DESPLIEGUE DE NODOS PARA MEDIDAS DE IRRADIACIÓN SOLAR
DEPLOYMENT OF NODES FOR SOLAR IRRADIATION MEASUREMENTS

TRABAJO FIN DE MÁSTER
CURSO 2020-2021

AUTOR
MATHIAS SAURY ECHAGÜE

DIRECTOR
JOSÉ IGNACIO GÓMEZ PÉREZ
CHRISTIAN TOMAS TENLLADO VAN REIJDEN

DESPLIEGUE DE NODOS PARA MEDIDAS DE IRRADIACIÓN SOLAR
DEPLOYMENT OF NODES FOR SOLAR IRRADIATION MEASUREMENTS

TRABAJO DE FIN DE MÁSTER EN INTERNET DE LAS COSAS
DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

AUTOR
MATHIAS SAURY ECHAGÜE

DIRECTOR
JOSÉ IGNACIO GÓMEZ PÉREZ
CHRISTIAN TOMAS TENLLADO VAN REIJDEN

CONVOCATORIA: JUNIO 2021
CALIFICACIÓN: 7.5

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

16 DE JULIO DE 2021

DEDICATORIA

A mis padres, que nunca han dudado en
invertir en mi educación y darme su apoyo en
los buenos y malos momentos.

AGRADECIMIENTOS

Agradecer a mis tutores José Ignacio y Christian, por proponerme este proyecto y guiarme a lo largo del año para sacarlo adelante.

A todos mis compañeros de master, especialmente a aquellos que han hecho mi estancia en Madrid más especial y divertida, haciendo la experiencia universitaria mucho más enriquecedora.

Por último, a aquellas personas, estén a mi lado o no, que se han esforzado tanto en hacerme feliz, jamás hubiese llegado tan lejos sin ellas.

RESUMEN

Despliegue de nodos para medidas de irradiación solar

Este trabajo de fin de master, se engloba en el marco de proyectos de Irradiación solar, empezado en 2017. Concretamente, se ha enfocado en la búsqueda, aprendizaje y puesta en marcha de una plataforma IoT para la gestión de nodos y en el desarrollo de código para los nodos que permita su despliegue y monitorización.

Para ello, se han esclarecido una serie de requisitos previos para la elección de una plataforma y, a continuación, se ha realizado un estudio para encontrar la plataforma adecuada. Una vez escogida, se ha puesto en marcha el despliegue de los distintos módulos que permitan la completa gestión de los nodos.

Paralelamente, se ha desarrollado el código necesario para permitir la correcta comunicación del nodo con la plataforma, se ha optimizado el código existente y se han implementado servicios como actualización inalámbrica (OTA), aprovisionamiento y desmantelamiento. Adicionalmente, se han escogido los protocolos y tecnologías de comunicación inalámbrica que mejor podían responder a las necesidades del proyecto.

Palabras clave

Irradiación solar, OTA, aprovisionamiento, desmantelamiento, plataforma IoT, MicroPython, Pycom, LoPy, Bosch.

ABSTRACT

Deployment of nodes for solar irradiation measurements

This master's thesis is part of the Solar Irradiation project framework, started in 2017. Specifically, it has focused on the search, learning and implementation of an IoT platform for the management of nodes and the development of code for the nodes that allows its deployment and monitoring.

For this purpose, a series of prerequisites for the choice of a platform have been clarified and then a study has been carried out to find the right platform. Once chosen, the deployment of the different modules that allow the complete management of the nodes has been started.

At the same time, the required code has been developed to allow the correct communication of the node with the platform, the existing code has been optimized and services such as wireless update (OTA), provisioning and decommissioning have been implemented. Additionally, we have chosen the wireless communication protocols and technologies that could best meet the needs of the project.

Keywords

Solar irradiation, OTA, provisioning, decommissioning, IoT platform, MicroPython, Pycom, LoPy, Bosch.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - **Introducción**

La generación de energía eléctrica ha sido uno de los mayores desafíos de la humanidad en los últimos tiempos. La sociedad actual, en los países desarrollados, depende prácticamente en su totalidad de la energía eléctrica, en los trabajos, en el hogar, en el transporte público… Desde que nos levantamos hasta que vamos a dormir, estamos continuamente dependiendo de la electricidad, casi de manera automática e inconsciente.

El exponencial crecimiento de la población, la industrialización de los países en vías de desarrollo y países del tercer mundo que intentan ofrecer mejor calidad de vida a sus ciudadanos ha elevado el consumo energético a lo largo de los últimos años (Figura 1).

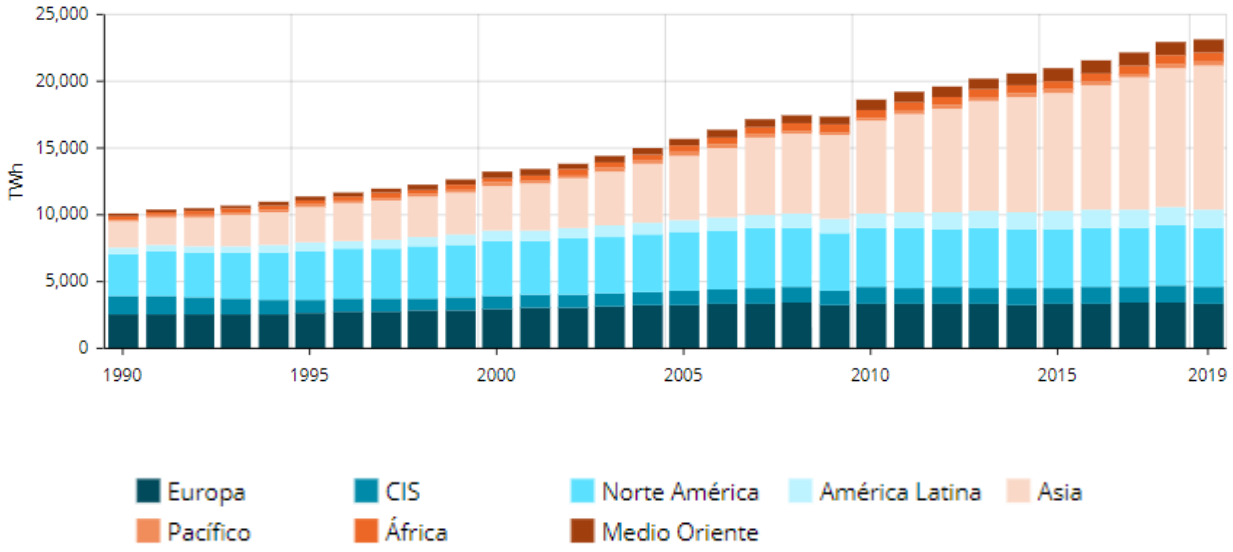


***Figura 1**. Crecimiento del consumo energético [1].*

La mayor problemática actual se haya en que casi la gran mayoría de la energía producida mundialmente es dependiente de combustibles fósiles. Estos combustibles fósiles tienen un impacto medioambiental muy negativo, ya sea por el CO_2 generado por la quema de combustibles, aumentando el efecto invernadero y la polución en las ciudades; o ya sea por el uso de uranio en las plantas nucleares, las cuales generan residuos radiactivos de difícil tratamiento y degradación.

Cada vez se invierte más en fuentes de energía renovable, donde los recursos son ilimitados y su uso produce menos consecuencias ambientales. Por esta razón, las grandes naciones han puesto su ojo en el desarrollo tecnológico de estas últimas, ya que, una vez desarrollado, generaría independencia energética de los países con respecto a los combustibles fósiles. Actualmente, el consumo mundial depende de los combustibles fósiles, siendo la energía de fuentes renovables muy pequeña en proporción a las demás (Figura 2).

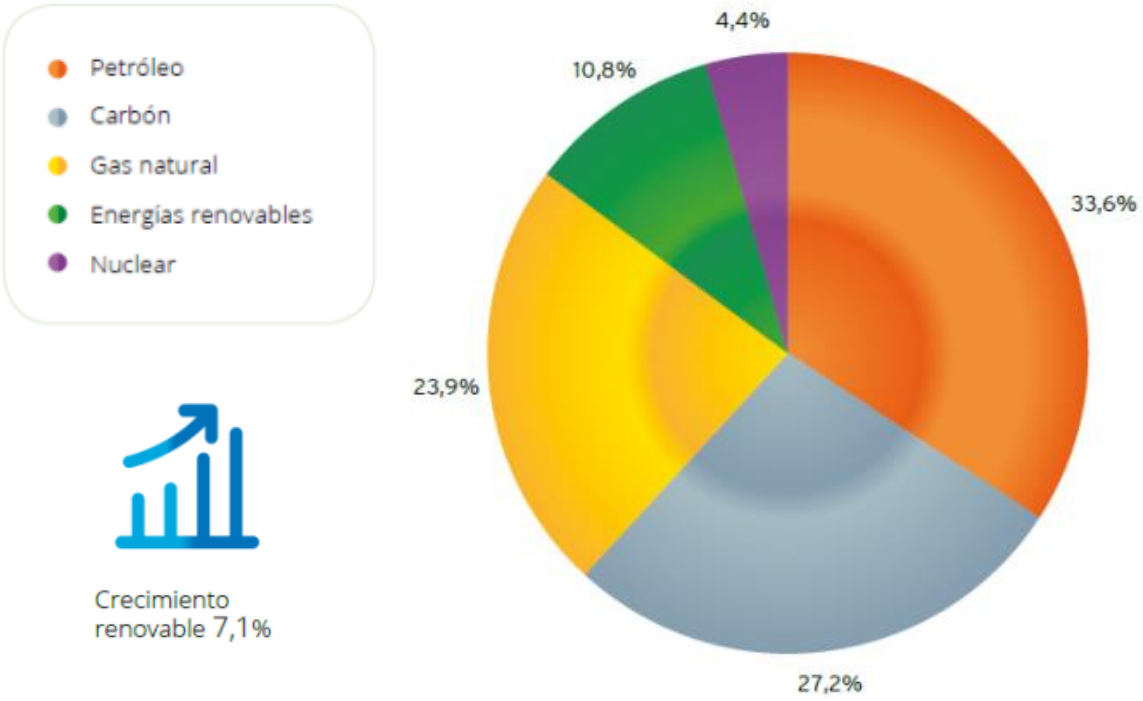


***Figura 2**. Porcentajes de uso de distintas fuentes de energía [2].*

La energía solar es la que más está creciendo en los últimos años en términos de producción, debido a la alta disponibilidad de energía solar y lo poco explotado que está actualmente. Como punto negativo, puede decirse que no puede satisfacer la demanda del cliente continuamente, ya que depende de factores meteorológicos y, adicionalmente, no puede proporcionar energía eléctrica durante la noche. También se necesita de un amplio espacio para instalación de placas en áreas de gran demanda y su almacenamiento dependería de baterías, las cuales tienen un problema de tratamiento y reciclado posterior.

Una solución al problema de la impredecibilidad de satisfacer la demanda con energía solar es a través de modelos predictivos de la irradiación solar. A través del aprendizaje automático y la recolección de datos de irradiación solar, podría elaborarse un modelo que permita predecirla a corto plazo.

1.1. Motivación

La creciente demanda mundial, la posesión de los recursos de combustibles fósiles por algunos pocos países, los problemas e influencias políticas que esto supone y el impacto medioambiental tan elevado que tiene el uso de la obtención de energía actual, obliga a seguir invirtiendo y trabajando en la línea de las energías renovables. Por ello, este proyecto se centra en la predicción de energía solar a corto plazo, con el objetivo de disminuir la demanda en el futuro de las energías más contaminantes.

La llegada del IoT en la actual Industria 4.0, pone a disposición de las actuales tecnologías energéticas una serie de ventajas que aún están en fase inicial, con un amplio abanico de estudios por realizar y ventajas que explotar. Con la tecnología IoT, se puede obtener una enorme

cantidad de datos para la aplicación deseada, y con ello, obtener modelos matemáticos que nos permitan predecir el futuro en el contexto utilizado.

Es por ello, que se pretende ampliar el área de conocimiento en el despliegue de red de sensores completamente monitorizadas por el usuario, el uso de distintas tecnologías IoT como las plataformas para el control de dispositivos y poder obtener así continuamente datos de la irradiación solar en parcelas de un lugar elegido. Con esto, podría obtenerse un conjunto de datos suficientemente grande en el tiempo para poder hacer uso de aprendizaje automático y crear modelos de predicción.

1.2. Objetivos

La idea de este proyecto es el desarrollo de una red de sensores que permita desplegarse en el campus de la Universidad Complutense de Madrid, gestionados a través de una plataforma IoT, con el fin de obtener valores de la irradiación solar en distintos puntos de la universidad a través de células fotovoltaicas. Por lo tanto, hay dos objetivos principales, el desarrollo de código para los dispositivos que se instalarán en la universidad y la implantación de una plataforma IoT para la gestión y monitorización de los nodos. Para lograr los objetivos y tener un mejor esquema organizativo, se han propuesto los siguientes subobjetivos:

- Optimización del código actual desarrollado en los trabajos realizados desde 2017.
- Identificación de nodos de manera independiente (uso de estándares abiertos).
- Implementación de actualización inalámbrica (OTA) que permita actualizaciones de programas y firmware de los dispositivos.
- Implementación de aprovisionamiento[1] (semi) automático de dispositivos nuevos y desmantelamiento[2] de dispositivos.
- Montar infraestructura que permita monitorizar los datos, controlar los dispositivos y realizar actualizaciones.
- A través de la plataforma, poder llevar a cabo análisis de los resultados obtenidos.
- Llevar a cabo la caracterización energética de los dispositivos con el propósito de obtener un perfil energético que permita adquirir o diseñar el hardware de manera eficiente.

1.3. Plan de trabajo

La primera etapa del trabajo se centra en realizar un estudio profundo de los trabajos realizados en el marco de proyectos en el que se trabaja, con el fin de extraer información relevante y poder seguir en la línea de los anteriores proyectos. Una vez estudiado el código de estudios anteriores, puede implementarse mejoras y nuevas implementaciones que permitan el objetivo definido.

[1] En inglés provisioning, es la instalación de software, proporción de parámetros y programas a dispositivos nuevo para un sistema determinado.

[2] En inglés decommissioning, es el desinstalado del dispositivo de un sistema.

La segunda etapa, comprende la búsqueda de la plataforma IoT adecuada para este proyecto, que permita la obtención de todos los módulos propuestos y su correcto desarrollo. Una vez elegida, se debe estudiar la documentación correspondiente a la elección e implementar los servicios propuestos en el módulo adecuado. En los casos en los que no sea posible o no se haya podido implementar en dicha plataforma, se deberá buscar distintas soluciones para su implementación.

Por último, se realiza un estudio energético del dispositivo para las tareas más recurrentes, con la finalidad de elegir el hardware adecuado. Este estudio se realiza en el laboratorio de la facultad de ciencias físicas de la universidad complutense de Madrid. Para cada tarea, se define el consumo energético de la placa y se calcula el tiempo de duración de batería para distintos escenarios.

Capítulo 2 - **Estado del arte**

Para contextualizar el trabajo, se ha realizado un estudio de las distintas tecnologías que podían emplearse en este trabajo, empezando por la comunicación inalámbrica, los distintos protocolos de aplicación y plataformas IoT. Por último, se da una visión general de cómo está el sistema de irradiación actual a través de los distintos trabajos realizados con anterioridad a este proyecto.

2.1. Plataformas de IoT

Una plataforma IoT es el software que proporciona conectividad entre hardware, redes de datos y puntos de acceso, proporcionando al consumidor una interfaz de usuario que le permite gestionar todos los dispositivos IoT y los servicios.

Las plataformas, a pesar de tener el mismo propósito y tener funcionalidades similares, normalmente se diferencian en las siguientes características:

- Confianza: tolerancia a fallos y errores, como la pérdida de datos o el bloqueo de la plataforma.
- Customización: algunas plataformas pueden dar servicios definidos y cerrados para el cliente. Al contrario, existen plataformas que dan al consumidor la flexibilidad de incorporar librerías, servicios de terceros o la modificación de alguno de ellos.
- Escalabilidad: capacidad de la plataforma ante el aumento de dispositivos y de la cantidad de datos que pueda suceder en el futuro.
- Precio: para un proyecto académico, va a ser una característica a tener en cuenta. El precio puede ser determinado por el número de dispositivos conectados, por la cantidad de servicios adquiridos o por la cantidad de datos transmitidos.
- Protocolos que acepta la plataforma para la comunicación entre dispositivos y aplicaciones.
- Seguridad: la capacidad que tiene la plataforma para ofrecer a los dispositivos cifrado de datos y autenticación.
- Soporte: servicio extra que tienen algunas compañías para atender las demandas o los problemas de los consumidores, además de mantener actualizaciones de la plataforma [3].

Además, se desea una serie de características que cumpla la plataforma escogida, que permita:

- Mantenimiento/gestión remota de los nodos de forma individualizada: lectura de los datos enviados por cada nodo, modificación de sus parámetros y capacidad de realizar *reset*.
- Realización de actualización OTA de los nodos (de forma individualizada y/o broadcast).
- Funcionalidad *digital twins*. Es un concepto tecnológico moderno, que se basa en tener un "nodo virtual" para cada nodo físico, que permita la abstracción del nodo físico concreto (modelo, firmware…). Las capas superiores de la plataforma se comunican con los "nodos virtuales" de forma homogénea. De esta forma, se puede observar el estado del nodo físico de manera actualizada, además de comunicarse con él para poder

cambiar sus parámetros y/o darle órdenes. En el siguiente cuadro, con el propósito de entender mejor el concepto, se representa para el presente proyecto como sería en los 3 casos. En el mundo real, se tiene el propio dispositivo, representado en el gemelo digital como una cadena de caracteres que permita clasificar el nodo. Para las distintas características, almacenadas en formato JSON en el llamado gemelo digital, pueden almacenarse números o cadenas de caracteres, que en la interfaz gráfica se representa mediante dibujos, gráficas, coordenadas geográficas en un mapa, etc.

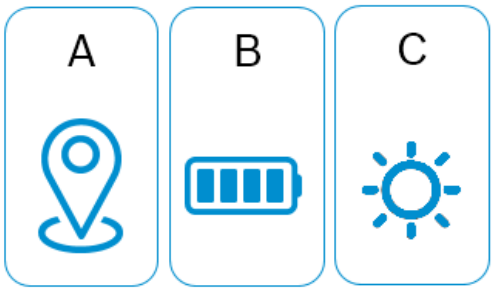
Mundo real	**Gemelo digital**	**Tu representación**
Dispositivo físico	Things IoT	GUI gemela digital
LoPy 4.0	ID = my.irradiation.nodo: Nodo1-via-gw-1	
Localización	ID / características / ubicación ..latitud 40.453 ..longitud -3.733	
Nivel de batería	ID / características / batería /.. lleno / medio / vacío	

***Tabla 1**. Representación real, virtual y gráfica.*

- Es deseable que la plataforma incluya soporte para la fase de *provisioning* y *decommissioning*. Es decir, que permita gestionar la incorporación y el desmantelamiento de nuevos nodos de forma automática y de forma sencilla.
- Es imprescindible que la plataforma no ligue a un hardware concreto para los nodos. Al ser una serie de proyectos en constante evolución, no se descarta cambiar en el futuro a otro hardware dependiendo de las necesidades del proyecto.

En el marco de proyectos en el que se trabaja, anteriormente se utilizaba Thingsboard. Es una plataforma IoT de código abierto que permite la recopilación, el procesamiento, la visualización y la gestión de dispositivos IoT. Posee una interfaz muy intuitiva, fácil de utilizar y al ser de código abierto, permite implementar servicios de terceros.

Con Thingsboard, concretamente en uno de los proyectos anteriores [4], se consiguió una plataforma que permitía la recepción y almacenamiento de datos de la LoPy 1.0r (Figura 3), pudiendo crear gráficas (Figura 4) y monitorizar el dispositivo de manera sencilla. Además, se implementó un panel de control que permitía cambiar la configuración de la LoPy a través de mensajes MQTT enviados desde Thingsboard.

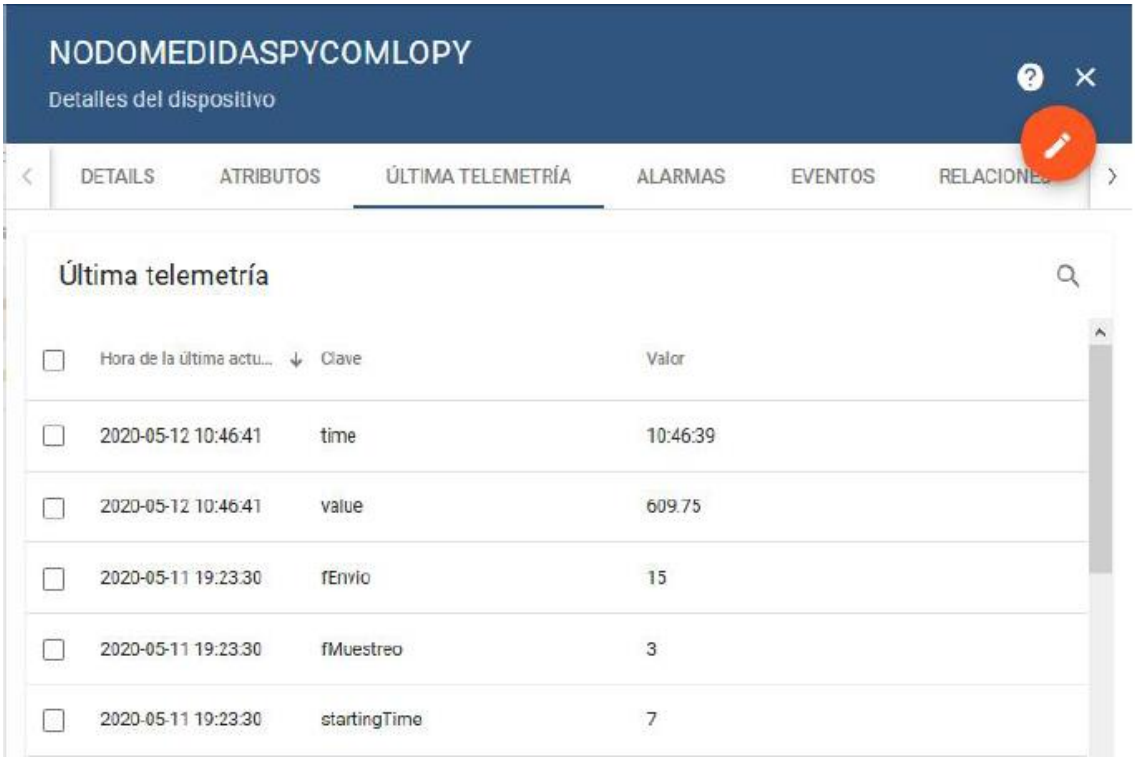


***Figura 3**. Dato de telemetría en Thingsboard.*

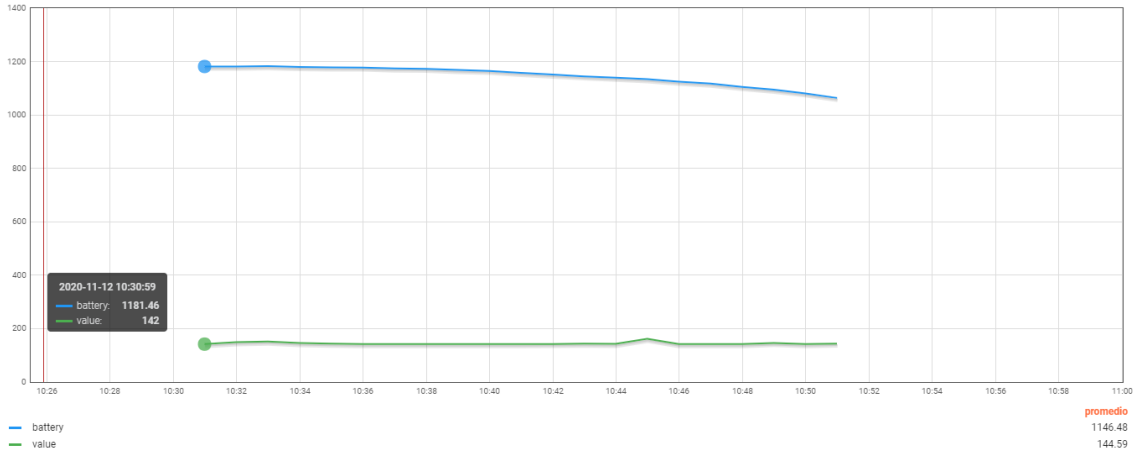


***Figura 4**. Gráfica en Thingsboard.*

El problema es que no posee las mismas capacidades o servicios implementados que las actuales plataformas, lo que hace que algunos de los requisitos deseados poseen un nivel de implementación complejo. En la actualidad, hay una amplia oferta de plataformas. Las que

actualmente lideran el mercado y cumplen con los requisitos específicos del proyecto (por nombrar algunas) son:

- Amazon Web Service IoT: con presencia en 190 países, es una plataforma que posee un amplio conjunto de herramientas y es la mejor opción para la construcción de ciudades inteligentes. Fácilmente escalable, con alto poder computacional y las capacidades avanzadas de la inteligencia artificial de Amazon. También está a la vanguardia en la construcción de productos para el hogar conectado, impulsados por Alexa Voice Service[3].
- Google Cloud IoT: posee un potente servicio para administrar datos de millones de dispositivos, optimizando las operaciones en sectores de fabricación, construcción y energía. Además, aprovechando el servicio de la plataforma Google Maps, aportan soluciones de transporte y ciudades inteligentes, permitiendo visualización de datos geográficos.
- Azure IoT: desde 2018, Microsoft invirtió enormemente en tecnologías IoT, teniendo actualmente su plataforma con una amplia gama de herramientas para todo tipo de dispositivos. Posee una fuerte presencia en el sector de salud, comercio, fabricación, energía, logística y transporte. Dispone además de la ventaja del más alto nivel de ciberseguridad, gracias a la inversión que Microsoft realiza anualmente en protección de equipos y tecnologías de ciberseguridad.
- Cisco IoT: lidera el mercado actualmente en términos de desarrollo *Egde Computing*, invirtiendo fuertemente en software y hardware para dirigir las cargas de trabajo en múltiples nodos, reduciendo la latencia de los sistemas IoT. Esto lo hace muy competente en los vehículos conectados, fabricación y Smart cities.
- IBM Watson IoT: líder de la industria en inteligencia artificial y aprendizaje automático, como pueden ser Watson Decision Platform para la agricultura o Watson Supply Chain Insights para la logística y el transporte conectado.
- Estas 5 plataformas son comerciales por lo que, debido a la demanda de espacio, datos transmitidos, número de dispositivos o servicios requeridos, tenía un coste económico. Como para un proyecto universitario es deseable una plataforma gratuita, se utiliza la plataforma Bosch IoT Suite, la cual cumple con todos los requisitos demandados. Más adelante, en Capítulo 4 - se explica su composición, una explicación detallada de su funcionamiento y su implementación.

2.2. Comunicación inalámbrica

En comunicación inalámbrica, a pesar de que existen más tecnologías de transporte de las que se nombran aquí, se ha orientado el estudio hacia aquellas que ofrecía el hardware que se tiene a disposición.

[3] Servicio en la nube que proporciona API para interactuar con Alexa, el asistente virtual de Amazon.

2.2.1. Sigfox

Esta tecnología nace de la mano de los franceses Ludovic Le Moan y Christophe Fourtet en 2010, con el objetivo de crear una red dedicada a internet de las cosas basada en datos de tamaño reducido, largo alcance y bajo consumo, ofreciendo un servicio de extremo a extremo. Destaca por los siguientes elementos:

- Posee una topología estrella, que, a diferencia del funcionamiento clásico de la arquitectura en estrella, no se conecta necesariamente a una antena específica, si no a cualquiera que tenga en su radio de alcance.
- Tiene un tamaño de carga de 12 bytes y tarda en promedio 2 segundos en llegar a las estaciones. Para un tamaño de 12 bytes de carga, el paquete tendrá un tamaño total de 26 bytes. El tamaño máximo de recepción de datos por los dispositivos es de 8 bytes.
- Opera en 200 kHz en la banda pública de 868 MHz, donde cada mensaje tiene un ancho de 100 Hz. Posee un alcance de 10 km en entorno urbano y 40 km en entorno rural.
- Velocidad de 10 a 1000 bits por segundo.
- Es una tecnología patentada, por lo que el uso de esta red requiere de un plan de subscripción [5].

***Figura 5**. Arquitectura Sigfox [5].*

2.2.2. LoRa y LoRaWAN

Es una tecnología actualmente administrada por LoRa Alliance, una organización sin ánimo de lucro comprometida en implementar a gran escala redes de área amplia y bajo consumo. Concretamente, LoRa es una técnica de modulación de espectro ensanchado, utilizando radiofrecuencias sin licencia y cubriendo la capa física del modelo OSI.

LoRaWAN por su parte, es el protocolo de red que usa la tecnología LoRa, una red de bajo consumo y largo alcance, diseñada para la conectividad de dispositivos de bajos recursos. Este protocolo posee las siguientes características:

- Utiliza una topología en estrella, pero permite la construcción de redes en malla.
- Alta sensibilidad para recibir datos.
- Largo alcance: 5km en entorno urbano y 15 km en rural.
- En Europa, opera en la banda de los 868 MHz.

- Su velocidad varía entre 0.3 kbps a 50 kbps.
- Los nodos se conectan al puerto de enlace que mejor le convenga, no a uno específico. De modo predefinido, las puertas de enlace (Gateways) se conectan al servidor de LoRaWAN (Figura 6).
- Tamaño de payload definido por el usuario, con un máximo de 255 bytes [6] [7].

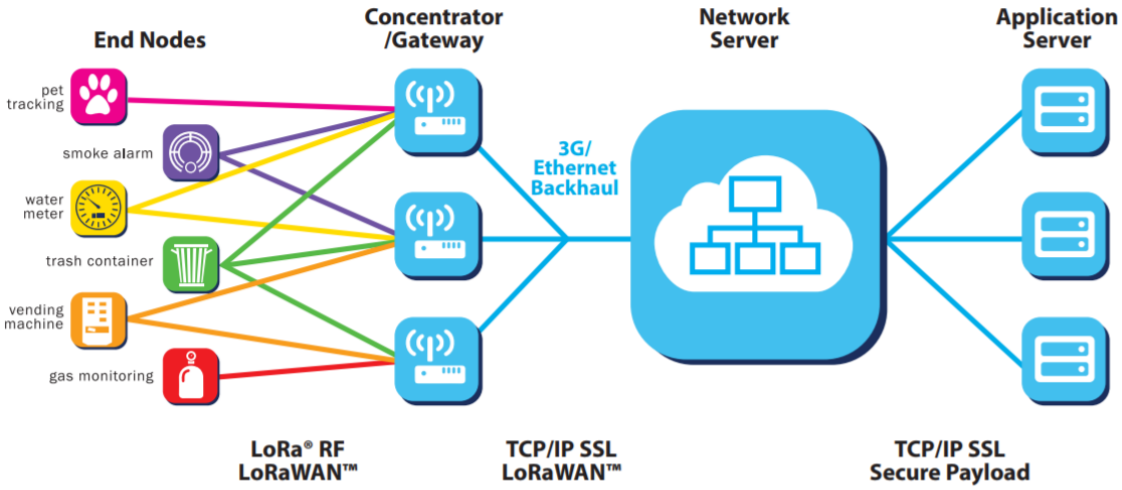


***Figura 6**. Arquitectura LoRaWAN [4].*

2.2.3. Bluetooth

Es una tecnología de comunicación de corto alcance creada por Bluetooth Special Interest Group, que posibilita la transmisión de voz y datos entre dispositivos mediante un enlace por radiofrecuencia en la banda 2,4 GHz, utilizando también la familia de estándares de 802.11. Es conexión orientada, es decir, la conexión entre dispositivos se mantiene, aunque no haya datos que transmitir.

Posee 40 canales, funcionando con un ancho de banda de 2 MHz. Usa 3 canales para la emisión de anuncios (paquetes que usan los dispositivos periféricos, de recursos limitados, para demostrar su existencia a los demás dispositivos). Estos canales están situados en la banda 2,4 GHz de tal forma que no solape con los canales 1, 6 y 11 de wifi, los más utilizados, para evitar interferencias. Los otros 37 canales son para la transmisión de datos. Envía datos a una velocidad de alrededor 1 Mbps con un payload de hasta 343 bytes [8].

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

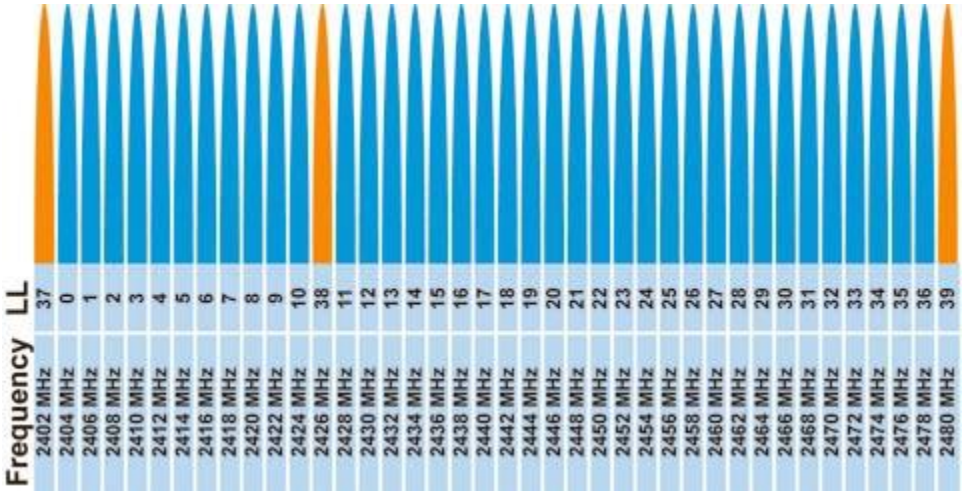


***Figura 7**. Canales Bluetooth [5].*

2.2.4. Wifi

Es un estándar abierto que permite la conexión inalámbrica entre dispositivos. Los dispositivos dotados con tecnología wifi pueden conectarse a internet a través de un punto de acceso. En la capa inferior del modelo OSI, funciona bajo la familia de estándares 802.11.

Operan en la banda ISM de:

- 2,4 GHz: esta banda ofrece solamente 3 canales que no se superponen. Está compuesta de 14 canales, funcionando cada uno de ellos con un ancho de banda de 22 MHz. La principal desventaja de esta banda es que, como se ha dicho anteriormente, los canales se superponen, además de operar con otros dispositivos en la misma banda (como bluetooth, microondas) pudiendo producir interferencias en la red.

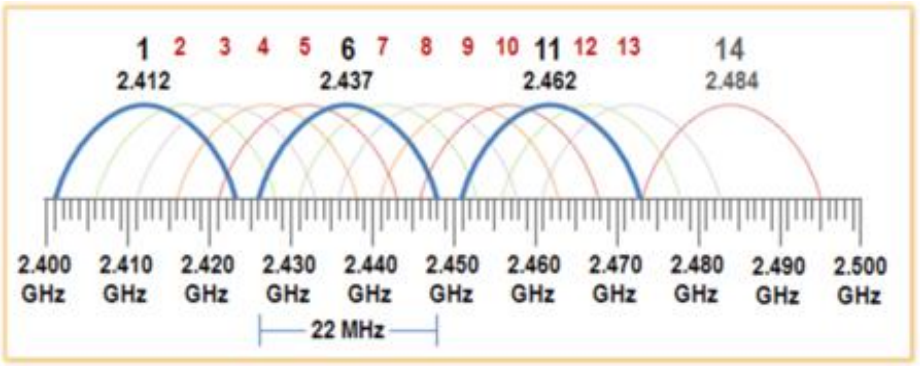


***Figura 8**. Canales wifi en 2,4 GHz [6].*

[5]Fuente de la imagen: https://www.comunicacionesinalambricashoy.com/imagenes/2013/03/La-tecnolog%C3%ADa-Bluetooth-de-bajo-consumo-utiliza-40-canales.jpg

[6] Fuente de la imagen: https://i.blogs.es/4a4b94/2410i3e1f4ccb87adef1e/450_1000.png

- 5 GHz: esta banda se usa en los dispositivos más modernos, por lo que se encuentra actualmente mucho menos congestionada. Además, con ancho de banda de 20 MHz, posee 21 canales que no se superponen. Algunos estándares de la familia 802.11 permite además operar con anchos de banda mayores, como 40, 80 y 160 MHz [9].

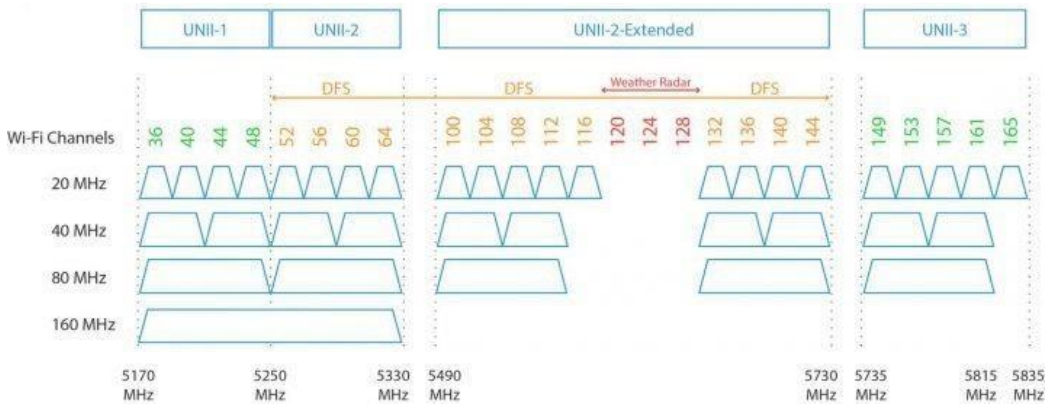


***Figura 9**. Canales wifi en 5GHz [9].*

La velocidad de transmisión varía enormemente, desde los 54 Mbps hasta 1.3 Gbps y el tamaño de payload máximo es de 1460 bytes.

2.2.5. Comparativa de soluciones

Las distintas alternativas que ofrece el hardware utilizado dan un amplio abanico de posibilidades para distintos escenarios. El objetivo no es decidir cuál es la mejor tecnología, ya que cada una cuenta con sus pros y contras, sino definir cuál es la más apropiada para el caso que concierne el trabajo desarrollado por parte del desarrollador IoT (Tabla 2). En trabajos previos a este, se ha utilizado la tecnología LoRaWAN, ya que poseía las características adecuadas para el envío de datos de telemetría, combinándolo con un bajo consumo.

Al haberse desarrollado distintos módulos que requieren de una transmisión de datos mayores en un reducido intervalo de tiempo, como puede ser la actualización online (OTA), el uso de mensajes con cifrado o la fase de provisioning, se requiere de una tecnología con la capacidad de enviar tamaños de payload relativamente grandes. Debido a este punto, y al hecho de tener en el campus de la universidad desplegado la infraestructura wifi, permite la instalación y puesta en marcha de los nodos de una manera mucho más rápida con cualquier otra tecnología. A pesar de que wifi no es la mejor en términos de ahorro de batería, el hecho de que el hardware utilizado vaya a estar alimentado por placas solares permite escoger esta opción.

	Sigfox	LoRaWAN	Bluetooth	Wifi
Rango	10-40 km	5-15 km	10-150 m	10-150 m
Velocidad	10-1000 bps	0.3-50 kbps	125 kbps – 2 Mbps	54 Mbps – 1.3 Gbps
Energía consumida	Muy bajo	Bajo	Bajo	Medio
Frecuencias	868 MHz (Europa)	868 MHz (Europa)	2.4 GHz	2.4, 5 GHz
Coste	Moderado	Moderado	Bajo	Bajo
Topología	Estrella	Estrella, malla	Estrella, malla, broadcast, P2P	Estrella, malla
Tamaño de payload	8 – 12 bytes	255 bytes	343 bytes	1460 bytes

***Tabla 2**. Comparación tecnologías inalámbricas.*

2.3. Capa de aplicación

La enorme disponibilidad de protocolos de la capa de aplicación permite obtener un protocolo que se adapte a las necesidades de cualquier proyecto. En los últimos años, han surgido protocolos o han cobrado mayor relevancia gracias al IoT. En este subapartado, se exponen los protocolos de la capa de aplicación que Bosch IoT Suite permite utilizar con su plataforma, a través de los adaptadores que posee.

2.3.1. CoAP

El protocolo CoAP (Constrained Application Protocol) está diseñado especialmente para nodos con pocos recursos y ancho de banda de red limitado. Su principal objetivo es que dispositivos con capacidades limitadas puedan conectarse máquina a máquina (M2M). Funciona bajo el mismo concepto de HTTP, con el mismo paradigma que en la RESTful API. Además de las características del modelo REST, posee particularidades especialmente diseñadas para IoT:

- Usa UDP como capa de transporte, por lo que el consumo de datos es más reducido que con el uso de TCP.
- Facilidad para convertir a HTTP con un proxy, por lo que lo hace compatible con una enorme cantidad de aplicaciones y dispositivos. Además, usa cabeceras más reducidas en sus peticiones.
- Posee mecanismos de subscripción.
- Si la información tiene un tamaño mayor que el paquete UDP, puede fraccionarla en bloques.
- Soporta seguridad mediante DTLS.

Funciona basándose en el intercambio de mensajes asíncronos entre dos nodos, uno actuando como cliente que envía peticiones al otro, el servidor, quien atenderá a dicha petición.

También, permite el descubrimiento de recursos alojados en el servidor, a través de una URI relativa (well-known/core). Es el servidor el encargado de decidir qué recursos permite descubrir [10] [11].

2.3.2. MQTT

Diseñado como un protocolo de mensajería de publicación/subscripción, con el objetivo de ser liviano, que permita el ahorro de ancho de banda y de energía consumida y la conexión de equipos remotos.

- MQTT es un protocolo de la capa de transporte cliente-servidor basado en publicación subscripción, a través de un servidor *broker* que funciona de intermediario. Funciona sobre TCP/IP, pudiendo ser modificado para funcionar sobre UDP.
- Es un protocolo de bajo consumo, abierto, simple, extremadamente ligero, lo que lo hace ideal para aplicaciones de internet de las cosas.
- Es fácil de implementar en diversas situaciones ya que posee numerosas librerías existentes, además de no necesitar mucho código para su implementación.

Además, posee otra serie de características deseables, como:

- La posibilidad de añadir cifrado a la comunicación a través de TLS y la autenticación de clientes.
- Entrega de mensajes confiable, llamado calidad de servicio, el cual es configurable dependiendo de las necesidades de la aplicación (Tabla 3) [12].

Para poder recibir un mensaje, un cliente debe estar conectado a un *broker* MQTT y subscribirse a un *topic*[7]. Si la conexión se interrumpe y se ha definido como no persistente, se detiene la subscripción y el cliente debe conectarse nuevamente, además de perder todos los mensajes que el cliente no haya recibido. Al pedir una sesión persistente al broker cuando se conecta, se guarda la información a través del ID del cliente, además de almacenar en el broker los mensajes con una calidad de 1 o 2 [13].

[7] Tema que permite filtrar los mensajes que se desean recibir.

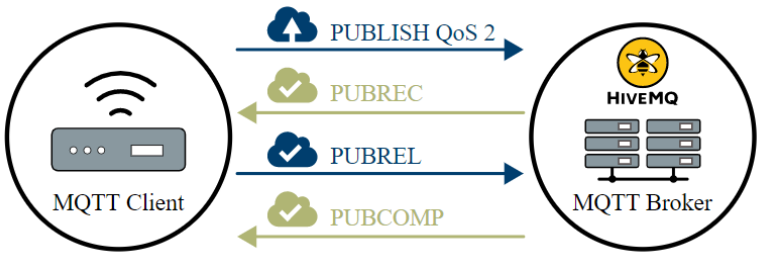
QoS=0 El mensaje no tiene garantía de entrega. Es el más ligero de todos, ya que el mensaje envía un solo paquete publish.	
QoS=1 El mensaje se entrega al menos una vez. El mensaje se almacena hasta que el cliente recibe un paquete PUBACK. Se ha escogido este nivel para el proyecto.	
QoS=2 El mensaje se entrega exactamente una vez. Esto se produce a través de dos flujos de solicitud-respuesta entre emisor y recetor. Es el más seguro y pesado.	

***Tabla 3**. Distintas calidades de servicio MQTT [13].*

MQTT es el protocolo elegido en este proyecto para permitir la comunicación entre dispositivos y plataforma. La razón de esta elección se debe a que está enfocado a la comunicación de varios clientes a varios clientes, permitiendo enviar mensajes específicos a cada cliente o de forma masiva a todos los clientes que estén subscritos, siendo este uno de los propósitos del proyecto. CoAP sin embargo, al seguir el paradigma RESTful, se enfoca en la comunicación cliente-servidor (Figura 10). Además, MQTT posee la ventaja de tener una amplia cantidad de bibliotecas, por lo que la implementación de este protocolo es más eficiente.

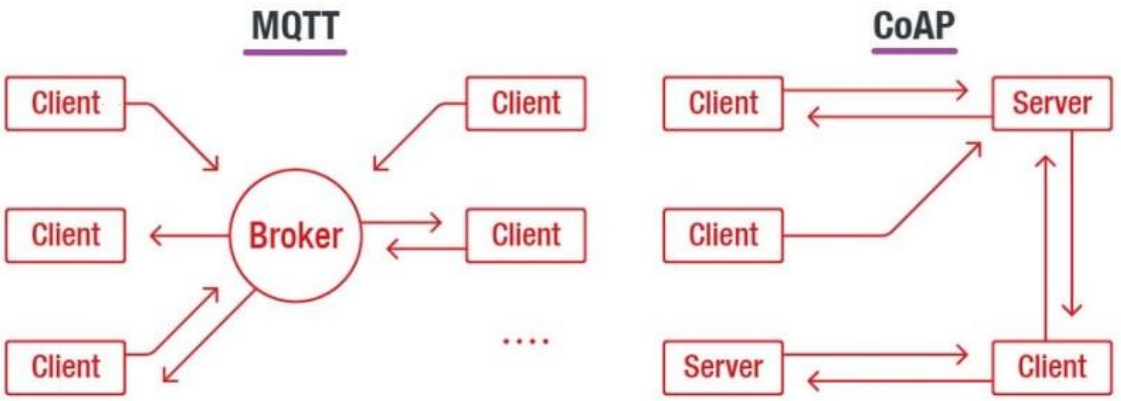


***Figura 10**. Esquema de funcionamiento MQTT vs CoAP [8].*

2.3.3. HTTP

De sus siglas Hypertext Transfer Protocol, es el protocolo de la capa de aplicación, el cual permite realizar peticiones de datos a través de la web con una estructura cliente-servidor. La petición de datos es iniciada por el elemento que recibirá los datos. HTPP posee las siguientes características:

- Funciona sobre TCP/IP.
- HTTP es un protocolo sencillo y fácilmente interpretable.
- Es extensible, pudiendo ampliarlo a distintas funcionalidades.
- Es un protocolo de sesiones, sin estado. Es decir, no guarda datos entre dos peticiones de la misma sesión.

En el caso que concierne este proyecto, HTTP es utilizado para realizar la comunicación de datos del servicio OTA implementado. El nodo, realiza la petición de archivos para su actualización vía HTTP a un servidor de la universidad. El servidor, es una computadora de la universidad que contiene, para nuestra aplicación, los archivos necesarios para la actualización del nodo [14].

2.3.4. AMQP

El protocolo de cola de mensajes avanzado (AMQP) es un estándar abierto que permite que aplicaciones cliente se comuniquen entre aplicaciones u organizaciones. Conecta sistemas, procesos comerciales con la información transmitiendo de manera confiable las instrucciones. Este protocolo, se encarga de una transmisión sólida de datos y almacenar mensajes en una cola, permitiendo una comunicación asíncrona. Posee las siguientes características:

- Es un protocolo binario que ofrece interoperabilidad y fiabilidad.
- Al igual que MQTT, se basa en mensajería publicación/subscripción.

[8] Fuente de la imagen: https://iotbyhvm.ooo/mqtt-protocol-mqtt-in-depth-mqtt-vs-coap/

- A pesar de las similitudes que este protocolo presenta con MQTT, AMQP está orientado a aplicaciones corporativas, con mayor rendimiento y redes de baja latencia.
- Es independiente de la plataforma y el lenguaje.
- Utiliza SASL y TLS para garantizar seguridad.

Debido a que el propósito de AMQP es la comunicación entre aplicaciones y en este trabajo se trabaja con datos de telemetría, es el protocolo utilizado para comunicar entre los distintos módulos de la plataforma IoT [15] [16].

2.4. Trabajos anteriores

Por orden cronológico, se nombran y explican los trabajos que han permitido la realización de este proyecto desde su punto de partida:

- Diseño e Implementación de un Nodo Sensor de Radiación Solar de Bajo Coste - Manuel Martínez Cebreiros (2016-2017). El primer trabajo, donde se desarrolla la circuitería del sistema de carga y medida, además del hardware necesario para llevarlo a cabo. Despliegan una red de sensores funcionando en LoRa y usan Node-Red en la puerta de enlace para la gestión de dispositivos.
- SHORT-TERM SOLAR IRRADIATION FROM A SPARSE PYRANOMETER NETWORK- Annette Eschenbach (2017-2018). Este proyecto comienza el otro punto principal de esta serie de proyectos, la predicción de la irradiación solar a corto plazo. Estudia los algoritmos adecuados para realizar las predicciones y realiza todas las tareas de preparación del algoritmo y evaluación del mismo.
- Desarrollo de red de sensores de irradiación solar de Iván García Palomino (TFG) (2018-2019). Desarrolla el código necesario para implementar los servicios de envío, muestreo y recepción de datos. Además, utiliza la plataforma ThingsBoard para el almacenamiento y visualización de los datos de los dispositivos.
- Predicción a corto plazo de radiación solar de Nicolas Mora (TFM) (2018-2019). Sigue en la línea de aprendizaje automático, dedicado a la predicción de la irradiación solar para distintos escenarios.
- Informe de prácticas de Lucio Giordano (2020). El más reciente, y el punto de partida de este trabajo, especialmente para el código del dispositivo. Además de crearse los scripts que a lo largo de este proyecto se explican, en ThingsBoard también se añadió la capacidad de poder lanzar órdenes a los dispositivos a través de la plataforma.

Capítulo 3 - **Hardware utilizado**

En el siguiente apartado se detallan los elementos de hardware que se han utilizado en este proyecto. Se dispone de placas LoPy 4.0 del fabricante Pycom, una compañía que ofrece dispositivos programados en MicroPython con soporte para múltiples tecnologías de red. Pretende ofrecer dispositivos que se programen de manera rápida y permitan desempeñarse en distintas redes inalámbricas.

3.1. LoPy 4.0

En el conjunto de proyectos de irradiación solar que abarca este trabajo, se ha utilizado el dispositivo LoPy 4.0 (Figura 11). Es una placa de desarrollo que funciona con el circuito integrado de ESP32, además de 4MB+520kB de memoria RAM. Es capaz de trabajar con distintas tecnologías de comunicación inalámbrica, ya que posee transceptor LoRa y Sigfox, además de una antena Bluetooth y wifi [17].

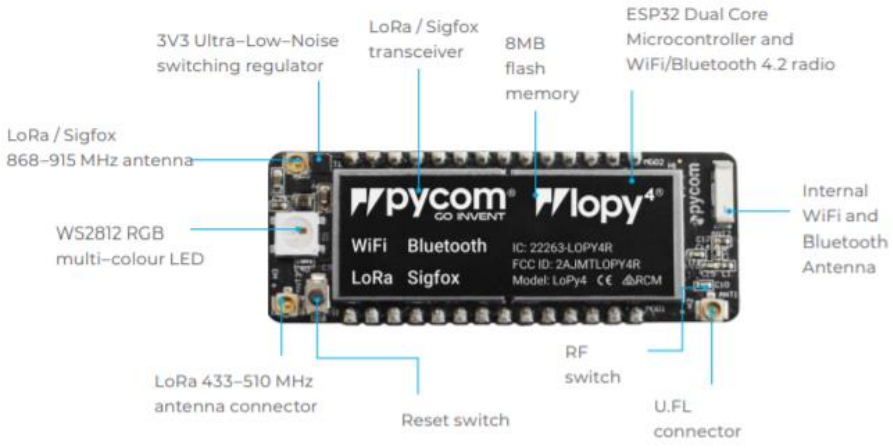


***Figura 11**. Placa Lopy 4.0 [17].*

En este proyecto, podría haberse optado por modelos más económicos de la misma empresa, que no incluyen la tecnología LoRa y Sigfox, como WiPy. Como es una serie de proyectos en constante evolución, no se cierra las puertas a otras tecnologías especialmente diseñadas para IoT, ni a otras plataformas que cumplan los requerimientos del proyecto.

Para programar la LoPy, se ha utilizado una placa de expansión 3.0 (Figura 12) alimentado a través de puerto USB con acceso serial y ranura para tarjetas microSD.

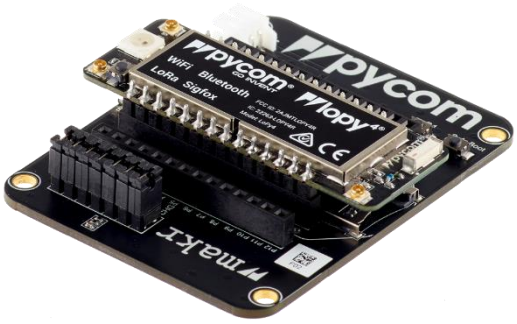


***Figura 12**. Placa de expansión 3.0 [17].*

3.2. Sensor de radiación

El componente que sirve para la toma de medidas de la irradiación solar es una IXOLAR SolarMD, un producto de módulo solar monocristalino, de células solares de alta eficiencia. El producto está pensado para servir de carga de productos portátiles, con sus células de alta eficiencia del 22% que pueden extender su funcionamiento en situaciones de baja irradiación, permitiendo conectar en serie o en paralelo dependiendo de las necesidades del producto.

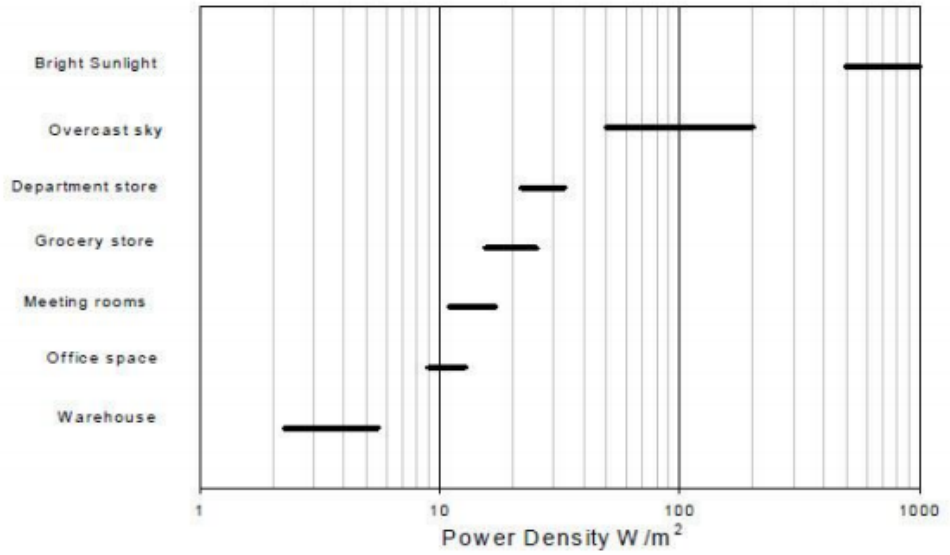


***Figura 13**. Gráfica con densidad de energía en distintos lugares [18].*

Debido a su sensibilidad en rangos de 300 nm (cerca de ultravioleta) hasta los 1100 nm (cerca de infrarrojos) incluyendo la luz visible (400 hasta 700 nm), es una opción adecuada para usarlo como sensor de irradiación y adaptar el voltaje recibido por esta placa para hacer una conversión a irradiación solar. En la Figura 13, vemos la densidad energética de la placa para distintos lugares [18].

3.3. Panel solar y batería

Para la alimentación de la placa y recarga de la batería durante el día, el sistema cuenta con un panel solar. La circuitería del sistema, adapta el voltaje de este panel solar para permitir la alimentación de todo el circuito y la carga de la batería.

Este panel solar es un AdaFruit 200 (Figura 14), que cuenta con paneles y celdas solares Medium. Estos, proporcionan una energía de salida de 2 W, una corriente máxima de salida de 330 mA y un voltaje máximo de salida de 6 V [19].

***Figura 14**. Placa solar y batería utilizada [19] [20].*

Otra pieza clave es la batería (Figura 14). Para este proyecto, se cuentan con dos modelos de distintas capacidades. Es objeto de estudio en el Capítulo 6 - , para comprobar la autonomía del dispositivo y si el material del que se dispone es suficiente o habría que aplicar modificaciones en el futuro.

- La primera batería es L503448 CELLEVIA, de polímero de litio con un voltaje de 3.7 V y una capacidad de 800 mA [21].
- La segunda batería es ACCU-LP103040/CL CELLEVIA, de polímero de litio con un voltaje de 3.7 V y una capacidad de 1200 mA [20].

Capítulo 4 - **Solución plataforma IoT**

Esta sección tiene la finalidad de explicar detalladamente la arquitectura del proyecto, la forma en la que se ha desarrollado cada parte y las soluciones escogidas para la plataforma cloud Bosch IoT. A grandes rasgos, el objetivo de la misma es la gestión de los dispositivos y la obtención de información de los sensores.

4.1. Arquitectura general

Se ha diseñado una arquitectura que permita la separación de las tareas, además de crear estructuras independientes. Con ello se tiene la finalidad de que la mayoría de los módulos puedan reutilizarse en el futuro o estén blindadas ante cambios en algún elemento de la arquitectura. En el siguiente esquema (Figura 15), se puede ver la arquitectura del sistema, estructurada en 3 bloques.

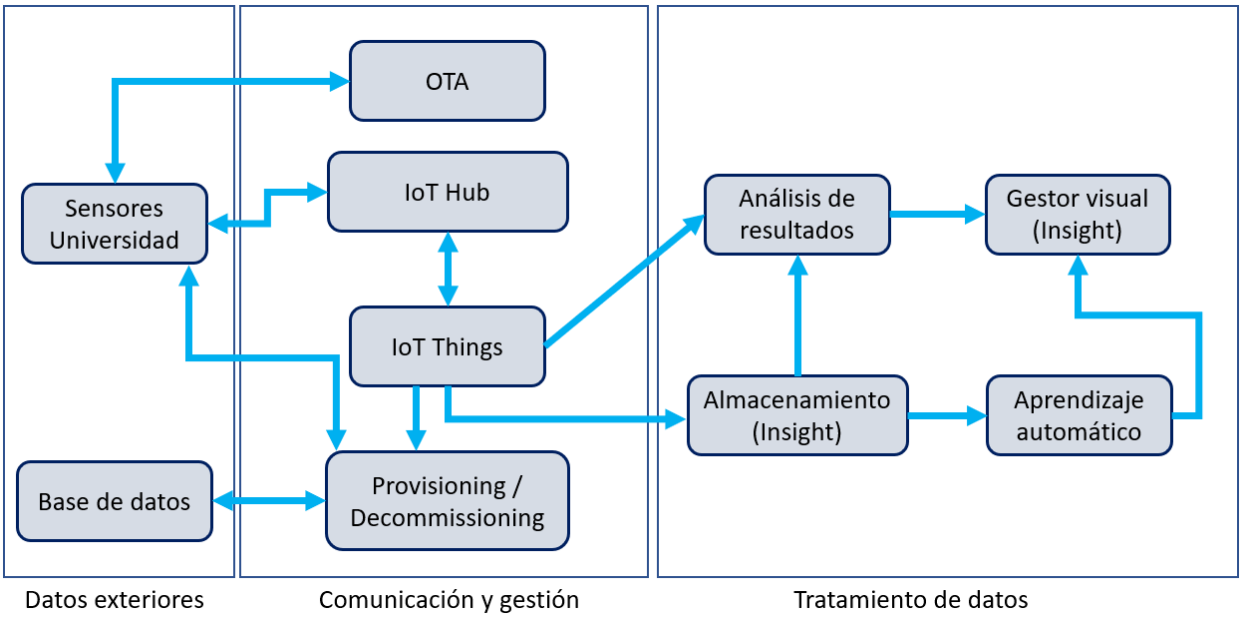


Figura 15. Arquitectura del sistema.

Cuando se habla de datos exteriores, se identifican aquellos que vienen tanto de los dispositivos que estén enviando datos del medio (los sensores), como aquellos alojados en otro software y que añadirán un flujo de entrada al sistema. Entre ellos, actualmente se encuentra:

- Los sensores de irradiación solar, se instalarán por distintas azoteas de edificios pertenecientes a la Universidad Complutense de Madrid. Están compuestos por placas LoPy 4.0 funcionando con MicroPython. Cada una de ellas está instalada de tal manera que tengan acceso a una red wifi y evidentemente exposición al sol constante (si las condiciones meteorológicas lo permiten).
- La base de datos MongoDB, encargada de proporcionar a los sensores de irradiación los datos necesarios para su funcionamiento y puesta en marcha. Se ha elegido debido a la facilidad de implementación, además de la ventaja que tiene el almacenamiento de

documentos en JSON. Esto facilita las tareas de envío sobre MQTT, además del tratamiento de los datos en Python.

El conjunto de comunicación y gestión es el punto intermedio y se encarga, como su nombre indica, de la gestión de los datos exteriores y almacenados, además de administrar los dispositivos. Dispone de los siguientes módulos:

- IoT Hub: es un módulo de la plataforma Bosch IoT Suite y un servicio en la nube que permite la conectividad entre la plataforma IoT y los dispositivos a través de distintos protocolos. En este caso, se utiliza el protocolo MQTT.
- IoT Things: es un módulo de la plataforma Bosch IoT Suite y un servicio en la nube que permite la creación y gestión de los llamados gemelos digitales. Se explicará más adelante, a grandes rasgos, permite gestionar los dispositivos conectados a la plataforma mediante un dispositivo virtual que simula al dispositivo físico.
- OTA: se encarga de la administración de las actualizaciones, registrando en un documento JSON los ficheros, archivos o firmwares que actualizar, eliminar o crear en el dispositivo destino. Además, entrega el contenido de la actualización al dispositivo vía HTTP.
- Provisioning/Decomissioning: realiza las operaciones de entrega de datos de la base de datos a los sensores en la fase de aprovisionamiento. Además, en el caso de realizar el desmantelamiento de un nodo, este se encarga de actualizar la base de datos.

El grupo tratamiento de datos en este trabajo es el menos desarrollado de los 3, ya que este proyecto se ha centrado en desarrollar la gestión mediante IoT de dispositivos. Se tienen y en el futuro se desea desarrollar los siguientes módulos:

- Almacenamiento: módulo donde se guardan todos los datos respectivos a los sensores, correspondientes al estado de la batería y la irradiación solar. Por defecto, en el software de Bosch Insight todos los datos se guardan en formato JSON.
- Gestor visual: se encarga de representar visualmente todos los datos del módulo de almacenamiento. Principalmente, se pueden representar gráficas sobre el histórico de datos, especialmente útil para los datos de irradiación solar, y el estado actual de la batería.
- Aprendizaje automático (no desarrollado): encargado de aprender a través de los datos almacenados y ser capaz de predecir la irradiación solar.
- Análisis de resultados (no desarrollado): a través de los datos provenientes de IoT Things o los almacenados, realiza un análisis y si es necesario lleva acabo una acción.

4.2. Bosch IoT Device Management

Bosch, ofrece 2 paquetes pre-configurados con las comunicaciones entre los distintos módulos de trabajo:

- Bosch IoT for Asset Communication, el cual permite el control por mensajes y la gestión de los distintos dispositivos.
- Bosch IoT for Device Management, que además de incluir todos los servicios del paquete anterior, cuenta con módulos para el provisioning, la gestión automática de dispositivos y la actualización OTA de dispositivos.

Inicialmente se iba a utilizar el paquete Bosch IoT Device Management para usar (además de los paquetes de Asset Communication) el módulo de *Rollouts* y *Manager*:

- Bosch IoT Rollouts: es un servicio que permite implementar actualizaciones de software en dispositivos, sirve un repositorio de dispositivos y software, de código abierto y que permite la incorporación de aplicaciones IoT (Figura 16).
- Bosch IoT Manager: es un servicio que proporciona un conjunto de funcionalidades para la gestión de dispositivos, como son la administración masiva de dispositivos, automatización basada en reglas, monitoreo, diagnóstico y resolución de problemas para los dispositivos conectados a través de distintos protocolos.

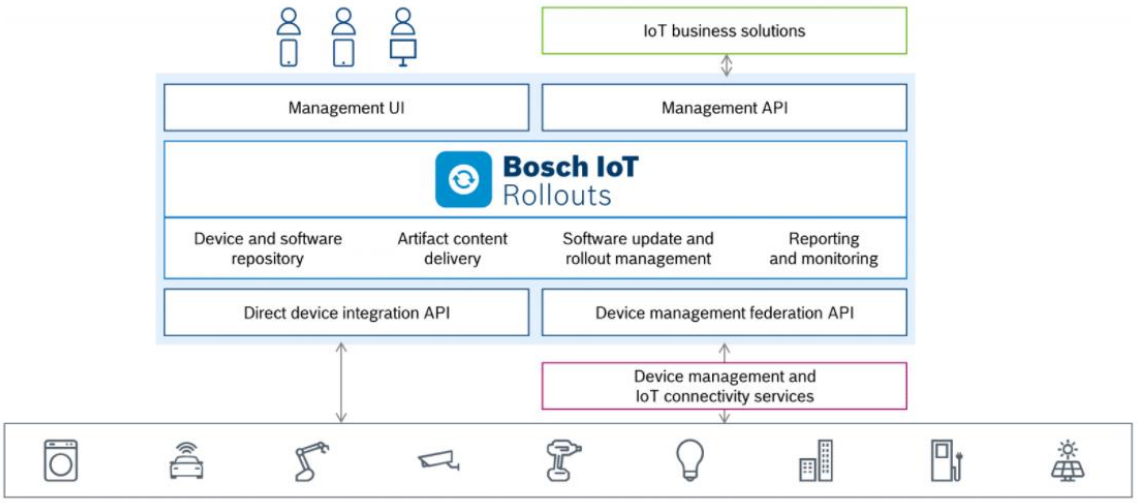


***Figura 16**. Arquitectura Rollouts.*

La intención, era implementar todos los servicios posibles (Figura 17) a través de la Suite de Bosch. A medida que se fue estudiando los distintos módulos, algunos de ellos carecían de documentación detallada para casos prácticos, especialmente para el desarrollo de OTA (con Rollouts) y del provisioning (con Manager), por lo que complicaba excesivamente la implementación. La facilidad que ofrecía Pycom para poder implementar OTA con ficheros Python en un servidor y el desarrollo de código para un servidor de aprovisionamiento era más eficiente que la puesta en marcha de Rollouts y Manager.

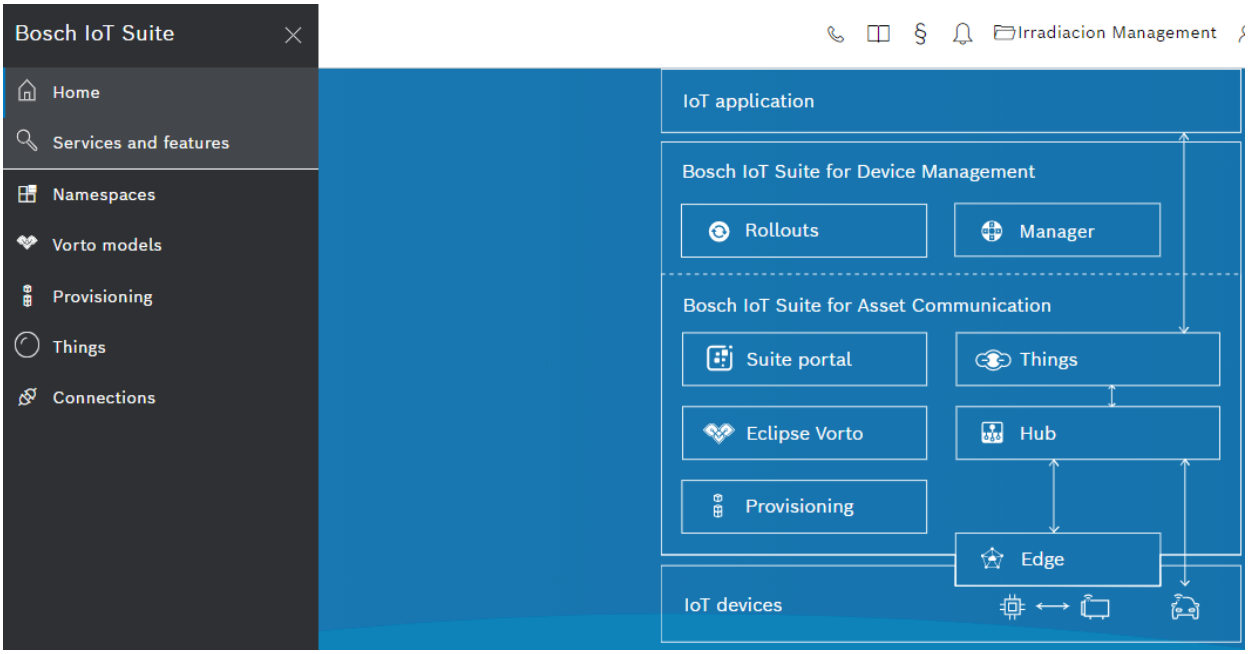


***Figura 17**. Pantalla principal Bosch IoT Suite.*

4.2.1. Bosch IoT Hub

Es el servicio de Bosch encargado de conectar los dispositivos a aplicaciones a través de distintos protocolos (en este caso, MQTT). Usando este servicio, las aplicaciones pueden recuperar datos de telemetría de dispositivos y enviar mensajes de control a los dispositivos. Está construido sobre componentes de código abierto desarrollados en el ecosistema eclipse IoT y otros. El que más destaca entre todos es Eclipse Hono.

Bosch IoT Hub, ofrece una solución para administrar los dispositivos y sus credenciales a escala, de modo que el usuario se centre en el desarrollo de la aplicación en lugar de los problemas técnicos para conectar los dispositivos. Con esto, se tiene un módulo que permite trabajar con distintos protocolos de IoT y que trabaja en la incorporación de los protocolos más novedosos.

Algunas de sus características son:

- Permite un soporte completo de comunicación dispositivo a nube y nube a dispositivo.
- Soporte listo para MQTT, AMQP 1.0, HTTP y CoAP.
- Soporte para redes LoRa con dispositivos habilitados con LoRaWAN.
- Gestión de dispositivos y sus credenciales para una conexión segura.
- Autenticación y autorización [22].

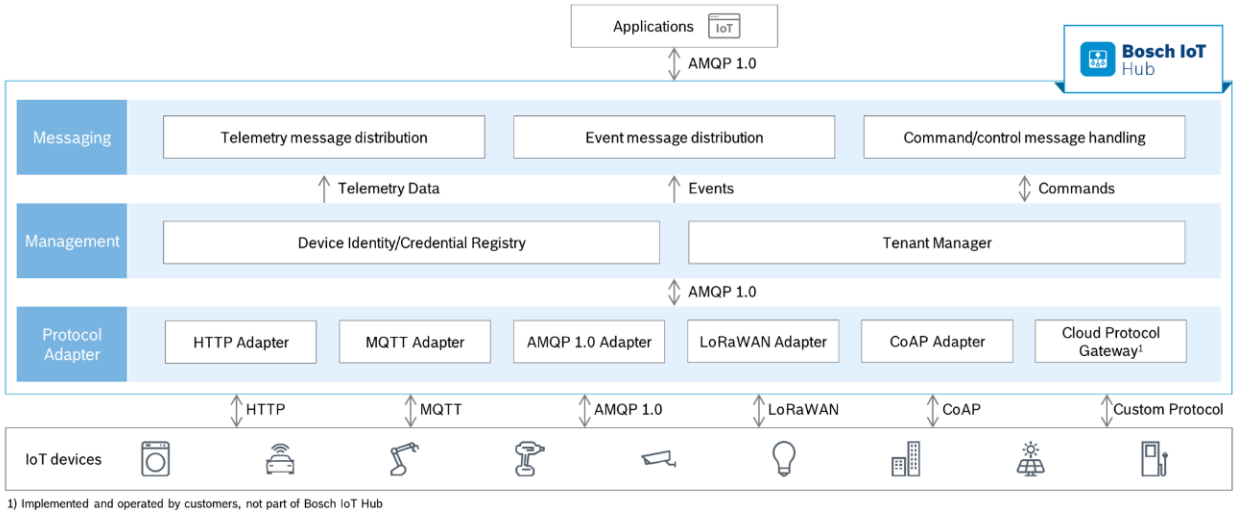


***Figura 18**. Arquitectura Bosch IoT Hub [22].*

En este proyecto, la forma de acceso a Bosch IoT Hub ha sido a través de MQTT. Para la gestión de los datos, dentro de la plataforma, Hub gestiona el mensaje con el username (nameSpace_ThingName@TenantID) y la contraseña, y a través del protocolo AMQP envía el mensaje a la aplicación correspondiente.

4.2.2. Bosch IoT Things

El módulo de Bosch IoT Things es un servicio administrado totalmente en la nube, que permite gestionar los gemelos digitales de los dispositivos registrados. Las aplicaciones que deseen acceder al módulo, pueden administrar datos de activos, recibir notificaciones automáticamente sobre los cambios y compartir datos/funcionalidad del dispositivo con aplicaciones de terceros. Este servicio, se basa y funciona con el proyecto de código abierto Eclipse Ditto. Algunas de sus características más notables son:

- Alta disponibilidad (del 99,5%) y confiabilidad, con una alta resiliencia.
- Estabilidad, crecimiento bajo demanda.
- Como se ha dicho, es código abierto, por lo que le da mucha flexibilidad a la hora de integrarse con aplicaciones de terceros y distintas infraestructuras.
- Ciclo de vida sencillo de los gemelos digitales:
    - Creación, en el aprovisionamiento.
    - Actualizaciones basadas en datos de telemetría y eventos del dispositivo, además de mensajes de aplicaciones comerciales.
    - Eliminación, en el desmantelamiento del dispositivo.
- A nivel de seguridad, contiene un control de acceso basado en políticas de cada interacción con los datos y funcionalidad del dispositivo. El documento de la política, contiene las reglas que describe con quien y que puede hacer con cada elemento del sistema. En nuestro caso, al gemelo digital a través de su política, se le indica permisos tales como lectura, escritura, envío y recepción de mensajes.

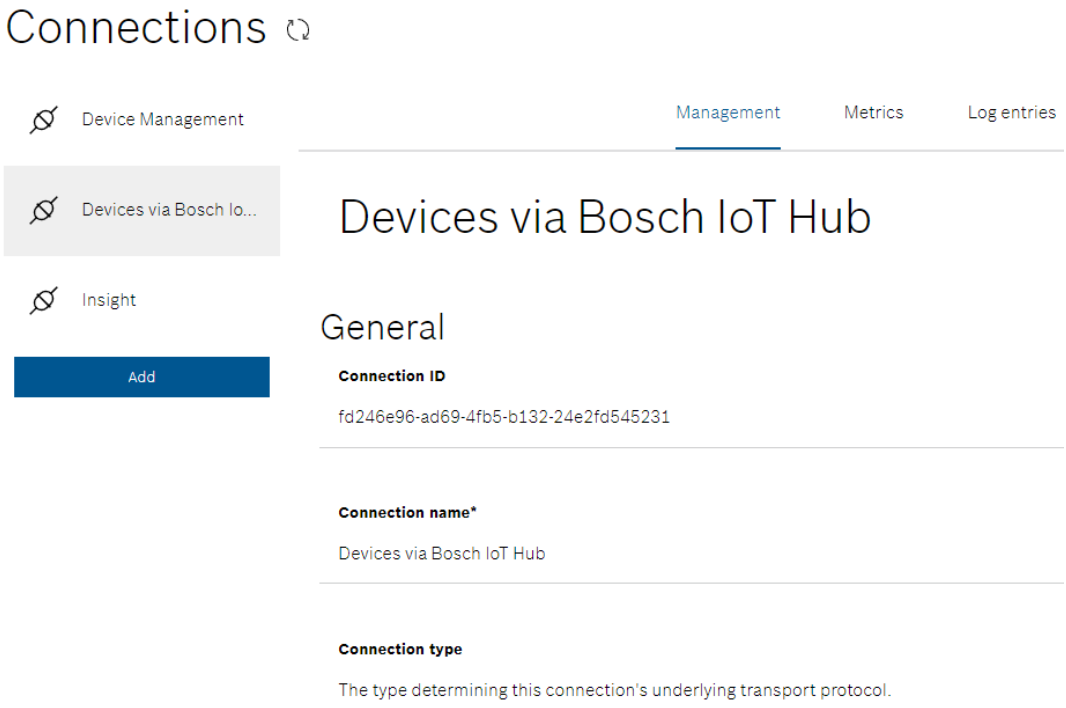


***Figura 19**. Apartado conexiones.*

La conexión con el módulo Bosch IoT Hub venía preconfigurada con el paquete que se ha contratado, por lo que la única configuración que necesaria es para el aprovisionamiento. Por ello, en el apartado conexiones de la Suite de Bosch (Figura 19), se diseña una plantilla para el gemelo digital una vez que un dispositivo físico desea registrarse en la plataforma en formato JSON, definido de la siguiente manera:

```
{
  "thingId": "{{ header:device_id }}",
  "_copyPolicyFrom": "{{ header:Gateway_id }}",
  "attributes": {
    "Info": {
      "GatewayId": "{{ header:Gateway_id }}"
      "Version": "1.0.0"
    }
  },
  "features": {
    "Irradiation": {
      "properties": {
        "measureI": 0
      }
    },
    "Battery": {
      "properties": {
        "measureB": 0
      }
    }
  }
}
```

La interfaz de IoT Things resulta intuitiva y fácil de utilizar, compuesta por 3 tipos distintos de dispositivos:

- Stand-alone: dispositivos que se conectan directamente con IoT Hub.
- Gateway: dispositivos que sirven de puerta de enlace de IoT Hub con otros dispositivos virtuales.
- Virtual-device: dispositivos virtuales conectados al Gateway.

Para este proyecto, se ha utilizado los dos últimos. La razón, es que el uso de un dispositivo Gateway permite automatizar la tarea de aprovisionamiento, explicado posteriormente en Provisioning.

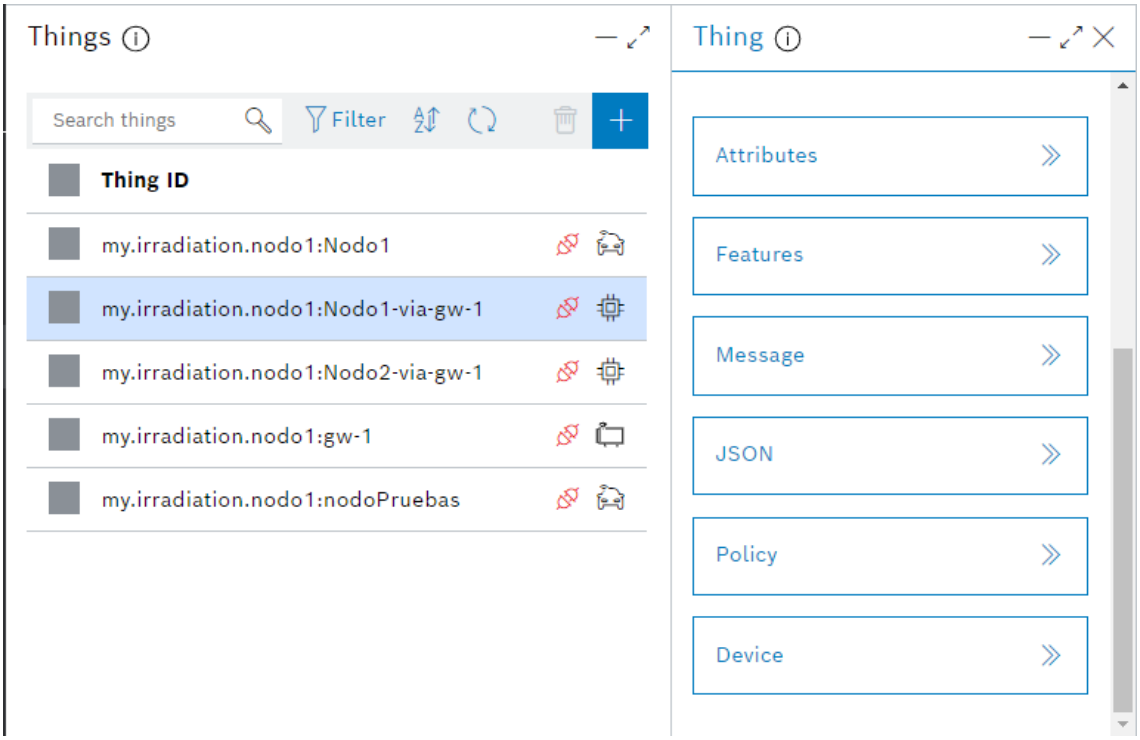


***Figura 20**. Things IoT con gemelos digitales.*

En la Figura 20, se encuentran algunos gemelos digitales creados, explícita o implícitamente. Cada uno de ellos en su interior posee las siguientes cualidades:

- Attributes: describen el dispositivo con más detalle, siendo una cualidad del gemelo que no varía, pudiendo ser de cualquier tipo. En el caso de este proyecto, como se puede ver en la Figura 21 , se ha puesto las coordenadas, información adicional (a que Gateway está conectado), el lugar donde está posicionado, los distintos tiempos de trabajo y la versión del programa que tiene instalada.
- Features: se añaden las características del dispositivo que se desea controlar, es decir, las que varían con el tiempo. En el caso que ocupa, será el nivel de la batería y la irradiación solar, cambiando en función del tiempo de envío especificado en los parámetros del dispositivo.
- Message: permite enviar mensajes con un tema (topic) desde Things hasta el dispositivo asociado a este. Se envía en formato JSON, y en el caso de este trabajo lo importante es

el subject, ya que el dispositivo en función del tema que tiene el mensaje realiza una acción determinada en la función callback de la subscripción.

- JSON: la definición en JSON del dispositivo de manera completa, permite modificar de manera directa el resto de atributos, características y política del dispositivo.
- Policy: se define los permisos de entrada, salida, escritura y lectura de los gemelos digitales. Proporciona acciones de gestión para las políticas definidas en los distintos dispositivos.
- Device: en esta vista, se puede administrar las credenciales del dispositivo, además de deshabilitar fácilmente la comunicación entre un dispositivo físico e IoT Hub.

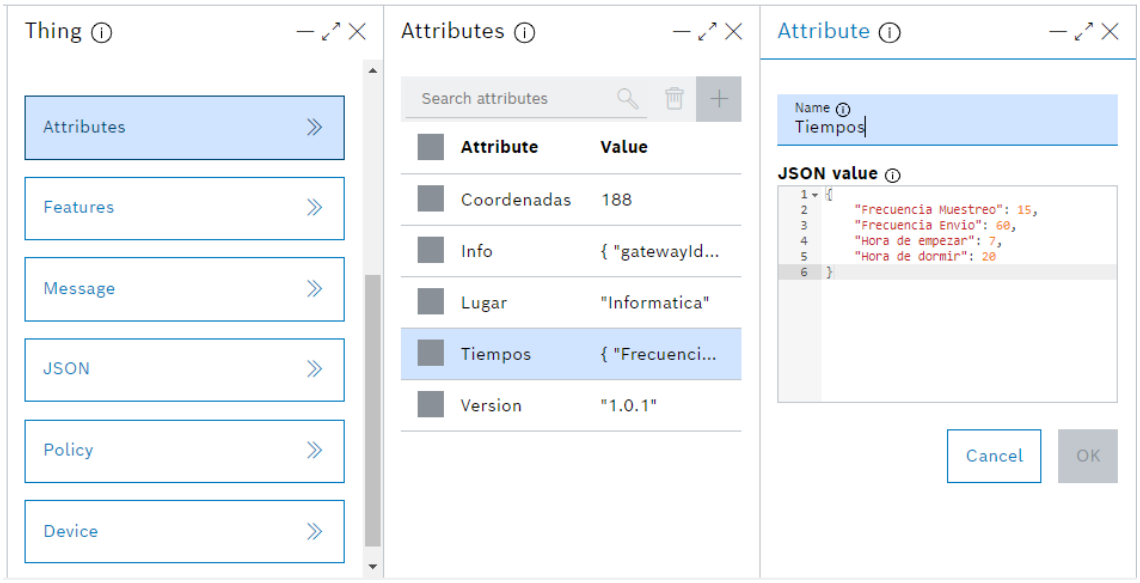


***Figura 21**. Partes de un gemelo digital de Bosch.*

4.3. Bosch IoT Insight

Para el almacenamiento de datos y su representación gráfica se contrató el paquete de Bosch IoT Insight. Insight, es un servicio en la nube que proporciona servicios API rest y una interfaz de usuario, permitiendo almacenar, consultar y aislar datos. Los datos en el caso de nuestro trabajo, vienen del módulo IoT Things vía HTTP, que permite la recepción de los datos de los gemelos digitales. Lamentablemente, no permite una conexión bidireccional con el módulo de Things, no pudiendo crear un Dashboard que permitiese también el control del dispositivo.

Insight permite el procesamiento de datos de cualquier dispositivo, donde se almacenan los datos sin procesar (Figura 22). Una vez almacenados, el servicio permite tomar distintas acciones sobre estos datos, como puede ser el uso de bases de datos NoSQL como MongoDB (no utilizado en este trabajo) y visualizarlos en los paneles de control que Insight ofrece [22].

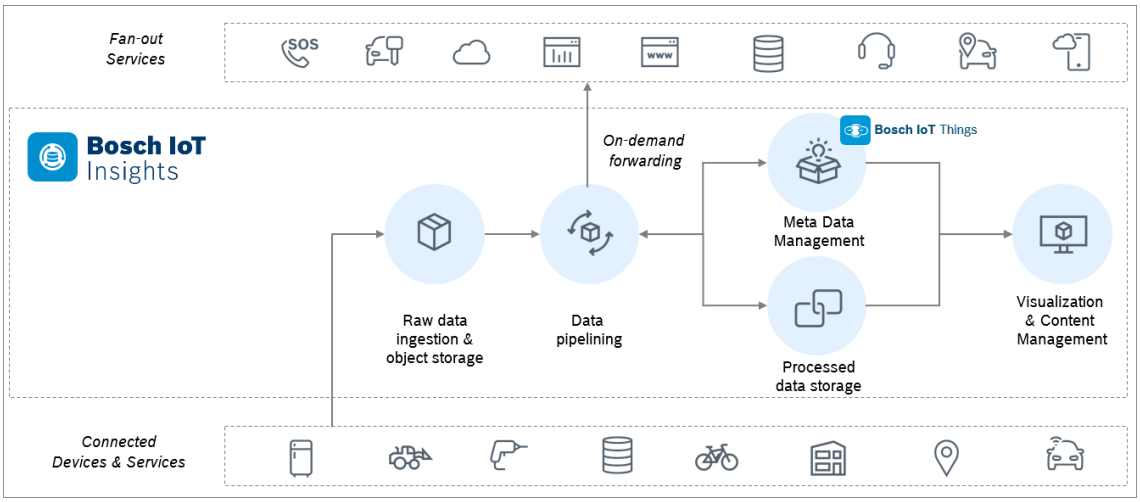


***Figura 22**. Arquitectura de Insight [22].*

En el caso de este proyecto, se ha utilizado como almacenamiento de datos en crudo, con el objetivo de analizarlos y a través de aprendizaje automático poder predecir en el futuro la radiación a corto plazo en los distintos puntos de la universidad. Además, se ha podido representar en gráficas la variación de la radiación solar a través de un gráfico de líneas (Figura 23).

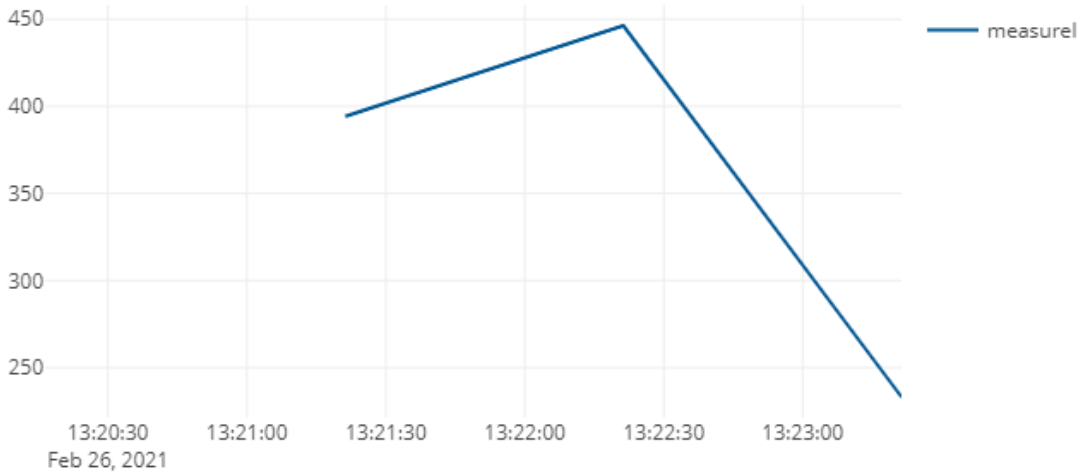


***Figura 23**. Ejemplo gráfica irradiación solar.*

También, gracias al envío de los datos del voltaje de la batería, se puede representar el estado actual de la batería de cada dispositivo (Figura 24). El entorno es muy útil para almacenar los datos que llegan del dispositivo para ser tratados posteriormente o visualizarlos, pero como Dashboard aún tiene funciones limitadas.

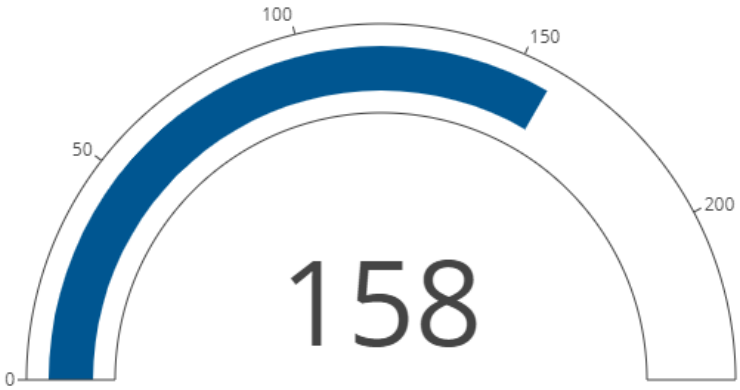


***Figura 24**. Estado voltaje batería.*

4.4. OTA

Las actualizaciones inalámbricas OTA (over the air) son los métodos de distribución de nuevo software y configuraciones para todo tipo de dispositivos. Una de sus características principales, es que una unidad central (como un servidor), puede enviar las correspondientes actualizaciones a todos los usuarios a través de tecnologías inalámbricas. Para este trabajo, donde se usan varios nodos distribuidos en distintos puntos geográficos, esta operación se vuelve indispensable para el correcto mantenimiento y mejora continua de los nodos. De no incorporar este servicio, habría que desplazarse físicamente a cada nodo, conectarlo al ordenador y proporcionarle las actualizaciones que se desea.

Para el módulo OTA, anteriormente se explicó que no se usó el servicio Rollouts integrado en Bosch, si no que el módulo se incorpora en un servidor (el ordenador que utilice el usuario). Se ha seguido el esquema y las directrices de funcionamiento del fabricante Pycom, ya que estos dispositivos vienen con la librería Pycom, la cual posee funciones que permiten desarrollar esta acción.

Para empezar, se ha implementado un ejecutable de Python que hace la función de servidor. Este servidor funciona con el protocolo HTTP en el puerto 8000, generando un manifiesto en formato JSON, donde se describe la acción a realizar para cada archivo y proporciona al cliente (el dispositivo) el contenido de la actualización. Este script, debe ejecutarse en un directorio donde posea los códigos destinados al cliente, ordenados por números de versiones. Siempre toma como archivos a actualizar los que posean un número de versión mayor. Como ejemplo, se tiene el siguiente directorio:

```
server directory
|- OTA_server.py
|- 1.0.0
|  |- flash
|  |   |- lib
|  |   |  |- lib_a.py
|  |   |- main.py
|  |   |- boot.py
|  |- sd
|     |- some_asset.txt
|     |- asset_that_will_be_removed.wav
|- 1.0.1
|  |- flash
|  |   |- lib
|  |   |  |- lib_a.py
|  |   |  |- new_lib.py
|  |   |- main.py
|  |   |- boot.py
|  |- sd
|     |- some_asset.txt
|- firmware_1.0.0.bin
|- firmware_1.0.1.bin
```

Por lo tanto, se puede actualizar tanto el código que posee como la versión del firmware del nodo. Al solicitar una actualización, en el manifiesto anteriormente mencionado, se registra una lista de archivos que deben ser eliminados, creados o actualizados. Este es un ejemplo de cómo quedaría el manifiesto:

```
{
   "delete": [
      "flash/old_file.py",
      "flash/other_old_file.py"
   ],
   "firmware": {
       "URL": "http://192.168.1.144:8000/firmware_1.0.1b.bin",
       "hash": "ccc6914a457eb4af8855ec02f6909316526bdd08"
   },
   "new": [
       {
           "URL": "http://192.168.1.144:8000/1.0.1b/flash/lib/new_lib.py",
           "dst_path": "flash/lib/new_lib.py",
           "hash": "1095df8213aac2983efd68dba9420c8efc9c7c4a"
       }
   ],
   "update": [
       {
           "URL": "http://192.168.1.144:8000/1.0.1b/flash/changed_file.py",
           "dst_path": "flash/changed_file.py",
           "hash": "1095df8213aac2983efd68dba9420c8efc9c7c4a"
       }
   ],
   "version": "1.0.1b"
}
```

Para el caso del dispositivo, se ha utilizado también el esquema propuesto por Pycom. La biblioteca OTA para el cliente posee dos capas organizadas en dos clases dentro del fichero OTA. En la capa de alto nivel, se encuentra las funciones relacionadas con la gestión de la actualización, como el análisis del manifiesto JSON creado en el servidor, hacer copias de seguridad de los archivos, y la creación/actualización/eliminación de los programas elegidos.

En la segunda capa, está la tecnología empleada (wifi) y el protocolo utilizado (HTTP). Esta organización, permite la reutilización de código y, si en el futuro se quiera cambiar de tecnología o de protocolo, el código de la capa de actualización puede ser válido. Cada nodo, posee en su fichero de configuración el número de versión que tiene actualmente, realizando la actualización solo si el número de versión que posee el dispositivo es inferior al almacenado en el servidor. En el Algoritmo 1, se explica cómo funciona el código que permita la actualización OTA dentro del dispositivo.

Algoritmo 1: OTA dispositivo

Entrada: Dirección servidor IP, puerto P, contraseña pass, SSID
Salida: Peticiones get HTTP (IP, P) HTTP_GET

```
1:  Crea objeto OTA
2:  Conecta a Wifi(SSID, pass)
3:  Obtiene último manifiesto vía HTTP (IP, P)
4:  Si manifiesto vacío
5 :     Salir
6 :  Para manifiesto [‘actualizar’ y ‘nuevos’]
7:      HTTP_GET
8:      Si manifiesto == ‘actualizar’
9:          Guarda backup
10:     Instala archivo
11: Para manifiesto [‘borrar’]
12:     Borra archivo
13: Si hay firmware en manifiesto
14:     Instala firmware
15: Guarda la versión instalada
16: Resetea el nodo
```

4.5. Provisioning y decommissioning

4.5.1. Provisioning

Como se ha comentado en secciones anteriores, para el provisioning se ha creado un conjunto de programas para el dispositivo que permita gestionar este módulo. Los programas podrán introducirse en cualquier dispositivo independientemente de sus características, ya que el dispositivo accede a la base de datos, obtiene los datos propios para el nodo elegido y se actualiza a través del módulo de OTA con los programas deseados por el usuario.

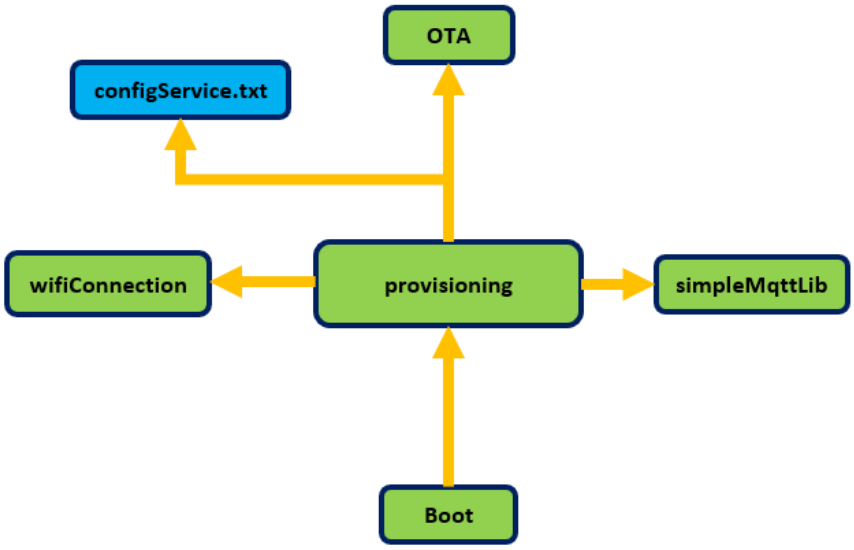


***Figura 25**. Diagrama programas dispositivo en provisioning.*

Como se ve en la Figura 25, tiene una estructura similar al dispositivo funcionando en modo activo, pero como elemento principal tiene el fichero provisioning. Contiene las funciones necesarias para poder adquirir los datos para el nodo en la base de datos. Con el fin de reducir el uso de memoria RAM, no se usa un servicio de acceso a base de datos en el nodo, si no que el servicio de acceso a la base de datos se incorporó en un servidor, el cual trata los datos y los envía al dispositivo una vez obtenidos.

Algoritmo 2: Provisioning dispositivo
Entrada: Configuración C
Salida: MAC dispositivo
1: Crea objeto OTA y cliente MQTT
2: Conecta cliente MQTT
3: Subscribe topic provisioning
4: Publica MAC
5: **Mientras** infinito
6: **Si** llega mensaje
7: Escribe C y la guarda en fichero
8: Realiza OTA

En la Figura 26, cuando el dispositivo se inicia y el programa provisioning.py entra en funcionamiento, el nodo vía MQTT siguiendo la secuencia de instrucciones definida en Algoritmo 2, pide al servidor que le entregue los datos del nodo correspondiente, además de entregarle la dirección MAC del dispositivo para que la almacene en la base de datos. La finalidad es saber qué información está asociada a cada dispositivo en la base de datos y no acceder a ella por otros. El servidor (funcionando como está definido en Algoritmo 3), a través del protocolo Mongo DB Wire, accede a la base de datos MongoDB, recoge los datos, los trata y los envía a través de MQTT al dispositivo.

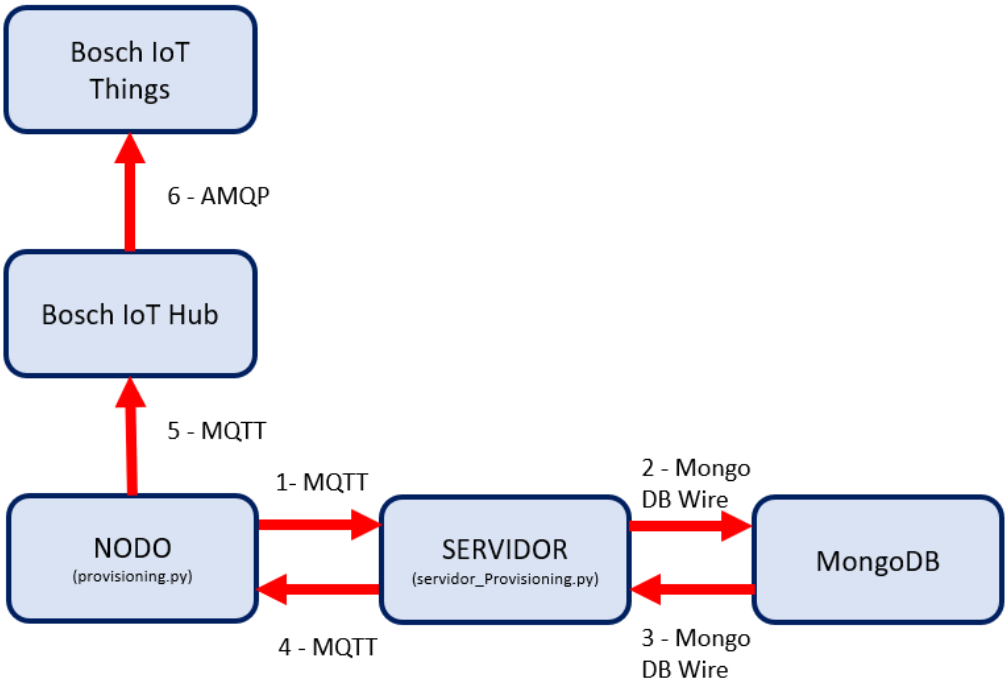


***Figura 26**. Diagrama arquitectura provisioning.*

Algoritmo 3: Provisioning/decommisioning servidor

Entrada: MAC, configuración C, Elección usuario E, lugar L, número dispositivo ID
Salida: C

```
1:   Crea y conecta cliente MQTT
2:   Subscribe topics
3:   Mientras infinito
4:      Si topic provisioning
5 :        Inicia base de datos
6 :        Si E
7:            Devuelve lista lugares
8:            Elige y recibe L
9:            Escribe MAC en BBDD
10:       En caso contrario
11:           Recibe L aleatorio
12:           Escribe MAC en BBDD
13:       Publica C(L)
14:     Si topic decommisioning
15:        Inicia base de datos
16:        Borra MAC de la BBDD de ID
```

Una vez que el dispositivo actualiza al software nuevo, con los parámetros obtenidos de la base de datos en su configuración, automáticamente se conecta a la plataforma Bosch IoT Suite. A través de Bosch IoT Hub, accede al módulo Things. Según la topología que se ha elegido para la organización de dispositivos, se conecta al dispositivo Gateway. Este, dependiendo del parámetro “topic”, crea un dispositivo virtual asociado a dicho Gateway.

telemetry/t5b0601f78fd24842a279bdd6c28929c8_hub/ucm.asset.example:Nodo1-via-solarGateway

En el ejemplo mostrado, toma la cadena de caracteres remarcadas en rojo como nombre del dispositivo virtual. Este dispositivo virtual, tomará las características propias del nodo, siendo la representación virtual del nodo en la plataforma, donde tendrá las características de irradiación, batería, lugar, etc.

4.5.2. Decommissioning

En el caso de que se quiera prescindir de un dispositivo, se ha implementado una fase de desmantelamiento (Figura 27), el cual se inicia desde el módulo Bosch IoT Things. A través de Messages con un subject específico (llamado “decommissioning”), se envía un mensaje al dispositivo. El nodo, al recibir el mensaje con el topic correspondiente, realiza una acción donde el dispositivo se pone en modo bajo consumo[9] de manera indefinida.

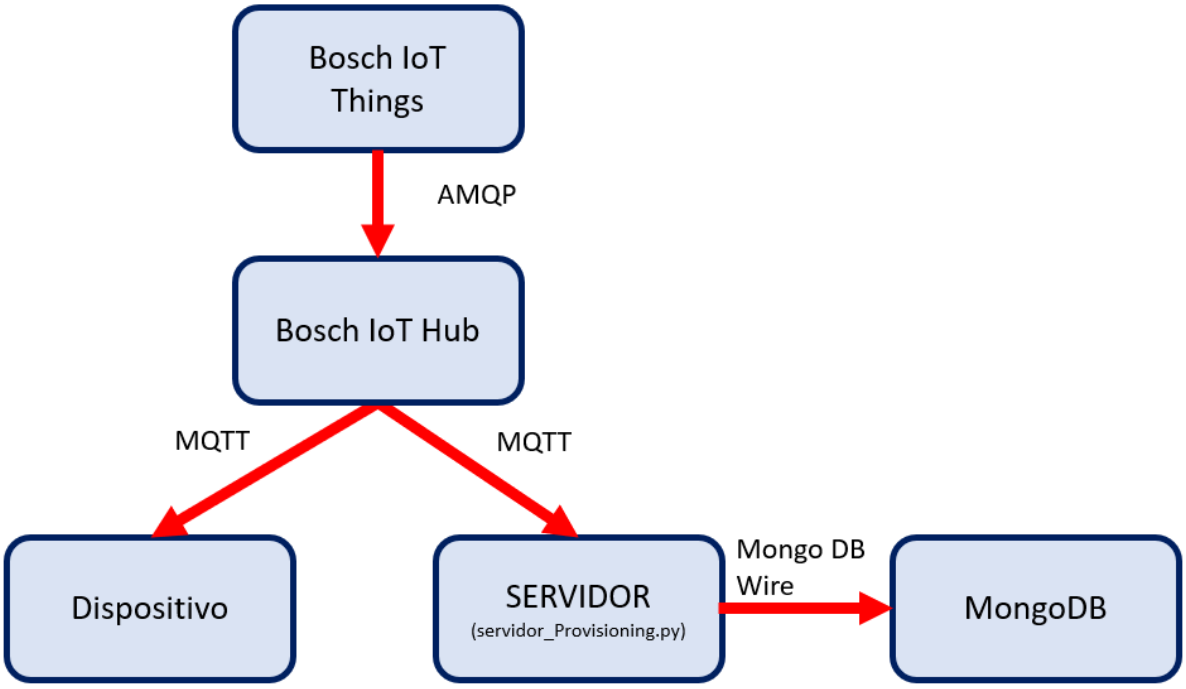


***Figura 27**. Arquitectura decommissioning.*

Al enviar el mensaje, el servidor (funcionando como en Algoritmo 3) tiene habilitada funciones para poder tratar este mensaje y realizar una acción que, en este caso, es una petición a la base de datos para que elimine la MAC del dispositivo desmantelado.

[9] Lo ideal sería poder apagar totalmente el dispositivo. MicroPython no tiene una funcionalidad para apagar totalmente, por lo que habría que diseñar un programa para poder implementar esta funcionalidad.

Capítulo 5 - **Solución para el dispositivo**

Tal y como se explicó en anteriores secciones, para la LoPy 4.0 se ha modificado el contenido del nodo para mejorar, añadir funcionalidades en los anteriores trabajos [4] y también para añadir nuevos programas que permitan alcanzar el objetivo establecido. A continuación, se explica la organización de los programas con sus distintas dependencias, además de los códigos y parámetros utilizados en este proyecto para el dispositivo. En la Figura 28 se ha representado la arquitectura del nodo para su normal desempeño con las distintas dependencias y llamadas entre programas.

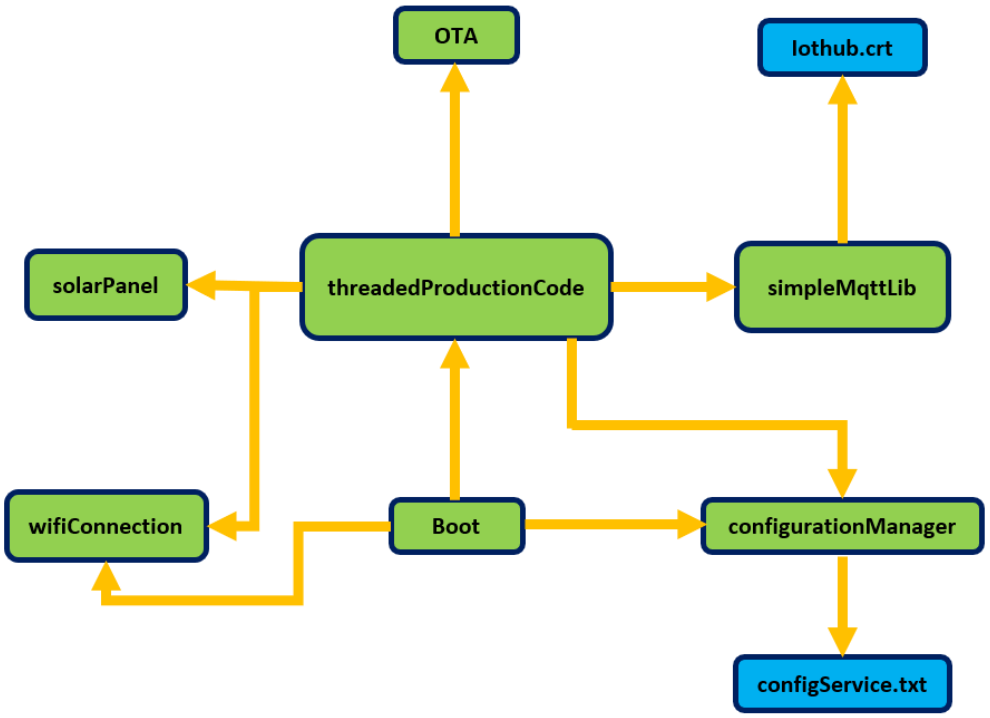


***Figura 28**. Diagrama programas placa.*

5.1. Boot

Siguiendo la directriz de Pycom, nombrándole boot, el programa se inicia automáticamente sin necesidad de un llamado; y es por tanto el punto de partida del funcionamiento de los demás programas.

1. Inicia el nodo, empezando por la conexión wifi, la cual hace a través del programa wifiConnection.
2. Con los protocolos RTC y NTP obtiene la hora, aunque hay que definir la zona horaria para tener la hora correspondiente al país de uso. Almacena posteriormente el valor en una variable que utiliza hasta que el programa entra en deepsleep.
3. Llama a la clase configurationManager, para abrir el fichero de configuración del dispositivo y en caso de no existir, crea una configuración predefinida.

4. Llama a la única función de este programa th_workingTime(), que define si debe entrar en modo activo o debe poner el nodo en modo bajo consumo (deepsleep), dependiendo de las horas de funcionamiento definidas. Si entra en modo activo, continúa y llama a la clase threadedProductionCode, donde está definido el funcionamiento del nodo en modo activo. Si no, entra en modo bajo consumo a través de la función deepsleep. Se puede ver en el Algoritmo 4, el funcionamiento del programa.

Algoritmo 4: Boot

Entrada: configuración C

Salida:

```
1:  Inicio conexión Wifi
2:  Intervalo tiempo activo O = C
3:  Inicia reloj T
4:  Si T dentro de O
5:      Calcula tiempo en actividad
6:      Inicia el modo activo
7:  En caso contrario
8:      Calcula tiempo de deepsleep
9:      Inicia el modo deepsleep
```

5.2. ThreadedProductionCode

Este programa contiene el modo activo del dispositivo y es el eje central del funcionamiento del nodo (Algoritmo 5). Llama a los programas encargados de tomar medidas de irradiación solar para ser enviados a través del protocolo MQTT a la plataforma IoT. Además, también gestiona los mensajes que el usuario envía con MQTT desde la plataforma. El programa llama al módulo _thread, encargado de crear los hilos o subtareas, que permite realizar tareas de forma paralela.

Algoritmo 5: Programa principal

Entrada: Fichero configuración C

Salida: datos sensores nodo

```
1:  Parámetros = C
2:  Crea objeto panel solar
3:  Crea conexión Wifi y conecta
4:  Crea cliente MQTT
5:  Si conexión Wifi está conectado
6:      Conecta el cliente MQTT con sesión persistente
7:  En caso contrario
8:      Reconecta WiFi
9:      Conecta el cliente MQTT con sesión persistente
10: Crea el nodo implícitamente en la plataforma IoT con los parámetros
11: Inicia los 3 hilos de funcionamiento
```

En el caso de este trabajo, tenemos 3 hilos:

- Th_sampling(): se encarga de llamar a las funciones del programa solarPanel, el cual obtiene los valores de la batería y de la irradiación. Esto lo hace en un bucle infinito, en un intervalo definido por el periodo de muestreo especificado por el usuario en la configuración (Algoritmo 5.1).

Algoritmo 5.1: Hilo muestreo

Entrada: Tiempo de muestreo Tm y horario de trabajo O
Salida: Nivel batería Vb y valor irradiación I

```
1:  Bucle infinito
2:      Lee la hora actual
3:      Si la hora actual en O
4:          Lee I
5:          Lee Vb
6:          Espera Tm
7:      En caso contrario
8:          Resetea el nodo
```

- Th_checkMessage(): realiza la operación de subscripción a un tema (topic) dado por el usuario en la configuración a través del cliente MQTT. Inserta una función callback, donde dependiendo del mensaje que reciba, activa esta función y realiza una acción determinada (Algoritmo 5.2).

Algoritmo 5.2: Hilo subscripción

Entrada: Mensaje MQTT plataforma (M)
Salida:

```
1:  Subscribe el tópico con qos=1
2:  Bucle infinito
3:      Si está conectado
4:          Espera a M
5:          Si M == { reset → reinicia la placa
                    { sleep → pone la placa en modo sleep
                    { conf − x → cambia la configuracion según x
                    { configuration → envía las configuraciones a la plataforma
                    { DoOTA → realiza una actualización de la placa
                    { decommissioning → desmantela el nodo
6:      En caso contrario
7:          Espera a reconexión Wifi
```

- Th_sending(): se encarga de empaquetar los datos obtenidos de los sensores en Th_sampling en formato JSON y enviarlo a la plataforma Bosch IoT Suite con el cliente MQTT. Realiza esta operación cada periodo de envío especificado por el usuario en la configuración (Algoritmo 5.3).

Algoritmo 5.3: Hilo envío

Entrada: Valores de los sensores muestreo Vb y I

Salida: Valor batería a enviar Vbe, valor irradiación a enviar Ie, mensajes globales Mo

1: Inicializa las variables
2: **Bucle infinito**
3: Calcula $\boldsymbol{V_{be}} = \frac{\sum_{i=0}^{Te/Tm} \boldsymbol{Vbi}}{\boldsymbol{Tm/Te}}$ $\boldsymbol{I_e} = \frac{\sum_{i=0}^{Te/Tm} \boldsymbol{Ii}}{\boldsymbol{Tm/Te}}$
4: **Si** está conectado
5: Publica valor Vbe, I, Mo con qos=1
6: **Si** hay valores Offline
7: Publica valores Offline con qos=1
8: **En caso contrario**
9: Intenta reconexión Wifi
10: Almacena valores Ve e Ie Offline para su envío

5.3. configurationManager

Tiene la función de gestionar la configuración del nodo. La configuración y la información exterior viene en formato JSON y se trata con la llamada a la clase ujson. A través de las distintas funciones que tiene, puede crear configuraciones nuevas, leer la configuración existente en configService.txt y acceder a parámetros de manera independiente.

5.4. wifiConnection

Este programa es el encargado de la gestión de la conexión wifi. Principalmente, hace uso del módulo de Python llamado *Network*, que proporciona controladores de red y enrutamiento. Esta clase, crea el objeto WLAN, el cual es el controlador utilizado, realiza funciones de escaneo de redes, conecta a la red y permite comprobar el estado de la conexión.

5.5. solarPanel

Es la clase encargada de leer los valores del ADC. Debido al diseño del circuito, se requiere hacer algunas configuraciones especiales en esta clase:

- Además de inicializar el ADC, también crea un objeto DAC en el PIN22, con una tensión de referencia de 0,5V , que es utilizada por distintas partes del circuito.
- Al empezar el trabajo, y en anteriores desarrollos se creyó que el pin P8 correspondía a la funcionalidad powerPin, el cual activa todos los sensores del módulo conectados a la placa de expansión. Durante el desarrollo de este trabajo, se comprobó que este valor corresponde al P12.
- El P18 se utiliza como una tensión de referencia para el panel de irradiación solar, el cual está conectado a P13.
- Por último, la medida del voltaje de la batería se encuentra en el P14.

Además de esta función de construcción del panel solar, tiene las funciones de acceso a los valores de voltaje de batería y del panel para los programas que lo requieran.

5.6. SimpleMqttLib

Es una librería cliente de MQTT, por lo que tiene todo lo necesario para gestionar la comunicación del protocolo MQTT. Realiza todo el proceso de comunicación con el broker, desde la creación de socket y desconexión con él hasta la subscripción y publicación de mensajes. Los mensajes que publica, son los correspondientes al voltaje de la batería y de la irradiación solar con una calidad qos 1. El nodo, se subscribe a todos los topics correspondientes a su ID de cliente con el que accede al broker, con una sesión persistente. El mensaje llega con un topic asociado y cada topic dentro del nodo, como se explica en el Algoritmo 5.2, realiza una función determinada.

5.7. Parámetros

La configuración del nodo se encuentra en el fichero configService.txt, y tiene todos los parámetros necesarios para la comunicación con la plataforma IoT, además de los relacionados con tiempos de trabajo. Los parámetros están definidos en formato JSON, de la siguiente manera:

- serverPlatform: es la dirección IP de la plataforma IoT escogida. En el caso de este TFM y como es común a todos los dispositivos corresponde a "mqtt.bosch-iot-hub.com".
- portPlatform: es el puerto utilizado para la conexión MQTT. Para el caso del funcionamiento normal en esta plataforma, al ser una conexión segura con TLS, se va a utilizar el puerto 8883.
- Usernameplatform: para la configuración escogida y explicada en los capítulos posteriores de la plataforma IoT, este parámetro es común a todos los nodos (en el caso de este proyecto). Se compone de:
    - Namespace: útil para separar elementos de distintas soluciones. En este caso y debido al número reducido de dispositivos solo se define uno. Ejemplo: “ucm.asset.example”.
    - Thing-name: es el nombre del dispositivo creado en la plataforma y al que llegan los datos en el módulo Bosch IoT Things. En este caso, se trata de un Gateway común a todos. Ejemplo: “solarGateway”.
    - Tenant_ID: una instancia correspondiente a la subscripción del servicio Bosch IoT Suite. Ejemplo: “ta4488d39e6e942329bb942d1a6b45bd5_hub”.

  Por lo que el usernamePlatform con las distintas partes que lo componen se define así: ucm.asset.example_solarGateway@ta4488d39e6e942329bb942d1a6b45bd5_hub
- Password: debido a que es una comunicación cifrada, se utiliza contraseña para acceder a la plataforma. La contraseña depende del dispositivo, y como en este caso es un Gateway, es común a todos los dispositivos.
- clientID: ID del cliente único para cada dispositivo. Este permite acceder a su información correspondiente en caso de subscripción en el broker. Se formaría como semejante al caso anterior, con la diferencia que el Thing-name corresponde al del dispositivo virtual creado. Ejemplo:

ucm.asset.example:Nodo1-via-solarGateway@ta4488d39e6e942329bb942d1a6b 45bd5_hub

- topic: como se ha explicado anteriormente, es una cadena de texto que le sirve al broker para filtrar los mensajes. Para este caso, toma relevancia ya que el topic es el parámetro encargado de realizar el provisioning en el módulo IoT Things, además de dirigir el mensaje al dispositivo virtual. "telemetry/t5b0601f78fd24842a279bdd6c28929c8_hub/ucm.asset.example:Nodo1-via-solarGateway".
- payloadTopic: parámetro necesario en la estructura del mensaje enviado al nodo virtual en la plataforma, siendo diferente para cada nodo. Ejemplo: "ucm.asset.example / Nodo1-via-solarGateway /things/twin/commands/modify"
- VERSION: número de versión del sistema.
- Coordinates: posición en coordenadas cartesianas donde se sitúa el dispositivo.
- Place: nombre del lugar donde se sitúa el dispositivo.
- fMuestreo: tiempo entre toma de datos de la placa solar.
- fEnvio: tiempo entre envío y envío.
- startingTime: tiempo de inicio del modo activo.
- stoppingTime: tiempo de inicio del modo reposo.
- SSID: identificación de la red wifi donde este situado el dispositivo.
- wifiPass: contraseña de la red wifi donde este situado el dispositivo.

Capítulo 6 - **Caracterización energética**

En el marco de proyectos que engloba este trabajo, inicialmente se utilizaba una LoPy 1.0, utilizando este dispositivo para el desarrollo de código y pruebas. Las características de estos dispositivos, comparadas con las de LoPy 4.0, especialmente en memoria, obligó a programar el código con un rendimiento óptimo, ya que el uso de certificados para asegurar la comunicación (además del uso de hilos) consumía excesiva memoria RAM, produciendo en numerosas ocasiones el reinicio o apagado del dispositivo. Aunque posteriormente se usó la LoPy 4.0, el desarrollo en unas condiciones desfavorables ha ayudado a conseguir programas que reduzcan el consumo de la batería.

La LoPy 4.0, trabaja principalmente de 2 maneras. La primera de ellas es el modo activo, donde se desarrollan todas las operaciones de toma de medidas y posterior envío a la plataforma. Entre tiempos de envío y de muestro, la placa se pode en modo espera, donde idealmente estaría en suspensión. El modo de bajo consumo, con la llamada a deepsleep, reduce el gasto energético durante las horas de poca irradiación solar.

En el laboratorio, utilizando el osciloscopio Agilent, se han tomado las medidas con un voltaje de 3.7 V. La alimentación se ha realizado por el puerto de la batería. El tiempo de muestreo del estado energético del Agilent es de 1 ms, con el fin de ver los cambios de manera precisa. Para la caracterización, se han diseñado programas sencillos escritos en MicroPython que emulan el comportamiento del dispositivo para las principales acciones que realiza.

6.1. Deepsleep

En modo deepsleep la placa entra en modo bajo consumo durante las horas nocturnas, con un tiempo definido por el usuario. Este modo, detiene la CPU y todos los periféricos, incluidas las interfaces de red. Se calcula el promedio total de las medidas tomadas, ya que normalmente debería ser un amperaje muy reducido. El código es sencillo, se llama al método deepsleep por tiempo indefinido.

```
def boot():
    machine.deepsleep()
```

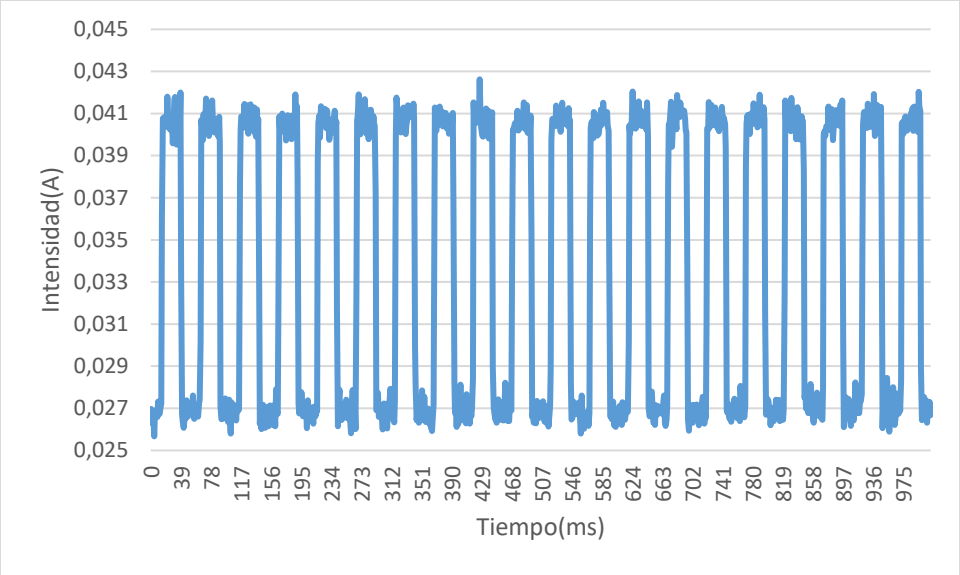


***Figura 29**. Consumo en deepsleep.*

Curiosamente, forma una onda periódica en vez de tomar un valor casi constante como se esperaba (Figura 29). El valor no es cercano a 0, consumiendo una media de 33,73 mA, lo que hace pensar que en Pycom, en el modo deepsleep, debe haber una anomalía que no reproduce un estado deepsleep 100%, ya que el consumo debería ser más reducido. Con el objetivo de ver si el problema provenía del hardware o era por el lenguaje de programación utilizado, se ha realizado la medida con el mismo programa escrito en lenguaje C. Como se puede ver en el Figura 30, la media del consumo es de 9,8 mA.

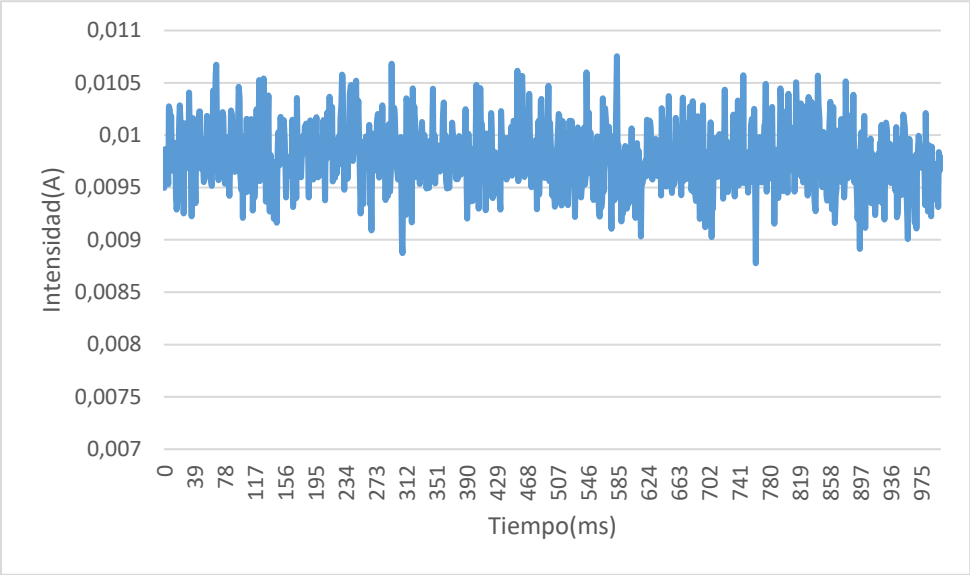


***Figura 30**. Consumo en deepsleep en lenguaje C.*

6.2. Envío

El envío refiere a la publicación de los datos de los sensores vía MQTT durante el modo activo. El código se compone de una conexión wifi, tiempos en modo bajo consumo para poder diferenciar mejor los distintos procesos, la creación de un cliente MQTT y su conexión.

```
def main(server="localhost"):
    conn = wifiConnection.Wifi_Connection()
    time.sleep(0.5)
    conn.connect(conn.ssid, conn.passw)

    time.sleep(0.5)
    c = MQTTClient("scr",server="broker.mqttdashboard.com")
    c.connect()
    while(1):
        for i in range(0, 2000):
            c.publish(b"foo_topic", b"hello")
        machine.deepsleep(1000)
```

Para el envío, con un periodo de muestreo tan pequeño es muy difícil poder mostrar varios periodos de tiempo si se usa el modo deepsleep, ya que produce un reinicio de la placa, por lo que se ha representado un periodo de esta honda en Figura 31.

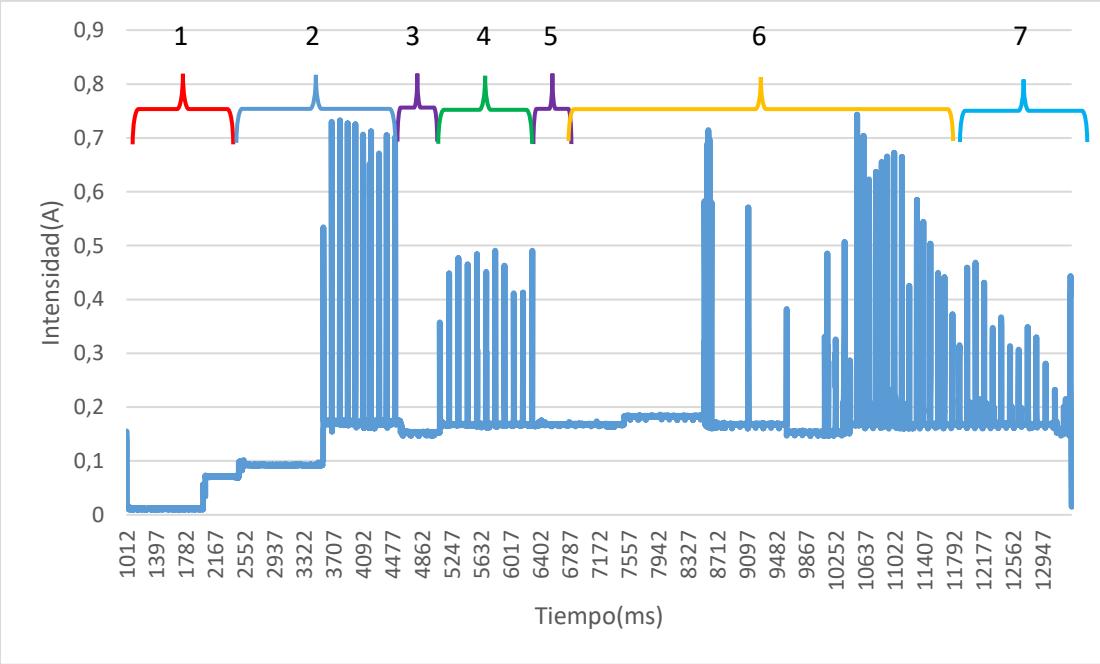


***Figura 31**. Proceso de inicio del dispositivo.*

1. En la primera parte tenemos el nodo en modo bajo consumo reiniciandose. Con un valor de 0,01 Amperios.
2. En la segunda, se tiene el inicio del nodo, desde el reset que realiza el deepsleep hasta el inicio de todos los componentes. La zona donde presenta picos muy altos, que abarca alrededor de un segundo, a pesar de los picos altos que llegan a 700 mA, son valores

muy puntuales. Extrayendo la media total de ese intervalo de tiempo, da un valor de 182,77 mA.

3. Tiempo de espera con la función time_sleep.
4. Posteriormente, hay una zona donde se inicia la conexión wifi del nodo.
5. Hay un margen de tiempo para esperar que la conexión tenga éxito, realizado con time_sleep.
6. Luego, se inicia la conexión MQTT.
7. Envío de paquetes MQTT en un periodo de tiempo más o menos de 2 segundos.

Para estar seguro de las partes, también se ha representado en Figura 32 el intento de la conexión wifi en un intervalo de tiempo. Se puede observar que presenta picos altos, pero muy puntuales, por lo que el consumo medio no es tan elevado.

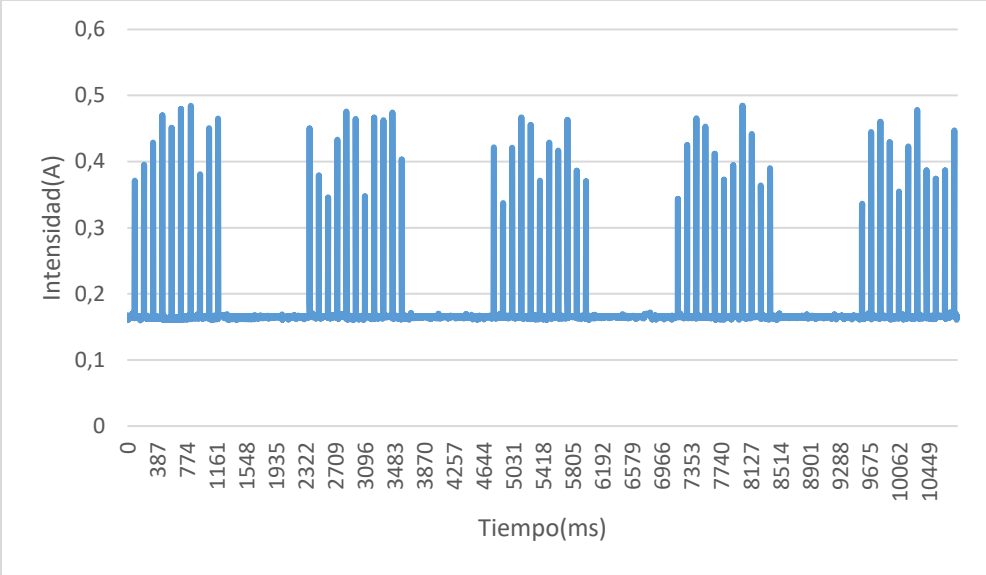


***Figura 32**. Conexión wifi.*

Para poder extraer mejor el consumo del nodo se ha representado periódicamente el consumo del envío tras una espera en time_sleep (Figura 33). No se ha utilizado el modo sleep de la biblioteca machine, el cual produce un estado de bajo consumo o suspensión (sin detener la CPU, como el modo deepsleep). Debido a una anomalía del código no reinicia la conexión wifi tras esta función, por lo que se utiliza el sleep de la biblioteca time, que no conlleva el mismo ahorro energético.

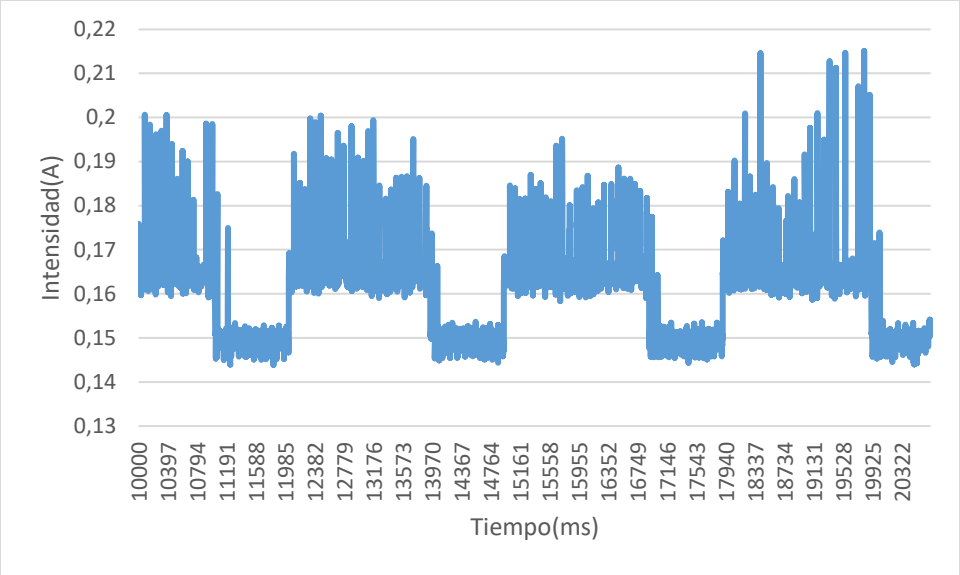


***Figura 33**. Envío de mensajes MQTT.*

6.3. Subscripción

Para la subscripción en la función se conecta a un cliente MQTT (sin TLS, utilizando el broker de Hive) para ver el consumo al estar solamente con un cliente activo. Por lo que una vez conectado al cliente y llamada a la función callback, se inicia un bucle infinito que espera mensajes MQTT.

```
def main(server="localhost"):

    conn = wifiConnection.Wifi_Connection()
    time.sleep(0.5)
    conn.connect(conn.ssid, conn.passw)

    time.sleep(0.5)
    c = MQTTClient("scr",server="broker.mqttdashboard.com")
    c.connect()

    def sub_cb(topic, msg):
        print("eh")

    c.set_callback(sub_cb)

    c.subscribe("mqtt/OTA")
    print("Connected")

    while 1:

        time.sleep(5)
```

Además, se han publicado algunos paquetes desde el cliente HiveMQTT para estudiar su consumo (Figura 34). El recibo de paquetes MQTT no es una tarea recurrente, pero conviene valorar todos los estados posibles.

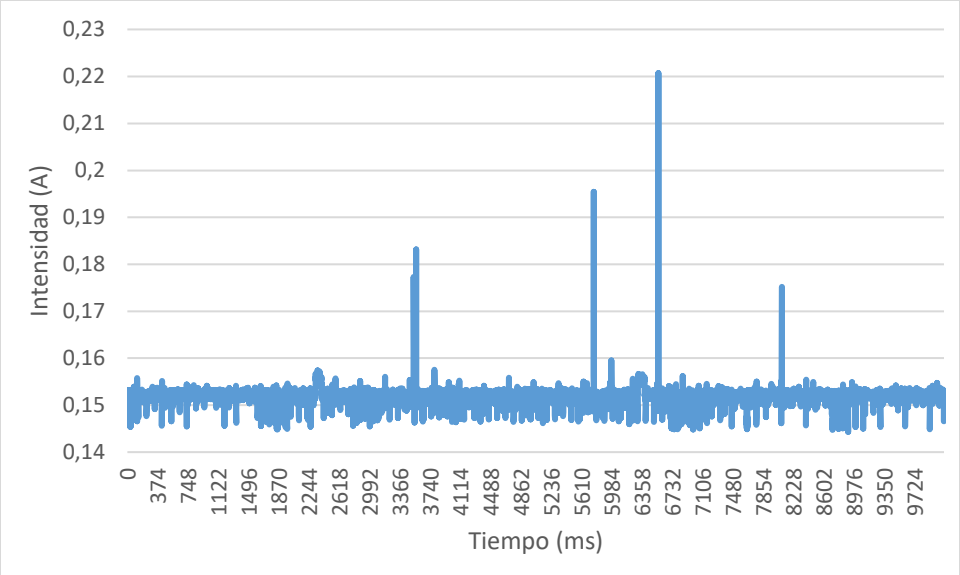


***Figura 34**. Recepción de paquetes MQTT.*

Se ha representado en la Figura 35, el consumo de la conexión con el broker y recepción de paquetes para el mismo código escrito en lenguaje C y, adicionalmente, con el modo sleep de la biblioteca machine. Se puede ver, que este modo sería de especial interés debido al ahorro en el consumo, con un voltaje medio de 27,01 mA, comparado con el programa escrito en MicroPython con el time_sleep, que consume una media de 150 mA (con wifi encendido).

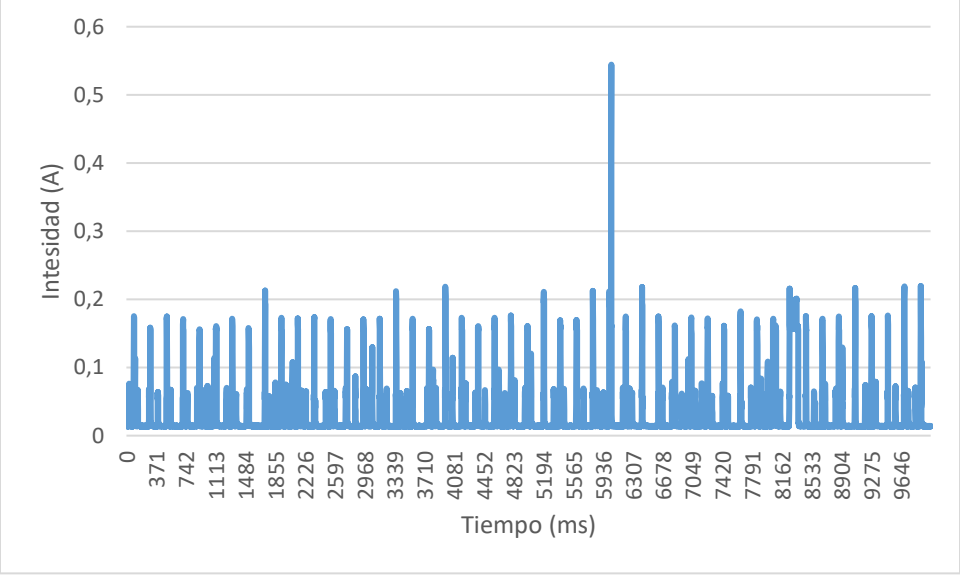


***Figura 35**. Recepción de paquetes en C.*

6.4. Lectura de valores ADC

El código se basa en lecturas del ACD para la toma de medidas de voltaje e irradiación solar, en un bucle infinito que se repite 1000 veces, para posteriormente entrar en modo bajo consumo (sin detener la CPU) durante un segundo.

```
def main():
    while 1:
        for i in range (0,1000):
            panelValue = panel.readAdc()
            v_battery=panel.readBattery()

        machine.sleep(1000)
```

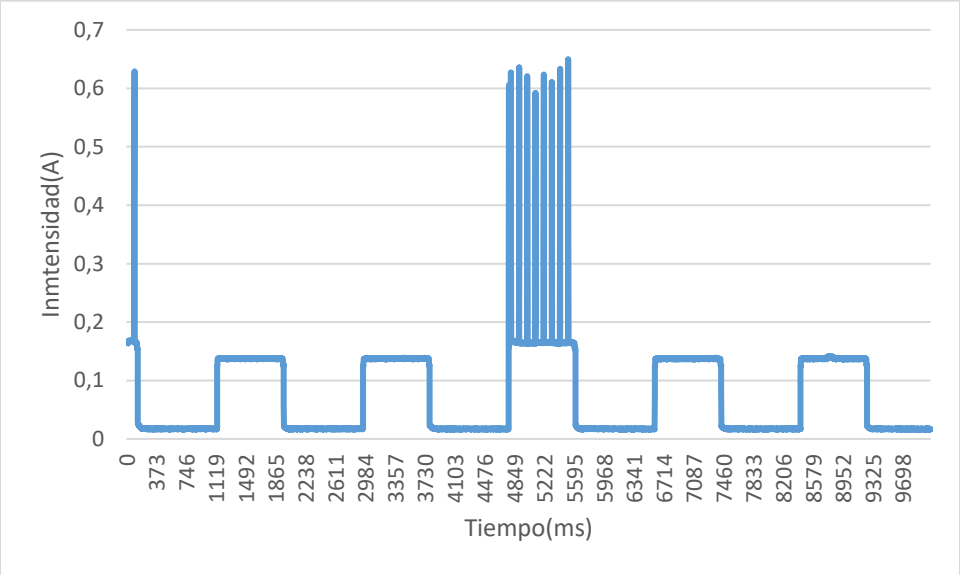


***Figura 36**. Lectura de valores ADC.*

Las medidas del ADC, presentaron momentos puntuales donde la intensidad aumentaba enormemente Figura 36. Para comprobar si estás ondas cuadradas eran anomalías o el reinicio del dispositivo, también se tomaron medidas de la lectura de valores con un intervalo de tiempo (de menor tamaño) entre las medidas.

```
def main():
    while 1:
        for i in range (0,1000):
            machine.sleep(50)
            panelValue = panel.readAdc()
            v_battery=panel.readBattery()

        machine.sleep(1000)
```

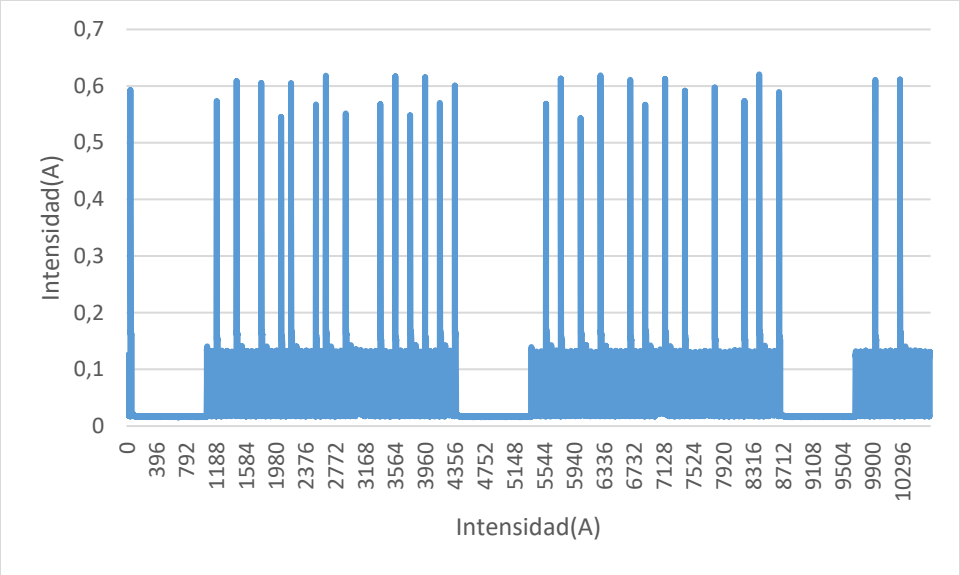


***Figura 37**. Lectura ADC con intervalos entre medidas.*

Al parecer, se trata de momentos específicos en el tiempo (Figura 37), de los cuales se desconoce el motivo. No es debido a un reinicio del dispositivo, ya que en ese caso los valores altos se repetirían constantemente en una reducida franja de tiempo, como pudo verse en Figura 31.

6.5. Consumo

Para calcular el consumo, se hace uso de las distintas medidas obtenidas en el laboratorio y calculadas en los apartados anteriores (Tabla 4). También se requiere de las especificaciones técnicas de las baterías, que en este caso corresponden a 800 mAh y 1200 mAh.

Estado	Intensidad media (mA)	Potencia media (mW)
Bajo consumo (deepsleep)	33,73	124,8
Envío	181,67	672,18
Recepción	182,33	674,62
Lectura ADC	150,08	555,3
Tiempo ocioso[10]	140	518

***Tabla 4**. Consumo del dispositivo según las acciones.*

Para ello, se va a realizar el cálculo de duración de la batería para las distintas configuraciones predefinidas en trabajos anteriores [4]. El método consiste en calcular el número de veces que se repite las acciones de muestreo y envío en una hora (3600 acciones). El número de recepciones se fija en un hipotético caso de que recibe 10 mensajes al día. El consumo de

[10] Valor de la función time_sleep (sin suspensión ni bajo consumo) con radio apagada.

espera es el número de acciones restante, que en este caso es tiempo en modo espera (Tabla 5).

	Configuración 1	Configuración 2	Configuración 3
Muestreo	1200(3s)	600(6s)	240(15s)
Envío	240(15s)	120(30s)	60(60s)
Recepción	10	10	10
Espera	2150	2870	3290

***Tabla 5**. Número de acciones en distintas configuraciones.*

6.5.1. Configuración 1

$$Intensidad\ promedio = 181{,}67 \times \frac{1200}{3600} + 182{,}33 \times \frac{10}{3600} + 150{,}08 \times \frac{240}{3600} + 140 \times \frac{2150}{3600}$$

Consumo medio según acciones: 146,25 mA

$$Tiempo\ de\ funcionamiento = \frac{800}{146{,}26} = 5\ horas\ 28\ minutos$$

$$Tiempo\ de\ funcionamiento = \frac{1200}{146{,}26} = 8\ horas\ 12\ minutos$$

6.5.2. Configuración 2

$$Intensidad\ promedio = 181{,}67 \times \frac{1200}{3600} + 182{,}33 \times \frac{10}{3600} + 150{,}08 \times \frac{240}{3600} + 140 \times \frac{2150}{3600}$$

Consumo medio según acciones: 143,19 mA

$$Tiempo\ de\ funcionamiento = \frac{800}{143{,}19} = 5\ horas\ 35\ minutos$$

$$Tiempo\ de\ funcionamiento = \frac{1200}{143{,}19} = 8\ horas\ 22\ minutos$$

6.5.3. Configuración 3

$$Intensidad\ promedio = 181{,}67 \times \frac{1200}{3600} + 182{,}33 \times \frac{10}{3600} + 150{,}08 \times \frac{240}{3600} + 140 \times \frac{2150}{3600}$$

Consumo medio según acciones: 141,48 mA

$$Tiempo\ de\ funcionamiento = \frac{800}{141{,}48} = 5\ horas\ 39\ minutos$$

$$Tiempo\ de\ funcionamiento = \frac{1200}{141{,}48} = 8\ horas\ 29\ minutos$$

6.5.4. Modo bajo consumo

Consumo 33,73 mA

$$Tiempo\ de\ funcionamiento = \frac{800}{33{,}73} = 23\ horas\ 43\ minutos$$

$$Tiempo\ de\ funcionamiento = \frac{1200}{33{,}73} = 35\ horas\ 35\ minutos$$

6.5.5. Análisis

Se debe tener en cuenta que durante el tiempo de funcionamiento del dispositivo está alimentado por el panel solar que permite recargar la batería y además alimentar el circuito. Se estudia el caso en un supuesto donde la situación meteorológica es desfavorable durante un día, y, por lo tanto, depende 100% de la batería para su funcionamiento.

	Batería 800 mAh	Batería 1200 mAh
Configuración 1	5 horas 28 min	8 horas 12 min
Configuración 2	5 horas 35 min	8 horas 22 min
Configuración 3	5 horas 39 min	8 horas 29 min
Modo bajo consumo	23 horas 43 min	35 horas 35 min

***Tabla 6**. Tiempo máximo de trabajo.*

Según se observa en la Tabla 6, las distintas configuraciones de tiempos de muestreo y envío no influyen excesivamente en los tiempos de trabajo de las distintas baterías. Esto se debe al hecho de no poder poner el dispositivo en modo suspensión entre medidas, lo que produce que el tiempo en espera tenga un consumo muy elevado. El tiempo de funcionamiento del dispositivo es de 8 horas, por lo que en una situación en la que la batería y el circuito no puedan alimentarse convendría invertir en las baterías de 1200 mAh.

Capítulo 7 - **Conclusiones y trabajo futuro**

7.1. Conclusiones

A lo largo de este trabajo, se ha estudiado y realizado la implementación de un sistema IoT, donde se ha configurado desde los dispositivos que se instalarán en las azoteas de distintos puntos de la universidad, hasta la plataforma IoT. Además, se ha realizado el estudio energético del dispositivo, lo que nos lleva a extraer una serie de resultados y conclusiones.

Para el código del dispositivo, ha podido completarse con éxito todos los objetivos. Se mejoró notablemente el rendimiento de los programas, especialmente las conexiones MQTT, además de implementarse el provisioning/decommissioning, las actualizaciones online OTA y una comunicación bidireccional de dispositivo a plataforma. Adicionalmente, la comunicación se produce de forma segura a través del protocolo criptográfico TLS.

Se comprobó que a pesar de las facilidades que programar en MicroPython proporciona, el gasto en ciertos escenarios, como el de bajo consumo o en tiempo ocioso, presenta una serie de anomalías que consumen enormemente la batería (en comparación con otros lenguajes de programación, como C). Que en modo bajo consumo el gasto energético sea elevado es importante y desfavorable, ya que representa el 66,66% del tiempo de funcionamiento del dispositivo en un día.

La plataforma Bosch IoT Suite ha permitido llevar a cabo los objetivos planteados para este proyecto. La puesta en marcha de los diferentes módulos no ha sido una tarea fácil, incorpora mucha documentación, pero no para casos reales. Además, hay ciertos módulos de trabajo que no han podido llegar a completarse debido a la escasez de información para un caso práctico, por lo que el tiempo que debía emplearse para aprender estos módulos era mucho mayor que implementarlos como módulos ajenos a la plataforma.

Utilizar el protocolo MQTT ha permitido un desarrollo más fluido, debido a que el hardware con el que se ha trabajado incorpora bibliotecas o pueden encontrarse fácilmente clientes MQTT en la web. También ha posibilitado tener una conexión ligera, segura y con tolerancia a errores, pudiendo recuperar en todo momento cualquier paquete que se vaya a enviar o recibir.

Para los resultados obtenidos en la caracterización energética, sin tener en cuenta la carga y la alimentación del circuito por parte de la placa solar, convendría decantarse por una batería que permita su funcionamiento en condiciones meteorológicas desfavorables al menos durante un día entero. Hasta que no se solucione los distintos problemas que posee MicroPython con el bajo consumo y el tiempo ocioso, conviene usar la batería de 1200 mA para asegurarse la total autonomía del dispositivo.

7.2. Trabajo Futuro

De las conclusiones obtenidas, se puede observar que hay todavía trabajo a realizar en esta serie de proyectos, que podrían permitir un funcionamiento más eficiente tanto a nivel energético como de implementación.

Hay una serie de módulos (Figura 15) que todavía no se han desarrollado, debido a la falta de tiempo y a que no era el principal objetivo de este proyecto. Corresponden tanto el módulo de machine learning como el análisis de resultados. En Trabajos anteriores, hay una serie de proyectos dedicados a machine learning para la predicción de datos de irradiación solar. Se espera que, en un futuro, cuando los nodos ya estén funcionando al 100% en las azoteas, se pueda obtener un conjunto de datos suficientemente grande como para incorporar este módulo. El módulo de análisis de resultados podría ser interesante para avisar al usuario cuando el nivel de batería esta excesivamente bajo, cuando se vaya a apagar o cuando los datos de irradiación son anómalos.

Sería adecuado poder llevar a cabo un estudio más profundo a cerca de la plataforma Bosch, y ver como poder implementar el provisioning/decommissioning y OTA en su Suite, en vez de tenerlos de manera independiente, aprovechando así todo el potencial de esta plataforma. Es decir, poder incorporar los módulos llamados: Rollouts y Manager.

Se puede mejorar el método de provisioning, debido a que el método utilizado actualmente se basa en el deber del dispositivo en conectarse al servidor para obtener los archivos. Lo ideal, sería tener los distintos nodos funcionando como punto de acceso y que el servidor se conecte a ellos, para así proporcionarle todos los archivos. Esto evitaría problemas de gestión como, por ejemplo, el cambio de IP si se utiliza un servidor diferente (habría que cambiar la IP del servidor en todas los nodos).

Actualmente OTA puede realizarse dispositivo a dispositivo, no de manera masiva. No se debe al módulo OTA implementado, si no debido a que no se ha encontrado una manera de poder mandar un mensaje con un topic común desde Bosch IoT Things a todas los nodos. Entonces, en un futuro convendría poder enviar un solo mensaje para poder actualizar todos los dispositivos a la vez (si el usuario lo requiere).

Conviene cambiar en el código, cuando los problemas con MicroPython se solucionen, los modos de espera por modos de suspensión, con el objetivo de reducir el consumo de la batería entre medidas. También, podría ser interesante cambiar de lenguaje de programación, ya que por ejemplo el modo bajo consumo (deepsleep) en MicroPython consume mucho más que en C.

Capítulo 8 - **Introduction**

The generation of electrical energy has been one of the greatest challenges of mankind in recent times. Today's society, in developed countries, is almost completely dependent on electric power, at work, at home, in public transport, etc. From the moment someone gets up to the moment they go to sleep, they almost automatically and unconsciously rely upon electricity.

The exponential growth of population, the industrialization of developing countries and third world countries trying to offer a better quality of life to their citizens has increased energy consumption over the past few years (Figure 1).

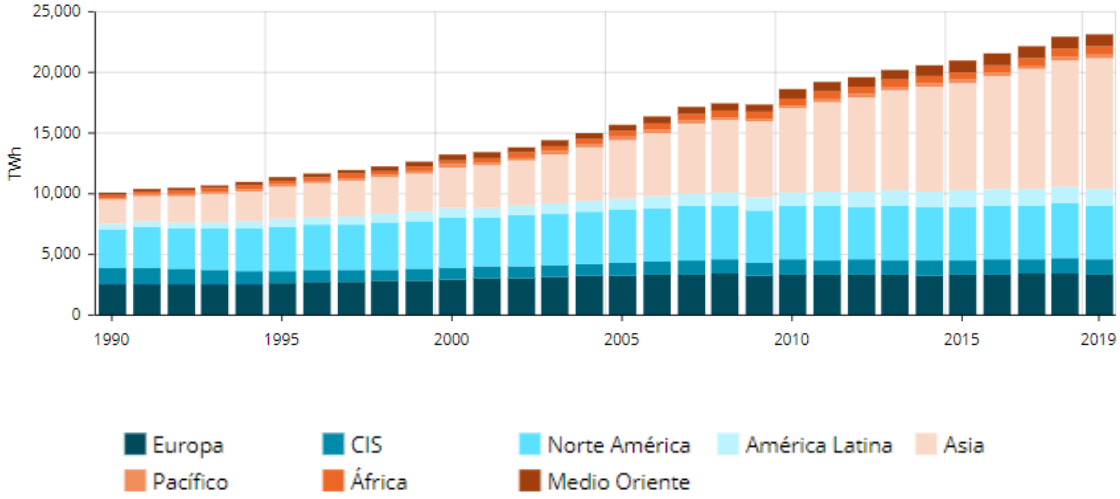


Figure 1. Growth in energy consumption [1].

Currently, the major problem is that nearly the vast majority of the energy produced worldwide depends on fossil fuels. They have a very negative environmental impact, either because of the CO_2 generated by the burning of fuels, increasing the greenhouse effect and pollution in cities; or because of the use of uranium in nuclear plants, which generates radioactive waste that is difficult to treat and degrade.

The investment in renewable energy sources has exponentially increased, where resources are unlimited and their use produces fewer environmental consequences. For this reason, large nations have increasingly focused their attention on the technological development of the latter, since once developed, it would provide energy independence from countries providing fossil fuels. Currently the world consumption of electric power depends on fossil fuels, with energy from renewable sources being very small compared to the others (Figure 2).

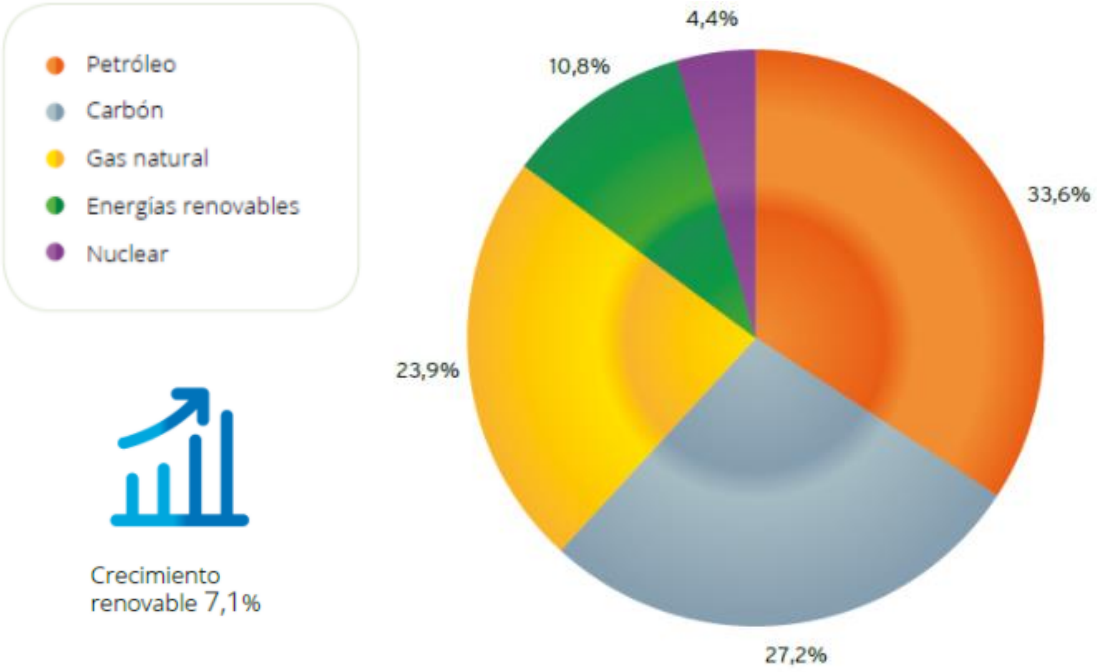


Figure 2. Percentage use of different energy sources [2].

Solar energy has been growing the most in recent years in terms of production, due to the high availability of solar energy and how little it is currently exploited. A negative point is that it cannot satisfy customer demand continuously since it depends on meteorological factors and, additionally, it cannot provide electric energy during the night. It also requires large spaces for installation of plates in areas of high demand and its storage would depend on batteries, which have a problem of treatment and subsequent recycling.

One solution to the problem of the unpredictability of meeting demand with solar energy is through predictive modeling of solar irradiance. Through machine learning and the collection of solar irradiance data, a model could be developed to predict solar irradiance in the short term.

8.1. Motivation

The growing world demand, the possession of fossil fuel resources by a few countries, the problems and political influences that this entails and the high environmental impact of the use of current energy production, forces to continuously investing and working in the line of renewable energies. Therefore, this thesis focuses on the prediction of solar energy in the short term, with the aim of reducing the demand in the future of the most polluting energies.

The advent of IoT in the current Industry 4.0, makes available to the current energy technologies a number of advantages that are still in the early stages, with a wide range of studies to be carried out and advantages to exploit. With IoT technology, a huge amount of data can be obtained for the desired application, and with it, mathematical models that allow us to predict the future in the context used.

Therefore, we intend to expand the area of knowledge in the deployment of sensor networks fully monitored by the user, the use of different IoT technologies such as platforms for device control and to continuously obtain solar irradiation data in plots of a chosen location. With this, a sufficiently large dataset could be obtained over time to be able to make use of machine learning and create predictive models.

8.2. Targets

The idea of this project is the development of a sensor network to be deployed on the campus of the Complutense University of Madrid, managed through an IoT platform, in order to obtain solar irradiation values at different points of the university through photovoltaic cells. Therefore, there are two main objectives, the development of code for the panels which are installed at the university, and also the implementation of an IoT platform for the management and monitoring of the nodes. The following sub-objectives have been proposed in order to achieve the targets and to have a better organizational scheme:

- Optimization of the current code developed in the works carried out since 2017.

- Identification of nodes independently (use of open standards).

- Implementation of wireless update (OTA) allowing software and firmware updates of the devices.

- Implementation of (semi) automatic provisioning of new devices and decommissioning of devices.

- Set up infrastructure to monitor data, control devices and perform updates.

- Through the platform, availability to carry out analysis of the results obtained.

- Achieving energy characterization of the devices in order to obtain an energy profile that allows acquiring or designing the hardware in an efficient way.

8.3. Work Plan

The first stage of the work is focused on conducting an in-depth study of the work carried out in the framework of the projects in which the work is developed, in order to extract relevant information and to be able to continue along the lines of previous projects. Once the code from previous studies has been processed, improvements and new implementations can be implemented to achieve the defined objective.

The second stage includes the search for the appropriate IoT platform for this project, which would allow to obtain all the proposed modules and their correct development. Once chosen, the documentation corresponding to the choice must be studied and the proposed services must be implemented in the appropriate module. In cases where it is not possible or it has not been possible to implement it on the platform, different solutions for its implementation might be sought.

Finally, an energy study of the device is implemented for the most recurrent tasks, in order to choose the appropriate hardware. This study is carried out in the laboratory of the Faculty of Physical Sciences of the Complutense University of Madrid. For each task, the power consumption of the board is defined and the battery life time is calculated for different scenarios.

Capítulo 9 - **Conclusions and future work**

9.1. Conclusions

Throughout this project, the implementation of an IoT system has been studied and carried out, where the devices to be installed on the rooftops of different points of the university, as well as the IoT platform, have been setup. In addition, the energy study of the device has been carried out, which leads us to draw a series of results and conclusions.

For the device code, all the objectives have been successfully completed. Program performance was significantly improved, especially MQTT connections. Furthermore, the services of provisioning/decommissioning, online OTA updates and bidirectional device-to-platform communication were also implemented. Additionally, communication is secure via the TLS cryptographic protocol.

It was found that despite the simplicity that programming in MicroPython provides, the expense in certain scenarios, such as low power, presents a number of anomalies that consume the battery enormously (compared to other programming languages, such as C). The fact that in low-power mode the energy consumption is high is important and unfavorable since it represents 66.66% of the operating time of the board in a day.

The Bosch IoT Suite platform has made it possible to achieve the objectives set for this project. The implementation of the different modules has not been an easy task, as it involves a lot of documentation, but not for real cases. In addition, there are certain work modules that could not be completed due to the scarcity of information for a practical case, so the time that had to be spent to learn these modules was much greater than implementing them as modules outside the platform.

Using the MQTT protocol has allowed a smoother development, because the hardware we have been working with has built-in libraries or MQTT clients can be easily found on the web. It has also made it possible to have a lightweight, secure, and error-tolerant connection, being able to recover any package to be sent or received at any time.

For the results obtained for the energy characterization, without taking into account the load and the power supply of the circuit by the solar panel, it would be advisable to opt for a battery that allows operation in unfavorable weather conditions for at least a whole day. Until MicroPython's various problems with low power consumption are solved, it is advisable to use the 1200 mA battery to ensure full autonomy of the board.

9.2. Future Work

From the conclusions obtained, we can conclude that there is still work to be done in this series of projects, which could allow a more efficient operation both in terms of energy and implementation.

There are a number of modules (Figure 15) that have not yet been developed, due to lack of time and because it was not the main objective of this project. They relate to both the machine

learning module and the results analysis. In previous studies, there have been a number of projects dedicated to machine learning for the prediction of solar irradiance data. It is expected that, in the future, when the panels are already operating at 100% on the rooftops, a sufficiently large data set can be obtained to incorporate this module. The results analysis module could be interesting to warn the user when the battery level is too low, when it is going to be turned off or when the irradiation data are anomalous.

It would be appropriate to carry out a deeper study about the Bosch platform, and see how to implement provisioning/decommissioning and OTA in the platform, instead of having them independently, thus taking advantage of the full potential of this platform. In other words, to be able to incorporate the modules called: Rollouts and Manager.

The provisioning method can be improved, because the method currently used is based on the duty of the board to connect to the server to get the files. Ideally, it would be better to have the boards working as an access point and have the server connect to the boards, thus providing it with all the files. This would avoid management problems such as, for example, changing IP if a different server is used (you would have to change the server IP on all boards).

Currently OTA can be done device by device, not in bulk. This is not due to the implemented OTA module, but because no way has been found to send a message to a common topic from Bosch IoT Things to all boards. So, in the future it would be convenient to be able to send a single message to update all devices at the same time (if the user requires it).

It would be convenient to change in the code, once the problems with MicroPython are solved, the standby modes for sleep modes, in order to reduce the battery consumption between steps. Also, it might be interesting to change the programming language, since for instance the deepsleep mode in MicroPython consumes much more than in C.

Bibliografía

[1] ENERDATA, «Consumo nacional de electricidad,» 2021. [En línea]. Available: https://datos.enerdata.net/electricidad/datos-consumo-electricidad-hogar.html.

[2] APPA Renovables, «Renovables en el mundo y en Europa,» 2021. [En línea]. Available: https://www.appa.es/energias-renovables/renovables-en-el-mundo-y-en-europa/.

[3] J. J. Brun, Septiembre 2018. [En línea]. Available: https://eprints.ucm.es/id/eprint/49433/1/Memoria_Jorge_Jarne.pdf.

[4] L. Giordano, «Informe de prácticas,» Madrid, 2020.

[5] Sigfox, «SIGFOX,» 2021. [En línea]. Available: https://www.sigfox.com/en.

[6] LoRa Alliance, «LoRaWAN,» 2021. [En línea]. Available: https://lorawan.es/.

[7] R. P. García, «Evaluación de LoRa/LoRaWAN para escenarios de Smart City,» 2017. [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/100922/memoria.pdf.

[8] Bluetooth, «Bluetooth Technology Overview,» 2021. [En línea]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/.

[9] ADSL zone, «Wifi,» 10 Septiembre 2020. [En línea]. Available: https://www.adslzone.net/reportajes/wifi/2-4-5-ghz.

[10] C. Bormann, «CoAP,» 2016. [En línea]. Available: http://coap.technology/.

[11] P. D. Malleiro, «Desarrollo y prueba de una pasarela CoAP para el internet de las cosas,» 2015. [En línea]. Available: http://castor.det.uvigo.es:8080/xmlui/bitstream/handle/123456789/27/TFG%20Pablo%20dominguez%20Malleiro.pdf?sequence=5&isAllowed=y.

[12] MQTT, «MQTT: The Standard for IoT Messaging,» 2020. [En línea]. Available: https://mqtt.org/.

[13] HiveMQ, «Quality of Service 0,1 & 2 - MQTT Essentials: Part 6,» 16 Febrero 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

[14] MDN contributors, «Generalidades del protocolo HTTP,» 13 Junio 2021. [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Overview.

[15] OASIS, «The problem we solve,» 2021. [En línea]. Available: https://www.amqp.org/product/solve.

[16] IONOS, «AMQP: conoce el Advanced Message Queuing Protocol,» 03 Mayo 2019. [En línea]. Available: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/advanced-message-queuing-protocol-amqp/.

[17] Pycom, «LoPy4,» 2020. [En línea]. Available: https://pycom.io/product/lopy4/.

[18] IXYS, «High Efficiency SolarMD,» 2010.

[19] Mouser electronics, «Adafruit 200,» 2021. [En línea]. Available: https://www.mouser.es/ProductDetail/Adafruit/200/?qs=sGAEpiMZZMsMyYRRhGMFNh0UmQdKZ6yJQn2PF3%2FaB1Y=.

[20] Electronic Components, «LP103040 CELLEVETIA BATTERIES,» 2021. [En línea]. Available: https://www.tme.eu/es/details/accu-lp103040_cl/baterias/cellevia-batteries/.

[21] Electronic Components, «l503448 CELLEVETIA BATTERIES,» 2021. [En línea]. Available: https://www.tme.eu/es/details/accu-lp503448_cl/baterias/cellevia-batteries/l503448/.

[22] Bosch, «Documentation,» 2021. [En línea]. Available: https://developer.bosch-iot-Suite.com/documentation/.