



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Ingeniería de Computadores

TRABAJO FIN DE GRADO

Despliegue y evaluación de rendimiento de una LP-WAN
basada en LoRa de dispositivos neutrales en energía en un
escenario de Agricultura Inteligente

Daniel Ballesteros Almazán

Julio, 2020



UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

Tecnologías y Sistemas de la Información

Ingeniería de Computadores

TRABAJO FIN DE GRADO

**Despliegue y evaluación de rendimiento de una LP-WAN
basada en LoRa de dispositivos neutrales en energía en
un escenario de Agricultura Inteligente**

Autor: Daniel Ballesteros Almazán

Tutor: María Soledad Escolar Díaz

Julio, 2020

Despliegue y evaluación de rendimiento de una LP-WAN basada en LoRa de dispositivos neutrales en energía en un entorno de Agricultura Inteligente

© Daniel Ballesteros Almazán, 2020

Este documento se distribuye con licencia CC BY-NC-SA 4.0. El texto completo de la licencia puede obtenerse en <https://creativecommons.org/licenses/by-nc-sa/4.0/>.

La copia y distribución de esta obra está permitida en todo el mundo, sin regalías y por cualquier medio, siempre que esta nota sea preservada. Se concede permiso para copiar y distribuir traducciones de este libro desde el español original a otro idioma, siempre que la traducción sea aprobada por el autor del libro y tanto el aviso de copyright como esta nota de permiso, sean preservados en todas las copias.

Este texto ha sido preparado con la plantilla \LaTeX de TFG para la UCLM publicada por [Jesús Salido](#) en GitHub¹ y Overleaf² como parte del curso « *\LaTeX esencial para preparación de TFG, Tesis y otros documentos académicos*» impartido en la Escuela Superior de Informática de la Universidad de Castilla-La Mancha.



¹https://github.com/JesusSalido/TFG_ESI_UCLM

²<https://www.overleaf.com/latex/templates/plantilla-de-tfg-escuela-superior-de-informatica-uclm/phjgscmfqtsw>

TRIBUNAL:

Presidente: _____

Vocal: _____

Secretario: _____

FECHA DE DEFENSA: _____

CALIFICACIÓN: _____

PRESIDENTE

VOCAL

SECRETARIO

Fdo.:

Fdo.:

Fdo.:

*A mi familia
Por estar ahí en los momentos más difíciles*

Resumen

Ante el aumento de la población mundial hasta cerca de los 9.000 millones de personas que habrá en 2050, la agricultura se enfrenta a unos de los mayores retos de su historia, ya que deberá producir alrededor de un 60 % más de alimentos que en la actualidad. Para ello, se busca reducir y mejorar el uso de los recursos naturales mediante la aplicación de las tecnologías de la información.

Este Trabajo Fin de Grado propone el despliegue de una red Low Power-Wireless Area Network de dispositivos *energy-harvesting* en el contexto de agricultura inteligente. Los dispositivos usan paneles solares para la recolección de energía, recogen datos relacionados con el medio ambiente a través de sensores y transmiten esos datos a través de un módulo de comunicaciones LoRa hacia una plataforma de *Internet of Things*, FIWARE, donde los datos pueden ser almacenados de forma permanente para que terceros puedan consultarlos, procesarlos y analizarlos, con el objetivo final de incrementar el conocimiento que los agricultores tienen sobre sus cultivos y ayudar a los agricultores a tomar decisiones basadas en la información recogida. Los dispositivos *energy-harvesting* emplean un algoritmo neutral en energía para que, en combinación con el panel solar, puedan generar la energía eléctrica necesaria y adaptar la ejecución de las aplicaciones a la producción de energía, para permitirles operar ininterrumpidamente en el tiempo.

Abstract

With the world's population increasing to nearly 9 billion people by 2050, agriculture faces one of the greatest challenges in its history, as it will have to produce about 60 percent more food than it does today. To this end, it seeks to reduce and improve the use of natural resources through the application of information technologies.

This End of Degree Project proposes the deployment of a Low Power-Wireless Area Network of devices *energy-harvesting* in the context of smart agriculture. The devices use solar panels for energy collection, collect data related to the environment through sensors and transmit those data through a LoRa communication module to FIWARE, an *Internet of Things* platform where the data can be permanently stored in order to third parties may consult, process and analyze them, with the final objective of increasing the knowledge that farmers have about their crops and helping farmers to make decisions based on the information collected. The *energy-harvesting* devices use an energy-neutral algorithm so that, in combination with the solar panel, they can generate the necessary electrical energy and adapt the execution of the applications to the production of energy, to allow them to operate uninterruptedly over time.

AGRADECIMIENTOS

Parece mentira que tras cuatro años en esta Escuela, esta etapa haya llegado a su fin. Aunque el fin de esta etapa signifique un punto y final, prefiero pensar que es un punto y seguido. Tras estos cuatro años increíbles que he vivido, lo único que puedo hacer es agradecer a la gente que ha estado a mi alrededor durante todo este tiempo.

Primero de todo, me gustaría agradecer a las personas más importantes de mi vida: mi familia. Para mí, ellos lo son todo. Son mi mayor fuente de ilusión y energía. Quiero dar las gracias a mis padres, *Vicen* y *Ramón*, por todo el apoyo, amor y compromiso que me han dado desde siempre, y del cuál nunca les podré agradecer lo suficiente. Gracias a mi hermano *Jorge*, por estar ahí siempre que me ha echo falta y por todos los momentos que hemos vivido juntos.

No puedo olvidarme de mis compañeros de Ingeniería de Computadores, con los que se ha forjado una gran amistad. Gracias a ellos, ir a clase se hacía mucho más ameno y divertido. Por más momentos en la cafetería de la ESI hablando de nuestra vida y riendo juntos.

Tampoco puedo olvidarme de mis amigos de la residencia universitaria *Santo Tomás de Villanueva*, pero principalmente de *Domingo*, el cual es y será el mejor compañero de habitación y amigo que alguien puede tener. Gracias por todos los momentos que hemos vivido juntos estos años, y que sigamos jugando al fútbolín después de cada comida.

Por último, pero no menos importante, quiero agradecer a mi tutora de proyecto, *Soledad*, por ofrecerme la oportunidad de realizar este proyecto tan interesante a la par que bonito, y por ayudarme cada vez que lo necesitaba.

Daniel Ballesteros Almazán

Ciudad Real, 2020

ÍNDICE GENERAL

| | |
|--|--------------|
| Resumen | IX |
| Abstract | XI |
| Agradecimientos | XIII |
| Índice de figuras | XVII |
| Índice de tablas | XIX |
| Índice de listados | XXI |
| Índice de algoritmos | XXIII |
| 1. Introducción | 1 |
| 1.1. Motivación | 1 |
| 1.2. Estructura del documento | 2 |
| 2. Objetivo | 3 |
| 2.1. Objetivo general | 3 |
| 2.2. Objetivos parciales | 3 |
| 2.2.1. Prototipado de un conjunto de dispositivos finales | 3 |
| 2.2.2. Implementación de un algoritmo neutral en energía | 3 |
| 2.2.3. Implementación de un protocolo de comunicación basado en LoRa | 4 |
| 2.2.4. Implementación de un protocolo de comunicaciones basado en REST entre el gateway y la plataforma FIWARE | 4 |
| 2.2.5. Despliegue de la red en entorno real | 4 |
| 2.2.6. Evaluación de la red mediante un experimento real | 4 |
| 2.2.7. Publicación del <i>dataset</i> generado en un repositorio público en <i>GitHub</i> | 4 |
| 3. Metodología | 5 |
| 3.1. Elección y breve descripción de la metodología | 5 |
| 3.2. Gestión del proyecto | 6 |
| 3.2.1. Aplicación | 6 |
| 3.3. Marco tecnológico | 6 |
| 3.3.1. Herramientas hardware | 6 |
| 3.3.2. Herramientas para la gestión de proyectos | 7 |
| 3.3.3. Herramientas, lenguajes y tecnologías para el desarrollo de software | 7 |
| 3.3.4. Herramientas para la elaboración de la documentación | 7 |
| 4. Resultados | 9 |
| 4.1. Fase de Inicio | 9 |

| | | |
|-----------|---|-----------|
| 4.1.1. | Captura de requisitos | 9 |
| 4.1.2. | Planificación del proyecto | 11 |
| 4.2. | Fase de Elaboración | 13 |
| 4.2.1. | Iteración 1. Estudio del estado del arte | 13 |
| 4.3. | Fase de Construcción | 24 |
| 4.3.1. | Iteración 2. Implementación y evaluación del algoritmo neutral en energía | 25 |
| 4.3.2. | Iteración 3. Implementación de la aplicación de agricultura inteligente | 35 |
| 4.3.3. | Iteración 4. Despliegue del entorno en FIWARE Lab | 42 |
| 4.3.4. | Iteración 5. Integración del sistema final y evaluación | 52 |
| 4.4. | Fase de Transición | 60 |
| 4.4.1. | Iteración 6: Generación del dataset y publicación del mismo en GitHub | 60 |
| 4.4.2. | Iteración 7: Documentación del proyecto | 61 |
| 5. | Conclusiones | 63 |
| 5.1. | Revisión de los objetivos | 63 |
| 5.2. | Competencias adquiridas | 64 |
| 5.3. | Costes del proyecto | 64 |
| 5.4. | Trabajo futuro | 64 |
| | Bibliografía | 67 |
| | A. Métodos adicionales | 73 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| 4.1. Modelo de casos de uso | 11 |
| 4.2. Ventajas de cada una de las tecnologías LP-WAN con respeto a los factores IoT. Imagen tomada de [35] | 15 |
| 4.3. Dispositivo PLATINO | 16 |
| 4.4. Gráficas de descarga y carga de batería. Imágenes tomadas de [15] | 18 |
| 4.5. Producción energética por horas para diferentes valores medios de irradiación mensual. Imagen tomada de [15] | 18 |
| 4.6. Gráfica de nivel de batería (eje y) por instante de tiempo (eje x) | 19 |
| 4.7. Ejemplo de gráfica de carga/descarga de batería y planes de ejecución por slot del mes de agosto | 33 |
| 4.8. Gráficas de carga/descarga y de asignación de planes de ejecución del mes de abril | 34 |
| 4.9. Gráficas de carga/descarga y de asignación de planes de ejecución del mes de marzo | 35 |
| 4.10. Crear un proyecto en PlatformIO | 36 |
| 4.11. Formulario para crear la cuenta de FIWARE Lab | 44 |
| 4.12. Pedir y crear cuenta Community de FIWARE Lab | 45 |
| 4.13. Vista general de FIWARE Lab | 45 |
| 4.14. Creación del grupo de seguridad | 46 |
| 4.15. Agregación de reglas en el grupo de seguridad | 46 |
| 4.16. Crear un par de claves | 46 |
| 4.17. Detalles para lanzar una instancia | 47 |
| 4.18. Acceso y seguridad para lanzar la instancia | 47 |
| 4.19. Asociar y seleccionar IP flotante para nuestra instancia | 47 |
| 4.20. Arquitectura de los contenedores Docker | 49 |
| 4.21. Modelo de datos de NGSI-v2. Figura obtenida de [5] | 49 |
| 4.22. Kit de desarrollo LoPy4 | 52 |
| 4.23. Esquema de comunicación entre las LoPy4 y FIWARE Lab | 53 |
| 4.24. Upload del código del productor en el nodo a través de Visual Studio Code | 57 |
| 4.25. Upload del código de la pasarela a través de Visual Studio Code | 58 |

ÍNDICE DE TABLAS

| | |
|---|----|
| 3.1. Planificación de iteraciones | 6 |
| 4.1. Trazabilidad de requisitos | 10 |
| 4.2. Iteración preliminar | 11 |
| 4.3. Iteración 1 | 11 |
| 4.4. Iteración 2 | 11 |
| 4.5. Iteración 3 | 12 |
| 4.6. Iteración 4 | 12 |
| 4.7. Iteración 5 | 12 |
| 4.8. Iteración 6 | 12 |
| 4.9. Iteración 7 | 12 |
| 4.10. Parámetros utilizados | 26 |
| 4.11. Parámetros de los planes de ejecución | 26 |
| 4.12. Comparación de calidad de servicio entre la asignación inicial y final del mes de abril | 33 |
| 4.13. Comparación de calidad de servicio entre la asignación inicial y final del mes de marzo | 34 |
| 5.1. Coste total del proyecto | 64 |

ÍNDICE DE LISTADOS

| | | |
|-------|---|----|
| 4.1. | Algoritmo de asignación inicial | 27 |
| 4.2. | Código del método <i>Upgrade</i> | 28 |
| 4.3. | Código del método <i>Downgrade</i> | 28 |
| 4.4. | Algoritmo para la reoptimización | 29 |
| 4.5. | Fragmento del archivo CSV correspondiente al mes de enero para mostrar el formato de los mismos | 30 |
| 4.6. | Fragmento del archivo CSV de la reoptimización del mes de enero para mostrar el formato de los mismos | 31 |
| 4.7. | Script para la creación de figuras sobre los archivos CSV | 31 |
| 4.8. | Fragmento del CSV del mes de abril para mostrar como se ha realizado el <i>Upgrade</i> | 33 |
| 4.9. | Fragmento del CSV del mes de marzo para mostrar como se ha realizado el <i>Downgrade</i> | 34 |
| 4.10. | Métodos para obtener los valores de los sensores | 37 |
| 4.11. | Código usado para guardar información en la tarjeta MicroSD | 37 |
| 4.12. | Formato del mensaje para recibir el identificador del dron | 37 |
| 4.13. | Formato del mensaje para enviar el identificador del nodo al dron | 38 |
| 4.14. | Formato del mensaje para enviar los datos leídos de los sensores al dron | 38 |
| 4.15. | Formato del mensaje si falta algún mensaje | 38 |
| 4.16. | Código para enviar mensajes al dron a través de LoRa | 38 |
| 4.17. | Código del método <i>receiveSYNC</i> para recibir el identificador del dron | 39 |
| 4.18. | Código del método <i>sendMyIDToDrone</i> para enviar el identificador del nodo al dron | 39 |
| 4.19. | Código del método <i>sendDataToDrone</i> para enviar los datos al dron | 40 |
| 4.20. | Código del método <i>loop</i> de la aplicación de agricultura inteligente | 41 |
| 4.21. | Comando para conectarse a la instancia de FIWARE Lab | 45 |
| 4.22. | Código del script <i>import-data</i> | 50 |
| 4.23. | Código del script <i>provision-devices</i> para crear un grupo de servicio | 50 |
| 4.24. | Código del script <i>provision-devices</i> para aprovisionar un sensor | 50 |
| 4.25. | Código para el script <i>subscription</i> | 51 |
| 4.26. | Función para generar un numero aleatorio entre dos valores | 53 |
| 4.27. | Inicialización del módulo LoRa y creación del socket LoRa | 54 |
| 4.28. | Método para enviar el número aleatorio | 54 |
| 4.29. | Función para conectarse a una red Wi-Fi | 55 |
| 4.30. | Método para recibir datos del productor | 55 |
| 4.31. | Método para enviar datos a FIWARE Lab | 56 |
| 4.32. | Log del nodo | 56 |
| 4.33. | Log de la pasarela | 58 |
| 4.34. | Comando para obtener los tres últimos valores registrados | 59 |
| 4.35. | Últimos valores registrados en CrateDB | 59 |
| 4.36. | Script para generar el dataset | 60 |
| A.1. | Método <i>compute_cost_assignment</i> | 73 |
| A.2. | Método <i>recompute_battery_level</i> | 74 |

ÍNDICE DE ALGORITMOS

| | | |
|------|---|----|
| 4.1. | Algoritmo de asignación inicial | 21 |
| 4.2. | Función <i>Upgrade</i> | 22 |
| 4.3. | Función <i>Downgrade</i> | 22 |
| 4.4. | Algoritmo de reoptimización | 23 |

INTRODUCCIÓN

La agricultura es uno de los grandes pilares de la alimentación del ser humano. Lleva con nosotros desde hace miles de años, y no podríamos estar vivos hoy en día de no ser por ella. En la actualidad, la agricultura se enfrenta a uno de los mayores retos de su historia: alimentar alrededor de 9.500 millones de personas dentro de 30 años, lo cual equivale a un incremento de producción de un 60 % con respecto al que tenemos hoy en día. Esto, evidentemente, tendrá su repercusión en el medio ambiente, ya que se necesitarán mayores zonas de cultivo, se producirán factores que agravarán el cambio climático, como la erosión del suelo o la destrucción del hábitat de animales[51].

Tradicionalmente, la tecnología ha ayudado en muchos aspectos de la producción agrícola, permitiendo la automatización de procesos que debían realizarse manualmente. En la actualidad, las tecnologías de la información y comunicación (ICT) de las que disponemos hoy, pueden asistir el proceso agrícola a lo largo de todo su ciclo de vida, proporcionando información sobre el estado de los cultivos precisa, actualizada y de una granularidad sin precedentes. Algunos ejemplos de ICT aplicada a agricultura son aviones no tripulados (drones) que se utilizan para obtener imágenes de los cultivos que puedan ser usadas para detectar sequías o plagas, sistemas de geo-posicionamiento o basados en LIDAR para medir la altura de los cultivos, sensores de última generación para medir todo tipo de variables o el empleo de técnicas de Machine Learning y Big Data para encontrar todo tipo de patrones con los datos almacenados. La monitorización de los cultivos puede realizarse mediante redes de sensores inalámbricas [34], formadas por un conjunto de dispositivos de pequeño tamaño, con capacidad de medición de distintas variables de interés (medioambientales, de la planta, o del suelo), procesamiento y comunicación a un computador externo, la *estación base* donde la información puede ser almacenada, procesada, y analizada posteriormente por expertos para la toma de decisiones. Más recientemente, el concepto de Internet of Things (IoT), define la capacidad de "empoderar a las computadoras de hoy con los medios necesarios para obtener información, de manera que puedan ver, escuchar y oler el mundo por sí mismos". Así, los objetos cotidianos que se encuentran a nuestro alrededor (no necesariamente dispositivos *ad-hoc* como en el caso de las redes de sensores), por ejemplo lavadoras, frigoríficos, papeleras, autobuses, etc. cuentan con los medios necesarios para *medir* el entorno y comunicar esos datos a múltiples destinatarios. Una aplicación IoT es la Agricultura Inteligente o *Smart Agriculture* que pretende usar todo el potencial de las ICT para mejorar la producción agrícola a través de un proceso sostenible. La Agricultura Inteligente se puede enfocar al mundo de las tecnologías de la información, implantando una serie de sensores en los cultivos. Dichos sensores ayudan a la recopilación de datos sobre el suelo y los cultivos, como por ejemplo conocer la falta o exceso de agua, humedad, velocidad del viento, irradiación solar, etc.

1.1. MOTIVACIÓN

Desde el punto de vista del diseño de hardware, los dispositivos que se emplean en las aplicaciones de agricultura inteligente deben abordar, al menos, dos retos importantes: el primero de ellos es la necesidad de operar de manera ininterrumpida en el terreno a monitorizar en ausencia, generalmente,

de conexión a la red eléctrica. Es habitual que estos dispositivos vengan equipados con unas baterías que permitan alimentar su circuitería y cuando estas baterías se agotan, deberán reemplazarse para poder continuar la operación. El segundo reto, es la necesidad de comunicación inalámbrica de los dispositivos con algún otro componente de red (pasarela, punto de acceso Wi-Fi, etc.) que se utilice como intermediario antes de alcanzar la red TCP/IP. Sin embargo, en las áreas rurales que se pretenden monitorizar, generalmente, no existe ningún tipo de infraestructura de comunicación que proporcione esta función. Para dar solución a estos dos problemas, el grupo ARCO de la UCLM ha prototipado un dispositivo específicamente diseñado para soportar un amplio conjunto de aplicaciones de agricultura inteligente. Esta plataforma es llamada plataforma PLATINO, de la cual se hablará más adelante.

El principal objetivo del nodo PLATINO es proporcionar una herramienta flexible y modular para soportar la ejecución ininterrumpida de aplicaciones por medio de un recolector de energía, el cual permite aumentar el nivel de batería dependiendo del consumo de la aplicación que se está ejecutando y de las condiciones climatológicas. La placa se compone de dos microcontroladores independientes que proporcionan funciones especializadas usando diferentes tecnologías.

Para poder alcanzar los objetivos de operación prolongada y comunicación de datos, además del soporte hardware, se necesita dotar al dispositivo de los algoritmos necesarios para realizar la gestión eficiente de la energía y la comunicación. Además, estos algoritmos deben implementarse sobre esta arquitectura de hardware que cuenta con ciertas limitaciones, por ejemplo, el espacio de almacenamiento. Precisamente, con esta motivación surge el presente Trabajo Fin de Grado, en adelante, TFG. El principal objetivo es utilizar el nodo PLATINO como arquitectura de hardware para implementar un algoritmo que realice la gestión eficiente de la energía y hago uso de una radio LoRa para la comunicación de datos. Para poder evaluar las prestaciones de estos algoritmos, se desplegará y evaluará el rendimiento de una red Low-Power Area Network (LP-WAN), de dispositivos PLATINO, neutrales en energía, los cuales se comunican a través de la red inalámbrica LoRa. Dichos dispositivos generarán una serie de datos (mediciones de los sensores) que serán enviados a un *gateway* LoRa utilizando un protocolo de comunicaciones que se desarrollará en este TFG. Por su parte, el gateway reenviará los datos a una plataforma de IoT denominada FIWARE Lab. Con los datos almacenados, se procederá después a generar un *dataset* que se publicará en GitHub y en la propia plataforma FIWARE Lab para su descarga.

1.2. ESTRUCTURA DEL DOCUMENTO

Tras introducir al lector en este proyecto, se procederá a explicar la estructura del documento a partir de este capítulo:

- **Capítulo 2. Objetivos.** Se presenta el objetivo general que se pretende conseguir con el desarrollo de este Trabajo Fin de Grado y se detallarán objetivos más específicos para llevar a cabo el objetivo principal.
- **Capítulo 3. Metodología de trabajo.** En este capítulo se explica la metodología que se ha seguido para la elaboración de este Trabajo Fin de Grado y el marco tecnológico, tanto hardware como software, en el que se desarrolla.
- **Capítulo 4. Resultados.** En este capítulo se describe la aplicación de la metodología seleccionada en el capítulo anterior para alcanzar los objetivos planteados.
- **Capítulo 5. Conclusiones.** En este capítulo final, se describen las principales conclusiones obtenidas después de la realización de este trabajo, el grado de consecución de los objetivos planteados, y se describen nuevas aplicaciones que se podrían desarrollar como futuros casos de estudio y Trabajos Fin de Grado.

OBJETIVO

En este capítulo, se describe el objetivo principal que se pretende conseguir en este TFG. Para abordar este objetivo principal, se procederá a dividir este objetivo en varios objetivos parciales, más específicos, que abordan aspectos tanto técnicos como académicos.

2.1. OBJETIVO GENERAL

El objetivo general de este proyecto es llevar a cabo un despliegue real de una red Low-Power Wide Area Network (LP-WAN) de dispositivos neutrales en energía conectados a través de la tecnología inalámbrica LoRa en un escenario de Agricultura Inteligente. Este despliegue permitirá generar un conjunto de datos (*dataset*) con los datos muestreados a través de los distintos sensores que se publicará para que posteriormente pueda ser utilizado por terceros para múltiples propósitos de investigación o académicos.

2.2. OBJETIVOS PARCIALES

A continuación, se exponen los objetivos parciales del proyecto.

2.2.1. Prototipado de un conjunto de dispositivos finales

Este proyecto no tendría sentido si no se tuviera un conjunto de dispositivos finales con los que trabajar. Por eso mismo, se debe seleccionar una serie de dispositivos que recolecten energía a través de un panel solar y que se puedan comunicar con otros dispositivos o con una pasarela a través de un módulo de comunicaciones LoRa. También, se necesitarán una serie de sensores que recojan la información del entorno en el que se encuentran, como velocidad del aire, temperatura, nivel de humedad, etc. Para ello, se tomará como referencia el nodo PLATINO, del cual hemos hablado en el Capítulo 1, y se añadirán las mejoras necesarias a su diseño de hardware. En base a este prototipo, se procederá a la fabricación de un conjunto de nodos LoRa que sean utilizados para el despliegue final.

2.2.2. Implementación de un algoritmo neutral en energía

Para que un dispositivo puedan mantenerse funcionando ininterrumpidamente se necesita dotar a este dispositivo de un algoritmo *neutral en energía*, que permita garantizar que el nivel de batería del dispositivo al final de un período de referencia, es siempre mayor o igual que el nivel de batería al principio del mismo período.

Algunos algoritmos neutrales en energía ya han sido previamente propuestos y para este proyecto vamos a utilizar el algoritmo publicado en [15]. Este algoritmo ha sido simulado y no implementado sobre una plataforma física, por lo que tendremos que implementarlo en los dispositivos PLATINO. Así pues, este objetivo persigue la implementación de dicho algoritmo sobre la plataforma PLATINO,

de manera que se convierta al dispositivo en neutral en energía y tenga autonomía suficiente para trabajar a cualquier hora del día, aunque no siempre se pueda obtener energía desde el panel solar.

2.2.3. Implementación de un protocolo de comunicación basado en LoRa

Los dispositivos finales deben poder comunicarse a través de LoRa con un *gateway* o pasarela que recoja dicha información para luego enviarla a algún servidor. Por lo tanto, es necesario implementar un protocolo de comunicaciones basado en la tecnología de radio-frecuencia *LoRa* para que cada uno de los dispositivos envíe la información que recolectan los sensores a la pasarela.

2.2.4. Implementación de un protocolo de comunicaciones basado en REST entre el gateway y la plataforma FIWARE

Cuando el gateway recibe los datos procedentes de los distintos dispositivos, actuará como pasarela y deberá reenviar esos datos a algún otro punto de la red donde los datos puedan ser almacenados de manera permanente y procesados para obtener información relevante que permita el proceso de toma de decisiones. En este TFG se ha optado por utilizar FIWARE como plataforma de almacenamiento de datos en la nube. Para tal finalidad, se debe de implementar un protocolo de comunicaciones, basado en los principios REST, que permita la comunicación del gateway con la plataforma de FIWARE, la cual, almacenará los datos en una base de datos para que sean tratados más adelante. También, para saber si el protocolo de comunicaciones que ha sido implementado es fiable y correcto, se realizará una evaluación del protocolo para comprobar su rendimiento, en términos de sobrecarga, entrega fiable de paquetes, etc.

2.2.5. Despliegue de la red en entorno real

Una vez desarrollados y evaluados todos los elementos de la arquitectura, se procederá a desplegar una red de dispositivos, basados en el nodo PLATINO, y un gateway de comunicaciones sobre un entorno real, específicamente, en los alrededores del Instituto de Tecnologías y Sistemas de la Información (ITSI).

2.2.6. Evaluación de la red mediante un experimento real

Este objetivo persigue evaluar la red desplegada mediante una aplicación real en el contexto de agricultura inteligente para comprobar su correcto funcionamiento y sus prestaciones.

2.2.7. Publicación del *dataset* generado en un repositorio público en *GitHub*

Por último, de los datos que hemos recibido de los distintos dispositivos una vez que ya se ha desplegado la red y que han sido recogidos por el gateway, que enviará dichos datos a FIWARE, se generará un *dataset* de los datos guardados en la base de datos de la plataforma FIWARE, el cuál será compartido en *GitHub* en un repositorio público.

METODOLOGÍA

En este capítulo se identificará y se describirá la metodología de trabajo que se ha seguido para el desarrollo este TFG y conseguir los objetivos descritos en el Capítulo 2. Se presenta también el marco tecnológico, las herramientas hardware y software que hemos utilizado para su desarrollo.

3.1. ELECCIÓN Y BREVE DESCRIPCIÓN DE LA METODOLOGÍA

Para la elaboración de este TFG se ha utilizado la metodología del Proceso Unificado de Desarrollo, la cuál llamaremos PUD a partir de ahora. La definición de esta metodología, de acuerdo a [36], es la siguiente:

El PUD es un marco genérico que puede adaptarse para una amplia gama de sistemas de software, para varias áreas de aplicación, varios tipos de aplicaciones, varios tipos de organizaciones y proyectos de varios tamaños.

El PUD se caracteriza por ser:

- **Está orientado a casos de uso:** Un caso de uso es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. Aparte, modelan los requerimientos funcionales del sistema. Todos los casos de uso juntos constituyen el modelo de casos de uso. También guían el proceso de desarrollo, en el que los desarrolladores pueden crear una serie de modelos de diseño e implementación que llevan a cabo los casos de uso. Los casos de uso, no solo inician el proceso de desarrollo, sino que le proporcionan un hilo conductor.
- **Es iterativo e incremental:** Es conveniente dividir el esfuerzo de desarrollo de un proyecto en partes más pequeñas. Cada una de esas partes se denomina iteración, que resulta en un incremento. Las iteraciones deben de ser controladas, lo que significa que deben seleccionarse y ejecutarse de una forma planificada. En cada iteración, los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía para implementar dichos casos de uso. Si la iteración cumple con sus objetivos, se continúa con la próxima. Sino deben revisarse las decisiones previas y probar un nuevo enfoque.
- **Está centrado en la arquitectura:** La arquitectura es un conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, las interfaces entre ellos, su comportamiento, sus colaboraciones y su composición. Los casos de uso y la arquitectura están estrechamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez, la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos.

3.2. GESTIÓN DEL PROYECTO

El PUD puede ser dividido en cuatro fases, cada una de ellas representa un ciclo de desarrollo en la vida de un producto de software:

1. **Inicio:** En esta fase se hace una descripción del producto final a partir de una especificación de requisitos. De esta fase, se obtiene un modelo de casos de uso simplificado.
2. **Elaboración:** Durante esta fase se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura. A partir de esta fase, se obtiene la línea base de la arquitectura.
3. **Construcción:** Durante esta fase se crea el producto. La línea base de la arquitectura crece hasta convertirse en el sistema completo. Al final de esta fase, todos los casos de uso han sido implementados, pero puede haber errores y que el sistema no esté completo.
4. **Transición:** En esta fase el producto se convierte en su versión beta y se evalúa el mismo en busca de errores. A partir de esta fase se obtiene el producto final y la documentación del desarrollo.

3.2.1. Aplicación

En este apartado se muestra cómo se va a aplicar el PUD para la gestión de este proyecto. Más adelante, en el Capítulo 4 se mostrarán los resultados de cada una de las iteraciones del proyecto. En la Tabla 3.1 se puede observar la planificación que se ha seguido en el proyecto y la asignación de las iteraciones a las fases del PUD, así como los objetivos más relevantes.

Tabla 3.1: Planificación de iteraciones

| Fase | Iteración | Objetivos |
|--------------|------------|---|
| Inicio | Preliminar | Planificación del proyecto y captura de requisitos. |
| Elaboración | 1 | Estudio del estado del arte. |
| Construcción | 2 | Implementación y evaluación del algoritmo neutral en energía. |
| | 3 | Implementación de la aplicación de agricultura inteligente. |
| | 4 | Despliegue del entorno en FIWARE Lab. |
| | 5 | Integración del sistema final y evaluación. |
| Transición | 6 | Generación del dataset y publicación de resultados. |
| | 7 | Documentación del proyecto. |

3.3. MARCO TECNOLÓGICO

3.3.1. Herramientas hardware

- **Ordenador portátil**[39]: HP Pavilion 15 BC-409NS, empleado para el desarrollo y la redacción de la documentación del proyecto.
- **LoPy4**[42]: Microcontroladores utilizados para la implementación del nodo y la pasarela.
- **Nodo Platino**[16]: Plataforma hardware diseñada por el grupo de investigación ARCO de la UCLM, la cuál ha sido usada para el desarrollo de este proyecto.
- **Ordenador All in One HP**[40]: Ordenador utilizado para la realización del proyecto en el grupo de investigación ARCO.

3.3.2. Herramientas para la gestión de proyectos

- **GitHub**[11]: GitHub es el servicio de alojamiento remoto de repositorios de Microsoft, que maneja el software de control de versiones Git.
- **Git**[53]: Herramienta de control de versiones de software libre creada por el desarrollador del kernel de *Linux* Linus Torvalds.
- **Trello**[3]: Trello es una herramienta de gestión y administración de proyectos que simplifica los recursos para realizar su seguimiento.
- **Teams**[12]: Teams es una herramienta para realizar conferencias y videollamadas desarrollada por Microsoft. Se ha utilizado Teams para la comunicación con la tutora que dirige este proyecto.

3.3.3. Herramientas, lenguajes y tecnologías para el desarrollo de software

- **Visual Studio Code**[13]: Visual Studio Code es un editor de texto moderno desarrollado por Microsoft.
- **PlatformIO**[41]: PlatformIO es una herramienta profesional multiplataforma de arquitectura cruzada y marco múltiple para desarrolladores de software que escriben aplicaciones para productos embebidos. Cuenta con un gran soporte para la mayoría de microcontroladores más famosos (Arduino, Adafruit, ESP32, etc.).
- **Pymakr**[44]: Pymakr, como extensión para Visual Studio Code, permite programar los microcontroladores de Pycom, en nuestro caso la LoPy4, añadiendo una consola con la que poder interactuar.
- **C++**[50]: Lenguaje de programación de alto nivel creado por Bjarne Stroustrup. Se ha utilizado para implementar el algoritmo *energy-harvesting*.
- **Python**[19]: Python es un lenguaje de programación de alto nivel. Se ha usado su versión 3.6.9 para la creación de gráficas de consumo de la batería de los dispositivos y selección de planes de ejecución, y para la creación del dataset.
- **MicroPython**[30]: MicroPython es una versión de Python 3 diseñada para su uso en microcontroladores.
- **Docker**[24]: Docker es una herramienta que permite la rápida creación, despliegue y ejecución de aplicaciones mediante el uso de contenedores.
- **Docker Compose**[25]: Docker Compose es una herramienta para definir y ejecutar aplicaciones con múltiples contenedores Docker.
- **FIWARE Orion Context Broker**[7]: El Orion Context Broker es una implementación en C++ de la API REST NGSIv2, en la que se puede consultar y actualizar información de contexto.

3.3.4. Herramientas para la elaboración de la documentación

- **L^AT_EX**[52]: L^AT_EX es un sistema de composición de documentos, con el cual ha sido realizado el presente documento.
- **Overleaf**[38]: Overleaf es un editor online de L^AT_EX utilizado para realizar la documentación del proyecto.

- **Draw.io**^[33]: Draw.io es una herramienta web para la creación de distintos tipos de diagramas.

RESULTADOS

Este capítulo describe las fases del ciclo de vida de la metodología PUD, descrito en el capítulo 3, y el desarrollo de las iteraciones que conforman cada una de esas fases, de acuerdo a la Tabla 3.1. Además, se muestra como la metodología satisface los objetivos definidos en el Capítulo 2.

4.1. FASE DE INICIO

En esta fase del PUD se lleva a cabo una iteración *Captura de requisitos y planificación del proyecto* tal y como se puede ver en la Tabla 3.1. Se trata de una primera iteración, que hemos denominado *Preliminar* debido a que debe realizarse con anticipación a cualquier desarrollo de este proyecto..

4.1.1. Captura de requisitos

Tras observar cuales son los los objetivos a cumplir mencionados en el Capítulo 2, se analizaron cuales son las necesidades y funcionalidades que debe de cumplir nuestro proyecto, así como sus restricciones. Hecho esto, se obtuvieron los siguientes requisitos funcionales (denominados RF, seguido de un número de orden):

- **RF.01 Mantener la ejecución ininterrumpida del dispositivo PLATINO:** Para lograr este requisito, será necesario implementar un algoritmo neutral en energía, que, conjuntamente con la habilidad de obtener la energía del sol mediante un panel fotovoltaico, permita la ejecución prolongada del dispositivo en el tiempo.
- **RF.02 Analizar y evaluar el algoritmo neutral en energía:** El usuario podrá analizar y evaluar el algoritmo neutral en energía, para verificar que la planificación de tareas que genera el algoritmo es correcta, está actualizada, y se adapta a las condiciones de producción solar.
- **RF.03 Recolección de datos desde el dispositivo PLATINO:** El dispositivo PLATINO, a través de los sensores disponibles, obtendrá datos periódicamente sobre el terreno a monitorizar y sobre el estado energético del dispositivo.
- **RF.04 Despliegue de una red LP-WAN de dispositivos neutrales en energía sobre el terreno:** Se deberá desplegar una red LP-WAN, utilizando alguna tecnología de comunicaciones de largo alcance. La red LP-WAN estará formada por dispositivos finales PLATINO, que deberán transmitir los datos recolectados a un gateway de comunicaciones o pasarela que permita comunicar alguna red exterior basada en TCP/IP.
- **RF.05 Almacenamiento de los datos:** Los datos reenviados desde el gateway deberán poder almacenarse en soporte secundario, con el objetivo de ser posteriormente procesados y analizados, y puedan ser utilizados en algún proceso de toma de decisiones.

- **RF.06 Generación de un *dataset*:** El conjunto de datos almacenados, procedentes de los distintos dispositivos de la red, se usarán para generar un dataset, que será publicado para que terceras personas puedan utilizarlo para distintos fines.

A continuación, se identifican los siguientes requisitos no funcionales (denominados RNF, seguido de un número de orden):

- **RNF.01 Alojamiento en la nube:** El almacenamiento al que se refiere el RF.05, deberá ser implementado en una plataforma *cloud*, específicamente, la plataforma de Internet of Things denominada FIWARE Lab.
- **RNF.02 Comunicaciones de largo alcance:** Las comunicaciones de los dispositivos con la pasarela debe de ser de largo alcance, sin que esto aumente en exceso el coste del proyecto.

Los requisitos del sistema que se deben de tener en cuenta para este proyecto son los siguientes:

- **RS.01:** El sistema operativo que se use en el PC con el que se esté desarrollando el proyecto debe de ser una distribución de GNU/Linux.
- **RS.02:** El sistema operativo que se use en la plataforma de FIWARE Lab tiene que ser una distribución de GNU/Linux.
- **RS.03:** Se debe de tener instalada una versión de Python que sea 3.6.9 o superior.
- **RS.04:** Se debe de tener instalado el editor Visual Studio Code con la extensión de PlatformIO instalada.

Con estos requisitos ya definidos, podemos obtener una versión inicial del modelo de casos de uso del sistema tras aplicar la trazabilidad de requisitos a casos de usos con el uso de PUD, tal y como se puede ver en la Tabla 4.1. Partiendo de los requisitos definidos anteriormente y de la suposición de que varios requisitos se pueden mapear en un caso de uso, se obtienen los siguientes casos de uso:

| Caso de Uso | Requisito funcional |
|--|--|
| CdU.01 Obtener un plan de ejecución de aplicaciones neutrales en energía | RF.01 Mantener la ejecución ininterrumpida del dispositivo PLATINO RF.02 Analizar y evaluar el algoritmo neutral en energía |
| CdU.02 Recolección de datos del terreno y estado energético | RF.03 Recolección de datos del terreno |
| CdU.03 Despliegue de una red LP-WAN | RF.04 Despliegue de una red LP-WAN de dispositivos neutrales en energía |
| CdU.04 Almacenamiento de los datos | RF.05 Almacenamiento de los datos |
| CdU.05 Generación de un dataset | RF.06 Generación de un dataset |

Tabla 4.1: Trazabilidad de requisitos

En la Figura 4.1 se puede apreciar el modelo de casos de uso del proyecto. Como se puede apreciar, el usuario puede generar un dataset, y los sensores pueden obtener un plan de ejecución y recolectar datos del terreno. Para la generación del dataset, se necesita que primero se hayan almacenado una serie de datos, y para recolectar los datos del terreno, se necesita almacenar dichos datos. Por último, para poder hacer la recolección de datos del terreno, se necesita que haya un despliegue de una red LP-WAN.

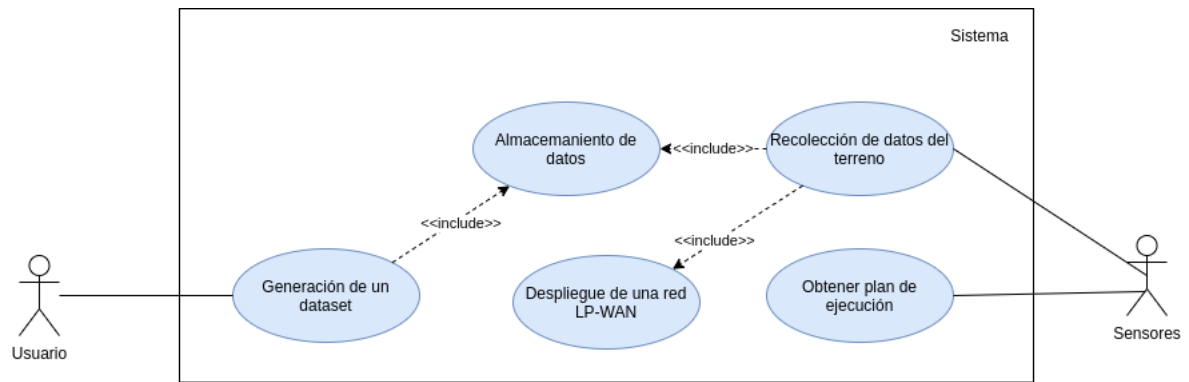


Figura 4.1: Modelo de casos de uso

4.1.2. Planificación del proyecto

Se ha llevado a cabo una planificación del proyecto, identificando las distintas iteraciones a completar del mismo. En las siguientes tablas se muestra la descripción de cada iteración, junto con los objetivos, los artefactos, el inicio y el final, así como su duración.

Tabla 4.2: Iteración preliminar

| | |
|-------------------|---|
| Iteración | Preliminar |
| Fase | Inicio |
| Objetivos | Captura de Requisitos Versión inicial del modelo de casos de uso Planificación del proyecto |
| Artefactos | Requisitos funcionales y no funcionales del proyecto Modelo de casos de uso |
| Inicio | 20/01/2020 |
| Final | 24/01/2020 |
| Duración | 20h |

Tabla 4.3: Iteración 1

| | |
|-------------------|----------------------------------|
| Iteración | 1 |
| Fase | Elaboración |
| Objetivos | Estudio del estado del arte |
| Artefactos | Conocimiento del estado del arte |
| Inicio | 27/01/2020 |
| Final | 12/02/2020 |
| Duración | 40h |

Tabla 4.4: Iteración 2

| | |
|-------------------|--|
| Iteración | 2 |
| Fase | Construcción |
| Objetivos | Implementación y evaluación del algoritmo neutral en energía |
| Artefactos | Planes de ejecución neutrales en energía (CdU.01) |
| Inicio | 13/02/2020 |
| Final | 09/04/2020 |
| Duración | 140h |

Tabla 4.5: Iteración 3

| | |
|-------------------|--|
| Iteración | 3 |
| Fase | Construcción |
| Objetivos | Implementación de la aplicación de agricultura inteligente |
| Artefactos | Aplicaciones para los dispositivos PLATINO y pasarela LoRa |
| Inicio | 09/04/2020 |
| Final | 05/05/2020 |
| Duración | 60h |

Tabla 4.6: Iteración 4

| | |
|-------------------|--|
| Iteración | 4 |
| Fase | Construcción |
| Objetivos | Crear una cuenta de usuario en FIWARE Lab Despliegue de la instancia e instalación de componentes en la arquitectura FIWARE |
| Artefactos | Entorno FIWARE configurado y desplegado |
| Inicio | 06/05/2020 |
| Final | 22/05/2020 |
| Duración | 50h |

Tabla 4.7: Iteración 5

| | |
|-------------------|---|
| Iteración | 5 |
| Fase | Construcción |
| Objetivos | Integración de la red LoRa con la plataforma FIWARE |
| Artefactos | Protocolo de comunicación entre el gateway y FIWARE Primera versión del sistema (CdU.02) |
| Inicio | 25/05/2020 |
| Final | 03/06/2020 |
| Duración | 20h |

Tabla 4.8: Iteración 6

| | |
|-------------------|---|
| Iteración | 6 |
| Fase | Transición |
| Objetivos | Generación del dataset a partir de los datos almacenados |
| Artefactos | Publicación del dataset Dataset en <i>GitHub</i> y FIWARE (CdU.03) |
| Inicio | 04/06/2020 |
| Final | 08/06/2020 |
| Duración | 10h |

Tabla 4.9: Iteración 7

| | |
|-------------------|---|
| Iteración | 7 |
| Fase | Transición |
| Objetivos | Documentación del proyecto |
| Artefactos | Memoria del TFG y repositorio de código |
| Inicio | 09/06/2020 |
| Final | 14/07/2020 |
| Duración | 50h |

4.2. FASE DE ELABORACIÓN

De acuerdo a la Tabla 3.1 la fase de elaboración está formada por una única iteración: *Estudio del estado del arte*, que se describe a continuación.

4.2.1. Iteración 1. Estudio del estado del arte

Para poder abordar este proyecto, es necesario el estudio del estado del arte de diversas tecnologías alineadas con los objetivos de este proyecto. Estas tecnologías se pueden agrupar en cuatro categorías, que se describen en las siguientes secciones:

1. Low Power-Wireless Area Networks (LP-WAN).
2. Plataforma de hardware PLATINO.
3. Sistemas energy-harvesting basados en paneles solares y algoritmos neutrales en energía.
4. Plataformas de almacenamiento en la nube.

Low Power-Wireless Area Networks (LP-WAN)

Uno de los pilares clave del IoT es la interconexión e intercambio de datos entre cualquier tipo de dispositivos, en cualquier lugar y en cualquier momento. Dependiendo de la aplicación de IoT, como por ejemplo agricultura inteligente (escenario en el que se desarrolla este trabajo), uno de los requisitos específicos es la comunicación de largo alcance, con baja tasa de datos, bajo consumo energético y que sea rentable. Para satisfacer estos requisitos existen distintas alternativas. Por un lado, las tecnologías de radio de corto alcance (como ZigBee o Bluetooth) no se adaptan bien a este escenario, dada la limitación de la distancia entre los dispositivos que comunican. Las soluciones basadas en comunicación móvil (e.g. 2G, 3G y 4G) proporcionan gran cobertura, pero su consumo es excesivo. Es por esto que se necesita buscar otro tipo de tecnologías de comunicaciones inalámbricas, como son las basadas en Low Power-Wireless Area Networks (LP-WAN), que han ido aumentando en popularidad en estos escenarios debido fundamentalmente al largo alcance, bajo consumo y coste de las comunicaciones. Las principales tecnologías LP-WAN son Sigfox, LoRa y NB-IoT, las cuales se describen brevemente a continuación:

- **Sigfox:** Sigfox [47] es un operador de red LP-WAN que ofrece una solución IoT para la conexión punto-a-punto de los dispositivos, basada en sus tecnologías patentadas. Sigfox despliega sus estaciones base propietarias que recoge la información de los dispositivos y los conecta a un servidor usando una red basada en IP. Utiliza las bandas ISM sin licencias, como por ejemplo 868 MHz en Europa, 915 MHz en Norte América, o 433 MHz en Asia. Usando estas bandas ultra-estrechas, Sigfox usa el ancho de banda de la frecuencia de forma eficiente y con bajos niveles de ruido y bajo consumo energético, produciendo una tasa de datos de solo 100 bps. Sigfox cuenta con comunicación bidireccional, con un límite de 140 mensajes de subida al día (*uplink*) con una pasarela de tamaño máximo de 12 bytes, y solo 4 mensajes de bajada (*downlink*) con un tamaño de pasarela máximo de 8 bytes. Debido a esta limitación del número de mensajes que se pueden enviar y recibir, no es posible enviar un asentimiento o ACK de subida. Sin este mecanismo, los mensajes se envían múltiples veces (3 por defecto) a través de diferentes canales de frecuencia. Para este propósito, en Europa por ejemplo, la banda entre 868.180 MHz y 868.220 MHz es dividida en 400 canales ortogonales de 100 Hz cada uno. Como la estación base puede recibir mensajes de forma simultánea a través de todos los canales, el dispositivo puede elegir de forma aleatoria el canal por el que transmitir sus mensajes.
- **LoRa:** LoRa [2] (Long Range communications) es una tecnología de capa física que también usa las bandas no licenciadas ISM, al igual que Sigfox. La comunicación bidireccional se proporciona a través de un esquema de modulación denominado CSS (Chirp Spread Spectrum), que esparce una señal de banda estrecha sobre un ancho de banda de canal más amplio. LoRa usa 6 factores

de propagación (*Spreading Factor*). El factor de propagación es el número de símbolos mandados en por bit, ya que 1 bit es codificado como múltiples *chirps*. Estos factores de propagación van numerados desde el 7 al 12 (de SF7 a SF12) para adaptar la velocidad de transmisión de datos y el rango. Factores de propagación más grandes permiten mayor alcance a costa de una menor velocidad de envío de datos, y viceversa. La tasa de datos de LoRa está entre los 300 bps y los 50 kbps, dependiendo del factor de propagación y el ancho de banda del canal. El tamaño máximo de pasarela para cada mensaje es 243 bytes. En 2015 se estandarizó un protocolo de acceso al medio y de red (niveles 2 y 3, respectivamente, de los niveles OSI de protocolos) que usa LoRa como capa física, llamado LoRaWAN[49]. Usando LoRaWAN, todos los mensajes que manden los dispositivos serán recibidos por todas las estaciones bases disponibles. Los mensajes duplicados que se puedan recibir por varias bases serán filtrados en el servidor de red, que incluye el procesamiento necesario para comprobar la seguridad, enviar asentimientos o ACKs a los dispositivos, y enviar el mensaje al correspondiente servidor de aplicaciones. Gracias a estos mensajes duplicados recibidos por varias bases, se puede localizar al dispositivo que lo envió, gracias a técnica basada en la diferencia de tiempo de llegada (TDOA). Además, múltiples recepciones del mismo mensaje en diferentes estaciones base evitan la entrega en la red LoRaWAN. Aparte, LoRaWAN proporciona varias clases de dispositivos finales para abordar los diferentes requisitos de una amplia gama de aplicaciones de IoT:

- *Dispositivos bidireccionales (class A)*: un dispositivo de clase A permite una comunicación bidireccional, en la que un mensaje de subida es seguido de dos cortas ventanas de recepción de bajada. La comunicación de clase A es la que menos consume, y se usa para aplicaciones que solo necesitan una comunicación corta de bajada después de que el dispositivo haya enviado un mensaje de subida.
 - *Dispositivos bidireccionales con programación de recibo de lotes (class B)*: Además de las ventanas de recibo aleatorias de la clase A, los dispositivos de la clase B abren ventanas de recepción adicionales en un tiempo programado. Para abrir ventanas de recepción en el instante programado, los dispositivos reciben un mensaje de sincronización desde la base. Eso permite al servidor de aplicaciones saber cuando un dispositivo está escuchando.
 - *Dispositivos bidireccionales con espacios máximos de recibo (class C)*: Los dispositivos de clase C tienen abiertas ventanas de recepción casi de forma continua, y solo se cierran cuando se transmite a expensas de un consumo excesivo de energía.
- **NB-IoT**: NB-IoT [1] es una tecnología de banda estrecha de IoT creada en 2016 por 3GPP. NB-IoT coexiste con GSM y LTE bajo bandas de frecuencia licenciadas, como por ejemplo, 700, 800 y 900 MHz. Ocupa un ancho de banda de frecuencia de 200 KHz, que corresponde a un bloque de recursos en la transmisión GSM y LTE. Con esta selección de bandas de frecuencia, se pueden tener los siguientes tipos de operaciones:
- *Operación stand-alone*: un posible escenario es la utilización de bandas de frecuencia de GSM usadas actualmente.
 - *Operación guard-band*: se utilizan los bloques de recursos no utilizados dentro de la banda de guardia de un operador LTE.
 - *Operación in-band*: se utilizan los bloques de recursos de un operador LTE.

NB-IoT está basado en el protocolo LTE. De hecho, NB-IoT reduce las funcionalidades de LTE al mínimo y las mejora para su uso en aplicaciones IoT. Aparte, NB-IoT puede verse como una nueva interfaz aérea desde el punto de vista de la pila de protocolos, estando construida en bajo la infraestructura de LTE. NB-IoT permite una conectividad de hasta 100.000 dispositivos por celda con el potencial de ampliar la capacidad para más operadores NB-IoT. NB-IoT utiliza el acceso múltiple por división de frecuencia de una sola portadora (FDMA) para

la subida de mensajes y FDMA ortogonal (OFDMA) para la bajada, empleando la modulación por desplazamiento de fase cuaternaria (QPSK). La velocidad de envío de datos es de 200 para la bajada y de 20 kbps para la subida. El tamaño máximo de la pasarela es de 1600 bytes.

Se deben de considerar varios factores a la hora de elegir una tecnología LP-WAN para una aplicación de IoT, como es la calidad de servicio (del inglés, Quality of Service, QoS) el consumo de batería, latencia, escalabilidad, longitud de la pasarela, cobertura, rango, despliegue y coste. La Figura 4.2 muestra una comparación de características de las tecnologías descritas anteriormente con respecto de los factores ya comentados. Como se puede apreciar en la imagen, NB-IoT tiene mejor escalabilidad, mejor latencia, tamaño de pasarela y calidad de servicio, pero no tiene tan buen rendimiento para el uso de batería, eficiencia en coste, despliegue, cobertura y rango. Por el contrario, Sigfox es la que mejor se comporta en los puntos flacos de NB-IoT: rango, cobertura, eficiente en coste y duración de batería. NB-IoT utiliza las banda licenciadas de LTE, por lo que se tiene un coste para poder usarlas y, además, no es tan eficiente como SigFox o LoRa. Sigfox tiene más rango y cobertura que LoRa, pero el hecho de que se tenga un límite de subida y bajada de mensajes, y que se tenga que pagar para poder usar sus estaciones base, son puntos débiles de esta tecnología. Por todos estos aspectos, junto con el hecho de que los dispositivos PLATINO que utilizaremos como plataforma en este trabajo, usan una radio LoRa, se ha determinado la elección de LoRa como tecnología de largo alcance para la red de dispositivos que desplegaremos en este proyecto.

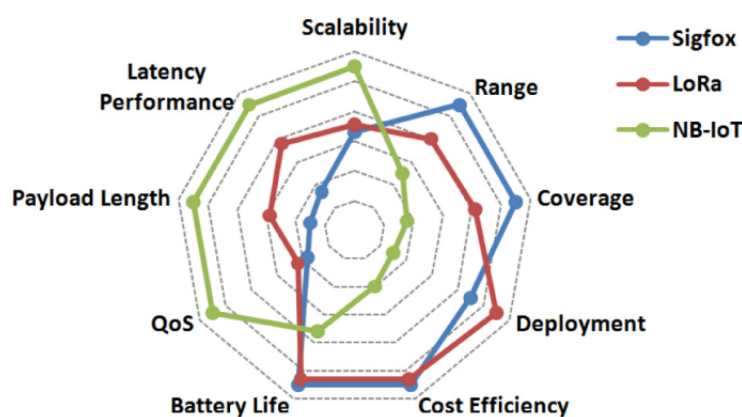


Figura 4.2: Ventajas de cada una de las tecnologías LP-WAN con respecto a los factores IoT. Imagen tomada de [35]

Plataforma hardware PLATINO

Para la realización de este proyecto, se va a utilizar la plataforma hardware PLATINO. Como ya se explicó en el Capítulo 1.1, esta plataforma permite solucionar dos problemas principales a los que se enfrenta la agricultura inteligente: operar de manera ininterrumpida y la necesidad de comunicación inalámbrica. Esta plataforma ha sido diseñada y prototipada por el grupo de investigación ARCO de la UCLM. PLATINO proporciona una herramienta flexible y modular para soportar la ejecución ininterrumpida de aplicaciones por medio de un recolector de energía, el cual permite que el nivel de la batería aumente dependiendo del consumo de la aplicación que se esté ejecutando y de las condiciones meteorológicas. La plataforma se compone de dos micro-sistemas independientes que proporcionan funcionalidades usando diferentes tecnologías: el Smart Farming Device, que incluye un conjunto de sensores para implementar un conjunto de aplicaciones de monitorización al aire libre, y el Data Logger, que recoge de manera independiente los parámetros relacionados con la batería y el subsistema de recolección de energía. Ambos micro-sistemas tienen fuentes de alimentación separadas y están conectados a través de la interfaz serie. La Figura 4.3 muestra como es la plataforma PLATINO.

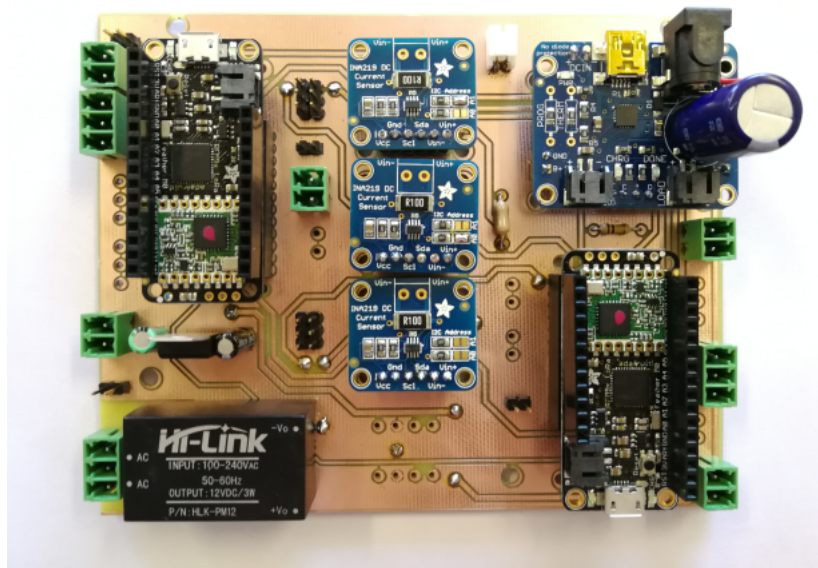


Figura 4.3: Dispositivo PLATINO

Las características de cada uno de los micro-sistemas son las siguientes:

- **Smart Farming Device:** Este dispositivo está basado en un microcontrolador Adafruit Feather M0 con una una radio LoRa RFM95 y con capacidad de *energy harvesting*, es decir, recolectar energía para así aumentar la autonomía del mismo y su tiempo de vida. El recolector de energía es un panel solar fotovoltaico de 2 Watios, un cargador solar para convertir la energía solar en electricidad y una batería recargable de litio con una capacidad de 2000 mAh, un voltaje nominal de 3.7 V, cuya salida va desde los 4.2 V cuando está totalmente cargada y 3.3 V cuando está descargada. El panel solar proporciona una corriente nominal de 340 mA con un voltaje de salida de 6.5 V. El panel solar puede proporcionar la potencia de salida directamente a la salida de carga, donde está conectado el el dispositivo Smart Farming si la energía obtenida es suficiente para mantener la operación del dispositivo y la batería esta cargada por completo. Por el contrario, si la batería no está cargada por completo y la producción energética del panel no es suficiente, la corriente del panel solar se usa para cargar la batería y la corriente se suministra desde la misma. El subsistema de sensado incluye 20 pines de entrada/salida de propósito general, con 10 pines de entrada analógica, 8 para PWM, e interfaces para comunicación con UART, SPI y I2C. También incluye un sensor SHT10 para medir la temperatura entre -40 y 123.8 °C, y la humedad relativa entre el 0 y 100 %.
- **Data Logger Device:** También está basado en un microcontrolador Adafruit Feather M0 con una radio LoRa RFM95, con la capacidad adicional de poder guardar información a través de una tarjeta MicroSD, un real-time clock (RTC), la visualización del estado actual a través de una pantalla OLED monocromo y la monitorización de condiciones energéticas. Para este último propósito, utiliza 3 sensores INA219 para medir las corrientes y voltajes de la batería, del Smart Farming Device y del panel solar. Este módulo soporta la interfaz I2C, operando solo como un dispositivo esclavo, y cuenta con hasta 16 direcciones programables. Es usada la configuración de registro predeterminada, permitiendo medir un rango de ± 3.2 A, con una resolución de ± 0.8 mA. Además, calcula la radiación solar mediante un mini relé que mide la corriente del cortocircuito del panel. Por último, usa un anemómetro para medir la velocidad del viento con una salida analógica en el rango 0.4-2V y una velocidad del viento de entre 0.5 a 50 m/s.

La plataforma PLATINO se utiliza actualmente en el proyecto de investigación PLATINO y ha sido publicada con éxito en distintos artículos de investigación [4, 16, 29]. En el proyecto PLATINO de agricultura inteligente se usa como pasarela LoRa un dron, el cual sobrevuela el campo donde

estén todas las plataformas PLATINO desplegadas, e irá recibiendo la información recogida por cada una de dichas plataformas. En nuestro TFG, se usará una plataforma LoRa con conexión a una red Wi-Fi para que se pueda acceder a una plataforma IoT.

A continuación se va a hablar de la funcionalidad de la plataforma PLATINO. Como ya se comentó anteriormente, esta dispone de dos micro-sistemas: Un Smart Farming Device y un Data Logger Device. El Smart Farming Device se dedica a recoger datos relacionados con el medio ambiente a través de los sensores ya comentados anteriormente. Estos datos recogidos por los sensores, el Data Logger Device los recoge del Smart Farming Device, ya que estos dispositivos se comunican a través del puerto serie. Una vez que el Data Logger Device tiene estos datos, aparte también recoge los datos relacionados con las condiciones energéticas como ya se comentó antes. Estos datos los puede guardar en una tarjeta MicroSD o los puede enviar a través de LoRa a la pasarela.

Cabe destacar que la plataforma PLATINO no es una plataforma *ad-hoc*, ya que está pensada para la ejecución de múltiples aplicaciones: para la recogida de datos a través de sensores, para temas de consumo energético o de producción, etc. En otras palabras, la plataforma PLATINO es una plataforma multipropósito.

Algoritmos neutrales en energía

En las redes inalámbricas de sensores, los *sensores* son pequeños dispositivos embebidos inalámbricos capaces de monitorizar en tiempo real distintos fenómenos medio ambientales, como la temperatura, la humedad o la velocidad del viento, a través de transductores, y con capacidad de comunicación para enviar datos a otros sensores en la misma red inalámbrica o bien a una pasarela o *gateway* que permite comunicar con otra red externa.

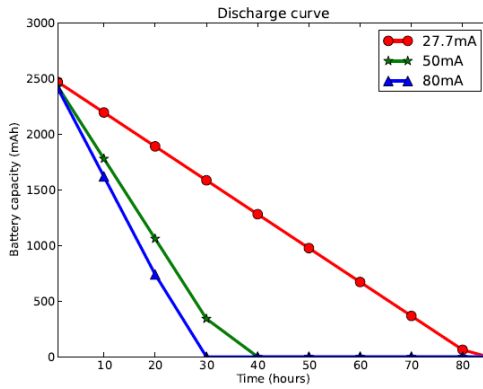
Normalmente, estos sensores están alimentados a través de una batería de capacidad limitada. Dicha capacidad se mide en miliamperios por hora (mAh). Estas baterías alimentan la circuitería de los sensores, proporcionándoles la corriente eléctrica necesaria para poder operar. Cada uno de los componentes hardware que integra un sensor requiere una intensidad de corriente distinta y además, cada componente puede operar en distintos estados energéticos, cada uno de ellos con un consumo distinto. El consumo de energía de un determinado componente se puede expresar con la siguiente ecuación:

$$E = V * I * T \quad (4.1)$$

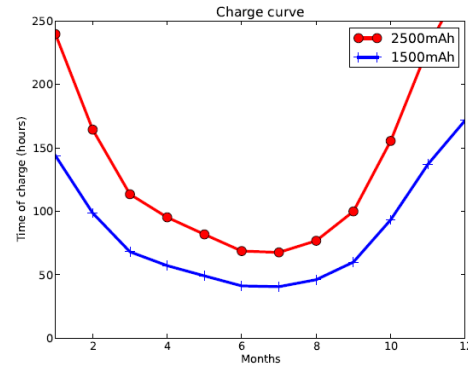
donde: V es el voltaje del componente en uso, I la intensidad de corriente de dicho componente y T el tiempo de uso del componente. La energía resultante E o consumo energético se mide en julios (J).

El consumo total de energía de un sensor es la suma de la energía que necesita cada uno de los componentes que una aplicación usa. Por lo tanto, la vida útil del sensor es el tiempo necesario para descargar su batería por debajo del nivel mínimo de carga que se requiere para mantener la operación del sensor. En la Figura 4.4a se puede apreciar las curvas de descarga de una batería dependiendo del consumo de corriente.

Para que los sensores estén funcionando de forma continua y poder recargar las baterías del sensor, se necesita el uso de un recolector de energía, como por ejemplo un panel solar o un molino de viento eléctrico. En la Figura 4.4b se puede apreciar una gráfica con la curva de carga de una batería a través de un recolector de energía dependido de la producción solar en cada mes del año (datos basados en el artículo [15]). Como se puede apreciar, en los meses centrales se tiene mayor producción energética, por lo que cuesta menos poder cargar la batería que en los otros meses. En la Figura 4.5 se puede observar una gráfica de producción energética por horas dependiendo del valor medio de irradiación mensual. Como se puede ver, la producción solar diaria teórica genera una gráfica con forma de parábola, donde la producción comienza a crecer desde el amanecer hasta las horas centrales del día y luego decrece hasta convertirse en 0 en el atardecer. Por desgracia, en escenarios reales, la producción solar, aunque puede predecirse podría sufrir desviaciones y, generalmente, no es



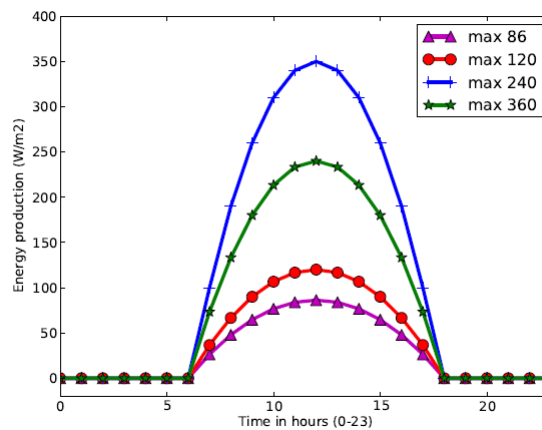
(a) Gráfica de descarga de la batería



(b) Gráfica de tiempo de carga de la batería según el mes del año

Figura 4.4: Gráficas de descarga y carga de batería. Imágenes tomadas de [15]

una producción continua sino intermitente, por lo que todavía se necesita el uso de una batería que permita almacenar energía y ser recargada para proporcionar suministro eléctrico en esos períodos en que no existe producción de energía.

**Figura 4.5:** Producción energética por horas para diferentes valores medios de irradiación mensual. Imagen tomada de [15]

Cuando la producción de energía del recolector es mayor que la necesitada por el sensor, lo usual es que el excedente de la misma se pueda utilizar para recargar la batería y, cuando la producción no es suficiente, la batería se usa para alimentar al nodo. Sin embargo, se puede dar el caso en el que la carga de la batería no sea la suficiente para alimentar al sensor, y que la energía producida por el colector tampoco lo sea. Esto hará que nuestro sensor esté inactivo hasta que el colector o la batería puedan proporcionarle energía para seguir funcionando. Para reducir estos tiempos de espera de los nodos, e incluso evitarlos, los ciclos de carga de batería y producción del recolector se deben de gestionar adecuadamente. Para ello, se necesita implementar algoritmos que mantengan nuestros dispositivos neutrales en energía.

Que nuestro dispositivo se mantenga neutral en energía quiere decir que, dado un período de tiempo $[a, b]$ la energía consumida en el período tiene que ser siempre menor o igual que la energía producida en el mismo período. Con el uso de una batería, esto quiere decir que, el nivel de batería al final de un determinado espacio de tiempo tiene que ser mayor o igual que al principio de ese espacio de tiempo. En la Figura 4.6 se puede observar una gráfica con el nivel de batería por espacio, o slot de tiempo, en la cual, el nivel de batería en el instante b (el final del período) es mayor que en el instante a (el inicio del período), por lo que se garantiza que existe neutralidad energética. Los algoritmos *neutrales en energía*, combinados adecuadamente con los sistemas recolectores de

energía, proporcionan al nodo la valiosísima característica de extender el tiempo de vida del nodo e incluso conseguir la ejecución ininterrumpida del mismo, lo cual tiene importantes ventajas como son el ahorro del tiempo y costes de mantenimiento y reemplazo en las baterías, y proporcionar un conocimiento del medio continuo y completo.

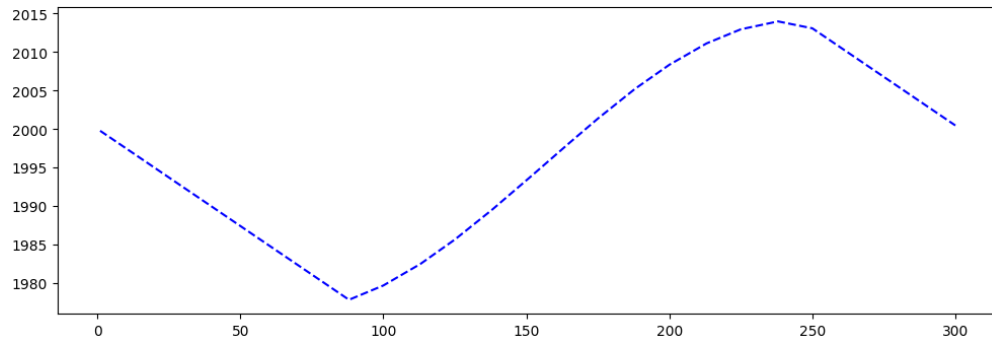


Figura 4.6: Gráfica de nivel de batería (eje y) por instante de tiempo (eje x)

Existen en el estado del arte muchos algoritmos que tratan de conseguir este objetivo. A continuación se exponen algunos de ellos:

- El primero de estos algoritmos es el propuesto por Kansal y Srivastava en [28], el cual defendía la asignación de tareas distribuidas entre los dispositivos de recolección de una red de acuerdo con su nivel de energía. También propusieron, años más tarde, maximizar el rendimiento de los dispositivos escalando dinámicamente sus ciclos de trabajo. Esta propuesta fue validada por medio de un experimento real donde una plataforma Heliomote fue desplegada en una área residencial en Los Ángeles y se estuvo ejecutando durante 67 días en el verano de 2005.
- LSA[37] es un algoritmo de programación óptimo en tiempo real que encuentra el programa óptimo factible donde se pueden cumplir los plazos de la tarea. Mediante simulación, se demuestra la viabilidad de su enfoque proporcionando cuántas tareas y en qué orden se ejecutarán, y cuantifican la sobrecarga impuesta por el planificador.
- Tal y como se propone en [14], se considera solo un recolector de energía y un modelo básico de tareas en el que cada tarea tiene una calidad relacionada a su frecuencia de muestreo. Un planificador debe encontrar una asignación de tareas a espacios de tiempo durante el período de referencia, tal que la calidad general se maximice en ese período de tiempo. Al final de cada slot i se decide que tarea debe tener el slot de tiempo $i + 1$ dependiendo del nivel de batería, que se computa en función de producción real de los slots pasados y el actual, el consumo de las aplicaciones ya ejecutadas, y de una estimación de la producción energética de los próximos slots de tiempo.

En este TFG, se ha optado sin embargo por implementar el algoritmo publicado en [15] sobre el nodo PLATINO y evaluar su capacidad de recolección de energía sobre un dispositivo real. El algoritmo obtiene la planificación de aplicaciones (sub)óptima que mantiene el nodo neutral en energía y además consigue maximizar la calidad de servicio (QoS) proporcionada por las aplicaciones que se ejecutan durante el período de referencia que es 24 horas. Para ello, es preciso implementar un algoritmo que optimice la QoS dependiendo de la producción de energía, el consumo de energía, y la carga de batería. El algoritmo consiste en encontrar las aplicaciones que puedan ser asignadas a cada *slot* de tiempo (unidad mínima de tiempo en que se divide el período de referencia) tal que dicha asignación mantiene el nodo neutral en energía y además maximiza la calidad de servicio. Para ello, el algoritmo parte de una asignación inicial de aplicaciones a slots, que se irá optimizando en sucesivas iteraciones hasta encontrar una asignación tal que ya no se pueda ser mejorada.

Este algoritmo, considera un conjunto de aplicaciones "candidatas" que podrían ser asignadas en

cada slot. Para modelar la aplicación, se consideran un conjunto de tareas que pueden ser implementadas para proporcionar una mayor o menor calidad de servicio. Por ejemplo, una de las principales tareas de los sensores es la de monitorizar el entorno, la cual se puede efectuar de diferentes maneras dependiendo, por ejemplo, de las frecuencias de muestreo, donde una frecuencia alta implica mayor consumo y mayor nivel de calidad de servicio. También se puede tener varios transductores trabajando a la vez y cada uno con distintos costes energéticos, aparte de tener distintos patrones de comunicación. Por ello, para modelar las aplicaciones que se ejecutan en el nodo, se pueden considerar las distintas formas que se tienen para poder realizar las tareas que realizan los sensores y las distintas calidades de servicio para cada una de esas formas. El objetivo del algoritmo es obtener el plan de ejecución, el conjunto de aplicaciones más adecuadas para todos los slots del período de referencia que cumpla estas dos condiciones: mantener el nodo neutral en energía y maximizar la calidad de servicio de las aplicaciones asignadas a cada slot. Para conseguir este objetivo, se debe crear un planificador que encuentre ese plan de ejecución.

El algoritmo en el cual nos vamos a basar, publicado en [15], presenta un modelo de tareas más refinado, en el que cada tarea tiene un tiempo de ejecución, un consumo y un período, y en el que se construyen varios planes de programación a partir de un subconjunto de tareas definidas de esta manera. En este algoritmo, el planificador asigna inicialmente a cada uno de los slots el plan más eficiente, esto es, el que mejor relación entre calidad de servicio y consumo tiene, y después, siguiendo una estrategia voraz, se intenta mejorar (*Upgrade*) la calidad de la asignación dependiendo del nivel de batería y de la energía producida en ese slot. Si la asignación inicial calculada ya no se puede mejorar, al contrario, no mantiene el sistema neutral en energía, entonces se procederá a degradar dicha asignación (*Downgrade*), asignando planes menos eficientes con el fin de convertir el sistema neutral en energía. En este algoritmo, se identifican 4 funciones fundamentales: la asignación principal, la función (*Upgrade*), la función (*Downgrade*) y la función de reoptimización de los planes de ejecución. A continuación se van a describir cada uno de ellas.

Asignación Inicial. La primera parte del algoritmo sería la asignación inicial, tal y como se puede ver en el Algoritmo 4.1. Este algoritmo consigue de forma progresiva una solución subóptima en varias iteraciones. Empieza inicializando cada slot con el plan de ejecución mas eficiente (dado que los planes se ordenan de mayor a menor eficiencia en un vector P , el primer plan y el más eficiente es $P[1]$). Tras asignar el plan, el algoritmo evalúa la condición de neutralidad energética comparando el nivel de la batería en el slot 0 con el de el slot k , el cual es el último. Se consigue una solución óptima si el nivel de la batería es igual en ambos slots. Por el contrario, pueden ocurrir dos casos:

1. Si el nivel de batería en el slot k es mayor que el nivel de batería en el slot 0, esto quiere decir que la producción energética estimada es mayor que la consumida, y por lo tanto, existe la posibilidad de encontrar una asignación mejor que la inicial. Entonces, el algoritmo selecciona el siguiente plan en el vector $P[]$ menos eficiente, pero con mayor calidad y más consumo, el cual reemplaza al plan asignado inicialmente en cada uno de los slots. Este reemplazo se hace slot a slot, hasta llegar a un slot en el cual el reemplazo convierte el sistema no neutral en energía, o bien, hasta hacer el reemplazo en el último slot k , en cuyo caso, se vuelve a probar con los siguientes planes en el vector P , menos eficientes, pero de mayor calidad y, por tanto, mayor consumo. Este algoritmo se lleva a cabo por la función *Upgrade*, de la que se hablará más adelante.
2. Si el nivel de batería en el slot k es menor que el nivel de batería en el slot 0, esto quiere decir que la actual asignación no es sostenible, por lo que el algoritmo debe de seleccionar el plan en $P[]$ que tenga el siguiente menor coste que el plan asignado, el cual reemplazará al plan asignado en cada uno de los slots. Este reemplazamiento se hace hasta que se consiga que la solución sea neutral en energía, y es llevada a cabo por la función *Downgrade*, de la que se hablará más tarde.

El algoritmo continua iterando hasta que no exista ningún plan en $P[]$ que mejore la solución obtenida.

Algoritmo 4.1: Algoritmo de asignación inicial

Data: $P[]$: el vector de planes de ejecución
Result: $S[]$: el vector de asignación de planes de ejecución
 $S[s] = P[1] \forall s$
case 1: $(B(k) \geq B(0)) \wedge (B(i) \geq B_{min} \forall i)$
loop
 $P'[] = P[]|_q$;
 if $B(k) == B(0) \vee P'[] == 0$ **then**
 | *the solution is optimal; exit;*
 end
 $S[] = Upgrade(S[], P'[1])$;
 $P[] = P'[]$;
endLoop
case 2: $(B(k) < B(0)) \vee (B(i) < B_{min} \text{ for some } i)$
loop
 $P'[] = P[]|^{cp}$;
 if $P'[] == 0$ **then**
 | *there is not admissibel solution; exit;*
 end
 $S[] = Downgrade(S[], P'[1])$;
 $P[] = P'[]$;
 if $B(k) \geq B(0) \vee B(i) \geq B_{min} \forall i$ **then**
 | *go to case 1;*
 end
endLoop

Upgrade y Downgrade. Las funciones *Upgrade* y *Downgrade* se muestran en los Algoritmos 4.2 y 4.3, respectivamente. Estas funciones toman una asignación de planes de ejecución $S[]$, y comienzan a reemplazar slot a slot cada uno de sus elementos con un nuevo plan P_i , empezando desde un slot inicial. Por un lado, la función *Upgrade* reemplaza elementos de $S[]$ con planes de ejecución que tengan mayor nivel de calidad y coste. Por esta razón, se empieza a reemplazar a partir del slot correspondiente a la salida del sol (el slot r_1), ya que en estos slots la producción energética compensará al mayor nivel de consumo. La función *Upgrade* reemplaza elementos siempre y cuando se cumpla con la condición de neutralidad en energía y el nivel de batería sea al menos B_{min} para

todos los slots.

Algoritmo 4.2: Función *Upgrade*

Data: $S[]$: el vector de asignación de planes de ejecución; P_i : un plan de ejecución
Result: $S'[]$: el vector de asignación resultante
 $S'[] = S[]$;
 $s = r_1; j = 1$;
while $((B(k) - B(0) \geq c_i - c(s)) \wedge (j \leq k))$ **do**
 $H = S'[s]; S'[s] = P_i$;
 if $B(i) < B_{min}$ *for some* i **then**
 $S'[s] = H$;
 return $S'[]$ % undo the assignment and exit
 end
 $s = (s + 1) \bmod k; j++$;
end
return $S'[]$

Por el contrario, la función *Downgrade* reemplaza elementos de $S[]$ con planes de ejecución que tienen un coste y calidad menor. Por esta razón, empieza a reemplazar a partir del slot correspondiente a la puesta del sol (el slot r_2) en aquellos slots en los que no haya producción de energía. Esta función deja de reemplazar elementos cuando la asignación se convierte neutral en energía y el nivel de batería en cada slot es de al menos B_{min} .

Algoritmo 4.3: Función *Downgrade*

Data: $S[]$: el vector de asignación de planes de ejecución; P_i : un plan de ejecución
Result: $S'[]$: el vector de asignación resultante
 $S'[] = S[]$;
 $s = r_2; j = 1$;
while $((B(k) - B(0) < 0) \vee (B(i) < B_{min}) \text{ for some } i \wedge (j \leq k))$ **do**
 $S'[j] = P_i$;
 $s = (s + 1) \bmod k; j++$;
end
return $S'[]$

Reoptimización. Por último, vamos a hablar del algoritmo de reoptimización, el cual se puede ver representado en el Algoritmo 4.4. Como ya se ha visto en el Algoritmo 4.1, se obtiene un vector de asignación inicial que debe mantenerse para todos los slots en un determinado tiempo de referencia que, en nuestro caso, es un día, basándonos en las estimaciones de la producción energética que se dan en un lugar geográfico y en un tiempo y una hora y el coste de los planes de ejecución. Sin embargo, en tiempo de ejecución las condiciones solares pueden variar, derivando en un consumo mayor o menor del esperado, haciendo que la asignación inicial sea ineficiente o no neutral en energía, por lo que se debe de adaptar a las nuevas condiciones. Para tal finalidad, se ha propuesto el Algoritmo 4.4. Este algoritmo reoptimiza la asignación empezando desde el slot x hasta el slot k . Se asume que en el slot $x - 1$ se detecta una diferencia entre el consumo real y el detectado. Pueden ocurrir dos casos:

1. Si la carga esperada de la batería al final del slot k excede al nivel de batería en el slot 0 en más del parámetro ϵ , quiere decir que el presupuesto energético ha aumentado y la asignación actual puede ser mejorada para maximizar la calidad de servicio resultante. Para este propósito, el algoritmo sigue asignando mejores planes entre los slots x y k . Esta mejora se sigue realizando hasta que ya no exista ningún plan que mejore la solución obtenida o que haga que el sistema no sea neutral en energía.

2. Por el contrario, si la carga de la batería en el slot k es menor que la del slot 0 por más del parámetro ϵ , o bien $B(i) < B_{min}$ para algún slot i , el algoritmo busca una asignación con menor coste. Para ello, el algoritmo trata de encontrar el siguiente plan que degrada la solución inicial y lo asigna entre los slots x y k . Esta mejora se sigue realizando hasta que ya no exista ningún plan que mejore la solución obtenida o la asignación se vuelva otra vez neutral en energía y el nivel de batería es de al menos B_{min} para todos los slots.

Algoritmo 4.4: Algoritmo de reoptimización

Data: $S[]$: el vector de asignación de planes de ejecución; $P[]$: el vector de planes de ejecución; ϵ : parámetro de tolerancia

Result: $S'[]$: el vector de asignación resultante

$S'[] = S[]$;

case 1: $(B(k) \geq B(0) + \epsilon) \wedge (B(i) \geq B_{min} \forall i)$

Replaced = True;

while Replaced **do**

$s = x$; Replaced = True;

while $s \leq k$ **do**

Let P_j be the scheduling plan of $S'[]$;

if $(j < n) \wedge (B(k) - B(0) \geq c_{j+1} - c_j)$ **then**

$S'[s] = P_{j+1}$; Replaced = True;

if $(B(i) < B_{min} \text{ for some } i)$ **then**

$S'[s] = P_j$;

Replaced = False % undo the assignment;

end

end

$s++$;

end

end

case 2: $(B(k) \leq B(0) - \epsilon) \vee (B(i) < B_{min} \text{ for some } i)$

Replaced = True;

while Replaced **do**

$s = x$; Replaced = False;

while $s \leq k$ **do**

Let P_j be the scheduling plan of $S'[s]$;

if $(j > 1) \wedge ((B(k) < B(0)) \vee (B(i) < B_{min} \text{ for some } i))$ **then**

$S'[s] = P_{j-1}$;

Replaced = True;

end

$s++$;

end

end

Para nuestro proyecto, vamos a utilizar los algoritmos anteriormente descritos, adaptándolos para los microcontroladores Adafruit Feather M0. Esta implementación se explica en las siguientes iteraciones.

Plataformas en la nube para IoT

En el Capítulo 2 se describió la necesidad de usar alguna plataforma en la nube que esté enfocada para aplicaciones IoT que proporcionara la funcionalidad para almacenar los datos recolectados por los sensores y también la funcionalidad para acceder a los datos por parte de los usuarios. A

continuación, se van a exponer algunas posibles alternativas para poder cumplir este requisito.

- **Sofia2**[27]: Sofia2 es una plataforma IoT & Big Data pensada para facilitar y acelerar la construcción de nuevos sistemas y soluciones digitales, para así lograr la transformación y disrupción de los negocios. Sofia2 surge de un proyecto I+D europeo denominado SOFIA. SOFIA es el acrónimo de *Smart Objects For Intelligent Applications*, una plataforma que surge de un proyecto de I+D de tres años llamado Artemis, el cual fue finalizado en marzo de 2012, en el que participaron 19 partners de cuatro países miembros de la UE, entre los cuales se incluye Nokia, Philips, Fiat Acciona e Indra. Tras el proyecto Artemis, Indra ha continuado desarrollando el proyecto SOFIA, creando a partir del mismo una plataforma enfocada a su uso empresarial, llamada Sofia2. Sofia2 está enfocada en las áreas de adaptación al mundo empresarial, interfaces Big Data, capacidades de integración con backends, o el uso de interfaces REST.
- **IBM Watson IoT**[22]: IBM Watson IoT es un servicio gestionado alojado en cloud creado por el gigante informático IBM. Esta plataforma presenta funcionalidades que permiten realizar una gestión global, sencilla e intuitiva de los dispositivos desplegados y de la información que proporcionan. Además, permite conectar servicios adicionales de manera muy simple, servicios que incluyen capacidades de analítica, procesamiento de eventos y *Blockchain*. De esta manera, toda la información registrada y provista por los dispositivos es automáticamente procesada en la plataforma, obteniendo en tiempo real métricas e indicadores calculados que resultan de gran utilidad para los usuarios, tanto aquellos que necesitan conocer el estado de los despliegues para saber si todo funciona correctamente, como para los responsables que requieren una visión de más alto nivel.
- **Azure IoT Hub**[10]: Azure IoT Hub es un servicio administrado para habilitar la comunicación bidireccional entre los dispositivos IoT y la IaaS de Microsoft, Azure. Permite crear un canal de comunicación seguro para enviar y recibir mensajes desde los dispositivos IoT, administrar y aprovisionar dispositivos integrados para administrar los dispositivos, y compatibilidad con Azure IoT Edge para crear aplicaciones IoT híbridadas.
- **FIWARE Lab**[8]: FIWARE Lab es un entorno donde se pueden probar todas las aplicaciones y tecnologías de FIWARE, la cual es una plataforma de código abierto, impulsada por la Unión Europea, para el desarrollo y despliegue global de aplicaciones de Internet de las Cosas, o IoT. Fue iniciado por el VII Programa Marco de la Unión Europea[17] dentro de su proyecto de colaboración público privada para el Internet del Futuro. FIWARE Lab es un entorno solamente de prueba y no comercial, en el que facilitan al usuario la creación de máquinas virtuales en la nube y la utilización de los diferentes *Generic Enablers* (GE) desarrollados por FIWARE. También se permite el uso de datos de organizaciones publicas que ya están colgados en dicha plataforma así como alguno de los sensores abiertos situados por Europa.

Tras ver las opciones que tenemos para usar en el proyecto, se tiene que elegir una de las plataformas citadas anteriormente. Se ha intentando acceder al servicio de Sofia2, pero al parecer no es accesible para todo el mundo, solo es accesible para los desarrolladores de Indra, los cuales son los que están desarrollando dicha plataforma. IBM Watson IoT y Azure IoT Hub son plataformas de pago. Por lo tanto, para poder ahorrar coste en el proyecto, se ha decidido que la plataforma a usar para el proyecto sea FIWARE Lab, ya que es accesible y ofrece un servicio gratuito.

4.3. FASE DE CONSTRUCCIÓN

De acuerdo a la Tabla 3.1 la fase de Construcción está formada por cuatro iteraciones, que se desarrollan a continuación.

4.3.1. Iteración 2. Implementación y evaluación del algoritmo neutral en energía

En esta iteración se va a explicar como se ha realizado la implementación del algoritmo neutral en energía presentado en la sección anterior, y que debe ejecutar sobre la plataforma PLATINO integrada por los dispositivos Adafruit Feather M0. Además, se evaluará el comportamiento de dicho algoritmo, mostrando los resultados que permiten validar su correcto funcionamiento. Todos los algoritmos han sido programados en C++ mediante la herramienta Visual Studio Code.

El algoritmo de asignación de planes tiene la estructura típica de un programa C++ con dos archivos: el primero una librería en la que se define la clase con sus métodos y parámetros necesarios, y por otro lado otro archivo en el que se implementa el código de dichos métodos. En esta sección nos vamos a centrar en los métodos que más nos interesan que son los que se representaron con anterioridad en los Algoritmos 4.1, 4.2, 4.3 y 4.4, es decir, asignación inicial, los métodos *Upgrade* y *Downgrade*, y el método para reoptimizar la asignación inicial.

Para la implementación de este algoritmo, se han utilizado distintos parámetros, como se pueden observar en la Tabla 4.10, los cuales se describen a continuación:

- El parámetro CAPACITY hace referencia a la capacidad total de la batería que se va a utilizar para el proyecto, la cual se mide en mAh.
- BATTERIES representa el número de baterías que se utilizan para el algoritmo.
- VOLTAGE es el voltaje de las baterías, el cual se mide en voltios (V).
- EFFICIENCY es la eficiencia del panel solar.
- S se refiere a la superficie del panel solar medido en m^2 .
- Vld se refiere al voltaje específico de carga, el cual se mide en voltios (V).
- H hace referencia a la hora de mayor producción solar del día, en nuestro caso, las 12:00 am.
- MONTHS hace referencia al número de meses que tiene el año.
- SUNRISE_SLOT representa la hora en la que se produce la salida del sol, para luego más tarde usar esta variable para obtener el slot de tiempo al que hace referencia.
- SUNSET_SLOT representa la hora en la que se produce la salida del sol, para luego más tarde usar esta variable para obtener el slot de tiempo al que hace referencia.
- EPSILON hace referencia al valor del ϵ usado en la reoptimización.
- COEFFICIENT hace referencia al valor usado en la reoptimización para aumentar o reducir el consumo.
- NUMBER_OF_PLANS es el número de planes de ejecución que tiene el dispositivo.
- ENERGY_IDLE_TASK es el consumo energético de la tarea inactiva.
- TUPLA hace referencia al número de elementos que tiene cada una de las tareas.
- TIME_SLOTS es el tiempo que dura cada uno de los slots de tiempo.
- SLOTS es el número de slots que tiene un día.
- QoS hace referencia a los distintos valores de calidad de servicio que tiene cada uno de los 5 planes de ejecución, más otro para el plan auxiliar.
- D indica los valores de irradiancia mensual medidos por el panel solar en Madrid. Estos valores se miden en W/m^2 .
- STD_HOURS hace referencia a la media del número de horas de sol hay en cada mes del año.
- MIN_HOURS hace referencia a la media de la hora en la que amanece en cada mes del año.
- MAX_HOURS hace referencia a la media de la hora en la que se pone el sol en cada mes del año.

Además de estos valores, se necesita definir los *planes de ejecución*, es decir, distintas aplicaciones "candidatas" que pueden ser seleccionadas para ejecución en cada slot por el algoritmo neutral en energía. Cada plan de ejecución, estará formada por un conjunto variable de tareas, donde cada una de estas tareas se define mediante una tupla de tres elementos: $\langle t, p, w \rangle$ donde t es el tiempo de ejecución de la tarea, p es el período de la tarea y w es su consumo. Para nuestra evaluación, se han definido un total de 6 planes de ejecución $P_1, P_2, P_3, P_4, P_5, P_{aux}$, cuya descripción se muestra en

| Variable | Valor |
|------------------|---|
| CAPACITY | 2500 |
| BATTERIES | 2 |
| VOLTAGE | 1.5 |
| EFFICIENCY | 11.38 |
| S | 0.0036 |
| Vld | 3.3 |
| H | 12 |
| MONTHS | 12 |
| SUNRISE_SLOT | 6 |
| SUNSET_SLOT | 18 |
| EPSILON | 0.0001 |
| COEFFICIENT | (-0.01) |
| ENERGY_IDLE_TASK | 2 |
| TUPLA | 3 |
| TIME_SLOTS | 288 |
| SLOTS | (3600 * 24 / TIME_SLOTS) |
| D | 84.58, 123.33, 178.75, 212.92, 247.92, 295.42, 300.00, 264.17, 202.92, 130.42, 88.75, 70.83 |
| STD_HOURS | 8, 8, 8, 10, 11, 13, 13, 13, 12, 9, 8, 8 |
| MIN_HOURS | 8, 8, 8, 8, 7, 7, 7, 7, 8, 8, 8 |
| MAX_HOURS | 16, 16, 16, 18, 18, 20, 20, 20, 19, 17, 16, 16 |

Tabla 4.10: Parámetros utilizados

la Tabla 4.11. El P_{aux} es un plan auxiliar que tiene el mínimo consumo y por tanto proporciona la mínima QoS. La primera columna de la tabla representa los planes de ejecución, en las siguientes 4 columnas se describen las tareas que tienen cada uno de ellos (T_1, T_2, T_3, T_4), donde cada tarea se representa mediante la tupla de 3 valores descritos anteriormente, y el resto de columnas describen el consumo del plan, la QoS del plan y su eficiencia, calculada como el cociente entre la QoS y el consumo del plan.

| Plan | T_1 | T_2 | T_3 | T_4 | Coste | QoS | Eficiencia |
|-----------|----------------|----------------|-------------|-------------|-------|--------|------------|
| P_1 | < 1, 2, 3 > | < 1, 2, 4 > | < 1, 2, 2 > | - | 3.5 | 100 % | 28.57 |
| P_2 | < 3, 5, 2 > | < 1, 3, 3 > | < 1, 3, 3 > | < 1, 3, 5 > | 2.866 | 87.5 % | 30.53 |
| P_3 | < 7, 10, 10 > | - | - | - | 1.6 | 75 % | 46.87 |
| P_4 | < 8, 15, 10 > | < 10, 15, 12 > | - | - | 1.06 | 62.5 % | 58.96 |
| P_5 | < 15, 20, 10 > | - | - | - | 1 | 50 % | 50 |
| P_{aux} | < 18, 20, 2 > | - | - | - | 0.3 | 70 % | 233 |

Tabla 4.11: Parámetros de los planes de ejecución

Implementación del algoritmo de asignación inicial

En el Listado 4.1 se muestra el código principal del algoritmo de asignación inicial de planes de ejecución. Como se puede ver en el mismo, se hace un bucle *while* hasta que se encuentre una solución que sea óptima o admisible a partir de la línea 19. En la línea 20, comprueba si el nivel de batería en el último slot es mayor o igual al del slot inicial. Si es así, aparte se comprueba si el nivel de batería del último slot y el del primero son iguales, y si lo son, es que se ha encontrado una solución óptima. Si no es así, se buscará el siguiente plan con mayor calidad, como se puede ver en la línea 25. Si no hay ninguno, significa que se ha llegado a una solución óptima, y si hay alguno, se llama al método para mejorar la asignación (*Upgrade*) como se puede ver en la línea 29. Si el nivel de batería en el último

slot no es mayor o igual al del slot 0, entonces buscamos el siguiente plan con menor coste y calidad, como se puede ver en la línea 32. Si no queda ningún otro plan con menor coste o calidad, hemos obtenido una solución que es admisible. Si no es así, tenemos que llamar al método *Downgrade*, tal y como se hace en la línea 36. Tras hacer la asignación, en la línea 43 se hace una reoptimización de dicha asignación

Listado 4.1: Algoritmo de asignación inicial

```

1  int* assignmentClass::assign_plan(int month) {
2      /* inicializar vector de tiempo */
3      int hour = 0;
4      int x, r1, r2;
5      compute_efficiency(QoS, cost_of_plan, vector_efficiency, ↵
        ↵ NUMBER_OF_PLANS_1);
6      order_plans_by_efficiency(vector_efficiency, plans, ↵
        ↵ NUMBER_OF_PLANS_1);
7      get_time_from_seconds(0, ddMMyyhhmmss);
8
9      r1 = look_for_slot(SUNRISE_SLOT);
10     r2 = look_for_slot(SUNSET_SLOT);
11     x = rand() % (r2 - r1 + 1) + r1;
12
13     for (int i = 0; i < SLOTS; i++) assignments[i] = plans[0];
14
15     compute_cost_assignment(plans[0], cost_of_plan, ↵
        ↵ ddMMyyhhmmss, battery_at_slots, &hour, 1, ↵
        ↵ ddMMyyhhmmss[1] - 1);
16
17     int optimal = 0, admisible = 0, n = NUMBER_OF_PLANS_1;
18
19     while ((!optimal) && (!admissible)) {
20         if (battery_at_slots[SLOTS] >= battery_at_slots[0]) {
21             if (battery_at_slots[SLOTS] == battery_at_slots[0]) {
22                 optimal = 1; break;
23             }
24
25             n = remove_plan_quality(plans[0], ↵
                ↵ vector_efficiency, plans, n);
26             if (n == 0) {
27                 optimal = 1; break;
28             }
29             upgrade(plans[0], assignments, battery_at_slots, ↵
                ↵ cost_of_plan, QoS, ddMMyyhhmmss[1] - 1);
30         }
31         else {
32             n = remove_plan_cost(plans[0], cost_of_plan, ↵
                ↵ vector_efficiency, plans, n);
33             if (n == 0) {
34                 admisible = 1; break;
35             }
36             downgrade(plans[0], assignments, battery_at_slots, ↵
                ↵ cost_of_plan, QoS, ddMMyyhhmmss[1] - 1);
37         }
38     }
39     compute_qos_assignment(assignments, QoS, ddMMyyhhmmss[1]);
40     compute_efficiency(QoS, cost_of_plan, vector_efficiency, ↵
        ↵ NUMBER_OF_PLANS_1);
41     order_plans_by_efficiency(vector_efficiency, plans, ↵
        ↵ NUMBER_OF_PLANS_1);
42
43     if (reoptimization(assignments, x, battery_at_slots, ↵
        ↵ ddMMyyhhmmss[1] - 1, cost_of_plan, QoS, plans) == -1)
44         std::cout << "Not␣admissible␣solution" << std::endl;

```

```

45
46     return assignments;
47 }

```

Implementación del método *Upgrade*

En el Listado 4.2 se puede apreciar el código del método *Upgrade*. Primero, se calculan los consumos para el plan nuevo y actual, y se obtiene el slot en el que amanece. Después, en la línea 15 se hace un bucle *while*, en el que comprobamos si la diferencia entre el nivel de batería del último slot con el primero es mayor o igual que la diferencia de los consumos que hemos calculado antes. Si es así, en el slot que hemos obtenido anteriormente, se le asigna ese plan nuevo. Esto se hace hasta que no se cumpla la condición del bucle *while*.

Listado 4.2: Código del método *Upgrade*

```

1 void assignmentClass::upgrade(int plan, int *assignments, ↵
    ↵ double *battery_at_slots, float *cost_of_plan, float ↵
    ↵ *QoS, int month)
2 {
3
4     int i = 0;
5     int s;
6     double consumption_new_plan = 0.0;
7     double consumption_old_plan = 0.0;
8     int old;
9
10    s = look_for_slot(SUNRISE_SLOT);
11    consumption_new_plan = (cost_of_plan[plan] / (60 * 60)) * Vld ↵
    ↵ * TIME_SLOTS / (VOLTAGE * BATTERIES);
12
13    consumption_old_plan = (cost_of_plan[assignments[s - 1]] / ↵
    ↵ (60 * 60)) * Vld * TIME_SLOTS / (VOLTAGE * BATTERIES);
14
15    while (((battery_at_slots[SLOTS] - battery_at_slots[0]) >= ↵
    ↵ (consumption_new_plan - consumption_old_plan)) && (i < ↵
    ↵ SLOTS))
16    {
17        old = assignments[s - 1];
18        consumption_old_plan = (cost_of_plan[old] / (60 * 60)) * ↵
    ↵ Vld * TIME_SLOTS / (VOLTAGE * BATTERIES);
19        assignments[s - 1] = plan;
20        recompute_battery_level(plan, assignments, ↵
    ↵ battery_at_slots, cost_of_plan, s, SUNRISE_SLOT, QoS, ↵
    ↵ month, 0);
21        s = (s % SLOTS) + 1;
22        i++;
23    }
24 }

```

Implementación del método *Downgrade*

En el Listado 4.3 se puede apreciar el código del método *Downgrade*. Primero, se obtiene el slot en el que se produce la puesta de sol. Después, en el bucle *while* de la línea 7 se comprueba si la diferencia de nivel de batería del último y primer slot es menor que 0. Si es así, se asigna en el slot que obtuvimos antes el nuevo plan.

Listado 4.3: Código del método *Downgrade*

```

1 void assignmentClass::downgrade(int plan, int *assignments, ↵
  ↵ double *battery_at_slots, float *cost_of_plan, float ↵
  ↵ *QoS, int month)
2 {
3     int i = 0;
4     int s;
5     int old;
6     s = look_for_slot(SUNSET_SLOT);
7     while (((battery_at_slots[SLOTS] - battery_at_slots[0]) < 0) ↵
  ↵ && (i < SLOTS))
8     {
9         old = assignments[s - 1];
10        assignments[s - 1] = plan;
11        recompute_battery_level(plan, assignments, ↵
  ↵ battery_at_slots, cost_of_plan, s, SUNSET_SLOT, QoS, ↵
  ↵ month, 0);
12        s = (s % SLOTS) + 1;
13        i++;
14    }
15 }

```

Implementación del algoritmo de reoptimización

En el Listado 4.4 se puede apreciar el código que se ha usado para el algoritmo de la reoptimización de la asignación de planes de ejecución. Lo primero que se hace es comprobar si el nivel de la batería en último slot es mayor o igual al del slot 0 más la variación *epsilon* ϵ . Si esto es así, el algoritmo intenta maximizar la calidad de servicio, asignando planes de ejecución que tengan mayor calidad de servicio, y por consiguiente, mayor coste, a los slots entre x y k (el último slot). Si esto no es así, se hará lo contrario: se asignarán planes de ejecución con menor calidad de servicio a los slots que están entre x y k .

Listado 4.4: Algoritmo para la reoptimización

```

1 if (battery_at_slots[SLOTS] >= (battery_at_slots[0] + EPSILON)) {
2     replaced = 1; plan_old = assignments[x]; j = 0;
3     while ((plan_old != plans[j]) && (j < NUMBER_OF_PLANS_1)) j++;
4     if (j >= NUMBER_OF_PLANS_1 - 1) return -1;
5     j++; k = plans[j];
6     while (replaced) {
7         replaced = 0; s = x + 1;
8         while (s <= SLOTS) {
9             plan_old = assignments[s - 1];
10            consumption_old = (cost_of_plan[plan_old] / (60 * 60)) * ↵
  ↵ Vld * TIME_SLOTS / (VOLTAGE * BATTERIES);
11            consumption_new = (cost_of_plan[k] / (60 * 60)) * Vld * ↵
  ↵ TIME_SLOTS / (VOLTAGE * BATTERIES);
12            if ((k < NUMBER_OF_PLANS_1 - 1) && (k >= 0) && ↵
  ↵ ((battery_at_slots[SLOTS] - battery_at_slots[0]) >= ↵
  ↵ (consumption_new - consumption_old))) {
13                assignment_new[s - 1] = k;
14                recompute_battery_level(k, assignment_new, ↵
  ↵ battery_at_slots, cost_of_plan, s, ↵
  ↵ ddMMyyhhmmss[3], QoS, month, 1);
15                replaced = 1;
16                get_time_from_seconds(TIME_SLOTS, ddMMyyhhmmss);
17            }
18            s++;
19        }
20        j++;
21        if (j <= NUMBER_OF_PLANS_1 - 1) k = plans[j];
22        else replaced = 0;

```

```

23     }
24 }
25 else {
26     plan_old = assignments[x]; plan = plan_old; j = 0;
27     while ((plan_old != plans[j]) && (j < NUMBER_OF_PLANS_1)) j++;
28     j--;
29     if (j == 0) return -1;
30     k = plans[j]; replaced = 1;
31     while (replaced) {
32         s = x + 1;
33         replaced = 0;
34         while (s <= SLOTS) {
35             if((k >= 1) && (battery_at_slots[SLOTS] < ↵
36                 ↵ battery_at_slots[0])) {
37                 assignment_new[s - 1] = k;
38                 recompute_battery_level(k, assignment_new, ↵
39                     ↵ battery_at_slots, cost_of_plan, s, ↵
40                     ↵ ddMMyyhhmmss[3], QoS, month, 1);
41                 get_time_from_seconds(TIME_SLOTS, ddMMyyhhmmss);
42                 replaced = 1;
43             }
44             s++;
45         }
46         plan_old = k; j--;
47         if (j < 0) replaced = 0;
48         else k = plans[j];
49     }
50 }

```

Evaluación del algoritmo neutral en energía

Una vez que ya hemos implementado los algoritmos previamente descritos, se procede a evaluar el comportamiento de los mismos. Para ello, en la implementación que hemos realizado antes, en los métodos *compute_cost_assignment* y *recompute_cost_assignment*, cuyo código puede consultarse en el Anexo A, y antes del algoritmo de reoptimización, se ha implementado la funcionalidad de imprimir en varios archivos CSV con datos relacionados con los algoritmos. Por cada mes se generaran dos archivos CSV:

1. Un archivo en el que se incluyen el número de slot, la fecha, el plan, la calidad de servicio, el consumo y la producción energética de cada uno de los slots. Además, cada vez que se vuelve a recalcular el nivel de la batería, también se escribe en este archivo. Por tanto, dado que algoritmo itera hasta encontrar la mejor solución, las 300 últimas líneas (una para cada slot, dado que el número de slots es 300) de cada uno de estos archivos corresponden con la asignación que se ha hecho para ese día del mes. Estos archivos tienen una estructura como la del Listado 4.5.
2. Otro archivo en el que se muestra la producción energética y la variación de la misma por cada slot. En el campo de variación, si la producción energética es mayor que el consumo, se expresa que hay un superhábit de energía. Por el contrario, hay un déficit. Estos archivos se realizan a la hora de hacer la reoptimización. Estos archivos tienen la estructura que se refleja en el Listado 4.6.

Listado 4.5: Fragmento del archivo CSV correspondiente al mes de enero para mostrar el formato de los mismos

```

1 slot,date,plan,QoS,battery,consumption,ep_slot
2 1,1-1-2020:0:0:0,5,70.000000,1999.973600,0.026400,0.000000
3 2,1-1-2020:0:4:48,5,70.000000,1999.947200,0.026400,0.000000
4 3,1-1-2020:0:9:36,5,70.000000,1999.920800,0.026400,0.000000

```

```

5 4,1-1-2020:0:14:24,5,70.000000,1999.894400,0.026400,0.000000
6 5,1-1-2020:0:19:12,5,70.000000,1999.868000,0.026400,0.000000
7 6,1-1-2020:0:24:0,5,70.000000,1999.841600,0.026400,0.000000
8 7,1-1-2020:0:28:48,5,70.000000,1999.815200,0.026400,0.000000
9 8,1-1-2020:0:33:36,5,70.000000,1999.788800,0.026400,0.000000
10 9,1-1-2020:0:38:24,5,70.000000,1999.762400,0.026400,0.000000
11 10,1-1-2020:0:43:12,5,70.000000,1999.736000,0.026400,0.000000
12 11,1-1-2020:0:48:0,5,70.000000,1999.709600,0.026400,0.000000
13 12,1-1-2020:0:52:48,5,70.000000,1999.683200,0.026400,0.000000
14 13,1-1-2020:0:57:36,5,70.000000,1999.656800,0.026400,0.000000
15 14,1-1-2020:1:2:24,5,70.000000,1999.630400,0.026400,0.000000
16 15,1-1-2020:1:7:12,5,70.000000,1999.604000,0.026400,0.000000

```

Listado 4.6: Fragmento del archivo CSV de la reoptimización del mes de enero para mostrar el formato de los mismos

```

1 slot,ep_slot,variation
2 81,0.000000,deficit
3 82,0.000000,deficit
4 83,0.000000,deficit
5 84,0.000000,deficit
6 85,0.000000,deficit
7 86,0.000000,deficit
8 87,0.000000,deficit
9 88,0.000000,deficit
10 89,0.000000,deficit
11 90,0.000000,deficit
12 91,0.000000,deficit
13 92,0.000000,deficit
14 93,0.000000,deficit
15 94,0.000000,deficit
16 95,0.000000,deficit
17 96,0.000000,deficit
18 97,0.000000,deficit
19 98,0.000000,deficit
20 99,0.000000,deficit
21 100,0.000000,deficit
22 101,0.433134,superhabit
23 102,0.433134,superhabit
24 103,0.433134,superhabit

```

Aparte, también se ha creado un script en Python para representar la curva de carga/descarga de la batería y el slot seleccionado por cada uno de los slots de tiempo. El script es el que se muestra en el Listado 4.7. Para realizar este script se ha hecho uso de los componentes `csv` para poder manejar archivos CSV en Python y `matplotlib` [21] para realizar la representación de figuras. Al ejecutar dicho script, se le pasa con el argumento `-f` el archivo CSV generado por la asignación inicial que deseemos, y obtendremos una imagen con dos gráficas: una arriba, que representa la curva de carga y descarga de la batería por cada slot, y la segunda, abajo, en la que se representan los planes de ejecución que tiene cada slot. Un ejemplo de estas figuras es la que aparece en la Figura 4.7

Listado 4.7: Script para la creación de figuras sobre los archivos CSV

```

1 #!/usr/bin/python3
2
3 import csv
4 import argparse
5 import matplotlib.pyplot as plt
6 from datetime import datetime
7
8 parser = argparse.ArgumentParser(description='Choose file to plot')
9 parser.add_argument("-f", "--file", required=True, help='File to plot')

```

```

10
11 args = parser.parse_args()
12
13 path = args.file
14 file = open(path, newline='')
15 reader = csv.reader(file)
16
17 header = next(reader)
18
19 data = []
20 for row in reader:
21     slot = int(row[0])
22     date = datetime.strptime(row[1], '%d-%m-%Y:%H:%M:%S')
23     plan = int(row[2])
24     QoS = float(row[3])
25     battery = float(row[4])
26     consumption = float(row[5])
27     ep_slot = float(row[6])
28
29     data.append([slot, date, plan, QoS, battery, consumption, ↵
30                 ↵ ep_slot])
31
32 battery_values = []
33 plan_values = []
34
35 for d in data:
36     plan_values.append(d[2])
37     battery_values.append(d[4])
38
39 battery_values = battery_values[-300:]
40 plan_values = plan_values[-300:]
41
42 plt.figure()
43 plt.subplot(211)
44 plt.plot(list(range(1, 301)), battery_values, 'b--')
45 plt.ylabel('Battery_level')
46 plt.title('Battery_level_per_slot')
47
48 plt.subplot(212)
49 plt.plot(list(range(1, 301)), plan_values, 'r--')
50 plt.xlabel('Slots')
51 plt.ylabel('Plans')
52 plt.title('Plans_per_slot')
53 plt.show()

```

Para la evaluación del algoritmo, se han realizado la siguientes pruebas:

1. Una prueba para observar como se ha realizado la mejora del plan de ejecución (*Upgrade*).
2. Una prueba para observar como se ha realizado el empeoramiento del plan de ejecución (*Downgrade*).

La evaluación del algoritmo ha usado los planes de ejecución mostrados en la Tabla 4.11. Aunque se han realizado pruebas para todos los meses del año, solo se van a mostrar algunos de ellos en los siguientes ejemplos. En los siguientes párrafos se usa la siguiente notación: el plan P_1 se ha renombrado como el plan de ejecución 0, el P_2 es el plan 1 y así sucesivamente: el plan auxiliar P_{aux} se ha renombrado como plan 5.

Ejemplo de Upgrade A continuación se va a mostrar un ejemplo de *Upgrade*. Como ya se explicó anteriormente, cuando se hace Upgrade, se cambia un plan de ejecución por otro plan que tenga mayor consumo, ya que el consumo energético y la producción solar así lo permiten, maximizando

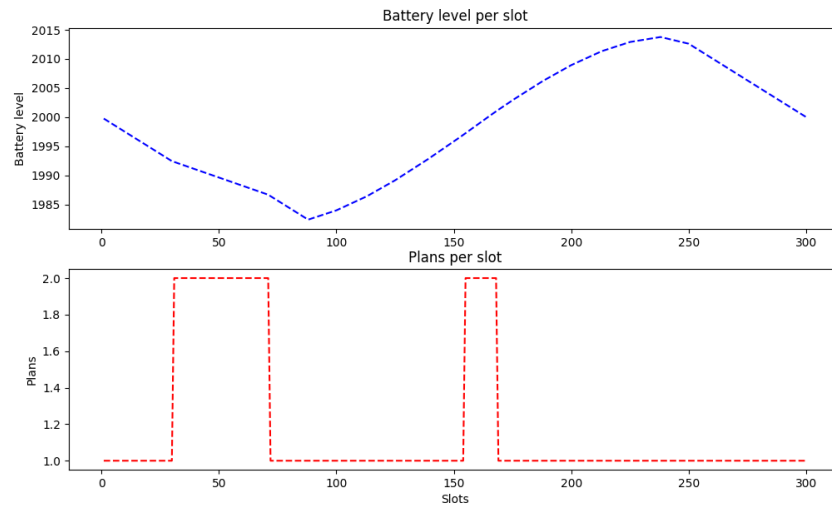


Figura 4.7: Ejemplo de gráfica de carga/descarga de batería y planes de ejecución por slot del mes de agosto

el nivel de calidad de servicio. Para este ejemplo, se va a tomar como ejemplo el mes de abril. En el Listado 4.8 se puede apreciar como se realizado el *Upgrade*. Hasta el slot 71 se puede apreciar como cada slot estaba ejecutando el plan de ejecución 5, que tiene una calidad de servicio del 70 %, pero a partir del slot 72 se mejora el plan de ejecución, pasando a ejecutar el plan 2, con el que se mejora la calidad de servicio del mismo, la cual pasa a ser del 75 %. En la Figura 4.8 se pueden apreciar las gráficas de carga y descarga y asignación de planes de ejecución del mes de abril, y se puede observar como se cambia del plan 5 al 2 al principio de la misma. Aparte, en la Tabla 4.12 se puede ver la evolución de la calidad de servicio entre la asignación inicial y la final. Como se puede comprobar, la calidad de servicio ha aumentado.

Listado 4.8: Fragmento del CSV del mes de abril para mostrar como se ha realizado el *Upgrade*

```

1 68,1-4-2020:5:21:36,5,70.000000,1992.141600,0.026400,0.000000
2 69,1-4-2020:5:26:24,5,70.000000,1992.115200,0.026400,0.000000
3 70,1-4-2020:5:31:12,5,70.000000,1992.088800,0.026400,0.000000
4 71,1-4-2020:5:36:0,5,70.000000,1992.062400,0.026400,0.000000
5 72,1-4-2020:5:40:48,2,75.000000,1991.921600,0.140800,0.000000
6 73,1-4-2020:5:45:36,2,75.000000,1991.780800,0.140800,0.000000
7 74,1-4-2020:5:50:24,2,75.000000,1991.640000,0.140800,0.000000
8 75,1-4-2020:5:55:12,2,75.000000,1991.499200,0.140800,0.000000
9 76,1-4-2020:6:0:0,2,75.000000,1991.358400,0.140800,0.000000
10 77,1-4-2020:6:4:48,2,75.000000,1991.217600,0.140800,0.000000
11 78,1-4-2020:6:9:36,2,75.000000,1991.076800,0.140800,0.000000

```

| Asignación | QoS |
|--------------------------------------|---------|
| Asignación inicial | 70 % |
| Asignación final tras <i>Upgrade</i> | 74,57 % |

Tabla 4.12: Comparación de calidad de servicio entre la asignación inicial y final del mes de abril

Ejemplo de *Downgrade* A continuación se va a mostrar un ejemplo de *Downgrade*. Tal y como se explicó anteriormente, cuando se hace *Downgrade*, se pasa de un plan de ejecución a otro que tenga menor calidad de servicio que el anterior, ya que el consumo del primer plan es mayor que el que vamos a cambiar, y hace que el sistema no se pueda mantener neutral en energía. Para este ejemplo vamos a tomar como referencia el mes de marzo. En el Listado 4.9 se muestra un fragmento del CSV

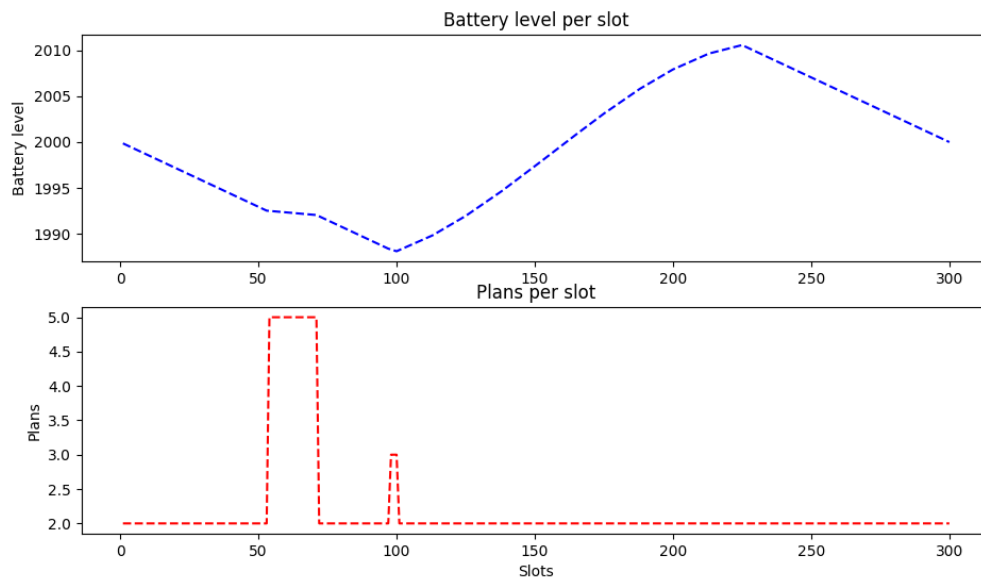


Figura 4.8: Gráficas de carga/descarga y de asignación de planes de ejecución del mes de abril

del mes de marzo. Como se puede apreciar, hasta el slot 263, se ha ejecutado el plan de ejecución 2, el cual tiene una calidad de servicio del 75 %, pero a partir del slot 264 se ejecuta el plan 5, el cual tiene una calidad de servicio del 70 %. Como dicho plan nuevo tiene menor calidad de servicio, también es menor su consumo, y por lo tanto se puede apreciar que la descarga de la batería es más suave que cuando se tiene el plan de ejecución 2. En la Figura 4.9 se pueden apreciar las gráficas de carga y descarga de la batería y la asignación de planes de ejecución del mes de marzo. Como se puede apreciar en las gráficas, al final se asigna el plan 5, y se puede apreciar como la descarga de la batería es más suave. Como se puede apreciar en la Tabla 4.13, la calidad de servicio en la asignación final ha aumentado con respecto a la inicial.

Listado 4.9: Fragmento del CSV del mes de marzo para mostrar como se ha realizado el *Downgrade*

```

1 256,1-3-2020:20:24:0,2,75.000000,2001.965228,0.140800,0.000000
2 257,1-3-2020:20:28:48,2,75.000000,2001.824428,0.140800,0.000000
3 258,1-3-2020:20:33:36,2,75.000000,2001.683628,0.140800,0.000000
4 259,1-3-2020:20:38:24,2,75.000000,2001.542828,0.140800,0.000000
5 260,1-3-2020:20:43:12,2,75.000000,2001.402028,0.140800,0.000000
6 261,1-3-2020:20:48:0,2,75.000000,2001.261228,0.140800,0.000000
7 262,1-3-2020:20:52:48,2,75.000000,2001.120428,0.140800,0.000000
8 263,1-3-2020:20:57:36,2,75.000000,2000.979628,0.140800,0.000000
9 264,1-3-2020:21:2:24,5,70.000000,2000.953228,0.026400,0.000000
10 265,1-3-2020:21:7:12,5,70.000000,2000.926828,0.026400,0.000000
11 266,1-3-2020:21:12:0,5,70.000000,2000.900428,0.026400,0.000000
12 267,1-3-2020:21:16:48,5,70.000000,2000.874028,0.026400,0.000000
13 268,1-3-2020:21:21:36,5,70.000000,2000.847628,0.026400,0.000000
14 269,1-3-2020:21:26:24,5,70.000000,2000.821228,0.026400,0.000000
15 270,1-3-2020:21:31:12,5,70.000000,2000.794828,0.026400,0.000000

```

| Asignación | QoS |
|--|---------|
| Asignación inicial | 70 % |
| Asignación final tras <i>Downgrade</i> | 71,91 % |

Tabla 4.13: Comparación de calidad de servicio entre la asignación inicial y final del mes de marzo

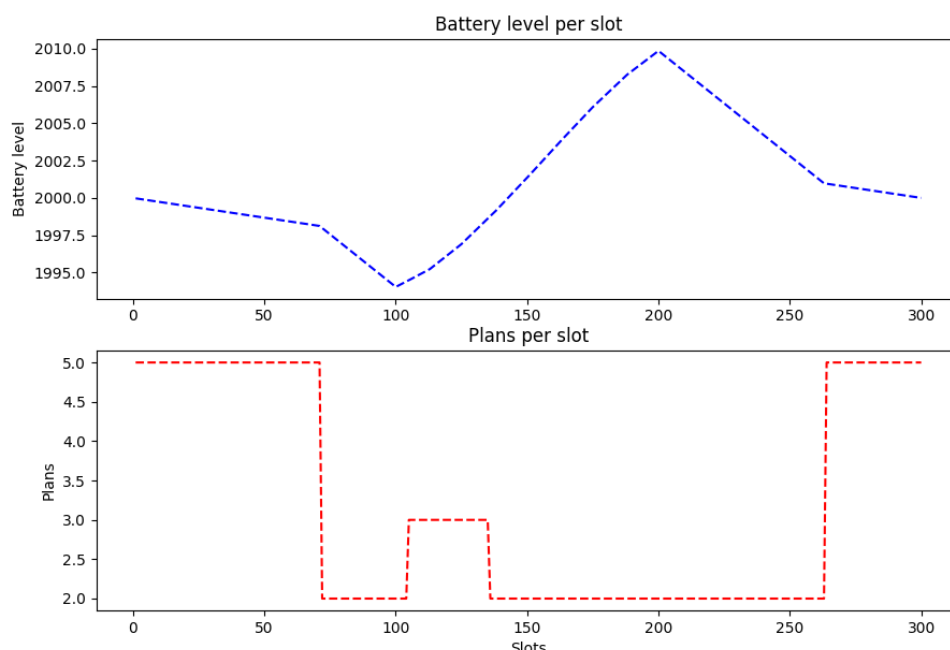


Figura 4.9: Gráficas de carga/descarga y de asignación de planes de ejecución del mes de marzo

4.3.2. Iteración 3. Implementación de la aplicación de agricultura inteligente

En esta iteración vamos a ver como se ha llevado a cabo la implementación de la aplicación de agricultura inteligente de los dispositivos Adafruit Feather M0. La funcionalidad de esta aplicación es la de recoger información de los sensores que tiene la plataforma PLATINO, y enviar dicha información a una pasarela, que como ya se comentó en el Capítulo 4.2.1, estará compuesta de un dron que hará dos vuelos al día, uno por la mañana y otro por la tarde. Para que estos dispositivos se mantengan neutrales en energía, se utilizará el código que se ha implementado en el Capítulo 4.3.1, y se mostrará más adelante como incluir dicho código en nuestro proyecto.

Lo primero de todo que tenemos que hacer es crearnos un proyecto en Visual Studio Code con la extensión de PlatformIO. Para ello, en la propia extensión le damos al botón de *New Project* y nos aparecerá una ventana como la de la Figura 4.11 en la que tenemos que especificar el nombre del proyecto, el tipo de placa que vamos a utilizar, en nuestro caso la Adafruit Feather M0, y el tipo de framework que vamos a usar con la placa, que en nuestro caso será el de Arduino.

Una vez que ya tengamos creado el proyecto, tenemos que modificar el archivo *platformio.ini*, en el que tenemos que añadirle las dependencias de librerías de terceros que vamos a tener que utilizar. Las librerías que tenemos que añadir son las siguientes:

- **Nanopb0.4.0:** Para poder hacer uso de los *Protocol Buffer* de Google.
- **868:** Para poder leer y escribir en una tarjeta MicroSD.
- **RTCLib:** Para la utilización del Real Time Clock (RTC) de la placa Adafruit.
- **RadioHead:** Librería que se utiliza para la comunicación con LoRa a través de la radio RFM95 del Adafruit Feather M0
- **Adafruit GFX Library:** Librería utilizada para el uso de gráficos.
- **Adafruit SSD1306:** Librería usada para el uso de pantallas OLED monocromo.
- **Adafruit INA219:** Librería usada para utilizar los sensores INA219 de la plataforma PLATINO.
- **Arduino Thread:** Librería usada para el uso de hilos en la programación de las Adafruit Feather M0.

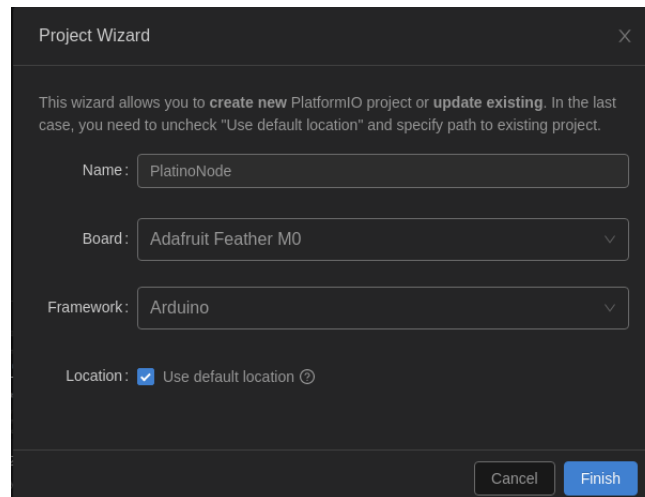


Figura 4.10: Crear un proyecto en PlatformIO

- **DHT sensor library:** Librería usada para utilizarla con el sensor DHT de la plataforma PLATINO.

Aparte de las librerías comentadas anteriormente, también se usará la librería *platform*, la cuál sirve para interactuar con la plataforma PLATINO. Esta librería esta compuesta por varias funciones que encapsulan la funcionalidad básica sobre la plataforma PLATINO, por ejemplo, obtener los valores que producen los sensores DHT e INA219, poder manejar la escritura de valores en la tarjeta MicroSD, controlar el RTC, realizar las comunicaciones a través de la radio LoRa que incorporan los micro sistemas de la plataforma, y manejar la escritura o impresión de datos en una pantalla OLED. Se puede ver la referencia a dicha librería y su funcionamiento en [4]. También, nos crearemos una librería con el código de asignación de planes de ejecución que se ha visto en la anterior iteración. Para utilizar ambas librerías, en la carpeta *lib* de nuestro proyecto, tenemos que incluir estas dos librerías.

Algoritmo de la aplicación de agricultura inteligente

Para esta aplicación, vamos a tener dos hilos de ejecución, uno será el *trabajador*, que se va a encargar de ejecutar el plan de ejecución asignado en cada uno de los slots, y el otro será el *planificador*, que se encargará de calcular los planes de ejecución óptimos para cada uno de los slots de tiempo y de asignar a cada slot de tiempo la aplicación que le corresponde, es decir, es el que crea los hilos trabajadores. Aparte, se hará una asignación de planes de ejecución antes de que la aplicación ejecute su código principal.

Una vez que ya hemos modificado el archivo *platformio.ini*, podemos empezar a programar nuestra aplicación. En la carpeta de *src* se encuentra el código principal, que es el que vamos a modificar. Este archivo tiene la misma estructura que los archivos *ino* de Arduino, es decir, un método *setup* en el que se definen una serie de parámetros para la aplicación, y un método *loop* que se va a ir repitiendo con cada iteración del mismo. En el método *setup* vamos a poner nuestro código, es cuál tiene que incluir el *baud rate*, la inicialización del display, del RTC y su ajuste, de la radio LoRa, y de la MicroSD, así como el intervalo de tiempo que tiene que durar un hilo de ejecución de las aplicaciones, que en nuestro caso, tiene que ser el mismo que el tiempo que dura cada uno de los slots de tiempo, es decir, 288 segundos. Aparte, se hará la asignación de planes de energía inicial, de la cual hablaremos más adelante. En el método *loop*, para cada uno de los slots de tiempo, como se comentaba en el Capítulo 4.2.1, se le asigna la aplicación que le toca según la asignación que se ha hecho previamente. Además, si se llegan a las 23 horas, se hace otra asignación de planes, para que cuando sean las 0 horas del día siguiente, se tenga un plan nuevo.

El hilo *trabajador* tiene la siguiente funcionalidad:

1. La aplicación, usando la librería *platform*, leerá los datos de todos los sensores de la plataforma PLATINO, es decir, lee de el sensor SHT-10 y los sensores de corriente INA219 obteniendo los datos de temperatura, humedad, velocidad del viento, corrientes y voltajes del panel solar, del sistema Smart Farming y de la batería. Los métodos a los que hay que llamar para obtener dichos datos son los que se muestran en el Listado 4.10, en el que se puede ver como se obtienen la corriente de carga (línea 1), la corriente de la batería (línea 2), la corriente del panel (línea 3), la velocidad del viento (línea 4), la temperatura (línea 5), la humedad (línea 6) y el voltaje de la batería (línea 7).

Listado 4.10: Métodos para obtener los valores de los sensores

```

1  value = platform.getLoadCurrent();
2  value = platform.getBatteryCurrent();
3  value = platform.getPanelCurrent();
4  value = platform.getSpeedOfWind();
5  value = platform.getTemperature();
6  value = platform.getHumidity();
7  value = platform.getBatteryVoltage();

```

2. Guardar dicha información en la tarjeta MicroSD que lleva incorporada la plataforma PLATINO, también usando la librería de *platform*. Para escribir en la tarjeta MicroSD se usa el código de el Listado 4.11, en el que en la línea 1 se puede apreciar que se inicializa el modo de escritura en la tarjeta, en la línea 2 se escriben los datos en la tarjeta, y en la línea 3 se cierra el escritor de la tarjeta.

Listado 4.11: Código usado para guardar información en la tarjeta MicroSD

```

1  platform.open(filename, mode);
2  platform.writeline(line);
3  platform.close();

```

3. Enviar la información que se ha obtenido desde los sensores a través de la radio LoRa. Para enviar esta información, se ha usado el formato de Protocol Buffers de Google [20]. Para ello, es necesario definir los formatos y mensajes que van a intercambiarse el dispositivo LoRa y la pasarela LoRa, es decir, el protocolo de aplicación.

Cuando se va a enviar los datos a través de LoRa al dron, se espera a que el dron envíe a la plataforma PLATINO un mensaje de sincronización (su identificador), cuyo formato se presenta en el Listado 4.12. Hasta que no lo reciba, la plataforma no enviará al dron su identificador, usando el mensaje del Listado 4.13. Una vez que se ha enviado el identificador al dron, la plataforma envía los datos que se han recogido antes de los sensores usando el mensaje que se muestra en el Listado 4.14. Si algún mensaje no se recibe de manera correcta por el dron, éste enviará a la plataforma el identificador del nodo que procede a la retransmisión y el número del paquete que necesita, usando el mensaje que aparece en el Listado 4.15. Más, en detalle, el intercambio de mensajes entre el dron y la plataforma PLATINO es la siguiente:

- a) El gateway LoRa (dron) envía al dispositivo un mensaje simple con su identificador, cuyo formato es el que aparece en el Listado 4.12.

Listado 4.12: Formato del mensaje para recibir el identificador del dron

```

1  message MSG_PHASE1_DRONE_ID {
2      int32 id = 1;

```

```
3     }
```

- b) Tras recibir el identificador del dron, el nodo responde al dron con su identificador. El buffer de protocolo correspondiente a este mensaje se puede ver en el Listado 4.13.

Listado 4.13: Formato del mensaje para enviar el identificador del nodo al dron

```
1     message MSG_PHASE1_ENDDEVICE_ID {
2         int32 id = 1;
3         int32 numPackets = 2;
4     }
```

- c) Tras este mensaje, el dispositivo procede a enviar un mensaje para cada período de muestreo en el que se han leído todos los sensores, el cual tiene el formato del Listado 4.14. Tal y como se puede apreciar en dicho Listado, los datos que se van a enviar en el mensaje son el identificador de la plataforma, en número del paquete, la carga del nodo, la carga de la batería, la carga del panel solar, la velocidad del viento, la temperatura, la humedad, el voltaje de la batería y el tiempo de muestreo. Cuando se han ido obteniendo los valores de los sensores, estos datos se han ido guardando dentro del mensaje que se va a mandar.

Listado 4.14: Formato del mensaje para enviar los datos leídos de los sensores al dron

```
1     message MSG_PHASE2_ENDDEVICE_DATA {
2         int32 idDev = 1;
3         int32 idPacket = 2;
4         float nodeLoad = 3;
5         float batteryLoad = 4;
6         float panelLoad = 5;
7         float windSpeed = 6;
8         float temperature = 7;
9         float humidity = 8;
10        float batteryVolt = 9;
11        int32 samplingTime = 10;
12    }
```

- d) Si el dron detecta que le falta un paquete por recibir, se hace uso de un mensaje NACK (Non ACK) que aparece en el Listado 4.15

Listado 4.15: Formato del mensaje si falta algún mensaje

```
1     message MSG_PHASE2_DRONE_NACK {
2         int32 id = 1;
3         int32 idPacket = 2;
4     }
```

Para realizar el envío de un mensaje se ha usado el código del Listado 4.16, en el cual se puede apreciar como en la línea 4 se recibe el identificador del dron, si este es menos uno, se sale del método, como se puede ver en la línea 5. Después, se manda el identificador de la plataforma al dron en la línea 8, y luego se va mandando el mensaje completo al dron. Más adelante, se explicarán los métodos utilizados para el envío por LoRa, es decir, a los que se invocan en las líneas 3, 7 y 14.

Listado 4.16: Código para enviar mensajes al dron a través de LoRa

```
1     void appClass::sendToDrone() {
2         // Phase 1: the end-device receives a message from the drone
3         ret = receiveSYNC();
```

```

4         if (ret == -1) return;
5
6         // sends its identifier
7         ret = sendMyIDToDrone();
8
9         // Phase 2: data transmission
10        cont = 0;
11        while (cont < N) {
12            // Send the data to the drone
13            msg3.idPacket = cont;
14            ret = sendDataToDrone(&msg3);
15            cont = cont + 1;
16            if (cont == N)
17                continue;
18        }
19        delay(t);
20    }

```

Para poder recibir el identificador del dron se usa el método *receiveSYNC()*, cuyo código es el que se muestra en el Listado 4.17, en el que en la línea 2 se recibe un mensaje por medio de LoRa, después en las líneas 3 y 4 se decodifica dicho mensaje usando el formato de mensajes de Protobuf del Listado 4.12. Por último, se comprueba si el identificador que se ha recibido es el correcto.

Listado 4.17: Código del método *receiveSYNC* para recibir el identificador del dron

```

1    int appsClass::receiveSYNC() {
2        platform.receiveLoRa(buffer, &len);
3        istream = pb_istream_from_buffer(buffer, len);
4        pb_decode(&istream, ←
                    ← mtpPlatino_MSG_PHASE1_DRONE_ID_fields, ←
                    ← &msg1);
5
6        Serial.print("Node_");
7        Serial.print(IDDev);
8        Serial.print(":");
9        Serial.print(msg1.id);
10
11        if (msg1.id != IDDrone) {
12            Serial.println("_DRONE_not_recognized");
13            return -1;
14        }
15        else {
16            Serial.println("_DRONE_recognized");
17        }
18        return 0;
19    }

```

Tras recibir el identificador correcto del dron, se pasa a enviar el identificador de la plataforma al dron, usando el código del Listado 4.18, en el que se forma un mensaje de Protobuf, con la estructura del Listado 4.13 en las líneas 2 y 3, luego en la línea 5 se codifica dicho mensaje, y por último se envía dicho mensaje a través de LoRa.

Listado 4.18: Código del método *sendMyIDToDrone* para enviar el identificador del nodo al dron

```

1    int appsClass::sendMyIDToDrone() {
2        msg2.id = IDDev;
3        msg2.numPackets = N;
4        ostream = pb_ostream_from_buffer(buffer, ←
                    ← sizeof(buffer));

```

```

5     pb_encode(&ostream,
6         mtpPlatino_MSG_PHASE1_ENDDEVICE_ID_fields,
7         &msg2);
8     len = ostream.bytes_written;
9     platform.sendLoRaPB(buffer, len);
10    return 0;
11 }

```

Por último, para enviar todos los datos que se han obtenido de los sensores, y después de haber enviado el identificador de la plataforma al dron, se utiliza el código del Listado 4.19, en el que en la línea 6 se codifica el mensaje de Protobuf con la estructura del Listado 4.14, y se envía dicho mensaje a través de LoRa en la línea 10. Tras enviar el mensaje, se espera a recibir un NACK desde el dron en la línea 13. Si no se ha recibido ningún mensaje, se sale del método, pero si se recibe alguno, se descodifica en la línea 18 con el formato de mensaje que aparece en el Listado 4.15. Tras la decodificación, se comprueba si el identificador del paquete del mensaje que se ha recibido es igual al del paquete que se ha enviado.

Listado 4.19: Código del método *sendDataToDrone* para enviar los datos al dron

```

1  int appsClass::sendDataToDrone(
2      mtpPlatino_MSG_PHASE2_ENDDEVICE_DATA *msg3) {
3      bool retransmission = 0;
4      while (!retransmission) {
5          ostream = pb_ostream_from_buffer(buffer,
6              sizeof(buffer));
7          pb_encode(&ostream,
8              mtpPlatino_MSG_PHASE2_ENDDEVICE_DATA_fields,
9              &msg3);
10         len = ostream.bytes_written;
11         platform.sendLoRaPB(buffer, len);
12
13         /* And wait for a while to see if you
14            receive a NACK from the drone */
15         platform.receiveLoRa(buffer, &len);
16         if (len == 0)
17             return 0;
18
19         istream = pb_istream_from_buffer(buffer, len);
20         pb_decode(&istream,
21             mtpPlatino_MSG_PHASE2_DRONE_NACK_fields,
22             &msg4);
23
24         // We need to check that msg4.idPacket == msg3.idPacket
25         if (msg4.idPacket == msg3->idPacket)
26             retransmission = 1;
27         else
28             retransmission = 0;
29     }
30     return 0;
31 }

```

Planes de ejecución La aplicación que se acaba de describir, da lugar a cinco posibles planes de ejecución diferentes, donde cada uno de ellos hace ninguna, varias o todas las funcionalidades que se han descrito antes, formando así combinaciones de tareas que dan lugar a un plan de ejecución. Los planes de ejecución que deben ser ejecutados en cada slot de tiempo son calculados por el proceso *planificador*, mientras que el hilo *trabajador* será el encargado de poner el plan seleccionado a ejecutar en cada uno de los slots de tiempo. A continuación, se describen cinco planes de ejecución que pueden

obtenerse de la aplicación de agricultura inteligente descrita en la sección anterior:

- P_1 : Este plan de ejecución no hace nada, solo es un bucle *while* que se ejecuta continuamente, en el que no hay ninguna línea de código para ejecutar. Es el plan, por tanto, que tendrá mínimo coste y mínima calidad.
- P_2 : Este plan de ejecución solo recoge información de los sensores INA219 y DHT haciendo uso de la librería de *platform*.
- P_3 : Este plan de ejecución recoge información de los sensores haciendo uso de la librería *platform* y además guarda esa información en la tarjeta MicroSD.
- P_4 : Este plan de ejecución recoge información de los sensores haciendo uso de la librería *platform* y envía esta información por medio de la radio LoRa a la pasarela, es decir, al dron.
- P_5 : Este plan de ejecución incluye todas las funcionalidades anteriormente descritas, es decir, recoge la información de los sensores haciendo uso de la librería *platform*, guarda dicha información en la tarjeta MicroSD y la envía los datos a través de la radio LoRa al dron.

Cada uno de estos planes de ejecución se ha implementado como un método de una clase, a la que se ha llamado *apps*. El nombre del método que recibe cada uno de los planes de ejecución es el siguiente: el plan de ejecución P_1 es el método *app1*, el plan P_2 , el método *app2*, y así sucesivamente. Dicha clase se guarda en la carpeta *lib* de nuestro proyecto, para así poder ser usada en el código principal de la aplicación, el que se encuentra en la carpeta *src*.

Las asignaciones de planes de ejecución se hace desde el método *loop*, como se ha comentado antes, cuyo código se puede apreciar en el Listado 4.20. Como se puede apreciar, se hace un bucle *for* para iterar a través de todos los slots. Si la hora es igual a las 23:00 horas, el hilo planificador busca una nueva asignación de planes de ejecución para el próximo día, y a las 00:00 horas del día siguiente se aplica el nuevo plan. Dependiendo del plan que se tenga en cada uno de los slots de tiempo, el hilo planificador crea un hilo trabajador para cada slot de tiempo con el plan de ejecución que le corresponde, tal y como podemos ver en el *switch* de la línea 21. En la línea 43 se comprueba si el plan que se ha asignado se puede ejecutar, y si es así, se le ordena que se ejecute. En la línea 46 se itera a través de un bucle *while* hasta que la aplicación pueda ser ejecutada.

Listado 4.20: Código del método *loop* de la aplicación de agricultura inteligente

```

1 void loop() {
2     int *aux_assignment;
3
4     for (int i = 0; i < SLOTS; i++) {
5         if (platform.getHour() == 23 && !assigned) {
6             int *time = platform.getTime();
7
8             time[0] += 1;
9             time[3] = 0;
10            time[4] = 0;
11            time[5] = 0;
12
13            aux_assignment = assignment.assign_plan(time);
14            assigned = true;
15        }
16        else if (platform.getHour() == 0)
17            plan_assignment = aux_assignment;
18
19        int plan = plan_assignment[i];
20
21        switch (plan) {
22            case 1:

```

```

23     app_thread.onRun(apps.app1);
24     break;
25
26     case 2:
27         app_thread.onRun(apps.app2);
28         break;
29
30     case 3:
31         app_thread.onRun(apps.app3);
32         break;
33
34     case 4:
35         app_thread.onRun(apps.app4);
36         break;
37
38     case 5:
39         app_thread.onRun(apps.app5);
40         break;
41 }
42
43 if (app_thread.shouldRun())
44     app_thread.run();
45
46 while (!app_thread.shouldRun())
47     continue;
48 }
49 }

```

Por desgracia, debido a las circunstancias que hemos vivido los últimos meses de estado de alarma y pandemia mundial del COVID-19, esta implementación no ha podido ser evaluada sobre la plataforma PLATINO debido al régimen de no presencialidad ya que no se ha podido tener acceso a la propia plataforma. Sin embargo, las evaluaciones positivas sobre PC que hemos obtenido del algoritmo neutral en energía (ver iteración 2) y la simplicidad de este algoritmo nos hacen pensar que la aplicación de agricultura inteligente, combinada con el algoritmo neutral en energía, funcionaría sin demasiados cambios sobre la plataforma.

4.3.3. Iteración 4. Despliegue del entorno en FIWARE Lab

Los datos que producen las plataformas PLATINO y que son enviados a la pasarela a través de LoRa, deben de ser guardados en algún tipo de plataforma dedicada al IoT. Por desgracia, como ya se comentó al final del Capítulo 4.3.2, dichas plataformas PLATINO no han podido estar disponibles debido al estado de alarma que se ha vivido. Por lo tanto, como ya se explicará en el Capítulo 4.3.4, se podrá probar el envío de dichos datos a dicha plataforma de otra forma que ya se comentará. Para este proyecto, tal y como se explicaba en el Capítulo 4.2.1, se ha elegido que la plataforma de IoT que se va a usar sea FIWARE Lab para dicho almacenamiento. En esta iteración, por lo tanto, se mostrará como obtener una cuenta en dicha plataforma, como configurar nuestra cuenta en temas de seguridad, como crear una instancia en la plataforma y los componentes que debemos de instalar en dicha plataforma.

Crear cuenta en FIWARE Lab

Para poder tener acceso a la plataforma FIWARE, es necesario crearse una cuenta en FIWARE Lab, la cual se puede hacer desde[6]. Al hacer click sobre este enlace, aparecerá un formulario de Google, tal y como aparece en la Figura 4.11 que nos permite crear una cuenta de FIWARE Lab. Para crear la cuenta, se necesita proporcionar un nombre, un e-mail y la región en la que estarán ubicadas las instancias de FIWARE Lab. Una vez enviada la petición, hay que esperar a que nos autoricen la cuenta y nos den una contraseña para poder entrar a FIWARE Lab. Con esta cuenta se puede tener acceso

a FIWARE Lab por un período de 2 semanas. Para la realización de este TFG, este tiempo es muy limitado y no daría tiempo a poder desarrollarlo por completo, por lo que existe la posibilidad de crear otro tipo de cuenta, llamada *Community Account*, con la cual se tendrá acceso a la nube durante un periodo de 9 meses. Esta cuenta se puede pedir desde la pantalla de *login* de FIWARE Lab, tal y como se ve en la Figura 4.12a. Al entrar en el formulario, aparecerá algo parecido a la Figura 4.12b. En este formulario, se nos solicita la siguiente información:

- **User full name:** Nuestro nombre de usuario.
- **User account email:** El correo electrónico que usaremos para entrar a FIWARE Lab. Si ya tenemos una cuenta, tenemos que ingresar el correo electrónico de esa cuenta. En nuestro caso, el correo que ya habíamos dado antes para crear la cuenta normal de FIWARE Lab.
- **Are you already registered in FIWARE Lab? YES-NO:** Confirmamos si tenemos o no una cuenta en FIWARE Lab.
- **Company:** Compañía a la que pertenecemos. En nuestro caso, la UCLM.
- **Department:** El departamento al que se pertenece.
- **Number of developers:** El número de desarrolladores involucrados en el proyecto.
- **Startup/Project name:** El nombre del proyecto.
- **Accelerator Programme name:** Nombre del acelerador de programa. En nuestro caso, seleccionamos *fiware-based application*, ya que nuestro proyecto no está relacionado con ningún programa de aceleración de FIWARE.
- **Startup/Project description:** Descripción del proyecto incluyendo el objetivo, áreas de aplicación, descripción de la aplicación que se desea desarrollar, etc.
- **Preferred FIWARE Lab Node:** Nodo preferido para FIWARE Lab. En nuestro caso, seleccionamos *Lannion*, ya que es la que más cerca está de nuestra ubicación y la que da menos problemas.
- **Proof of concept URL:** URL donde se pueda acceder a la prueba de concepto del proyecto, en caso de tenerla.
- **Number of VMs:** Número de máquinas virtuales que vamos a necesitar para nuestro proyecto. En nuestro caso, solo nos hace falta una.
- **Total # vCPUs:** Número de vCPUs que se van a necesitar. En nuestro caso, 4.
- **Total RAM:** Número de gigabytes de RAM que se va a necesitar en el proyecto. En nuestro caso, 8 GB.
- **Total harddisk:** Número de gigabytes de almacenamiento que se requiere para el proyecto. En nuestro caso, 40 GB.
- **# public IPs:** Número de IPs públicas que vas a necesitar. En nuestro caso, solo nos hace falta una.
- **Object Storage:** Número de gigabytes de *Object Storage* (arquitectura de almacenamiento que gestiona datos como objetos). En nuestro caso, no nos hace falta nada de eso, así que no lo pedimos.
- **Additional comments/requirements/essential questions:** Comentarios adicionales, peticiones o preguntas que tengamos.
- **Name:** Nuestro nombre completo (opcional).
- **Email:** Correo electrónico (opcional).

Tras mandar la petición, tenemos que esperar, igual que cuando pedimos la cuenta normal, a que nos manden por correo electrónico nuestra contraseña y nos den acceso a FIWARE Lab. Una vez que tenemos la contraseña, intentamos acceder a FIWARE Lab a través de la URL [18]. Si podemos entrar, lo primero de todo es comprobar si nos han dado todos los recursos que pedimos (vCPUs, RAM, VMs, etc). En nuestro caso, no nos proporcionaron todos los recursos solicitados, sino 2 vCPUs y 4GB de RAM que, en cualquier caso, son suficientes para el desarrollo de este TFG. Esto es debido a que los recursos que tienen son compartidos por muchos usuarios y, por tanto, limitados. Esto se puede ver en la vista general de la Figura 4.13.

Create new FIWARE Lab user account

Enter your data and preferences in order to create the FIWARE Lab user account. It is important that you accept the Terms & Conditions before we create the account.

***Obligatorio**

Name *
User name to be stored in the created account.
Daniel

Email *
Email address to be used to access to the FIWARE Lab Cloud.
Daniel.Ballesteros1@alu.uclm.es

FIWARE Lab region *
Selected FIWARE Lab region in which you want to work.
Lannion5

I accept the FIWARE Lab Terms & Conditions *
http://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_LAB_Terms_and_Conditions

☒ I have read and understand the Terms & Conditions

Enviar

Figura 4.11: Formulario para crear la cuenta de FIWARE Lab

Configuración de acceso y seguridad

Al acceder a FIWARE Lab por primera vez, tendremos que realizar unos ajustes en la sección de *Acceso y seguridad*. Dentro de ésta, en la sección de *Grupos de seguridad*, se ha de crear un grupo nuevo. Al hacer click, aparecerá un recuadro como el de la Figura 4.14, en el cuál tenemos que poner el nombre del grupo y, opcionalmente, una descripción. Una vez tenemos el grupo de seguridad, debemos de configurarlo para nuestras necesidades. Para ello, le damos al botón de *Administrar reglas* del grupo que hemos creado. Después, debemos agregar dos reglas: una para permitir conectarnos a nuestra VM de FIWARE Lab por SSH, y la segunda para abrir el puerto TCP 7896 (se describirá seguidamente). Para añadir una regla, le damos al botón de *Agregar regla*. Al darle al botón, aparecerá un recuadro como el de la Figura 4.15. En el apartado de *Regla*, añadimos *SSH* y le damos a *Añadir*. Después, volvemos a añadir otra regla. En este caso, lo único que hay que tocar es el apartado de puerto, en el que tenemos que poner el puerto 7896, dejando los demás valores por defecto. Con estas dos reglas, tendríamos nuestro grupo de seguridad configurado.

El siguiente paso es crear un par de claves. Para ello, volvemos a *Acceso y seguridad* y nos dirigimos a *Par de claves*. Una vez allí, le damos a *Crear un par de claves* y aparecerá un recuadro como el de la Figura 4.16, en el que tenemos que poner el nombre del par de claves que vamos a crear. Una vez creada, se nos descargará un archivo con extensión `.pem` con el nombre del par de claves. Este archivo nos servirá para acceder a las VMs que tengamos a través de SSH. Más adelante se mostrará cómo se usa.

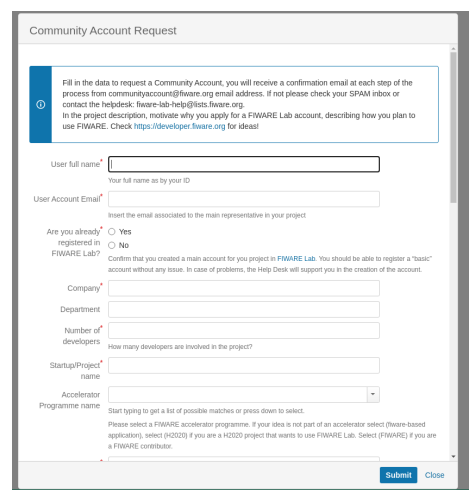
Creación de la instancia

Una vez se ha configurado las opciones de acceso y seguridad, se procede a crear la instancia con la que trabajaremos a partir de ahora. Para crear una instancia, tenemos que ir a la pestaña de *Instancias* en la izquierda. Después, pulsaremos en el botón de *Lanzar Instancia*. Al hacerlo, nos saldrá una ventana como la de la Figura 4.17. Primero, definimos el nombre de la instancia, el sabor de la instancia (distintas configuraciones de recursos que podemos tener en la instancia), el número de instancias que queremos generar, y el origen de arranque de la instancia. En nuestro caso, hemos optado por



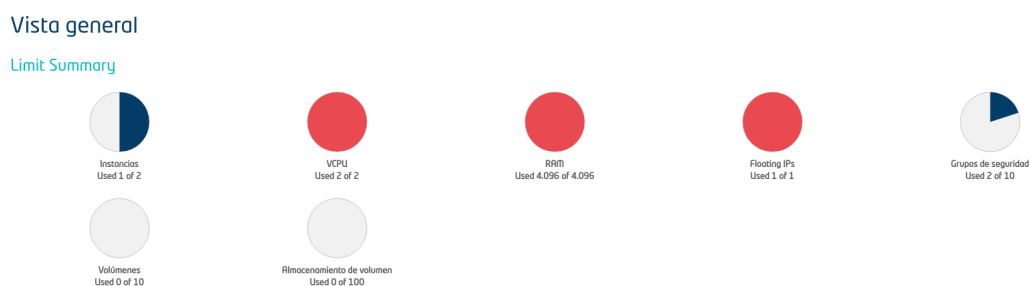
The image shows the FIWARE Lab Cloud Portal login page. It features the FIWARE Lab logo at the top. Below the logo, there is a 'Log In' link. Underneath, there are two input fields: 'Usuario' (Username) and 'Contraseña' (Password). At the bottom of the login section, there are two buttons: 'Create Account' and 'Connect'. A red box highlights a link labeled 'Request Community Account' located below the 'Connect' button.

(a) Pedir cuenta Community de FIWARE Lab



The image shows the 'Community Account Request' form. It contains several fields for user information: 'User full name', 'User Account Email', 'Company', 'Department', 'Number of developers', 'Startup/Project name', and 'Accelerator Programme name'. There are also checkboxes for 'Are you already registered in FIWARE Lab?' and 'Confirm that you created a main account for your project in FIWARE Lab'. A 'Submit' button is at the bottom right.

(b) Crear cuenta Community en FIWARE Lab

Figura 4.12: Pedir y crear cuenta Community de FIWARE Lab**Figura 4.13:** Vista general de FIWARE Lab

crear una instancia basada en Ubuntu 18.04. Para ello, elegimos *Arrancar desde una imagen* y luego seleccionamos la imagen *base_ubuntu_18.04*.

Después, volvemos de nuevo a la pestaña de *Acceso y seguridad*, como se puede ver en la Figura 4.18. Primero, tenemos que elegir el par de claves que hemos creado antes, y seleccionar el grupo de seguridad que configuramos con anterioridad.

Tras esto, ya podemos lanzar nuestra instancia mediante el botón *Lanzar* que está abajo a la izquierda, como se puede ver en las Figuras 4.17 y 4.18. Cuando ya esté creada, lo siguiente que haremos es asociar la IP pública que nos han dado con nuestra instancia. Para ello, en la parte derecha de la instancia hacemos click sobre *Asociar IP flotante*, como se puede ver en la Figura 4.19a. Al hacer click, nos aparece una ventana como la de la Figura 4.19b, en la que seleccionamos la IP flotante que vamos a asociar con nuestra instancia.

Una vez que ya hemos asociado la IP flotante, ya podemos conectar a nuestra instancia. Para ello, en una terminal, tenemos que situarnos donde hayamos guardado el par de claves que nos descargamos antes y ejecutar el comando:

Listado 4.21: Comando para conectarse a la instancia de FIWARE Lab

```
1 $ ssh -i <nombre de la clave> ubuntu@<ip flotante>
```

donde *nombre de la clave* es el nombre del par de claves que hemos creado con anterioridad y hace referencia al archivo que nos hemos descargado con la misma, y *IP flotante* es la IP flotante que nos han asignado a nuestro entorno.

Crear grupo de seguridad

Nombre *

grupo_daniel

Descripción

Descripción:

Los grupos de Seguridad son conjuntos de reglas de filtros de IP que son aplicados a la configuración de red para la máquina virtual. Después de que el grupo de seguridad es creado, puedes agregar reglas al grupo de seguridad.

Cancelar Crear grupo de seguridad

Figura 4.14: Creación del grupo de seguridad

Agregar regla

Regla *

Regla TCP a medida

Dirección

Entrante

Puerto abierto *

Puerto

Puerto

Remoto *

CIDR

CIDR

0.0.0.0/0

Descripción:

Los reglas definen el tráfico permitido a las instancias asociadas al grupo de seguridad. Una regla de un grupo de seguridad contiene tres partes principales:

Regla: Puede especificar una plantilla de reglas deseada o usar reglas TCP, UDP e ICMP personalizadas.

Puerto abierto/Rango de puertos Para las reglas de TCP y UDP puede optar por abrir un solo puerto o un rango de ellos. La opción "Rango de puertos" le proporcionará el espacio para especificar tanto el puerto de comienzo como de final del rango. Para las reglas de ICMP por el contrario debe especificar el tipo y código ICMP en los espacios proporcionados.

Remoto: Debe especificar el origen del tráfico a permitir a través de esta regla. Lo puede hacer bien con el formato de un bloque de direcciones IP (CIDR) o especificando un grupo de origen (Grupo de Seguridad). Al seleccionar un grupo de seguridad como origen, se permitirá que cualquier instancia de ese grupo de seguridad pueda acceder a cualquier otra instancia a través de esta regla.

Cancelar Añadir

Figura 4.15: Agregación de reglas en el grupo de seguridad

Crear par de claves

Nombre de par de claves *

clave_fisware_lab

Descripción:

Los pares de claves son credenciales ssh que se inyectan en las imágenes al lanzarlos. Al crear un par de claves nuevo, se almacena la clave pública y se descarga la clave privada (un fichero .pem)

Proteja y use la clave como haría con cualquier clave privada ssh.

Cancelar Crear par de claves

Figura 4.16: Crear un par de claves

Despliegue de componentes Docker en la instancia

Una vez hemos conectado con nuestra instancia a través de SSH, lo primero que tenemos que hacer es actualizar los paquetes de la misma. Esto se puede hacer fácilmente con los comandos *apt update* y *apt upgrade*. Tras actualizar los paquetes y reiniciar la instancia, tenemos que instalar Docker y Docker Compose.

Docker es un conjunto de productos para implementar la filosofía de plataforma como servicio (*Platform as a Service, PaaS*) que utilizan la virtualización a nivel del sistema operativo para entregar software en paquetes llamados contenedores. Los contenedores están aislados unos de otros y agrupan su propio software, bibliotecas y archivos de configuración y pueden comunicarse entre sí a través de canales bien definidos. Todos los contenedores son ejecutados por el kernel de Linux. Esto hace

Lanzar instancia

Detalles

Acceso y seguridad

Redes

Pos-creación

Opciones avanzadas

Zona de Disponibilidad

nova

Nombre de la instancia

Ubuntu

Sabor

m1.medium

Recuento de instancias

1

Origen de arranque de la instancia

Arrancar desde una imagen

Nombre de la imagen

base_ubuntu_18.04 (566,1 MB)

Especifique los detalles de la instancia a lanzar.

La siguiente tabla muestra los recursos utilizados por este proyecto en relación a sus cuotas.

Detalle del sabor

| | |
|----------------|-----------|
| Nombre | m1.medium |
| VCPU | 2 |
| Disco raíz | 40 GB |
| Disco efímero | 0 GB |
| Total de Disco | 40 GB |
| RAM | 4,096 MB |

Límites del proyecto

Número de instancias

0 de 2 Usados

Número de VCPU

0 de 2 Usados

Total RAM

0 de 4,096 MB Usados

Cancelar

Lanzar

Figura 4.17: Detalles para lanzar una instancia

Lanzar instancia

Detalles

Acceso y seguridad

Redes

Pos-creación

Opciones avanzadas

Par de claves

clave_fiware_lab

Grupos de seguridad

☒ grupo_daniel

☐ default

☐ prueba

Controle el acceso a sus instancias a través de pares de claves, grupos de seguridad y otros mecanismos.

Cancelar

Lanzar

Figura 4.18: Acceso y seguridad para lanzar la instancia

Terminar instancias

More Actions

| su creación | Actions |
|-------------|--|
| | <div>Crear instantánea</div> <div>Asociar IP flotante</div> <div>Conectar interfaz</div> <div>Desconectar interfaz</div> |

(a) Asociar IP flotante a la instancia

Gestionar asociaciones de IP flotantes

Dirección IP

195.220.224.174

Puerto a asociar

Ubuntu: 192.168.0.73

Seleccione la dirección IP que quiere asociar con una determinada instancia o puerto.

Cancelar

Asociar

(b) Seleccionar IP flotante para nuestra instancia

Figura 4.19: Asociar y seleccionar IP flotante para nuestra instancia

que se utilicen menos recursos que una máquina virtual convencional. Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker de contenedores múltiples. Docker Compose

utiliza un archivo YAML para configurar los servicios de su aplicación. Con un simple comando, se pueden ejecutar a la vez todos los servicios definidos en el archivo de configuración.

Para instalar Docker y Docker Compose, se han utilizado las guías [23] y [31]. Una vez instalados, tenemos que tener claro cuales son los componentes que debe de tener nuestra estructura de contenedores. Los componentes que vamos a necesitar son los siguientes:

- **Orion Context Broker (OCB)**[7]: Este componente permite administrar la información de contexto, consultarla y actualizarla. Para ello, utiliza NGSI-v2, el cual es un protocolo para manejar información de contexto, del cual se hablará de su modelo de datos más adelante. El OCB permite la publicación de información de contexto por entidades, como por ejemplo sensores, de manera que la información de contexto publicada se encuentra disponible para otras entidades. El OCB siempre está escuchando en el puerto 1026. Aparte, se comunica con la base de datos mongoDB para almacenar el estado actual de las entidades.
- **IoT Agent**: Un IoT Agent es un componente que permite a un grupo de dispositivos enviar sus datos y que sean gestionados a través de un Context Broker usando sus protocolos nativos (NGSI-v2). Los dispositivos se pueden conectar al IoT Agent a través de HTTP o MQTT con un payload Ultralight 2.0, o por LoRaWAN. En nuestro caso, vamos a utilizar HTTP Ultralight 2.0 por su sencillez. El IoT Agent usa el puerto 4041 para poder configurarlo, y el puerto 7896, el cual hemos abierto en el grupo de seguridad, para que cualquier dispositivo o sensor pueda mandar peticiones HTTP Ultralight 2.0 a través de él.
- **QuantumLeap**: QuantumLeap es un servicio REST para guardar, consultar y recuperar datos NGSI-v2, los cuales los convierte en forma de tabla y los guarda en una base de datos de series temporales.
- **mongoDB**: mongoDB es un sistema de base de datos no relacional, la cual es usada por el OCB y el IoT Agent para guardar el estado de las entidades, dispositivos o grupos de servicio.
- **CrateDB**: CrateDB es otro sistema de bases de datos, pero esta vez SQL, la cuál guarda las consultas de QuantumLeap.

La arquitectura de los servicios que vamos a crear con los contenedores Docker se muestra en la Figura 4.20.

NGSI-v2 tiene un modelo de datos específico, el cual se va explicar a continuación. Dicho modelo de datos se puede ver representado en la Figura 4.21, en el que se pueden observar 3 elementos de contexto: *Entity*, *Attributes* y *Metadata*. A continuación, se va a pasar a explicar cada uno de estos elementos:

- **Context Entity**: Las *Context Entities*, o entidades, son la parte principal dentro del modelo de información de NGSI. Una entidad representa una cosa, por ejemplo, cualquier objeto físico o lógico, como una casa, un coche, o un sensor. Cada una de estas entidades tienen un identificador. Además, también se puede determinar de que tipo es esa entidad. Normalmente, los tipos de las entidades son tipos semánticos, es decir, ayudan a describir el tipo de cosa que se esté representando en la entidad.
- **Context Attributes**: Los atributos de contexto son propiedades de las entidades. En el modelo de datos de NGSI, los atributos tienen un *nombre de atributo*, un *tipo de atributo*, un *valor de atributo* y *metadata*. El nombre de atributo describe que tipo de propiedad representa el valor del atributo. El tipo del atributo representa el tipo de valor NGSI del valor del atributo. Finalmente, el valor del atributo contiene el dato del valor y una *metadata* que es opcional, la cual describe propiedades del valor del atributo.

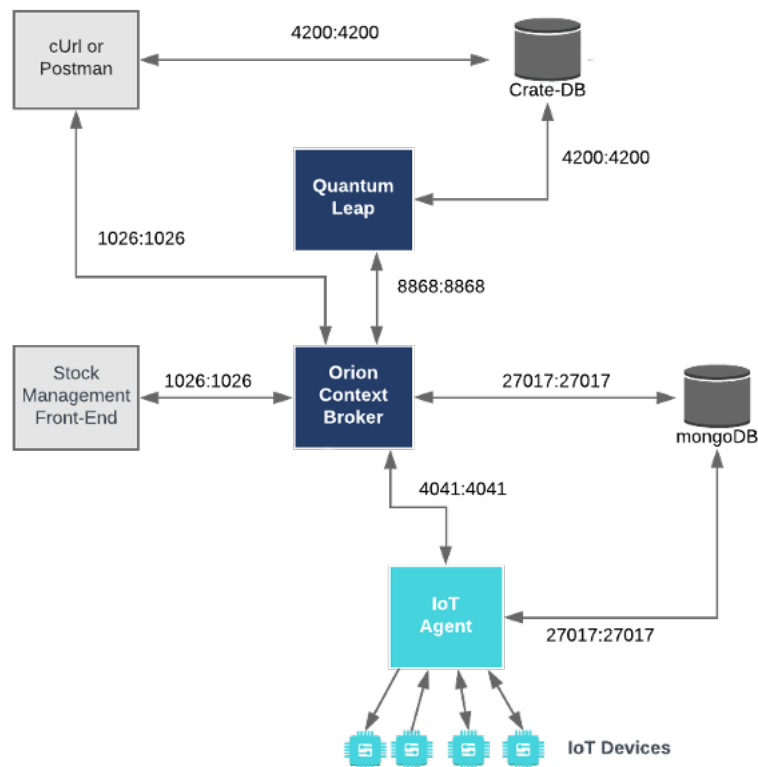


Figura 4.20: Arquitectura de los contenedores Docker

- **Context Metadata:** Los metadatos de contexto son usados en NGSI en distintos sitios, uno de ellos siendo una parte opcional de los atributos de contexto, como ya hemos visto. Al igual que los atributos, estos tienen un nombre, el cual describe el rol de la metadata que se está definiendo, un tipo y un valor.

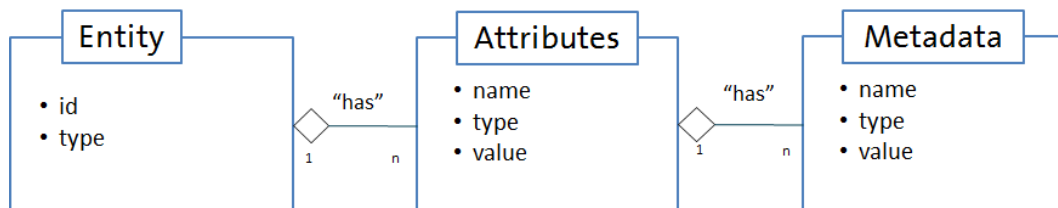


Figura 4.21: Modelo de datos de NGSI-v2. Figura obtenida de [5]

Para utilizar todos estos componentes, nos hemos basado en uno de los repositorios de ejemplos que tiene FIWARE. Más concretamente, el que tiene todos los componentes que vamos a necesitar es que aparece en la referencia [9]. Dicho repositorio tiene varios archivos:

- **Archivo *docker-compose.yml*:** En este archivo se definen todos los componentes que vamos a usar para nuestro proyecto, pero no se van a utilizar todos los servicios que se definen en dicho archivo. Los servicios que no vamos a utilizar son *tutorial* y *grafana*.
- **Archivo *import-data*:** Este archivo se utiliza para añadir entidades de contexto.
- **Archivo *provision-devices*:** En este archivo se describen los distintos grupos de servicio para todos los dispositivos IoT, así como los propios sensores que se definen.
- **Archivo *services*:** Este archivo sirve para automatizar el despliegue de todos los componentes que se describen en el archivo *docker-compose.yml*, así como la ejecución de los archivos *import-data* y *provision-devices*.

- **Archivo .env:** En este archivo se definen una serie de variables de entorno usadas por los contenedores Docker.

Para poder utilizar dicho repositorio, debemos de clonarlo en nuestra instancia, que con el simple comando `git clone https://github.com/FIWARE/tutorials.Time-Series-Data.git` lo podemos tener clonado en nuestra instancia. Una vez que tengamos clonado el repositorio, tenemos que modificar los archivos que se han comentado antes. El primero de todos es el `docker-compose.yml`, al que tenemos que borrar los servicios que no vamos a necesitar, los cuales son Grafana y *tutorial*. Aparte, también tenemos que borrar todo el contenido que tienen los archivos *import-data* y *provision-devices*, los cuales incluyen bastantes entidades, grupos de servicio y sensores que no nos hacen falta, ya que tenemos que incluir los componentes que necesitamos para nuestro proyecto.

El primero de los archivos que vamos a modificar después del `docker-compose.yml` será el archivo de *import-data*, cuyo contenido es el que aparece en el Listado 4.22. En dicho Listado, añadimos una entidad, la cual va a ser del tipo *Home*, con el atributo *name* que tiene como nombre *name*, tipo *Text* y valor *Daniel House*, de acuerdo al formato NGSI descrito previamente. Todo esto, se tiene que hacer a través de una petición *HTTP* al puerto del Orion Context Broker, el cual es el 1026.

Listado 4.22: Código del script *import-data*

```
1 #!/bin/bash
2 curl -s -o /dev/null -X POST 'http://orion:1026/v2/op/update' \
3 -H 'Content-Type: application/json' \
4 -g -d '{"actionType": "append", "entities": [
5     {"id": "urn:ngsi-ld:Home:001", "type": "Home",
6      "name": {"type": "Text", "value": "Daniel House"}}
7   ]
8 },'
```

Después, se modifica el archivo *provision-devices*. En el, se va a crear un grupo de servicio para el sensor, y aparte crearemos un sensor. En el Listado 4.23, se crea un grupo de servicio para todos los dispositivos IoT Ultralight a través de una petición *HTTP* al puerto del IoT Agent, el 4041. Para ello, se le pasa un JSON con una *apikey*, se especifica la URL del Context Broker, el tipo de entidad, que en nuestro caso es *Thing*, y el recurso.

Listado 4.23: Código del script *provision-devices* para crear un grupo de servicio

```
1 #!/bin/bash
2 curl -s -o /dev/null -X POST \
3 'http://iot-agent:4041/iot/services' \
4 -H 'Content-Type: application/json' \
5 -H 'fiware-service: openiot' \
6 -H 'fiware-servicepath: /' \
7 -d '{"services": [
8     {"apikey": "1068318794", "cbroker": "http://orion:1026",
9      "entity_type": "Thing", "resource": "/iot/d"}
10  ]
11 },'
```

Después, en el Listado 4.24 se hace otra petición *HTTP* para aprovisionar un sensor, el cuál es el que recibirá los datos de nuestra pasarela. Para ello, también se le pasa otro JSON en el que se especifica el id del dispositivo (*sensor001*), el nombre y tipo de la entidad (*Sensor:001* y *Sensor*, respectivamente), los atributos, que en nuestro caso es la variable *temperature*, y los atributos estáticos, en los que se hace referencia a entidades. En nuestro caso, hacemos referencia a la entidad ya creada en el Listado 4.22.

Listado 4.24: Código del script *provision-devices* para aprovisionar un sensor

```
1 #!/bin/bash
```

```

2 curl -s -o /dev/null -X POST \
3   'http://iot-agent:4041/iot/devices' \
4   -H 'Content-Type: application/json' \
5   -H 'fiware-service: openiot' \
6   -H 'fiware-servicepath: /' \
7   -d '{
8     "devices": [
9       {
10        "device_id": "sensor001",
11        "entity_name": "Sensor:001",
12        "entity_type": "Sensor",
13        "timezone": "Europe/Madrid",
14        "attributes": [
15          {"object_id": "temp", "name": "temperature", "type": ↵
16            ↵ "Float"}
17        ],
18        "static_attributes": [
19          {"name": "refHome", "type": "Relationship",
20            "value": "urn:ngsi-ld:Home:001"}
21        ]
22      }
23    ],
24  },'

```

También hay que modificar el archivo `.env`, eliminando las variables correspondientes con el servicio de *tutorial*, el cual se ha borrado del `docker-compose.yml`

Una vez modificado los scripts anteriores, es hora de crear los contenedores. Para ello, hacemos uso del script `services`, ejecutando los comandos `./services create` para descargarnos las imágenes Docker que se van a utilizar y `./services start` para crear los contenedores. Por último, lo que tenemos que hacer es crear un script, al que se le ha llamado *subscription*, cuyo contenido será el del Listado 4.25. En este script, lo que se pretende hacer es, a través de una petición HTTP, subscribir el componente de QuantumLeap al componente del Orion Context Broker, para así obtener la información de contexto de este último para que el componente del QuantumLeap transforme dicha información en formato de tabla para guardarla en la base de datos CrateDB.

Listado 4.25: Código para el script *subscription*

```

1  #!/bin/bash
2
3  curl -iX POST \
4    'http://localhost:8668/v2/subscribe?orionUrl=http://orion:1026/v2&
5    quantumleapUrl=http://quantumleap:8668/v2' \
6    -H 'Content-Type: application/json' \
7    -H 'fiware-service: openiot' \
8    -H 'fiware-servicepath: /' \
9    -d '{
10     "description": "Notify QuantumLeap of count changes of any ↵
11       ↵ Motion Sensor",
12     "subject": {
13       "entities": [{"idPattern": "Sensor.*"}],
14       "condition": {"attrs": ["temperature"]}
15     },
16     "notification": {
17       "http": {"url": "http://orion:1026/v2/notify"},
18       "attrs": ["temeperature"],
19       "metadata": ["dateCreated", "dateModified"]
20     }, "throttling": 1
21   },'

```

4.3.4. Iteración 5. Integración del sistema final y evaluación

El propósito de esta iteración es la de integrar todos los componentes de la arquitectura, que se han desarrollado en las iteraciones anteriores y la de evaluar el sistema globalmente. Sin embargo, debido a la situación de no presencialidad que hemos vivido en los últimos meses debido al estado de alarma y la situación de pandemia mundial, esta iteración ha tenido que ser modificada y reorientada, al no poder tener acceso a los dispositivos PLATINO que existen en el laboratorio del grupo ARCO y que son los dispositivos originalmente pensados para la recolección de datos. Por este motivo, no se ha podido probar el código implementado en el Capítulo 4.3.2 y, por tanto, el sistema no ha podido evaluarse de manera global.

No obstante, para poder hacer una evaluación de todas las funcionalidades del sistema, a falta de los dispositivos PLATINO, se decidió adquirir un kit LoPy4 de desarrollo LoRa [43] que permitiera probar el aspecto de la comunicación LoRa entre un dispositivo y un gateway y, además, la comunicación entre el gateway y la plataforma FIWARE Lab. En la Figura 4.22 se puede observar una imagen del kit. Dicho kit se compone de los siguientes elementos:

1. Dos placas LoPy4[42], una de ellas tendrá el rol de nodo que produce datos, simulando el envío de un valor de temperatura; y la otra, hará el rol de pasarela (gateway), que recoge los datos enviados por el nodo a través de LoRa y reenvía dichos datos a la plataforma FIWARE Lab a través de la red TCP/IP.
2. Dos Expansion Board 3.0, utilizadas para poder programar las placas LoPy4, ya que éstas no disponen de puerto USB. Aparte, incluye un slot para tarjetas MicroSD y la posibilidad para poder alimentarse mediante una batería.
3. Dos antenas LoRa, con la frecuencia de 868 MHz necesaria en Europa, y que usaremos para enviar y recibir datos entre el nodo y la pasarela, respectivamente.



Figura 4.22: Kit de desarrollo LoPy4

El objetivo de la evaluación se reescribe por tanto como sigue: el dispositivo LoPy4 recoge datos y los retransmite a la pasarela, la cuál reenviará todos los datos a la plataforma FIWARE Lab. Se deberá comprobar que dicha información ha sido recibida y se deberá observar correctamente los datos en la base de datos CrateDB. Para realizar la implementación en las placas LoPy4, se tiene que utilizar el lenguaje de MicroPython [30], que es el único que soportar dichas placas.

Protocolo de comunicaciones

Para hacer la comunicación entre las LoPy4 con la plataforma FIWARE Lab, primero se debe de tener en cuenta como debe de ser la conexión entre ellos. En la Figura 4.23 se puede observar un esquema de la comunicación entre los distintos componentes. El nodo LoPy4, que produce un valor de temperatura, envía dicho valor a la pasarela a través de la tecnología de LoRa. La pasarela recibe también el valor de la temperatura a través de LoRa, y envía dicho valor a la plataforma de FIWARE Lab a través de la red de TCP/IP, primero enviándolo al router al cual está conectada la pasarela a través de Wi-Fi, y luego al puerto 7896, el cuál ya se comentó su funcionalidad, de la plataforma FIWARE Lab.

Los mensajes que enviamos y recibimos a través de los nodos LoPy4 es necesario que tengan un formato determinado para el envío y para la recepción de ACKs. El formato que se ha elegido para el envío de datos a la pasarela es muy simple, y esta compuesto de un byte para el identificador del nodo, y un *double* para el valor de la temperatura que se a mandar. El formato que tendrán los ACK será de 3 bytes: el primero para el identificador de la pasarela, el segundo para el tamaño del paquete, y el segundo para el código de estado. Estos formatos serán guardados en las variables *_LORA_PKG_FORMAT* y *_LORA_PKG_ACK_FORMAT* respectivamente en cada uno de los dispositivos como ya veremos más adelante.

El nodo tiene un período de envío de 20 segundos, es decir, cada 20 segundos manda un nuevo valor de temperatura. Si al enviar este valor no se recibe ningún ACK por parte de la pasarela, el mensaje se vuelve a mandar y se deja un segundo de espera. Esto se hacer hasta que el nodo reciba un ACK válido.

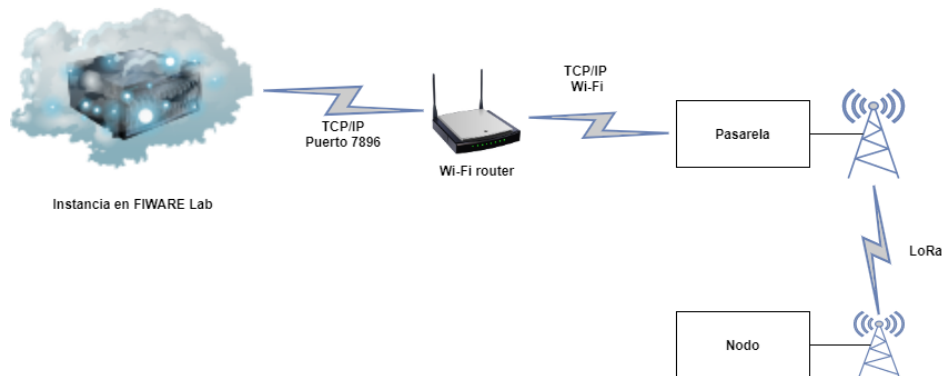


Figura 4.23: Esquema de comunicación entre las LoPy4 y FIWARE Lab

Implementación del dispositivo LoRa productor

La funcionalidad que tiene el productor es generar un número decimal aleatorio (que hace las veces de temperatura) y que se envía a la pasarela. Para generar dicho número aleatorio, se ha usado la función *random_number*, la cual está basada en una función definida por el usuario *livius* del foro de Pycom[32]. Dicha función se puede observar en el Listado 4.26.

Listado 4.26: Función para generar un numero aleatorio entre dos valores

```

1 def random_number(rfrom, rto):
2     def Random():
3         r = crypto.getrandbits(32)
4         return ((r[0] << 24)+(r[1] << 16)+(r[2] << 8)+r[3])/4294967295.0
5
6     return Random()*(rto-rfrom)+rfrom

```

Esta función devuelve un número aleatorio entre un rango de dos números, que en nuestro caso será de entre 15 y 30, los cuales corresponden con los dos argumentos de la función *rfrom* y *rto*. Para calcular dicho número, primero se llama a la función *Random* definida en la línea 2, la cual genera un número aleatorio entre 0 y 1, el cuál luego se multiplica por (*rto-rfrom*) y se le suma *rfrom*.

Una vez que ya tengamos el número aleatorio correspondiente con la temperatura, tenemos que enviarlo a la pasarela. Para ello, antes de nada, tenemos que activar el módulo LoRa del nodo y crear un socket LoRa, como se puede ver en el Listado 4.27.

Listado 4.27: Inicialización del módulo LoRa y creación del socket LoRa

```
1 lora = LoRa(mode=LoRa.LORA, region=LoRa.EU868)
2 lora_sock = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
3 lora_sock.setblocking(False)
```

En la primera línea, se inicia el módulo LoRa en modo *LORA* y se le especifica la región en la que estamos, en nuestro caso Europa, con la frecuencia de 868 MHz. Después se crea un socket dedicado para LoRa de tipo RAW, ya que se van utilizar los formatos de mensajes ya definidos con anterioridad. Por último, se configura el socket como *no bloqueante*. Ahora, para poder enviar el número aleatorio, se hace uso de la función *send_msg* presentada en el Listado 4.28.

Listado 4.28: Método para enviar el número aleatorio

```
1 def send_msg(temperature):
2     msg = struct.pack(_LORA_PKG_FORMAT, NODE_ID, temperature)
3
4     wait_ack = True
5     recv_timeout = 0
6
7     while wait_ack and recv_timeout < 20:
8         lora_sock.send(msg)
9         print('Message sent to gateway with temperature ↵
10             ↵ {}'.format(temperature))
11
12         recv_ack = lora_sock.recv(256)
13         recv_timeout += 1
14
15         if len(recv_ack) > 0:
16             device_id, pkg_len, status = struct.unpack(
17                 _LORA_PKG_ACK_FORMAT, recv_ack)
18
19             if NODE_ID == device_id:
20                 if status == 200:
21                     wait_ack = False
22                     print('ACK received {}'.format(status))
23                 else:
24                     wait_ack = False
25                     print('Message failed')
26
27             time.sleep(1)
28
29         if recv_timeout == 20:
30             lora_sock.close()
31             sys.exit(-1)
```

Lo primero que se hace en esta función es generar un paquete. Dicho paquete debe de tener el formato definido en la variable global *_LORA_PKG_FORMAT*, el cuál es un byte para identificar el dispositivo que lo envía y un *double* que es el número aleatorio, tal y como ya se explicó anteriormente. Para ello, se hace uso de la función *pack* del módulo *struct* de Python[45], el cuál también se encuentra en MicroPython. Después, dicho paquete se envía en la línea 8 y se espera a recibir un *ACK* de la

pasarela en la línea 11. El paquete *ACK* que se recibe tiene el formato *_LORA_PKG_ACK_FORMAT* y una longitud de 3 bytes, como se explicó anteriormente. Si se ha recibido un *ACK* y el estado del mismo es igual a 200, lo que significa que el estado es correcto, se termina la función. Si esto no es así, se vuelve a transmitir el paquete de datos a la pasarela hasta que se reciba el *ACK* (retransmisión). También, se ha definido que el número máximo de intentos de envío sea de 20. Pasados esos 20 intentos, sino se recibe el *ACK* correspondiente, se cerrará el socket LoRa que habíamos definido antes, y se dejará de enviar datos a la pasarela.

Implementación de la pasarela

La principal funcionalidad de la pasarela es recibir los datos de los dispositivos finales LoRa y reenviarlos a la plataforma de FIWARE Lab. En este caso, recibe los números aleatorios de temperatura generados por el único dispositivo productor que tenemos. Para ello, se tiene que inicializar el módulo LoRa y crear un socket LoRa, tal y como se ve en el Listado 4.27. Después, se tiene que conectar la pasarela a una red Wi-Fi, para que los datos que reciba del productor, los pueda enviar a la plataforma de FIWARE Lab utilizando esta tecnología inalámbrica. Para ello, se ha usado la función definida en el Listado 4.29.

Listado 4.29: Función para conectarse a una red Wi-Fi

```

1  def connect():
2      ssid = "" # el nombre de tu red Wi-Fi
3      password = "" # la contraseña de tu red Wi-Fi
4
5      wlan = WLAN(mode=WLAN.STA)
6
7      if wlan.isconnected() == True:
8          print("Already connected")
9          return
10
11     wlan.connect(ssid, auth=(WLAN.WPA2, password), timeout=5000)
12     while not wlan.isconnected():
13         machine.idle()
14
15     print("Connection successful")
16     ip = wlan.ifconfig()
17     return ip[0]
```

Como se puede ver en el anterior listado, en la línea 5 se inicializa el módulo Wi-Fi con la clase *WLAN* del módulo *network* con el modo *STA*, el cuál sirve para conectarse a una red Wi-Fi. Después, se comprueba si el módulo Wi-Fi está conectado a una red. Si es así, se sale de la función. Si no, tenemos que conectarnos a una red con el método *connect*, en la línea 12, pasando como argumentos el *SSID* de nuestra red y la tupla *auth*, compuesta de el método de protección de nuestra red y la contraseña. Adicionalmente, se puede definir un *timeout*. Tras intentar conectarse a la red, se comprueba si el nodo se ha conectado a la misma. Si es así, devuelve la dirección IP de la red en la que nos hemos conectado. Si no, se espera a que estemos conectados. Cuando la pasarela ya esté conectada a la red Wi-Fi, ya podemos recibir paquetes desde el productor. Para ello, se utiliza la función *get_data* en el Listado 4.30.

Listado 4.30: Método para recibir datos del productor

```

1  def getData():
2      data = lora_sock.recv(512)
3
4      if data:
5          device_id, msg = struct.unpack(_LORA_PKG_FORMAT, data)
6          print('Message received {}'.format(msg))
7          sendDataToFIWARE(msg)
```

```

8
9     ack_pkg = struct.pack(_LORA_PKG_ACK_FORMAT, device_id, ↵
    ↵ 1, 200)
10
11     lora_sock.send(ack_pkg)
12
13     time.sleep_ms(50)

```

Como se puede observar se empieza por recibir datos en la línea 2. Si se ha recibido algún dato, se intenta desempaquetar el paquete que nos haya llegado con el formato ya comentado antes (`_LORA_PKG_FORMAT`). Después, se reenvía el mensaje a la plataforma de FIWARE Lab en la línea 7, mediante el método `sendDataToFIWARE`, y se compone un paquete `ACK` en la línea 9 para enviarle al productor en la línea 11. Para enviar datos a FIWARE Lab, se usa el método `sendDataToFIWARE` de la línea 7, cuya implementación se muestra en el Listado 4.31.

Listado 4.31: Método para enviar datos a FIWARE Lab

```

1 def sendDataToFIWARE(msg):
2     r = requests.post(
3         'http://{/}/iot/d?k=1068318794&i=sensor001'.format(FIWARE_ADDR),
4         headers={'Content-Type': 'text/plain'},
5         data='temp|{}'.format(msg))
6     if r.status_code == 200:
7         print('Enviado con éxito a FIWARE')
8     else:
9         print('Error')

```

Para enviar el valor de la temperatura que se reciba, se tiene que realizar una petición `POST` por `HTTP`. Para ello, se ha usado la librería `urequests` [54] de MicroPython, basada en la popular librería `requests` de Python. Para hacer la petición `POST`, se ha utilizado el método `post`, al cuál se le pasa como parámetros la URL a la que vamos a hacer la petición, la cuál se compone de la IP pública de nuestra instancia en FIWARE Lab junto con el puerto que se ha abierto en dicha instancia, que en nuestro caso es el puerto 7896 de TCP, aparte del endpoint `/iot/d`, al que le tenemos que especificar la `apikey` y el identificador del dispositivo correspondiente; también hay que especificar el `header` necesario, que para nuestro caso será `Content-Type: text/plain`, y los datos que vamos a mandar. Si el código de respuesta recibido después de enviar los datos a través de `HTTP` es igual a 200, quiere decir que la petición ha sido correcta, y en caso contrario, habrá existido algún problema que será necesario depurar y corregir.

Integración de los componentes

Una vez que ya hemos implementado nuestro código en el productor y la pasarela, pasamos a integrar todos los componentes. Para ello, lo primero que se ha de hacer es conectar los dispositivos a través de la herramienta Pymakr de Visual Studio Code para poder cargar el código en los dispositivos y que éstos lo ejecuten. Como se pueden ver en las Figuras 4.24 y 4.25, tras tener los dispositivos conectados, tenemos que pulsar en el botón *Upload*, que tenemos en la parte inferior izquierda, para subir los archivos de cada proyecto a su respectivo LoPy4. Tras realizar esta operación con ambos dispositivos como se ha dicho anteriormente, vamos a comprobar que se recogen los datos correctos que envía el nodo y que dichos datos se pueden observar en la plataforma de FIWARE Lab.

El Listado 4.32 muestra el log del dispositivo productor, específicamente, puede observarse que se carga correctamente el código del nodo, las líneas 19, 21 y 23 muestran que se han enviado datos a la pasarela, y las líneas 20, 22 y 24 muestra que se reciben los ACKs correspondientes desde la pasarela.

Listado 4.32: Log del nodo

```

1 Uploading project (main folder)...

```




```

main.py x pymkr.conf
main.py > ...
8  import crypto
9
10
11  _LORA_PKG_FORMAT = "Bd"
12  _LORA_PKG_ACK_FORMAT = '!BBB'
13
14  # Open a Lora Socket, use tx_iq to avoid listening to our own messages
15  lora = LoRa(mode=LoRa.LORA, region=LoRa.EU868)
16  lora_sock = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
17  lora_sock.setblocking(False)
18
19  mac_addr = lora.mac()
20  NODE_ID = mac_addr[7]
21
22  def random_number(rfrom, rto):
23      def Random():
24          r = crypto.getrandbits(32)
25          return ((r[0] << 24)+(r[1] << 16)+(r[2] << 8)+r[3])/4294967295.0
26
27      return Random()*(rto-rfrom)+rfrom
28
29  # Method to send messages
30
31
32  def send_msg(temperature):
33      msg = struct.pack(_LORA_PKG_FORMAT, NODE_ID, temperature)
34
35      wait_ack = True
36      rcv_timeout = 0
37
38      while wait_ack and rcv_timeout < 20:
39          lora_sock.send(msg)
40          print('Message sented to gateway with temperature {}'.format(temperature))
41
42          rcv_ack = lora_sock.recv(256)
43          rcv_timeout += 1

```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL

```

OKets Jun  8 2016 00:22:57

rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff8020,len:8
load:0x3fff8028,len:2140
ho 0 tail 12 room 4
load:0x4009fa00,len:19760
entry 0x400a05bc

```

Run Upload Download All commands

Figura 4.24: Upload del código del productor en el nodo a través de Visual Studio Code

```

2 Safe booting device... (see settings for more info)
3 Uploading to /flash...
4 Reading file status
5 No files to upload
6 Upload done, resetting board...
7 OKets Jun  8 2016 00:22:57
8
9 rst:0x7 (TG0WDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
10 config: 0, SPIWP:0xee
11 clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,
12 wp_drv:0x00
13 mode:DIO, clock div:1
14 load:0x3fff8020,len:8
15 load:0x3fff8028,len:2140
16 ho 0 tail 12 room 4

```



```

main.py ./ x  main.py /home/.../node
main.py > getData
22     if r.status_code == 200:
23         print('Enviado con éxito a FIWARE')
24     else:
25         print('Error')
26
27 def getData():
28     data = lora_sock.recv(512)
29
30     if data:
31         device_id, msg = struct.unpack(_LORA_PKG_FORMAT, data)
32         print('Message received {}'.format(msg))
33         sendDataToFIWARE(msg)
34
35         ack_pkg = struct.pack(_LORA_PKG_ACK_FORMAT, device_id, 1, 200)
36
37         lora_sock.send(ack_pkg)
38
39         time.sleep_ms(50)
40
41
42 if __name__ == '__main__':
43     #print('eeee')
44     ip = wifiConnect.connect()
45     print('Connected to WiFi with IP: {}'.format(ip))
46     while True:
47         getData()
48
PROBLEMS 8  OUTPUT  DEBUG CONSOLE  TERMINAL
rst:0x7 (TG0WDT SYS RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config:ip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff8020,len:8
load:0x3fff8028,len:2140
ho 0 tail 12 room 4
load:0x4009fa00,len:19760
entry 0x400a05bc
Connection successful
Connected to WiFi with IP: 192.168.1.52
▶ Run ▲ Upload ▼ Download 🏠 ⚙ All commands

```

Figura 4.25: Upload del código de la pasarela a través de Visual Studio Code

```

17 load:0x4009fa00 ,len:19760
18 entry 0x400a05bc
19 Message sent to gateway with temperature 24.7
20 ACK received 200
21 Message sent to gateway with temperature 25.2
22 ACK received 200
23 Message sent to gateway with temperature 16.6
24 ACK received 200

```

De la misma manera, el Listado 4.33 muestra como se cargan los archivos en el dispositivo LoPy4 que hace de pasarela. En la línea 20 muestra la dirección IP que tiene la pasarela en la red en la que se ha conectado. En las líneas 21, 23 y 25 se muestra como ha recibido los mensajes procedentes del nodo. En las líneas 22, 24 y 26 confirman que se han enviado esos mensajes a FIWARE Lab con éxito.

Listado 4.33: Log de la pasarela

```

1 Uploading to /flash...
2 Reading file status
3 [1/3] Writing file main.py (1kb)
4 [2/3] Writing file urequests.py (3kb)
5 [3/3] Writing file wifiConnect.py (0kb)
6 Upload done, resetting board...
7 OKets Jun  8 2016 00:22:57
8
9 rst:0x7 (TGOWDT_SYS_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
10 configsip: 0, SPIWP:0xee
11 clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00
12 ,wp_drv:0x00
13 mode:DIO, clock div:1
14 load:0x3fff8020,len:8
15 load:0x3fff8028,len:2140
16 ho 0 tail 12 room 4
17 load:0x4009fa00,len:19760
18 entry 0x400a05bc
19 Connection successful
20 Connected to WiFi with IP: 192.168.1.52
21 Message received 24.7
22 Enviado con exito a FIWARE
23 Message received 25.2
24 Enviado con exito a FIWARE
25 Message received 16.6
26 Enviado con exito a FIWARE

```

Después de que la pasarela haya enviado los mensajes procedentes del nodo productor a la plataforma de FIWARE Lab, vamos a comprobar que dichos datos se han guardado correctamente en la plataforma. Para ello, estando conectados a nuestra instancia en FIWARE Lab, tenemos que ejecutar el comando del Listado 4.34 para obtener los datos que hemos enviado desde la pasarela. En el Listado 4.35 se puede ver que en CrateDB se han guardado los tres valores que se han enviado desde los dispositivos LoPy4.

Listado 4.34: Comando para obtener los tres últimos valores registrados

```

1 curl -iX POST \
2 'http://localhost:4200/_sql' \
3 -H 'Content-Type: application/json' \
4 -d '{"stmt":"SELECT * FROM mtopeniot.etsensor WHERE entity_id = ↵
    ↵ '\',\'Sensor:001\'\' ORDER BY time_index DESC LIMIT 3"}'

```

Listado 4.35: Últimos valores registrados en CrateDB

```

1 $ curl -iX POST \
2 > 'http://localhost:4200/_sql' \
3 > -H 'Content-Type: application/json' \
4 > -d '{"stmt": "SELECT * FROM mtopeniot.etsensor WHERE ↵
    ↵ entity_id = '\',\'Sensor:001\'\'
5 > ORDER BY time_index DESC LIMIT 3"}'
6 ,
7 HTTP/1.1 200 OK
8 content-type: application/json; charset=UTF-8
9 content-length: 355
10
11 {
12   "cols":[
13     "entity_id", "entity_type", "fiware_servicepath", ↵
    ↵ "refhome", "temperature", "time_index", "timeinstant"
14   ],

```

```

15     "rows": [
16         ["Sensor:001", "Sensor", "/", null, "16.6", ←
           ↳ 1593550445000, 1593550445000],
17         ["Sensor:001", "Sensor", "/", null, "25.2", ←
           ↳ 1593550438000, 1593550430000],
18         ["Sensor:001", "Sensor", "/", null, "24.7", ←
           ↳ 1593550430000, 1593550430000]
19     ], "rowcount": 3, "duration": 13.070735
20 }

```

4.4. FASE DE TRANSICIÓN

En esta fase del PUD y de acuerdo a la Tabla 3.1 se llevan a cabo dos iteraciones que concluyen con el sistema final completamente desarrollado: *Iteración 6. Generación del dataset y publicación de resultados* e *Iteración 7. Documentación del proyecto*. A continuación, se describen ambas.

4.4.1. Iteración 6: Generación del dataset y publicación del mismo en GitHub

El objetivo de esta iteración es obtener un conjunto de datos (dataset) con los datos recolectados por los sensores y que se encuentran almacenados en la plataforma de FIWARE Lab y hacerlos disponibles públicamente para que puedan ser accedidos y descargados por terceros con distintos propósitos. El uso de datasets generados mediante sensores y otros dispositivos de monitorización se está haciendo cada vez más popular y se publican en sitios web *Open Data* para incrementar el conocimiento que se tiene sobre distintos ámbitos y para que puedan retroalimentar algoritmos de predicción, entre otros.

En esta iteración, vamos a crear un dataset a partir de los datos que hemos obtenido como se describe en la Sección 4.3.4. Es importante considerar aquí, que nuestro conjunto de datos va a ser muy pequeño dado que sólo tenemos un sensor debido a las limitaciones mencionadas anteriormente, pero que lo que se pretende demostrar aquí no es el conjunto de datos sino cómo se obtiene, por tanto esta operación es independiente del volumen de datos que hayamos sido capaces de recolectar.

Para ello, desde nuestro entrono de FIWARE, vamos a tener que bajarnos dos paquetes de Python: *crate* y *csv*. Para ello, lo primero es instalarnos el instalador de paquetes de Python *pip* con el comando *apt install python3-pip*. Después, usamos el comando *pip3 install* y el nombre del paquete que vayamos a instalar. Tras instalar los paquetes, hay que crear un script en Python para generar el dataset, tal y como muestra el Listado 4.36. En este Listado se importan los paquetes que hemos instalado anteriormente. Tras ello, en la línea 4, nos conectamos a la base de datos de CrateDB. En la línea 6, ejecutamos la *query* necesaria para obtener todos los datos de la tabla en la que hemos guardado los datos con el *time_index* de forma ascendente y que la entidad sea igual a *Sensor:001*. Tras ello, en la línea 7, se pasan todas entradas de esa *query* a la variable *rows*. Luego, en la línea 9 creamos un escritor CSV, con el que escribir datos a un archivo con formato CSV. En la línea 10 se crea el encabezado del archivo CSV, y en la línea 11 se escribe. Por último, en las líneas 13 y 14, se escriben todas las filas en el archivo CSV.

Listado 4.36: Script para generar el dataset

```

1  from crate import client
2  import csv
3
4  connection = client.connect('http://localhost:4200')
5  cursor = connection.cursor()
6  cursor.execute("SELECT * FROM mtopeniot.etsensor WHERE ←
                  ↳ entity_id = 'Sensor:001' ORDER BY time_index ASC", ("Sensor:001",))
7  rows = cursor.fetchall()
8
9  f = csv.writer(open("dataset.csv", 'w'))

```

```
10 first_row = [column[0] for column in cursor.description]
11 f.writerow(first_row)
12
13 for row in rows:
14     f.writerow(row)
```

Cuando ya tengamos generado el dataset, el último paso sería subir el dataset y el script a un repositorio público de GitHub, el cuál va a ser el repositorio de este proyecto. Para ello, en la carpeta raíz del repositorio, se crea una carpeta dedicada para el dataset y se copian los archivos. Hecho esto, se añaden estos cambios al repositorio con el comando *git add -A*. Después, se hace un commit con el comando *git commit -m "Added dataset and script to generate it"*. Por último, solo faltaría subir estos cambios al repositorio con el comando *git push*.

4.4.2. Iteración 7: Documentación del proyecto

Esta iteración está dedicada a la escritura de la documentación del TFG, la cuál representa este documento. El repositorio del proyecto se puede encontrar en siguiente enlace: https://github.com/danibaal98/TFG-Protocolo_LoRa

CONCLUSIONES

En este capítulo se comentarán las conclusiones obtenidas tras la finalización de este TFG, incluyendo los objetivos que se han podido o no completar, las competencias que, ya conseguidas a lo largo del grado, se han visto reforzadas, el coste final del proyecto, y por último las posibles propuestas de trabajo futuro.

5.1. REVISIÓN DE LOS OBJETIVOS

En esta sección se van a revisar los objetivos definidos en el Capítulo 2 para comprobar que objetivos se han cumplido:

1. **Prototipado de un conjunto pequeño de dispositivos finales con capacidad de recolección de energía a través de un panel solar y un módulo de comunicaciones LoRa.** Este objetivo parcial no se ha cumplido de forma parcial, ya que debido a la situación de estado de alarma no se ha podido tener acceso al plataforma PLATINO, la cual iba a ser la pieza principal de este proyecto. No obstante, para probar la comunicación LoRa entre un nodo y su pasarela, y entre la pasarela y la plataforma de FIWARE Lab se han utilizado las placas LoPy4.
2. **Implementación de un algoritmo neutral en energía.** Este objetivo se ha conseguido con éxito, ya que se ha conseguido implementar un algoritmo neutral en energía y la evaluación del mismo, pero, como ha pasado con el primer objetivo, no se ha podido implementar en la plataforma PLATINO debido a que no se ha podido disponer de ella.
3. **Implementación de un protocolo de comunicación basado en LoRa.** Este objetivo parcial se ha conseguido con éxito, ya que se ha conseguido que un nodo productor envíe los datos producidos a través de LoRa a una pasarela, pero no implementado en la plataforma PLATINO, sino en las placas LoPy4.
4. **Implementación de un protocolo de comunicación basado en REST entre el gateway y la plataforma FIWARE.** Este objetivo parcial se ha conseguido con éxito, ya que, aunque no el nodo PLATINO, la placa LoPy4 que hace de pasarela puede comunicarse con FIWARE Lab a través de HTTP.
5. **Despliegue de la red en entorno real.** Este objetivo parcial no se ha podido realizar porque debido a la situación de pandemia y de confinamiento, no se ha podido disponer de las plataformas PLATINO, por lo que no se ha podido realizar dicho despliegue.
6. **Evaluación de la red mediante un experimento real.** Este objetivo parcial tampoco se ha podido llevar a cabo debido a los problemas anteriormente descritos, ya que no se ha podido disponer de la plataforma PLATINO ni de una red desplegada.

7. **Publicación del dataset generado en un repositorio público en *GitHub*.** Este objetivo se ha conseguido realizar con éxito, ya que con los datos que las placas LoPy4 han producido y se han guardado en FIWARE Lab se ha podido realizar un dataset y publicarlo en el repositorio del proyecto en *GitHub*.

5.2. COMPETENCIAS ADQUIRIDAS

En la elaboración de este TFG, se han reforzado las siguientes competencias específicas de la Intensificación de Ingeniería de Computadores:

- **Capacidad de diseñar e implementar software de sistema y de comunicaciones.** En este proyecto se ha realizado un sistema de comunicaciones entre los distintos nodos y la pasarela a través de LoRa, así como la propia comunicación de la pasarela con la plataforma FIWARE Lab a través de la red TCP/IP.
- **Capacidad para analizar, evaluar, seleccionar y configurar plataformas hardware para el desarrollo y ejecución de aplicaciones y servicios informáticos.** Se han estudiado, analizado y evaluado distintas tecnologías hardware, como la plataforma PLATINO, la cual es la plataforma principal del proyecto, o las placas LoPy4, ya que debido a que la plataforma PLATINO no ha podido ser usada en el proyecto, se han tenido que buscar otras alternativas.
- **Capacidad para diseñar, desplegar, administrar y gestionar redes de computadores.** Para la realización de este proyecto se ha tenido que hacer el despliegue y configuración en la plataforma de FIWARE Lab.

5.3. COSTES DEL PROYECTO

En esta sección se va a calcular el coste total aproximado del proyecto. Teniendo en cuenta las horas realizadas, tal y como se puede ver en el Capítulo 4.1.2, se obtiene que se han dedicado un total de 390 horas en el desarrollo del proyecto, y el sueldo medio de un desarrollador software en España es de 16,95 € [26], por lo que el coste total con respecto al desarrollo es de 6.610,5 €.

En la Tabla 5.1 se puede observar un resumen de los costes del proyecto.

| Recurso | Coste | Cantidad | Total |
|------------------------|-----------|----------|-----------|
| Personal | 6.610,5 € | | 6.610,5 € |
| LoPy4 Multipack | 110,90 € | 1 | 110,90 € |
| HP Pavilion 15-BC409NS | 800 € | 1 | 800 € |
| HP All in One | 500 € | 1 | 500 € |
| Total | | | 8.021,4 € |

Tabla 5.1: Coste total del proyecto

5.4. TRABAJO FUTURO

Para acabar con este TFG, se van a exponer una serie de posibles trabajos futuros con los que poder ampliar dicho proyecto:

- Ya que debido a la situación producida por el COVID-19 y el estado de alarma, no se ha podido tener acceso al nodo PLATINO, un posible trabajo futuro sería realizar todos los objetivos que no se han podido cumplir debido a no poder disponer del nodo PLATINO, es decir, hacer el prototipado, la implementación del algoritmo neutral en energía y del protocolo de comunicación en el nodo, hacer el despliegue de la red y su evaluación.

- Utilización de dos o más recolectores de energía de forma simultanea, para que en los períodos de tiempo que el panel solar no esté recibiendo energía, otro recolector pueda proporcionarla, y así reducir los períodos de inactividad.
- Creación de una interfaz web en la que poder visualizar en tiempo real los datos recibidos a través de la plataforma FIWARE Lab. También se podría usar tecnologías de monitorización y visualización de datos como Grafana para poder visualizar dichos datos.

BIBLIOGRAFÍA

- [1] 3GPP, *NB-IoT*. dirección: https://www.3gpp.org/news-events/1785-nb_iot_complete.
- [2] L. Alliance, *LoRa*. dirección: <https://lora-alliance.org/>.
- [3] Atalassian, *Trello*. dirección: <https://trello.com/es>.
- [4] A. Caruso y col., «Experimenting Forecasting Models for Solar Energy Harvesting Devices for Large Smart Cities Deployments», inf. téc.
- [5] F. Community, *FIWARE-NGSI v2 Specification*, 2018. dirección: <http://fiware.github.io/specifications/ngsiv2/stable/>.
- [6] F. Community, *Crear cuenta en FIWARE Lab*, 2020. dirección: <https://docs.google.com/forms/d/e/1FAIpQLSflz2JxJ8zwytwYG9CJpauYPaOkjEs88SMqBul9x1UWdzh3w/viewform>.
- [7] F. Community, *Orion Context Broker*, 2020. dirección: https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/ocb/.
- [8] F. Community, *FIWARE Lab*. dirección: <https://www.fiware.org/developers/fiware-lab/>.
- [9] F. Community, *Time-Series Data*. dirección: <https://github.com/FIWARE/tutorials.Time-Series-Data>.
- [10] M. Corporation, *Azure IoT Hub*. dirección: <https://azure.microsoft.com/es-es/services/iot-hub/#features>.
- [11] M. Corporation, *GitHub*. dirección: <https://github.com/>.
- [12] M. Corporation, *Microsoft Teams*. dirección: <https://www.microsoft.com/es-es/microsoft-365/microsoft-teams/group-chat-software>.
- [13] M. Corporation, *Visual Studio Code*. dirección: <https://code.visualstudio.com/>.
- [14] S. Escolar, S. Chessa y J. Carretero, «Optimization of Quality of Service in Wireless Sensor Networks Powered by Solar Cells», jul. de 2012, págs. 269-276, ISBN: 978-1-4673-1631-6. DOI: [10.1109/ISPA.2012.43](https://doi.org/10.1109/ISPA.2012.43).
- [15] S. Escolar, S. Chessa y J. Carretero, «Energy Management in Solar Cells Powered Wireless Sensor Networks for Quality of Service Optimization», inf. téc.
- [16] S. Escolar y col., «The PLATINO Experience: A LoRa-based Network of Energy-Harvesting Devices for Smart Farming», inf. téc.
- [17] ESHorizonte2020, *7º Programa Marco*, 2013. dirección: <https://eshorizonte2020.es/mas-europa/71-programa-marco>.
- [18] FIWARE Community, *Acceso a FIWARE Lab*. dirección: <https://cloud.lab.fiware.org/auth/login/>.
- [19] P. S. Foundation, *Python*, 2018. dirección: <https://docs.python.org/2/tutorial/>.
- [20] Google, *Protocol Buffers*. dirección: <https://developers.google.com/protocol-buffers>.
- [21] J. D. Hunter, «Matplotlib: A 2D graphics environment», *Computing in Science & Engineering*, vol. 9, n.º 3, págs. 90-95, 2007. DOI: [10.1109/MCSE.2007.55](https://doi.org/10.1109/MCSE.2007.55).
- [22] IBM, *Watson IoT Platform*. dirección: <https://www.ibm.com/internet-of-things/solutions/iot-platform/watson-iot-platform>.
- [23] D. Inc., *Install Docker Engine on Ubuntu*. dirección: <https://docs.docker.com/engine/install/ubuntu/>.
- [24] D. Incorporated, *Docker*, 2020. dirección: <https://www.docker.com/>.
- [25] D. Incorporated, *Doker Compose*, 2020. dirección: <https://docs.docker.com/compose/>.

- [26] Indeed, *Salarios para empleos de Desarrollador/a de software en España*. dirección: <https://es.indeed.com/salaries/desarrollador-de-software-Salaries?period=hourly>.
- [27] Indra, *Sofia2*, 2016. dirección: <https://sofia2.readthedocs.io/en/latest/index.html>.
- [28] A. Kansal y M. B. Srivastava, «An environmental energy harvesting framework for sensor networks», en *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03.*, 2003, págs. 481-486.
- [29] M. Kuzman y col., «A Testbed and an Experimental Public Dataset for Energy-Harvested IoT Solutions», en *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, págs. 869-876.
- [30] G. R. Limited, *MicroPython*, 2018. dirección: <https://micropython.org/>.
- [31] Linuxize, *How to Install and Use Docker Compose on Ubuntu 18.04*. dirección: <https://linuxize.com/post/how-to-install-and-use-docker-compose-on-ubuntu-18-04/>.
- [32] livius, *How to get a random number in a range*, 2017. dirección: https://forum.pycom.io/topic/1378/solved-how-to-get-random-number-in-a-range/3?_=1592586241982&lang=es.
- [33] J. Ltd., *Draw.io*, 2020. dirección: app.diagrams.net.
- [34] M. Matin y M. Islam, «Overview of Wireless Sensor Network», en *Wireless Sensor Networks - Technology and Protocols*, InTech, sep. de 2012. DOI: 10.5772/49376. dirección: <http://www.intechopen.com/books/wireless-sensor-networks-technology-and-protocols/overview-of-wireless-sensor-network>.
- [35] K. Mekki y col., «A comparative study of LPWAN technologies for large-scale IoT deployment», *ICT Express*, vol. 5, n.º 1, págs. 1-7, mar. de 2019, ISSN: 24059595. DOI: 10.1016/j.icte.2017.12.005.
- [36] I. C. y Moisés Rodríguez, *Tema 01.02. Proceso Unificado de Desarrollo*, v1.6, Ingeniería del Software II, Escuela Superior de Informática, Ciudad Real, España, 2018.
- [37] C. Moser, J. Chen y L. Thiele, «Dynamic power management in environmentally powered systems», en *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2010, págs. 81-88.
- [38] Overleaf, *Overleaf*, 2020. dirección: <https://www.overleaf.com/>.
- [39] H. Packard, *HP Pavilion 15-BC409NS*. dirección: <https://support.hp.com/es-es/document/c06200443>.
- [40] H. Packard, *PC Desktop HP ENVY 23-d101es TouchSmart All-in-One*. dirección: <https://support.hp.com/es-es/product/hp-envy-23-d100-touchsmart-all-in-one-desktop-pc-series/5330735/model/5363240/manuals>.
- [41] PlatformIO, *PlatformIO*. dirección: <https://docs.platformio.org/en/latest/>.
- [42] Pycom, *LoPy4 datasheet*. dirección: <https://docs.pycom.io/datasheets/development/lopy4/>.
- [43] Pycom, *LoPy4 Multipack*. dirección: <https://pycom.io/product/lopy4-multipack/>.
- [44] Pycom, *Pymakr*. dirección: <https://docs.pycom.io/gettingstarted/installation/pymakr/>.
- [45] Python, *struct library*, 2020. dirección: <https://docs.python.org/3/library/struct.html>.
- [46] D. Rakhmatov y S. Vrudhula, «An Analytical High-Level Battery Model for Use in Energy Management of Portable Electronic Systems.», feb. de 2001, págs. 488-493, ISBN: 0-7803-7247-6. DOI: 10.1109/ICCAD.2001.968687.
- [47] S. S.A., *Sigfox*. dirección: <https://www.sigfox.com>.
- [48] K. Sohrabi y col., «Protocols for self-organization of a wireless sensor network», *IEEE Personal Communications*, vol. 7, n.º 5, págs. 16-27, 2000.
- [49] N. Sornin y col., «LoRaWAN Specification», inf. téc., 2015.
- [50] B. Stroustrup, *C++*. dirección: <https://www.cplusplus.com/>.
- [51] P. Systems, *Los retos de la agricultura 4.0 en el siglo XXI*, 2019. dirección: <https://www.proximasystems.net/agricultura/los-retos-de-la-agricultura-4-0-en-el-siglo-xxi/>.
- [52] 3. P. Team, *L^AT_EX2_ε*, 2019. dirección: <https://www.latex-project.org/help/documentation/usrguide.pdf>.
- [53] L. Torvalds, *Git*. dirección: <https://git-scm.com/>.

-
- [54] *urequests*, 2018. dirección: <https://github.com/micropython/micropython-lib/blob/master/urequests/urequests.py>.

ANEXOS

MÉTODOS ADICIONALES

En este anexo se muestra el código fuente de los métodos *compute_cost_assignment* y *recompute_battery_level*, los cuales sirven para calcular la asignación de coste para la asignación inicial de planes de ejecución, y para reajustar el nivel de la batería en cada uno de los slots cada vez que se hace *Upgrade*, *Downgrade* o se realiza una reoptimización, respectivamente. Cada uno de estos archivos imprime los datos en archivos CSV, tal y como se comenta en el proyecto.

Listado A.1: Método *compute_cost_assignment*

```

1 void assignmentClass::compute_cost_assignment(int plan, float ↵
    ↵ *cost_of_plan, int *ddMMyyhhmmss, double ↵
    ↵ *battery_at_slots, int *hour, int init, int month) {
2     int i;
3     double battery_level;
4     float ep_hour = 0.0;
5     float ep_slot = 0.0;
6     float ep_slot_remaining;
7     float EP = 0.0;
8     double consumption = 0.0;
9     float K = D[month];
10    char filename[256];
11
12    // register results
13    FILE *fs;
14    sprintf(filename, "%s-%d.csv", "results/initial_assignment", ↵
        ↵ month);
15    fs = fopen(filename, "w");
16    fprintf(fs, "slot,date,plan,QoS,battery,consumption,ep_slot\n");
17
18    std::cout << "El mes es " << month << std::endl;
19
20    // instantaneous energy production ->
21    // we compute the energy produced in every month and...
22    EP = (D[month] * (EFFICIENCY / 100) * S) / Vld; //energy produced in
        the hour i IN Amperes
23    EP = EP * 1000; //energy produced in milliAmperes
24
25    if ((*hour >= MIN_HOUR[month]) && (*hour < MAX_HOUR[month])) {
26        ep_hour = (-1 * (*hour - H)) * (*hour - H) * (EP / (4 * ↵
            ↵ STD_HOURS[month])) + EP;
27        ep_slot = ep_hour / (SLOTS / 24);
28        ep_slot_remaining = ep_hour - (ep_slot * (SLOTS / 24));
29    }
30    else {
31        ep_slot = 0;
32        ep_slot_remaining = 0;
33    }
34
35    for (i = init; i <= SLOTS; i++) {

```

```

36 // in every month we compute the irradiance in the month and we distribute in
37 if (ddMMyyhhmmss[3] != *hour) {
38     // the new hour
39     *hour = ddMMyyhhmmss[3];
40
41     // is the average of the month and the correction factor due to the hour
42     if ((*hour >= MIN_HOUR[month]) && (*hour < ↵
43         ↵ MAX_HOUR[month])) {
44         ep_hour = (-1 * (*hour - H)) * (*hour - H) * (EP / (4 * ↵
45             ↵ STD_HOURS[month])) + EP;
46         ep_slot = ep_hour / (SLOTS / 24);
47         ep_slot_remaining = ep_hour - (ep_slot * (SLOTS / 24));
48     }
49     else {
50         ep_slot = 0;
51         ep_slot_remaining = 0;
52     }
53     if (i == SLOTS)
54         ep_slot += ep_slot_remaining;
55
56     consumption = (cost_of_plan[plan] / (60 * 60)) * Vld * ↵
57         ↵ TIME_SLOTS / (VOLTAGE * BATTERIES);
58     // compute the battery level at slot i
59     battery_level = battery_at_slots[i - 1] + (EFFICIENCY / ↵
60         ↵ 100) * rectifier(ep_slot, consumption) - ↵
61         ↵ rectifier(consumption, ep_slot);
62     if (battery_level > CAPACITY)
63         battery_at_slots[i] = CAPACITY;
64     else
65         battery_at_slots[i] = battery_level;
66
67     // register data to file
68     fprintf(fs, "%d,", i);
69     fprintf(fs, "%d-%d-%d:%d:%d:%d,", ddMMyyhhmmss[0], ↵
70         ↵ ddMMyyhhmmss[1], ddMMyyhhmmss[2], ddMMyyhhmmss[3], ↵
71         ↵ ddMMyyhhmmss[4], ddMMyyhhmmss[5]);
72     fprintf(fs, "%d,", plan);
73     fprintf(fs, "%f,", QoS[plan]);
74     fprintf(fs, "%f,", battery_at_slots[i]);
75     fprintf(fs, "%f,", consumption);
76     fprintf(fs, "%f\n", ep_slot);
77
78     get_time_from_seconds(TIME_SLOTS, ddMMyyhhmmss);
79 }
80 fclose(fs);
81 }

```

Listado A.2: Método recompute_battery_level

```

1 void assignmentClass::recompute_battery_level(int plan, int ↵
2     ↵ *assignments, double *battery_at_slots, float ↵
3     ↵ *cost_of_plan, int s, int hour, float *QoS, int month, ↵
4     ↵ int optimization) {
5     double battery_level, consumption = 0;
6     int i;
7     float ep_hour = 0.0;
8     float ep_slot = 0.0;
9     float ep_slot_remaining;
10    float EP = 0.0;
11    float K = D[month];
12    char filename_assignment[256], filename_repotimization[256];
13    FILE *fs, *fr;

```

```

11 // debe estar la hora a 0!!
12 int new_date[6] = {1, month + 1, 2012, 0, 0, 0};
13 EP = (D[month] * (EFFICIENCY / 100) * S) / Vld; //energy produced in
    the hour i IN Amperes
14 EP = EP * 1000; //energy produced in milliAmperes
15
16 if ((hour >= MIN_HOUR[month]) && (hour < MAX_HOUR[month])) {
17     std::cout << "Dentro_mes_" << month << std::endl;
18     ep_hour = (-1 * (hour - H)) * (hour - H) * (EP / (4 * ←
    ← STD_HOURS[month])) + EP;
19     if (optimization)
20         ep_slot = (ep_hour + (ep_hour * COEFFICIENT)) / (SLOTS / 24);
21     else
22         ep_slot = ep_hour / (SLOTS / 24);
23     ep_slot_remaining = ep_hour - (ep_slot * (SLOTS / 24));
24 }
25 else {
26     std::cout << "Dentro_2_mes_" << month << std::endl;
27     ep_slot = 0;
28     ep_slot_remaining = 0;
29 }
30
31 sprintf(filename_assignment, "%s-%d.csv", ←
    ← "results/initial_assignment", month);
32 fs = fopen(filename_assignment, "a");
33
34 sprintf(filename_repotimization, "%s-%d-%f.csv", ←
    ← "results/reoptimization", month, COEFFICIENT);
35 fr = fopen(filename_repotimization, "a");
36 // compute the production at this slot
37
38 for (i = 1; i <= SLOTS; i++) {
39
40     consumption = (cost_of_plan[assignments[i - 1]] / (60 * ←
    ← 60)) * Vld * TIME_SLOTS / (VOLTAGE * BATTERIES);
41     if (i >= s) {
42         if (new_date[3] != hour) {
43             //the new hour
44             hour = new_date[3];
45
46             //is the average of the month and the correction factor due to the hour
47             if ((hour >= MIN_HOUR[month]) && (hour < ←
    ← MAX_HOUR[month])) {
48                 ep_hour = (-1 * (hour - H)) * (hour - H) * (EP / (4 * ←
    ← STD_HOURS[month])) + EP;
49                 if (optimization)
50                     ep_slot = (ep_hour + (ep_hour * COEFFICIENT)) / ←
    ← (SLOTS / 24);
51                 else
52                     ep_slot = ep_hour / (SLOTS / 24);
53                 ep_slot_remaining = ep_hour - (ep_slot * (SLOTS / 24));
54             }
55             else {
56                 ep_slot = 0;
57                 ep_slot_remaining = 0;
58             }
59         }
60         if (i == SLOTS)
61             ep_slot += ep_slot_remaining;
62
63         battery_level = battery_at_slots[i - 1] + (EFFICIENCY / ←
    ← 100) * rectifier(ep_slot, consumption) - ←
    ← rectifier(consumption, ep_slot);
64         if (battery_level > CAPACITY)

```

```

65     battery_at_slots[i] = CAPACITY;
66     else
67         battery_at_slots[i] = battery_level;
68
69     // register data to file
70     if (optimization) {
71         fprintf(fr, "%d,", i);
72         fprintf(fr, "%f,", ep_slot);
73         if (ep_slot > consumption)
74             fprintf(fr, "%s\n", "superhabit");
75         else
76             fprintf(fr, "%s\n", "deficit");
77     }
78 }
79
80 fprintf(fs, "%d,", i);
81 fprintf(fs, "%d-%d-%d:%d:%d:%d,", new_date[0], new_date[1], ↵
82     ↵ new_date[2], new_date[3], new_date[4], new_date[5]);
83 fprintf(fs, "%d,", assignments[i - 1]);
84 fprintf(fs, "%f,", QoS[assignments[i - 1]]);
85 fprintf(fs, "%f,", battery_at_slots[i]);
86 fprintf(fs, "%f,", consumption);
87 if ((new_date[3] >= MIN_HOUR[month]) && (new_date[3] < ↵
88     ↵ MAX_HOUR[month]))
89     fprintf(fs, "%f\n", ep_slot);
90 else
91     fprintf(fs, "%f\n", 0.0);
92
93 get_time_from_seconds(TIME_SLOTS, new_date);
94 }
95 fclose(fs);
96 fclose(fr);
97 }

```