Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño e Implementación de redes LoRa malladas

Design and Implementation of LoRa Mesh Networks

Autor

Ángel Torre Tolosana

Directores

Francisco José Martínez Domínguez

Julio Alberto Sangüesa Escorihuela

Escuela Universitaria Politécnica de Teruel
Universidad de Zaragoza
2022

Contenido

Índice de figuras

ANEXOS

ANEXO 1. CONFIGURACIÓN DE TECNOLOGÍAS

1. CONFIGURACIÓN DEL ENTORNO ARDUINO

Para el desarrollo de código que va a definir el comportamiento de los nodos, se utiliza el ecosistema de Arduino en su versión 1.8.16. El lenguaje Arduino está basado en C y cuenta con una gran variedad de librerías que facilitan las comunicaciones LoRa y el desarrollo de proyectos de comunicación. Además, hay una gran cantidad de información sobre multitud de aspectos, debido al gran soporte que recibe por parte de su comunidad.

Los módulos admiten el desarrollo de código sobre Arduino, pero el IDE necesita de una configuración previa para cada dispositivo antes de poder cargar los programas.

1.1. CONFIGURACIÓN NODO HELTEC WIFI LORA 32 EN ARDUINO

Antes de empezar a con la programación de los nodos finales, debemos realizar unas modificaciones sobre el entorno de Arduino con el que vamos a trabajar.

Se deberá instalar la librería básica Heltec_ESP32_Dev-Boards de comunicaciones para nuestra placa WiFi LoRa 32.

De acuerdo a las instrucciones de la página oficial de Heltec [30], es necesario añadir la URL de la librería en preferencias, dentro de del entorno Arduino y en el apartado de "Gestor de URLs Adicionales". La URL es la siguiente:

https://github.com/Heltec-Aaron-Lee/WiFi_Kit_series/releases/download/0.0.6/package_heltec_esp32_index.json

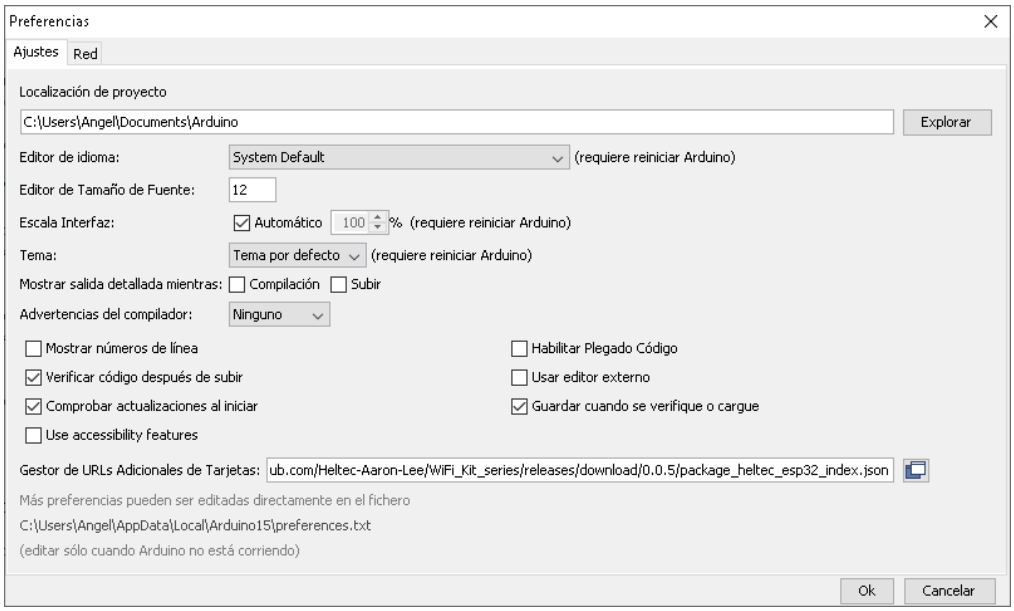


Figura 1. Configuración placa

Después, en el apartado Herramientas -> Placa -> Gestor de tarjetas, se buscar la librería Heltec ESP32 y se instala para poder trabajar con ella.

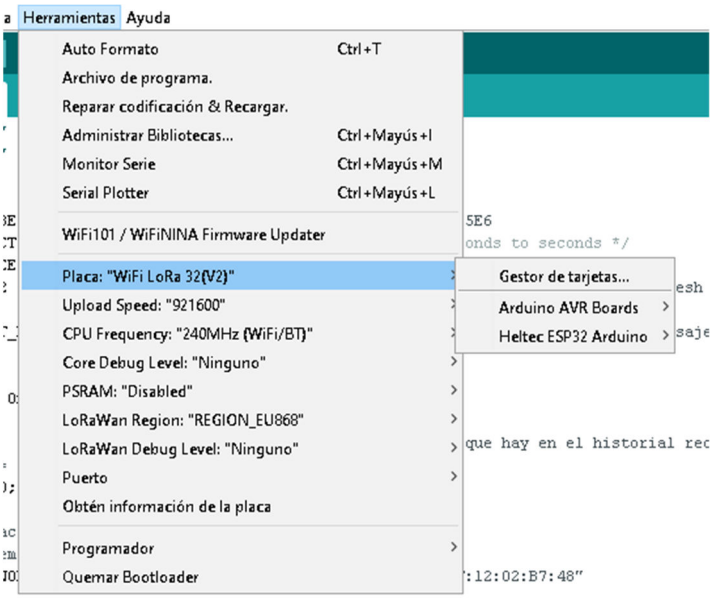


Figura 2. Configuración de la placa

Figura 3. Configuración de la placa

Ahora podemos utilizar las funciones de comunicación mediante LoRa que nos proporciona la librería de Heltec para las placas WiFi LoRa 32. Podemos acceder a los distintos ejemplos que nos facilita esta librería y cargar los programas básicos de envío y recepción de paquetes en las placas.

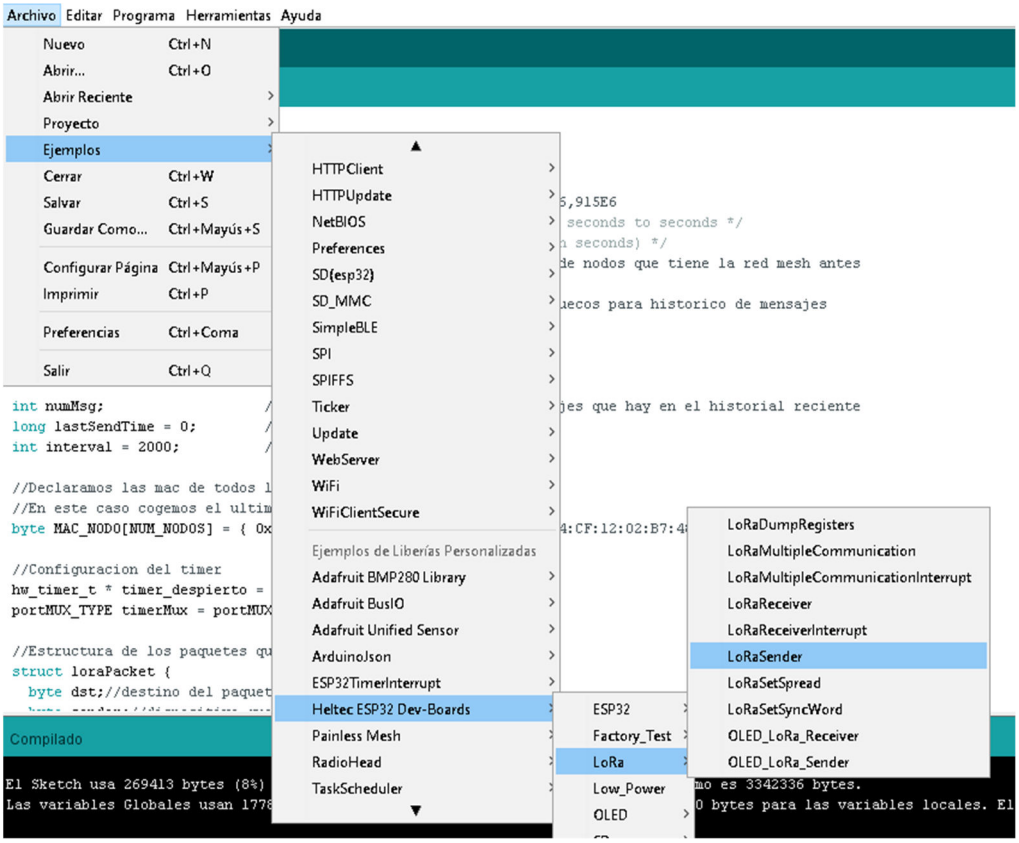


Figura 4. Ejemplos de comunicación LoRa

1.2. LIBRERÍAS

Para poder llevar a cabo la implementación, se deben programar las funciones que permitan las comunicaciones, tanto entre dispositivos LoRa como en la red LoRaWAN. Arduino presta un servicio de búsqueda dentro de su IDE, mediante el cual, permite la implementación de librerías que den distintas funcionalidades a los programas. En esta sección se van a describir las librerías más relevantes y sus funciones básicas utilizadas a lo largo de este proyecto.

Heltec_ESP32_Dev-Boards

Biblioteca básica para las placas Heltec ESP32 o ESP32+LoRa, para el uso de las comunicaciones entre dispositivos que utilicen la tecnología LoRa. Compatible con las placas: WiFi Kit 32, WiFi LoRa 32, Wireless Stick, Wireless Shell, facilita la implementación de comunicaciones entre nodos LoRa que trabajen bajo la misma frecuencia, mediante funciones de empaquetado, envío y recepción de la información [31].

- **Inicio de comunicaciones LoRa**

  Para el inicio de las comunicaciones LoRa entre dispositivos compatibles, primeramente se debe definir la frecuencia en que se comunican, y como ya hemos visto, la frecuencia europea es 868MHz. Se inicia la librería LoRa.h -> **LoRa.begin(frecuency)**

  Para saber si se ha iniciado de forma correcta y está listo para recibir o transmitir, esta función devuelve 1 en caso de éxito y 0 si ha fallado.

- **Finalización de comunicaciones LoRa -> LoRa.end()**

- **Envío de datos**
  Comienza la secuencia para envío de paquetes -> **LoRa.beginPacket()**
  Después de esto, se escribirán los datos en el paquete, con una capacidad máxima de hasta 255 bytes. Se dispone de una función **write()** que permite escribir de byte en byte sobre el paquete o directamente un buffer de bytes, donde es necesaria una especificación de tamaño de este buffer. Esta función devuelve el número de bytes que se han escrito en el paquete.

```
LoRa.write(byte);

LoRa.write(buffer, length);
```

Figura 5. Escritura de contenido en paquete LoRa

  Una vez se ha acabado de escribir, se indica el final del paquete -> **LoRa.endPacket()**
  Si es un éxito devuelve 1, o 0 en caso de error.

- **Recepción de datos**
  Los dispositivos deben ponerse a comprobar si hay otros dispositivos intentando comunicarse con ellos. La función **LoRa.parsePacket()** comprueba si se ha recibido un paquete y devuelve el tamaño en bytes, siendo este 0 si no se ha recibido ninguno. También da la opción de recibir paquetes con un tamaño de bytes esperado, definiéndolo como parámetro en esta función.

  La función **onReceive()** de la Figura 6 se utiliza junto a la anterior **LoRa.parsePacket()** para definir el comportamiento del dispositivo cuando reciba un paquete y darle lógica a los dispositivos.

```
LoRa.onReceive(onReceive);

void onReceive(int packetSize) {
 // ...
}
```

Figura 6. Define comportamiento de nodos al recibir paquete

  Finalmente, si se quiere poner a los dispositivos en un modo de recepción de datos constante, se usa la función de **LoRa.receive().** Si hemos definido la función **onReceive()** esta será llamada cuando reciba un paquete.
- **Lectura de paquetes recibidos**
  Una vez se ha recibido un paquete en un dispositivo, para la lectura de su contenido se disponen de varias funciones. La función **LoRa.read()** va a leer el siguiente byte del paquete y devuelve su valor, mientras que la función **LoRa.available()** devuelve el número de bytes disponibles para leer. De esta forma puede leerse un paquete entero ejecutando **LoRa.read()** siempre que **LoRa.available()** devuelva bytes disponibles para leer.

Otros datos implícitos al recibir un paquete que nos ayudan a obtener información sobre el remitente son las mediciones de RSSI y SNR.

El RSSI indica la potencia de la señal recibida en milivatios y medida en dB. Es un valor negativo y son considerados una señal fuerte con -30dBm y débil con -120dBm, siendo este el mínimo. Este valor puede obtenerse mediante la función **LoRa.packetRssi()**

El valor SNR es la relación señal/ruido, es decir, la relación entre la señal de potencia recibida y el nivel de potencia del piso de ruido (Área de todas las fuentes de señales interferentes). Este valor podemos obtenerlo mediante la función **LoRa.packetSnr().**

Para evaluar la comunicación entre dos dispositivos podemos hacer una relación de valores entre el RSSI y el SNR como puede observarse en la Figura 7.

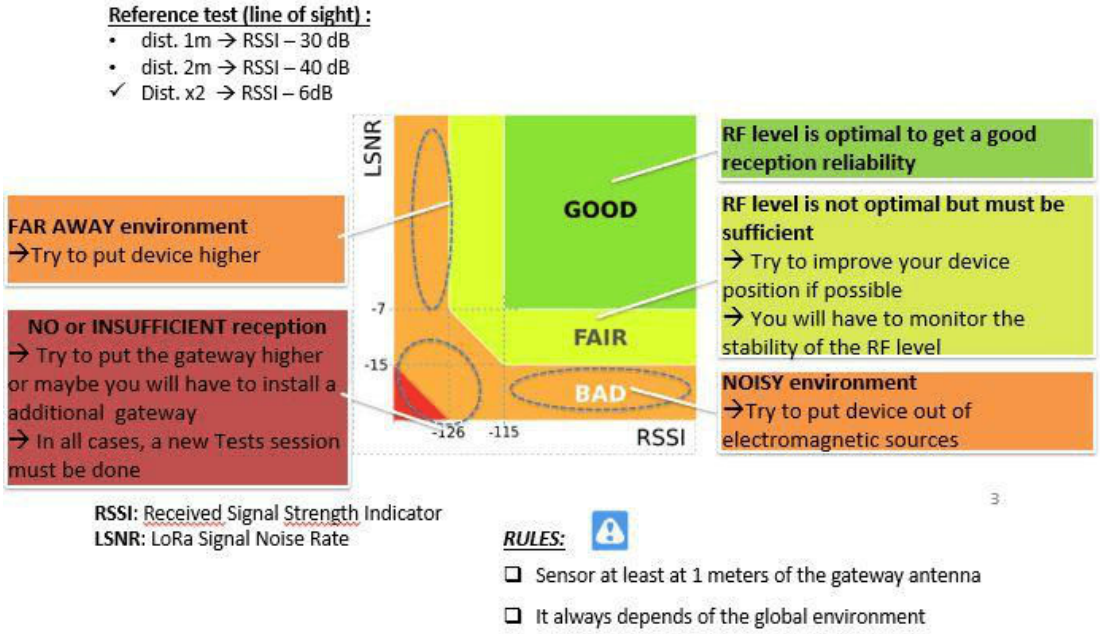


Figura 7. Comparación señal RSSI y SNR [22]

- **Parámetros de radio**

  Se pueden modificar algunos aspectos de radio, como la potencia TX o potencia óptica de transmisión de salida. La potencia por defecto es 17 dB. Mediante la siguiente función, podemos indicarle la potencia de transmisión al dispositivo. **LoRa.setTxPower(txPower)**

  La frecuencia es otro aspecto que se puede cambiar. Como ya se ha mostrado, LoRa trabaja con las frecuencias 433E6, 868E6 y 915E6 -> **LoRa.setFrecuency()**

  También se puede modificar el factor de dispersión o SF (Spreading Factor), que en términos de LoRa, es la cantidad de código de ensanchamiento aplicado a la señal de datos original. Dispone de 6 niveles (SF7 hasta SF12), siendo el mayor número el que se traduce en un mayor alcance de la señal.

LoRa.setSpreadingFactor(spreadingFactor)

Esta librería también da la posibilidad de establecer una palabra de sincronización de la radio, que va a permitir filtrar las comunicaciones de aquellos nodos que no estén utilizando la misma palabra de sincronización. El rango de la palabra va de 0 a 0xFF. -> **LoRa.setSyncWord(syncWord).**

Esp Deep Sleep

La librería Heltec_ESP32_Dev-Boards también provee de otras funciones básicas para dispositivos de esta clase. Dispone de funciones para ahorrar recursos, poniendo a dormir o en suspensión a los módulos. En este caso se ha decidido usar la función **esp_deep_sleep** que hará que el nodo se suspenda y no se realicen operaciones de CPU o WiFi, teniendo una frecuencia de reloj reducida y un menor voltaje durante un periodo establecido. Mediante la función **esp_sleep_enable_timer_wakeup(time_in_us)** estableceremos el tiempo que estará suspendido el nodo y mediante **esp_deep_sleep_start()**, haremos que se ejecute la hibernación del dispositivo.

Beelan-LoRaWAN

Esta librería permite implementar el protocolo LoRaWAN en un chip SX1276/SX1278 y cumple con las propuestas del estándar, proporcionando funciones para la conexión con el servidor, registro y autenticación del dispositivo, el envío de mensajes o la gestión de la ventana de recepción [32].

- **Inicialización de la librería LoRaWAN** -> **lora.init()**

- **Configuración de claves de autenticación:**

  Establece los métodos para definir las claves de autenticación, que van a ser requeridas para realizar la conexión con el servidor LoRaWAN.

```
void setDevAddr(unsigned char *devAddr_in);

void setAppSKey(unsigned char *ApskKey_in);
void setNwkSKey(unsigned char *NwkKey_in);

void setDevEUI(const char *devEUI_in);
void setAppEUI(const char *appEUI_in);
void setAppKey(const char *appKey_in);
```

Figura 8. Definición de claves de autenticación

- **Tipo de dispositivo final**
  Definimos el tipo de dispositivo final, en función de cómo va ser su capacidad para recibir los mensajes DownLink (Clase A, B o C) -> **lora.setDeviceClass(CLASS_A)**

- **Conexión con el servidor LoRaWAN**
  Una vez se haya dado de alta el dispositivo final y se hayan definido las claves proporcionadas por el Servidor para su autenticación, el dispositivo podrá unirse y comenzar las comunicaciones -> **lora.join()**

Tras haberse unido el dispositivo final a la red LoRaWAN, este podrá comunicarse con el servidor mediante mensajes Uplink y recibir información mediante Downlinks.

- **Uplink**
  Para enviar información desde el dispositivo final al servidor, se realiza mediante Uplink que contienen la información cifrada del PHYPayload.

```
void sendUplink(unsigned char *data, unsigned int len, unsigned char confirm);
```

Figura 9. Función para el envío de Uplinks

- **Downlink**
  Para recibir la información desde el servidor hasta el nodo, se dispone de la siguiente función que recoge los Downlink y los guarda en un buffer suministrado en el parámetro de la función.

```
void readData(void);
```

Figura 10. Función para lectura de Downlinks

Adafruit BMP280 Library

Librería para los sensores BMP280 que nos facilita la obtención de datos captados por el sensor BMP280. Es compatible con todas las arquitecturas [33].

- **Inicialización de librería**
  Declaración de variable tipo Adafruit **-> Adafruit_BMP280 bmp**
  Inicia la librería para utilizar sus funciones -> **bmp.begin()**

- **Lectura del sensor**
  Temperatura -> **bmp.readTemperature()**
  Presión -> **bmp.readPressure()**
  Altitud: -> **bmp.readAltitude()**

EEPROM

La librería EEPROM nos permite leer y escribir desde la memoria flash del ESP32. Esta biblioteca nos permite usar hasta 512 bytes en la memoria flash, es decir, 512 direcciones diferentes que pueden tener un valor entre 0 y 255. Esto es útil para guardar aquellos valores que no se quieren perder, en caso de que el dispositivo se apague accidental o intencionadamente. Estos valores pueden ser estados del sistema, ajustes, o cualquier tipo de dato que no quiera ser borrado. El único problema que presenta, es que el número de escrituras en la memoria flash está limitado a 10^5 operaciones, después de esto no garantiza su funcionamiento. Sin embargo, las operaciones de lectura no tienen límite.

- **Inicialización de la EEPROM**

  Se llama a la función **EEPROM.begin()** y se establece el tamaño que se va a utilizar en bytes.

- **Escritura en memoria**

  Utilizaremos la función **EEPROM.write()** para escribir un valor de tamaño de un byte en la posición de memoria indicada (address, value). Después de eso, ejecutaremos **EEPROM.commit()** para guardar los cambios.

```
EEPROM.write(address, value);

EEPROM.commit();
```

Figura 11. Escritura en memoria EEPROM

- **Lectura de memoria**

  Con la función **EEPROM.read(address)** se obtiene el valor de tamaño byte que está almacenado en la posición de la memoria indicada.

2. CONFIGURACIÓN NODO FINAL Y SENSOR BMP280

En la red mallada, como ya se ha explicado anteriormente, va a haber un tipo de nodo sensor, el cual va a tener conectado el sensor BMP280 con el que va a obtener la temperatura, dato que será empaquetado para su enrutamiento hacia el nodo destino.

Para su utilización requiere de la instalación de la librería Adafruit BMP280 Library, que permitirá acceder a las funciones utilizadas para la obtención de datos de temperatura y presión barométrica.

Para realizar las conexiones físicas entre el sensor BMP280 y el módulo LoRa, nos ayudaremos con el diagrama de pines para saber hacer la conexión de forma correcta (Ver Figura 12).

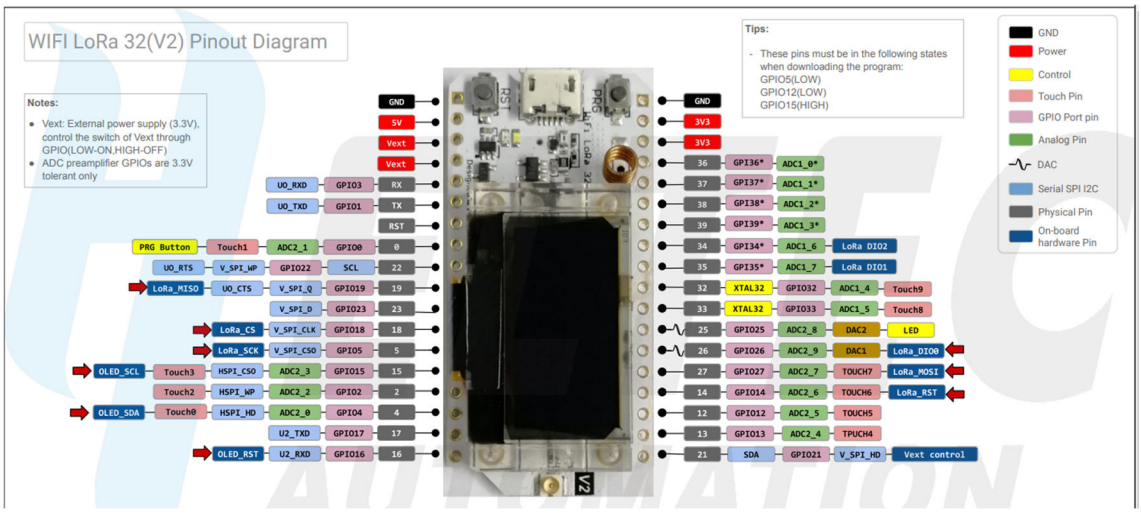


Figura 12. Diagrama de pines de la Placa WiFi LoRa 32(v2) [34]

La tabla para la conexión de pines queda de la siguiente manera:

BMP280	LoRa 32
VCC	3V3
GND	GND
SCL	GPIO 22
SDA	GPIO 21
CSB	
SDO	3V3

Figura 13. Configuración pines sensor

La implementación para este proyecto se ha realizado mediante una protoboard y siguiendo el diagrama de pines explicado anteriormente. La imagen final de la placa sensora está representada en la Figura 14.

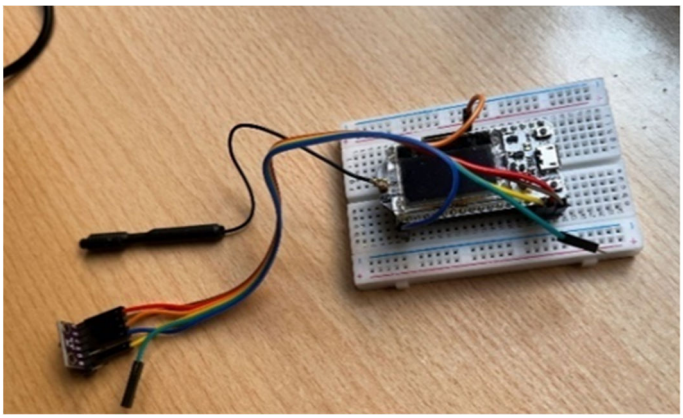


Figura 14. Placa sensora

3. CONFIGURACIÓN DE CHIRPSTACK

Para la utilización del servidor en despliegue real, se han de realizar una serie de pasos previos donde se debe registrar y activar los dispositivos que componen la red y habilitar las comunicaciones entre ellos. En este proyecto se ha utilizado un servidor ChirpStack proporcionado por el grupo de investigación iNiT de la Universidad de Zaragoza [35] y una puerta de enlace ya registrada en este servidor, por lo que la configuración que se documenta no parte desde cero.

3.1. CREACIÓN DEL PERFIL DE SERVICIO

Primeramente, se crea un perfil de servicio, donde se definen las funciones para el conjunto de aplicaciones o usuarios que estén asociados a este perfil, además de definir la tasa de mensajes máxima y mínima, a la que pueden transmitir a través de la red. Se definen los siguientes campos:

- AddGWMetadata: Permite agregar los datos RSSI, SNR y geolocalización del Gateway.
- NwkGeoLoc: Permite la geolocalización por red de los dispositivos.
- DevStatusRewFreq: Número de solicitudes de estado del dispositivo final que se permiten por día.
- ReportDevStatusBattery: Reporta el estado de batería del dispositivo final al servidor de aplicación.
- ReportDevStatusmargin: Reporta el estado de margen del dispositivo final al servidor de aplicación.
- DRMin: Establece una velocidad mínima de datos permitida.
- DRMax: Establece una velocidad máxima de datos permitida.

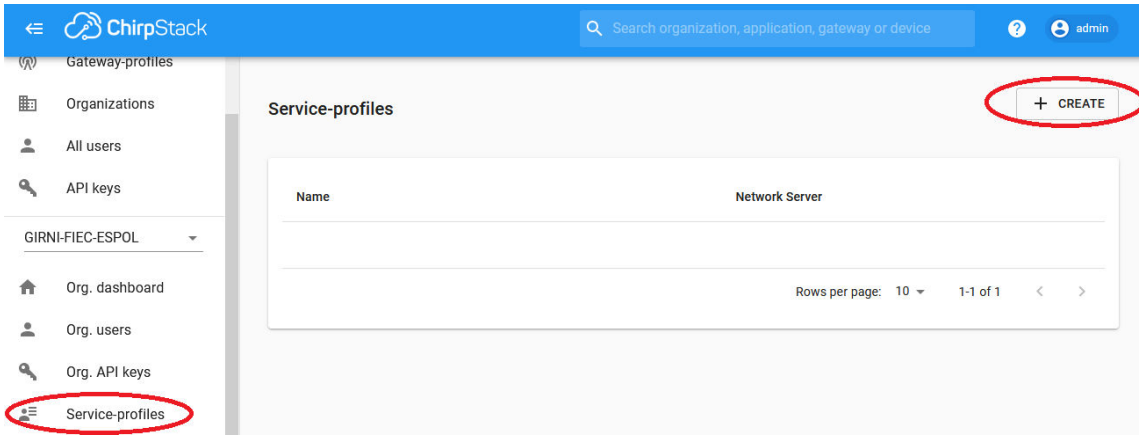


Figura 15. Creación de perfil de servicio

3.2. CREACIÓN DE UN APLICACIÓN

Los dispositivos se agrupan por aplicaciones, con la finalidad de organizarlos según la función que desempeñen. Por ello, se crea una nueva aplicación, a la que añadiremos todos nuestros nodos finales. Se debe dar un nombre legible a la aplicación y crear una nueva instancia del dispositivo que se da de alta en ChirpStack, asociarlo con su perfil de dispositivo y junto con el perfil de servicio creado anteriormente. En la Figura 16 se muestran todas las aplicaciones y el perfil de servicio al que están asignadas.

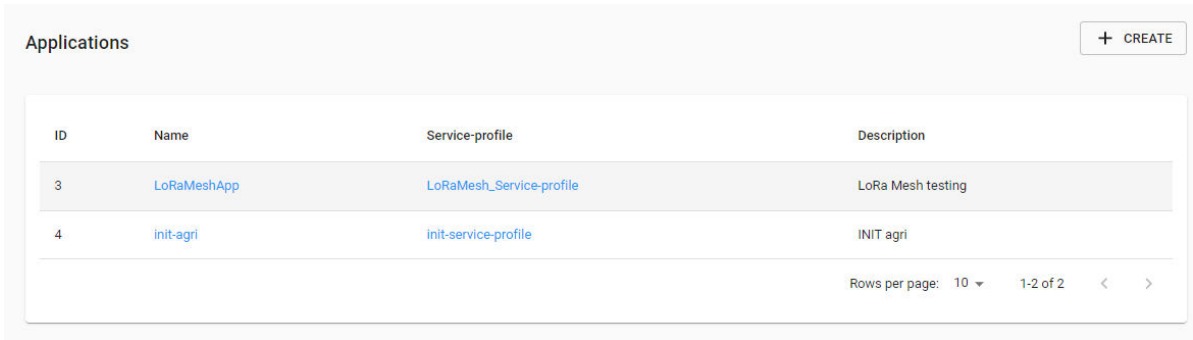


Figura 16. Visualización de aplicaciones

3.3. REGISTRO DE DISPOSITIVOS FINALES

Para incluir los dispositivos que van a componer la red LoRaWAN, se deben dar de alta y activar en ChirpStack. Como requisito, se debe saber la versión LoRaWAN MAC que implementa el dispositivo que se quiere agregar al servidor y revisar sus parámetros regionales.

CREACIÓN DE PERFIL DEL DISPOSITIVO

Al igual que al registrar el Gateway, aquí también se debe crear un perfil para el nodo final antes de agregarlo a ChirpStack. En el perfil se configuran las capacidades del dispositivo:

- LoRaWAN MAC versión: El formato del paquete que se va a recibir y que viene definido en la documentación oficial del dispositivo. En este caso para la placa WiFi LoRa 32 ejecuta el protocolo LoRaWAN 1.0.2.
- LoRaWAN Regional Parameters: Utiliza la versión "A"
- ADR algorithm: El algoritmo ADR (Adaptative Data Rate) va a permitir adaptar la velocidad y potencia de las transmisiones.
- Uplink Interval: El intervalo esperado en segundos en el que el dispositivo envía mensajes de enlace ascendente. Utilizado para determinar si un dispositivo está activo o inactivo.
- OTAA/ABP: Son dos modos de activación de dispositivos finales para LoRaWAN. Se recomienda el uso de OTAA (Over The Air Activated), ya que cuando un dispositivo final se une a la red LoRaWAN, se le asigna una DevAddr dinámica y se utilizan las claves raíz para crear claves de sesión. Este funcionamiento permite a los dispositivos moverse a diferentes redes/clúster. Sin embargo, ABP utiliza un único DevAddr por lo que solo puede funcionar correctamente en su red predefinida, e incluso aunque una red le permitiese registrarse con valores DevAddr, diferentes, no se enrutaría el tráfico de estos dispositivos a la red/clúster.
- Códec: Permite la codificación y decodificación de los mensajes recibidos (Uplink) o los enviados (Downlink). Da opciones, como la implementación propia de funciones en JavaScript o CayenneLPP, que ya trae definidas sus funciones de codificación y decodificación de paquetes Cayenne. En este proyecto se ha desarrollado una serie de funciones en JavaScript que permiten la decodificación del payload de los Uplinks realizados por los dispositivos finales.

ALTA DEL DISPOSITIVO

Desde la aplicación creada, se le da al botón de "Crear" para añadir el dispositivo final. Aquí se rellenan los datos relativos al nombre y descripción del dispositivo, se le asigna el perfil de dispositivo al que va a estar asociado y finalmente la clave DevEUI. El DevEUI es un identificador único para cada dispositivo que viene asignado por el fabricante, el cual debe facilitarse cuando se adquiere el producto. En caso de no poseer este DevEUI, ChirpStack puede generar uno en el momento de creación del dispositivo como se ve en la Figura 17.

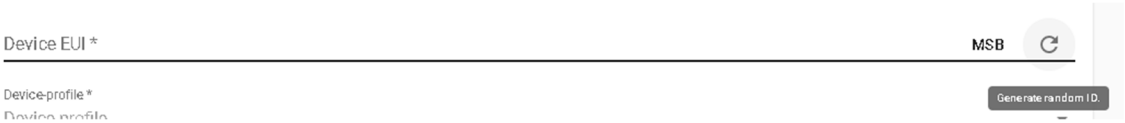
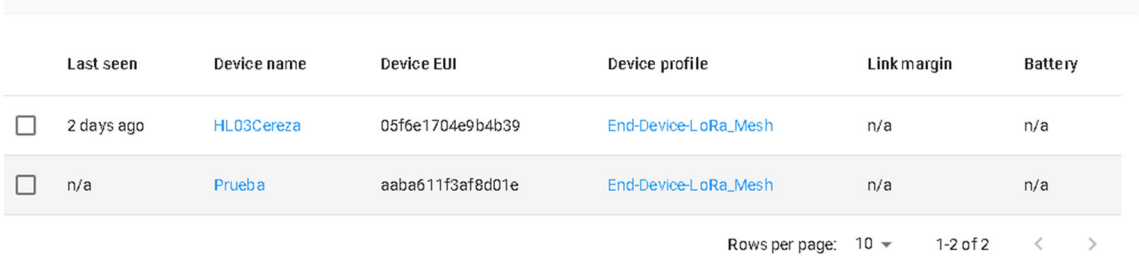


Figura 17. Generación aleatoria de DevEUI

Una vez se ha creado el dispositivo, si pinchamos sobre este, se pueden visualizar las claves OTAA y de Activación que vamos a necesitar incluir en el código que se carga sobre el dispositivo:

- Application key (AppKey): Puede gestionarse en la pestaña CLAVES(OTAA). Los dispositivos activados dinámicamente utilizan esta clave para derivar las dos claves de sesión (NwSKey y AppSKey) durante el proceso de activación.
- Device address (DevAddr): Clave de activación dinámica en OTAA, cuando el dispositivo se une a la red.
- Network sesión key (NwSKey): Es una clave de sesión utilizada en la interacción entre el dispositivo final y el servidor de red. Valida la integridad de los mensajes mediante MIC (Message Integrity Code), que tiene un funcionamiento similar a un checksum.
- Application sesion key (AppSKey): Clave de sesión que cifra y descifra la carga útil de los paquetes. Dicha carga está encriptada entre el dispositivo final y el Servidor, asegurando la confidencialidad del usuario.



	Last seen	Device name	Device EUI	Device profile	Link margin	Battery
☐	2 days ago	HL03Cereza	05f6e1704e9b4b39	End-Device-LoRa_Mesh	n/a	n/a
☐	n/a	Prueba	aaba611f3af8d01e	End-Device-LoRa_Mesh	n/a	n/a

Rows per page: 10 1-2 of 2

Figura 18. Registro de dispositivos finales de la aplicación

Las claves de sesión son únicas de cada dispositivo por sesión, y si se ha activado el dispositivo mediante OTAA, estas claves se vuelven a generar en cada activación.

Como paso final, se deben definir en el código que se carga sobre el dispositivo final, las claves DevEUI y AppKey mediante las funciones de la librería LoRaWAN, para que el dispositivo pueda autenticarse en el servidor de red (Ver Figura 19).

```
const char *devEui = "05f6e1704e9b4b39";
const char *appKey= "1b68a9693d0dadf55a4abe1f9ecb595e";
```

Figura 19. Definición de claves de autenticación en dispositivo final

VISUALIZACIÓN DE PAQUETES

Para comprobar que el dispositivo se ha dado de alta en el servidor de forma correcta, se carga un programa que contiene las claves de autenticación vistas en la Figura 19, una función para establecer conexión con el servidor y otra para el envío de mensajes Uplink. Estas tres funciones van a componer un programa básico para la comunicación del dispositivo final con el servidor ChirpStack.

ChirpStack nos ofrece una gran cantidad de herramientas para la monitorización de los datos obtenidos por los elementos de la red LoRaWAN. En la aplicación creada en el punto anterior, seleccionamos el dispositivo final del que se espera recibir datos, y en la sección Device Data (Ver Figura 20) podremos ver las interacciones que está teniendo el dispositivo con el servidor si se ha configurado todo correctamente. En la sección de detalles, también podemos comprobar el estado del dispositivo (Ver Figura 21)., visualizar gráficas de datos recibidos, comprobar posibles errores, métricas como RSSI y SNR, etc **(**Ver Figura 22).

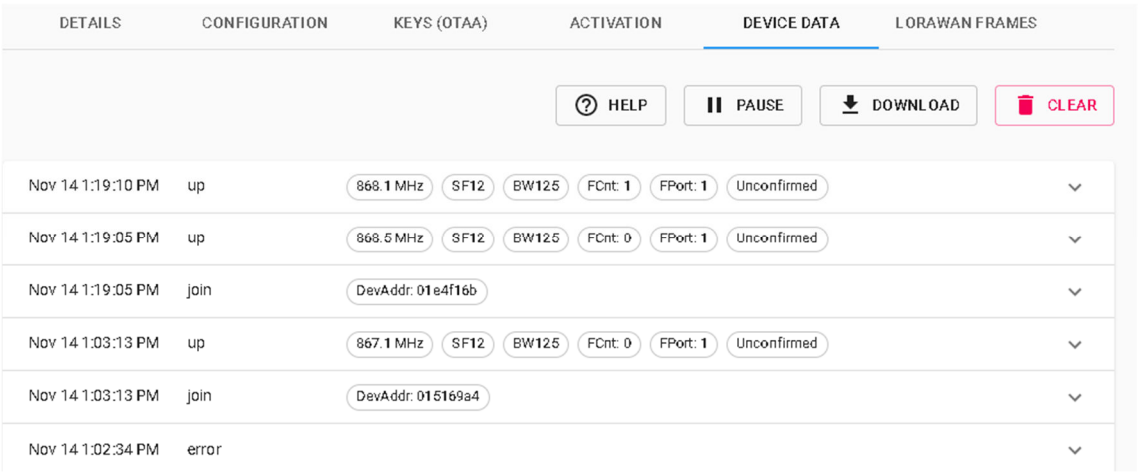


Figura 20. Registro de comunicaciones recibidas del dispositivo

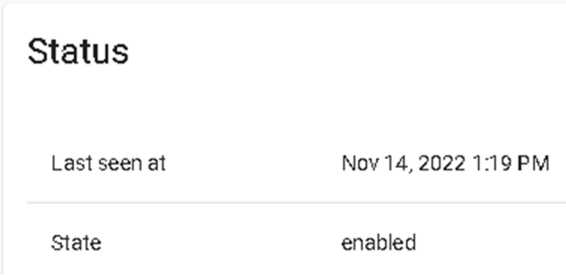


Figura 21. Estado del dispositivo

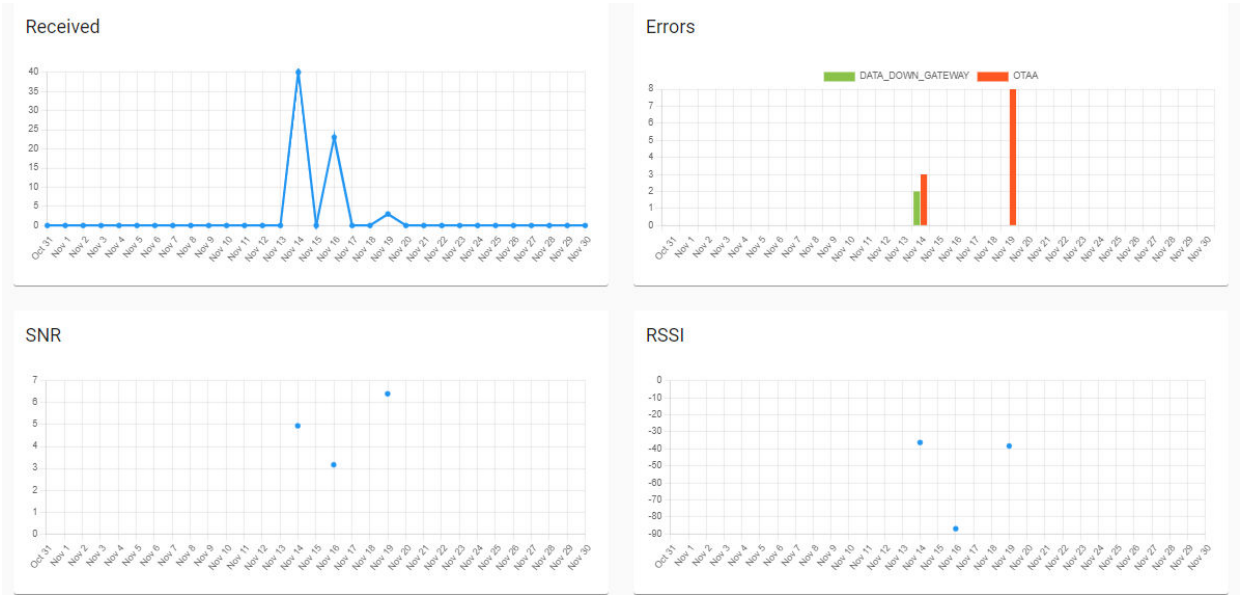


Figura 22. Gráficas de datos recibidos

REFERENCIAS BIBLIOGRÁFICAS

[30] Arduino configuration for Heltec´s nodes, Heltec, September 2022. https://docs.heltec.org/en/node/esp32/quick_start.html

[31] Heltec libraries for Esp32 and LoRa devices, Arduino, March 2014. https://www.arduinolibraries.info/libraries/heltec-esp32-dev-boards

[32] LoRaWAN library, ElectronicCats, 2018. https://github.com/ElectronicCats/Beelan-LoRaWAN

[33] Bmp280 sensor libraries, Adafruit, 2018. https://github.com/adafruit/Adafruit_BMP280_Library

[34] Pin map for Heltec wifi lora 32, The Hiveeyes community, February 2017. https://community.hiveeyes.org/t/heltec-wifi-lora-32/3125

[35] iNiT overview, Intelligent Networks and Information Technologies, 2022. http://init.unizar.es/