Escuela Superior de Ingeniería y Tecnología
Universidad de La Laguna

MEMORIA TRABAJO DE FIN DE GRADO

Desarrollo de aplicación de Internet de las Cosas (IoT) usando protocolos LoRa y LoRaWAN.

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA (ESIT)

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA (GIEIA)

Estudiante: Richard Arencibia Cubas

Tutor : D. Alberto Hamilton Castro

Fecha: 22/05/2024

Índice de Contenidos:

Índice de Ilustraciones y Tablas:

1 Resumen

Debido al avance de las nuevas tecnologías es cada vez más común la utilización de dispositivos capaces de conectarse a la red y establecer comunicación entre sí, con lo que se conoce como interacción machine-to-machine (M2M). Estas tecnologías se denominan Internet of Things(IoT). Las aplicaciones del IoT son cada vez más numerosas: industria, agricultura, medicina, defensa, etc.

Para la puesta en marcha de estos dispositivos se emplean tecnologías inalámbricas como las LPWAN(Low-Power-Wide-Area-Network), las cuales permiten enviar pequeños paquetes de datos a largas distancias con bajo consumo de energía. Un tipo de LPWAN es LoRa, la tecnología que se utilizará en este trabajo, cuyos dispositivos se comunican a través del protocolo LoRaWAN.

El objetivo de este Trabajo de Fin de Grado es la utilización de la placa CubeCell Dev-Board Plus (HTCC-AB02) para recoger las medidas de diferentes sensores (DHT11, Tilt Switch, KY-033, KY-036) y visualizarlos en una aplicación web. Se construirá una base de datos SQLite con los mensajes que llegan al servidor de red (The Things Stack) utilizando el protocolo de comunicación MQTT para obtener un registro de las magnitudes.

Palabras clave: IoT, LoRa, LoRaWAN, TTN, MQTT, Rest API, sqlite3, Dash

1.1 Abstract

With the development of new technologies nowadays it is more common to find devices that establish a connection with the network and communicate with each other, this is known as machine-to-machine interaction (M2M) and these technologies we call Internet of Things (IoT).The usage of these technologies is everywhere: industrial sector, agriculture, medecine, defense industry, etc.

For the set up of these devices we use wireless technologies such as LPWAN (Low-Power-Wide-Area-Network), which allows us to send data packages throughout long distances and with minimal power consumption. For this project we will use LoRa, a LPWAN type, whose devices use the LoRaWAN communication protocol.

The goal of this thesis is to utilize the CubeCell Dev-Board Plus (HTCC-AB02) to gather readings from different sensors (DHT11, Tilt Switch, KY-033, KY-036) in order to visualize them in a web applicaction. A SQLite database will be built with the messages that are sent to the network server (The Things Stack) using MQTT communication protocol so that it is possible to register the sensor readings.

Keywords: IoT, LoRa, LoRaWAN, TTN, MQTT, Rest API, sqlite3, Dash

2 Introducción

2.1 Objetivos

El objetivo de este Trabajo de Fin de Grado es la realización de una aplicación de IoT con dispositivos provistos de comunicación LoRa y LoRaWAN (CubeCell Dev-Board Plus (HTCC-AB02)). El objetivo principal se puede desglosar en las siguientes tareas a realizar:

- Familiarización y puesta en marcha del dispositivo y gateway (GW).

- Diseño de la aplicación a realizar en base a los sensores disponibles .

- Implementación de la aplicación programando el dispositivo final así como el servidor de aplicaciones en redes abiertas (The Things Network).

-Desarrollo de una aplicación web usando los datos de los sensores para poder visualizar y manipular el histórico de datos de los sensores.

2.2 Conocimientos previos

2.2.1 Aplicación de IoT y tecnologías usadas

2.2.1.1 Campos de aplicación de IoT

Algunos de los campos con gran aplicabilidad de IoT son:

1) **Consumidores:** una gran porción de los dispositivos IoT se crean para el uso del consumidor, entre los que se incluyen: vehículos conectados, domótica, salud conectada, dispositivos con monitoreo remoto, etc.

2) **Domótica:** incluye iluminación, calefacción y aire acondicionado, sistemas de cámaras y de seguridad. Ejemplos: Apple HomeKit, Samsung SmartThings, etc.

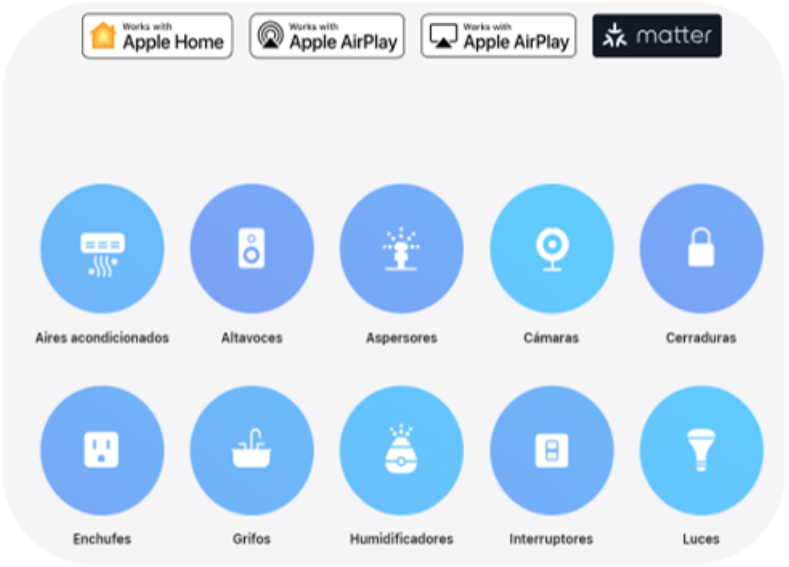


Ilustración 1 Algunas aplicaciones de Apple HomeKit.

3) **Cuidado de ancianos:** los hogares inteligentes también permiten acomodar las discapacidades específicas de sus dueños, como el control por voz que permite ayudar a usuarios con problemas de visión y movilidad; así como incluir otras medidas de seguridad que alerten a los servicios de emergencia en caso de caídas o convulsiones.

4) **Medicina y Salud:** Sanidad Inteligente (Smart Healthcare) o The Internet of Medical Things (IoMT) hace referencia a la porción de IoT cuya aplicabilidad recae en el campo de las ciencias médicas, se emplea en la recolección de datos que se usarán para investigación y monitorización. Ejemplos: monitorización de presión arterial, ritmo cardiaco, implantes como marcapasos, etc.

5) **Transporte:** IoT puede ayudar en la integración de comunicaciones, control y el procesamiento de la información en varios sistemas de transporte.

6) **Agricultura:** se usa principalmente en la recolección de datos como temperatura, lluvia, humedad, velocidad del viento, infestación de plagas y el contenido de suelos. Los datos obtenidos por los sensores conjuntamente con la intuición y conocimientos de los agricultores se usan para mejorar la productividad y disminuir costos.

7) **Gestión de la energía:** un gran número de dispositivos que requieren de alimentación vienen integrados con conexión a Internet y pueden ser controlados de forma remota por los usuarios o por una interfaz basada en servicios de la nube, lo cual permite planificar eventos y ayudar a la optimización tanto de la generación como del consumo de energía.

8) **Medio ambiente:** aplicaciones de IoT usan una serie de sensores que ayudan a la preservación de ecosistemas; monitorizan datos relativos a la calidad del agua y aire, condiciones atmosféricas y del suelo e incluso hacen un seguimiento de animales salvajes en sus hábitats naturales.

9) **Industrial:** también se conoce como Industrial Internet of Things (IIoT), se centra en el análisis de datos de maquinaria conectada, tecnologías operacionales (OT), lugares y personas y sirve para la regulación y monitoreo de sistemas industriales.

2.2.1.2 Tecnologías inalámbricas utilizadas para IoT

Los dispositivos inteligentes se comunican entre sí a través de tecnologías inalámbricas, estas se diferencian en cuanto al tamaño de los paquetes datos que son capaces de enviar, la distancia hasta la que es posible establecer comunicación entre dispositivos y la energía que consumen en el proceso de transmisión, las principales son:

1) **Comunicaciones celulares (4G, 5G):** las redes de comunicación de 4G están bastante extendidas y ofrecen comunicación de banda ancha que permiten llamadas de voz y transmisión de vídeos. Aunque las necesidades energéticas son relativamente altas, estas redes ofrecen buen ancho de banda y escalabilidad en cuanto a la interoperabilidad. El uso de 5G todavía no está tan extendido, pero ofrecen muy alta movilidad y ultra baja latencia de hasta 10 veces menos que 4G.

2) **Bluetooth y Bluetooth Low Energy (BLE):** se diseñó para la comunicación entre dispositivos punto a punto o punto a multipunto. BLE tiene menos requerimientos energéticos por lo que es ideal para aplicaciones destinadas al consumidor (es muy utilizado en fitness y dispositivos médicos usables), su rango extendido hace esta tecnología idónea para sensores IoT que quieren evitar el cambio de baterías o minimizarlo.

3) **Wi-Fi:** debido a su cobertura, escalabilidad y consumo de energía no es la tecnología más adecuada para sensores IoT que usan baterías, sino que se usa más en electrodomésticos inteligentes que se conectan a un enchufe.

4) **RFID (Radio Frequency Identification):** es muy utilizada en retail y sectores logísticos. Consiste en enviar pequeños paquetes de datos, utilizando ondas de radio, desde una tarjeta RFID hasta un lector que se encuentra a una distancia corta. Si se adhieren etiquetas RFID a productos y equipos, es posible hacer un registro del inventario en tiempo real; esto se emplea en aplicaciones IoT de estantes inteligentes y autoservicios.

5) **LPWANs :** son tecnologías recientes que permiten comunicaciones de largo rango, emplean radios de bajo coste y bajo requerimiento energético y sus baterías también son baratas y duraderas. Se pueden emplear para redes IoT de

gran escala sobre áreas amplias. Actualmente, existen tres grandes tecnologías LPWAN: Sigfox, LoRa/LoRaWan y NB-IoT.

2.2.2 LoRa

Ilustración 2 LoRa

LoRa fue patentada por Semtech y es una técnica de modulación inalámbrica que deriva del Espectro de Ensanchado Chirp o Chirp Spread Spectrum(CSS), la cual usa pulsos chirps modulados con una banda de frecuencia lineal para codificar la información, con el objetivo de ensanchar la señal chirp (se usa de portadora y contiene la información codificada que se quiere transmitir) en el dominio de la frecuencia.

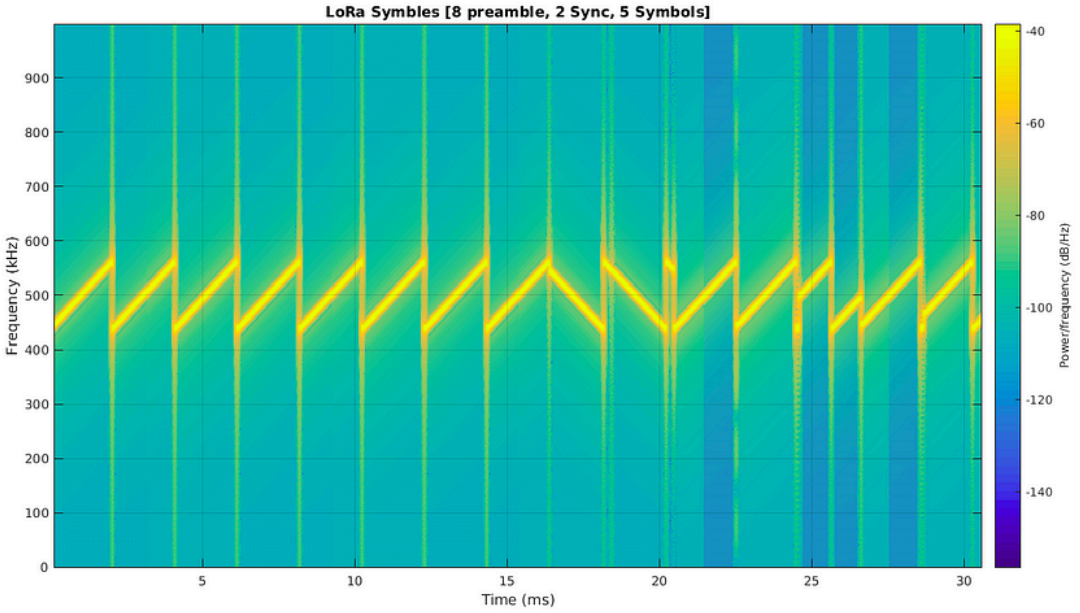


Ilustración 3 Mensaje LoRa

LoRa es ideal para aplicaciones que transmiten pequeños paquetes de datos con tasa de bits (Bit Rate) bajos. Los datos se pueden enviar a distancias más largas en comparación con otras tecnologías por lo que es idóneo para usar con sensores y actuadores que operan en modo bajo consumo.

En el modelo OSI, LoRa es el nivel físico (OSI-1) y LoRaWAN se refiere a la capa 2 y 3 (capa de Red y de enlace de Datos).

2.2.2.1 Banda ISM y Parámetros Regionales

LoRa opera en el espectro de radio sin licencia como parte de la banda de radio ISM (Industrial, Científica y Médica).

El protocolo LoRa tiene las especificaciones oficiales de cada región y se denominan Parámetros Regionales en los cuales se describen los parámetros de comunicación recomendados para el uso en cada país/región. En Europa se utiliza el Plan de Frecuencias EU 863-870, en España el Plan de Frecuencias asignado es el EU 863-870 y el EU 433 (todavía no hay un plan oficial) según la web oficial de The Things Network.

Parámetros importantes de comunicación

- **Canal de la banda:** frecuencia central que representa el canal.

- **Factor de esparcimiento o Spreading Factor(SF):** en la tecnología CSS los símbolos o chirps (nº de "pasos" en los que se divide la señal chirp) se usan para representar los datos, SF representa el nº de bits que tiene cada símbolo, 2^{SF} son los valores que se pueden representar en el símbolo y $SF = 7, 8, 9, 10, 11, 12$. El SF afecta el Data Rate (DR), Time-on-Air (ToA), la vida de la batería y la sensibilidad del receptor. A menores SFs mayor será la transmisión de datos ya que conforme se incrementa el SF, aumenta el tiempo de duración de los símbolos transmitidos .

- **Ancho de Banda o Bandwidth(BW):** indica el ancho de frecuencia que se va a usar, LoRa permite 3 anchos de banda: 125 KHz, 250 KHz y 500 KHz. En Europa se utilizan anchos de banda de 125 KHz y 250 KHz.

- **Coding Rate(CR):** nos permite añadir bits de más en la transmisión; para el caso en el que los datos sean corrompidos por interferencias, sea posible devolverlos a su estado original, esto se conoce como Corrección de Errores hacia Adelante o Forward Error Correction(FEC). El CR representa la proporción de datos transmitidos que contiene información útil.

- **Chip Rate (R_c):** representa el nº de chips transmitidos por segundo y se expresa en chips/segundo.

- **Symbol Rate (R_s):** representa el nº de símbolos transmitidos por segundo y se expresa en símbolos/segundo.

- **Data Rate o Bit Rate (R_b):** representa el nº de bits transmitidos por segundo y se expresa en bits/segundo. Siempre se cumplirá la relación: $R_c > R_s$.

- **Chip Duration (T_c):** es la inversa del Chip Rate, representa el tiempo de duración de cada chip transmitido.

- **Symbol Duration (T_s):** es la inversa del Symbol Rate, representa el tiempo de duración de cada símbolo transmitido.

- **Duty Cycle (DC):** indica la fracción de tiempo que un recurso está ocupado, está regulado por gobiernos; para el plan de frecuencias EU 868, el valor es del 1%. Como la banda ISM no requiere licencia, para disminuir las interferencias y el ruido, se impone un tiempo máximo de uso por canal. Por ejemplo en Europa, de 100 segundos, solo se puede estar emitiendo 1 segundo como máximo.

- **Time On Air (ToA):** tiempo que dura la emisión del paquete. El ToA se calcula de la siguiente forma:

$$ToA = T_{packet} = T_{preamble} + T_{payload} \qquad Ecuación\ 1$$

Donde :

$$T_{preamble} \equiv duración\ en\ segundos\ del\ preamble$$

$$T_{payload} \equiv duración\ en\ segundos\ del\ payload$$

Valores bajos del SF aumentan la velocidad transmisión de datos, ante un mismo tamaño de paquete, el ToA disminuye. Esto influye directamente en la duración de la vida útil de la batería del dispositivo, ya que el transceptor permanece en estado activo menos tiempo.

2.2.3 LoRaWAN

Ilustración 4 LoRaWAN

LoRaWAN es un protocolo de capa Media Access Control (MAC) construido encima de modulación LoRa, es una capa software que define cómo los dispositivos usan el hardware LoRa, por ejemplo cuando transmiten y el formato de los mensajes. La última versión es la 1.1 y fuen lanzada en 2017.

El protocolo fue desarrollado y es mantenido por la Alianza LoRa, la cual se encarga de garantizar la interoperabilidad de todos los productos y tecnologías LoRaWAN.

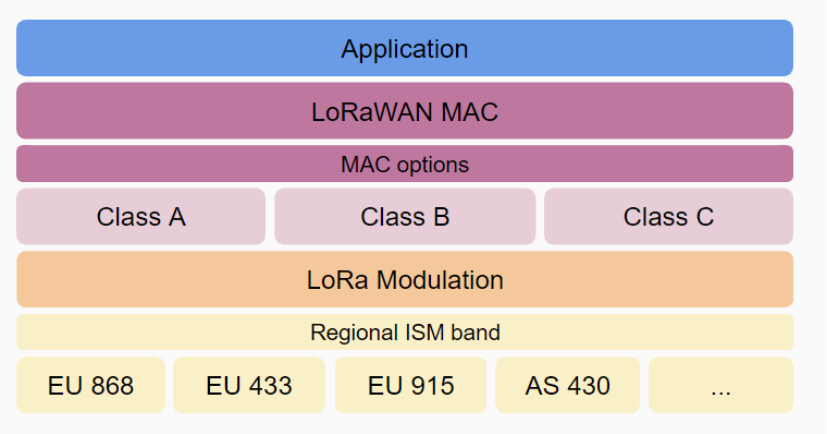


Ilustración 5 LoRaWAN Stack

Algunas ventajas de la tecnología LoRaWAN:

- Ultra bajo consumo: los dispositivos están optimizados para operar en modo bajo consumo y pueden durar hasta 10 años alimentándose con un botón de litio.
- Alta capacidad: los servidores de red LoRaWAN pueden procesar millones de mensajes enviados por miles de gateways.
- Implementación: es sencillo realizar implementaciones tanto para redes públicas como privadas utilizando el mismo software y hardware.
- Seguridad end-to-end: el protocolo LoRaWAN garantiza la seguridad desde el nodo hasta el servidor de aplicación usando la encriptación AES-128.
- Bajo coste: la infraestructura requerida es mínima, los dispositivos finales son de bajo coste y el software es open source.

Algunas de las aplicaciones de LoRaWAN recomendadas por Semtech:

- Monitorización de la cadena de frío de vacunas.
- Conservación de fauna: se puede usar con sensores de seguimiento en especies de animales protegidos.
- Seguridad Alimentaria: monitoreo de la temperatura de la comida para garantizar su calidad.
- Eficiencia en centros de trabajo: nivel de ocupación de habitaciones, monitoreo de temperatura y disponibilidad de aparcamiento, consumo energético, etc.

- Granjas inteligentes y agricultura: es uno de los campos con más aplicabilidad, se puede usar para monitorizar en tiempo real parámetros como la humedad del suelo y para optimizar el proceso de irrigación, de esta forma se puede reducir hasta un 30 % la cantidad de agua.

2.2.3.1 Arquitectura LoRaWAN

La topología de una red LoRaWAN es de tipo estella y los datos se envían a un elemento central a través de Gateways, las cuales se comunican con un servidor de red y finalmente con un servidor de aplicación. Presenta 4 elementos de red:

1) **Dispositivos/Nodos finales (End Device/Node)** : son los dispositivos de hardware físico que contienen capacidades de detección, algo de potencia de cálculo y un módulo de radio para traducir los datos en una señal de radio, además de los sensores o actuadores con los que se conectan. Un dispositivo final está compuesto por un módulo de radio con antena y por un microprocesador, tiene un transceptor y se alimenta por baterías.

   Los nodos finales envían de forma inalámbrica los mensajes modulados LoRa hacia la compuerta/concentrador y la comunicación es bidireccional: de nodo a compuerta(uplink) y de compuerta a nodo(downlink), la comunicación entre nodos no está definida.

   Existen 3 tipos de dispositivos definidos por la especificación LoRaWAN (Versión 1.0.4, octubre de 2020): Clase A, Clase B y Clase C. Todos los dispositivos implementarán la clase A, los Clase B y C son extensiones de la anterior.

   Clase A(ll): estos dispositivos son capaces de enviar mensajes uplink en cualquier momento, solo después se puede realizar un downlink. Una vez que se completa la transmisión , se abren 2 ventanas cortas de recepción (RX1 y RX2) para recibir los mensajes downlink de la red, la duración aproximada de ambas ventanas es de 1 segundo aunque dependerá del dispositivo. Existe un delay entre el final de la transmisión uplink y el inicio de las dos ventanas de recepción (RX1 Delay y RX2 Delay respectivamente). El servidor de red responderá en una de las ventanas de recepción; si esto no ocurre, la próxima transmisión downlink ocurrirá inmediatamente después de que finalice la próxima transmisión uplink.

   Estos dispositivos tienen muy bajo consumo energético y están casi siempre en modo “sleep”, en este estado el servidor de red no puede contactar con los dispositivos.

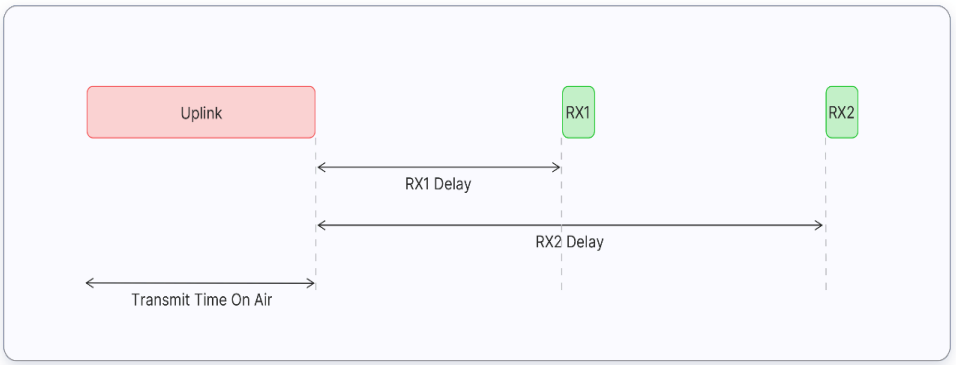


Ilustración 6 Dispositivo Clase A

Clase B(eacon): es una extensión de todas las características de los dispositivos Clase A, se abren de forma periódica ventanas de recepción llamadas ping slots para recibir los mensajes downlink. La red emite un beacon sincronizado a través de las compuertas de forma periódica que es recibida por los nodos finales. Este beacon permite a los nodos sincronizarse con el servidor de red, para que este sepa cuándo mandar un mensaje downlink a un dispositivo específico o a un grupo entero. Debido a que pasan más tiempo en modo activo, los dispositivos Clase B tienen mayor consumo energético en comparación con los Clase A.

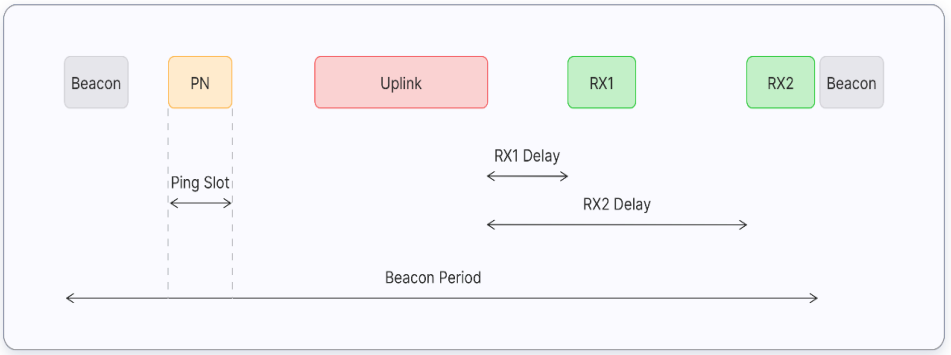


Ilustración 7 Dispositivo Clase B

Clase C(ontinuous): también son una extensión de la clase A y mantienen la ventana de recepción abierta a menos que ocurra una transmisión uplink por lo que pueden recibir un mensaje downlink en casi cualquier instante. Estos dispositivos, al igual que los Clase A, abren dos ventanas de recepción, RX1 y RX2; después de enviar un mensaje uplink se abre una RX2 corta, seguido de una RX1 corta y después se abre una RX2 continua hasta el próximo uplink. Es la clase con menor latencia y por ende la que mayor consumo de energía requiere.

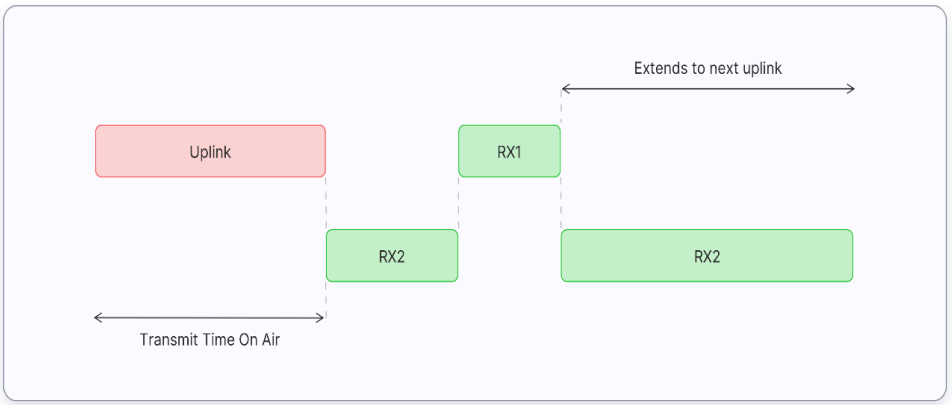


Ilustración 8 Dispositivo Clase C

2) **Concentrador/Compuerta (Concentrator/Gateway):** cada concentrador se registra en un servidor de red LoRaWAN y se encarga de recibir los mensajes LoRa transmitidos por los nodos finales y enviarlos hacia el servidor de red. Las compuertas se conectan con los servidores de red a través de otras tecnologías de red: celulares(3G/4G/5G), WiFi, Ethernet, etc.

3) **Servidor de Red (Network Server):** herramienta de software que conecta los sensores y gateways con las aplicaciones superiores. Aquí se envían desde las compuertas, los mensajes transmitidos por los nodos y donde tienen lugar los procesos más complicados relacionados con el tratamiento de datos. Se encarga principalmente de la gestión de nodos y gateways y del control y supervisión de la red.

   Cuando un nodo emite el mensaje, el paquete llega a varias compuertas y todos llegan al Servidor de Red, quien elegirá uno. El servidor también determina el GW que realiza la transmisión downlink, en caso necesario de que esté pendiente enviar algo, que recibe el dispositivo en RX1 o RX2.

4) **Servidor de Aplicación (Aplication Server):** son los encargados del tratamiento de los datos transmitidos y se comunican con los servidores de red, aunque sean independientes de la red LoRaWAN. Genera todas las payloads de la capa de aplicación y las envía a los dispositivos conectados a través del Servidor de Red. Una red LoRaWAN puede tener más de un Servidor de Aplicación.

5) **Servidor Join (Join Server):** programa ejecutándose en un servidor, que procesa los mensajes join-request enviados por los dispositivos.

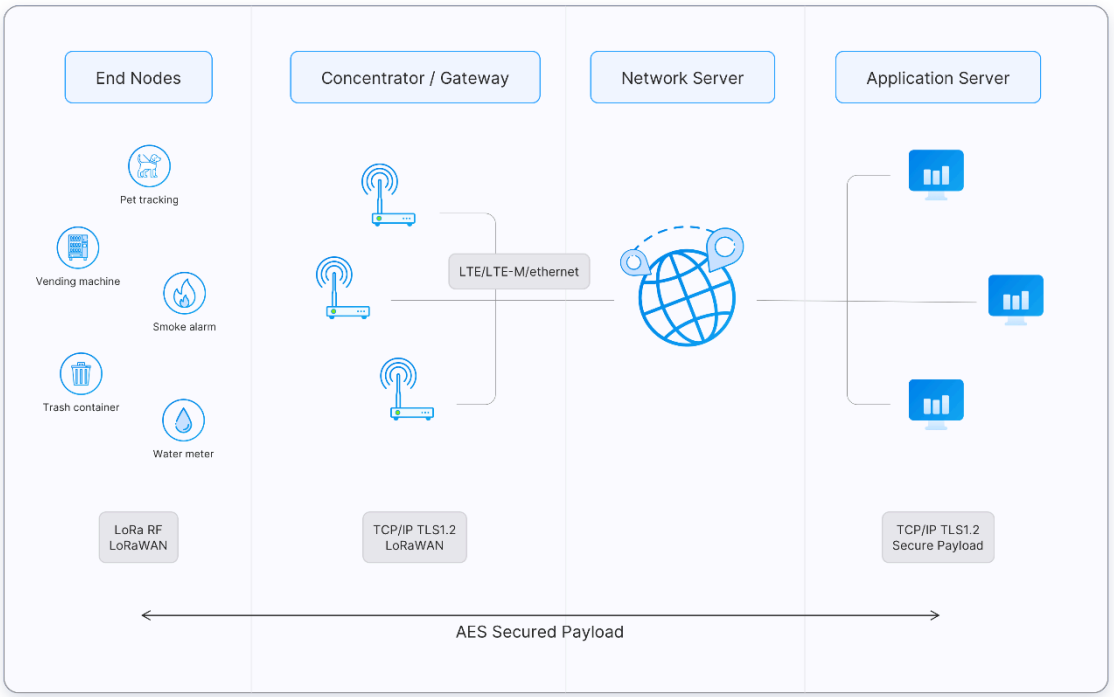


Ilustración 9 Arquitectura de una típica red LoRaWAN

Arquitectura de este proyecto:

1) **Dispositivo final:** placa CubeCell Dev-Board Plus (HTCC-AB02), nodo final de Clase C, que se conecta a diferentes sensores (DHT11, Tilt Switch, KY-033, KY-036)..

2) **Concentrador/Compuerta:** se ubica en la Torre Profesor Agustín Arévalo de la Universidad de La Laguna.

3) **Servidor de Red:** se utilizará The Things Stack como Servidor de Red LoRaWAN, construido por The Things Industries.

4) **Servidor de Aplicación (Aplication Server):** corresponde a la parte de The Things Network que, después de haber dado de alta la Aplicación, aplica el payload formatter a los paquetes uplink recibidos y los publica usando MQTT. La última parte del Servidor de Aplicación corresponde con la aplicación que guarda los datos en una base de datos SQLite y los visualiza en una aplicación web usando Dash.

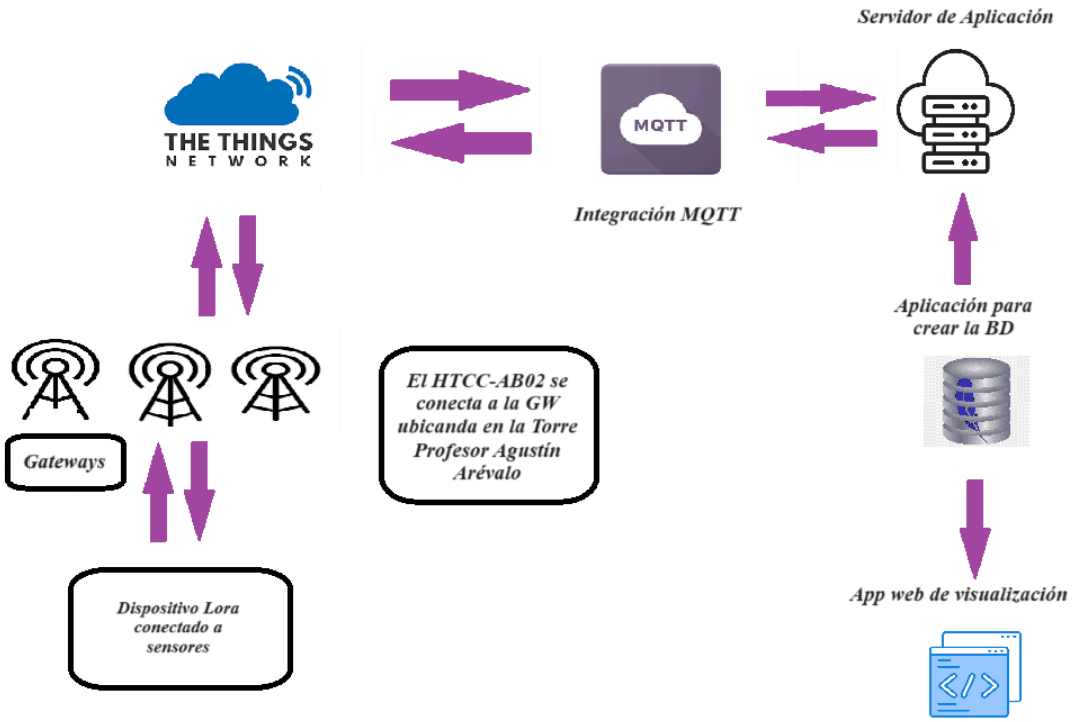


Ilustración 10 Arquitectura del proyecto

2.2.3.2 Tipos de Mensajes

Los mensajes se usan para transportar comandos MAC y datos de Aplicación. Los principales tipos de mensajes son:

1) **Uplink y Downlink:** estos mensajes se clasifican dependiendo de la dirección en que viajan.

   - Mensajes Uplink: estos mensajes son enviados por los nodos a través de una o varias gateways hacia el Servidor de Red, el cual los transmite a los servidores Join o de Aplicación si es necesario.

   - Mensajes Downlink: los envía el Servidor de Red a un nodo en específico y a través de una sola compuerta, estos mensajes pueden iniciarse en los servidores Join o de Aplicación.

2) **MAC:** el protocolo LoRaWAN define varios mensajes de tipo MAC, los más importantes son:

   - Join-request: siempre lo inicializa el dispositivo y lo envía al Servidor de Red, que después lo envía al Servidor Join del dispositivo, este mensaje no está encriptado.

- Join-accept: lo genera el Servidor Join, el mensaje está encriptado con llaves de seguridad y es enviado al dispositivo final correcto por el Servidor de Red.

- Rejoin-request: siempre lo inicializa el dispositivo y lo envía al Servidor de Red, hay 3 tipos (1, 2 y 3) y se utilizan para una nueva sesión para el dispositivo. La red responde al mensaje con un mensaje Join-accept.

3) **Mensajes de Datos:** se usan para transmitir tanto comandos a la capa MAC como datos de Aplicación; se pueden combinar en un solo mensaje. Pueden ser de 2 tipos:

   - Confirmados: necesitan confirmación por parte del receptor.

   - No confirmados: no necesitan confirmación por parte del receptor.

El paquete LoRa está compuesto por 3 partes: preamble, header (opcional) y el payload:

1) Preamble: secuencia de 10 up-chirps seguidos de 2 down-chirps, le indican al receptor el inicio del mensaje.

2) Header: indica el modo de operación por defecto y puede estar implícito o explícito.

3) Payload: contiene la información que se desea transmitir, en este proyecto la payload contiene los datos de los sensores.

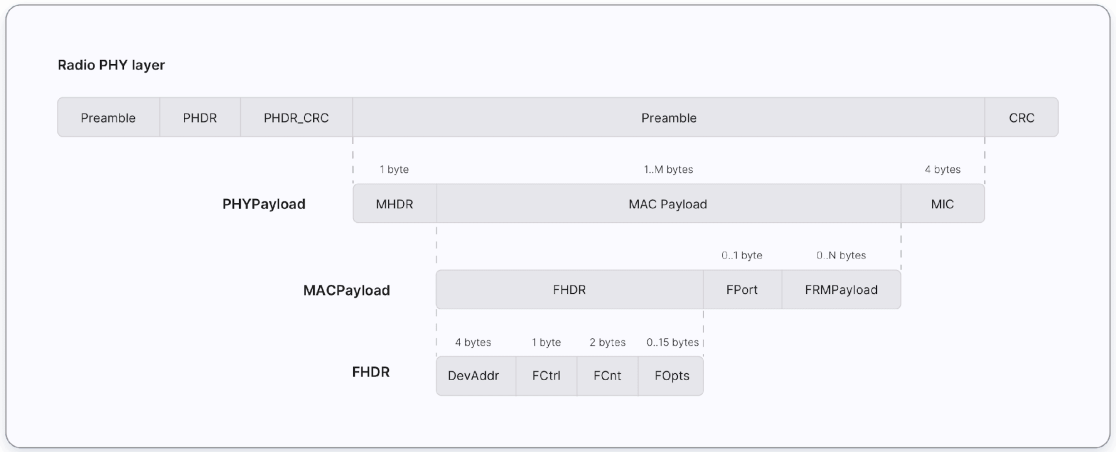


Ilustración 11 Estructura de un mensaje de datos

2.2.3.3 Seguridad

1) **Llaves de seguridad:** la versión 1.0 de LoRaWAN especifica un nº de llaves de seguridad: NwkSKey, AppSKey y la AppKey; todas son de 128 bits y el algoritmo usado es el AES-128. Este algoritmo se basa fundamentalmente en el IEE 802.15.4, estándar que define el nivel físico y de control de acceso al medio (MAC) en redes inalámbricas de área personal con tasas de transmisión bajas.

2) **Llaves de sesión:** cuando un nodo se une a la red, se generan las llaves de sesión de aplicación (AppSKey) y la de sesión de red (NwkSKey). La NwkSKey es compartida con la red mientras que la AppSKey se mantiene privada.

   La NwkSKey se usa como medio de interacción entre el Nodo y el Servidor de Red para validar la integridad de cada mensaje a través de su MIC (Message Integrity Code)

   La AppSKey para encriptar y desencriptar la payload, la cual permanece encriptada desde el Nodo hasta el componente del Servidor de Aplicación de The Things Network, por lo que sólo el usuario es capaz de leer el contenido de los mensajes que se envían o reciben.

   Dependiendo del método de activación del nodo, estas llaves permanecen constantes hasta que las cambie el usuario, o se regeneran en cada activación y son únicas para cada dispositivo y sesión, para OTAA y ABP respectivamente (se explicará en el siguiente apartado con más detalle)

3) **Llave de aplicación:** solo es conocida por el dispositivo y la aplicación. Los dispositivos activados por el método OTAA usan esta llave para obtener dos llaves de sesión en el proceso de activación.

2.2.3.4 Activación de Dispositivos Finales

El proceso de activación consiste en registrar los dispositivos en la red LoRa antes de proceder a la transmisión de mensajes. Existen dos métodos de activación:

1) **Over-The-Air-Activation (OTAA):** es el método de activación más seguro y recomendado. Los dispositivos realizan un procedimiento join con la red, durante el cual se le asigna al dispositivo llaves de seguridad y una dirección dinámica de dispositivo.

2) **Activation By Personalization (ABP):** requiere la codificación rígida de la dirección del dispositivo y sus llaves de seguridad en el propio dispositivo. Este método es menos seguro que OTAA y tiene la desventaja de que para cambiar de proveedor de red, hay que cambiar las llaves del dispositivo manualmente. Este

método no se explicará en detalle ya que para este proyecto se usó el método OTAA.

La siguiente imagen describe los pasos a seguir en el proceso de activación de dispositivos por el método OTAA:

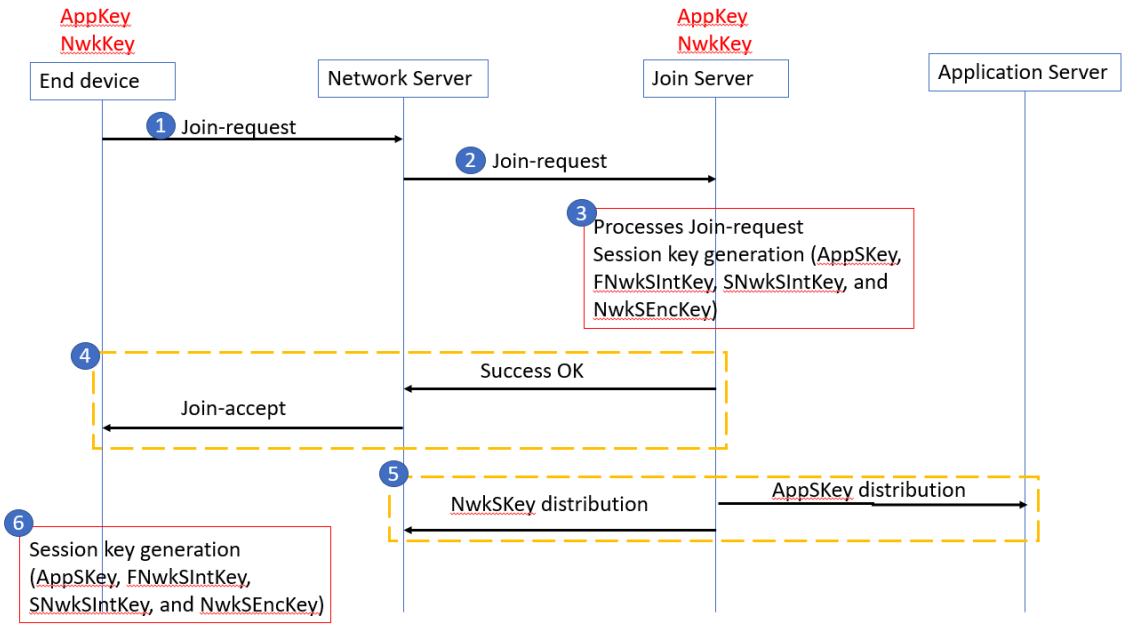


Ilustración 12 Flujo de proceso de activación por OTAA

Paso 1: el proceso de Join siempre lo inicializa el dispositivo el cual envía un mensaje Join-request , al servidor al que se va a unir, con los siguientes campos: JoinEUI, DevEUI y DevNonce.

El MIC se calcula usando el NwkKey usando todos los campos del mensaje Join-request y después se añade a este.

Paso 2: el Servidor de Red le envía el mensaje Join-request al correspondiente Servidor Join.

Paso 3: el Servidor Join procesa el mensaje Join-request y si el dispositivo tiene permitido unirse a la red, se generarán las llaves de sesión (AppKey, FNwkSIntKey, SNwkSIntKey, and NwkSEncKey)

Paso 4: si se realizan los pasos anteriores de forma exitosa, se genera un mensaje Join-accept el cual contiene los siguientes campos: JoinNonce, NetID, DevAddr, DLSetting, RxDelay y CFList.

El MIC se añade al mensaje Join-accept y es calculado usando la NwkKey. El mensaje Join-accept está encriptado con la NwkKey (si es una Join-request) o por la JSEnckey (si es una Rejoin-request). El Servidor de Red envía, como downlink, el mensaje Join-accept encriptado al dispositivo final.

Paso 5: el Servidor Join envía el AppSKey al Servidor de Aplicación y las 3 llaves de sesión (FNwkSIntKey, SNwkSIntKey y NwkSEncKey) al Servidor de Red.

Paso 6: el nodo desencripta el mensaje Join-accept usando la operación de encriptación AES y usa las AppKey, NwkKey y el JoinNonce para generar las llaves de sesión, de esta forma queda registrado el dispositivo.

2.2.4 MQTT

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería utilizado en IoT e IIoT entre dispositivos tales como sensores, dispositivos empotrados, PLCs industriales, etc y fue estandarizado por OASIS e ISO. El protocolo es un conjunto de reglas que define la forma en que los dispositivos IoT pueden publicar y suscribirse a datos en la red.

El protocolo, cuya versión más reciente es la 3.1.1, se basa en eventos y conecta los dispositivos usando un patrón publicar/suscribir (Pub/Sub). El emisor (Publisher) y el receptor (Subscriber) se comunican a través de tópicos (Topics) y están desacoplados entre sí, la conexión entre ambos ocurre a través de un broker MQTT, el cual filtra todos los mensajes entrantes y los y los distribuye hacia los Subscribers.

Un cliente MQTT es quien publica o se suscribe a los mensajes sobre vario(s) tópico(s), también puede realizar las dos acciones simultáneamente. Puede ser cualquier dispositivo (desde un microcontrolador hasta un servidor enorme) capaz de ejecutar una librería MQTT y conectarse a un broker en una red.

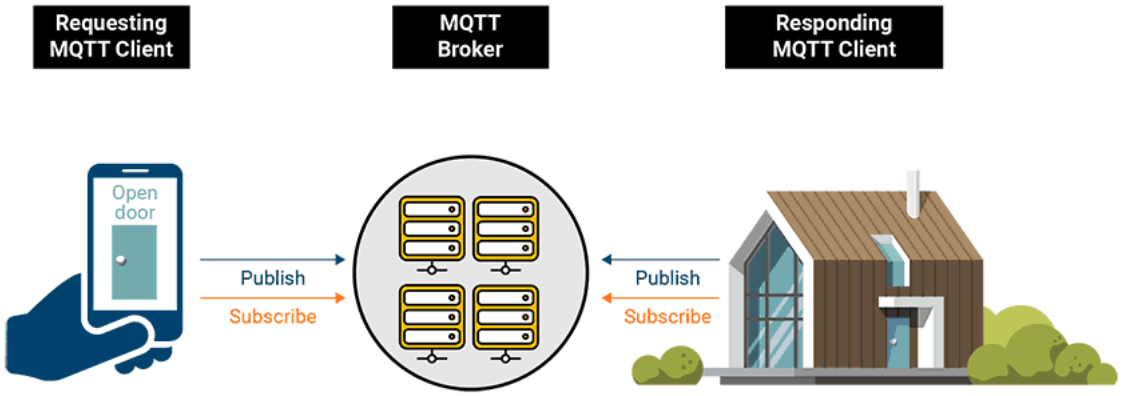


Ilustración 13 Ejemplo de abrir puerta inteligente con dispositivo móvil usando MQTT

Un cliente MQTT también se puede considerar a un ordenador, o cualquier dispositivo, que use protocolos de comunicación TCP/IP y ejecute un software con funcionalidades de MQTT Client.

El protocolo MQTT se basa en TCP/IP, por lo que tanto cliente como el broker tendrán el stack TCP/IP.

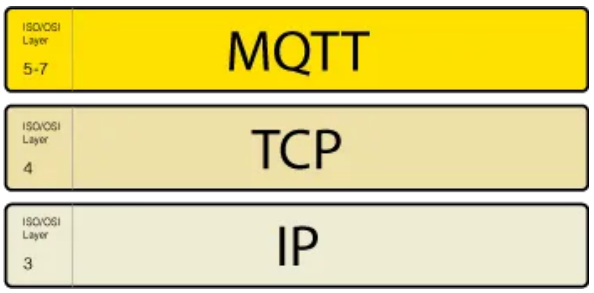


Ilustración 14 Stack TCP/IP

Ventajas del protocolo MQTT:

- El requerimiento de recursos es mínimo ya que es ligero y eficiente.
- Permite la mensajería bidireccional entre el dispositivo y la nube.
- Puede implementarse para conectar millones de dispositivos.
- Permite la entrega confiable de mensajes a través de los 3 niveles de QoS.
- Puede implementarse con redes inestables.
- Tiene seguridad habilitada por lo que funciona bien con TLS y protocolos de autentificación comunes.
- Óptimo para entornos con bajo BW y que requieren bajo consumo energético gracias al formato binario de sus mensajes.

2.2.4.1 Modelo de Mensajería MQTT

El modelo de mensajería se basa en tópicos (topics) y suscripciones (subscriptions), los tópicos son strings a los cuales se suscriben y publican los mensajes, son jerárquicos y pueden contener múltiples niveles separados por / (similar al path de un archivo dentro de un directorio).

Ejemplo de tópicos: myhome/kitchen/smartdishwasher

myhome/kitchen/smartfridge

Un cliente puede suscribirse a varios tópicos y a su vez un tópico puede tener varios suscriptores. Los clientes también pueden anular su suscripción a los tópicos enviándole al broker un mensaje UNSUBSCRIBE, el cual después de anular la suscripción responde con un mensaje UNSUBACK.

Una de las características más importantes del modelo es el filtrado, que se puede hacer ya sea por tópico (es el más utilizadoy consistente), contenido o tipo.

Las wildcards son un mecanismo para suscribirse a varios tópicos simultáneamente, hay dos tipos de wildcards:

- 1-nivel : se representa con el símbolo + y permite la sustitución de un solo nivel dentro de un tópico.

Ilustración 15 Wildcard de 1-nivel

- Multinivel: permite la suscripción a varios niveles dentro de un tópico y se representa con el símbolo #. El cliente recibe todos los mensajes que se publican en los tópicos de los niveles que están por debajo jerárquicamente de donde se coloca #; si solo se usa #, el cliente recibirá todos los mensajes enviados al roker.

Ilustración 16 Wildcard multinivel

MQTT define 3 niveles de QoS y cuanto mayor sea el nivel, necesitará más recursos y puede aumentar la latencia y el tráfico de red:

- QoS 0: los mensajes se envían una sola vez sin confirmación, por lo que existe el riesgo de que se pierdan. Este QoS se utiliza en situaciones cuando la pérdida del mensaje es aceptable o los mensajes no son críticos; su uso es común para enviar datos de sensores donde la pérdida ocasional de los datos, como en el caso de este TFG, no impacta significativamente los resultados.

- QoS 1: se envían al menos una vez los mensajes, existe la confirmación y se reenvían si es necesario. El publisher le envía el mensaje al broker y espera por confirmación antes de seguir adelante, si no la recibe el publisher le reenvía el mensaje al broker. Este QoS se usa cuando la pérdida de mensajes no es aceptable pero se toleran los mensajes duplicados.

- QoS 2: los mensajes se envían solo una vez, se espera confirmación y se reenvían hasta que son recibidos una vez por el suscriptor. Este QoS se usa en situaciones donde la pérdida o duplicación de mensajes es inaceptable y la

confirmación ocurre en 2 pasos: el broker almacena el mensaje hasta que el subscriber recibe y reconoce el mensaje.

Es importante prevenir la pérdida de mensajes en caso de que ocurra un fallo en la red o el servidor, para ello los mensajes se almacenan en el servidor hasta que se entregan al suscriptor. Hay 3 tipos de persistencia:

- Non-persistent: es la opción por defecto y los mensajes no se almacenan en el servidor, se usa para situaciones cuando la pérdida del mensaje es aceptable.

- Queued persistent: los mensajes están guardados en el servidor hasta que se entregan al suscriptor, si el suscriptor no está disponible (por ejemplo si ocurre un fallo en la red) se mantienen en la cola. Se usa cuando es necesario que el suscriptor reciba todos los mensajes aunque se haya desconectado de la red.

- Persistent with acknowledgement: los mensajes se almacenan en el servidor hasta que el suscriptor reconoce que los ha recibido. Es tipo se utiliza cuando es fundamental confirmar que el suscriptor recibe los mensajes.

2.2.5 Conceptos de desarrollo web y Dash

Una aplicación web es un programa que se ejecuta en un servidor web y se accede a través de un navegador (Google Chrome, Mozilla Firefox, Safari) utilizando una conexión a Internet; las aplicaciones web son accesibles desde cualquier dispositivo con un navegador compatible y acceso a la red y tienen dos partes principales:

1) Interfaz de usuario (Front-End): es la parte de la aplicación que interactúa directamente con el usuario, incluye elementos visuales y de interacción (botones, formularios, etc). Se suele programar en lenguajes como HTML, CSS y JavaScript; se ejecuta en el navegador web.

2) Lógica del Servidor (Back-End): parte que maneja la lógica de la aplicación, la gestión de bases de datos y el procesamiento de las solicitudes del usuario. Se desarrolla utilizando lenguajes de programación y frameworks como Python (Django, Flask), Ruby (Ruby on Rails), PHP (Laravel), JavaScript (Node.js), etc. Esta parte de la aplicación se ejecuta en el servidor web.

Los servidores web procesan las solicitudes que mandan los usuarios (clientes), el servidor de aplicación completa las tareas solicitadas y la base de datos se usa para almacenar información necesaria. Esta comunicación suele hacerse con HTTP y websocket, su diseño y gestión es una parte importante en el desarrollo de una web "tradicional".

En este proyecto se usó Dash [26] para crear la aplicación web, Dash es un poderoso framework de Python desarrollado por Plotly que se usa en el desarrollo de aplicaciones web interactivas y para la visualización de datos.

Dash incluye gran variedad herramientas y componentes pre-construidos que facilitan el proceso de hacer una app web, ya que no se requiere gran cantidad de código ni tener grandes conocimientos de HTML, CSS, JavaScript, etc.

Ventajas que ofrece Dash frente a otras tecnologías:

1) Es open source y se puede escalar lo suficiente para trabajar con grandes cantidades de datos y altos niveles de tráfico de usuario. También se puede implementar en un gran número de plataformas incluyendo servicios basados en la nube.

2) Incluye una gran cantidad de componentes pre-construidos como gráficas, tablas, dropdowns, sliders, tabs, etc y se pueden estilizar usando CSS.

3) Es posible crear callbacks que actualizan el contenido de la aplicación en tiempo real dependiendo de las entradas generadas por el usuario.

4) Dash implementa múltiples temas y layouts que ayudan a crear apps con aspecto profesional e integra librerías de visualización como Plotly que permiten generar todo tipo de gráficas.

5) La principal ventaja es que se programa una aplicación en Python que incluye tanto el Back-End como el Front-End y no es necesario saber JavaScript para programar el Front-End. Tampoco hay que preocuparse por la gestión de la comunicación entre ambas partes.

Las aplicaciones Dash se construyen con dos bloques fundamentales: Layout y Callbacks.

La Layout se refiere a la estructura de la aplicación web y define la disposición de los distintos componentes: tablas, gráficas, dropdowns, tabs, etc. Dentro de este bloque tenemos 3 componentes:

- `dash_html_components`: permite crear una serie de clases para etiquetas y elementos HTML como headers, paragraphs, divs, etc.

- `dash_core_components`: provee de elementos de más alto nivel que son interactivos: gráficas, dropdowns, tabs, etc; los cuales se construyen encima de los `dash_html_components`.

- `dash_bootstrap_components`: provee de componentes basados en el framework Bootstrap.

Los Callbacks de Dash permiten añadir dinamismo e interactividad a las aplicaciones, son funciones de Python que se activan en respuesta a entradas de usuario como seleccionar una opción en un dropdown o dar click en un botón. Dash llamará

automáticamente a las funciones asociadas a dichos callback para actualizar el contenido de la app. Los Callbacks presentan 2 componentes básicos: las `inputs` o entradas y las `outputs` o salidas. Las `outputs` son datos asociados al evento que provoca la ejecución del callback, pueden ser gráficas o cajas de texto y son los que se actualizan cuando se llama a la función del callback.

3 Componentes de Hardware y Software

3.1.1 Dispositivos LoRa

CubeCell Dev-Board Plus (HTCC-AB02)

CubeCell ™ es un producto de Heltec que se usa principalmente en aplicaciones nodo para LoRa/LoRaWAN.

Características principales:

- Compatible con Arduino.
- Chips basados en ASR6501x (ASR 650, ASR 6502)
- Soporta la versión LoRaWAN 1.0.2
- Mínimo consumo energético, 3.5 uA en modo “deep sleep”.
- Interfase de batería SH1.25-2 en el dispositivo y cuenta con un sistema de energía solar con el que se puede conectar directamente a un panel solar de 5.5-7 V.
- Interfase micro USB que cuenta con distintas protecciones: ESD, contra cortocircuito, RF, etc.
- Pantalla display OLED dot matrix de 0.96 inch.

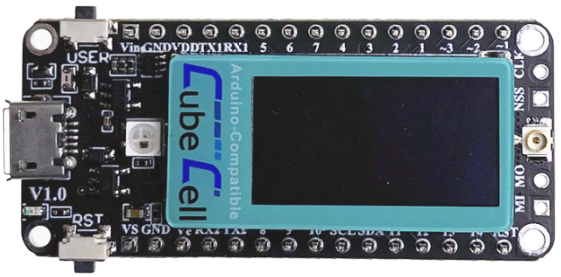


Ilustración 17 CubeCell Dev-Board Plus (HTCC-AB02)

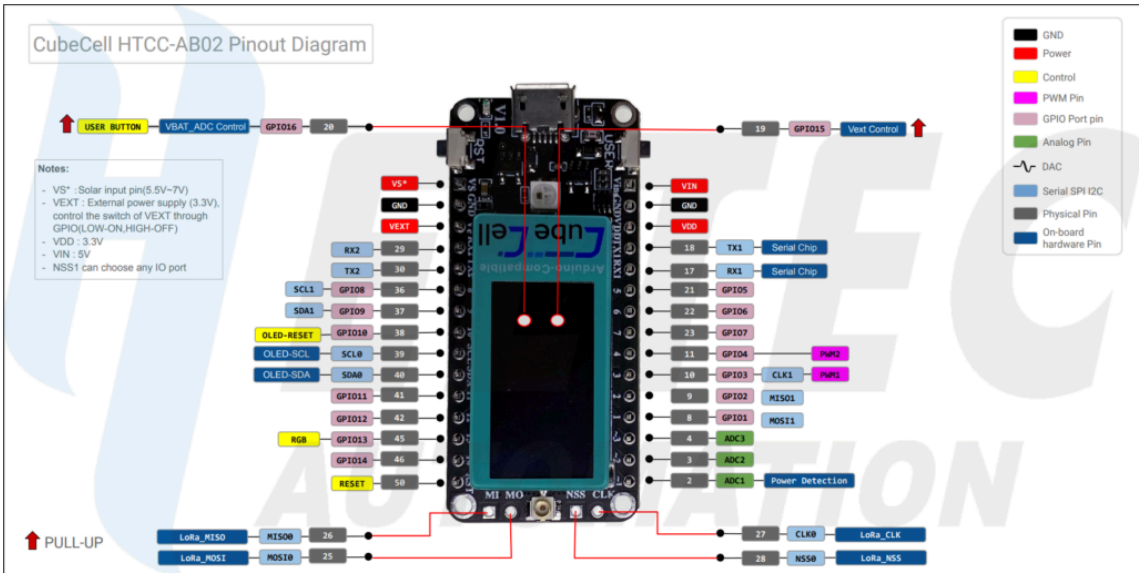


Ilustración 18 Diagrama de pines del CubeCell Dev-Board Plus (HTCC-AB02)

3.1.2 Sensores

3.1.2.1 DHT11

Es un sensor digital de Arduino que mide temperatura y humedad relativa.

Características principales:

- Muy barato y requiere bajo consumo.
- Alimentación 3.3 y 5 V.
- Rango de medida de temperatura: 0º a 50º con 5% de precisión.
- Rango de medida de humedad relativa: 20% a 80% de precisión.
- Rapidez de medida: 1 muestra/segundo.

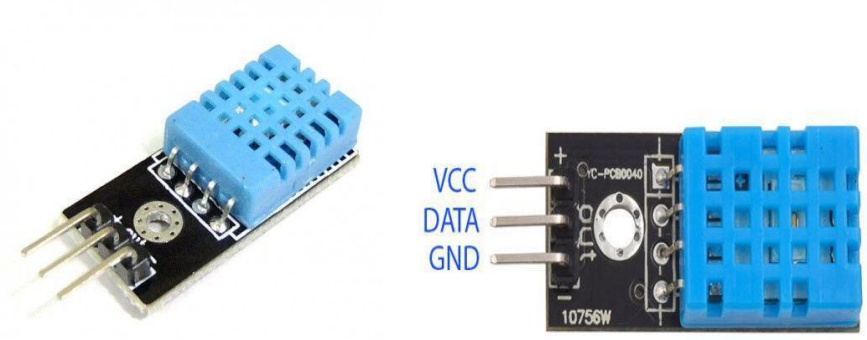


Ilustración 19 Imagen y diagrama de pines del HT11

El pin de salida de datos del sensor emite una señal digital y los datos consisten en una parte decimal y una entera, una transmisión completa es de 40 bits y el sensor primero envía la parte alta de la lectura.

3.1.2.2 Tilt Switch

Sensor de Arduino que detecta inclinación.

Características principales:

- Voltaje de operación de 5 V.
- Salida digital de 0 y 1.
- Alta sensibilidad.
- Se enciende la luz Led cuando se activa el módulo.

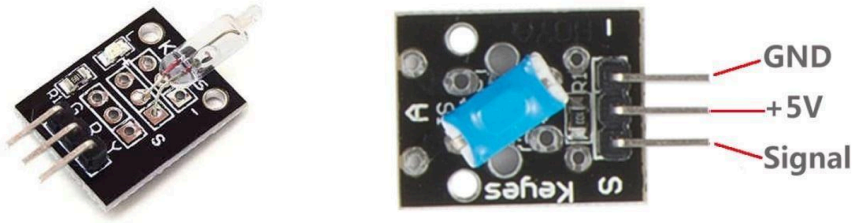


Ilustración 20 Tilt switch y diagrama de pines en encapsulado distinto

En el interior del sensor hay una bola conductora que cierra el circuito al hacer contacto con los pines metálicos presentes en el cilindro. Cuando la señal de salida está en ALTA significa que no hay inclinación y cuando está en BAJA es que no se ha cambiado la posición del sensor.

3.1.2.3 KY-036

Este módulo es un sensor táctil que funciona con un transistor Darlington KSP13, presenta una pequeña pata libre que cuando se toca manda una señal. La sensibilidad es ajustable con el potenciómetro.

Características principales:

- Voltaje de funcionamiento de 3.3-5 V.
- Emite una señal al led2 si se toca la base del sensor KSP13.
- Salida de señal digital de 0 y 1, que se emite ante la detección del contacto.
- Salida analógica: valor de medición directa de la unidad del sensor.

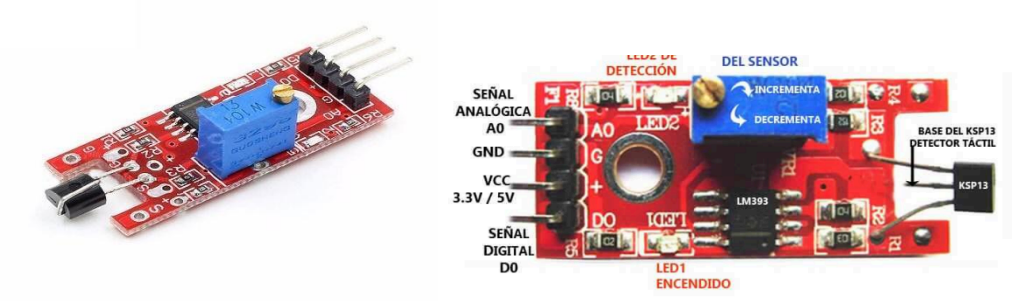


Ilustración 21 KY-036 y su diagrama de pines

3.1.2.4 KY-033

Es un sensor de infrarrojos que detecta si la superficie en frente es reflectiva u opaca (puede ser sensible a la luz ambiental). El módulo está compuesto por un emisor/receptor de infrarrojos, un potenciómetro, un comparador diferencial LM393, un LED indicador y 4 resistencias.

Características principales:

- Voltaje de operación: 3.3-5.5 V.
- Señal de salida TTL (1 si se detecta obstáculo, 0 si no se detecta)

Ilustración 22 KY-033 y su diagrama de pines

3.2 Librerías de Arduino

La programación del dispositivo final se realizó con la última versión del IDE Arduino, se usaron las siguientes librerías:

- Para el CubeCell Dev-Board Plus (HTCC-AB02):

  Se hace un `#include` de la librería LoRaWan_APP.h, esta se basa en la librería LoRaMac-node (stack desarrollado por Semtech) que a su vez utiliza LoRaMAC versión 4.6. La librería ofrece gran variedad de ejemplos para empezar a utilizar la tecnología LoRa/LoRaWAN.

- Para el DHT11:

  Para poder utilizar el sensor y acceder a las funciones que permiten la lectura de las magnitudes de temperatura y humedad relativa hay que instalar las librerías DHT_U.h, DHT.h y Adafruit_Sensor.h. Estas librerías pertenecen a Adafruit Unified Sensor, una biblioteca de software desarrollada por Adafruit Industries, la cual proporciona una interfaz unificada para gran variedad de sensores electrónicos.

- Para la pantalla OLED:

  Para la pantalla OLED es necesario instalar la librería 'HT_SH110Wire.h' que es específica para el controlador SH110.

  La instanciación de la librería se hace con objeto `display` que al estar ya declarado en la librería LoRaWan_APP, la declaración se hace tal que así:

  ```
  extern SH1107Wire display;
  ```

4 Desarrollo

4.1 Configuración del IDE Arduino

Después de descargar la última versión del IDE Arduino hay que instalar el framework relevante de CubeCell, se puede hacer vía Git, vía archivo local o directamente desde el propio IDE.

Para la tercera opción se siguen estos pasos:

1) Abrir el IDE Arduino y hacer click: *File* → *Preferences* → *Settings*. Aparece la siguiente ventana:

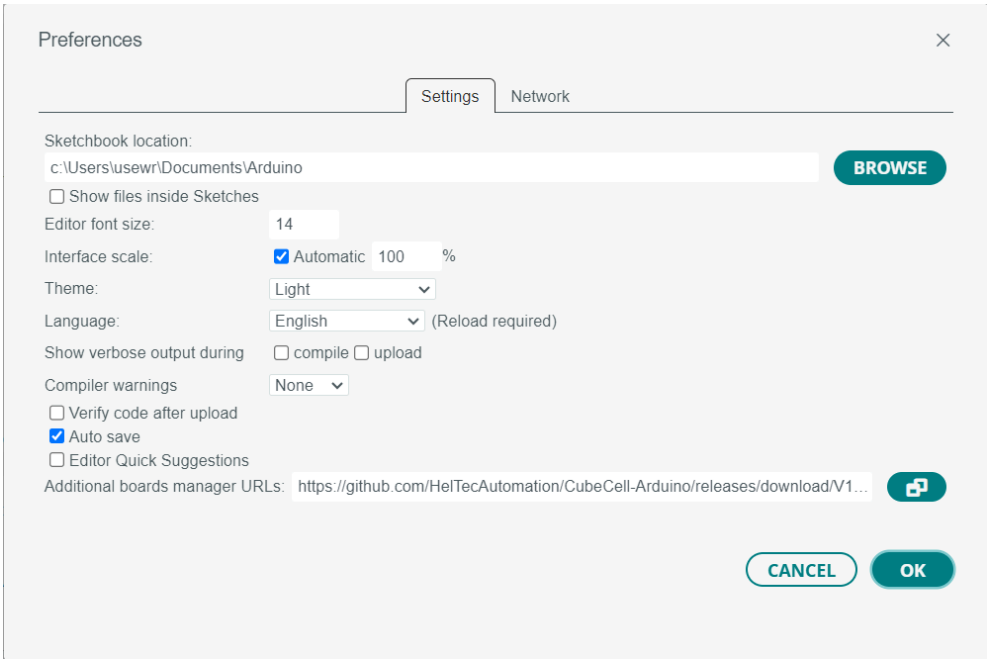


Ilustración 23 Ventana para instalar el framework de CubeCell

2) En la opción 'Additional boards manager URLs' hay que copiar el URL del siguiente json:

   https://github.com/HelTecAutomation/CubeCell-Arduino/releases/download/V1.5.0/package_CubeCell_index.json

3) Hacer click en *Tools* → *Board* → *Board Manager*… y en la barra de búsqueda escribir 'cube'. Seleccionar el framework e instalar la última versión.

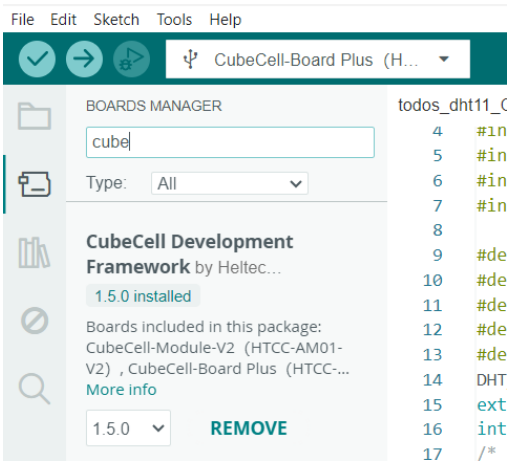


Ilustración 24 Ventana de instalación de librerías

4) Por último es necesario configurar los parámetros del menú Tools; donde se selecciona la placa y el puerto donde se conectará.

   También se especificará la clase del dispositivo, si el GUI que se usa es el que viene en la placa o es de creación propia, el modo de activación del dispositivo, la región de frecuencias en la que se usará y el tipo de mensaje de datos. Por último se activa la opción del Adapatative Data Rate (ADR), que es un mecanismo para optimizar el Data Rate (DR), ToA y el consumo energético de la red controlando parámetros de transmisión; en el resto de opciones se deja el valor por defecto.

   La configuración para este proyecto es la que se muestra a continuación:

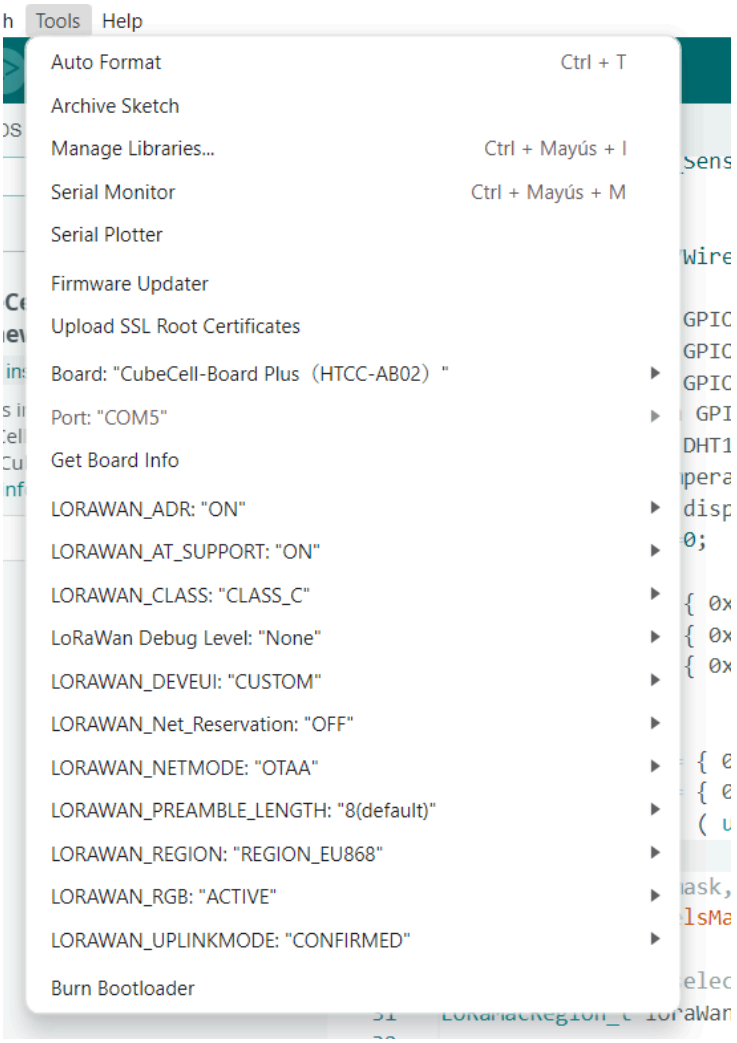


Ilustración 25 Configuración del menú Tools

4.2 Activación del nodo (vía OTAA)

Antes de poder registrar el dispositivo es necesario darse de alta en The Things Network:

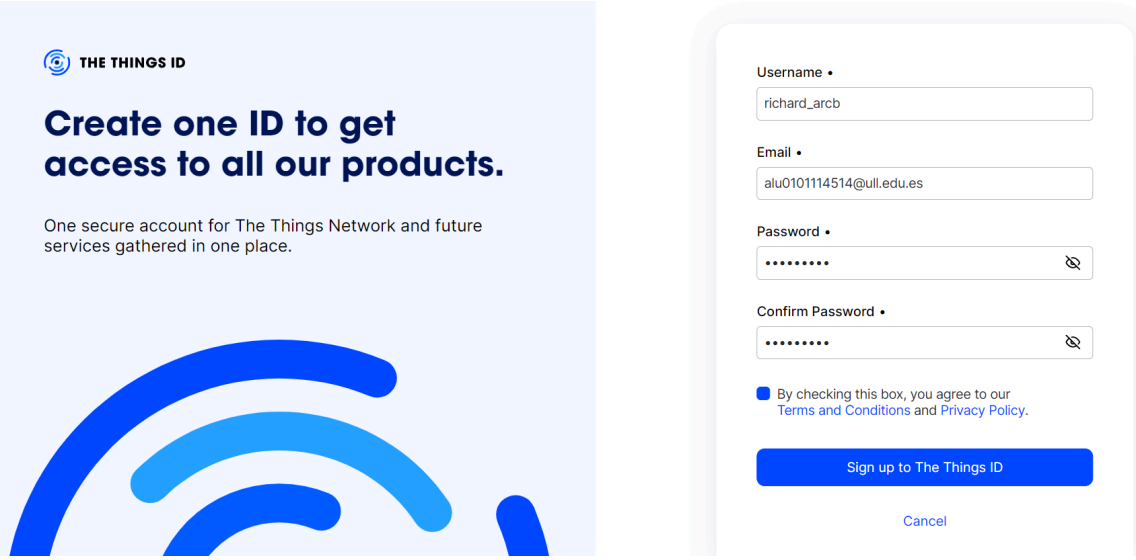


Ilustración 26 Crear cuenta en The Things Network

Hacer click en: *Usuario* → *Console* y después elegir país y el cluster de Europa.

También hay que crear una aplicación, hacer click en: *Go to applications* → *Create application*. Rellenar los campos:

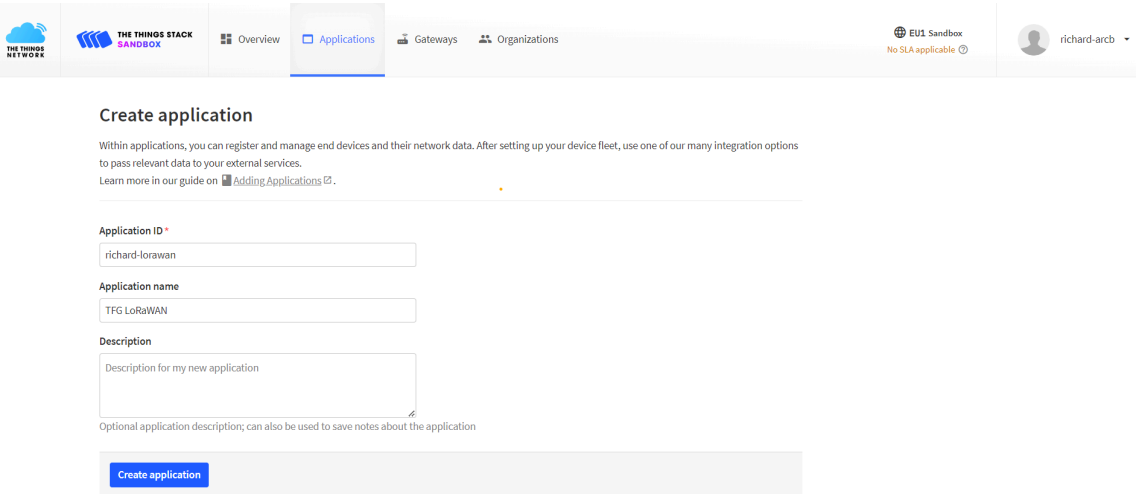


Ilustración 27 Ventana para crear aplicación en TTN

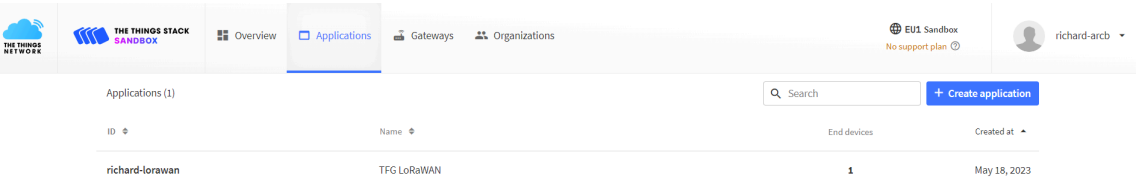


Ilustración 28 Aplicación creada

Una vez creada la aplicación se puede registrar el dispositivo, hacer click en la opción '+Register End Device'; para el registro necesitaremos un JoinEUI, para ello podemos usar un generador GUI online. Rellenar los campos; si se registra correctamente, TTN genera la AppEUI, DevEUI y la AppKey.

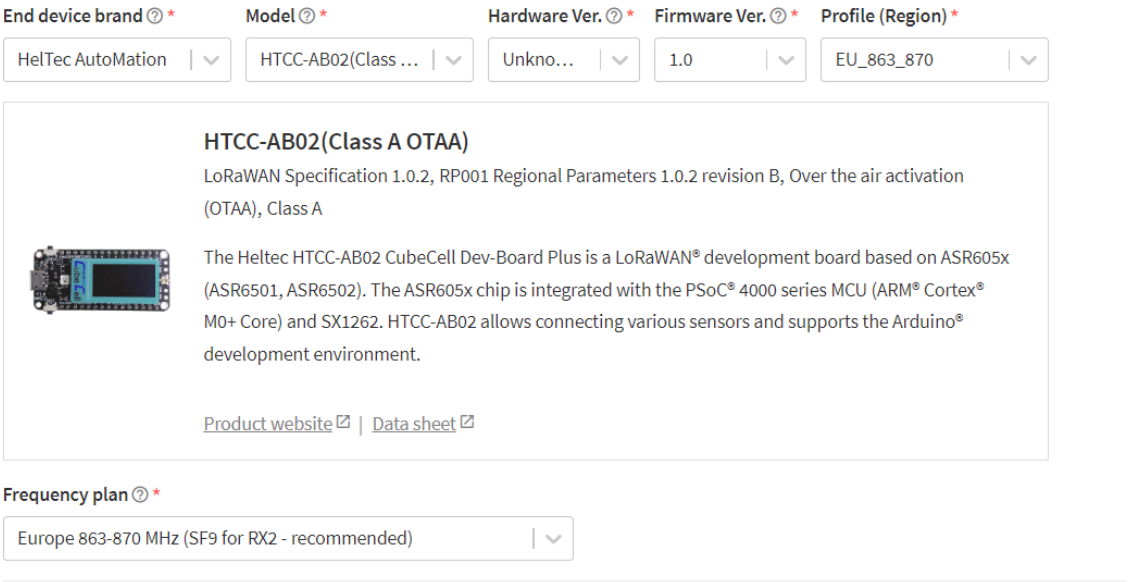


Ilustración 29 Nodo registrado en TTN

4.3 Programación del dispositivo final

Una vez completados los pasos anteriores se puede comenzar con la programación del nodo, el framework de CubeCell ofrece numerosos ejemplos para empezar a trabajar con la tecnología LoRa. En nuestro caso se partirá de un ejemplo para enviar datos de un sensor BH1750 (sensor que mide intensidad lumínica) hacia The Things Network, se modifica el código para cumplir con las necesidades del proyecto.

En los siguientes apartados se explicarán las secciones del código.

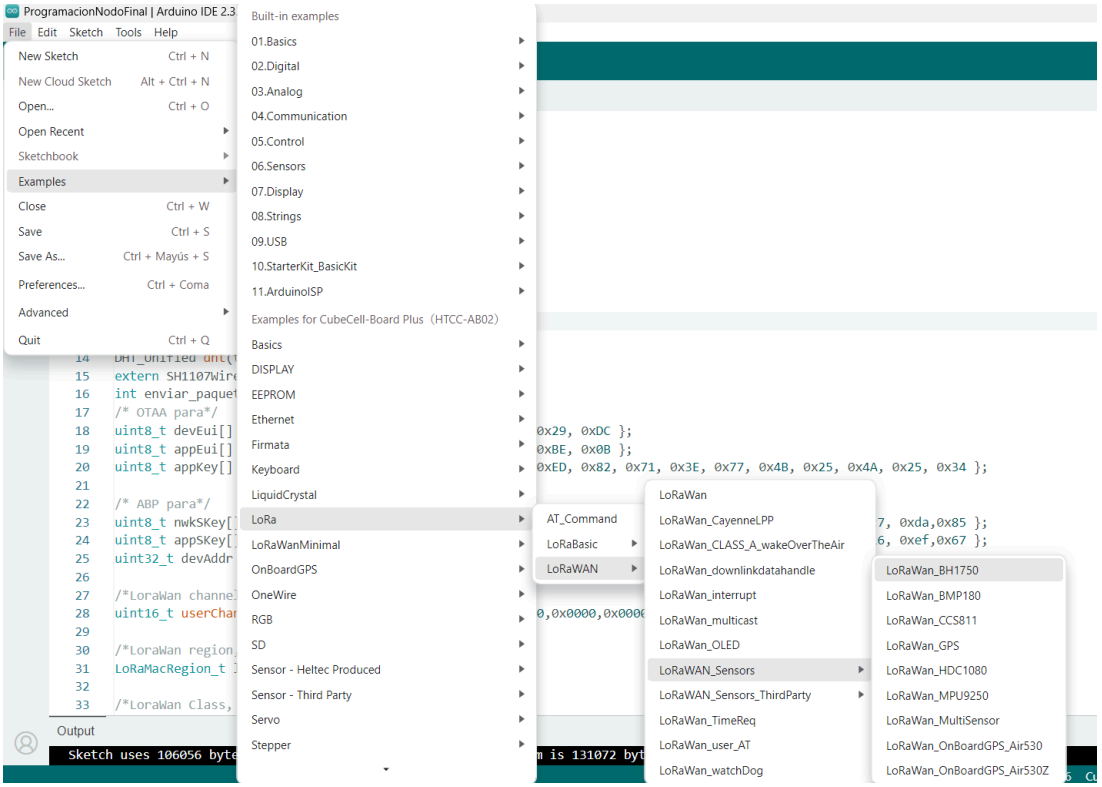


Ilustración 30 Ejemplos proporcionados por el framework de CubeCell

4.3.1.1 Instalación de librerías, definición de pines y activación del dispositivo

```
#include "LoRaWan_APP.h"
#include "Arduino.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include "HT_SH1107Wire.h"

#define touchMetal GPIO6
#define IrTracking GPIO12
#define sensorTilt GPIO2
#define temperatura GPIO4
#define DHTTYPE    DHT11
DHT_Unified dht(temperatura, DHTTYPE);
extern SH1107Wire  display;
int enviar_paquete=0;
/* OTAA para*/
uint8_t devEui[] = { 0x70, 0xB3, 0xD5, 0x7E, 0xD0, 0x06, 0x29, 0xDC };
uint8_t appEui[] = { 0x7B, 0xE6, 0x1B, 0x85, 0xE2, 0x94, 0xBE, 0x0B };
uint8_t appKey[] = { 0xC8, 0xA4, 0x7E, 0x00, 0x09, 0xF9, 0xED, 0x82, 0x71, 0x3E, 0x77, 0x4B, 0x25, 0x4A, 0x25, 0x34 };

```

Ilustración 31 Include de librerías, definición de pines y activación del nodo

Se comienza haciendo el `#include` de las librerías necesarias para poder usar la pantalla OLED, programar el DHT11 y poder mandar los datos al Servidor de Red. Se definen como constantes, con la directiva `#define`, los pines de la placa a los que se conectan los sensores, se instancian las clases DHT_Unified y SH1107Wire y se crea una variable global (`int enviar_paquete`), la cual se explicará con más detalle en los últimos apartados. Por último se completan los campos requeridos para la activación del dispositivo por el método OTAA.

4.3.1.2 Funciones del programa

4.3.1.2.1 Función para recoger datos, crear el paquete y visualizar las medidas por el Serial Monitor del IDE

```
/* Prepares the payload of the frame */
static void prepareTxFrame( uint8_t port )
{
  pinMode(touchMetal, INPUT); //Se pone el pin del sensor de tacto como entada
  pinMode(IrTracking, INPUT); //Se pone el pin del sensor de detección de obstáculos como entada
  pinMode(sensorTilt, INPUT); //Se pone el pin del sensor de inclinación como entada
  dht.begin();   //Se inicializa el sensor DHT11
  sensors_event_t event;

  dht.temperature().getEvent(&event);
  float temp=event.temperature; //Guardar valor de temperatura;
  dht.humidity().getEvent(&event);
  float hum=event.relative_humidity;//Guardar valor de humedad;
  int touchValue = digitalRead(touchMetal); // Leer el valor digital del sensor de tacto
  int tilt=digitalRead(sensorTilt); // Leer el valor digital del sensor de inclinación
  int IrTrackValue=digitalRead(IrTracking); // Leer el valor digital del sensor de detección de obstáculos

  if(!(isnan(hum)||isnan(temp))){
    int temp_entera=(int)( (temp*100));
    int temp_send_entera=temp_entera/100;//Valor entero de la temperatura que se enviará
    int temp_send_decimal=temp_entera%100;//Valor decimal de la temperatura que se enviará
    int hum_send=((int)(hum));
    /*Si la medida está fuera del rango de medida se envía los valores máximoy mínimos*/
    if(temp > 40.00 ){
      temp_send_entera=40;
    }else if (temp < 0.00) {
      temp_send_entera=0;
    }
```

Ilustración 32 Inicio de la función prepareTxFrame ()

Esta es la función más importante del programa ya que se programan los sensores y se construye el paquete de datos que enviará el dispositivo al servidor.

Comenzamos estableciendo como entrada el modo de los sensores digitales (Tilt switch, KY-036, KY-033) e inicializando el DHT11, para este último se crea un evento para

obtener las lecturas de humedad relativa y temperatura. Las medidas se guardan en las variables correspondientes.

Del DHT11 mediremos la temperatura en un rango acotado entre 0 y 40 ºC (líneas 24 a 28) ya que nos interesa medir la temperatura ambiente, si se superan estos límites se guarda el valor máximo o mínimo.

El resto de sensores, como tienen salida digital, basta con leer la salida del pin del dispositivo (GPIO) al que se conecta la salida de datos del sensor, usando la función `digitalRead()`

```
1  appDataSize = 3;
2      //Ponemos inicialmente todos los bytes a 0 para que no se produzcan errores
3      //Guardar valor del sensor de detección de obstáculos
4      appData[0]=appData[0] & 0x00;
5      appData[0]=appData[0]| (unsigned char)(IrTrackValue);
6      appData[0]=appData[0] << 1;
7      //Guardar valor del sensor de inclinación
8      appData[0]=appData[0]| (unsigned char)(tilt);
9      appData[0]=appData[0] << 6;
10     //Guardar la parte entera de la medida de temperatura
11     appData[0]=appData[0]| (unsigned char)(temp_send_entera);
12     //Guardar valor del sensor de tacto
13     appData[1]=appData[1] & 0x00;
14     appData[1]=appData[1]| (unsigned char)(touchValue);
15     appData[1]=appData[1] << 7;
16     //Guardar la parte decimal de la medida de temperatura
17     appData[1]=appData[1]| (unsigned char)(temp_send_decimal);
18     //Guardar valor de la medida de humedad
19     appData[2]=appData[2] & 0x00;
20     appData[2]=appData[2]| (unsigned char)(hum_send);
21
22     enviar_paquete=1;
```

Ilustración 33 Construcción del paquete de datos

Aunque el tamaño máximo que puede alcanzar la payload es 222 bytes, nos interesa compactar los datos de forma que ocupen el menor espacio de almacenamiento posible. Como se explicó en apartados anteriores, el ToA depende de factores como el SF y el tamaño de la payload, con gran esfuerzo se consiguió reducir el tamaño del paquete a tan solo 3 bytes.

Como los sensores digitales (Tilt Switch, KY-036 y KY-033) devuelven valores binarios, solo ocupan 3 bits dentro de la payload. La transformación hay que hacerla con los valores de temperatura y humedad ya que son de tipo float y tienen dos decimales de precisión.

Para transformar el dato de Temperatura (ºC) se siguió este procedimiento:

1- Multiplicar el valor de temperatura por 100 para obtener un valor entero.

2- Dividir el resultado por 100 y guardar el cociente (representa la parte entera de la medida) y el resto (representa la parte decimal de la medida) en variables distintas.

El valor máximo que alcanza la parte entera de la temperatura es 40 ºC y 40 se puede representar en el sistema binario con 6 bits ($2^6 = 64$), por otro lado la parte decimal alcanzaría un valor máximo de 0.99 ºC y 99 se puede representar en binario con 7 bits ($2^7 = 125$).

La medida de Humedad Relativa (%) alcanzaría un valor máximo de 100 y se sigue el mismo procedimiento que con el dato de temperatura solo que los decimales siempre tienen valor .00, por lo que nos interesa la parte entera solamente.

Los datos se guardarán en un `array` de bytes que se inicializa con tamaño 3. Se guardan los datos empezando por el elemento 0 del `array` y se emplean operaciones bit a bit para ir desplazando los datos y ocupar todo el byte; un `AND 0x00` para asegurarnos de que cada byte está libre, << para desplazar bits hacia la izquierda y también se realizan operaciones `OR` para guardar los datos sin borrar los que ya estaban almacenados.

Se le da un valor a la variable `enviar_paquete`, esto indica que se ha construido el paquete correctamente con todas las medidas y que está listo para enviarse en un mensaje uplink. En la figura siguiente se muestra su estructura.

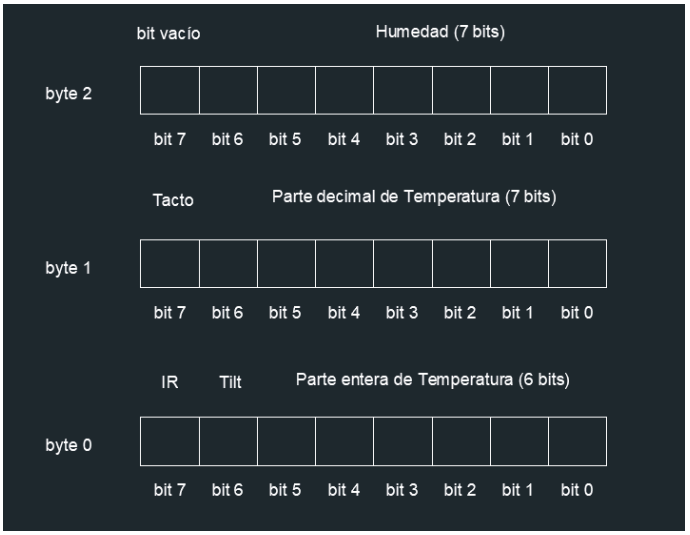


Ilustración 34 Estructura de la payload

```
Serial.println("\n\nLecturas sensores:\n");
    Serial.println("Lectura sensor de Temperatura y Humedad:");
    if (isnan(temp)) {
      Serial.println(F("Error al leer la temperatura!"));
   }
    else {
      Serial.print(F("Temperatura: "));
      Serial.print(temp);
      Serial.println(F("°C"));
    }
    if (isnan(hum)) {
      Serial.println(F("Error al leer la humedad!"));
    }
    else {
      Serial.print(F("Humedad: "));
      Serial.print(hum);
      Serial.println(F("%"));
    }
Serial.println("\n\n");
    display.clear();
    drawTextAlignmentDemo(temp,hum,tilt,touchValue,IrTrackValue);
    display.setTextAlignment(TEXT_ALIGN_RIGHT);
    display.drawString(10, 128, String(millis()));

    display.display();
    delay(appTxDutyCycle); // Tiempo de espera entre la toma de medidas
  }
```

Ilustración 35 Final de la función prepareTxFrame()

Por último se visualizan por el Serial Monitor del IDE todos los datos que van recogiendo los sensores y se especifica el tiempo de la toma de medidas, también se llama la función que los visualiza por la pantalla OLED.

```
Lecturas sensores:

Lectura sensor de Temperatura y Humedad:
Temperatura: 22.60°C
Humedad: 59.00%
Lectura sensor de inclinacion Tilt:
  No Inclinado
Lectura sensor de IR Tracking:
  Libre
Lectura sensor de tacto metalico:
  No hay contacto!
```

Ilustración 36 Visualización por el Serial Monitor

4.3.1.2.2 Función para visualizar datos por la pantalla OLED

```
1  //Mostrar lecturas en display OLED
2  void drawTextAlignmentDemo(float t, float h, int incl, int tacto, int obs) {
3
4    char str[30];
5    int x = 0;
6    int y = 0;
7
8    if (isnan(t) || isnan(h)) {
9      // The coordinates define the center of the text
10     display.setTextAlignment(TEXT_ALIGN_CENTER);
11     x = display.width()/3;
12     y = 0;
13     sprintf(str,"Lectura fallida ");
14     display.drawString(x, y, str);
15
16   }else{
17     // Mostrar Temperatura(ºC)
18     display.setTextAlignment(TEXT_ALIGN_CENTER);
19     x = display.width()/3;
20     y = 0;
21     sprintf(str,"Temp(ºC): %0.2f",t);
22     display.drawString(x, y, str);
23
24     // Mostrar Humedad Relativa(%)
25     display.setTextAlignment(TEXT_ALIGN_CENTER);
26     x = display.width()/3;
27     y = display.height()/5;
28     sprintf(str,"Hum.Rel: %0.2f",h);
29     display.drawString(x, y, str);
```

Ilustración 37 Inicio y definición de la función de visualización por pantalla OLED

Este código se adaptó de uno de los ejemplos disponibles que proporciona el framework para sacar datos por la pantalla que viene junto con la placa, teniendo en cuenta el espacio disponible se fijarán coordenadas distintas para las 5 medidas y el formato de salida de las lecturas del DHT11(dos decimales). Cada vez que se recogen datos nuevos se actualiza el display, los resultados se observan en la siguiente figura.

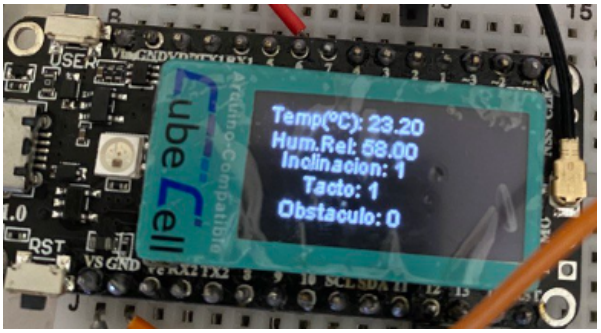


Ilustración 38 Visualización de datos

4.3.1.2.3 Función `setup()`

```
1  void setup() {
2
3    display.init();
4    display.setFont(ArialMT_Plain_10);
5    display.clear();
6
7    display.setTextAlignment(TEXT_ALIGN_CENTER);
8    int x0 = display.width()/3;
9    int y0 = 0;
10   display.drawString(x0, y0, "Display Iniciado");
11   display.display();
12
13   Serial.begin(115200);
14 #if(AT_SUPPORT)
15   enableAt();
16 #endif
17   deviceState = DEVICE_STATE_INIT;
18   LoRaWAN.ifskipjoin();
19 }
```

Ilustración 39 Función setup()

Esta es la primera función que se llama al ejecutar el programa, se suele usar para inicializar variables, modos de pines, etc. En nuestro caso se inicializa el display y la tasa de baudios.

También, si está habilitado el AT_SUPPORT, se pasa al primer `case` definido en la función `loop()`, el cual imprime los parámetros del dispositivo LoRa e inicializa el stack LoRaWAN con la clase y región especificada.

4.3.1.2.4 Función `loop()`

```
void loop() {

  switch( deviceState )
  {
    case DEVICE_STATE_INIT:
    {
#if(LORAWAN_DEVEUI_AUTO)
      LoRaWAN.generateDeveuiByChipID();
#endif
#if(AT_SUPPORT)
      getDevParam();
#endif
      printDevParam();
      LoRaWAN.init(loraWanClass,loraWanRegion);
      deviceState = DEVICE_STATE_JOIN;
      break;
    }
    case DEVICE_STATE_JOIN:
    {
      LoRaWAN.join();
      break;
    }
    case DEVICE_STATE_SEND:
    {
        prepareTxFrame( appPort );
        if (enviar_paquete==1){
          LoRaWAN.send();
        }
        deviceState = DEVICE_STATE_CYCLE;
        break;
    }
    case DEVICE_STATE_CYCLE:
    {
      // Schedule next packet transmission
      //El tiempo de espera de envío de cada paquetes tiene que ser de almenos 10 segundos
      txDutyCycleTime = appTxDutyCycle;
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      break;
    }
    case DEVICE_STATE_SLEEP:
    {
      LoRaWAN.sleep();
      break;
    }
    default:
    {
      deviceState = DEVICE_STATE_INIT;
      break;
    }
  }
}
```

Ilustración 40 Función loop()

En este bloque de código se define el dispositivo LoRa como una máquina finita de estados (FMS), en el estado `DEVICE_STATE_SEND` se hace la llamada de la función `prepareTxFrame()` y se envían los paquetes individuales cada `txDutyCycleTime`, que equivale a 10 segundos.

Este tiempo también se usa como tiempo de muestreo para que los sensores tomen medidas, `delay(10000);` y es el tiempo que tarda el dispositivo en la transmisión del paquete. Si el tiempo de toma de medidas fuese menor o mayor que 10 segundos se perderían algunos paquetes de datos ya que no se sincroniza con el intstante de tiempo en el que el dispositivo empieza la transmisión uplink. Destacar que el dispositivo entra al estado de enviar paquetes solo si se ha cambiado el valor de `enviar_paquete` a 1.

El ToA se puede calcular usando la calculadora online proporcionada por TTN [21] y además de los parámetros de transmisión también depende del porcentaje de uso justo que obliga la normativa (DC); para SP=8, CR=4/5, BW =125 KHz el resultado es 9.27, los datos se obtuvieron del JSON de uno de los mensajes uplinks accesibles desde la ventana de Live Data de TTN.

4.3.2 Payload Formatters

Los formatters permiten procesar los datos que van desde y hacia los dispositivos finales. Como la payload está en formato binario hay convertirla en campos legibles por humanos, la conversión se puede hacer sobre una aplicación entera pero para este proyecto se hizo sobre el nodo registrado en TTN.

Tipos de Payload Formatters:

1) JavaScript: el payload formatter se escribe en este lenguaje.

2) CayeneLPP #: el servidor puede decodificar automáticamente las payloads que están en este formato sin necesidad de implementar código nuevo.

3) Repositorio: los fabricantes de los dispositivos finales proveen de payload formatters específicos para sus dispositivos, se encuentran disponibles en The Things Stack Repository.

Para el proyecto se escribió el payload formatter en JavaScript ya que permite crear funciones capaces de codificar y decodificar los mensajes LoRa, en este caso uplink. Estas funciones se ejecutan usando el runtime JavaScript ECMAScript 5.1.

Cuando se recibe un mensaje uplink del dispositivo, se llama la función `decodeUplink()` para decodificar el mensaje, convierte la payload binaria en un objeto JSON legible por humanos que se envía hacia la aplicación vía upstream.

La función `decodeUplink()` recibe un objeto `input` y devuelve un objeto `output`. El objeto `input` contiene un `struct` cuyos campos son el fPort con su valor correspondiente y un `array` de bytes que representa la payload codificada. En el `output` están presentes posibles warnings y errores que se pudieran definir en el formatter además de otro `struct` con la payload decodificada.

```
function decodeUplink(input) {
  // input has the following structure:
  // {
  //   "bytes": [1, 2, 3], // FRMPayload (byte array)
  //   "fPort": 1
  // }
  return {
    data: {
      bytes: input.bytes,
    },
    warnings: ["warning 1", "warning 2"], // optional
    errors: ["error 1", "error 2"], // optional (if set, the decoding failed)
  };
}
```

Ilustración 41 Estructura de la función decodeUplink()

Dentro de la aplicación creada en TTN, en la ventana de End Devices se puede testear el payload formatter. Basta con copiar los bytes de la payload que aparecene en el mensaje uplink y la payload decodificada se muestra en la ventana Decoded test payload.

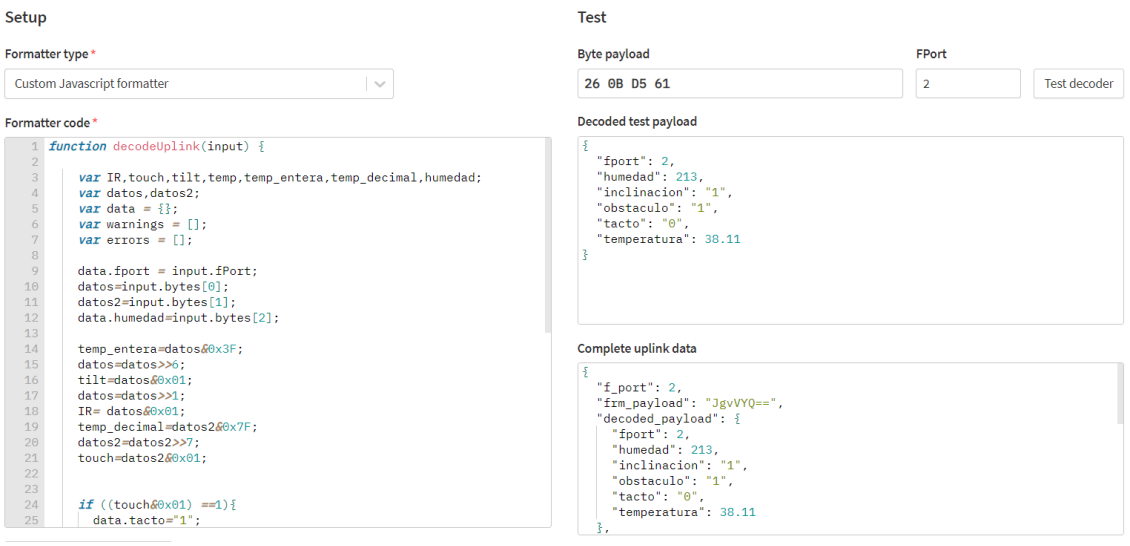


Ilustración 42 Payload decodificada

```
function decodeUplink(input) {

    var IR,touch,tilt,temp,temp_entera,temp_decimal,humedad,datos,datos2;
    var data = {};
    var warnings = [];
    var errors = [];

    data.fport = input.fPort;
    datos=input.bytes[0];
    datos2=input.bytes[1];
    data.humedad=input.bytes[2];

    temp_entera=datos&0x3F;
    datos=datos>>6;
    tilt=datos&0x01;
    datos=datos>>1;
    IR= datos&0x01;
    temp_decimal=datos2&0x7F;
    datos2=datos2>>7;
    touch=datos2&0x01;

    if ((touch&0x01) ==1){
      data.tacto="1";
    }else{
      data.tacto="0";
    }
    if ((IR&0x01) ==1){
      data.obstaculo="0 ";
    }else{
      data.obstaculo="1";
    }
    if ((tilt&0x01) ==1){
      data.inclinacion="0";
    }else{
      data.inclinacion="1";
    }
    temp=((temp_entera*100)+temp_decimal)/100;
    data.temperatura=temp;

    return {
      data: data,
      warnings: warnings,
      errors: errors
    };
}

```

Ilustración 43 Payload Formatter

La medidas se devuelve en el `return` de la función en la variable `data`, es un `struct` que contendrá los siguientes campos: “fport”, “humedad”, “inclinación”, ”obstaculo”, “tacto” y “temperatura”.

Para recuperar los datos del `array bytes` hay que recordar la construcción de la payload, en el último byte se guardó la humedad (ocupando 7 bits) y el valor no se transformó. Los bits de los sensores digitales se guardaron 2 en la parte ALTA del byte 0 y 1 en el bit más significativo del byte1 y finalmente la parte entera de la temperatura se almacena en los bits restantes del byte 0 y la parte decimal en el byte 1. Para acceder a los datos, de nuevo se emplean operaciones bit a bit y se van almacenando en las variables correspondientes para luego guardarlos dentro de sus respectivos campos dentro de `data`, en el caso de los valores binarios lo que se guarda depende del resultado de la operación `AND 0x01`.

4.3.3 Suscripción a mensajes uplink

Hay que conectarse al servidor de red (The Things Stack) para tener acceso a los mensajes que envía el dispositivo final con los datos de los sensores y para ello utilizaremos el protocolo de comunicación MQTT por su comodidad y eficiencia. The Things Stack sólo implementa la versión estándar MQTT 3.1.1 y con QoS 0.

Para poder usar el servidor MQTT es necesario crear una API Key para la autentificación. Dentro de la aplicación creada en TTN, hacer click en Integrations → MQTT → Generate new API Key. Es necesario hacer copia del valor del API Key ya que después dejará de ser visible por razones de seguridad.

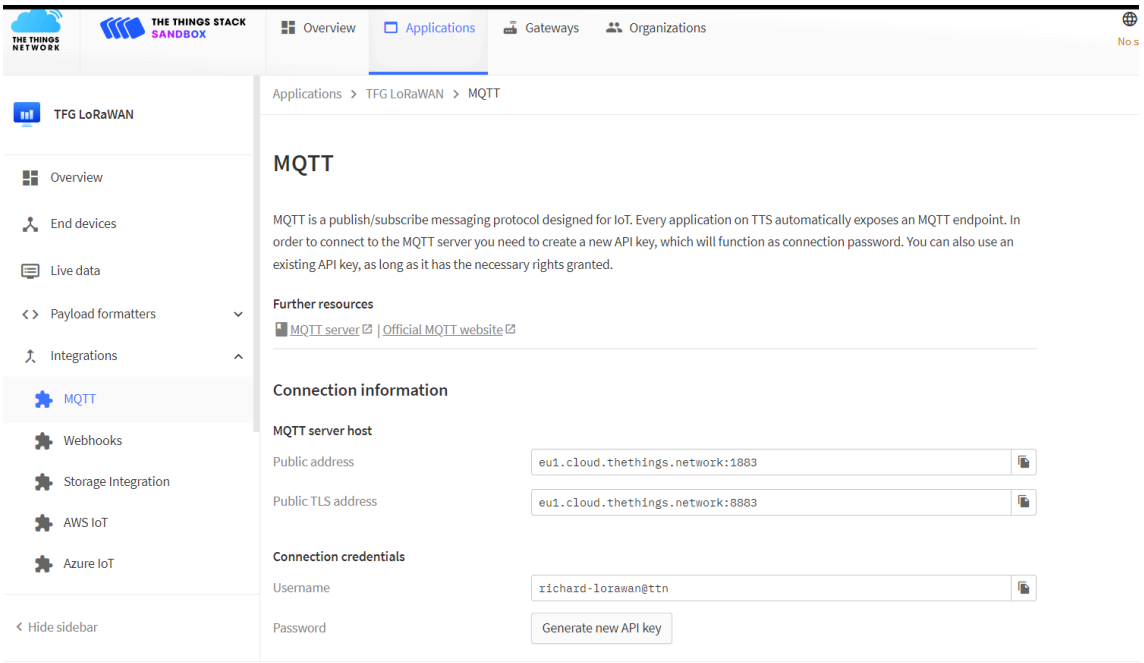


Ilustración 44 Creación del API Key

El servidor de aplicación publica tráfico uplink en varios tópicos:

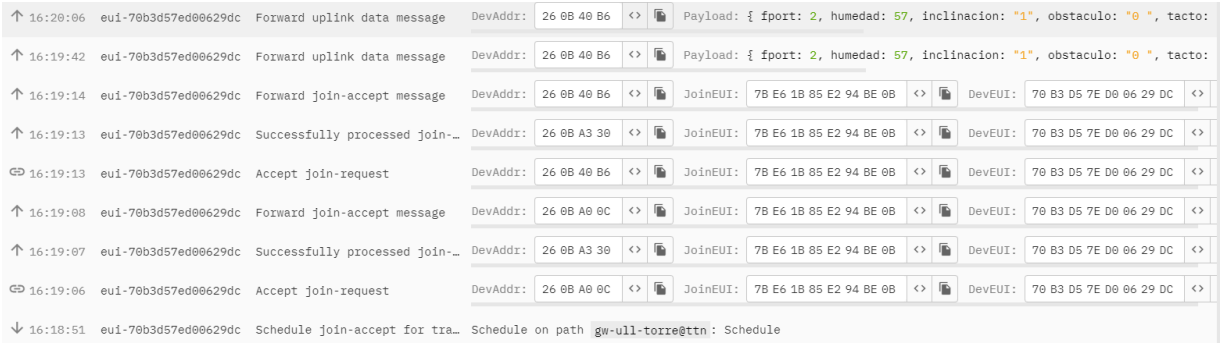


Ilustración 45 Live Data de TTN

Como solo nos interesan los mensajes que contienen los datos de los sensores, hay que suscribirse al tema correspondiente. Los tópicos tienen un formato definido por TTN:

- `v3/{application id}@{tenant id}/devices/{device id}/join`
- `v3/{application id}@{tenant id}/devices/{device id}/up`
- `v3/{application id}@{tenant id}/devices/{device id}/down/queued`
- `v3/{application id}@{tenant id}/devices/{device id}/down/sent`
- `v3/{application id}@{tenant id}/devices/{device id}/down/ack`
- `v3/{application id}@{tenant id}/devices/{device id}/down/nack`
- `v3/{application id}@{tenant id}/devices/{device id}/down/failed`
- `v3/{application id}@{tenant id}/devices/{device id}/service/data`
- `v3/{application id}@{tenant id}/devices/{device id}/location/solved`

Ilustración 46 Tópicos disponibles

Los campos se rellenan con la información disponible de la aplicación creada desde TTN, nos interesa saber:

- Application id @ tenant id: corresponde con el nombre de usuario @ttn: richard-lorawan@ttn.
- Device id : eui-70b3d57ed00629dc
- API KEY : no se pone en la memoria por razones de seguridad.
- Dirección pública del host del servidor MQTT: eu1.cloud.thethings.network:1883

El tópico a suscribirse es: v3/richard-lorawan@ttn/devices/eui-70b3d57ed00629dc/up

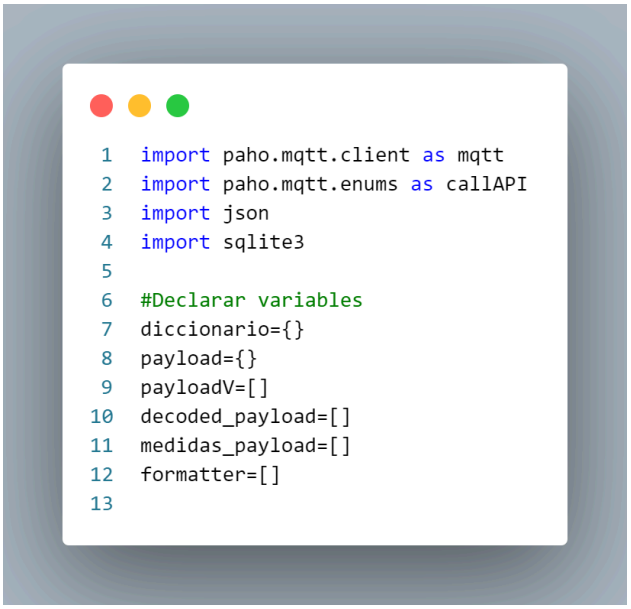
4.4 Recopilar y almacenar los datos

Al tener algo de experiencia con Python decidí desarrollar la parte final del Servidor de Aplicaciones en este lenguaje. La programación se hizo en el IDE Visual Studio Code (VSCode), ya que es gratuito y open source.

Se escribió un script (`conectarTTN_to_BD.py`) para acceder a la payload decodificada de los mensajes uplink, para construir una base de datos SQLite con las medidas registradas.

Se descarga la última versión del IDE Visual Studio Code (VSCode), como el API también se hará en Python es recomendable crear un entorno virtual. Un entorno virtual es como una instalación aislada de Python que permite descargar todas las librerías y dependencias necesarias en una sola localización, sin que se produzca conflicto con otros proyectos o con el nº de las versiones.

4.4.1 Instalación de librerías



```
1  import paho.mqtt.client as mqtt
2  import paho.mqtt.enums as callAPI
3  import json
4  import sqlite3
5
6  #Declarar variables
7  diccionario={}
8  payload={}
9  payloadV=[]
10 decoded_payload=[]
11 medidas_payload=[]
12 formatter=[]
13
```

Ilustración 47 Librerías importadas y variables

La librería MQTT a usar es Eclipse Paho MQTT Python Client[23], que implementa las versiones 5.0, 3.1.1 y 3.1 del protocolo MQTT y versiones de Python 3.7+. La librería fue creada por la fundación Eclipse y usa Mosquitto como broker open source.

La librería JSON [27]se usa para la conversión y manipulación de los datos que están en este formato, que en nuestro caso sería la payload decodificada. JSON es un acrónimo para JavaScript Object Notation y es un formato de texto sencillo para el intercambio de datos.

Por último la librería SQLite3 [24] se utiliza para interactuar con bases de datos SQLite, las cuales son bases de datos relacionales, ligeras y autónomas ya que no requieren un servidor de bases de datos completos para operar. Se suele emplear en el desarrollo de

aplicaciones pequeñas y medianas y en el desarrollo de pruebas antes de implementar bases de datos más grandes.

Para instalar estas librerías en el entorno virtual, en el IDE se abre un terminal y se escribe el comando `pip install` seguido del nombre de la librería:

```
pip install paho-mqtt
```

```
pip install json
```

```
pip install sqlite 3
```

En esta parte del código también se definen las variables que usaremos después.

4.4.2 Conexión con el servidor MQTT, crear cliente, suscripción al tópico

```
#Parametros:
mqttClient_id='richardTFG'
host_ttn='eu1.cloud.thethings.network'
APIver=callAPI.CallbackAPIVersion(2)
topic_ttn='v3/richard-lorawan@ttn/devices/eui-70b3d57ed00629dc/up'

#Crear instancia cliente
mqtt_cliente=mqtt.Client(APIver,mqttClient_id)

mqtt_cliente.username_pw_set(
    username="richard-lorawan@ttn",
    password="NNSXS.4RVWUUBW2W2JX6LNPBWZLWDQT3OGW47ZBMPGRKI.I65Y2HD3WJ3L6S3PHVH6TWPFEKACLMYYCCWRVL4Q5IIO55TVJP7A"
)
```

Ilustración 48 Parámetros para conectarse con el broker e instanciar el cliente MQTT

Primero se definen los parámetros de conexión: identificador del cliente que puede ser cualquier nombre , la dirección pública del host del servidor MQTT, la versión del API y por último el tópico a suscribirse.

Para definir el cliente se crea un objeto de la clase Client pasándole la versión del API y client_id id, pasar la autentificación del broker requiere usar el método `username_pw_set()` que tiene 2 parámetros: username que será application id @ tenant y una contraseña que será el API Key. Por último nos conectamos al broker ,con el método `connect()`, pasándole la dirección IP.

```python
def on_subscribe(client, userdata, mid, reason_code_list, properties):
    print("Suscripción exitosa al tema:", topic_ttn)

def se_desconecta(client, userdata, disconnect_flags, reason_code, properties):
    print('Se ha perdido la conexion con el broker\n')

def on_message(client, userdata, msg):
```

Ilustración 49 Definición del Callback MQTT

Hay que definir una serie de callbacks para que se ejecute el programa con la llegada de cada mensaje uplink, en la librería Paho MQTT los callbacks son funciones que se llaman automáticamente en respuesta a ciertos eventos como la llegada de un mensaje, la conexión al broker o la pérdida de conexión. Se definieron 3:

1) `on_subscribe()`: se llama automáticamente cuando se produce la suscripción al tópico, en este caso la llegada de los mensajes uplink al servidor.

2) `se_desconecta()`: avisa si se pierde la conexión con el broker.

3) `on_message()`: se llama cada vez que se publica un mensaje en el tema suscrito. Esta función se explicará con mayor detalle más adelante.

```python
#Subscribirme a los topics y conexion con la base de datos
conexion = conexion_DB()

mqtt_cliente.on_subscribe = on_subscribe
mqtt_cliente.on_message = on_message
mqtt_cliente.on_disconnect = se_desconecta
mqtt_cliente.subscribe(topic=topic_ttn)

mqtt_cliente.loop_forever()
```

Ilustración 50 Pasar a los atributos del objeto Cliente las direcciones de las callbacks

'on_subscribe', 'on_message' y 'on_disconnect' son todos atributos de la clase Cliente y el valor que se les da es el puntero a la función callback correspondiente. Por último

se usa el método loop_forever al final del código, este método llama a las funciones de red en un bucle infinito, así se ejecuta constantemente el cliente MQTT.

4.4.3 Creación de la base de datos SQLite

Los datos recopilados se guardarán en una base de datos, en este caso de SQLlite. En un fichero Python (`DB.py`) aparte, se creará una tabla por cada una de las 5 magnitudes que miden los sensores.

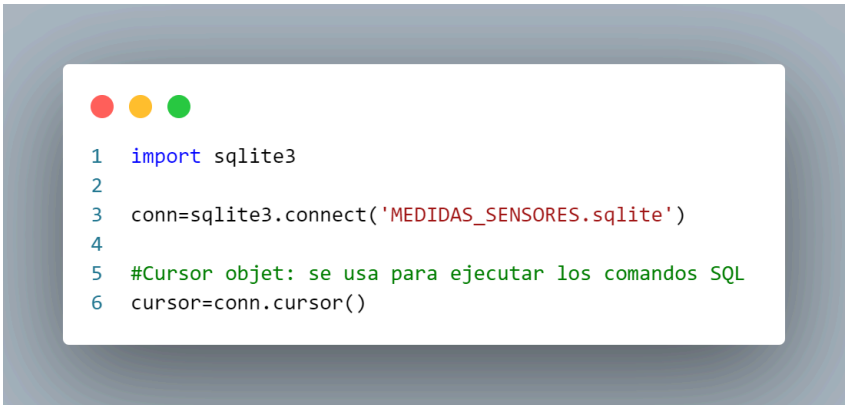


Ilustración 51 Conexión con la base de datos

Se crea un objeto de la clase sqlite3.Connection para establecer conexión con la base de datos, que generará el archivo 'MEDIDAS_SENSORES.sqlite'

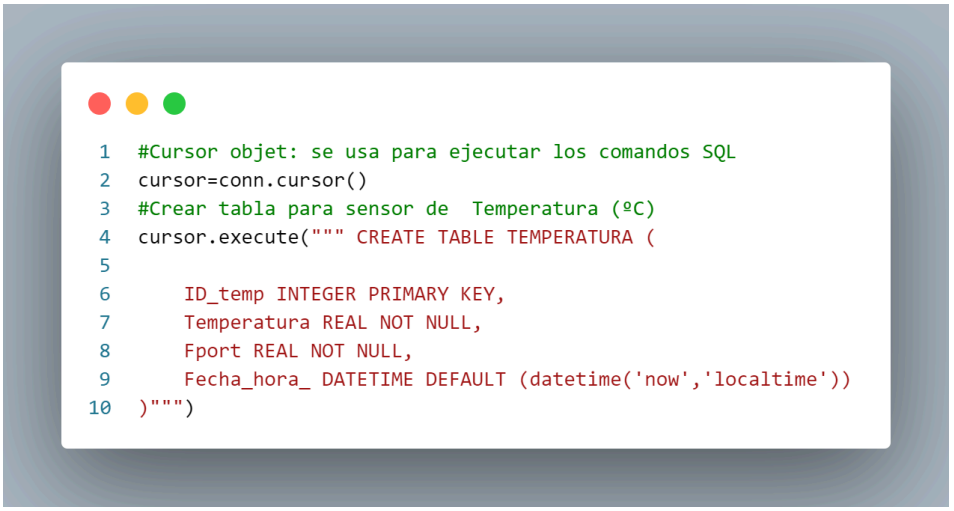


Ilustración 52 Creación de la Tabla de Temperatura

Para ejecutar los comandos SQL necesitamos un objeto cursor y con el método `execute()` creamos las tablas usando el comando `CREATE TABLE`. Cada tabla consta de 4 columnas: la llave primaria que identifica de forma inequívoca cada una de las filas y será de tipo entero, la columna del Fport y de la Magnitud del sensor en las que se impone la restricción de que no serán campos vacíos y por último una columna

que indica la fecha y hora en la que se guarda la medida. El tipo de la columna Magnitud será de tipo `REAL` para Temperatura y Humedad Relativa y de tipo `BOOL` para el resto de medidas.

Después de crear todas las tablas se finaliza la transacción, con `con.comit()`, para asegurarnos que los cambios generados se guardan de forma permanente y para finalizar se corta la conexión con la base de datos.

Es necesario obtener la payload para guardar las medidas de los sensores en el base de datos, por comodidad se define la función `conexion_BD()` que devuelve una conexión con la base de datos que se va a utilizar. Esta función se llama dentro del callback on_message de tal forma que cada vez que su publica un mensaje uplink, se inicia la conexión con la base de datos. Uno de los campos del mensaje uplink que devuelve el callback, es la payload como un JSON. El JSON devuelto tiene una estructura anidada un poco compleja como se muestra en la figura siguiente:

{'end_device_ids': {'device_id': 'eui-70b3d57ed00629dc', 'application_ids': {'application_id': 'richard-lorawan'}, 'dev_eui': '70B3D57ED00629DC', 'join_eui': '7BE61B85E294BE0B', 'dev_addr': '260B6FAC'
}, 'correlation_ids': ['gs:uplink:01HY1EE3ZN6PN636FS4798HNHH'], 'received_at': '2024-05-16T19:35:32.290384051Z', 'uplink_message': {'session_key_id': 'AYHC5oCI9SpKc6oqURL92g==', 'f_port': 2, 'frm_paylo
ad': 'lx44', 'decoded_payload': {'fport': 2, 'humedad': 56, 'inclinacion': '1', 'obstaculo': '0 ', 'tacto': '0', 'temperatura': 23.3}, 'rx_metadata': [{'gateway_ids': {'gateway_id': 'gw-ull-torre', 'eu
i': '60C5A8FFFE761458'}, 'time': '2024-05-16T19:35:32.014897108Z', 'timestamp': 3496440659, 'rssi': -106, 'channel_rssi': -106, 'snr': 4.5, 'location': {'latitude': 28.48173218, 'longitude': -16.317579
24, 'altitude': 580, 'source': 'SOURCE_REGISTRY'}, 'uplink_token': 'ChoKGAoMZ3ctdWxsLXRvcnJlEghgxaj//nYUWBDT5p2DDRoLCITAmbIGELT99CgguPjpoOHZBA==', 'received_at': '2024-05-16T19:35:32.052896061Z'}], 'se
ttings': {'data_rate': {'lora': {'bandwidth': 125000, 'spreading_factor': 7, 'coding_rate': '4/5'}}, 'frequency': '867300000', 'timestamp': 3496440659, 'time': '2024-05-16T19:35:32.014897108Z'}, 'receiv
ed_at': '2024-05-16T19:35:32.086564554Z', 'confirmed': True, 'consumed_airtime': '0.051456s', 'version_ids': {'brand_id': 'heltec', 'model_id': 'cubecell-dev-board-plus-class-a-otaa', 'hardware_versio
n': '_unknown_hw_version_', 'firmware_version': '1.0', 'band_id': 'EU_863_870'}, 'network_ids': {'net_id': '000013', 'ns_id': 'EC656E0000000181', 'tenant_id': 'ttn', 'cluster_id': 'eu1', 'cluster_addre
ss': 'eu1.cloud.thethings.network'}}}

Ilustración 53 Payload en formato JSON

```python
def on_message(client, userdata, msg):

    #Subscribirme a los topics
    conexion = conexion_DB()
    diccionario=msg.payload
    diccionario=diccionario.decode('latin-1')
    payload=json.loads(diccionario)
    payloadV.append(list(payload.values()))
    decoded_payload.append(payloadV[-1][-1])
    medidas_payload.append(decoded_payload[-1]['decoded_payload'])
    formatter.append(medidas_payload[-1])
    print('{:>4}{:>14}{:>24}{:>28}{:>24}{:>18}\n'.format(formatter[-1]['humedad'],
            formatter[-1]['temperatura'],formatter[-1]['tacto'],formatter[-1]['obstaculo'],
            formatter[-1]['inclinacion'],formatter[-1]['fport']))

    for z in formatter:

            new_fport = z['fport']
            new_humedad=z['humedad']
            new_inclinacion=z['inclinacion']
            new_obstaculo=z['obstaculo']
            new_tacto=z['tacto']
            new_temperatura=z['temperatura']
```

Ilustración 54 Código para parsear el JSON

Lo mejor es parsearlo para quedarnos con el campo que nos interesa, la payload decodificada. Primero lo convertimos en diccionario de Python, después en una lista y por último la guardamos en una lista llamada `formatter` cuyos elementos son diccionarios y contiene las magnitudes de la payload con su valor correspondiente. El resultado se observa en la siguiente figura.

```
[{'fport': 2, 'humedad': 60, 'inclinacion': '1', 'obstaculo': '0 ', 'tacto': '0', 'temperatura': 22}]
```

Ilustración 55 Vista de la payload decodificada

```
cursor=conexion.cursor()
    sql_temp= """ INSERT INTO TEMPERATURA (Temperatura,Fport)
            VALUES(?,?)"""
    sql_hum= """ INSERT INTO HUMEDAD (Humedad,Fport)
            VALUES(?,?)"""
    sql_incl= """ INSERT INTO INCLINACION (Inclinacion,Fport)
            VALUES(?,?)"""
    sql_obs= """ INSERT INTO OBSTACULO (Obstaculo,Fport)
            VALUES(?,?)"""
    sql_tacto= """ INSERT INTO TACTO (Tacto,Fport)
            VALUES(?,?)"""
    try:
        cursor=cursor.execute(sql_temp,(new_temperatura,new_fport))
        cursor=cursor.execute(sql_hum,(new_humedad,new_fport))
        cursor=cursor.execute(sql_incl,(new_inclinacion,new_fport))
        cursor=cursor.execute(sql_obs,(new_obstaculo,new_fport))
        cursor=cursor.execute(sql_tacto,(new_tacto,new_fport))
        conexion.commit()
    except sqlite3.Error as e:
        print("Error al insertar en la base de datos:", e)
        return "Error al insertar en la base de datos", 500
    cursor=conexion.close()
```

Ilustración 56 Insertar los valores recogidos en las tablas correspondientes

Por último solo falta insertar los valores recibidos en las tablas correspondientes usando el comando `INSERT INTO`, de esta forma todos los datos registrados por los sensores y que son enviados por el nodo hacia el servidor de red quedan guardados en una base de datos, la cual utilizaremos en el desarrollo de la aplicación web.

4.5 Desarrollo de Aplicación Web

Ésta corresponde con la última parte del Servidor de Aplicación, una vez que se han guardado los datos es conveniente visualizarlos. Aunque es cierto que las medidas instantáneas se pueden ver a través de la pantalla OLED de la placa, sería más útil poder observar el registro histórico de los datos tomados y la fecha y hora en el que se guardó la medida en la base de datos.

Para ello creamos un programa (`Visualizacion.py`) cuya función consiste en consultar las tablas de la base de datos y crea gráficas de cada una de ellas separadas en distintas tabs. Para dar un poco de interactividad a la aplicación las gráficas se van actualizando siempre y cuando haya nuevas entradas en la base de datos, de lo contrario permanecen estáticas.

4.5.1 Instalación de las librerías

```python
import sqlite3
from dash import Dash,dcc,html,Input, Output, no_update, callback
import pandas as pd
import plotly.graph_objects as go
import datetime as dt

#Fechas de las ultimas medidas de Base de Datos
fecha_temp="2000-01-1 00:00:00"
fecha_temp="2000-01-1 00:00:00"
fecha_hum="2000-01-1 00:00:00"
fecha_incl="2000-01-1 00:00:00"
fecha_obs="2000-01-1 00:00:00"
fecha_tacto="2000-01-1 00:00:00"

#Definir formato de los datetime strings
datetime_format = "%Y-%m-%d %H:%M:%S"
fecha_temp= dt.datetime.strptime(fecha_temp,datetime_format)
fecha_hum= dt.datetime.strptime(fecha_hum,datetime_format)
fecha_incl= dt.datetime.strptime(fecha_incl,datetime_format)
fecha_obs= dt.datetime.strptime(fecha_obs,datetime_format)
fecha_tacto= dt.datetime.strptime(fecha_tacto,datetime_format)
```

Ilustración 57 Instalación de librerías para crear la App Web

Las librerías se instalan siguiendo el mismo procedimiento descrito en el apartado 4.4.1.

De la librería Dash (este framework se describió detalladamente en el apartado 2.3.4) se importan los módulo necesarios para el desarrollo de la aplicación. Por otro lado la librería Plotly [25] se recomienda para implementar aplicaciones web analítcas con Dash ya que proporcionan gráficos interactivos y dinámicos.

La librería Datetime [29] se usa para tratar con datos de tipo fecha y hora de manera eficiente; por último la librería Pandas [28] es una herramienta muy potente y flexible en el análisis y manipulación de datos, contiene estructuras de datos como el `DataFrame` que son bidimensionales y similares a tablas u hojas de cálculo, además permite escribir y leer datos en bases de datos sin mucha dificultad.

4.5.2 Creación del Layout de la aplicación

```
#Inicializar aplicación Web
app=Dash(__name__)

app.layout = html.Div([

     dcc.Interval(
        id='interval-component',
        interval=5*1000,  # en milisegundos
        n_intervals=0
    ),
    dcc.Tabs(children=[
        dcc.Tab(label='Temperatura(ºC) DHT11', children=[
            html.Div([
                html.H1(children='Representación de medidas de sensores', style={'textAlign': 'center',
                                                                                 'color': '#66FF33',
                                                                                  'backgroundColor': '#800000'}),
                html.P(children="Medidas extraídas de la base de datos", style={'textAlign': 'center',
                                                                                'color': '#66FF33',
                                                                                'backgroundColor': '#800000'}),
                dcc.Graph(id='temp_BD')
            ])
        ]),
```

Ilustración 58 Creación del Layout de la App

Creamos la estructura de la aplicación usando distintos elementos del Layout. La gráfica de cada medida se representa en una ventana y contienen elementos del bloque `dash_html_components` como `htm.H1`, `html.P` y también elementos del bloque `dash_core_components` como `dcc.Graph`, `dcc.Tab` y `dcc.Interval` (se utiliza para definir el tiempo de actualización de las gráficas). Es importante destacar que el componente principal del Layout tiene estructura de árbol. En el elemento `htm.H1` se puede estilizar la apariencia de la tab.

4.5.3 Conexión con la base de datos

```python
#Extraer datos de la base de datos
def leer_DB():

    #Conectarse con la base de datos
    conn = sqlite3.connect('MEDIDAS_SENSORES.sqlite')
    cursor = conn.cursor()

    #Obtener datos de tabla TEMPERATURA
    sql_temp= """SELECT Temperatura,Fecha_hora_ FROM TEMPERATURA"""
    cursor.execute(sql_temp)
    data_temp = cursor.fetchall()
    nombre_columna=['Temperatura','Fecha_hora_']
    df_temp=pd.DataFrame(data_temp,columns=nombre_columna)

    cursor = conn.close()

    return df_temp,df_hum,df_obs,df_incl,df_tacto
```

Ilustración 59 Conexión con 'MEDIDAS_SENSORES.sqlite'

Se define la función `leer_DB()` para establecer conexión con la base de datos y extraer los datos de cada 1 de las 5 tablas usando el comando `SELECT`. Nos interesa tanto la columna de la magnitud registrada (será el eje vertical de la gráfica) como la columna con los datos de fecha y hora (que será el eje temporal). Todos los datos se guardan en `Data Frames` para que sea fácil su manipulación y se devuelven como valor de retorno de la función. Finalmente es necesario cerrar la conexión con la base de datos pues se volverá a establecer en otra parte del programa.

4.5.4 Callback y actualización de las gráficas.

```python
#Callback de aplicación Dash
#Definir Callback
@callback(
    Output(component_id='temp_BD', component_property='figure'),
    Output(component_id='hum_BD', component_property='figure'),
    Output(component_id='inc_BD', component_property='figure'),
    Output(component_id='obs_BD', component_property='figure'),
    Output(component_id='tacto_BD', component_property='figure'),
    Input(component_id='interval-component', component_property='n_intervals')
)
```

Ilustración 60 Definición del callback

Los callbacks en Dash tienen dos componentes: `inputs` y `outputs`. Se definen con el id y la propiedad del componente, en el caso de los `outputs` se refieren al elemento del Layout que se va a actualizar y para los `inputs` el que genera la activación del callback. En esta se busca actualizar las gráficas cada 5 segundos.

```python
def update_grafica(n_intervals):

    global fecha_temp
    dfT=leer_DB()

    #Obtener ultima fecha de cada tabla
    #Temperatura(ºC)
    fecha_temp_new=dt.datetime.strptime(str(dfT['Fecha_hora_'].iloc[-1]),datetime_format)
    fig_temp2= no_update

    if fecha_temp<fecha_temp_new :

        if fecha_temp_new>fecha_temp:
            #Actualizar gráfica de Temperatura
            print('Se ha actualizado la gráfica de Temperatura\n')
            print('Nueva medida de Temperatura(ºC): \n',dfT['Temperatura'].iloc[-1])
            fig_temp2= go.Figure()
            fig_temp2.add_trace(go.Scatter(x=dfT['Fecha_hora_'], y=dfT['Temperatura'], mode='lines', name='Temperatura'))
            fig_temp2.update_layout( xaxis_title='Tiempo', yaxis_title='Temperatura (°C)')

            fecha_temp=fecha_temp_new
    else:
        print('No hay datos nuevos que actualizar\n')

        fecha_temp=fecha_temp_new
        return fig_temp2
```

Ilustración 61 Función para actualizar las gráficas

En la figura anterior se muestra el código para actualizar la gráfica de Temperatura, se sigue el mismo procedimiento para actualizar el resto de las gráficas.

Al activarse el callback se hace la llamada de la función `update_grafica (n_intervals)`. Dentro de la misma se llama a `leer_DB()`, para obtener un `DataFrame` con los valores de las tablas, nos interesan los datos de la columna de la medida y los temporales . Para decidir si se actualiza o no la gráfica hay que comparar la última fecha de cada medida registrada con la de la más reciente.

Al principio cada figura se inicializa con `no_update` (se usa para solo actualizar elementos concretos de la aplicación cuando se activa el callback) y este estado cambia si se cumple la condición temporal, esto es si la fecha guardada de la iteración anterior es menor que la fecha recopilada con el disparo del callback actual.

Las gráficas se actualizan de forma independiente, por lo que si solo es uno de los sensores el que registra datos nuevos, su gráfica correspondiente se actualiza y el resto permanece estática.

5 Resultados y conclusiones

5.1 Resultados del proyecto

Al conectar el dispositivo final, se ejecuta el programa de Arduino y los sensores comienzan a recopilar medidas y se envía el paquete con los datos hacia el Servidor de Red (The Things Stack).

Al ejecutar `conectarTTN_to_BD.py` se obtiene la payload decodificada y se insertan los datos a sus correspondientes tablas en la base de datos.

```
Suscripción exitosa al tema: v3/richard-lorawan@ttn/devices/eui-70b3d57ed00629dc/up
Medidas tomadas
Humedad(%)   Temeratura(ºC)          Tacto                Obstaculo              Inclinacion          Fport
  63           21.4                   0                     0                     1                    2
  63           21.7                   1                     1                     1                    2
  62           21.8                   0                     1                     1                    2
  62             22                   0                     0                     0                    2
  61           22.1                   1                     0                     0                    2
  62           22.3                   1                     1                     0                    2
```

Ilustración 62 Toma de medidas con conectTTN_to_BD.py

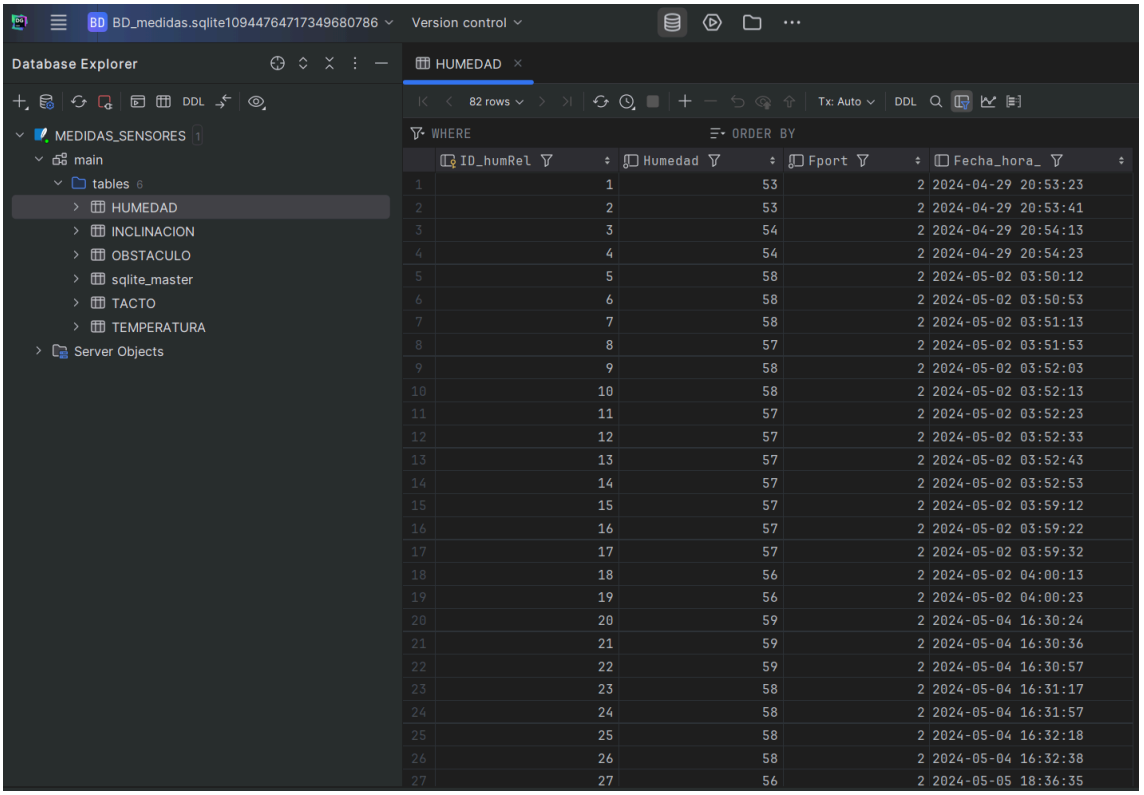


Ilustración 63 Aspecto de la tabla de Humedad Relativa en la BD

Por último al ejecutar `Visualizacion.py` se obtienen las gráficas de cada magnitud, si se ejecuta este fichero a la vez que se insertan nuevos datos en la base de datos las gráficas se actualizan.

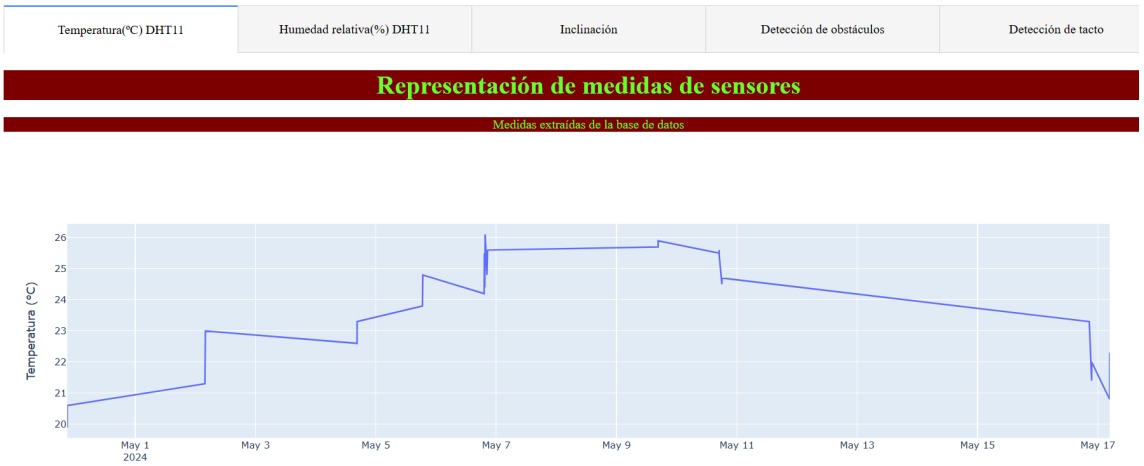


Ilustración 64 Gráfica de Temperatura (ºC)

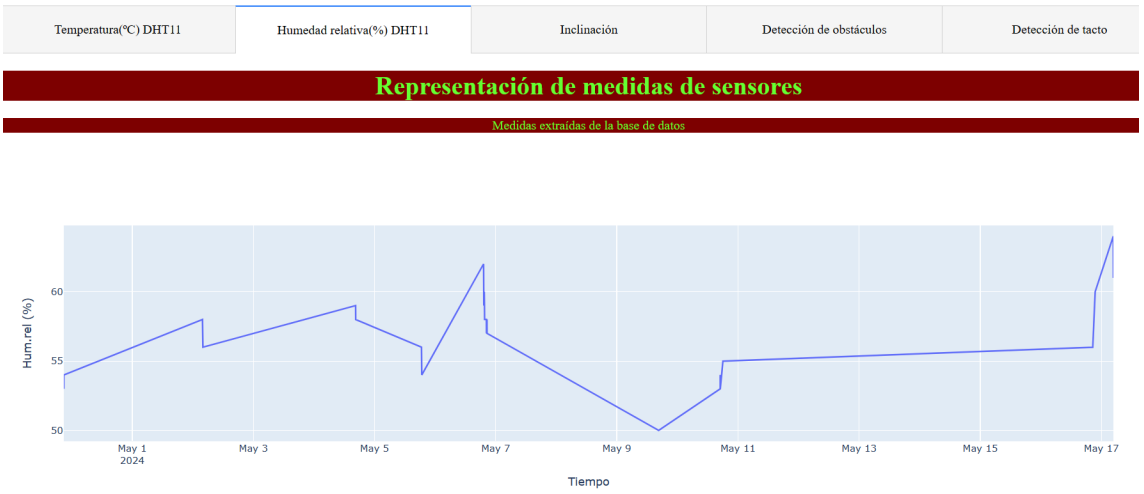


Ilustración 65 Gráfica de Humedad Relativa (%)

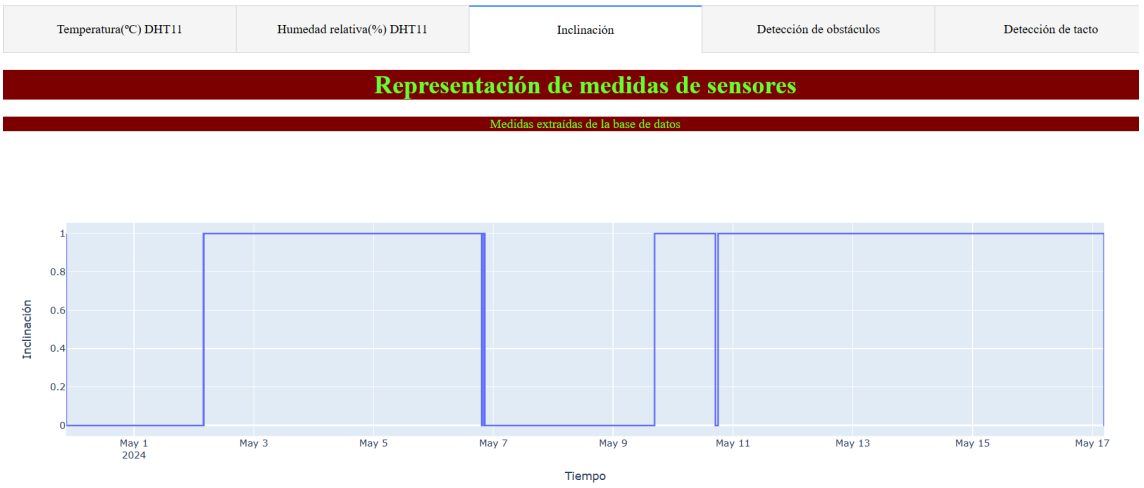


Ilustración 66 Gráfica del sensor de Inclinación

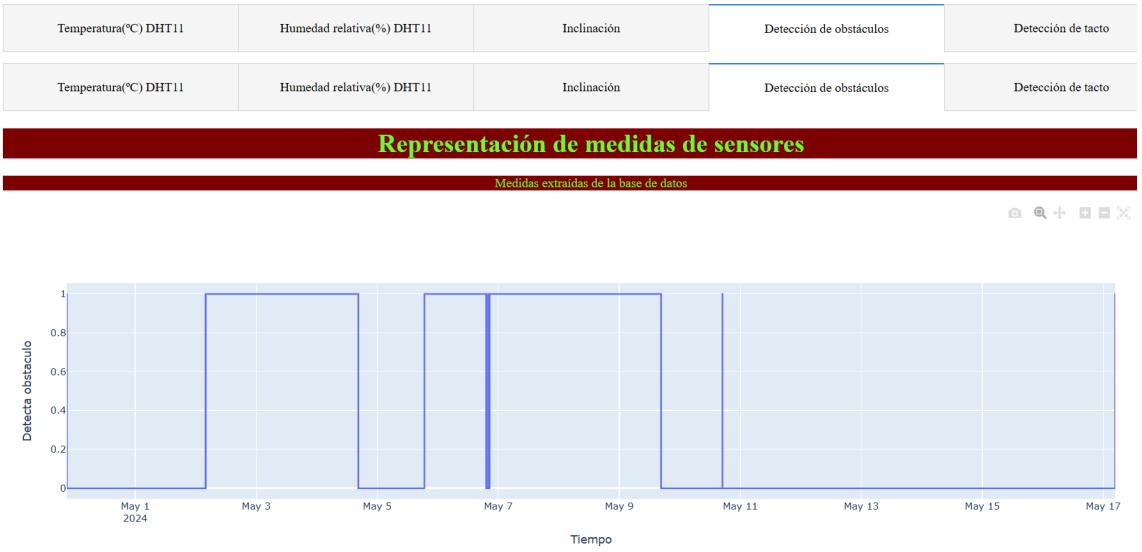


Ilustración 67 Gráfica del sensor de Detección de obstáculos

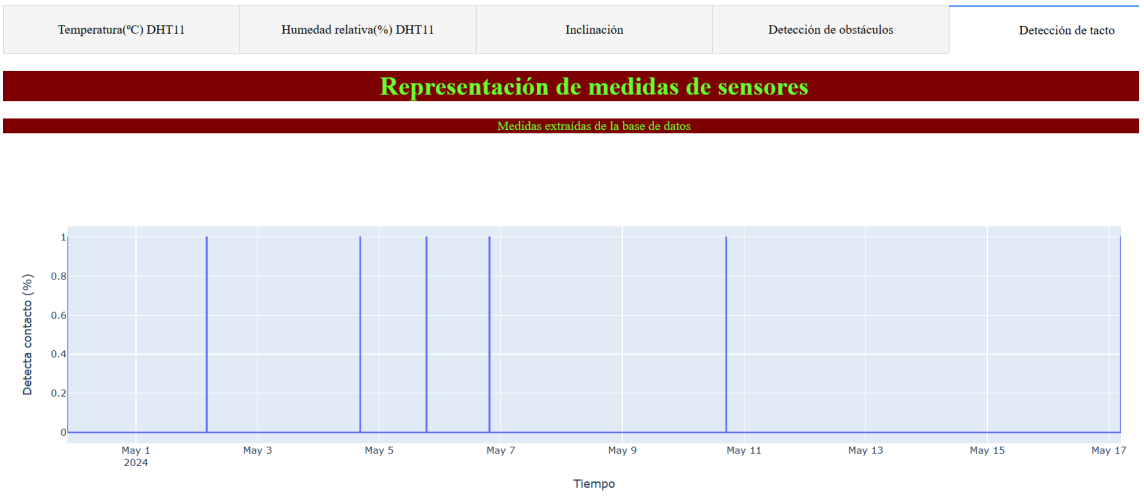


Ilustración 68 Gráfica del sensor de Tacto metálico

5.2 Conclusiones

Se llevó a cabo la programación en C++ de la placa CubeCell Dev-Board Plus (HTCC-AB02) en el IDE Arduino y se usaron una variedad de sensores de bajo costo y fáciles de adquirir. Además se diseñó un mecanismo para reducir el empaquetamiento de información logrando que la payload enviada fuese solo de 3 bytes.

También se creó una aplicación en The Things Network, se registró el nodo y se recibieron con éxito los mensajes uplink enviados por el dispositivo. Se creó un Payload Formatter en JavaScript para poder decodificar la payload de los mensajes.

Mediante el protocolo de comunicación MQTT se estableció comunicación con el servidor de red y fue posible suscribirse al tópico relacionado con la llegada de los mensajes uplink al servidor.

Se crearon dos aplicaciones en Python, con la primera se construyó una base de datos SQLite que guardó todas las medidas recogidas por los sensores en tablas independientes. Y por último una aplicación web usando Dash que hizo gráficas con los datos almacenados en la base de datos y que a su vez se actualizan si el servidor recibe nuevos mensajes del nodo.

En resumen, fue posible realizar un proyecto de IoT utilizando tecnologías LoRa/LoRaWAN y haciendo uso de múltiples herramientas gratuitas y de código abierto. No solo se cumplieron los objetivos propuestos si no que se profundizó en la parte relacionada con el Servidor de Aplicación.

5.3 Conclusions

C++ programming of the CubeCell Dev-Board Plus (HTCC-AB02) was carried out in the Arduino IDE using a variety of low-cost and easily available sensors. In addition, a mechanism was designed to reduce the packaging of information, ensuring that the payload sent was only 3 bytes.

An application was also created on The Things Network, the node was registered, and the uplink messages sent by the device were successfully received. A Payload Formatter was created in JavaScript to be able to decode the payload of the messages.

Through the MQTT communication protocol, communication was established with the network server and it was possible to subscribe to the topic related to the arrival of uplink messages to the server.

Two applications were created in Python, with the first an SQLite database was built that saved all the measurements collected by the sensors in independent tables. And finally, a web application using Dash that made graphs with the data stored in the

database and which in turn were updated if the server received new messages from the node.

In summary, it was possible to carry out an IoT project using LoRa/LoRaWAN technologies, using multiple free and open source tools. Not only were the proposed objectives met, but the part related to the Application Server was delved into.

5.4 Líneas abiertas

Una mejora inmediata para el proyecto sería aumentar el número de nodos finales y la variedad de sensores; por ejemplo programando con un dispositivo sensores analógicos tales como el DHT11, de presión, luminosidad, etc y con otro los sensores digitales. Se podría hacer la implementación alimentando las placas con baterías y estudiar los cambios necesarios para poder extender la vida útil de las mismas lo máximo posible.

También se podría configurar alarmas cuando las lecturas superan ciertos umbrales preestablecidos y sólo en esos casos enviar el paquete al servidor, lo cual requeriría un menor consumo de energía.

Para almacenar grandes volúmenes de datos se podría cambiar a una base de datos más potente como MySQL o almacenarlos en un servicio de la nube. Con grandes volúmenes de lecturas sería posible mediante técnicas de Machine Learning (Big Data) intentar predecir los valores esperados de temperatura, humedad relativa, presión, etc.

Por otro lado la aplicación web podría mejorarse al constuirse una interfaz más compleja que incluya características como la identificación de usuarios para restringir el acceso y las gráficas podrían visualizar una selección de datos en base a horas, días, semanas, etc.

5.5 Presupuesto

El coste total del desarrollo del proyecto se realizó teniendo en cuenta los materiales y el tiempo necesario para elaborarlo según lo descrito en la guía docente de la asignatura.

Concepto	Coste/Unidad (€)	Unidades	Total (€)
Coste/ hora de Ingeniero Industrial	17,00	150	2550,00
Cableado	0,10	30	3,00
Sensor DHT11	4,85	1	4,85
Sensor Til Switch	0,87	1	0,87
Sensor KY-036	1,99	1	1,99
Sensor KY-033	2,00	1	2,00
Dispositivo LoRa HTCC-AB02	15,5	1	15,5
Protoboards	4,85	1	4,85
Coste total (€)			**2583,06**

Tabla 1 Presupuesto del proyecto

Al ser un prototipo el coste inicial es elevado; si se decidiera hacer una implementación con miles de dispositivos finales y muchos más sensores, solo habría que tener en cuenta el coste de los componentes de hardware ya que el procedimiento sería similar y hay disponibles herramientas de software que son código abierto y gratuitas.

6 Bibliografía y Referencias

[1] Wikipedia contributors. Internet of things. Wikipedia, The Free Encyclopedia. Disponible en: https://en.wikipedia.org/wiki/Internet_of_things

[2] Enrgtech. Leading types of IoT wireless technologies. Enrgtech Blog. Disponible en: https://www.enrgtech.co.uk/blog/5-leading-types-of-iot-wireless-technologies/

[3]Alfa IoT. (2022, November 1). LoRa, una tecnología LPWAN ideal para el Internet de las Cosas. Alfa IoT. Disponible en:https://alfaiot.com/iot/lora-una-tecnologia-lpwan-ideal-para-el-internet-de-las-cosas/

[4] Baral, A. (2024). Understanding of LoRa. Medium. Disponible en: https://medium.com/@arghyabaral/understanding-of-lora-066d89ed7bf4

[5] Alfa IoT. (2022, November 17). LoRaWAN, el protocolo de red ideal para el Internet de las Cosas. Alfa IoT. Disponible en: https://alfaiot.com/iot/lorawan-el-protocolo-de-red-ideal-para-el-internet-de-las-cosas/

[6] UC Berkeley. (2021, February 8). LoRaWAN Explained [Video]. YouTube. Disponible en: https://www.youtube.com/watch?v=cUhAyyzlv2o&list=PLmL13yqb6OxdeOi97EvI8QeO8o-PqeQ0g&index=1

[7] Boot & Work Corp. S.L., Quesada Dani Salvans. (2020, November 17). ¿Qué es LoRaWAN? Industrial Shields. Disponible en: https://www.industrialshields.com/es_ES/blog/blog-industrial-open-source-1/que-es-lorawan-253

[8] The Things Network. Device classes. The Things Network. Disponible en: https://www.thethingsnetwork.org/docs/lorawan/classes/

[9] The Things Network. LoRaWAN architecture. The Things Network. Disponible en: https://www.thethingsnetwork.org/docs/lorawan/architecture/

[10] The Things Network. What is LoRaWAN? The Things Network. Disponible en:https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/

[11] The Things Network. LoRaWAN Message Types. The Things Network. Disponible en: https://www.thethingsnetwork.org/docs/lorawan/message-types/

[12] The Things Network. LoRaWAN Message Types. Disponible en: https://www.thethingsnetwork.org/docs/lorawan/message-types/

[13] The Things Network. End Device Activation. Disponible en: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/

[14] Heltec Automation. HTCC-AB02. Disponible en: https://heltec.org/project/htcc-ab02/

[15] Ultra-Lab. DHT11: Sensor de humedad y temperatura. Disponible en:https://ultra-lab.net/producto/dht11-sensor-de-humedad-y-temperatura/

[16] Proserquisa. Ventajas y desventajas del uso del sensor de humedad DHT11. Disponible en:https://proserquisa.com/principal/inicio/articulo/92

[17] uElectronics. Módulo KY-036: Sensor de metal. Disponible en: https://uelectronics.com/producto/modulo-ky-036-sensor-de-metal/

[18] ArduinoModules. KY-033 Line Tracking Sensor Module. Disponible en:https://arduinomodules.info/ky-033-line-tracking-sensor-module/

[19] Stackforce. LoRaMac-doc-v4.6.0. Disponible en:https://stackforce.github.io/LoRaMac-doc/LoRaMac-doc-v4.6.0/index.html

[20] Adafruit. Adafruit Sensor Library. Disponible en: https://github.com/adafruit/Adafruit_Sensor

[21] The Things Network. Airtime Calculator. Disponible en:https://www.thethingsnetwork.org/airtime-calculator

[22] HiveMQ. MQTT. Disponible en: https://www.hivemq.com/mqtt/

[23] Eclipse Foundation. Eclipse Paho MQTT Python Client. Disponible en:https://eclipse.dev/paho/files/paho.mqtt.python/html/client.html

[24] Python Software Foundation. sqlite3 - DB-API 2.0 interface for SQLite databases. Disponible en: https://docs.python.org/3/library/sqlite3.html

[25] Plotly Technologies Inc. Plotly Python Graphing Library. Disponible en: https://plotly.com/python/

[26] Plotly Technologies Inc. Dash User Guide & Documentation. Disponible en: https://dash.plotly.com/

[27] JSON. JSON: The JavaScript Object Notation. Disponible en: https://www.json.org/json-en.html

[28] pandas. pandas Documentation. Disponible en: https://pandas.pydata.org/docs/

[29] Python Software Foundation. datetime — Basic date and time types. Disponible en: https://docs.python.org/3/library/datetime.html