Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de fin de grado

Red IoT privada LoRaWAN con ChirpStack y aplicación de acceso a los datos

LoRaWAN private IoT network with ChirpStack and data access application

Álvaro Rodríguez Gómez

Grado en Ingeniería Informática

La Laguna, 14 de julio de 2023

D. **Alberto Hamilton Castro**, con N.I.F. 43.773.884-P profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"Red privada LoRaWAN independiente y aplicación de acceso a los datos"

ha sido realizada bajo su dirección por D. **Álvaro Rodríguez Gómez**

con N.I.F. **78.766.497-F.**

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

Quiero agradecer a mi madre y a mi padre, por darme la oportunidad de formarme y apoyarme incondicionalmente a conseguir mis metas.

A mi hermano, por haberme motivado a crecer siempre como persona y ayudarme a disfrutar los descansos de estudiar.

A mi pareja, por haber sido mi refugio, mi fuente de tranquilidad y comprensión, y por haberme impulsado siempre a mirar hacia adelante.

A mi tutor Alberto, por desarrollar mi interés en el ámbito del trabajo y hacer mucho más fácil su desarrollo gracias a sus orientaciones y valiosos consejos.

Quiero agradecer también a todos los amigos que he conocido durante estos años. Su compañerismo, alegría y disposición para compartir experiencias han hecho de mi paso por la universidad una etapa inolvidable

Licencia

Resumen

LoRaWAN (Long Range Wide Area Network) es un protocolo de comunicación inalámbrica de bajo consumo diseñado para conectar dispositivos de Internet de las Cosas (IoT) a largas distancias. Ha permitido lograr una amplia cobertura de red, mayor eficiencia energética y una conexión estable para aplicaciones de IoT a gran escala.

En este trabajo proponemos estudiar una red de estas características, para demostrar su funcionamiento y crear una aplicación web para acceder a los datos recabados en ella.

Para su desarrollo, se plantea una red privada LoRaWAN gestionada con ChirpStack y con varios sensores en una zona apartada de la conectividad y cubierta por dos gateways teniendo el servidor de red embebido en uno de ellos.

Se llevará a cabo un estudio de valoración acerca de las ventajas y desventajas de emplear una gestión de red y datos privada frente a utilizar un servicio en la nube ofrecido por terceros. Así como una revisión de conocimientos interesantes para tener criterio sobre las decisiones.

Palabras clave: Internet de las Cosas, red LoRaWAN, aplicación web, ChirpStack, gestión privada.

Abstract

LoRaWAN (Long Range Wide Area Network) is a low-power wireless communication protocol designed to connect Internet of Things (IoT) devices over long distances. It has enabled wide network coverage, higher energy efficiency, and stable connection for large-scale IoT applications.

In this work we propose to study a network of these characteristics, to demonstrate its operation and create a web application to access the data collected in it.

For its development, a private LoRaWAN network managed with ChirpStack is proposed with several sensors in a remote area of connectivity and covered by two gateways with the network server embedded in one of them.

A valuation study will be carried out on the advantages and disadvantages of using private network and data management versus using a cloud service offered by third parties. As well as a review of interesting knowledge to have criteria on decisions.

Keywords: Internet of Things, LoRaWAN network, web application, ChirpStack, private management.

Índice de contenido

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hoy en día, son cada vez más las empresas que están aplicando tecnologías del Internet de las cosas [1] en sus infraestructuras para un sin fin de situaciones: control de dispositivos, control de calidad del aire u otros parámetros del entorno, seguimiento de procesos, etc.

En parte, esto ha sido posible gracias al desarrollo de tecnologías de red diseñadas para este tipo de escenarios, en las que los dispositivos que participan mayoritariamente son inalámbricos, situados a una distancia notable de los receptores y que envían mensajes de un tamaño reducido. Entre estas tecnologías, la más reconocida es LoRaWAN [2].

En determinadas situaciones, como el despliegue de estas redes en localizaciones apartadas, como por ejemplo, granjas, cultivos, valles eólicos, etc. Nos podemos encontrar con algunas limitaciones a la hora de emplear los diferentes sistemas online con los que podemos desplegar estas redes.

1.1. Estado del arte

En internet disponemos de una gran variedad de aplicaciones tanto para desplegar una red LoRaWAN, configurar y gestionar todos los elementos de la red y definiendo reglas de funcionalidades y comportamiento; como para acceder a los datos recibidos en ella, almacenarlos o trabajar con ellos.

También existen redes LoRaWAN públicas como The Things Network [3], en las que podemos registrar nuestros dispositivos y captar esos datos sin necesidad de crear y controlar ningún aspecto de la red por nuestra cuenta.

Además, existen muchas aplicaciones diseñadas para mostrar y acceder a los datos recopilados con muchísimas funcionalidades. También podemos encontrar aplicaciones de este estilo específicas para determinados dispositivos.

1.2. Problemática

Si bien en internet disponemos de una gran variedad de estas aplicaciones y, dependiendo de nuestras necesidades, pueden ser interesantes. En algunas situaciones estas cuentan con una serie de puntos negativos.

En primer lugar, estas opciones son todas online, necesitas una conexión a internet permanente para que el sistema funcione. Esto en algunas zonas apartadas puede suponer directamente la imposibilidad de despliegue. Además, estas compañías tienen un alcance determinado, no tienes una conexión garantizada en cualquier parte del mundo.

Añadido a esa dependencia, perdemos la total seguridad que nos brinda una red privada al exponer nuestros datos a terceros para su control.

Empleando alguna de estas soluciones, no tendríamos libertad absoluta de gestión y personalización de red, así como de la utilidad que podamos implementar.

Por el lado de las aplicaciones web con las que podemos integrar estos gestores de red para acceder a los datos, como My devices Cayenne [4] o Things Board [5], aunque nos brindan un gran beneficio al ser útiles, fáciles de utilizar y poner en funcionamiento en su plan de uso gratuito, tienen una cantidad de dispositivos bastante limitados para un proyecto de talla profesional. Estamos hablando de que en este plan, el máximo número permitido normalmente es de 5 dispositivos, lo cual es una cantidad bastante escasa para la mayoría de casos.

1.3. Objetivo

Pero en este trabajo queríamos estudiar la viabilidad de ser completamente independientes de estos servicios y estudiar cómo desplegar una red LoRaWAN privada, estudiando para ello cuáles son las mejores opciones que tenemos disponibles.

Para cumplir ese propósito, cubriremos una zona de La Laguna con sensores de diferentes magnitudes, dos gateways y un servidor de red que la gestione. Para ello, deberemos configurar todos los dispositivos y elementos de control de la red.

Por otro lado, se creará un programa que reciba y almacene los datos recogidos por la red y una aplicación web donde serán expuestos para su visualización, modificación y uso posterior. Así como un sistema de notificaciones al usuario.

Capítulo 2

Conocimientos previos

Para comenzar este trabajo, el primer paso como siempre, consiste en adquirir el conocimiento relacionado con las tecnologías involucradas, la situación y opciones actuales para dar una solución al problema planteado. A continuación, realizaremos un pequeño análisis de las tecnologías empleadas y varios conceptos que interesa tener claros.

2.1. LoRa

LoRa [6], del inglés Long Range, es una técnica de comunicación inalámbrica por radio. Con ella, un emisor de baja potencia transmite paquetes de datos pequeños (0,3 - 5,5 kbps) a un receptor a larga distancia, con un alcance de generalmente 13 - 15 Km.

2.2. LoRaWAN

LoRaWAN, es un protocolo de comunicación estándar oficial a nivel de red, capa OSI 2-3, de redes basadas en LoRa. Fundada por SemTech y desarrollada bajo el control de LoRa Alliance [7], es un protocolo abierto y sin ánimo de lucro. Por ello, los fabricantes de dispositivos pueden abaratar costes. Dependiendo de la ubicación geográfica, LoRaWAN utiliza una determinada banda de frecuencia. En Europa las bandas utilizadas son 433 MHz y 868 MHz.

Las ventajas de usar esta tecnología son varias: los dispositivos se pueden comunicar a una distancia de hasta varios kilómetros; a pesar de ello, tienen un consumo de energía muy bajo al utilizar técnicas de ADR (Adaptative Data Rate [8]), con esto, los dispositivos pueden durar años utilizando baterías; Además, los dispositivos LoRaWAN tienen un coste muy asequible.

Con estas características, las redes LoRaWAN son altamente escalables al poder enlazar miles de dispositivos utilizando uno o varios gateways. Es una tecnología con una curva de aprendizaje rápida y con un alto nivel de seguridad de extremo a extremo.

2.3. Elementos en una red LoRaWAN

Dentro de estas redes, encontramos diversos elementos, cada uno de ellos con una función específica.

- **Nodos:** son los dispositivos finales que queremos conectar y que enviarán datos en la red. Estos pueden ser sensores, actuadores, trackeadores, etc

- **Gateways o concentradores:** dispositivos encargados de recibir los datos enviados por los nodos mediante LoRa y enviarlos al servidor de red mediante TCP/IP o UDP/IP.

- **Servidor de red:** encargado de gestionar la red, controla los dispositivos y gateways registrados. Procesa las recepciones duplicadas y puede transformar los datos recibidos a un formato especificado.

- **Servidor de aplicación:** es el programa que trabajará con los datos de la red. No tiene ningún control sobre la red. Según el servidor de red, podrá recibir estos datos por diferentes tecnologías como MQTT, API REST u otro del estilo.

Podemos distinguir tres tipos de nodos en una red LoRaWAN [9], en función de su comportamiento y consumo:

- **Clase A:** (All) dispositivos de ahorro máximo de energía. Estos dispositivos solo envían datos cuando ocurre un evento o tras un cierto tiempo programado. Al enviar un mensaje, tienen una corta ventana de tiempo para recibir datos del gateway antes de volver al modo reposo. Estos tipos de sensores pueden llegar a comunicar varios años con una única batería.

- **Clase B:** (Beacon) son dispositivos como los de clase A, pero que permiten configurar el tiempo de recepción.

- **Clase C:** (Continuous) dispositivos activos constantemente por lo que deben estar conectados a una fuente de alimentación.

2.4. Tipos de mensajes

Dentro de la terminología usada en este ámbito, observamos tres conceptos muy importantes relacionados con la comunicación:

- **Chirp:** es la forma de la onda utilizada para transmitir los datos. Su dispersión es en el dominio espectral, esto hace que la modulación en chirps permita que las señales LoRaWAN tengan una mayor resistencia a la atenuación y penetren mejor a través de los obstáculos, lo que da como resultado una mayor cobertura de red en comparación con otras tecnologías de comunicación inalámbrica.

- **Uplink:** se refiere a aquel mensaje de subida de datos emitido por un nodo hacia la red. Estos paquetes, contienen la información

tanto del dispositivo como de los elementos que han intervenido en la captación del mensaje y los parámetros de la recepción.
- **Downlink:** es aquel mensaje que se envía desde la red a un nodo.

2.5. Spreading Factor

El spreading factor [10] (factor de esparcimiento) controla la velocidad de transmisión de datos, es decir, la velocidad a la que se envían los chirps. Hay SF diferentes, del SF7 al SF12, siendo mayor a menor velocidad de transmisión.

A mayor velocidad, mayor transmisión de información pero menor rango debido a que reduce la ganancia de procesamiento. En cambio con SF12 da mayor alcance pero para la misma información se ocupa más tiempo el aire y por ello consume más energía.

La red usa el spreading factor para optimizar la transmisión con los dispositivos y además, minimizar la congestión de la red. Los spreading factores son ortogonales, esto significa que las señales moduladas con diferentes SF que son transmitidas por la misma frecuencia de canal al mismo tiempo no interfieren entre sí.

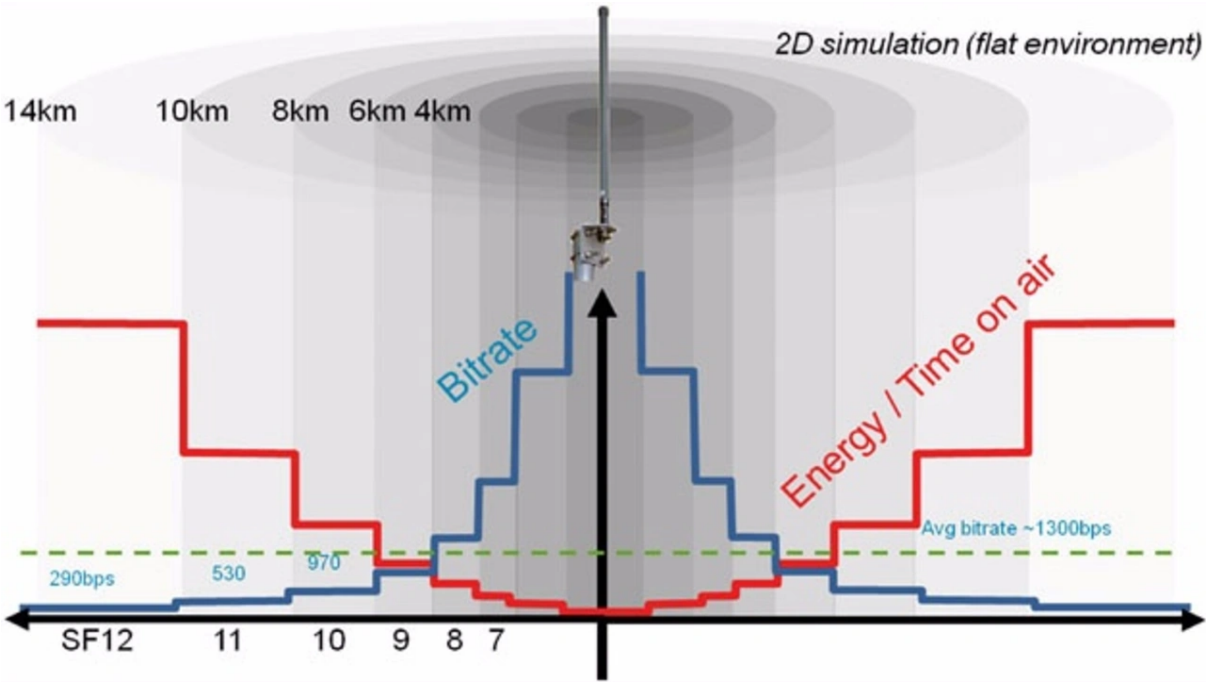


Figura 1.1: relación entre el bit-rate, el spreading factor y la distancia [11]

2.6. Duty cycle

El duty cycle [12] sirve para controlar la congestión de red y del uso general del tiempo en el aire de las redes LoRaWAN en determinadas zonas geográficas.

Determina el tiempo máximo en el cual un dispositivo puede estar emitiendo por el aire como canal.

En Europa y por tanto España, esta relación es del 1%, es decir, establece que si un dispositivo emite durante 1 segundo, deberá esperar 99 segundos para volver a emitir.

2.7. ADR

Es un algoritmo presente en los servidores de red, el cual se encarga de, a partir de los parámetros de los uplinks de un dispositivo, optimizar los parámetros de la comunicación, como son: el spreading factor, la potencia de transmisión y la banda ancha.

2.8. Capacidad

Veamos porque las redes LoRaWAN son tan versátiles. Pongamos el ejemplo de una red LoRaWAN europea que opera en la banda de frecuencia 868 Mhz. Estas redes son capaces de manejar hasta 8 canales en la banda ancha.

A priori, sólo podríamos tener hasta 8 dispositivos conectados simultáneamente. El spreading factor resuelve este problema, recordemos que las señales con diferente SF son ortogonales y que habían 6 diferentes SF, por lo que podríamos tener 6 dispositivos con SF diferentes transmitiendo en cada canal. Con esto aumentaremos a 8 * 6 = 48 dispositivos simultáneos, lo cual sigue siendo bastante pobre.

Aquí es donde entra en juego la regla del 1%, lo cual aumentaría el número de dispositivos conectados por 100. Con lo que conseguimos un total de 4800 dispositivos máximos conectados sin ningún tipo de interferencia.

2.9. Seguridad y sistemas de conexión

La seguridad de las redes LoRaWAN recae en la forma en la que se conectan los dispositivos al servidor de red [13].

- **ABP**: Activation by Personalization, en la que se configuran 3 parámetros en el dispositivo y en el servidor de red. Estos son:
    + **AppSKey:** Application Session Key, la clave con la que se encriptarán los datos.
    + **NwkSKey:** Network Session Key, con la que se autenticará el dispositivo.
    + **DevAddr:** Device Address, identificador único del dispositivo.

  En ABP no hay intercambio de información entre el dispositivo y el servidor de red, el cual no puede indicarle al dispositivo los parámetros de la comunicación como la frecuencia usada, el data-rate ni el SF de downlink.

- **OTAA:** igual que en el método anterior, se necesita configurar los tres parámetros en cada parte. Se diferencia en el método en el que se configura.

  Ahora cada parte deberá tener una credenciales, DevEUI, AppEUI, JoinEUI y AppKey. Y existe un procedimiento de enlace.

  El dispositivo envía un JOIN REQUEST, el servidor de red genera los parámetros del método anterior y se los envía al dispositivo en una trama llamada JOIN ACCEPT, en la cual también se añaden los parámetros de comunicación como la frecuencia usada, el data-rate y el SF de downlink.

  Este método es el usado actualmente por dos motivos:

  - Las claves generadas cambian cada vez que se enciende el dispositivo y envía un nuevo JOIN REQUEST, por lo cual es más seguro.
  - Se optimiza la comunicación al compartir datos útiles para la modulación.

2.10. Chirpstack

Chirpstack [14] es un servidor de red LoRaWAN de código abierto. Ofrece una interfaz web para gestionar gateways y dispositivos, así como integraciones de acceso a datos con terceros, bases de datos u otros servicios para manejar datos.

Es el encargado de gestionar todos los conceptos de los apartados anteriores. Veremos más acerca en la sección de desarrollo del trabajo.

2.11. Opciones en la nube

Para la gestión de este tipo de redes tenemos varias opciones con varias características a favor y en contra. Una opción muy popular cuando se quiere cubrir una zona con dispositivos LoRa es TheThingsNetwork, una red colaborativa de cubrimiento global, en la que un usuario puede registrar sus gateways y dispositivos y TTN se encarga de gestionarlos y notificarte de sus datos recibidos.

Es una opción muy cómoda y eficiente al abstraernos totalmente de la gestión de la red. El problema viene dado por el alcance limitado de esta red. Si queremos desplegar una red de sensores, deberemos hacerlo en una zona cubierta por algún gateway, a menos que registremos uno nosotros mismos que los cubra.

Por otro lado, no sería una red del todo privada, al registrar un gateway en la red, permiten a cualquiera que use TTN que utilice tu gateway, lo cual en determinados casos no es conveniente. Por el mismo motivo, tus

datos no serían del todo privados y además, TTN es significativamente más lento procesando los mensajes recibidos y devolviéndote el resultado.

2.12. MQTT protocol

Aparte de todas las tecnologías y conceptos específicos de LoRaWAN, también utilizaremos otras muy utilizadas en IoT que es interesante conocer.

MQTT [15] es un protocolo de mensajería basado en el modelo de publicación o suscripción mediante clientes y agentes.

Un cliente es cualquier dispositivo que ejecute una biblioteca MQTT. Si el cliente envía datos, actúa como editor y si recibe mensajes, como receptor.

Un Agente o broker es el sistema que coordina los mensajes entre los clientes. Recibe y filtra los mensajes, y los envía a los receptores. Se utilizan **tópicos**, que son básicamente palabras clave a las que pueden atender los elementos de la red.

Los editores podrán enviar datos relacionados con un tema, y los receptores estarán suscritos a estos temas para obtener la información.

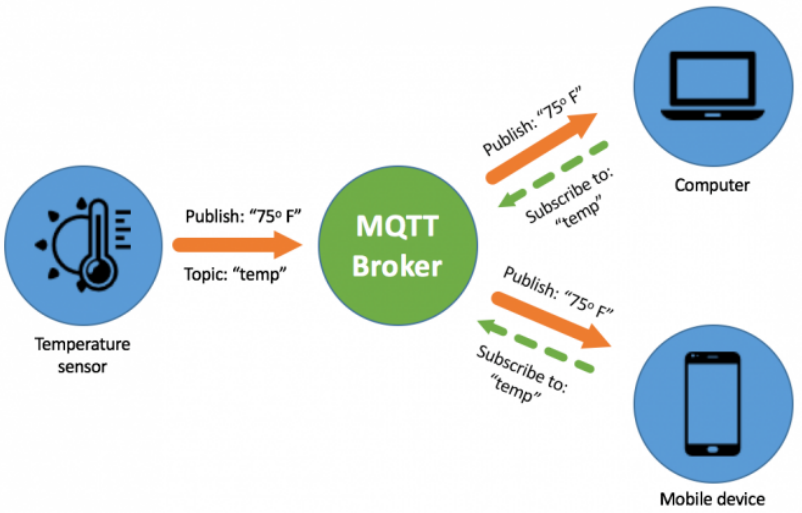


Figura 1.2: arquitectura MQTT [16]

Se ha convertido en un estándar para la transmisión de datos en IoT al ofrecernos numerosos beneficios:

- **Ligereza y eficiencia:** pensado para usar los mínimos recursos, los mensajes MQTT tienen encabezados pequeños para optimizar el ancho de banda. Un mensaje MQTT puede ser de tan solo 2 bytes.

Tiene capacidad para comunicar con millones de dispositivos IoT.

- **Seguridad:** contando con cifrado de mensajes y autenticación de dispositivos con los protocolos más modernos como TLS1.3, OAuth o certificados.
- **Soporte:** numerosos lenguajes como Python tienen una potente implementación del protocolo MQTT. Por lo que su integración en sus programas conlleva una codificación mínima.

Capítulo 3

Desarrollo

En esta sección analizaremos la estructura y componentes de la red propuesta y los pasos llevados a cabo para implementarla a nivel de red y de aplicación.

3.1. Dispositivos

En nuestra red, tendremos varios dispositivos conectados, cada uno con un objetivo específico. Situado en una ubicación diferente para cubrir ciertas magnitudes de interés. Dado que nuestra red únicamente tiene el objetivo de obtener datos, los dispositivos que encontramos en ella son sensores y los gateways que recibirán los datos.

3.1.1. Sensores

Los sensores serán los encargados de recoger la información del entorno y enviar los datos recibidos. Entre ellos encontramos:

LHT65 Temperature & Humidity Sensor

Figura 2.1: sensor de humedad y temperatura DRAGINO LHT65 [17]

Sensor de temperatura y humedad de la marca DRAGINO. Permite además controlar su batería. Disponemos de dos sensores. Cuenta con una batería integrada con una duración aproximada de entre 5 y 10 años.

AI Workplace Occupancy Sensor MileSight VS121

Figura 2.2: sensor de presencia MileSight [18]

Sensor apoyado por IA con capacidad de conteo de personas en una zona, o detección de cruzado de una línea determinada. Está alimentado por cable.

Ambience Monitoring Sensor MileSight AM307

Figura 2.3: sensor de varios parámetros de calidad del aire MileSight [19]

Sensor que permite controlar 9 parámetros diferentes de la calidad del aire. Cuenta con una opción de funcionamiento con baterías o de manera enchufada.

3.1.2. Gateways

Los gateways serán los encargados de capturar los mensajes enviados por los sensores y dirigirlos al servidor de red para ser procesados. Dispondremos de un gateway principal que aloja el servidor de red y un gateway secundario para ampliar la cobertura de la red.

RAK7244 LoRa Gateway

Figura 2.4: RAK7244 Gateway [20]

Gateway principal, con todos los componentes necesarios: network server, LoRa Gateway Bridge y packet forwarder. Además, tiene una Raspberry Pi 4 integrada, en la cual se alojan todos los programas que implementaremos en el trabajo y veremos en siguientes secciones.

Dragino LG308 Gateway

Figura 2.5: Dragino LG308 Gateway [21]

Gateway secundario de la marca dragino, será usado para aumentar la integridad de nuestra red y observar el funcionamiento de una red con más de un gateway, la eliminación de mensajes duplicados y demás.

3.2. Aplicación

Como hemos visto, Chirpstack se encarga únicamente de gestionar la red. Controlar la conexión de los dispositivos, traducir los mensajes recibidos al formato que nosotros le hayamos indicado y publicar los resultados en un MQTT Broker.

Nuestro objetivo es desarrollar una aplicación que atienda a estos eventos y nos permita acceder y trabajar con los datos recibidos por los

dispositivos, añadir datos sobre ellos, como su ubicación y definir métodos para visualizar los datos.

3.2.1. Gateway RAK7244

Todo este trabajo será desarrollado en nuestro gateway principal, el cual tiene un sistema operativo en el que está instalado el servidor de red Chirpstack. Aquí se almacenarán los datos y se ejecutarán todos los procesos necesarios para la captura y la visualización de datos.

El RAK7244 viene con una tarjeta microSD con el sistema operativo. En su primer uso, es muy recomendable actualizar el firmware para tener una instalación limpia de su última versión. Para ello, deberemos seguir las instrucciones oficiales de RAK [22].

La instalación es muy sencilla, deberemos encender el gateway LoRaWAN, asegurándonos de que la antena está conectada antes de encenderlo. Para acceder al gateway optamos por conectarlo con un cable ethernet a un router y desde su página de configuración, obtener la ip que le había sido asignada. A la cuál, accederemos vía ssh con usuario: pi y contraseña: raspberry.

Una vez en el gateway, lo primero será actualizar el sistema operativo y configurar las características del propio gateway con el comando `sudo gateway-config`. Se nos mostrará la pantalla de la figura 3.1:

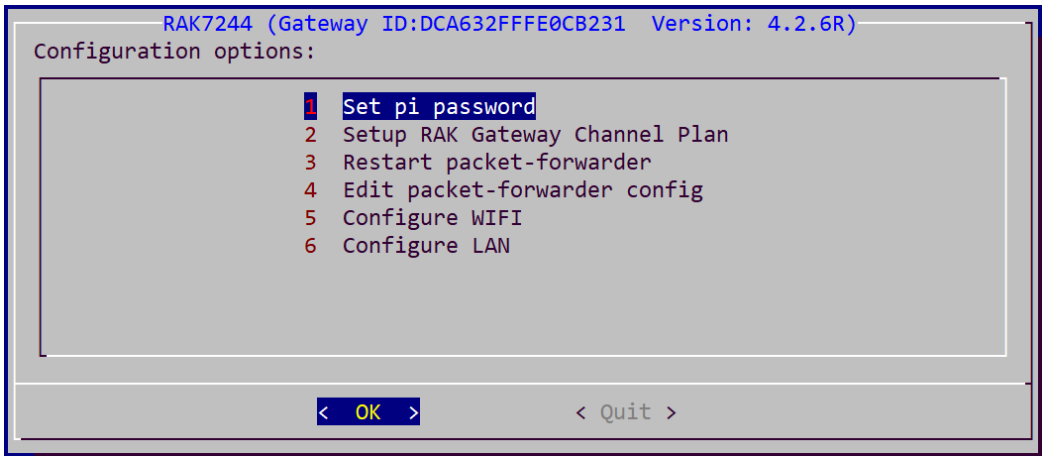


Figura 3.1: pantalla de configuración del gateway principal

Deberemos rellenar cada opción acorde a nuestra situación, en nuestro caso, en la opción 2, nuestro servidor de red es Chirpstack y la banda de frecuencia es EU_863_870 y activar el servicio de ADR.

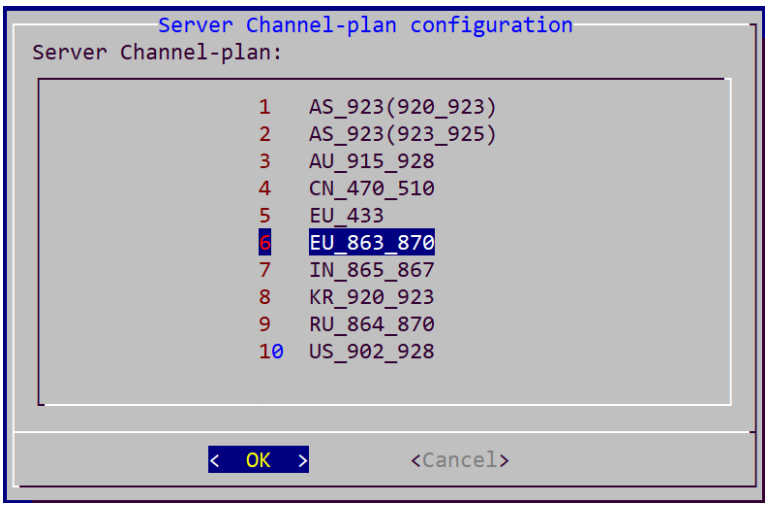


Figura 3.2: pantalla de selección de frecuencia del servidor

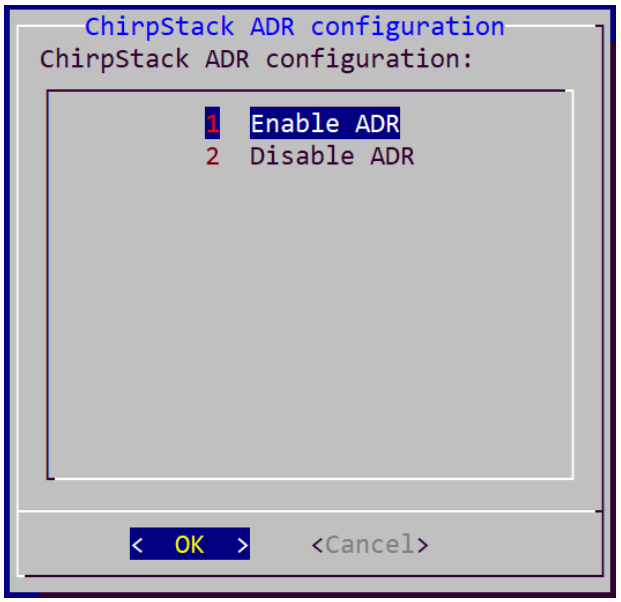


Figura 3.3: activar ADR

Luego para una conexión más cómoda decidimos en el apartado 5 configurar el gateway como cliente WiFi con las credenciales correspondientes.

Un sistema operativo actualizado y configurado para alojar todos los procesos del proyecto debe contar con las librerías y software necesarios. He decidido implementar tanto el frontend y el backend en Flask [23], debido a la facilidad de implementación y flexibilidad que aporta. Además de la gran cantidad de librerías útiles de manejo de datos que presenta al estar basado en Python.

Hay que tener en cuenta que no es el lenguaje más veloz en tiempo de ejecución, algo deseable en redes con un gran número de dispositivos y

cantidades ingentes de datos a procesar. La solución que proponemos no está pensada para ese tipo de redes, dado que el proyecto está pensado para clientes con menores requerimientos.

Ahora podemos pasar a usar la aplicación web de Chirpstack para configurar lo necesario para nuestra red. Esta aplicación se encuentra ejecutándose en el puerto 8080. La primera vez accederemos con user: admin y contraseña: admin.

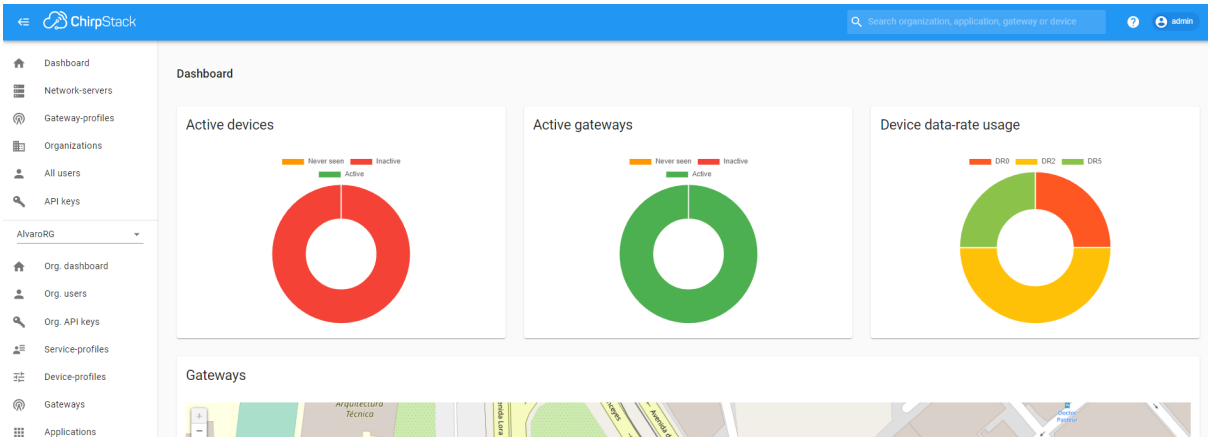


Figura 3.4: pantalla principal ChirpStack

Hagamos un rápido repaso por las funcionalidades que nos brinda Chirpstack:

- **Network server:** nos permite crear nuestro servidor de red. Usaremos el que viene creado por defecto ya que no necesitamos ninguna funcionalidad especial.

- **Gateway profiles:** configurar parámetros para conectar gateways de un determinado perfil o tipo.

- **Organizaciones:** crear un grupo con una serie de permisos sobre la gestión de la red.

- **Service profiles:** definir que tipo de acciones realizarán ciertos elementos de la red.

- **Device profiles:** cada tipo de dispositivo tendrá tu perfil, en el que se indica que tipo de dispositivo es, versiones y características referentes a cómo se conectará a la red. Aquí podemos definir la forma en la que se transforma el payload binario a un formato legible especificado por nosotros.

- **Gateways:** apartado donde registramos los gateways que podrán enviar información al network server. Podemos monitorizar el estado y características de las conexiones de estos.

- **Applications:** chirpstack define una aplicación como una agrupación de dispositivos conectados. Cada una de estas aplicaciones publicará eventos de sus dispositivos.

  En este apartado es donde creamos la aplicación que usaremos, llamada Control-TFG. En ella, indicamos qué dispositivos forman parte de ella. Para registrar un dispositivo, tendremos que rellenar sus campos de información general y , dependiendo de la versión de loraWAN que empleen, insertar las claves de conexión.

  Sabremos que lo hemos hecho bien si empezamos a recibir uplinks.

Una vez que tenemos la red creada y funcionando, pasamos a desarrollar el servidor de aplicación. Cuando nuestros dispositivos envíen datos, estos serán publicados en el MQTT Broker del servidor de red. La idea será almacenar todos estos datos y metadatos y mostrarlos en una aplicación web.

3.2.2. Ejemplo de conexión al servidor de red

Veamos qué pasos debemos seguir para conectar un dispositivo a nuestra aplicación. El dispositivo con el que haremos la prueba será el Milesight-307.

Deberemos crear un perfil para el dispositivo en el que definiremos las características de este y como funcionará con la red. Rellenamos el formulario de chirpstack donde indicamos parámetros como la versión del dispositivo, el modo de conexión OTAA o ABP, el algoritmo ADR, la clase de dispositivo y lo más interesante, cómo se interpretará el payload de los uplinks. Para definir esto, chirpstack nos permite ingresar funciones javascript para codificar y decodificar mensajes. Estas funciones son dadas normalmente por el creador del dispositivo y son abiertamente modificables. En nuestro caso se encuentran en el manual de instrucciones de la página web del dispositivo.

Vamos a conectar los dispositivos con OTAA, por lo que necesitaremos las tres claves, el DevEUI, AppEUI y AppKey. En este caso, descargamos la aplicación de móvil Milesigh ToolBox, siguiendo las instrucciones y accederemos al sensor con NFC para obtener los datos necesarios.

Antes de añadir el dispositivo a la red, hay que calcular y configurar el tiempo de transmisión para cumplir con la normativa del 1% duty cycle.

Para ello debemos obtener dos valores: el tamaño de la cabecera del mensaje y el de su payload. Utilizando una calculadora de AirTime [24], para este dispositivo estos valores son:

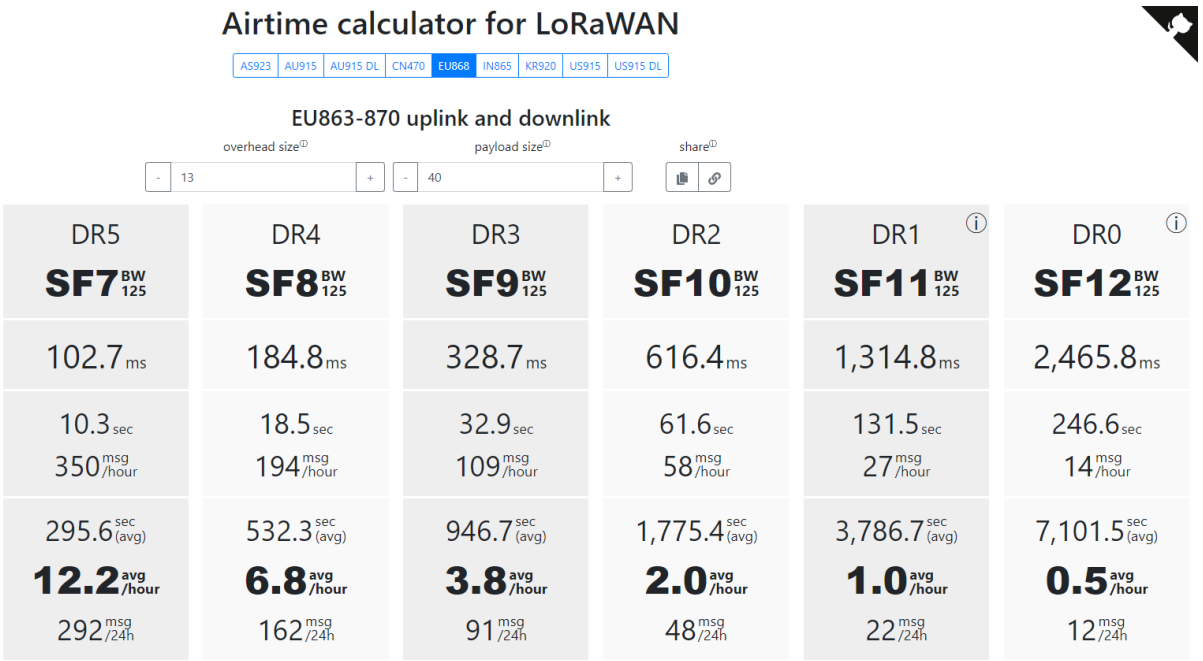


Figura 3.5: calculadora de AirTime

Dado que en Europa se utiliza la banda EU 863-870 y que los dispositivos están configurados para emitir en un máximo de SF9. En el caso de este dispositivo podríamos enviar un máximo de 3.8 mensajes por hora con SF9. Configuraremos con la aplicación este dispositivo para que emita cada 25 minutos.

Ahora podremos ingresar estas claves al crear dispositivo en el apartado de aplicaciones y comenzar a recibir uplinks.

Una vez se reciba el primer uplink, el ADR realizará su función con los datos recibidos y definirá el valor de los parámetros de comunicación óptimos, el spreading factor, la frecuencia y el data-rate, entre otros. El servidor enviará un downlink al dispositivo para indicarle los parámetros. Esta actividad se repetirá cada vez que el servidor detecte un cambio en estos parámetros.

Figura 3.6: mensajes intercambiados con el dispositivo

Como vemos, el dispositivo emite los mensajes de unión y emite su primer paquete en SF12, con ello, el gateway utiliza el ADR para optimizar la comunicación y dado que el dispositivo se encuentra cerca, se le asigna el SF7, con el que continúa emitiendo próximos eventos.

3.2.3. Backend

Abordemos el tema de la recepción y almacenaje de datos. Hemos tomado la decisión de usar python para desarrollar tanto el back end como también el frontend como veremos en futuros apartados.

Todo el código y ficheros creados a lo largo del trabajo se encuentran en el repositorio de github del trabajo [25]. Incluye un fichero README.md el cual contiene una explicación de los ficheros existentes.

Para el almacenamiento de datos se ha optado por un gestor de base de datos PostgreSQL [26] por su gran potencia y flexibilidad, así como su cómoda integración con python.

```
Unset
sudo apt install postgresql postgresql-contrib
```

Para inicializar la base de datos deberemos crear el usuario gestor y la base de datos en sí:

```
Unset
sudo -u postgres createuser --createdb --pwprompt admin
```

```
>password? = passwd
sudo -u postgres createdb admin app
# Ejecutar el script que crea las tablas
psql --host localhost --user admin
\i /ruta/al/script.sql
```

Para almacenar los datos que nos parecen interesantes hemos optado por diseñar el siguiente esquema de tablas:

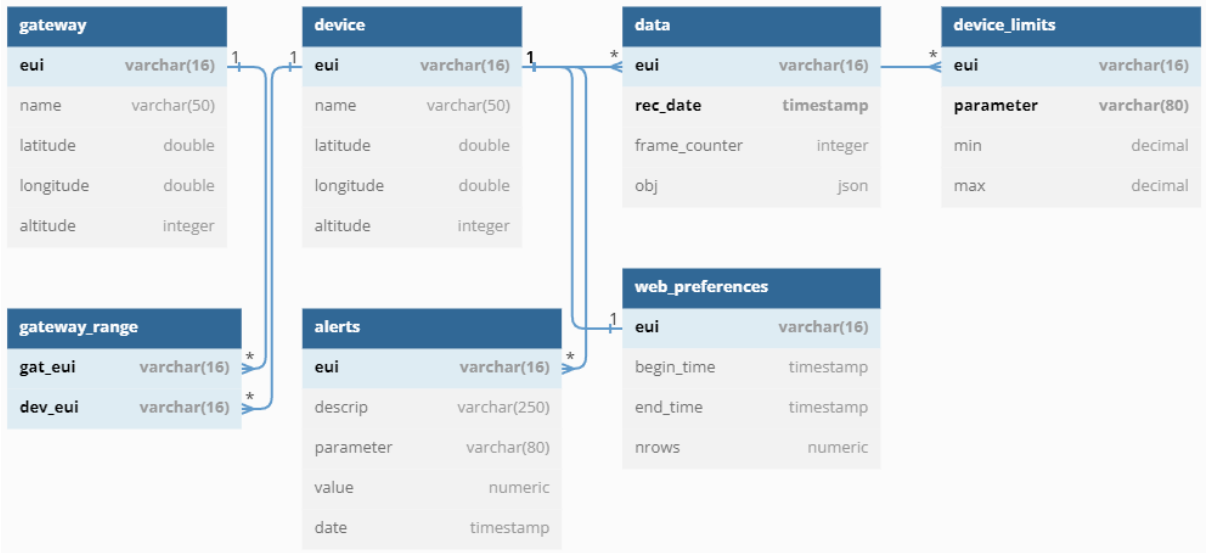


Figura 3.7: esquema de la base de datos

- **gateway:** los datos principales de cada gateway. Dado que es chirpstack quien se encarga de gestionar todo acerca de los gateways, la información que queremos de ellos es meramente informativa.
- **device:** los datos principales de cada dispositivo. Nombre, identificador y ubicación.
- **data:** en ella guardamos los datos recibidos en un momento determinado por un dispositivo. Aquí nos enfrentamos a una decisión de diseño. Cada dispositivo recopila datos de diferentes eventos y por tanto, tendrán una cantidad variable de campos en cada registro.

  Por ello, almacenamos el registro en formato JSON. Haciendo un crossover muy flexible entre la base de datos SQL con una parte noSQL. Este objeto JSON, es totalmente flexible y modificable en un futuro.
- **device_limits:** los límites establecidos por el usuario a cada parámetro de un sensor.

- **alerts:** almacena aquellos registros de cada dispositivo en los que un parámetro excede los límites impuestos a sus valores y una descripción.
- **gateways_range:** la relación entre dispositivos y gateways para conocer la cobertura de estos.
- **web_preferences:** todas aquellas configuraciones persistentes de la web como el rango de tiempo para mostrar datos.

Hemos diseñado e implementado la base de datos de manera robusta, definiendo condiciones de integridad para no introducir ningúna información incorrecta.

3.2.4. Entorno virtual python

Los programas serán implementados en Python, para ello, debemos instalar el intérprete y todas las librerías que vamos a utilizar. Para llevar un correcto estilo de desarrollo y manejar cómodamente estas librerías las instalaremos en un entorno virtual [27].

Deberemos realizar las siguientes instalaciones:

```
Unset
sudo apt install python3.10-distutils python3-dev libpq-dev
build-essential software-properties-common
```

Con esto ya podremos crear y activar nuestro entorno virtual, donde instalaremos todos los paquetes necesarios:

```
Unset
# Creamos el entorno virtual
python3 -m venv venv

# Activamos el entorno virtual
source venv/bin/activate
# Instalamos python y las librerías
wget https://www.python.org/ftp/python/3.10.11/Python-3.10.11.tgz

python3-psycopg2

pip3 install flask paho-mqtt pandas plotly requests fiona shapely
pyproj rtree
```

3.2.5. Script de recepción de datos

Ahora que ya tenemos donde almacenar los datos, crearemos el script en python que procesa los eventos publicados en el MQTT Broker del servidor de red.

Lo que hace el script es muy simple, está constantemente escuchando el endpoint del MQTT Broker. Cuando detecta un registro, en base a la información que contenga hará una acción determinada.

Los eventos que nos interesan de este MQTT Broker son los eventos de uplink, los de recogida de datos. Los demás, como los eventos de JoinRequest o similares, serán ignorados por nuestra aplicación.


```
{
  "applicationID": "2",
  "applicationName": "LoRaWAN-Network-AlvaroRG",
  "deviceName": "HyT-DRAGINO-2",
  "deviceProfileName": "DEVPROF-DRAGINO-EU868",
  "deviceProfileID": "195e9f30-bce8-408f-9764-c516fc871899",
  "devEUI": "a84041a8318279bb",
  "rxInfo": [
    {
      "gatewayID": "a840411eb3644150",
      "uplinkID": "ef0e776b-bd79-403c-adfa-26f0f44e5307",
      "name": "DraginoLG308-Gateway",
      "time": "2023-07-11T19:06:26.164117Z",
      "rssi": -72,
      "loRaSNR": 9.2,
      "location": {
        "latitude": 28.47901826402084,
        "longitude": -16.312116980552677,
        "altitude": 6
      }
    },
    {
      "gatewayID": "dca632fffe0cb231",
      "uplinkID": "2db22df6-9362-48b8-b0a4-11a583e6b6f4",
      "name": "RAK7244",
      "rssi": -117,
      "loRaSNR": 7.2,
      "location": {
        "latitude": 28.47915972230821,
        "longitude": -16.312181353569034,
        "altitude": 12
      }
    }
  ],
  "txInfo": {
    "frequency": 867300000,
    "dr": 5
  },
  "adr": true,
  "fCnt": 7,
  "fPort": 2,
  "data": "y/cKoAJeAQqlf/8=",
  "object": {
    "BatV": 3.063,
    "Hum_SHT": "60.6",
    "TempC_DS": "27.25",
    "TempC_SHT": "27.20"
  }
}
```


Figura 3.8: ejemplo uplink.

En la imagen 3.8 podemos ver los campos que contienen estos eventos. Contienen información del dispositivo que envió los datos, sus datos del registro en chirpstack y lo que nos interesa a nosotros, los datos de los gateways que han recibido ese uplink y el registro de datos.

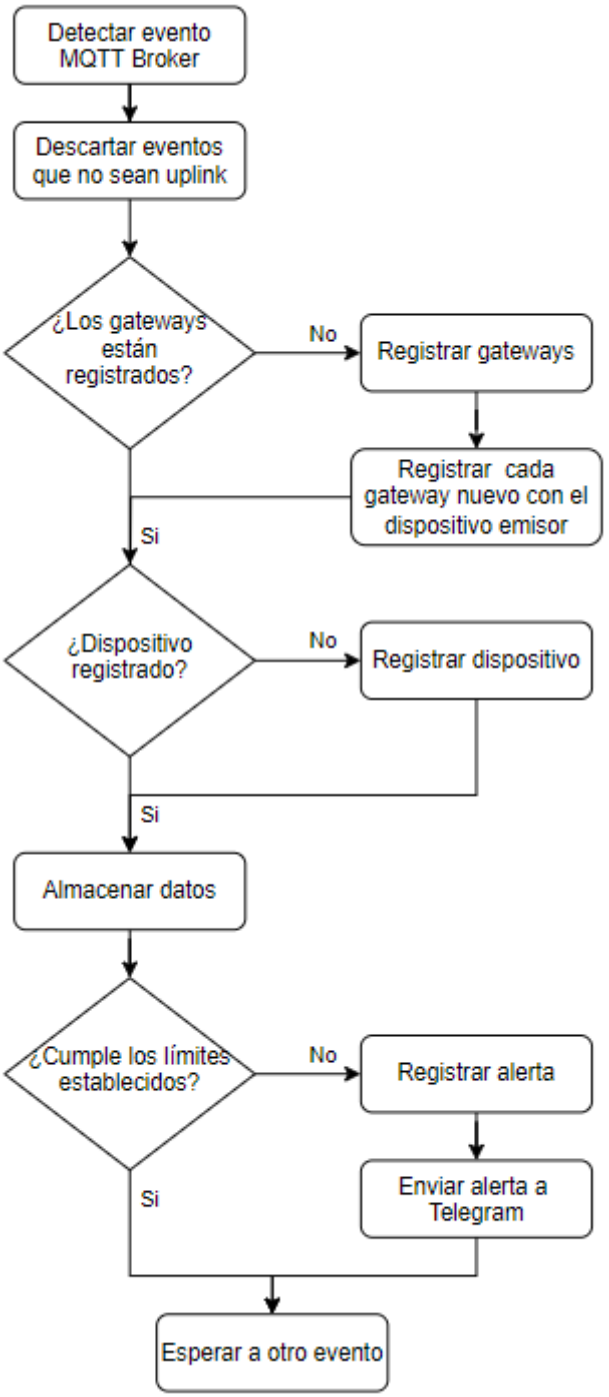


Figura 3.9: diagrama de flujo del script de recepción.

Como podemos observar en la figura 3.9, el script después de comprobar que el mensaje es un uplink, comprueba si están registrados tanto el dispositivo como cada uno de sus gateways. Si no lo están, los registra en

la base de datos. Luego, en ambos casos, el registro se almacena correctamente con el timestamp en el que se recibió.

```python
def on_message(client, userdata, message):
  """
  Define the function for the callback
  """
  message = str(message.payload.decode("utf-8"))
  msg = json.loads(message)

  if (len(msg) > UPLINK_MESSAGE_SIZE):
    eui = msg["devEUI"]
    if search_eui(eui) == None:
      register_device(eui, msg["deviceName"],
                      msg["rxInfo"][0]["location"]["latitude"],
                      msg["rxInfo"][0]["location"]["longitude"],
                      msg["rxInfo"][0]["location"]["altitude"])
    else:
      save_data(msg)
      check_limits(msg)
  look_for_new_gateways(msg)
```

Además, comprueba si existe algún límite impuesto a los parámetros del dispositivo emisor. Si existe, comprueba que esté en el rango permitido y en caso contrario, almacena un registro de alerta y el momento de recepción y envía la alerta a un chat de telegram donde se notifican estos avisos. Podemos ver en el cuadro de código superior la función que se ejecuta cada vez que se detecta un evento en el MQTT Broker.

3.2.6. Bot de telegram para alertas

Cuando se detecte una alerta, se enviará un mensaje a un canal de telegram con los detalles de la misma. Con esto, el usuario puede estar al tanto de cualquier situación esté donde esté.

Para crear el bot [28], telegram nos brinda una forma muy cómoda. En la app de telegram buscaremos el canal BotFather y realizar los siguientes comandos:

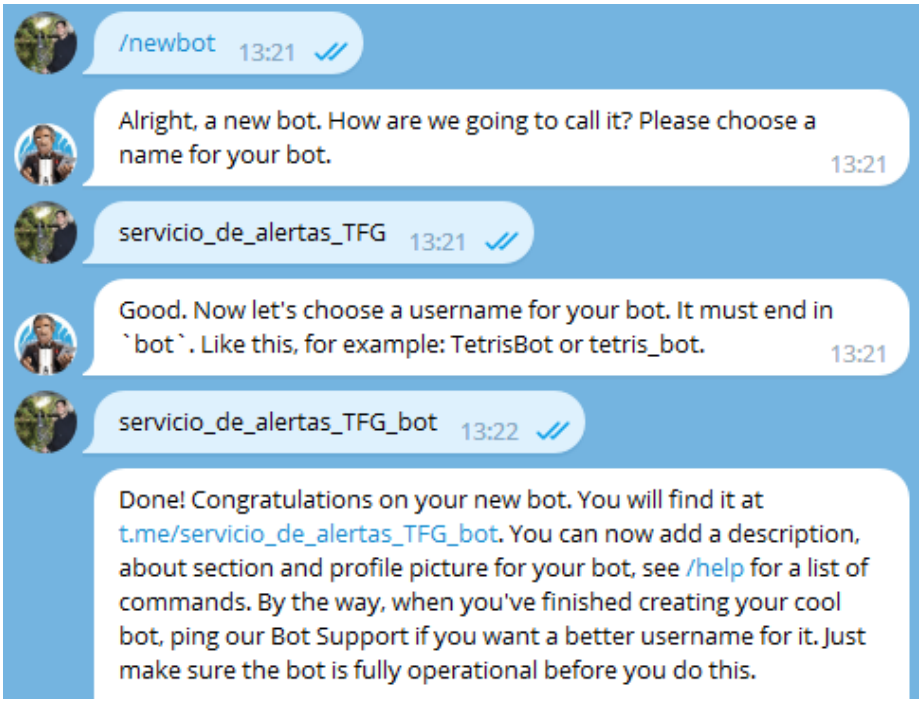


Figura 3.10: creación del bot con Bot Father.

Daremos de alta nuestro bot, con el cual podemos interactuar por medio de la API de telegram. Para ello, deberemos introducir en nuestro código las credenciales y los tokens para utilizarla. En nuestro caso, como observamos en el siguiente código, básicamente con los datos ya formateados en una cadena en Markdown, construimos la petición a la API. Una vez que la realicemos, se enviará el mensaje.

```python
def send_alert(msg):
  bot_token = open(".telegram_bot_token").read()
  bot_chatID = 'XXXXXXXXX'
  send_text  = 'https://api.telegram.org/bot' + bot_token
  send_text += '/sendMessage?chat_id=' + bot_chatID
  send_text += '&parse_mode=Markdown&text=' + msg
  requests.get(send_text)
```

Figura 3.11: chat de telegram con mensajes de alerta

3.2.7. Flask

La aplicación será creada por el framework minimalista en Python para desarrollar aplicaciones web Flask. Nos proveé una forma muy sencilla y con pocas líneas de código para desplegar una aplicación web.

Las aplicaciones de Flask permiten desarrollar tanto el frontend con templates de Jinja2 [29] como el backend. El contenido estático del frontend que representa a las diferentes pantallas de la app se aloja en el directorio templates.

Hace que sean sencillas ciertas operaciones de las páginas web. En resumidas cuentas, para cada vista, tenemos una ruta definida, en la cuál se ejecuta un método con las operaciones de backend necesarias. Para pasarle datos necesarios a una vista, Flask nos permite pasarlos como parámetros al renderizador de cada vista, de esta manera, podemos utilizar estas variables en el contenido html.

Algo muy útil, es que nos permite aplicar el concepto de herencia. Esto se consigue definiendo llamadas a bloques de código en una vista, los cuales se sustituyen por el código html definido en cada uno de ellos.

Otra característica interesante que nos atrajo fue el hecho de que a la hora de recibir los datos de un formulario, flask automáticamente analiza

las entradas y las sanitiza para protegerse frente ataques de inyección sql.

3.2.8. Aplicación web

Como hemos visto vamos a implementar la aplicación con Flask y para el apartado visual, en un principio, lo desarrollamos con el framework css Materialize [30], con el que nos hemos familiarizado a lo largo del grado. El problema aparece cuando ciertas funcionalidades ya no funcionan debido a que materializecss ya no tiene soporte. Por lo cual, decidimos emplear el framework más usado a nivel mundial Bootstrap [31], al trabajar con flask, usaremos su versión de Bootstrap-flask.

El backend está en su totalidad contenido en el programa ejecutable app.py. En ella, se implementa el código que resuelve todas las peticiones del frontend.

La aplicación web permite por un lado, representar la información de la estructura de la red, es decir, ubicación e identificación de dispositivos en la red. Y, acceder a los datos que capturan estos dispositivos de forma gráfica y cómoda.

La pantalla principal resume el primer punto. La ubicación de los dispositivos en un mapa y una lista de los dispositivos conectados, tanto gateways como dispositivos.

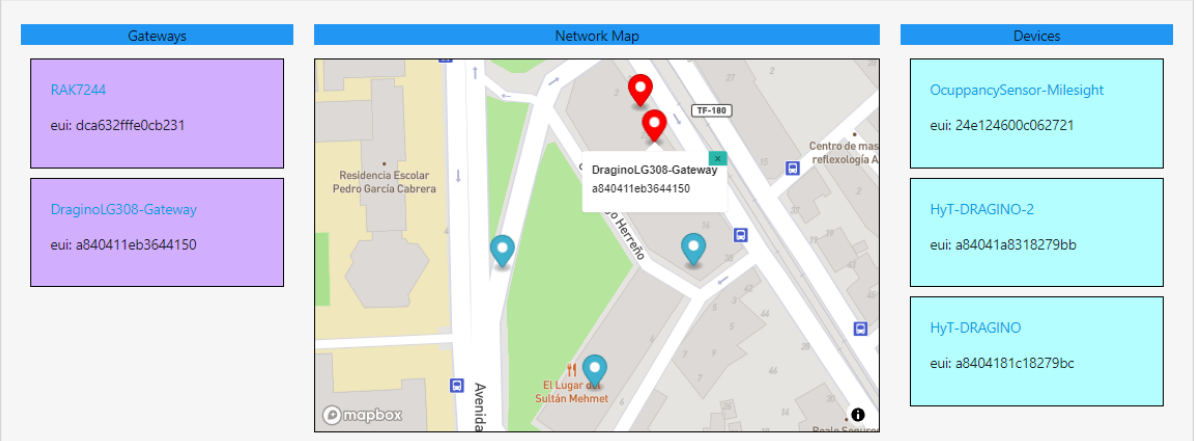


Figura 3.12: página principal de la aplicación

Cada punto en el mapa representa un dispositivo y al hacer clic sobre él, nos indica su nombre y eui para identificarlo.

```python
@app.route('/')
def index():
  conn = get_db_connection()
  cur = conn.cursor()
    cur.execute('SELECT eui, name, latitude, longitude, altitude FROM device;')
  devices = cur.fetchall()
   cur.execute('SELECT eui, name, latitude, longitude, altitude FROM gateway;')
  gateways = cur.fetchall()
  cur.close()
  conn.close()
    return render_template('index.html', devices_ = devices, gateways_ = gateways)
```

Como vemos, en el backend solicitamos los datos de los dispositivos y su ubicación. El frontend emplea un proveedor de mapas en línea llamado mapbox [32]. Con él, se calcula el centroide de los dispositivos para centrar el mapa en ese punto y ajustar el margen automático para que el zoom sea óptimo.

Si clicamos en el nombre de algún dispositivo, esto nos mostrará todos sus datos identificativos, ubicación exacta explícita, el último registro recibido y una gráfica por cada una de sus magnitudes.

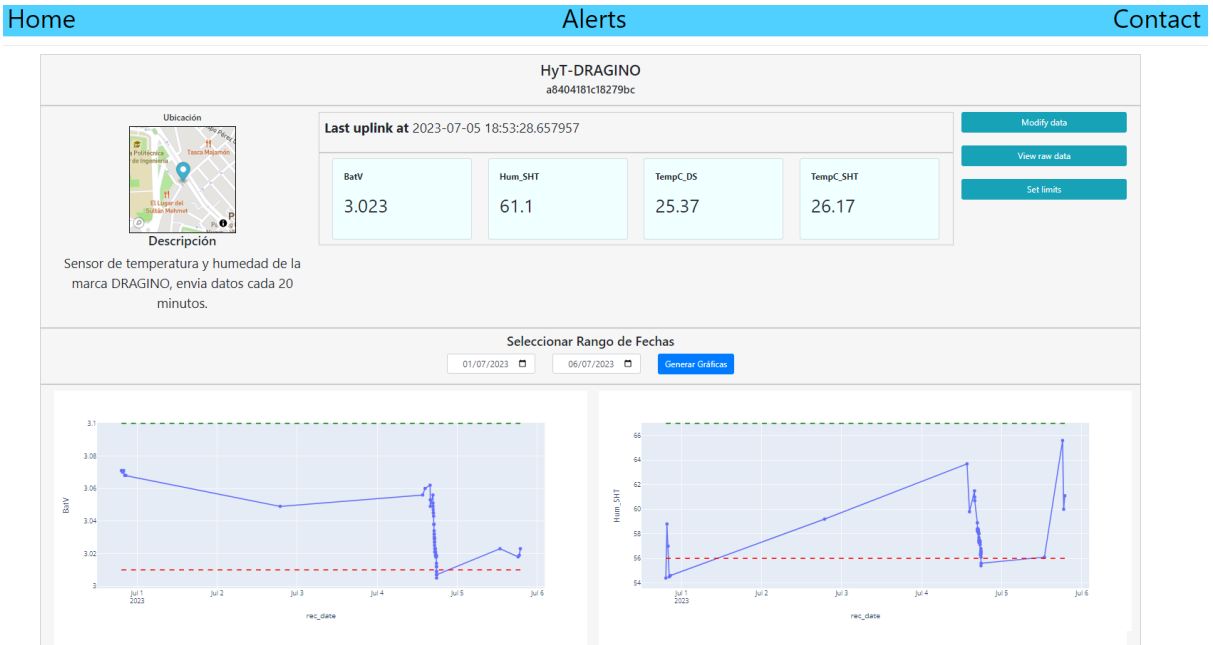


Figura 3.13: pantalla de datos del dispositivo.

El backend se encarga de obtener todos los datos registrados y crear un data frame normalizado con cada una de las magnitudes captadas y la fecha de recepción, así como añadir a las gráficas los límites que podemos definir a los parámetros

El frontend, se encarga de mostrar todos los datos propios del dispositivo y de renderizar las gráficas que muestran los datos de los data frames con la librería Plotly [33]. Estas gráficas son interactivas, el usuario puede acercar el cursor a los puntos y ver los datos a detalle, o seleccionar zonas de la gráfica para hacer zoom en ellas. Por defecto se mostrarán todos los datos recibidos por los dispositivos, pero podremos indicar el rango de tiempo que queremos mostrar. Estas preferencias, se almacenan en la base de datos de forma individual para cada dispositivo, para que el usuario no tenga que estar introduciendolas continuamente.

Figura 3.14: gráficos interactivos

Además, las páginas son responsive y el formato se actualiza cada vez que hay un cambio en el tamaño de pantalla gracias al manejo de eventos con javascript de los elementos visuales y los cambios de estado de algunos componentes.

He decidido implementar otras funcionalidades a la aplicación para que se pueda trabajar más con los datos. Se ha añadido una función en la que el usuario puede especificar los límites en los valores dentro de los cuales observamos una situación nominal. Estos límites se muestran, si existen, sobre las gráficas.

Si en una recepción se detecta un valor que sobrepasa los límites, se notificará al usuario con una alerta por telegram como habíamos visto. Estas alertas, se guardan en una memoria, para que el usuario pueda ver el historial de alertas y borrarlas si ya no le interesan.

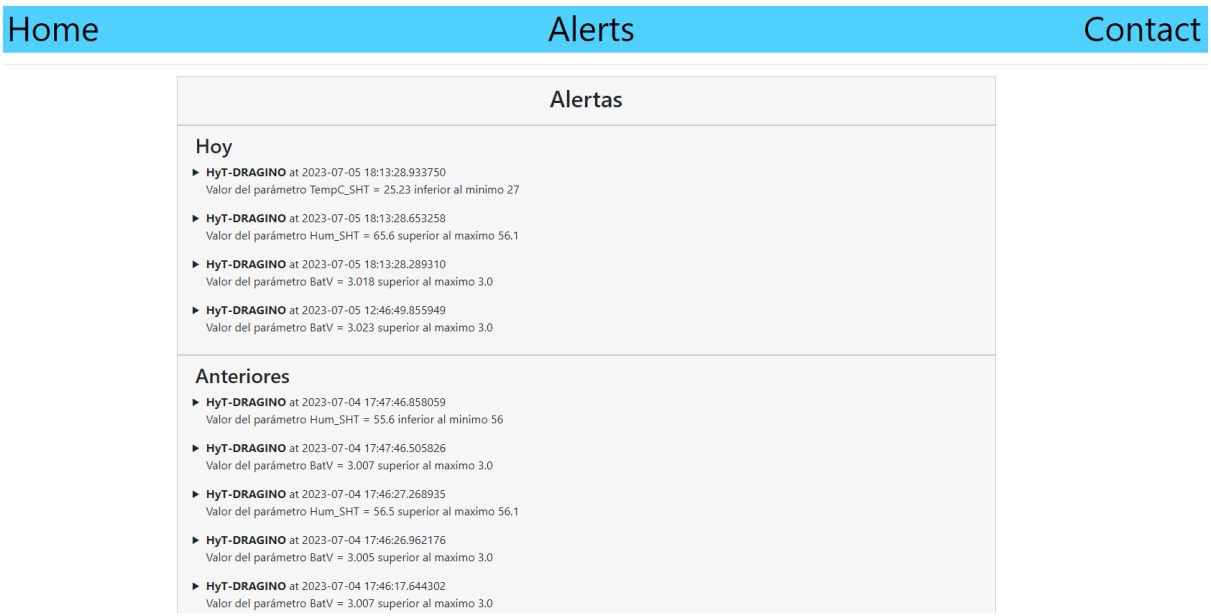
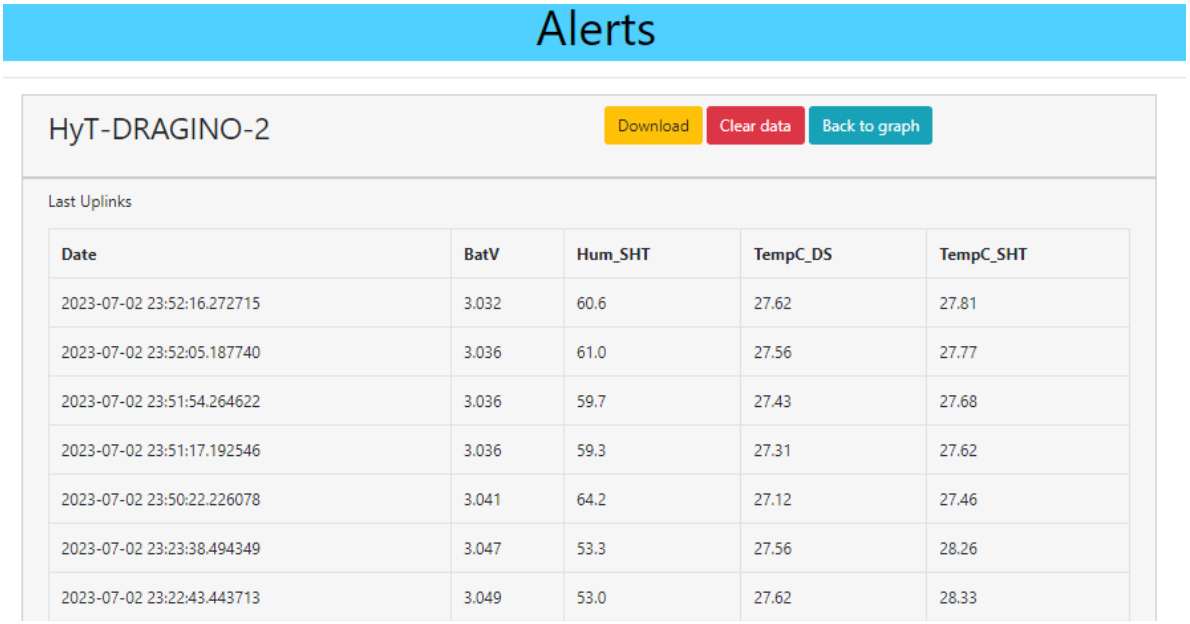


Figura 3.15: pantalla de alertas

Dichas alertas aparecerán resumidas en la página de alertas. Separadas por las alertas recibidas hoy y otra sección para las anteriores. Para cada una de ellas, se muestra el dispositivo que y la hora a las que se recibió, seguido del parámetro que obtuvo tal valor que traspasó los límites impuestos.

Si clicamos el botón “raw data” se mostrará una tabla con todos los datos recibidos por el dispositivo, en esta pantalla, podremos limpiar los datos o descargar un archivo con los datos.



Date	BatV	Hum_SHT	TempC_DS	TempC_SHT
2023-07-02 23:52:16.272715	3.032	60.6	27.62	27.81
2023-07-02 23:52:05.187740	3.036	61.0	27.56	27.77
2023-07-02 23:51:54.264622	3.036	59.7	27.43	27.68
2023-07-02 23:51:17.192546	3.036	59.3	27.31	27.62
2023-07-02 23:50:22.226078	3.041	64.2	27.12	27.46
2023-07-02 23:23:38.494349	3.047	53.3	27.56	28.26
2023-07-02 23:22:43.443713	3.049	53.0	27.62	28.33

Figura 3.16: pantalla de datos en crudo

La aplicación también permite modificar los datos de los dispositivos como su ubicación y descripción, por si el usuario desea colocarlo en otra localización.

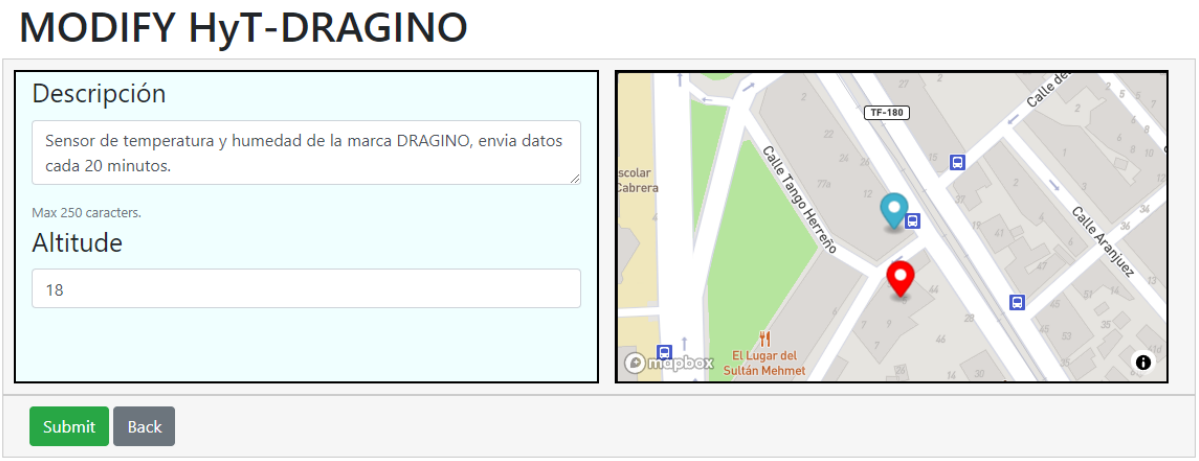


Figura 3.17: pantalla modificar datos

Por la parte de los datos de los gateways, aunque la aplicación esté enfocada a representar los datos de los nodos, para cada gateway, se muestran los dispositivos de los cuales ha recibido información, mostrando el momento de su último uplink.

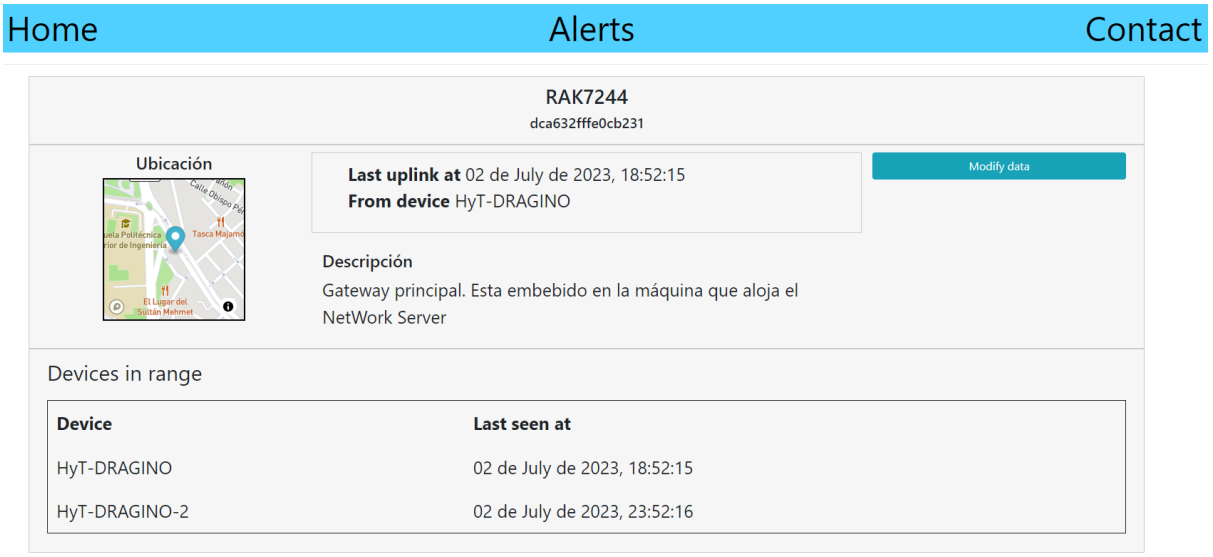


Figura 3.18: pantalla de datos de un gateway

Además, también se permite modificar el gateway de la misma manera que a un dispositivo, cambiando su ubicación y descripción.

El desarrollo de esta aplicación web se ha llevado a cabo también bajo un paradigma de código robusto para que cualquier problema que tenga el usuario, no sea perjudicial para el estado del sistema y se le de toda la información necesaria para subsanar dicho error. Asimismo, se ha implementado un diseño responsive y con buena usabilidad para el usuario final.

3.2.9. Ejecución de aplicaciones en arranque

Es de interés que el script que procesa y almacena los datos y la aplicación web estén siempre en funcionamiento. Por ello, en vez de ejecutar a mano ambos programas cada vez que se enciende el gateway, he decidido crear dos daemons para realizar esta operación. Haré uso de `systemd` y `systemctl` [34] para ello.

Crearemos un fichero systemd para cada una de las acciones, estos ficheros están ubicados en el directorio daemons. Aunque como veremos, deberemos alojar una copia de estos ficheros en unos directorios determinados.

Por ejemplo, para ejecutar el script de procesamiento y almacenamiento de datos crearemos un script de bash muy básico que haga exactamente esto. El fichero lo alojaremos en /usr/local/bin.

```
Unset
run_app() {
  cd /home/pi/TFG_LoRaWAN_Application_Server/
  source venv/bin/activate $
  python3.10 server.py
}
$(run_app)
```

Luego tenemos que crear el fichero systemd que indicará al sistema operativo la acción a realizar, es decir, ejecutar el script anterior.

```
Unset
After=network.target

[Service]
ExecStart=/usr/local/bin/server.sh
Restart=always

[Install]
WantedBy=default.target
```

Este indica que el script se ejecutará una vez se haya levantado el servicio de red. Se reiniciará siempre que, por cualquier razón, ocurra algún error. Y que no es necesario ejecutarlo antes que ningún otro demonio. Este archivo debe estar alojado en /etc/systemd/system y tendremos que

asignarle los permisos necesarios para convertirlo en ejecutable.

```
Unset
sudo chmod 0644 /etc/systemd/system/app.service
```

Con todo esto, solo falta reiniciar el gestor de demonios para que tenga en cuenta esta nueva tarea y habilitarla.

```
Unset
systemctl daemon-reload
systemctl enable miscript.service
systemctl start miscript.service
```

Para comprobar que todo ha funcionado correctamente deberemos comprobar el estado del servicio.

```
Unset
systemctl status miscript.service
```

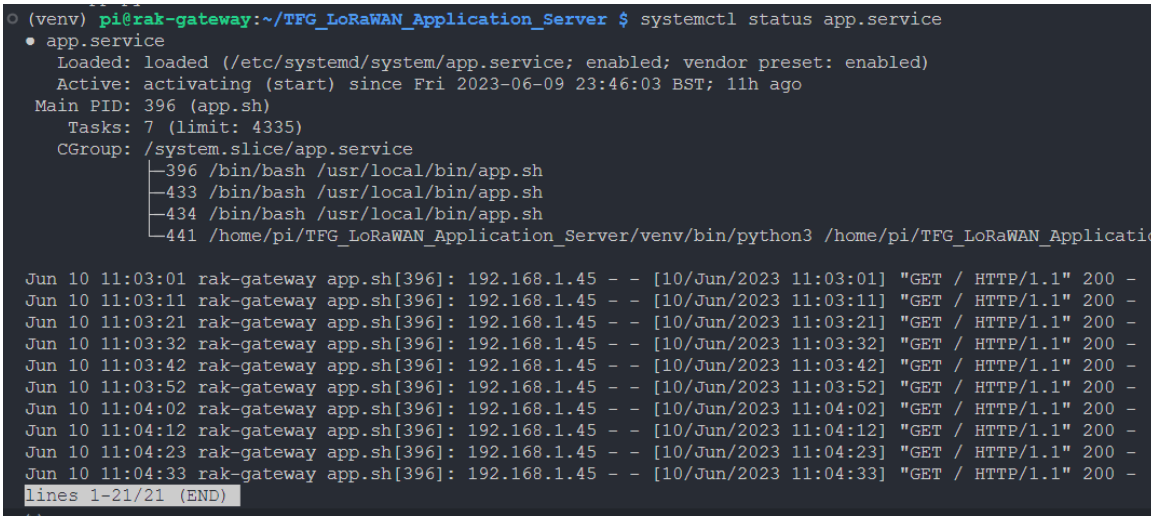


Figura 3.19: comprobación de correcto funcionamiento del servicio

3.2.10. Pruebas funcionales

Una vez que la red ha sido totalmente configurada con todos los apartados vistos en el trabajo. Con todos los dispositivos funcionando y la aplicación recibiendo los datos y mostrandolos en la web. Es hora de ver si la red funciona. Para comprobar efectivamente que todo esto

funciona correctamente y de forma íntegra. Procederemos a desconectar la antena LoRa del gateway principal. Hay que tener en cuenta que, por norma general, no es recomendable encender dispositivos transceptores sin su antena debido a que se puede dañar el equipo.

En este caso, lo hacemos porque una vez que los dispositivos han ingresado a la red y el gateway les ha indicado con un downlink sus parámetros de comunicación, el gateway no se pondrá en contacto de nuevo con ellos hasta que detecte un cambio. Y como el gateway no recibirá ningún paquete sin su antena, estamos a salvo de cualquier incidente.


```
{
  "applicationID": "2",
  "applicationName": "LoRaWAN-Network-AlvaroRG",
  "deviceName": "HyT-DRAGINO-2",
  "deviceProfileName": "DEVPROF-DRAGINO-EU868",
  "deviceProfileID": "195e9f30-bce8-408f-9764-c516fc871899",
  "devEUI": "a84041a8318279bb",
  "rxInfo": [
    {
      "gatewayID": "a840411eb3644150",
      "uplinkID": "b923383a-ee57-4aca-bed3-0687b2bc967c",
      "name": "DraginoLG308-Gateway",
      "time": "2023-07-11T18:49:37.286744Z",
      "rssi": -69,
      "loRaSNR": 9,
      "location": {
        "latitude": 28.47901826402084,
        "longitude": -16.312116980552677,
        "altitude": 6
      }
    }
  ],
  "txInfo": {
    "frequency": 868100000,
    "dr": 5
  },
  "adr": true,
  "fCnt": 3,
  "fPort": 2,
  "data": "y/kKewJwAQqYf/8=",
  "object": {
    "BatV": 3.065,
    "Hum_SHT": "62.4",
    "TempC_DS": "27.12",
    "TempC_SHT": "26.83"
  }
}
```


Figura 3.20: uplink recibido únicamente por el gateway secundario

Antes de quitar la antena, el formato de los uplinks era similar al de la figura 3.1, tras quitar la antena, recibimos el siguiente uplink en el MQTT Broker, el cual como vemos en la figura 3.20, indica que ha sido recibido por el gateway secundario, y no por el gateway principal que aloja el

servidor. Luego queda demostrado que el estado global del sistema no se vería afectado de ninguna manera. Simplemente todo el tráfico de la red pasa ahora por el gateway secundario.

Capítulo 4
Conclusiones y líneas futuras

4.1. Conclusiones

Con la presente memoria demostramos que hemos sido capaces de poner en funcionamiento una red privada LoRaWAN gestionada por un servidor de red ChirpStack, a la cual, hemos introducido varios sensores diferentes, para cubrir una zona de La Laguna.

Este proceso nos ha permitido aprender en profundidad cómo funcionan estas redes y la facilidad de uso ofrecida por programas de código abierto como ChirpStack. Refutando que si bien The Things Network podría haber sido utilizada en este caso, la red privada que hemos desplegado tiene un gran potencial de escalabilidad gratuita frente a él y un amplio repertorio de configuración y personalización añadida, con la más relevante que es la independencia.

Hemos visto que con pocas líneas de código, hemos desplegado tanto un servicio de procesamiento de los eventos del MQTT Broker del servidor de red como toda la aplicación web. Siendo el sistema muy flexible y cómodo de inicializar. Basta con configurar los dispositivos en el gestor de red y arrancar los servicios. Esto irá registrando automáticamente todos los dispositivos conectados y sus datos.

Además, hemos podido observar las ventajas que nos ofrecen las redes LoRaWAN para este tipo de proyectos y la gran flexibilidad que nos brinda este protocolo, tanto en temas de alcance y optimización de mensajes con LoRa, como gestión y cobertura de dispositivos. Y también la holgura que nos da el protocolo MQTT para integrar diferentes servicios con la red.

4.2. Líneas futuras

Una vez terminado el proyecto, una posible continuación del trabajo podría ir en la dirección de utilizar un framework de desarrollo web más capaz de dotar a la web de interacción y conexión con otros servicios como webhooks para integrar más datos.

Sería interesante el desarrollo de una API para que otras aplicaciones interactúen con la nuestra. O desarrollar un método de detección de futuras alertas, detectando patrones en los valores recabados.

También podría ser una vía de mejora, para optimizar los recursos en proyectos más grandes, llevar a cabo un script de procesamiento de

eventos que emplee programación concurrente para aprovechar al máximo el tiempo y hardware.

Además, ChirpStack nos permite crear diferentes aplicaciones, que es una forma de agrupar los diferentes dispositivos conectados a la red. En este trabajo solo se ha utilizado una, pero se podría diseñar una aplicación web que nos permita tener varias zonas de control, cada una de ellas con los dispositivos pertenecientes.

Capítulo 5
Summary and Conclusions

5.1. Summary

With this memory we demonstrate that we have been able to put into operation a private LoRaWAN network managed by a ChirpStack network server, to which we have introduced several different sensors, to cover an area of La Laguna.

This process has allowed us to learn in depth how these networks work and the ease of use that open source programs like ChirpStack offer. Rebutting that while The Things Network could have been used in this case, the private network we've deployed has a lot of free scalability potential in front of it and a vast repertoire of configuration and added customization, with the most relevant being independence.

We have seen that with a few lines of code, we have deployed both a service for processing the events of the MQTT Broker of the network server and the entire web application. The system is very flexible and easy to initialize. It is enough to configure the devices in the network manager and start the services. This will automatically record all connected devices and their data.

In addition, we have been able to observe the advantages that LoRaWAN networks offer us for this type of project and the great flexibility that this protocol offers us, both in terms of reach and optimization of messages with LoRa, as well as device management and coverage. And also the slack that the MQTT protocol gives us to integrate different services with the network.

5.1. Future lines

Once the project is finished, a possible continuation of the work could go in the direction of using a web development framework more capable of providing the web with interaction and connection with other services such as webhooks to integrate more data.

It would be interesting to develop an API for other applications to interact with ours. Or develop a detection method for future alerts, detecting patterns in the collected values.

It could also be a way of improvement, to optimize resources in larger projects, to carry out an event processing script that uses concurrent programming to make the most of time and hardware.

Furthermore, ChirpStack allows us to create different applications, which is a way of grouping the different devices connected to the network. In this work only one has been used, but a web application could be designed that allows us to have several control zones, each one with the belonging devices.

Capítulo 6
Presupuesto

En este capítulo resumimos todos los requisitos económicos del proyecto.

Por la parte física, esto es, los sensores y gateways utilizados, debido a que no ha hecho falta ningún servicio de pago externo.

Sensor	**Precio unidad**	**Unidades**	**Precio**
Dragino LH65 [35]	43,93 €	2	87,86 €
Milesight VS121 [36]	239,05 €	1	239,05 €
Milesight 307 [37]	369,99 €	1	369,99 €
RAK7244 [38]	210,00 €	1	210,00 €
Dragino LG308 [39]	296,59 €	1	296,59 €
		Total:	1203,49 €

Tabla 1: resumen presupuesto dispositivos

Vemos que se trata de una instalación relativamente barata si tenemos en cuenta la durabilidad de estos dispositivos y que estamos empleando una gestión privada de estos. En otros casos, para mantener tantos dispositivos y con las mismas prestaciones, deberemos estar pagando una alta mensualidad por la gestión.

Por otro lado, todo el trabajo realizado para llevar a cabo este trabajo se ve reflejado en la tabla 2. Contando con que en España el salario promedio de un ingeniero informático recién egresado ronda los 26.500€ al año o 13,59€ por hora, obtenemos los siguientes resultados:

Concepto	**Horas**	**Precio**
Estudio de requerimientos	20	271,8 €
Investigación sobre dispositivos y cubrimiento del área	20	271,8 €

Concepto	**Horas**	**Precio**
Instalación y configuración del servidor	40	543,6 €
Implementación de la aplicación Web	200	2718 €
Instalación y puesta en marcha de dispositivos	20	271,8 €
Documentación	40	543,6 €
Total:	340	4620,6 €

Tabla 2: resumen presupuesto de las tareas

Hay que tener en cuenta, que muchas de estas horas han sido destinadas a investigación para aprender los conceptos detrás del desarrollo. Y que tras haber diseñado este trabajo y tener unas guías de operación, posibles futuros proyectos tendrían una menor cantidad de horas.

Bibliografía

[1] *Wikipedia contributors. (2023). Internet of things. Wikipedia. Recuperado de https://en.wikipedia.org/wiki/Internet_of_things*

[2] *Conceptos básicos que te ayudarán a entender LoRa y LoRaWAN en minutos. (2023, 27 abril). Recuperado de https://becolve.com/blog/conceptos-tecnicos-basicos-que-te-ayudaran-a-entender-lora-y-lorawan-low-power-wide-area-network-en-pocos-minutos/*

[3] *The Things Network. (s. f.). The Things Network. Recuperado de https://www.thethingsnetwork.org/*

[4] *myDevices. (2023, 6 abril). myDevices | Simplify Sensor Deployments. Recuperado de https://mydevices.com/*

[5] *Thingsboard. (s. f.). ThingsBoard - Open-source IoT Platform. Recuperado de https://thingsboard.io/*

[6] *LORA: ¿Qué es y para qué sirve? - AlaiSecure - España. (2022, 19 abril). Recuperado de https://alaisecure.es/glosario/lora-que-es-y-para-que-sirve/*

[7] *LoRa Alliance. (2023, 18 abril). Homepage - LoRa Alliance®. Recuperado de https://lora-alliance.org/*

[8] *Understanding ADR | DEVELOPER PORTAL. (s. f.). Recuperado de https://lora-developers.semtech.com/documentation/tech-papers-and-guides/understanding-adr/*

[9] *Conceptos básicos que te ayudarán a entender LoRa y LoRaWAN en minutos | Clasificación de los dispositivos LoRaWAN (nodos) según su consumo. (2023, 27 abril). Recuperado de https://becolve.com/blog/conceptos-tecnicos-basicos-que-te-ayudaran-a-entender-lora-y-lorawan-low-power-wide-area-network-en-pocos-minutos/*

[10] *Spreading Factors. (2021, 26 noviembre). Recuperado de https://www.thethingsnetwork.org/docs/lorawan/spreading-factors*

[11] LoRaWAN Network - Technical Description - Telemetry2U. (n.d.). https://telemetry2u.com/Documentation/lorawan-iot-platform-technical-description

[12] *LoRa y el duty cycle. (2022, 9 marzo). Recuperado de https://lpwan.es/lora/lora-y-el-duty-cycle/*

[13] ABP vs OTAA. (s. f.). Recuperado de https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/

[14] ChirpStack open-source LoRaWAN Network Server. (s. f.). Recuperado de https://www.chirpstack.io/

[15] ¿Qué es el MQTT? - Explicación del protocolo MQTT - AWS. (s. f.). Recuperado de https://aws.amazon.com/es/what-is/mqtt/

[16] Locatelli, C. (2022, May 30). Monitoramento e Controle por Aplicativo - MQTT. Componentes Eletrônicos e Arduino. https://curtocircuito.com.br/blog/Categoria%20IoT/monitoramento-e-controle-por-aplicativo-mqtt

[17] Senet. (n.d.). Dragino LHT65 LoRaWAN Temperature & Humidity Sensor –Senet. Senet. https://senetco.com/marketplace/dragino-lht65-lorawan-temperature-humidity-sensor/

[18] Milesight IoT. (2023b, April 26). AI Workplace Occupancy Sensor VS121 | Milesight IoT. Milesight IoT - IoT Solution Provider. https://www.milesight-iot.com/lorawan/sensor/vs121/

[19] Milesight IoT. (2023, 14 febrero). Ambience Monitoring Sensor | AM300 Series | Milesight Lorawan Sensor. Milesight IoT - IoT Solution Provider. https://www.milesight-iot.com/lorawan/sensor/am300/

[20] Keep Your Project Connected to the IoT Cloud with the RAK7244 from RAKwireless - IoT Made Easy. (n.d.). https://www.rakwireless.com/en-us/products/lpwan-gateways-and-concentrators/rak7244

[21] LG308 Indoor LoRaWAN Gateway. (n.d.). https://www.dragino.com/products/lora-lorawan-gateway/item/140-lg308.html

[22] WisGate Developer Gateway Firmware Setup. (s. f.). Recuperado de https://docs.rakwireless.com/Knowledge-Hub/Learn/WisGate-Developer-Gateway-Firmware-Burning/

[23] Welcome to Flask — Flask Documentation (2.3.x). (s. f.). Recuperado de https://flask.palletsprojects.com/en/2.3.x/

[24] Airtime calculator for LoRaWAN-EU863-870. (s. f.). Recuperado de https://avbentem.github.io/airtime-calculator/ttn/eu868

[25] AlvaroRGZ. (s. f.). Github - alvarorgz/tfg_lorawan_application _server.GitHub. https://github.com/AlvaroRGZ/TFG_LoRaWAN_Application_Server/tree/main

[26] PostgreSQL. (2023, 7 julio). Recuperado de https://www.postgresql.org/

[27] venv — Creación de entornos virtuales. (s. f.). Recuperado de https://docs.python.org/es/3/library/venv.html

[28] Anthony_manotoa. (2021). Cómo crear tu propio bot de Telegram con Python. Platzi. Recuperado de https://platzi.com/blog/bot-python/

[29] Jinja — Jinja Documentation (3.1.x). (s. f.). Recuperado de https://jinja.palletsprojects.com/en/3.1.x/

[30] Documentation - Materialize. (s. f.). Recuperado de https://materializecss.com/

[31] Contributors, M. O. J. T. A. B. (s. f.). Bootstrap. Recuperado de https://getbootstrap.com/

[32] Maps, geocoding, and navigation APIs & SDKs | Mapbox. (s. f.). Recuperado de https://www.mapbox.com/

[33] Plotly. (s. f.). Recuperado de https://plotly.com/python/

[34] Ejecutar un script al arrancar Linux (debian). (s. f.). Recuperado de https://www.bonaval.com/kb/sistemas/debian/ejecutar-un-script-al-arrancar-linux-debian

[35] Dragino LH65. https://es.aliexpress.com/item/4000109818179.html

[36] Milesight VS121. Visitado el 03/07/2023. Recuperado de https://www.landatel.com/shop/product/mls-vs121-868m-w-milesight-vs121-868m-sensor-con-inteligencia-artificial-para-deteccion-y-conteo-de-personas-lorawan-868-mhz-14494?gclid=CjwKCAjwzJmlBhBBEiwAEJyLu2ap3HUTo9IyujMsk3FoEAkaiAr8LpliOKW3TjSTU3C3XlKJRqiBJBoCRsYQAvD_BwE#attr=

[37] Milesight 307. Visitado el 03/07/2023. Recuperado de https://aithings.store/indoor-monitoring/milesight-am307-868m-lorawan-premium-sensor-for-indoor-air-quality-ambience-monitoring

[38] RAK7244. Visitado el 06/07/2023. Recuperado de https://shop.marcomweb.it/en/shop-online/iot/lorawan/wisgate-developer-d4-eu868-dettagli.html

[39] Dragino LG308. Visitado el 03/07/2023. Recuperado de https://www.grooves-inc.es/dragino-gateway-lora-indoor-gateway-lg308n-868-mit-lte-dragino-accessories-pZZa1-2100804968.html?currency=EUR&language=es&_z=es&camp=es_smart&gclid=CjwKCAjwzJmlBhBBEiwAEJyLu07Ssx3izXBCLQlTDoQYVmToXOd7njQVi0cjXlZlTpBdNDaiJqjgRxoC2kYQAvD_BwE