora
ENV MONGODB_PASS=lora

COPY /initialMongo.sh /initialMongo.sh
```

```
RUN chmod 755 /initialMongo.sh

CMD ["/initialMongo.sh"]
```

La versión de MongoDB utilizada es la 3.6 una de las últimas estables, en la que se crea una base de datos con usuario y contraseña, y se ejecuta un script que lanza el demonio de MongoDB para crear los usuarios y las colecciones donde se insertarán los datos, finalmente se lanza MongoDB donde es necesario conectarse mediante usuario y contraseña.

```
#!/bin/bash
set -m

mongod --bind_ip_all --smallfiles &

USER=${MONGODB_USER:-"lora"}
DATABASE=${MONGODB_DATABASE:-"lora"}
PASS=${MONGODB_PASS:-"lora"}
_word=$( [ ${MONGODB_PASS} ] && echo "preset" || echo "random" )
sleep 3
echo "=> Creating an ${USER} user with a ${_word} password in MongoDB"
/usr/bin/mongo $DATABASE --eval  "db.createUser({ user: '$USER', pwd:
'$PASS', roles:['dbOwner']});"
/usr/bin/mongo admin --eval  "db.createUser({ user: '$USER', pwd: '$PASS',
roles:['root']});"
/usr/bin/mongo $DATABASE --eval  "db.createCollection('documents');"
mongo admin -u $USER -p $PASS << EOF
use test;
db.createUser({ user: '$USER' , pwd: '$PASS', roles: ['dbOwner']});
EOF
echo "=> Done!"

echo "=> Creating an ${USER} user with a ${_word} password in MongoDB"

mongod --shutdown

echo
"========================================================================"
echo "You can now connect to this MongoDB server using:"
echo ""
echo "    mongo $DATABASE -u $USER -p $PASS --host <host> --port <port>"
echo ""
echo "Please remember to change the above password as soon as possible!"
echo
"========================================================================"

mongod --bind_ip_all --smallfiles #--auth
```

Las colecciones serán consultadas por otro servicio de NodeJS, este será el encargado de leer de la base de datos de los documentos insertados. Al procesarlos obtendrá la latitud, longitud, altura y la intensidad con la que se recibe la señal. El Dockerfile de la instalación es el siguiente:

```
FROM node:carbon
# Create app directory
WORKDIR /usr/src/app
# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm install --only=production
# Bundle app source
COPY . .
EXPOSE 8082
CMD [ "node", "server.js" ]
```

El comando del Dockerfile es lanzar un NodeJS:

```
var express = require("express"),
    app = express(),
    bodyParser  = require("body-parser"),
    methodOverride = require("method-override"),
    mongoose = require('mongoose');

// Connection to DB
mongoose.connect('mongodb://localhost/lora', function(err, res) {
  if(err) throw err;
  console.log('Connected to Database');
});

// Middlewares

app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
app.use(methodOverride());


// Import Models and controllers
var models     = require('./models/datashow')(app, mongoose);
var DataShowCtrl = require('./controllers/datashow');
var router = express.Router();

//ROUTE
router.get('/', function(req, res) {
   res.send("Hello World!");
});
app.use(router);
```

```
// API routes
var datashows = express.Router();

datashows.route('/datashow').get(DataShowCtrl.findAllData);

app.use('/api', datashows);

// Start server
app.listen(8082, function() {
  console.log("Node server running on http://localhost:8082");
});
```

Este archivo instala NodeJS Express para poder lanzar una *url* para visibilizar los datos, primero se conecta a MongoDB y procesa los datos. Por último, publica en una *url* un *html* y JavaScript en el cliente, que crea un HeatMap. Es un mapa que tiene puntos calientes, es decir, con los datos anteriormente se puede visualizar de forma gráfica los datos leídos de la base de datos. Para generar estos mapas se ha utilizado la API de Google Maps, donde se pueden consultar los mapas y aplicar colores sobre ellos.

El siguiente JavaScript genera los HeatMaps:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Heatmaps</title>
    <style>
      /* Always set the map height explicitly to define the size of the div
       * element that contains the map. */
      #map {
        height: 100%;
      }
      /* Optional: Makes the sample page fill the window. */
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
      }
      #floating-panel {
        position: absolute;
        top: 10px;
        left: 25%;
        z-index: 5;
        background-color: #fff;
        padding: 5px;
        border: 1px solid #999;
        text-align: center;
        font-family: 'Roboto','sans-serif';
        line-height: 30px;
        padding-left: 10px;
```

```
            }
            #floating-panel {
              background-color: #fff;
              border: 1px solid #999;
              left: 25%;
              padding: 5px;
              position: absolute;
              top: 10px;
              z-index: 5;
            }
          </style>
        </head>

        <body>
          <div id="floating-panel">
            <button onclick="toggleHeatmap()">Toggle Heatmap</button>
            <button onclick="changeGradient()">Change gradient</button>
            <button onclick="changeRadius()">Change radius</button>
            <button onclick="changeOpacity()">Change opacity</button>
          </div>
          <div id="map"></div>
          <script>

            // This example requires the Visualization library. Include the
libraries=visualization
            // parameter when you first load the API. For example:
            // <script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=visualizati
on">

            var map, heatmap;

            function initMap() {
              map = new google.maps.Map(document.getElementById('map'), {
                zoom: 13,
                center: {lat: 40.153067, lng: -3.633310},
                mapTypeId: 'satellite'
              });

              heatmap = new google.maps.visualization.HeatmapLayer({
                data: getPoints(),
                map: map
              });
            }

            function toggleHeatmap() {
              heatmap.setMap(heatmap.getMap() ? null : map);
            }

            function changeGradient() {
              var gradient = [
                'rgba(0, 255, 255, 0)',
                'rgba(0, 255, 255, 1)',
                'rgba(0, 191, 255, 1)',
```

```
                'rgba(0, 127, 255, 1)',
                'rgba(0, 63, 255, 1)',
                'rgba(0, 0, 255, 1)',
                'rgba(0, 0, 223, 1)',
                'rgba(0, 0, 191, 1)',
                'rgba(0, 0, 159, 1)',
                'rgba(0, 0, 127, 1)',
                'rgba(63, 0, 91, 1)',
                'rgba(127, 0, 63, 1)',
                'rgba(191, 0, 31, 1)',
                'rgba(255, 0, 0, 1)'
              ]
              heatmap.set('gradient', heatmap.get('gradient') ? null : gradient);
            }

            function changeRadius() {
              heatmap.set('radius', heatmap.get('radius') ? null : 20);
            }

            function changeOpacity() {
              heatmap.set('opacity', heatmap.get('opacity') ? null : 0.2);
            }


            function getPoints() {
              return [
                {location: new google.maps.LatLng(40.153067, -3.633310), weight:
0.5},
                {location: new google.maps.LatLng(40.153065, -3.633310), weight:
2},
                {location: new google.maps.LatLng(40.153065, -3.633312), weight:
2},
                {location: new google.maps.LatLng(40.153132, -3.633225), weight:
2},
                {location: new google.maps.LatLng(40.153130, -3.633222), weight:
6},
                {location: new google.maps.LatLng(40.153130, -3.633223), weight:
6},
                {location: new google.maps.LatLng(40.152990, -3.633220), weight:
1},
                {location: new google.maps.LatLng(40.153080, -3.633247), weight:
4},
                {location: new google.maps.LatLng(40.153083, -3.633245), weight:
4}
              ];
            }
          </script>
          <script async defer

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDyn47O6dopwoz-
j53SDFIH636GERSd3ok&libraries=visualization&callback=initMap">
          </script>
        </body>
      </html>
```

El siguiente comando (`{location: new google.maps.LatLng(40.153067, -3.633310), weight: 0.5}`) crea un objeto de Google Maps con la latitud y longitud, el último parámetro es la intensidad de la señal de menor a mayor.

Si consultamos la URL se puede ver el mapa con los puntos calientes que se han creado. También se puede configurar el punto de inicio del mapa y la calidad de este:

```
map = new google.maps.Map(document.getElementById('map'), {
          zoom: 13,
          center: {lat: 40.153067, lng: -3.633310},
          mapTypeId: 'satellite'
        });
```

Se trata de un objeto Mapa de Google Maps. Center es la posición donde al cargar la web se iniciará el mapa, el zoom y el tipo de mapa que se quiere mostrar.

5.3. Configuración automática Raspberry

Esta sección consiste en la configuración automática de los parámetros LoRaWAN, mediante la interfaz gráfica se añaden los parámetros con los que se enviarán los paquetes de LoRa. Una vez se rellena el formulario y se envían los parámetros, la aplicación usa Ansible para ejecutar un *playbook* que mediante SSH se conecte con la Raspberry. El *playbook* contiene un role que ejecuta el comando en la Raspberry. En especial, usa el comando *Make* que se ha definido anteriormente.

6. Pruebas de concepto

En este apartado se aportarán los resultados obtenidos por la aplicación. Se han realizado varias pruebas con diferentes parámetros LoRa y condiciones que pueden alterar la aplicación. Para probar la aplicación se ha instalado el *gateway* en la facultad de Físicas de la Universidad Complutense de Madrid, desde allí se realizó un recorrido por los alrededores de la Facultad.

Se ha decidido realizar dos pruebas, las dos con una frecuencia de 868MHz siendo la frecuencia europea estándar. El ancho de banda también será constante en ambas de 125 KHz, si el ancho de banda se aumentase como reacción se aumentaría la tasa de bits de envío y el ruido que se genera, para solventarlo la potencia necesaria debería aumentar. Los dos parámetros que variarán serán Code Rating y Spread Factor:

1. En la primera prueba se utilizará un Spread Factor de 7 y un Code Rating de 4/5. Estos parámetros están definidos para enviar a distancias cortas, con un bajo gasto de batería y un alto envío de datos. Es decir, transferencias rápidas, eficientes en distancias de poco alcance y con un índice alto de errores.
2. En la segunda prueba se utilizará un Spread Factor de 12 y un Code Rating de 4/8. Con estos nuevos parámetros están definidos para lo contrario que los anteriores, un gasto de batería superior, conseguir una distancia de envío mucho superior, reducir el índice de errores y transferencias de datos lentas.

6.1. Primera prueba

Se han obtenido los siguientes puntos geográficos, que concuerdan con las dos primeras columnas, y como tercera columna la intensidad con la que ha llegado la señal en un intervalo de 0 a 60, donde las señales superiores a 30 son de muy buena intensidad, las señales entre 30 y 5 son de buena calidad y las inferiores a 5 no tienen suficiente intensidad para una conexión estable:

Latitud	Longitud	Intensidad	Latitud	Longitud	Intensidad
40,4504578	-3,727476597	10	40,45047995	-3,72559794	39
40,44962269	-3,725721322	13	40,45057792	-3,725614034	40,5
40,45010439	-3,72551211	14	40,45020236	-3,726525985	42
40,45013939	-3,727342486	14	40,45053944	-3,72712791	42
40,44992478	-3,725989543	14,5	40,45068406	-3,725640856	42
40,44954921	-3,726375781	16	40,45027584	-3,726177298	43
40,44956554	-3,726273857	18	40,45028401	-3,726134382	45
40,45049862	-3,727363944	18	40,4505721	-3,726827502	46
40,44959004	-3,725823246	20	40,45037381	-3,726102196	47
40,44992478	-3,726885401	20,5	40,45053944	-3,727042079	48
40,4497125	-3,726139747	22	40,45053944	-3,726907969	48,5
40,44997376	-3,726944409	22	40,4505721	-3,726634	49
40,44998427	-3,727245927	22	40,45060241	-3,726402603	50
40,44962269	-3,72592517	22,5	40,4506864	-3,726097941	50
40,44959004	-3,726595722	23	40,45069456	-3,726553917	50
40,4499656	-3,727142893	23	40,45076804	-3,726017475	50
40,44953289	-3,72652062	24	40,45017787	-3,726584993	50,5
40,44978598	-3,726198755	24	40,45046362	-3,726064645	50,5
40,44998427	-3,727245927	24	40,45052894	-3,72605928	50,5
40,44980231	-3,72628995	25	40,45055343	-3,726552807	50,5
40,44958187	-3,726150475	25,5	40,45057792	-3,726086102	50,5
40,4495982	-3,726080738	26	40,45060241	-3,726461612	50,5
40,44964719	-3,726091467	27	40,45061874	-3,72620412	50,5
40,44999009	-3,726992689	29	40,45063507	-3,726247035	50,5
40,44989212	-3,726440154	29,5	40,4506619	-3,725604415	50,5
40,45020236	-3,725496016	29,5	40,45069456	-3,725738525	50,5
40,44960637	-3,726686917	30	40,45071089	-3,726376891	50,5
40,44981048	-3,725968085	30	40,45071905	-3,725894094	50,5
40,44991661	-3,726509891	30	40,45072722	-3,726119399	50,5
40,45002509	-3,727090359	30	40,45074355	-3,726242781	50,5
40,45014521	-3,725860797	30	40,45076804	-3,726135492	50,5
40,4504228	-3,725517474	30	40,4507762	-3,726038933	50,5
40,45050678	-3,727240562	30	40,45079253	-3,725894094	50,5
40,45033299	-3,725555025	31	40,45082519	-3,726130128	50,5
40,45063507	-3,725807153	31	40,44966352	-3,725989543	51
40,45005775	-3,727396131	32	40,45021053	-3,726552807	51
40,45061058	-3,725673042	32	40,45034116	-3,726000272	51
40,44967168	-3,726767384	32,5	40,45059425	-3,72652062	51
40,44985946	-3,725909077	33	40,45061058	-3,726043187	51
40,44997376	-3,7265689	33,5	40,45063507	-3,726327501	51
40,45004724	-3,726627909	34	40,45063507	-3,725957356	51
40,44981048	-3,726794206	36	40,45065956	-3,726123653	51
40,44997376	-3,72721263	36	40,45070273	-3,726199865	51

40,45014521	-3,726230942	36	40,45071089	-3,725996017	51
40,45012072	-3,726558171	36,5	40,45072722	-3,726451993	51
40,44970434	-3,726804934	38	40,45075988	-3,726140857	51
40,45008806	-3,72628995	38	40,45075988	-3,725577593	51
40,45015338	-3,726445518	38	40,4505721	-3,726714849	51,5
40,45023502	-3,726214848	39	40,45078437	-3,725802898	51,5

En los datos de la tabla anterior se puede observar que la intensidad por encima de 30 está presente en más del 75% de los datos, es decir, la conexión es de muy buena calidad en la mayoría de los puntos cercanos al *gateway*, por lo que confirma que la configuración de la primera prueba consiste en conexiones de menor distancia, pero eficientes. Los datos de la tabla con menor intensidad son los puntos más alejados del *gateway* situado en la facultad de Físicas.

Para la recolección de 100 datos, es decir, 100 paquetes enviados por nuestro sistema y recibidos por el *gateway* correctamente han sido necesarios 20 minutos. En cuanto al consumo de batería, este sistema tiene una durabilidad de aproximadamente 5 horas y media emitiendo señales LoRa con 3000 mA, siendo capaz de recolectar 1.650 puntos de geolocalización.

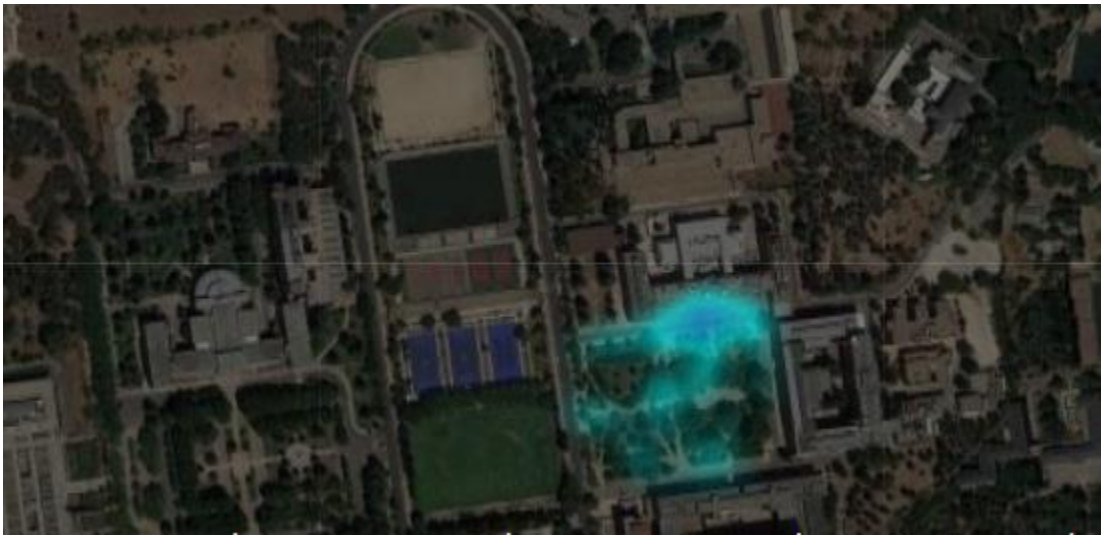

Figura 26. Resultados reales, en la imagen se pueden ver los resultados reales de las pruebas en la Universidad Complutense de Madrid con un ancho de banda de 125 KHz, una frecuencia de 868MHz, Spread factor de 7 y un Code Rating de 4/5.

6.2. Segunda prueba

Se han obtenido los siguientes puntos geográficos, que concuerdan con las dos primeras columnas, y como tercera columna la intensidad con la que ha llegado la señal en un intervalo de 0 a 60, donde las señales superiores a 30 son de muy buena intensidad, las señales entre 30 y 5 son de buena calidad y las inferiores a 5 no tienen suficiente intensidad para una conexión estable:

Latitud	Longitud	Intensidad	Latitud	Longitud	Intensidad
40,4495327	-3,72666959	31,5	40,45065938	-3,725596707	38
40,44921429	-3,727452795	33	40,45032465	-3,726020496	38,5
40,45009605	-3,72602586	33,5	40,45047977	-3,72589175	38,5
40,44956536	-3,727350872	34	40,45057774	-3,725698631	38,5
40,45010421	-3,725773733	34	40,45067571	-3,725559156	38,5
40,4501287	-3,725875657	34	40,45074919	-3,725521605	38,5
40,44922246	-3,727383058	34,5	40,45043078	-3,726004403	39
40,44967966	-3,725462596	34,5	40,45054508	-3,725618164	39
40,4501287	-3,725703995	34,5	40,45055325	-3,725526969	39
40,4495572	-3,725822012	35	40,45071653	-3,725489418	39
40,44979396	-3,72547869	35	40,45074919	-3,7256128	39
40,45010421	-3,725532334	35	40,45076552	-3,725779097	39
40,45013687	-3,725644987	35	40,44952454	-3,72699682	39,5
40,4501287	-3,726117055	35	40,4503573	-3,72597758	39,5
40,44926328	-3,727393787	35,5	40,4504961	-3,725779097	39,5
40,44952454	-3,726782243	35,5	40,45057774	-3,725575249	39,5
40,44954903	-3,726567666	35,5	40,45074103	-3,72543041	39,5
40,44962251	-3,725628893	35,5	40,44991643	-3,727281134	40
40,44985928	-3,725500147	35,5	40,45075735	-3,725827377	40
40,45002257	-3,725510876	35,5	40,45000624	-3,72729434	40,5
40,45010421	-3,725966852	35,5	40,44985112	-3,72727577	40,5
40,44934492	-3,727388422	36	40,45064305	-3,725532334	40,5
40,44919796	-3,727334778	36	40,4507002	-3,725897114	40,5
40,44950005	-3,726374547	36	40,45074103	-3,726047318	40,5
40,4495572	-3,726047318	36	40,4508145	-3,726106327	40,5
40,44960618	-3,725505512	36	40,44995725	-3,727203145	41
40,44994092	-3,725500147	36	40,4500389	-3,727079763	41,5
40,45017769	-3,72556452	36	40,45073286	-3,725988309	41,5
40,45033281	-3,726111691	36	40,44998991	-3,726977839	42
40,44943473	-3,727393787	36,5	40,44998991	-3,727074399	42
40,44927144	-3,727265041	36,5	40,45000624	-3,727144136	42,5
40,44957353	-3,725934665	36,5	40,45032465	-3,726596966	43
40,45020218	-3,726138513	36,5	40,45003073	-3,726827636	43
40,44956536	-3,726240437	37	40,4500389	-3,726875915	43

40,44958985	-3,725730817	37	40,45022668	-3,726677432	43,5
40,44963068	-3,725403588	37	40,45006339	-3,726790085	43,5
40,45022668	-3,726015131	37	40,4505859	-3,726280465	44
40,45038996	-3,726068776	37	40,45010421	-3,726731076	44
40,44972049	-3,727318685	37,5	40,45062673	-3,726226821	44,5
40,44963884	-3,727302592	37,5	40,45069204	-3,726114168	45
40,44949188	-3,727377694	37,5	40,45046344	-3,726425304	45
40,44941024	-3,727265041	37,5	40,45040629	-3,726495042	45
40,44946739	-3,727152388	37,5	40,45016136	-3,726672068	45
40,44952454	-3,726884167	37,5	40,45056141	-3,726339474	45,5
40,44951638	-3,726476471	37,5	40,45036547	-3,726527228	45,5
40,4495572	-3,726127784	37,5	40,45029199	-3,726645245	45,5
40,44973681	-3,725408952	37,5	40,45025117	-3,726596966	45,5
40,45031648	-3,72612242	37,5	40,4507247	-3,726076617	46
40,44977764	-3,727345507	38	40,45067571	-3,726178541	46

En los datos de la tabla anterior se puede observar que la intensidad por encima de 30 está presente en el 100% de los datos, es decir, la conexión es de muy buena calidad en todos los puntos, confirmando que un Spread Factor de 12 junto con un Code Rating de 4/8 aumentan la distancia de conexiones estables y evitan un alto número de mensajes erróneos.

Para la recolección de 100 datos, es decir, 100 paquetes enviados por nuestro sistema y recibidos por el *gateway* correctamente han sido necesarios 25 minutos, llegando a la conclusión de que a mayor Spread Factor menor es la velocidad de envíos de bits. En cuanto al consumo de batería, este sistema tiene una durabilidad de aproximadamente 5 horas emitiendo señales LoRa con 3000 mA, siendo capaz de recolectar 1.200 puntos de geolocalización.

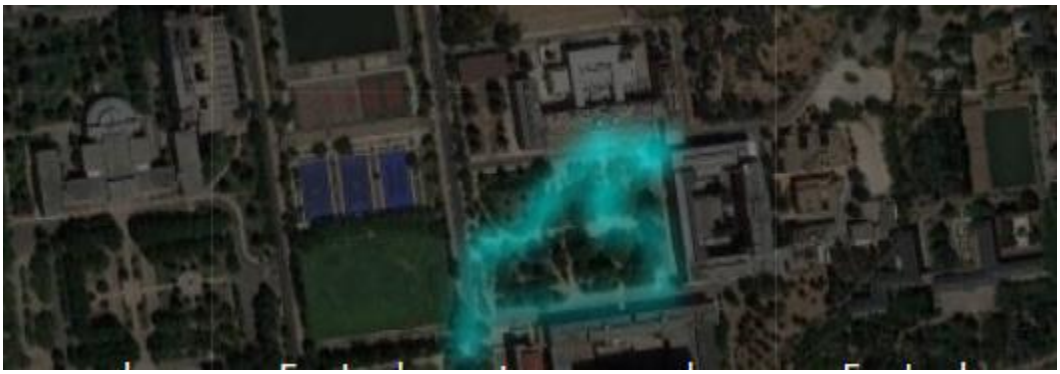

Figura 27. Resultados reales, en la imagen se pueden ver los resultados reales de las pruebas en la Universidad Complutense de Madrid con un ancho de banda de 125 KHz, una frecuencia de 868MHz, Spread factor de 12 y un Code Rating de 4/8.

7. Conclusiones y trabajo futuro

En conclusión, la herramienta para facilitar el estudio detallado y automático de la cobertura de red LoRa en entornos reales, aporta una funcionalidad positiva a la hora de diseñar arquitecturas IoT. Analizando los terrenos con la aplicación, se puede analizar en detalle la posición de los *gateways* que recolectaran las señales LoRa del resto de sensores, sin necesidad de desplegar por completo la red de sensores para comprobar su correcto funcionamiento o subsanar posibles errores de conexiones en arquitecturas ya puestas en producción.

Por otro lado, esta herramienta puede ser usada para el testeo de nuevas incorporaciones tecnológicas como puede ser un nuevo *Gateway*; al generar informes gráficos, la posibilidad de hacer comparaciones entre las distintas posibilidades es un factor a favor de la herramienta. El auge de la industria 4.0 y la inmadurez de las tecnologías novedosas hace que todavía quede mucho recorrido para esta industria, dando un punto a favor para el desarrollo de nuevas herramientas que ayuden a la madurez del IoT.

Respecto a las tecnologías utilizadas en este proyecto, LoRa es uno de los pilares de la aplicación donde debemos definir con anterioridad los parámetros que se van a utilizar, a mayor Spread Factor, ancho de banda y Code Rating se consigue mejor tasa de envío, menor distancia y un consumo de batería bajo. Por el contrario, su reducción consigue mayor distancia, un consumo mayor y una tasa de envío menor.

En cuanto al tiempo de despliegue de la aplicación es relativamente rápido, se divide en tres partes:

1. La instalación en la Raspberry del software necesario.
2. La configuración del *Gateway*.
3. La instalación con Docker Compose del *subscriber*.

Una vez desplegada la aplicación la recolecta de datos depende de la configuración de los parámetros, en torno a 500 datos por hora.

Como trabajo futuro, existen dos puntos que aportarían un valor positivo a la herramienta:

- La inicialización del programa mediante un botón. En la actualidad la aplicación está diseñada para iniciarse cuando la Raspberry Pi está conectada a la red, y una mejora sería la posibilidad de iniciar el envío de datos cuando el usuario quisiera. La configuración de los parámetros de cobertura LoRa sí debería seguir realizándose con conexión a la red.

- Otra aportación para poder geolocalizar la señal de LoRa sin necesidad de GPS es instalar dos balizas que envíen paquetes LoRa; si se conocen sus posiciones sería posible calcular la posición del sensor, esta aportación sería positiva para lugares donde GPS puede no dar señal, como una cueva o el interior de un edificio.

Anexo: Manual de instalación

Para facilitar la usabilidad de la aplicación se facilita un manual de instalación de la aplicación, transmitiendo soluciones a problemas que pueden surgir a la hora de instalar en diversos sistemas operativos.

A.1. Publisher

En la sección del *Publisher*, se debe conectar la Raspberry PI a la antena LoRa mediante el *brigde* de Cooking Hack, y el GPS mediante USB.

Tras conectar los dos accesorios, se deberá acceder a la Raspberry para continuar su instalación. La primera parte es la conexión con el GPS, solo es necesario si es la primera vez que conectamos el sistema:

```
#Editar el inicio de la raspberry
sudo nano /boot/cmdline.txt
      dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
      rootfstype=ext4 elevator=deadline rootwait
#Detenemos los servicios GPS por defecto de Rasberry PI
sudo systemctl stop serial-getty@ttyUSB0.service
sudo systemctl disable serial-getty@ttyUSB0.service
sudo shutdown -r now
#Se instala GPS para poder visualizar los resultados
sudo apt-get install gpsd gpsd-clients python-gps
#Paramos los socket de GPS ahora activos
sudo systemctl stop gpsd.socket
sudo systemctl disable gpsd.socket
#Se crea el nuevo socket GPS
sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
#Se detiene para poder volverlo a lanzar
sudo killall gpsd
sudo gpsd /dev/ttyUSB0 -F /var/run/gpsd.sock
#Se definen nuevos parámetros para GPS (probar por defecto anteriormente)
sudo nano /etc/default/gpsd
      START_DAEMON="true"
      GPSD_OPTIONS="/dev/ttyUSB0"
      DEVICES=""
      USBAUTO="true"
      GPSD_SOCKET="/var/run/gpsd.sock"
#Lanzamos la librería para obtener los resultados
cgps -s
```

Una vez el GPS está corriendo, descargar de GitHub el código:

```
//Se clona el repositorio
git clone https://github.com/JoniSanchez/heatmap_lora_signal.git
//Entrada en la carpeta de LoRaWAN
cd heat_lora_signal/cooking_hack/examples/LoRaWAN
// Guía de make para ver que se hay definido
make
//Lanzar la aplicación
make send-packages-lora-gps
```

Dentro del *Makefile* se puede cambiar los parámetros que inician la aplicación.

```
##Variables
SPREADING_FACTOR?=sf7
FRECUENCY?=868000000
BANDWIDTH?=125
CODING_RATE?=4/5
```

A.2. Gateway

Configuración del *gateway*, para ello se debe conectar el *gateway* mediante ethernet a nuestra red, esto nos dará una IP para conectarnos mediante http. Una vez dentro de la interfaz gráfica acceder a LoRa.

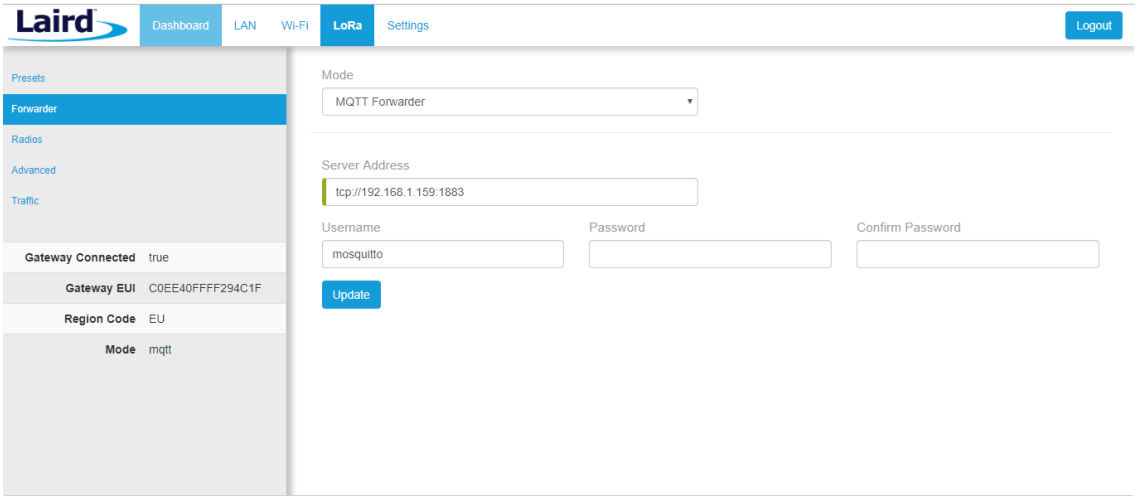

Figura 28. Configuración del Gateway.

En la figura [29] se configura el *forwarder* que va a utilizar el *gateway*. En el modo de forwarder se debe añadir MQTT *forwarder*, "IP:puerto" que utilice el bróker MQTT, por último el usuario y la contraseña necesaria para la conexión con el *broker*.

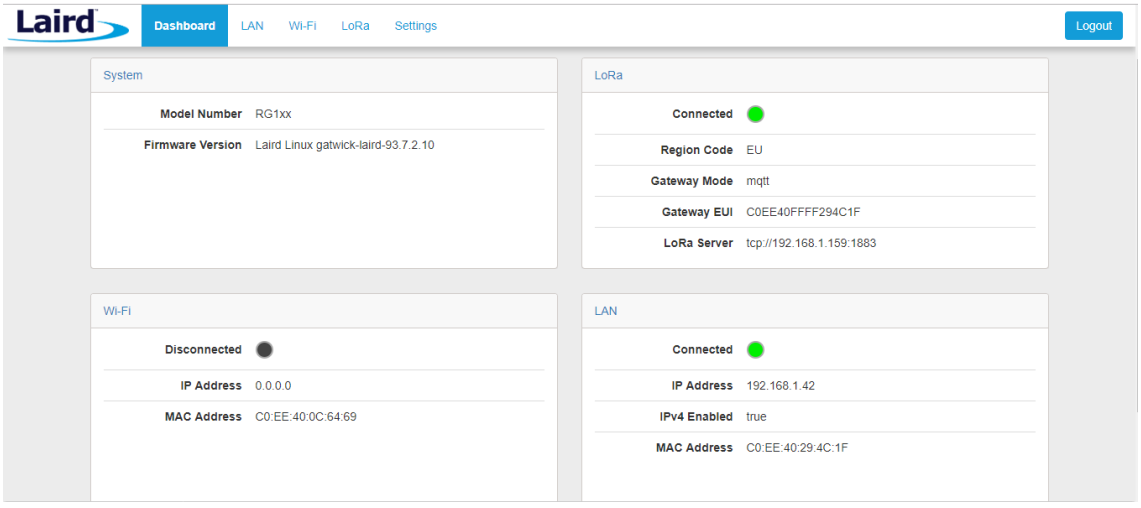

Figura 29. Dashboard página de inicio.

En la figura [30] se puede observar las conexiones necesarias creadas anteriormente, la conexión a LAN mediante el cable de ethernet y la conexión con el *broker* MQTT.

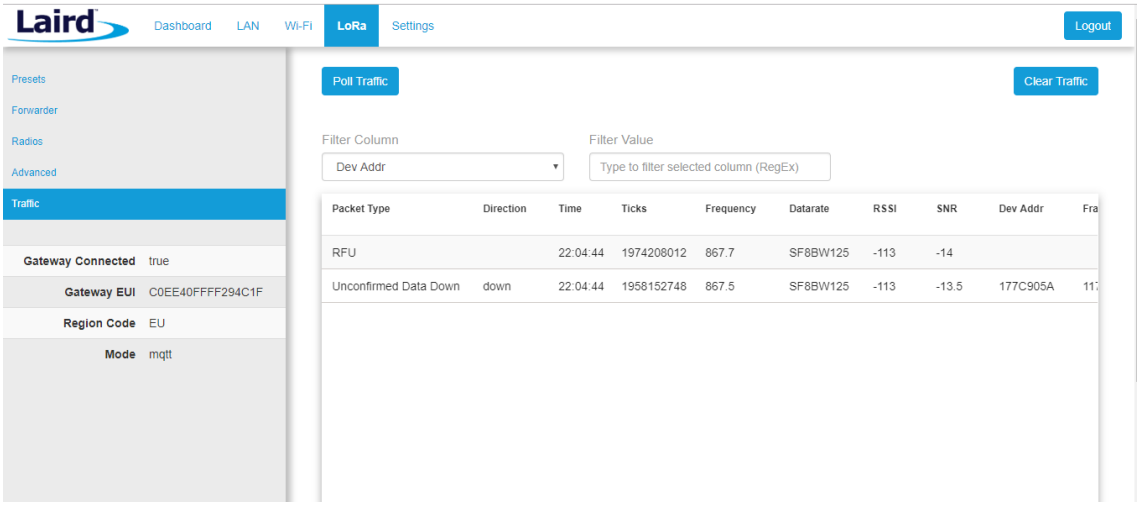

Figura 30. Debug del reenvío de paquetes.

En la figura [31] se puede observar el tráfico que pasa por el *gateway*, los paquetes LoRa recibidos y reenviados al *broker* MQTT con el *topic* "Gateway/MAC".

A.3. Subscriber

La parte del *subscriber*, debemos tener un equipo con sistema operativo base Linux, con la instalación de Docker y Docker Compose, este software se encargará de la conexión entre aplicaciones y de levantar los contenedores de cada módulo de la aplicación.

```
//Se clona el repositorio
git clone https://github.com/JoniSanchez/heatmap_lora_signal.git
//Entrada en la carpeta principal
cd heat_lora_signal
// Guía de make para ver que se hay definido
Make
//Construye y despliega la aplicación
make build-deploy-application
```

El comando consiste en utilizar Docker Compose para lanzar las imágenes Docker de las carpetas, Docker Compose es un orquestador que se ocupará de las conexiones entre las diversas

imágenes, facilitando la comunicación entre las diversas funcionalidades de la aplicación, por otro lado, publicará los puertos expuestos para las conexiones exteriores al contenedor, por ejemplo, consultar la web vía HTTP. Una vez lanzada se puede comprobar si todo ha ido correctamente consultando los contenedores que están corriendo, mongo, NodeJS, *broker* y *subscriber*:

```
[devops@devops-box ~]$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS
                                  NAMES
7c2db5b63e66        heatmaplorasignal_nodejs       "node /server.js"        2 hours ago         Up 2 hours          0.0.0.0:8082->8082/tc
p                                 heatmaplorasignal_nodejs_1
6640b270c7f6        heatmaplorasignal_subscriber   "node /server.js"        2 hours ago         Up 2 hours
                                  heatmaplorasignal_subscriber_1
57edc409d981        heatmaplorasignal_broker       "/run.sh mosquitto"      17 hours ago        Up 2 hours          0.0.0.0:1883->1883/tc
p, 0.0.0.0:9001->9001/tcp, 9883/tcp   heatmaplorasignal_broker_1
1246a32900f6        heatmaplorasignal_mongo        "docker-entrypoint.s…"   46 hours ago        Up 2 hours          0.0.0.0:27017->27017/
tcp                               heatmaplorasignal_mongo_1
```

Figura 31. Contenedores Docker.

Por otro lado, para comprobar se pueden observar los logs del Docker Compose, por lo general, la mayoría de los logs deberían ser *broker*, *subcriber* y MongoDB:

```
mongo_1       | 2018-08-29T10:49:06.990+0000 I NETWORK  [conn303] received client metadata from 172.19.0.4:60280 conn303: { driver: { nam
e: "nodejs", version: "2.2.36" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "3.10.0-514.26.2.el7.x86_64" }, platf
orm: "Node.js v8.11.4, LE, mongodb-core: 2.1.20" }
mongo_1       | 2018-08-29T10:49:06.991+0000 I NETWORK  [conn303] end connection 172.19.0.4:60280 (8 connections now open)
broker_1      | 1535539752: Received PUBLISH from NodeRedMultiTech (d0, q0, r0, m0, '/lora', ... (472 bytes))
broker_1      | 1535539752: Sending PUBLISH to mqttjs_65f15a81 (d0, q0, r0, m0, '/lora', ... (472 bytes))
mongo_1       | 2018-08-29T10:49:12.103+0000 I NETWORK  [listener] connection accepted from 172.19.0.4:60282 #304 (9 connections now open
)
mongo_1       | 2018-08-29T10:49:12.103+0000 I NETWORK  [conn304] received client metadata from 172.19.0.4:60282 conn304: { driver: { nam
e: "nodejs", version: "2.2.36" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "3.10.0-514.26.2.el7.x86_64" }, platf
orm: "Node.js v8.11.4, LE, mongodb-core: 2.1.20" }
mongo_1       | 2018-08-29T10:49:12.105+0000 I NETWORK  [conn304] end connection 172.19.0.4:60282 (8 connections now open)
broker_1      | 1535539752: Received PUBLISH from NodeRedMultiTech (d0, q0, r0, m0, '/lora', ... (18 bytes))
broker_1      | 1535539752: Sending PUBLISH to mqttjs_65f15a81 (d0, q0, r0, m0, '/lora', ... (18 bytes))
mongo_1       | 2018-08-29T10:49:12.519+0000 I NETWORK  [listener] connection accepted from 172.19.0.4:60284 #305 (9 connections now open
)
mongo_1       | 2018-08-29T10:49:12.520+0000 I NETWORK  [conn305] received client metadata from 172.19.0.4:60284 conn305: { driver: { nam
e: "nodejs", version: "2.2.36" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "3.10.0-514.26.2.el7.x86_64" }, platf
orm: "Node.js v8.11.4, LE, mongodb-core: 2.1.20" }
broker_1      | 1535539757: Received PUBLISH from NodeRedMultiTech (d0, q0, r0, m0, '/lora', ... (696 bytes))
broker_1      | 1535539757: Sending PUBLISH to mqttjs_65f15a81 (d0, q0, r0, m0, '/lora', ... (696 bytes))
mongo_1       | 2018-08-29T10:49:17.205+0000 I NETWORK  [listener] connection accepted from 172.19.0.4:60286 #306 (10 connections now ope
n)
mongo_1       | 2018-08-29T10:49:17.205+0000 I NETWORK  [conn306] received client metadata from 172.19.0.4:60286 conn306: { driver: { nam
e: "nodejs", version: "2.2.36" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "3.10.0-514.26.2.el7.x86_64" }, platf
orm: "Node.js v8.11.4, LE, mongodb-core: 2.1.20" }
mongo_1       | 2018-08-29T10:49:17.207+0000 I NETWORK  [conn306] end connection 172.19.0.4:60286 (9 connections now open)
broker_1      | 1535539757: Received PINGREQ from NodeRedMultiTech
broker_1      | 1535539757: Sending PINGRESP to NodeRedMultiTech
broker_1      | 1535539761: Saving in-memory database to /var/lib/mosquitto/mosquitto.db.
broker_1      | 1535539762: Received PUBLISH from NodeRedMultiTech (d0, q0, r0, m0, '/lora', ... (252 bytes))
broker_1      | 1535539762: Sending PUBLISH to mqttjs_65f15a81 (d0, q0, r0, m0, '/lora', ... (252 bytes))
mongo_1       | 2018-08-29T10:49:22.320+0000 I NETWORK  [listener] connection accepted from 172.19.0.4:60288 #307 (10 connections now ope
n)
mongo_1       | 2018-08-29T10:49:22.320+0000 I NETWORK  [conn307] received client metadata from 172.19.0.4:60288 conn307: { driver: { nam
e: "nodejs", version: "2.2.36" }, os: { type: "Linux", name: "linux", architecture: "x64", version: "3.10.0-514.26.2.el7.x86_64" }, platf
orm: "Node.js v8.11.4, LE, mongodb-core: 2.1.20" }
mongo_1       | 2018-08-29T10:49:22.334+0000 I NETWORK  [conn307] end connection 172.19.0.4:60288 (9 connections now open)
```

Figura 32. Logs de la aplicación.

A.4. Configuración mediante web

La configuración web de los parámetros LoRaWAN para emitir mensajes se puede realizar desde la interfaz web.

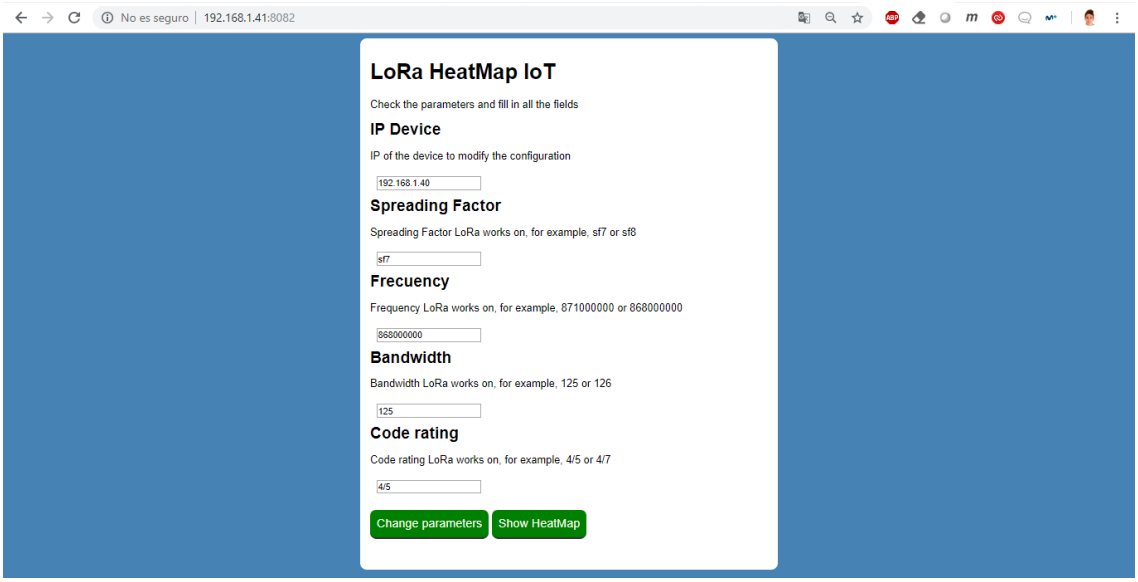

Figura 33. Interfaz gráfica para la configuración de los parámetros LoRaWAN.

En la figura [34] se puede configurar los parámetros añadiéndolos a los cuadros de texto, en el primero de ellos se añade la IP de la Raspberry para conectarse mediante SSH y ejecutar el *playbook*. En el resto de los cuadros se deben añadir los parámetros con los que se quiere ejecutar la aplicación. Finalmente, pulsando el botón "*Change parameters*" se ejecuta la aplicación con los parámetros introducidos.

8. Bibliografía

- [1] *LoRaWAN Specification v1.1*. LoRa Alliance (2017).
- [2] *MQTT v.3.1.1 OASIS Standard*. OASIS (2014).
- [3] L. M. Surhone, M. T. Tennoe, S. F. Henssonow (2010). *NodeJS*. Betascript Publishing, Mauritius.
- [4] K. Chodorow, M. Dirolf. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly (2017).
- [5] *Sigfox Technical Overview*. Sigfox (2017). https://www.disk91.com/wp-content/uploads/2017/05/4967675830228422064.pdf (Última consulta: 4 de febrero de 2019).
- [6] WiFi Heatmap. Google Play. https://play.google.com/store/apps/details?id=ua.com.wifisolutions.wifiheatmap&hl=es (Última consulta: 4 de febrero de 2019)
- [7] A. El-Rabbany. *Introduction to GPS: the global positioning system*. Artech house (2002).
- [8] TamoGraph Site Survey. Documentación de ayuda. https://www.tamos.com/docs/tg42es.pdf (Última consulta: 4 de febrero de 2019).
- [9] NodeJS Express API: http://expressjs.com/es/api.html (Última consulta 4 de febrero de 2019).
- [10] Cooking Hack shield. https://www.cooking-hacks.com/documentation/tutorials/extreme-range-lora-sx1272-module-shield-arduino-raspberry-pi-intel-galileo (Última consulta: 4 de febrero de 2019).

9. Índice de figuras

Abstract

The present work proposes the design and development of a tool that facilitates the detailed and automatic study of LoRa networks coverage in real environments. For this purpose, an application has been developed and implemented in two phases: the first consists of sending LoRa packets containing the geographic coordinates of the sensor position; the packets are sent to a LoRa gateway that will forward the packet data and the signal intensity received from the packet to an MQTT broker. The second phase consists in the broker's subscription to a NodeJS program, checking if the received data are correct to send them to a non-relational database (MongoDB) for its later reading and printing in a HeatMap, map with satellite vision where the sensor signal will be visualized. The goal of this project is to graphically detail the signal strength in a LoRa network to aid in architectural design, avoiding black spots when installing LoRa solutions.

Keywords

LoRa
Docker
Geolocation
Gateway
HeatMaps
MongoDB
MQTT
Ansible
Google Heatmaps
GPS

Introduction

Wireless communications between devices since their inception have evolved dramatically, with short-range communications (e.g., Wi-Fi) pioneering. There is a need for new technologies to achieve greater range and lower energy consumption, with the emergence of the IoT paradigm that will enable greater coverage ranges of kilometres or tens of kilometres, for example, in cities, business campuses or universities.

As a solution to the needs of IoT, LPWAN technologies have appeared with low power and long range, which is due to the increase in bandwidth. These features have been used in industry 4.0, where hundreds or thousands of sensors are distributed over a wide area. The sensors cannot consume much energy as it would be unviable to change the batteries of hundreds of sensors every short time, or even use GSM would also be unviable for the cost of a phone card in each of the sensors.

LPWAN is not a single technology, but a group of technologies that include patented and open standards and use licensed and unlicensed frequencies. Two of the most widespread examples are Sigfox and LoRa which are detailed below

SigFox is a French company that provides the LPWAN network service. It is one of the most widely deployed networks in the world, operating on a public frequency of 868 MHz in Europe. It uses UNB radio modulation technology, which means that only one operator is allowed on this band in each country. Sigfox imposes several restrictions on transferred messages: each device can send 140 messages per day with a limit of 7 messages per hour with a length of 12 bytes. To integrate Sigfox, you need a compatible radio module and a subscription plan for the device.

LoraWan is a patented standard that operates on frequency from 863 to 870 MHz in long range wireless networks for machine-to-machine M2M applications. LoRaWAN is an open protocol giving the possibility of use to any manufacturer. Unlike Sigfox, a subscription is not necessary and there is no limit to the number of messages.

LoRa and Sigfox

They are an implementation of LPWAN (low-power wide area network), or in other words, a long-range, low-power network. The LoRa and Sigfox networks are interesting because of their low power consumption, which allows devices to last for years on a single battery, because the reduced bandwidth reduces the rate of data being sent from 0.5 Kb/s to 50 Kb/s per channel. In contrast, Wifi has a bit rate of 54,000 Kb/s.

Motivation

In industry 4.0 there is many sensors and actuators distributed by hundreds of kilometres, for a good design of the architecture is necessary to know the coverage of our system all over the field. To solve this problem, we propose the introduction of an automatic signal monitoring tool. With the help of the tool it will be possible to visualize a real LoRa coverage map, obtaining a double advantage:

1. Elimination of blind spots and low intensity signals, by being able to see the coverage map graphically you can easily see the points where the signal does not arrive or that the intensity is low for a connection.
2. Reduction of costs, in the coverage map it is possible to detect the range of the LoRa signal, being able to reduce the number of gateways and sensors, even improving architectures already designed.

There are currently no applications on the market that help us graphically and automatically to design LoRa architectures, so adding a signal monitoring tool in real environments can be positive, helping to improve LoRa architecture designs.

Architecture

This is the combination of software and hardware, which aims to interconnect the components of a computer network.

Goals

The main objective of the project is to implement a tool that provides the user with a detailed and automatic study of the real LoRa coverage in real environments.

The developed tool oversees collecting the coverage signal that exists in a determined zone, and to graphically capture it in a HeatMap. This system allows us to locate blind spots and areas of limited coverage for specific LoRa parameters, being able to reorganize our gateways to obtain a new more efficient architecture providing better coverage in all points.

In some situations, it is necessary to have a specific parameter, for example, a certain bandwidth or range. This system facilitates the tests to eliminate blind spots for certain communication parameters such as terrain. In short, this project makes it possible to develop a system that allows LoRa coverage to be displayed graphically outdoors with specific parameters.

State of the Art

In view of the characteristics of LoRaWAN and Sigfox, they have been positioning themselves in the market in a very exponential way, the need to have devices capable of sending data with a minimum battery cost has been increasing. Growth continues to be booming, which concludes that it is an immature technology with many points to be determined. One of them is the need for a tool capable of displaying LoRa coverage data on a real environment, graphically and automatically. On the other hand, if you compare it with other technologies that have been on the market longer, such as WiFi you can find numerous applications such as:

- **WiFi Heatmap**. It is an Android application that can be installed on the mobile with access to the Wifi network, scans the intensity with which the packets arrive and saves them in a database. Then you can create a hot map with the results. The purpose of this application is like that of this project, but with several negative points, is not a portable application for various softwares, it is only available on Android. The persistence of the data is not accessible by other applications, however in the application of this project you can access the data collected in both MongoDB and MQTT subscribing to the topic. Finally, the most

negative point, does not pick up the signal with GPS parameters if not adding the current position manually.

- **TamoGraph**. It is an application for Mac and Windows payment that generates heat maps with the intensity of the WiFi network. Adding the possibility to scan outdoor environments using GPS. In comparison with the application of this project, it is a closed software, i.e., the data generated by the application can only be consulted by itself, instead this project, the user can consult the data in the broker MQTT or in the MongoDB database.

Work plan

The first phase is based on a contact and research of the technologies, which can be used for the system. The work plan and architecture of the application will be developed.

The first phase is based on a contact and research of the technologies that can be used for the system. The work plan and the architecture of the application will be developed.

The development of the tool is divided into four phases of two weeks each, leaving a week of margin for unforeseen events. The first phase will consist of sending LoRa packages from a reduced plate computer, including as parameters the coordinates of the location at a given moment of the sensor. The second phase will consist of receiving the message, processing the data and storing it in a broker. In the third phase the data stored in the broker will be consulted, processed and if they meet the requirements they will be stored persistently in a non-relational database. In the last development phase, the persistently stored data will be consulted and encapsulated in a HeatMap.

Finally, in the last two months the concept tests will be carried out, where the detected faults will be corrected, and the design improvements of the tool will be made, collecting the information to write the report.

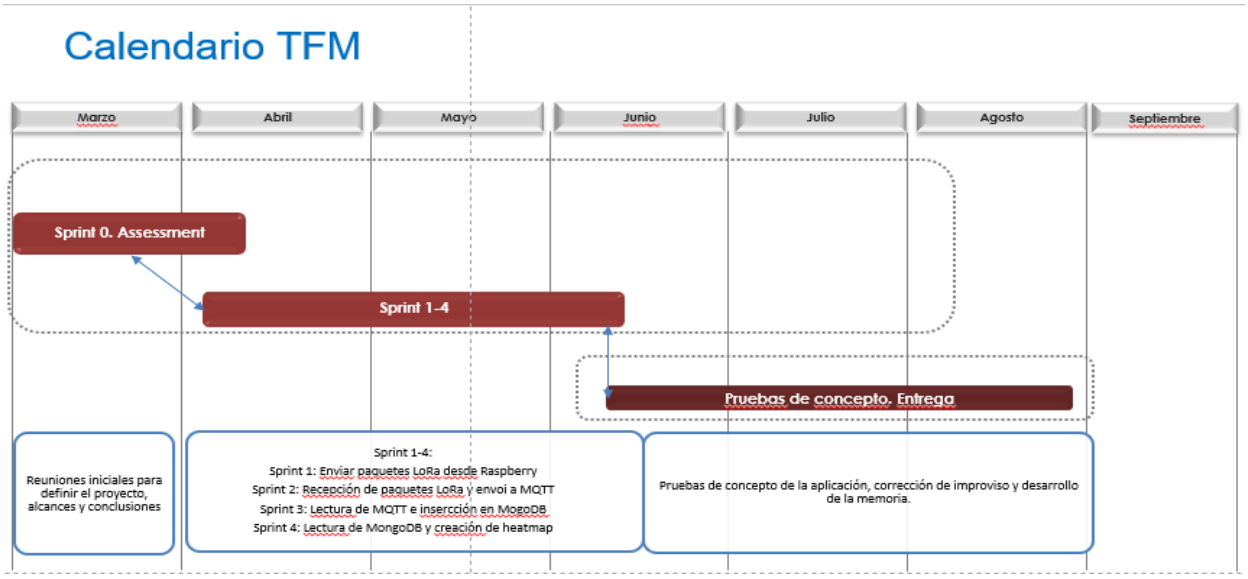

Figura 34. Project calendar.

Conclusions and future work

In conclusion, the tool to facilitate the detailed and automatic study of LoRa network coverage in real environments provides positive functionality when designing IoT architectures. By analysing the terrain with the application, it is possible to analyse in detail the position of the gateways that will collect the LoRa signals from the rest of the sensors, without the need to fully deploy the sensor network to check its correct operation or to correct possible connection errors in architectures that have already been put into production.

On the other hand, this tool can be used to test new technological incorporations such as a new gateway, when generating graphical reports, the possibility of making comparisons between the different possibilities is a handicap in favor of the tool. The boom of the industry 4.0 and the immaturity of new technologies makes that there is still a long way to go for this industry, giving a point in favor for the development of new tools that help the maturity of IoT.

Regarding the technologies used in this project, LoRa is one of the pillars of the application where we must define in advance the parameters to be used, the higher Spread Factor, bandwidth and Code Rating get better shipping rate, less distance and low battery consumption. On the contrary, you get more distance, a higher consumption and a lower shipping rate.

As for the deployment time of the application is relatively fast, it is divided into three parts:

1. The installation in the Raspberry of the necessary software.
2. The configuration of the Gateway.
3. The installation with Docker Compose of the subscriber.

Once deployed the application the data collection depends on the configuration parameters, around 500 data per hour.

As future work, there are two points that would add a positive value to the tool, namely:

- The initialization of the program by means of a button. Currently the application is designed to start when it is connected to the network, and an improvement would be the possibility to start sending data when the user wants. The configuration of the LoRaWAN coverage parameters should continue to be done with a network connection.
- Another contribution to be able to geolocate the LoRa signal without the need for GPS, is to install two beacons that send LoRa packets, if their positions are known it would be possible to calculate the position of the sensor, this contribution would be positive for places where GPS may not give a signal, such as a cave or the interior of a building.