UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA SOFTWARE PARA EL CONTROL DEL ALUMBRADO PÚBLICO MEDIANTE EL DESPLIEGUE DE REDES NB-IOT Y LTE-M

Miguel Pérez Ávila

2020

RESUMEN

El concepto de el Internet de las Cosas es relativamente joven, pero está decidido a cambiar completamente nuestras vidas. Desde controlar las luces en nuestros hogares hasta monitorizar el estado estructural de la cimentación de una presa, el Internet de las Cosas nos posibilita miles de nuevas aplicaciones. Dentro de estas aplicaciones destacan las llamadas ciudades inteligentes, que buscan optimizar cada vez más procesos y es en ellas donde se centra este Trabajo Fin de Máster, con el objetivo de diseñar un sistema de control de alumbrado público.

El control del alumbrado público no es un problema sencillo. El control de cientos de miles de farolas que deben ser identificadas individualmente, que reporten información en tiempo real y puedan ser controladas con un riguroso control de tiempo genera retos tecnológicos importantes.

Este Trabajo Fin de Máster muestra el estudio y desarrollo de una arquitectura capaz de solventar, de manera eficiente, los problemas a los que se enfrenta el control de alumbrado público. Esta eficiencia no solo se centra en solucionar problemas de una arquitectura ya existente, sino, además, obtener una capacidad modular que permita un desacoplo tecnológico. Este desacoplo otorga grandes capacidades a la red, permitiendo la inclusión de nuevos servicios sin necesidad de una restructuración completa o la posibilidad de rediseñar módulos de la arquitectura para adaptarse a nuevos retos sin afectar al resto del sistema.

SUMMARY

The concept of the Internet of Things is relatively young, but it is determined to completely change our lives. From controlling the lights in our homes to monitoring the structural state of the foundation of a dam, the Internet of Things enables thousands of new applications. Among these applications, the so-called smart cities stand out, which seek to optimize more and more processes and it is in them that this Master's Thesis is focused, with the aim of designing a public lighting control system.

Control of public lighting is not a simple problem. The control of hundreds of thousands of streetlights that must be individually identified that report information in real time and can be controlled with rigorous time control generates significant technological challenges.

This Master's Thesis shows the study and development of an architecture capable of efficiently solving the problems faced by the control of public lighting. This efficiency is not only focused on solving problems of an existing architecture, but also on obtaining a modular capacity that allows a technological decoupling. This decoupling gives the network great capabilities, allowing the inclusion of new services without the need for a complete restructuring or the possibility of redesigning architecture modules to adapt to new challenges without affecting the rest of the system.

AGRADECIMIENTOS

Este trabajo es el resultado de muchos meses de trabajo explorando campos de la ingeniería en los que no había profundizado hasta el momento. Donde, además, tuve que conciliar el desarrollo de este proyecto con las horas de trabajo en RBZ que resultaron en más de una noche de insomnio.

Por ello, quiero agradecer este trabajo a muchas personas, primero a RBZ y a mi tutor Daniel Amor por permitirme embarcarme en este aprendizaje de nuevas tecnologías y desarrollo que me ha permitido formarme como un ingeniero versátil. Continuar agradeciendo a mi cotutor Álvaro Gutiérrez, por haber confiado en mí y ayudar a desarrollar mi carrera profesional con grandes oportunidades.

A mis compañeros Daniel, Paula y Amadeo por haberme acompañado en esta aventura que es teleco. Con ellos el viaje ha sido mucho más sencillo.

Y, por último, a mi familia. Por soportar tantos momentos de silencio, con la cara agria por tener la cabeza en otro lado y por mi constante búsqueda de desafíos más allá de mis límites. Gracias.

PALABRAS CLAVE

IoT Internet de las Cosas
RTOS Sistemas Operativos en Tiempo Real
MongoDB.
Python.
MVVM Modelo, Vista, Vista Modelo.
C#
C
Zephyr.
Linux.
Raspberry Pi.

KEYWORDS

IoT Internet of Things
RTOS Real-Time Operating Systems
MongoDB.
Python.
MVVM Model, View, View Model.
C#
C
Zephyr.
Linux.
Raspberry Pi.

ÍNDICE DEL CONTENIDO

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. INTRODUCCIÓN Y MOTIVACIÓN

1.1. INTRODUCCIÓN AL INTERNET DE LAS COSAS (IOT)

En la década de los 70 se produjo en un laboratorio de Estados Unidos el que puede ser considerado el desarrollo tecnológico que ha hecho posible el mayor cambio de la sociedad en toda la historia del ser humano. En aquel laboratorio se desarrolló Arpanet [1] que permitió la conexión de distintos ordenadores formando una red descentralizada con múltiples caminos entre dos puntos. Esta red es ahora conocida como Internet [2].

Internet ha posibilitado el desarrollo de cientos de diferentes tipos de aplicaciones que han cambiado nuestra forma de vida desde el uso de los correos electrónicos a las videollamadas. El ingenio de nuevas aplicaciones está a la orden del día, aumentando de forma incesante las posibilidades que ofrece la red de Internet. Esto nos lleva al concepto de el Internet de las cosas o Internet of Things (IoT, por sus siglas en inglés) sobre el que se basa este TFM.

El IoT ha sido definido por la Recomendación UIT-T Y.2060 (06/2012) [3] como una infraestructura global para la sociedad de la información, que permite servicios avanzados mediante la interconexión de cosas (físicas y virtuales) basadas en tecnologías de información y comunicación interoperables existentes y en evolución.

Esta definición técnica viene a hablar de un mundo altamente conectado en el que varios cientos de millones de dispositivos electrónicos podrán comunicarse a través de la red. Cada uno de estos dispositivos tendrá el objetivo de aportar información sobre los elementos físicos que nos rodean en la vida real, siendo estos desde pulseras para controlar nuestra actividad física a pequeños sensores dentro de nuestros electrodomésticos para controlar el rendimiento y detectar posibles errores.

Con este nuevo concepto del IoT la industria se encuentra ante un nuevo abanico de posibilidades en el desarrollo de nuevas aplicaciones. Muchos campos son los que antes ya se abastecían del uso de Internet, pero ahora otros muchos más se inician en el uso de esta tecnología. Gracias al desarrollo de la electrónica y la miniaturización de los componentes electrónicos somos capaces de disponer de elementos que miden y actúan en sitios que antes no podíamos concebir. Como pueden ser, por ejemplo, sensores en los frenos de los coches para comprobar el estado del disco, dándonos información del desgaste y poder substituirlo antes de que deje de ser seguro.

Pero al poder incluir cada vez más sensores enviando información a través de la red surge la necesidad de diseñar nuevas redes capaces de gestionar esta ingente cantidad de datos, permitiendo una evolución constante de Internet. Estas nuevas redes son específicas para el desarrollo del IoT y algunas son utilizadas por la industria activamente como: LoRa [4] o más recientemente NB-IoT [5] o LTE-M [6].

1.1.1 EVOLUCIÓN DEL IOT

El concepto de IoT aparece públicamente en un artículo para el diario RFID [7] firmado por Kevin Ashton titulado “Esa cosa del ‘Internet de las cosas’”. En ella, Kevin Ashton introduce este concepto innovador de un mundo aún más conectado recopilando información en tiempo real. Fue después en entrevistas realizadas al autor en el que explicó que el concepto se llevaba manejando entre los grupos de investigación varios años antes.

Nos tenemos que remontar al año 1999 cuando este mismo hombre, Kevin Ashton, tuvo la iniciativa de formar una agrupación de investigadores llamada Auto-ID Center [8] en el Instituto Tecnológico de Massachussets (MIT) [9] donde su objetivo era investigar acerca del sucesor del código de barras y

como solución se discutió sobre la identificación por radiofrecuencia (RFID), un sistema de almacenamiento electrónico y recuperación de datos remotos mediante etiquetas.

Por otro lado, Cisco Systems [10] determina el nacimiento del Internet de las cosas en el momento en que los dispositivos conectados superen al número de personas que pueblan el planeta. En el año 2020 Naciones Unidas [11] considera que la población mundial es de alrededor de 7,6 mil millones de personas y, según Cisco, en el mismo año hay alrededor de 50 mil millones de dispositivos conectados a Internet, por lo que según su definición ya hemos entrado en la época del Internet de las cosas.

Cisco data el comienzo del Internet de las cosas alrededor del año 2008/2009, como se puede observar en la Ilustración 1. En la Ilustración se puede apreciar como en el año 2010 la población mundial era de 6,8 mil millones de personas y el número de dispositivos de 12,5 mil millones. También se debe destacar el crecimiento exponencial de los dispositivos conectados a la red.

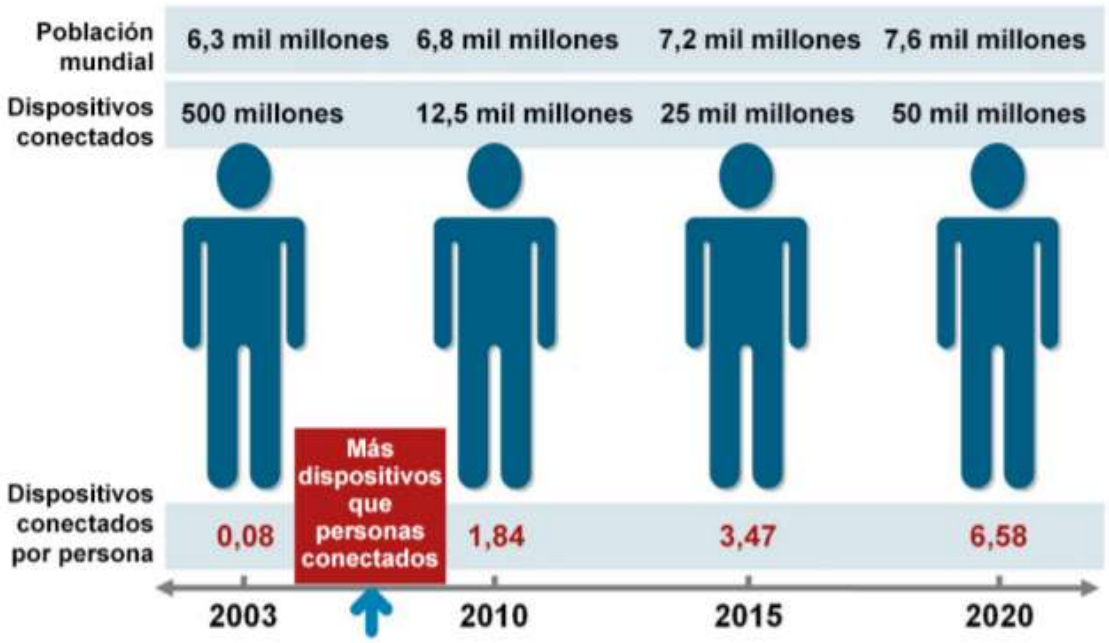

Ilustración 1 Nacimiento del IoT. Fuente ISBG Cisco.

Este crecimiento exponencial es debido a diversos factores, entre ellos el abaratamiento de los costes de desarrollo electrónico, el desarrollo de redes de telecomunicación más potentes, la miniaturización de la electrónica, la implementación de IPv6 etc.

Pero ¿Cuál es el futuro del IoT?, según la industria el Internet de las Cosas ha venido para quedarse y su crecimiento es exponencial hasta tal punto que surge el termino Internet de Todo (Internet of Everything). El IoE, según sus siglas en inglés, es llevar el Internet de las Cosas al extremo, no solo dotando a los elementos cotidianos de conexión de internet sino también de todos los elementos de nuestro alrededor. Pero IoE no está solo limitado a otorgar una conexión a cada elemento; según, nuevamente, Cisco IoE es “La conexión inteligente de personas, procesos, datos y cosas.” Es la inteligencia que proporciona la capacidad de unir las personas con la información recopilada de estos dispositivos.

1.1.2 CARACTERÍSTICAS Y REQUISITOS DE LAS REDES IOT

Se pueden resumir las características de una red IoT, según la Unión Internacional de Telecomunicaciones [12] y su documento "Descripción general de Internet de los objetos" [13] como se describen a continuación:

- Interconectividad. Esta característica es esencial en el IoT pues es la que consolida el concepto, gracias a que permite la compatibilidad y acceso a la infraestructura mundial de información y telecomunicación.
- Servicios relacionados con objetos. IoT gira en torno a la información que es capaz de obtener de los objetos del mundo cotidiano, dando servicios siempre relacionados con esa información.
- Heterogeneidad. Los dispositivos y redes IoT están basados en distintas plataformas hardware y software. Los sistemas IoT son capaces de interactuar entre distintas tecnologías.
- Cambios dinámicos. Los sistemas IoT son capaces de adaptarse a los distintos estados del dispositivo en cuestión, sea este reposo, activo, alarma, etc. Pero también son capaces de soportar cambios dinámicos en el número de dispositivos.
- Escalabilidad. El número de dispositivos se espera que tenga un crecimiento exponencial, por ello la capacidad de gestión de esta ingente cantidad de datos y su procesado convierten la escalabilidad en una característica importante.

Por otra parte, debemos definir los dispositivos IoT y los requisitos que tienen. Según la UIT, el requisito mínimo de todo dispositivo IoT es la capacidad de comunicación. Así mismo, también se incluyen una serie de requisitos de más alto nivel:

- Conectividad basada en la identificación. Cada uno de los dispositivos de la red debe tener un identificador único para que la red IoT pueda gestionar conexiones respecto a ese identificador único.
- Compatibilidad. Es indispensable garantizar una compatibilidad entre los sistemas distribuidos y heterogéneos que componen la red.
- Redes automáticas. Es necesario que las funciones de control de red IoT soporte redes automáticas, a fin de adaptarse al incremento de dispositivos o variaciones en las comunicaciones.
- Capacidades basadas en la ubicación. En algunos servicios las acciones a realizar dependerán de la ubicación de los dispositivos IoT.
- Seguridad. Todos los objetos conectados a la red presentan amenazas de seguridad, en ámbitos tales como la confidencialidad, autenticidad e integridad de datos y servicios.
- Autoconfiguración (plug and play). Los dispositivos IoT deben soportar una configuración automática de tal forma que al instalarse ya estén operativos.

Con estas definiciones se puede resumir una arquitectura IoT como la de la Ilustración 2. En ella se aprecian tres bloques: recopilación de datos (nodos), procesamiento de datos (nube) y visualización de datos (cliente).

Los bloques de procesamiento de datos y recopilación de datos son independientes y automáticos entre sí, por ello no dependen del tercer bloque para un correcto funcionamiento. El tercer bloque es el que permite al ser humano interaccionar con la red, visualizando los datos y permitiéndonos una mejor toma de decisiones.

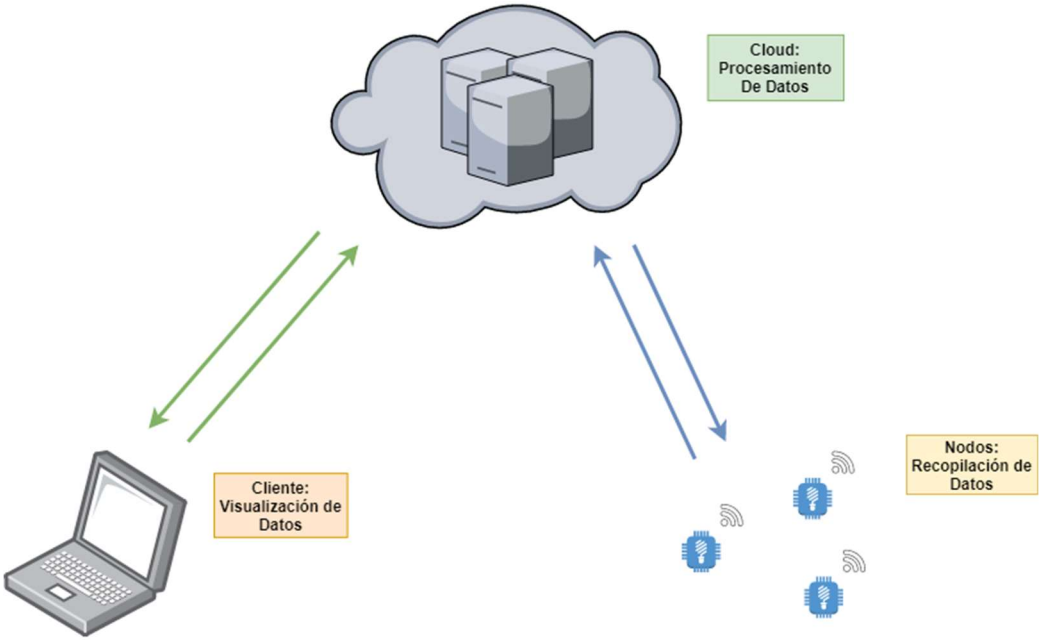

Ilustración 2 Arquitectura genérica IoT.

La anterior Ilustración muestra una arquitectura técnica de como se representaría una arquitectura IoT. Si intentamos representar una arquitectura IoT de una manera más abstracta llegamos a la Ilustración 3:

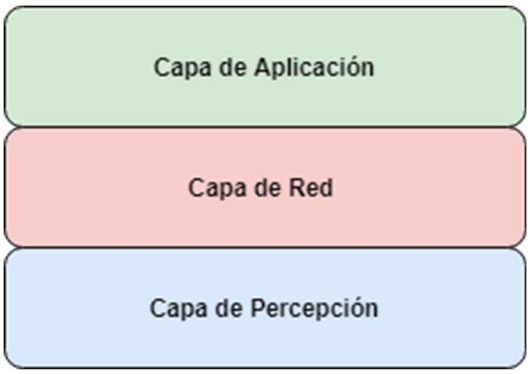

Ilustración 3 Abstracción conceptual de una red IoT.

Se definen por tanto tres capas jerárquicas:

- Capa de Percepción: este nivel es el responsable de recoger las propiedades físicas de los objetos (temperatura, humedad, localización…) mediante sensores y convertir la información en señales digitales para poder transmitirlas por la red. De esta forma, los sensores, la tecnología RFID, códigos de barras o GPS, entre otras, son las tecnologías claves en esta capa.
- Capa de Red: esta capa tiene como función principal enviar los datos recogidos por la capa de percepción hacia su destino mediante la red a través de LoRa, NB-IoT, LTE-M u otras tecnologías.
- Capa de Aplicación: la función de la capa de aplicación es el desarrollo de todo tipo de aplicaciones en función del objetivo que tenga y de los datos recogidos por la capa de percepción. Es el nivel más importante en cuanto al papel que juega en el desarrollo del IoT.

1.1.3 SOLUCIONES ACTUALES PARA REDES IOT

Las soluciones actuales de redes LPWAN (Low Power Wide Area Network) [14] que se utilizan en el sector del IoT se centran en dos tipos: las redes privadas y las redes públicas. Cada una de ellas presenta unas ventajas e inconvenientes. Las redes públicas son redes ya desplegadas por las grandes operadoras de telecomunicaciones del país y tienen la gran ventaja del ahorro en costes por el despliegue de la infraestructura. Sin embargo, usar redes públicas hace a los sistemas dependientes debido a que estas redes no se extienden de forma homogénea por todos los territorios. El uso de redes privadas permite al sistema no ser dependiente de terceros ya que la red es desplegada con la instalación de los elementos de la red, pero eso obliga a un incremento en el coste del desarrollo de la infraestructura y de su despliegue.

Como soluciones actuales comúnmente utilizadas en la industria podemos resumir tres: LoRaWAN, NB-IoT, LTE-M. La primera mencionada, LoRa [4] es una red privada y las dos siguientes, NB-IoT [5] y LTE-M [6] son redes públicas.

LoRaWAN es una red LPWAN diseñada para conectar inalámbricamente dispositivos alimentados por batería al Internet de las Cosas supliendo requisitos como pueden ser: bidireccionalidad en las comunicaciones, seguridad punto a punto, movilidad y servicios de localización. LoRaWAN funciona bajo la asociación de LoRa Alliance [4]. LoRa hace referencia a Long Range Radio, que se puede traducir como radio de largo alcance.

Las redes LoRaWAN contemplan dos tipos de entidades, los LoRa Gateway y los dispositivos finales o nodos. Estos se conectan en una arquitectura de estrella como se muestra en la Ilustración 4. El Gateway es el encargado de encapsular todo el tráfico procedente de los nodos y crear una conexión hacia Internet. Los nodos en cambio son los encargados de generar la conexión individualizada con el Gateway.

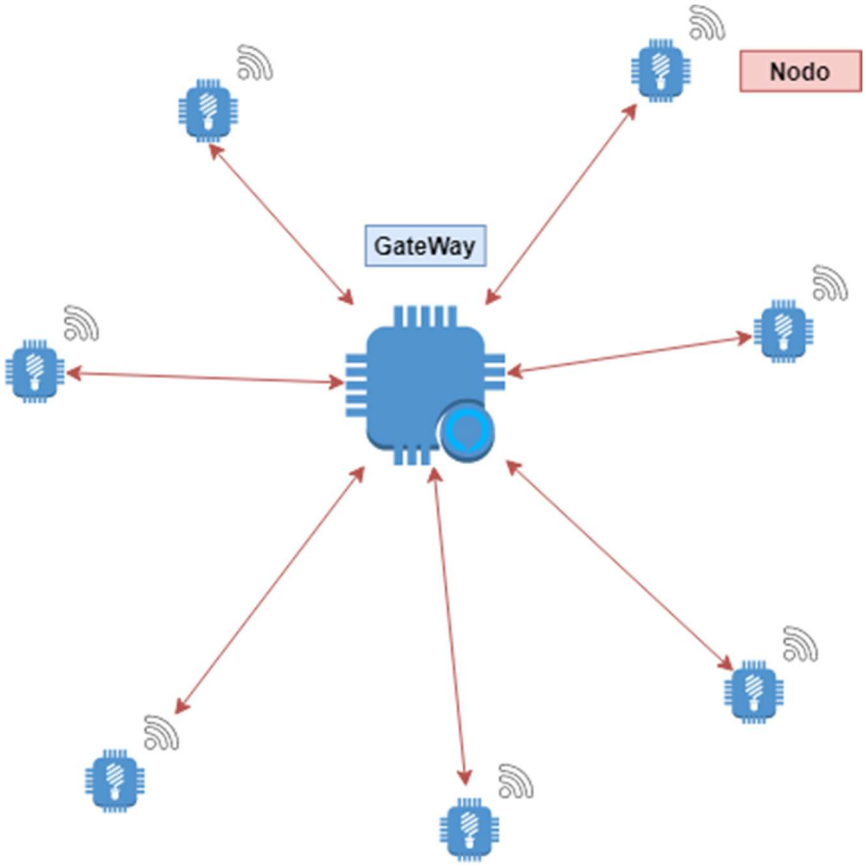

Ilustración 4 Arquitectura estrella.

LoRaWAN presenta las siguientes ventajas:

- Utiliza las frecuencias 868MHz/ 915MHz que están en el espectro libre de todo el planeta.
- Tiene un alcance muy elevado siendo este de 5 km en zonas urbanas y de hasta 15 km en zonas rurales.
- Tiene un consumo de batería muy bajo
- Un solo LoRa Gateway puede gestionar 1.000 nodos.

Y como desventajas:

- Solo puede usarse por aplicaciones que requieran muy bajas velocidades, de alrededor de 27kbps.
- El tamaño de la red está limitado por un parámetro llamado ciclo de trabajo, que se define como el porcentaje de tiempo en que el canal puede ser utilizado. Este parámetro proviene de la limitación de uso de los espectros de frecuencias libres.
- El despliegue de la red.
- Si existe otra red LoRa desplegada en la zona pueden ocurrir interferencias.

La otra solución que está ganando peso en el mercado es NB-IoT que hace referencia a "narrowband for IoT" que se puede traducir como bandas estrechas para el Internet de las Cosas. NB-IoT es una iniciativa de 3GPP [15], que es el organismo encargado de la estandarización de los sistemas de tecnología móvil. Esta iniciativa busca suplir las necesidades de dispositivos de muy baja velocidad de datos que necesitan conectarse a redes móviles y están alimentados por baterías. Como es una iniciativa que busca realizar un estándar el objetivo es la interoperabilidad entre los distintos sistemas IoT.

NB-IoT utiliza tecnologías inalámbricas de grado móvil lo que se corresponde con precios más elevados pero la calidad de los enlaces es mayor. Esto se traduce en que los sistemas tienen un alto rendimiento asociado a la conectividad de las redes móviles, pero son sistemas más complejos y con un consumo de energía mayor.

Como ventajas del NB-IoT podemos resumir:

- Muy buena cobertura urbana. NB-IoT funciona a través de 4G que fue diseñada para entornos urbanos.
- Tiempos de respuesta menores que LoRa con una mejor garantía de servicio.

Y como desventajas:

- Poca transferencia de datos que dificulta las actualizaciones de los dispositivos por aire.
- La transferencia de red y de torre son un problema. Esta pensado para dispositivos principalmente estáticos.

LTE-M es otra iniciativa de 3GPP para estandarizar el acceso a las redes IoT. LTE-M es la abreviatura de LTE-MTC que hace referencia a Long Term Evolution Machine Type Communication, siendo estas las redes de larga evolución 4G orientadas a las comunicaciones máquina a máquina.

Como ventajas de LTE-M se puede resumir:

- Mayores velocidades de envío de datos.
- Soporta movilidad de los elementos en la red.
- Llegaría a permitir enviar voz sobre la red VoIP [16].

Como desventajas:

- Necesita mayor ancho de banda.
- Tiene un coste más elevado.

Como conclusión la industria todavía no ha decidido una red única, esto es debido a que todavía las redes NB-IoT y LTE-M llevan poco tiempo desplegadas a nivel nacional. Pero los beneficios son

claros respecto a la minimización de costes de despliegue de la red y de complejidad de la red en el uso de las redes NB-IoT y LTE-M.

1.2. MOTIVACIÓN

La realización de este Trabajo Fin de Máster se gestó dentro de la organización RBZ Robot Design [17]. RBZ es una empresa enfocada al sector del diseño y producción de electrónica de consumo. Los proyectos que se manejan dentro de la compañía incluyen siempre un diseño y producción de electrónica y, además, se incluye un diseño software para otorgar funcionalidad. Dentro del diseño software se diferencian dos departamentos: el departamento de desarrollo Linux [18] y el departamento de desarrollo de software en tiempo real. El desarrollo de mi actividad fue en este último departamento.

Dentro de la electrónica de consumo, los proyectos relacionados con el Internet de las Cosas están en auge y por ello RBZ busca desarrollar nuevas soluciones acordes al mercado actual. Este Trabajo acude a solucionar esta necesidad enfocándose como un proyecto de investigación y desarrollo de nuevas soluciones dentro del marco del IoT.

RBZ tiene una gran experiencia en este sector con el desarrollo de un proyecto para el control del alumbrado público que esta desplegado y en funcionamiento desde hace varios años en algunos puntos de España. Este proyecto se basa en el despliegue de una red LoRa para la conectividad, control y gestión de estructuras de hasta 1000 nodos. A pesar de las grandes ventajas que propone una solución como LoRa, también presenta desventajas como principalmente es el despliegue de una red privada. Este despliegue conlleva varios problemas logísticos en varias fases de la vida del proyecto desde la posición de las antenas de cada nodo en los municipios hasta el mantenimiento, gestión y estabilidad del funcionamiento de la red y su salida a Internet.

El sector está evolucionando a nuevas soluciones basadas en la utilización de redes públicas como son NB-IoT o LTE-M. Estas redes están desplegadas por las grandes operadoras de telecomunicaciones del país y permiten abstraerse de todos los problemas antes mencionados que tiene una red privada. Estas redes están operativas y en funcionamiento en casi todo el territorio nacional y permiten acelerar los procesos de desarrollo y mantenimiento de los proyectos considerablemente.

1.2.1 DISEÑO ARQUITECTURA IOT

Como se ha mencionado, el sector está evolucionando al uso de redes alternativas. Por ello este Trabajo de Fin de Máster se centra en la investigación y desarrollo de un sistema de control de alumbrado público basado en estas nuevas redes. Con el objetivo de desarrollar una solución comercial el enfoque no fue tan solo una migración de los proyectos basados en LoRa, sino el desarrollo de un nuevo sistema completo.

Por ello se recorrieron todas las fases del proyecto, desde la búsqueda de microcontroladores que permitieran el acondicionamiento a estas redes hasta el desarrollo y validación de los resultados del sistema. Este TFM sigue la metodología de otros proyectos dentro de la empresa enfocados al desarrollo de software embebido aunque el proyecto ya no esté incluido en esta categoría.

El objetivo del TFM concierne al diseño y desarrollo de una arquitectura completa de IoT. Este horizonte no busca solo incluir en el porfolio de la empresa nuevas tecnologías para la conexión de los nodos en una red IoT, sino poder ofertar el sistema completo con un desarrollo extremo a extremo. Con desarrollo extremo a extremo se hace referencia al esquema general de una arquitectura IoT como se muestra en la Ilustración 2, donde todas las partes involucradas para el funcionamiento son diseñadas y desarrolladas.

Acometer el diseño de estas especificaciones presenta desafíos que otorgan un gran valor añadido al TFM, donde se presentan campos fuera del campo de conocimiento de desarrollo de la empresa. Junto con ellos, el diseño de la nueva arquitectura debe presentar modularidad en sus procesos, lo que se puede traducir como una arquitectura versátil. De esta manera, el día de mañana la arquitectura diseñada permitirá adaptarse a proyectos nuevos o distintos al control del alumbrado público.

Para poder ganar en esta versatilidad y modularidad de la arquitectura el TFM se define como un proyecto multidisciplinar, en el que diversos conceptos y campos tecnológicos distintos se comunican e interaccionan para cumplir las funcionalidades. Este aspecto multidisciplinar se fundamenta en que el proyecto no debe solo solucionar el problema de la gestión de la red de los nodos, sino, además, solucionar la gestión de la red completa de la arquitectura. Esta gestión completa hace referencia a aspectos como el procesamiento de la recopilación de datos, la persistencia de los datos recopilados, el control sobre las acciones a cada uno de los nodos y la visualización de los datos y acciones del sistema. Toda esta gestión se debe hacer con el objetivo de ser modular entre cada uno de los procesos, de tal manera que cada uno de los procesos presente un desacoplo tecnológico entre sí. Este desacoplo tecnológico permite que el cambio de tecnologías no afecte a toda la arquitectura, sino a módulos específicos.

En conclusión, los objetivos marcados en este TFM presentan desafíos y soluciones a un problema de diseño actual y recurrente e la industria aportando un gran valor en el desarrollo de arquitectura IoT.

1.3. ORGANIZACIÓN DEL DOCUMENTO

En el capítulo 2 se detalla la arquitectura del sistema. En ella se explican los distintos elementos de ésta, las tecnologías utilizadas y las decisiones tomadas durante el desarrollo. Se explican las interfaces de comunicación elegidas para comunicar los distintos elementos de la red y las características que presentan para fomentar la modularización e independencia de cada elemento.

Una vez descritos los grandes rasgos de la arquitectura en el capítulo 2.1, el documento se centra, en los siguientes capítulos, en cada uno de los elementos de la arquitectura, detallando los aspectos técnicos, las arquitecturas propias y las decisiones tomadas para el correcto funcionamiento.

Para poner en valor el sistema, el capítulo 3 presenta la validación de los resultados utilizando los propios elementos del diseño para mostrar gráficamente los resultados del sistema.

Por último, el capítulo 4 presenta las conclusiones y líneas futuras para esta línea de desarrollo.

2. ARQUITECTURA DEL SISTEMA

A lo largo del capítulo se procederá a explicar la arquitectura del sistema que se ha diseñado y desarrollado en este TFM. El capítulo se centrará en explicar primero los grandes bloques de la arquitectura, incluyendo los requisitos que debe cumplir, las soluciones tomadas y las características que aportan diferenciabilidad y, después, continuará detallando cada uno de los módulos que componen la arquitectura. En cada uno de los módulos se detallarán los requisitos impuestos al sistema y las decisiones tomadas para una correcta solución. También se explicarán las interfaces tecnológicas utilizadas para la comunicación entre los módulos y la capacidad de desacoplo tecnológico que tiene la arquitectura.

2.1. ARQUITECTURA

El diseño de una arquitectura funcional y eficiente es determinante en el desarrollo de un proyecto. Un buen diseño de la arquitectura del sistema protege al sistema contra problemas de difícil solución en las últimas fases del proyecto, como pueden ser la estabilidad o la escalabilidad. La metodología seguida para cumplir un buen diseño comienza con la imposición de unos requisitos adecuados. Por ello en este TFM se han considerado los siguientes aspectos como requisitos de una arquitectura IoT:

- Conectividad y comunicación. En los sistemas IoT es importante debido a que estos se caracterizan por formar un sistema distribuido donde cada uno de los elementos finales de la red o nodos se localiza en una posición del espacio, que es indiferente para el sistema. Por ello, es crucial que cada uno de los nodos o elementos de la arquitectura tenga un atributo de conectividad para mantener constante el flujo de información en la red.

- Gestión y control de dispositivos. La capacidad de controlar las comunicaciones con cada uno de los nodos de la red. Por ello, los dispositivos de la red deben poseer un identificador único y el sistema debe ser capaz de establecer canales de comunicación individualizados con cada uno de los dispositivos, ya sea para recopilar datos o para realizar acciones.

- Recolección, análisis y actuación de los datos. Las redes IoT se caracterizan por poseer un elevado número de dispositivos conectados entre sí. Por ello, la capacidad de recolección de datos es importante para mantener un rendimiento elevado durante toda la vida del sistema. El análisis y actuación de los datos obtenidos de la red es otro aspecto a destacar, debido a que el sistema tiene que procesar una cantidad elevada de datos y para ejecutar las correspondientes acciones a cada uno de los dispositivos conectados.

- Escalabilidad. Este es el factor más determinante de una red IoT. El concepto de IoT otorga más valor a la aplicación cuantos más dispositivos estén conectados al sistema. Por ello los sistemas diseñados deben ser capaces de adaptarse a una variación rápida del número de elementos conectados. Esta adaptación debe ejecutarse sin la perdida de prestaciones en el sistema, lo cual atañe a muchos parámetros como puede ser la capacidad de procesamiento o el tráfico que es capaz de gestionar la red.

Como se ha mencionado, existen varios tipos de redes de IoT existentes en el mercado que se clasifican como públicas y privadas. El objetivo de este TFM es explorar el uso de redes públicas para el control del sistema. Por ello, en la arquitectura de este TFM se utilizan las redes LTE-M o NB-IoT, las cuales ya están desplegadas por las grandes compañías telefónicas del país. El uso de estas redes garantiza la conectividad y comunicación de la red y, en parte, facilita la escalabilidad de la arquitectura, ya que la gestión del tráfico esta solventada.

Una vez solventado el problema de cómo gestionar la conectividad de los dispositivos de la red, aparecen otros problemas. Si nos fijamos en la Ilustración 3 existe una diferenciación en tres niveles; la capa de percepción, la capa de red y la capa de aplicación. El uso de las redes públicas NB-IoT o LTE-M permite solucionar la capa de percepción y parcialmente la capa de red. Para poder conectar la capa de aplicación, que en nuestro sistema es la capacidad de controlar el alumbrado público, y la capa de percepción debe definir más elementos de la red.

En la capa de red tenemos que definir las interfaces que comunican la recopilación de datos por parte de la capa de percepción con la capa de aplicación, que permite su visualización. Por ello conviene analizar protocolos de red para el envío y gestión de datos. En IoT existen dos tipos de protocolos muy utilizados para esta gestión como son: MQTT [19] y CoAP [20]. Los dos sistemas tienen sus ventajas y sus inconvenientes, pero este TFM se decantó por el uso del protocolo de red MQTT, por el hecho de haber sido utilizado con grandes resultados con anterioridad. Al decantarse por el uso de un protocolo de red conocido, los tiempos de desarrollo siempre disminuyen.

Como se explica más adelante, el protocolo de red MQTT genera una pasarela de comunicación. Por lo tanto, otro componente debe de ser incluido en la arquitectura para poder generar la interconexión entre la visualización de los datos y su recopilación. Este componente es la persistencia de datos, porque todos los datos procedentes de los nodos tienen que ser almacenados.

Las decisiones tomadas en los diseños de persistencia de datos tampoco pueden ser tomados a la ligera. Los sistemas IoT tienen que gestionar la recopilación de miles de dispositivos y es determinante que tras haber almacenado una gran cantidad de datos los tiempos de acceso para recuperarlos sean lo menor posible. Por ello, decidir entre el uso de bases de datos SQL [21] o NoSQL [22] , como explicaremos más adelante, es de vital importancia. Con la inclusión de una capa de persistencia de datos se puede dar por finalizada la abstracción de la capa de red y conectarla con la capa de aplicación.

Por último, queda el diseño de la capa de aplicación y qué objetivos se plantea. En este sistema la capa de aplicación busca una representación visual de los datos almacenados de la red. De tal forma, que un usuario sea capaz de visualizar todos los elementos conectados a la red y la visualización de los datos recopilados por días. Además, la capa de aplicación debe tener capacidades de control de la red, permitiendo al usuario configurar de manera individualizada un dispositivo de la red.

Estas decisiones tomadas se resumen en la Ilustración 5 donde se muestra un esquema de la arquitectura del sistema.

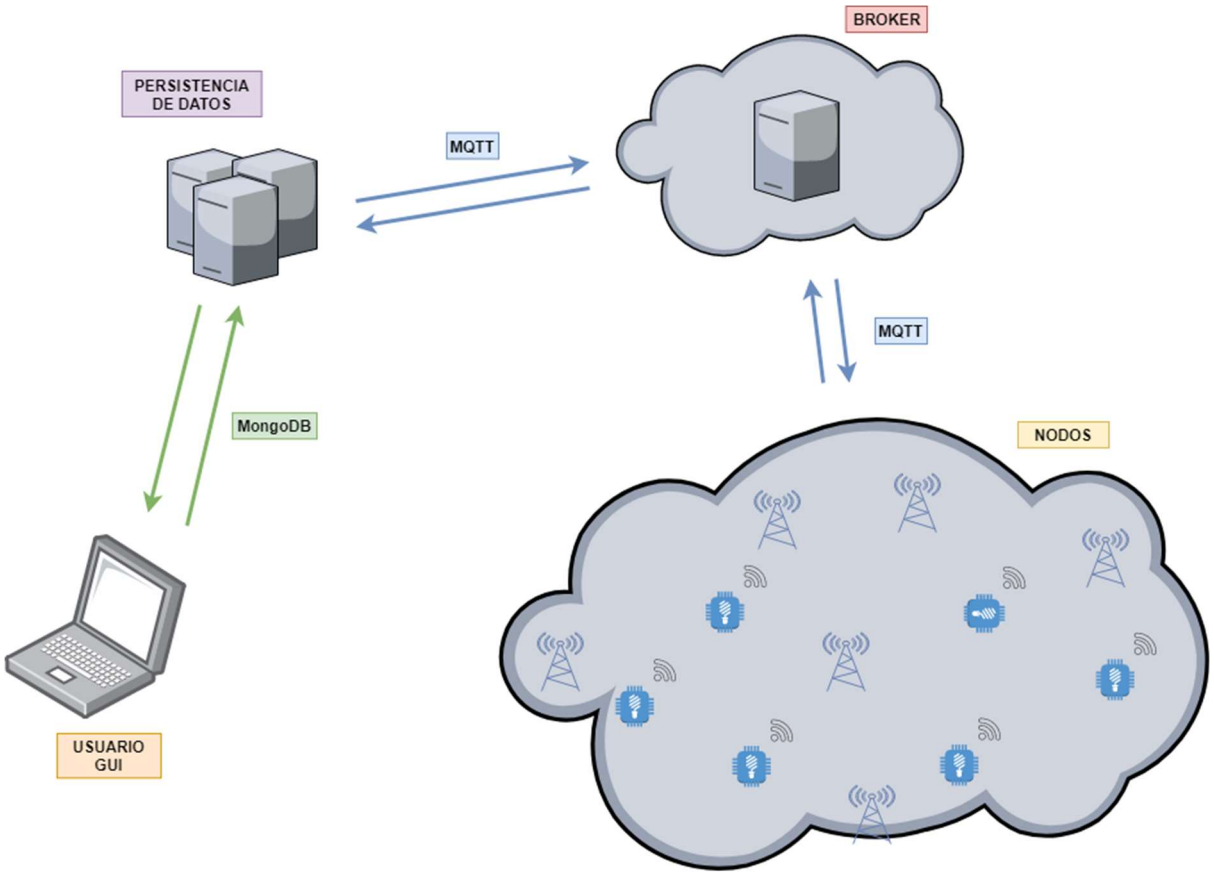

Ilustración 5 Arquitectura de la red.

Esta arquitectura contiene cuatro elementos bien diferenciados: nodos, Broker, persistencia de datos y la aplicación de usuario:

- Nodos. En el confluyen dos conceptos: los dispositivos de nuestra red y la red LTE-M que les otorga la conectividad. Estos nodos se comunican con el Broker, que es un elemento propio del protocolo de red MQTT. Este bróker permite realizar la tarea de recopilación de datos y, también, recibir nuevas configuraciones por parte de la aplicación de usuario.
- Broker. Este elemento es propio del protocolo de red MQTT. En capítulos posteriores se explicará con más detalle su funcionamiento, pero este es el encargado de gestionar el envío individualizado de configuraciones a cada uno de los nodos gracias a un sistema de tópicos.
- Capa de persistencia de datos. Esta capa tiene como tarea principal almacenar los datos recopilados de los nodos de manera estructurada. Pero también tiene otras tareas, esta capa sirve de nexo de comunicación entre la aplicación del usuario y el resto del sistema, de tal forma que debe de ser capaz de gestionar las peticiones procedentes de la aplicación de usuario y enviarlas al nodo correspondiente a través del Broker.
- Aplicación de usuario. Esta es la encargada de visualizar los datos recibidos de la red y de enviar las acciones a realizar a los nodos. Tiene como objetivo mostrar los datos recopilados de una forma sencilla.

Como se explicará en la siguiente sección este tipo de arquitectura presenta un valor añadido que convierte al sistema en robusto ante posibles cambios en el futuro.

Por último, la Ilustración 6 muestra todas las tecnologías implicadas en el desarrollo de esta arquitectura. Para ello, se utiliza el mismo esquema que en la Ilustración 5, pero en cada componente se incluye una estructura jerarquizada en la que se detalla: en verde el hardware utilizado, en rojo el sistema operativo que soporta el sistema y en amarillo la tecnología o lenguaje de programación que se utiliza para dar la funcionalidad al sistema.

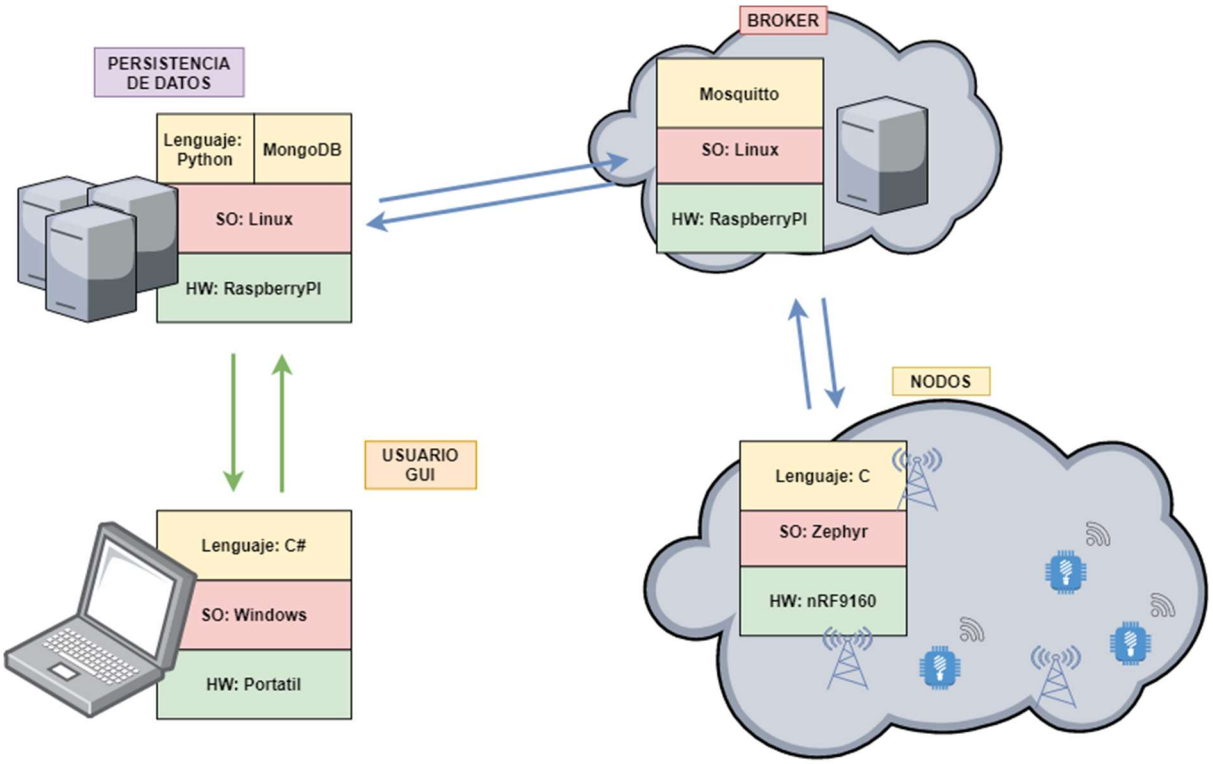

Ilustración 6 Tecnologías utilizadas en cada componente de la arquitectura.

2.1.1 MODULARIZACIÓN E INDEPENDENCIA DE CAPAS

Como se ha mencionado previamente esta estructura presenta una gran modularidad. Este atributo hace referencia a la independencia que tiene cada una de las estructuras del sistema con las demás. Si se devuelve la atención a la Ilustración 5 se distinguen 4 elementos, pero estos están conectados por dos interfaces de comunicación. Gracias a estas interfaces de comunicación y a la estructura del sistema logramos un objetivo muy ansiado en el diseño de arquitecturas: el desacoplo tecnológico.

Entre los nodos del sistema y la capa de persistencia de datos se utiliza el protocolo de red MQTT como interfaz de comunicación. Este protocolo es el que obliga a incluir el Broker para su funcionamiento. Con el uso del protocolo de red MQTT estamos haciendo independientes la capa de persistencia de datos de la capa de recolección de datos. Como representa la Ilustración 7, si en vez de utilizar el escenario del control del alumbrado público, el sistema gestionara la producción de un campo de cultivo, la arquitectura sería la misma. La capa de persistencia de datos continuaría recibiendo los datos por MQTT y los almacenaría en función del identificador de cada dispositivo.

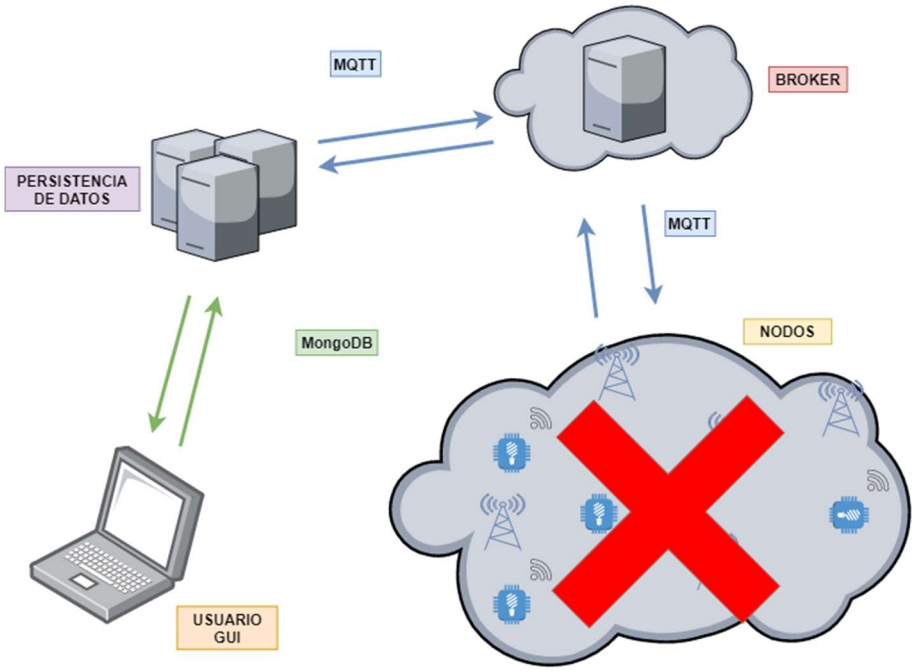

Ilustración 7 Desacoplo tecnológico nodos.

Continuando con la interfaz de comunicación entre la capa de persistencia de datos y los nodos, se observa otra ventaja de esta arquitectura. Gracias al sistema de almacenamiento de la arquitectura, basada en MongoDB y sus bases de datos NoSQL, el almacenamiento de los datos es indiferente al tipo de estos. Si la arquitectura utilizara otro tipo de sistemas de almacenamiento, en la capa de persistencia de datos se habrían diseñado una serie de tablas para estructurar los datos y así almacenarlos. Esto implica que, si en futuras actualizaciones de los datos recopilados por el sistema se añadieran nuevos parámetros, estas tablas tendrían que ser rediseñadas para almacenar los datos. Por ello, esta arquitectura es capaz de soportar actualizaciones en los dispositivos sin verse afectada la funcionalidad.

Por otro lado, existe la comunicación entre la capa visualización de datos y la capa de persistencia. Esta interfaz de comunicación también aporta un desacoplo tecnológico debido a que en ningún momento la aplicación de usuario depende de las tecnologías utilizadas para la comunicación como se muestra en la Ilustración 8. Esto beneficia a la arquitectura de tal forma que, si en un futuro el protocolo de red MQTT es actualizado a un nuevo protocolo de red, la arquitectura no debe de ser modificada y la capa de visualización es completamente reutilizable.

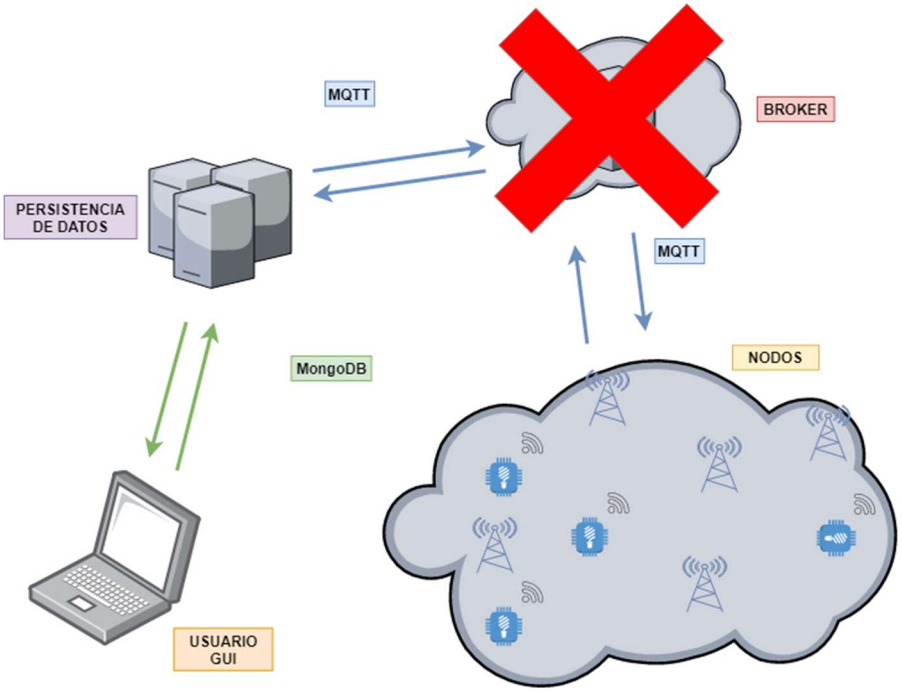

Ilustración 8 Desacoplo tecnológico MQTT.

La Ilustración 9 muestra que, si la capa de persistencia de datos es modificada y utiliza una nueva tecnología, los cambios en la arquitectura son menores. La comunicación con la aplicación de usuario se debería adaptar a esta nueva tecnología, pero la comunicación con los nodos se sigue basando en MQTT.

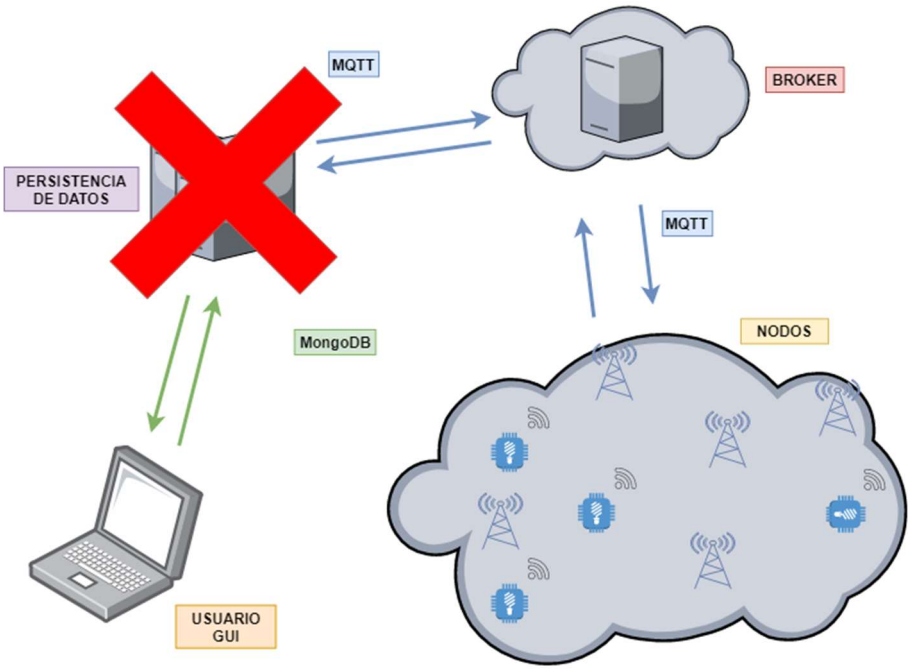

Ilustración 9 Desacoplo tecnológico persistencia.

Por último, queda el caso contemplado por la Ilustración 10. En éste se considera un rediseño completo de la aplicación de usuario. Como se puede contemplar, el resto de la arquitectura no se ve afectada, estando los procesos de recolección de datos y almacenamiento intactos tras este posible cambio.

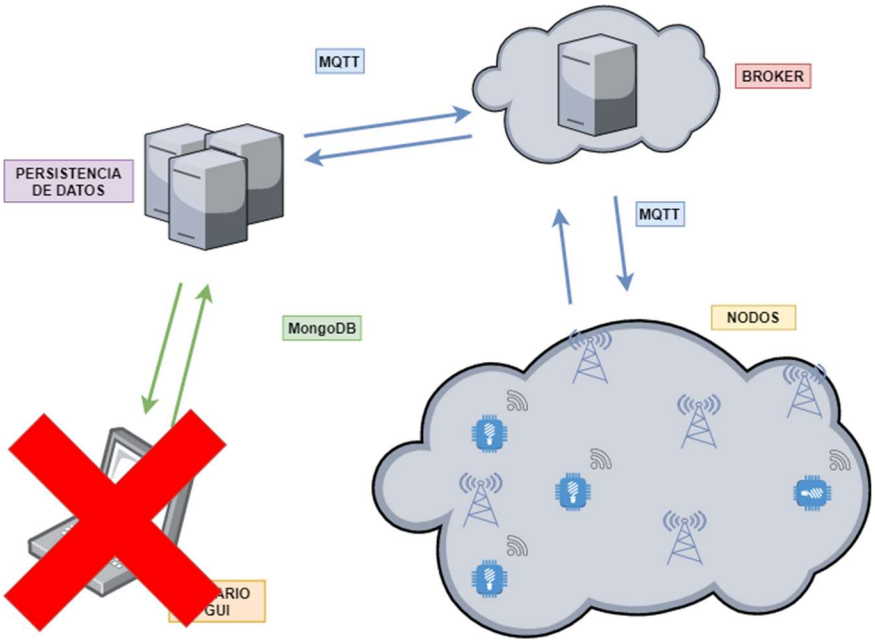

Ilustración 10 Desacoplo tecnológico visualización.

Con este último caso se demuestra que la arquitectura diseñada en el TFM cumple con el concepto de desacoplo tecnológico, el cual otorga una característica diferencial respecto a otras topologías.

2.2. NODO

Dentro de la arquitectura, el nodo es el elemento encargado de interactuar con el medio. Tiene como labores tanto encargarse de la recolección de los datos, como de ejecutar las acciones procedentes de la red. En este TFM, el nodo está encargado de gestionar el alumbrado público, reportando periódicamente el estado de iluminación y ejecutando la configuración recibida por la red.

Con el objetivo de controlar el alumbrado público los requisitos implementados en este nodo son los siguientes:

- Conectividad inalámbrica
- Persistencia de datos
- Ejecución en tiempo real.

La conectividad inalámbrica es un requisito del sistema, primero debido a que el enfoque de este TFM es el uso de redes NB-IoT o LTE-M y, segundo, debido a que en el desarrollo del nodo se busca independencia. Esta independencia se debe entender en el concepto del mínimo despliegue posible de cableado. Por ello, gracias a la utilización de una conectividad inalámbrica, el único cableado necesario es el del sistema de alimentación del nodo, que estando encargado del control del alumbrado público este sistema ya está desplegado.

La persistencia de datos es otro requisito muy importante en el sistema. El sistema debe de ser capaz de guardar en memoria una serie de parámetros de configuración que en el caso de una repentina perdida de alimentación no se vean afectados tras el reinicio del sistema. Con este objetivo se busca reducir el tiempo de restauración del sistema tras un apagón, debido a que una vez la alimentación reinicie los nodos de la red, estos no deberán esperar el reenvío de una configuración por parte del sistema.

El requisito de la ejecución en tiempo real concierne al manejo de los datos enviados por el sistema. Éste debe de ser capaz de mantener una referencia horaria y emitir los datos con una periodicidad de un minuto. Estos requisitos se pueden definir en una serie de requisitos hardware que el sistema debe poseer, y están detallados en las siguientes subsecciones.

2.3.1 ELECCIÓN DEL MICROCONTROLADOR

El nodo es un pequeño sistema hardware en el que un microcontrolador ejecuta una aplicación. Este microcontrolador tendrá como objetivo realizar las tareas de recopilación de datos en tiempo real y ejecución de instrucciones recibidas mediante MQTT. Para cumplir estas tareas el microcontrolador debe de ser capaz de cumplir los requisitos anteriormente citados, los cuales son trasladables a unos requisitos hardware como se explican a continuación.

La conectividad inalámbrica en un sistema electrónico se resuelve con la utilización de un modem y de una antena especializados en la tecnología de red utilizada. En nuestro caso, las redes que se pueden utilizar son NB-IoT o LTE-M. Como ya se ha mencionado, estas dos redes son diferentes y, por ello, se debe buscar un modem capaz de soportar ambas o al menos una de ellas.

La persistencia de datos en un sistema electrónico es la capacidad de guardar la información en una memoria no volátil. Por suerte, la gran mayoría de microcontroladores poseen una memoria interna, que permite resolver este problema. Si la cantidad de datos que se deben almacenar es elevada se debe acudir a una solución complementaria a esta. En nuestro caso la tabla de configuración del sistema es pequeña (64B) por lo que no es necesario un sistema complementario.

La ejecución en tiempo real se resuelve utilizando relojes de alta precisión o teniendo acceso a sistemas de tiempo de referencia. En general los microcontroladores no poseen sistemas de relojes de alta precisión internamente, estos se deben incluir en el diseño final de una manera externa. En nuestro caso, se utilizará el sistema de tiempo de referencia que reportan las redes NB-IoT o LTE-M que permiten disponer de una referencia calibrada y ajustada del uso horario en el que esté el dispositivo.

Junto con estos requisitos, se incluyó una exigencia para acelerar los procesos de desarrollo, que es la búsqueda de un sistema ya desarrollado para eliminar los tiempos de diseño y fabricación hardware. Con esto se hace referencia a que se debían buscar sistemas comerciales de prototipado para desarrollar el sistema. En general los fabricantes de microcontroladores ponen a disposición de sus clientes de tarjetas electrónicas que permiten experimentar con todas las capacidades del microcontrolador, ya que utilizando uno de estos sistemas de prototipado se reducen los tiempos de desarrollo.

Hay varios fabricantes de microcontroladores y electrónica en el mercado, pero para buscar el microcontrolador de este TFM la búsqueda se centró en los fabricantes: STMicroelectronics [23] , NXP [24] y Nordic Semiconductor [25]. Se centró en estos fabricantes por el hecho de haber trabajo con ellos con anterioridad y haber obtenido buenos resultados. Los tres son grandes fabricantes y ocupan una gran cuota de mercado.

Otra característica de estos tres fabricantes es que los tres utilizan arquitecturas ARM [26] para sus diseños. ARM es actualmente el mayor diseñador de arquitecturas microcontrolador y microprocesador del mercado. Restringiendo la búsqueda, ésta solo se centró en microcontroladores

que utilizaran arquitecturas Cortex-M [27] . Estas arquitecturas están diseñadas para sistemas empotrados.

El fabricante de STMicroelectronics no dispone de ningún sistema que incluya un microcontrolador y un modem conjuntamente. En cambio, la solución que ofrecen es la utilización del kit de desarrollo AT &T IoT Starter Kit (LTE-M, STM32L4) [28]. Este kit de desarrollo incluye un microcontrolador de la familia L4 [29] que es un microcontrolador de bajo consumo basado en la arquitectura Cortex-M4 [30] junto con una tarjeta de expansión que incluye un modem LTE-M. Además, el kit incluye una tarjeta SIM [31] para poder conectarse a la red con una pequeña tarifa de datos limitada contratada. También incluye una antena para conectar a la tarjeta de expansión para aumentar el rango de alcance del kit.

NXP no posee ninguna solución que contenga un modem LTE-M o NB-IoT. En cambio, se podría haber seguido utilizando un microcontrolador de NXP para este TFM si se hubiera utilizado un modem de otro fabricante y se hubiera establecido una comunicación con este microcontrolador. Por ejemplo, se podría haber utilizado la tarjeta de expansión de ST como modem LTE-M y haber elegido un microcontrolador de NXP para el sistema. Esta solución se rechazó debido a que habría que intercomunicar dos sistemas distintos y habría supuesto un aumento innecesario en el tiempo de desarrollo.

Nordic Semiconductor ofrece una solución mucho más atractiva. El kit de desarrollo nRF9160 [32] incluye un SiP (System-in-Package) el cual no es solo un microcontrolador. Este SiP es la fusión de un microcontrolador y un modem en el mismo integrado. En las soluciones anteriores el sistema poseía un microcontrolador y un modem en dos integrados distintos. En electrónica se puede reducir el consumo de los sistemas si estos están todos dentro del mismo integrado. Esto es posible debido a que son más pequeños y están más cerca, lo que permite obtener los mismos resultados con unas corrientes menores.

Además, Nordic anuncia que el modem incluido dentro del SiP es capaz de gestionar el acceso a las dos redes NB-IoT y LTE-M por separado. Durante el periodo de desarrollo se demostró que la electrónica del modem es capaz de gestionar las dos redes, pero la versión de firmware disponible por el fabricante solo soportaba el acceso a la red LTE-M en el momento de la realización del TFM.

Otro aspecto muy interesante de este kit de desarrollo es la arquitectura del microcontrolador. Si antes se ha dicho que la búsqueda estaba enfocada a los Cortex-M, este microcontrolador incluye la última arquitectura diseñada por ARM, el Cortex-M33 [33]. El Cortex-M33 incluye casi las mismas características que un Cortex-M4, lo que lo posiciona como un micro potente, capaz de realizar operaciones de coma flotante, pero, además, enfocado en la seguridad. Esta seguridad es reforzada mediante el uso del TrustZone [34]- Este sistema permite añadir unos nuevos parámetros al bus de direcciones. Usualmente el bus de direcciones posee los atributos, de lectura, de escritura o de lectura y escritura. Ahora con el sistema TrustZone, el bus de direcciones incluye, además, los atributos seguro y no seguro. Para poder explicar estos nuevos atributos se debe entender que un microcontrolador cuando ejecuta el código procedente de un programa utiliza una serie de operaciones que cogen valores de dos posiciones de memoria, realiza la operación y la guarda en otra posición de memoria. También se debe conocer que esta operación está guardada también en una posición de memoria. Por ello, utilizando estos nuevos atributos, si una operación está guardada en una posición de memoria no segura, esta no podrá acceder a datos guardados en memoria en posiciones seguras. En cambio, si la operación está guardada en una posición segura sí podrá acceder a datos en posiciones seguras.

Algunas aplicaciones tipo del TrustZone son, por ejemplo, el arranque seguro, en el que un código del programa está guardado en una zona segura y analiza si el resto de la aplicación no ha sido corrompida antes de proceder a ejecutarla, asegurando que, si alguien corrompe la aplicación, ésta no puede acceder a las posiciones de memoria del arranque del programa y modificarlo.

El modem versátil en las dos redes de acceso LTE-M y NB-IoT del kit de desarrollo de Nordic fue determinante para elegirlo para el desarrollo de este TFM. También cabe mencionar que este

microcontrolador cumplía los otros requisitos, debido a que incluye una memoria flash interna de 1MB y un reloj de gran precisión de 64MHz.

2.3.2 ARQUITECTURA FIRMWARE

En este apartado se detalla la arquitectura del firmware diseñada para el nodo de la red. Se define arquitectura firmware como la forma de estructurar y comunicar los componentes software con el objetivo de cumplir los requisitos del sistema. Los requisitos software deben cumplir los del sistema antes definidos apoyándose en los requisitos hardware mencionados con anterioridad.

El primer requisito por decidir, es si el sistema software a desarrollar se construirá sobre un sistema en tiempo real [35] (RTOS por sus siglas en inglés), o por el contrario se hará sin ningún sistema operativo, desarrollado en Bare-Metal [36]. La tendencia en el desarrollo de software embebido en los últimos años es del uso de sistemas operativos en tiempo real, debido a las varias ventajas que se obtienen. Por poner unos ejemplos, los RTOS permiten hacer uso de la concurrencia en el software desarrollado, que es tener varias tareas ejecutándose en paralelo donde cada una está encargada de la resolución de un problema. Una tarea podría estar resolviendo la conexión a internet mientras otra tarea se encarga de encender o apagar el sistema de alumbrado público y para el ojo humano estas dos acciones estarían ocurriendo a la vez.

El segundo requisito es la gestión de la conectividad. El sistema debe de ser capaz de comunicarse por Internet a través del modem del kit de desarrollo. Por ello es necesario tanto una librería que permita controlar el acceso al modem como una librería que gestione la pila de protocolos TCP/IP [37]. Existen varias librerías Open Source [38] con licencias comerciales que se pueden incluir en el sistema. En este caso, Nordic Semiconductor aporta un SDK (Software Development Kit) que es un conjunto de librerías para controlar los componentes hardware del kit de desarrollo e incluye una librería de gestión de la pila de protocolos TCP/IP.

La persistencia de datos es otro requisito a cumplir por el sistema software. Para ello, se necesita un componente software que sea capaz de comunicarse con la memoria no volátil y guardar los cambios de configuración del sistema.

Y, por último, la fiabilidad del sistema. El software debe de ser capaz de funcionar minimizando el número de errores al máximo. Pero como el riesgo de error es teóricamente imposible eliminar, se deben implementar mecanismos que permitan al sistema recomponerse de los errores en el menor tiempo posible. El kit de desarrollo posee dentro del SiP un componente hardware conocido como Watchdog [39]. Este es un contador el cual debe de ser refrescado periódicamente o, en caso contrario, reiniciará el sistema. Los Watchdog son sistemas muy utilizados en la industria para proteger contra errores no detectados durante el desarrollo, que permiten mantener el equipo funcionando hasta que una actualización de software solvente el problema.

Tras detallar los requisitos software, se va a explicar el desarrollo de tareas para resolver cada uno de los problemas del sistema, la comunicación entre ellas y el diseño de controladores para el acceso al hardware del sistema.

El sistema operativo en tiempo real que se utiliza en este sistema es Zephyr [40]. Este sistema operativo es un proyecto Open Source de la Linux Fundation [18]. Nordic Semiconductor ha apostado por el uso de este sistema operativo en sus microcontroladores. Aunque también incluye otros sistemas operativos portables, solo se encarga de mantener la portabilidad de Zephyr en su SDK. Entonces para reducir los tiempos de desarrollo e incluir un sistema operativo con un mantenimiento a largo plazo se decidió utilizar Zephyr.

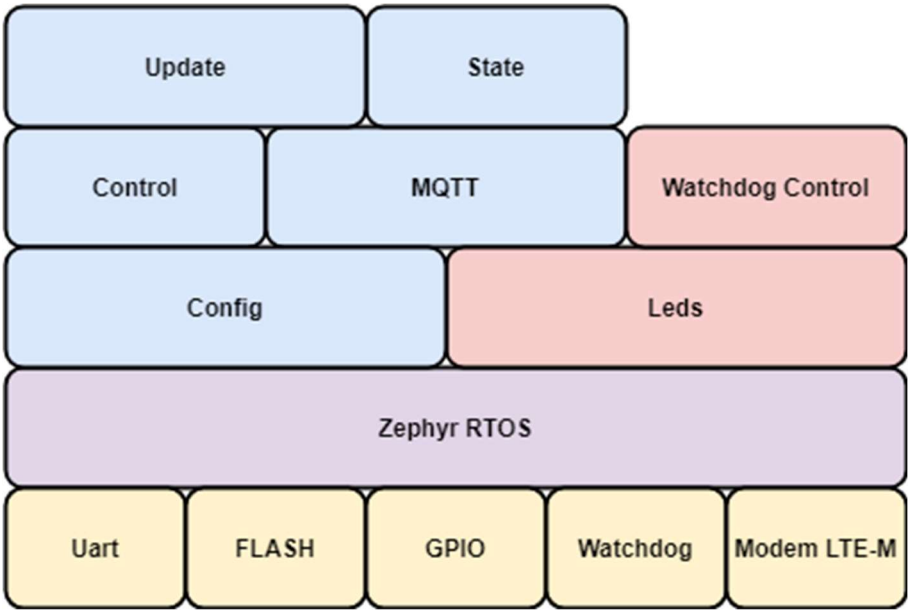

Ilustración 11 Arquitectura Software.

En la Ilustración 11 se representa la arquitectura del sistema. En color amarillo se encuentran los controladores hardware desarrollados. En morado se representa el sistema operativo Zephyr que controla la comunicación y gestión de las tareas representadas en color azul y rojo. En azul están las tareas encargadas de las funcionalidades imprescindibles del sistema y en rojo las tareas encargadas de la detección de los errores y de aportar estímulos visuales al usuario del estado del sistema.

Se diseñaron e implementaron cinco controladores hardware que tienen distintos propósitos. El controlador de la UART [41] tiene un propósito de mantenimiento del software y de comprobación del estado del sistema. Gracias a este controlador se pueden enviar trazas del sistema en tiempo de ejecución a un ordenador.

El controlador de la Flash es el encargado de la comunicación con la memoria no volátil. Tiene como objetivo detectar si hay errores al escribir en la memoria no volátil y comunicárselo a las capas superiores. Además, permite leer los datos escritos en la memoria. El controlador está organizado de tal manera, que las instancias de datos que se manejan tienen un identificador único. De esta forma el acceso a la lectura y escritura de esas instancias permite desconocer la posición de memoria en la que están guardadas. Por el contrario, cada vez que se quiera guardar una instancia nueva no contemplada antes, se debe crear un identificador único nuevo en tiempo de compilación.

El controlador de los GPIO [42] se encarga de gestionar los pines de salida del microcontrolador como salidas y les da un valor alto o bajo lógico. Este controlador es utilizado por la tarea de Control, pues es la encargada de habilitar el estado del sistema de alumbrado y por la tarea de los Leds, que gestiona 4 leds que posee el kit de desarrollo.

El controlador del Watchdog es el encargado de configurar este contador y de proporcionar las funciones para refrescarlo. El hardware del Watchdog permite configurar una vez el número de pulsos al comienzo del programa y después bloquea el sistema para que no se pueda modificar. En este sistema el Watchdog es configurado con un periodo de refresco de un segundo y se utiliza la tarea de Watchdog Control para refrescar el sistema con una frecuencia de 10Hz.

Por último, está el controlador del modem LTE-M. Este modem es capaz de gestionar el acceso a las redes NB-IoT y LTE-M. Durante el desarrollo se descubrió que Nordic Semiconductor aún no soporta esta dualidad en las redes y solo ofrece una versión firmware de acceso a la red LTE-M a través del modem. A parte de ello, el controlador es capaz de encender el modem y configurarlo para acceder a la red LTE-M mediante comandos AT [43]. Gracias a la versatilidad de los comandos AT, este

controlador también permite acceder al sistema horario que utiliza la red, permitiendo obtener una referencia absoluta de las medidas tomadas del sistema.

A continuación, se detallará el resto de la estructura de tareas del sistema. Se empezará detallando el funcionamiento de los componentes mostrados en la Ilustración 11, para después explicar cómo se comunican entre ellos.

Config

El primer componente es el encargado de la configuración del sistema, que en la Ilustración 11 se distingue como "Config". Este componente es el encargado de gestionar la tabla de configuración del sistema, la cual tiene los siguientes parámetros:

- Hora de inicio.
- Hora de apagado.
- Forzar apagado.
- Forzar encendido.
- Tiempo de vivo.
- Versión de Tópico.

La hora de inicio determina a partir de qué hora el alumbrado público tiene que estar encendido. La hora de apagado determina a partir de qué hora el sistema tiene que estar apagado. Forzar apagado, fuerza al sistema a estar apagado incluso cuando se ha superado la hora de inicio. Forzar encendido, hace lo contrario que forzar apagado y enciende el sistema incluso aunque se haya superado la hora de apagado. Entre los dos parámetros de forzar encendido y forzar apagado, si hubiera un error y los dos estuvieran activos, forzar apagado prevalece. El tiempo de vivo es un parámetro que se entenderá mejor cuando se explique el funcionamiento de la tarea de MQTT y en el apartado del Broker, pero es cada cuanto tiempo el sistema envía una señal recordando al Broker que está activo. Y la versión de tópico es un parámetro que permite organizar la recopilación de datos enviada por los nodos, se explica en la capa de persistencia de datos de la arquitectura.

El componente de Config, es una capa de abstracción entre la memoria no volátil y el resto de las tareas de la arquitectura. Este garantiza la exclusión mutua en estos recursos compartidos, gestionando que dos tareas no puedan acceder a la vez a un parámetro y así impidiendo las condiciones de carrera. Una condición de carrera ocurre cuando dos tareas simultáneamente intentan acceder a un recurso y una de ellas quiere escribir sobre él y otro leerlo. Sin un mecanismo de exclusión mutua la tarea que está leyendo el recurso no puede saber si está leyendo un recurso actualizado o no, dando lugar a posibles errores dependientes del uso de ese recurso. Además de gestionar la exclusión mutua, el componente de Config gestiona el acceso a la memoria no volátil, usando una copia de esta para su manipulación y cuando los cambios son aprobados guardándola de nuevo en la memoria no volátil.

Leds

El componente de los Leds tiene como objetivo realizar una tarea visual para el usuario, encargándose de la gestión de 4 leds. Para gestionar estos leds este componente es otra tarea que actúa de manera concurrente en el sistema, pero con una prioridad menor al resto. Los leds parpadean con un periodo de un segundo y dependiendo del número de leds que parpadeen significa un estado diferente. Estos son los diferentes estados:

- Conectando.
- Conectado.
- Error.

Desde la Ilustración 12 a la Ilustración 14 se muestran los leds asociados a cada uno de los estados. El estado de error bloquea el refresco del Watchdog y si se vuelve al estado de conectado se desbloquea el refresco del Watchdog.

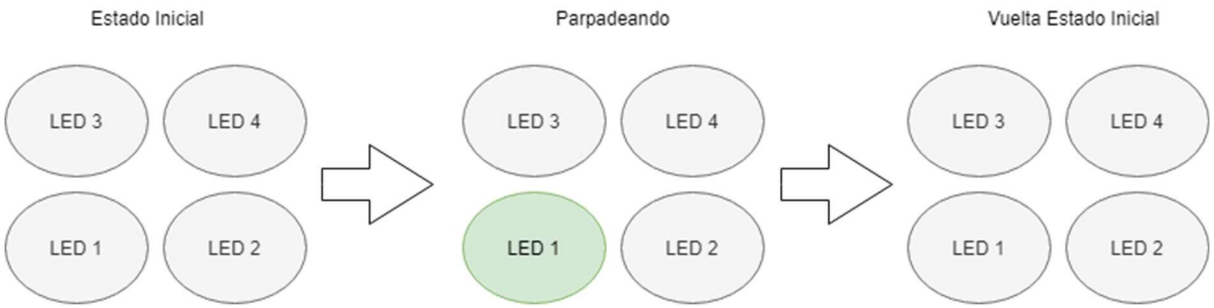

Ilustración 12 Leds estado conectandose.

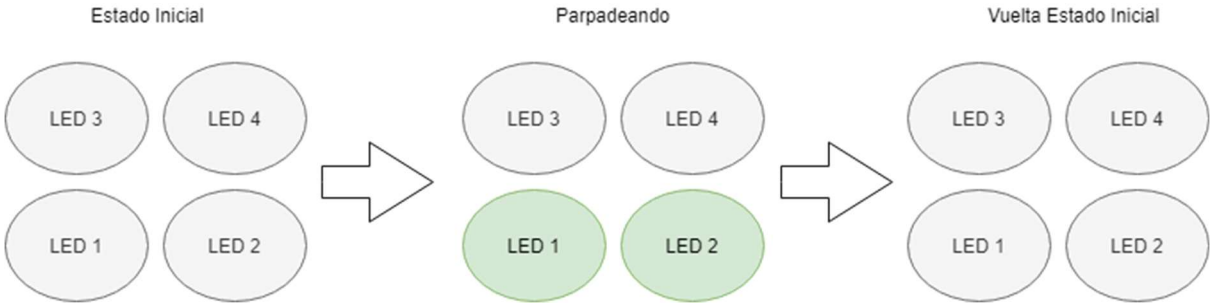

Ilustración 13 Leds estado conectado.

Ilustración 14 Leds estado error.

Control

La tarea de Control es la encargada de manipular el estado de iluminación del alumbrado público en función de los datos recogidos de la tabla de configuración y de la hora actual de la red. En la Ilustración 15 se muestra el funcionamiento de la tarea. El primer paso de la tarea es coger la información de la memoria compartida que gestiona el componente Config. Con los datos actualizados, la tarea sigue el orden descrito anteriormente de preferencia, primero forzar el apagado, después forzar el encendido y por último los limites horarios de hora de inicio y hora de apagado. Para comprobar la hora de inicio hace uso de una interfaz de comunicación con la tarea de MQTT, que se explicará después y reporta periódicamente la hora de la red actualizada. Con el dato de la hora comprueba los limites guardados en memoria. Estos tres condicionantes actualizan el estado de la iluminación que es trasladado en la salida GPIO de la iluminación. Posteriormente espera un minuto y vuelve a reiniciar el proceso.

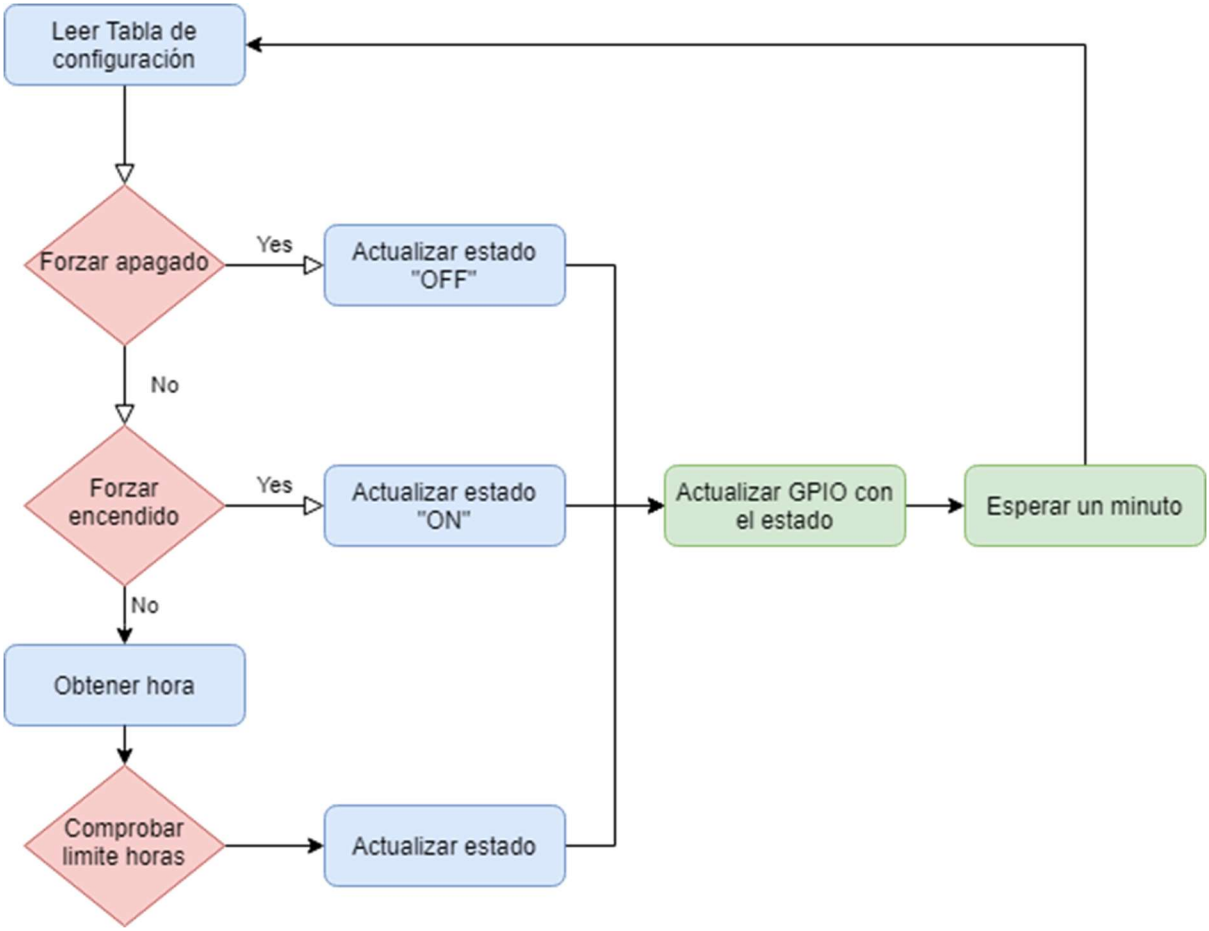

Ilustración 15 Tarea de Control.

MQTT

La tarea de MQTT es la tarea encargada de realizar la conectividad del sistema. Como se ha mencionado anteriormente, el sistema utiliza el protocolo MQTT para realizar la conexión con la red. Para entender esta tarea se explica brevemente el funcionamiento del protocolo MQTT. Una explicación detallada se realiza en la Sección 3.5 Broker. El protocolo MQTT se basa en el paradigma de publicación suscripción de mensajes sobre un tópico. Esto quiere decir que el nodo puede enviar dos tipos de peticiones al Broker: una de publicación de un mensaje a un tópico o suscribirse a un tópico y recibir todos los mensajes que le lleguen al Bróker.

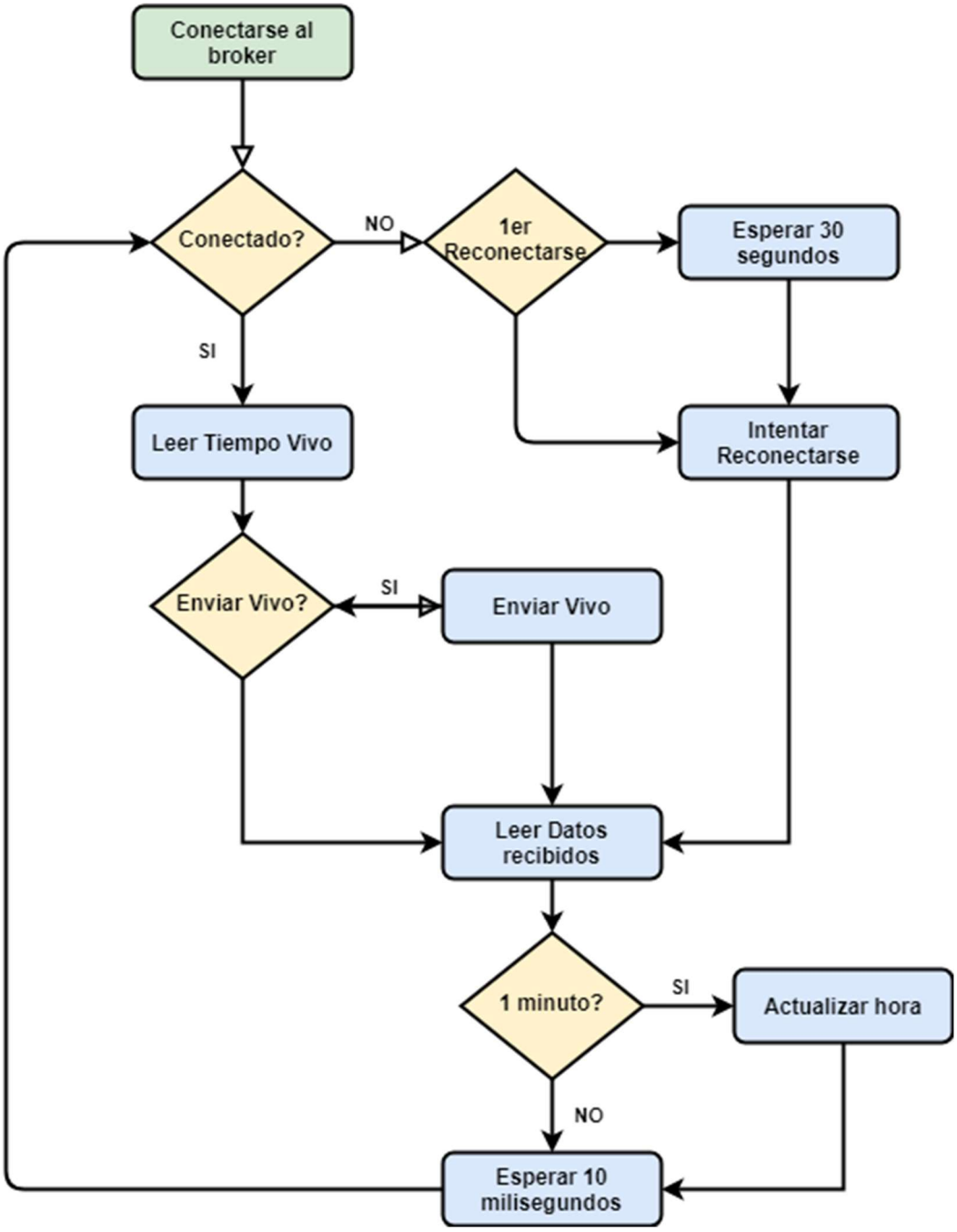

Ilustración 16 Tarea MQTT.

La Ilustración 16 muestra el funcionamiento de la tarea MQTT. La tarea comienza conectándose al Broker. Tras ello, entra en un bucle infinito en el que lo primero que contempla es si el nodo está conectado al Broker o ha sufrido una desconexión. Si sufre una desconexión intenta reconectarse una vez cada 30 segundos indefinidamente. Si en cambio, está conectado al Bróker, primero mira el tiempo de vivo de la memoria compartida y, si ha pasado el tiempo determinado desde el último envío, se envía uno nuevo. A partir de aquí los dos caminos se juntan y la tarea lee los mensajes procedentes del modem. Después comprueba si ha pasado 1 minuto y, en caso afirmativo, lee la hora actualizada de la red. Finalmente, espera 10 milisegundos y vuelve a comenzar. Esta tarea tiene un periodo tan pequeño debido a que las librerías de Nordic Semiconductor para el control de MQTT deben llamarse periódicamente para procesar los datos recibidos. Cuanto menor es el periodo, menor es el tiempo de respuesta del sistema.

Las librerías de Nordic Semiconductors para la gestión del stack de MQTT funcionan por devolución de llamadas, mediante funciones que se ejecutan tras un evento. Los eventos que se reciben son los siguientes:

- ACK Conectado
- Desconectado
- Publicación en tópico suscrito
- ACK mensaje publicado
- ACK temática suscrito

Las librerías permiten modificar estas funciones de devolución de llamadas de cada uno de los eventos. En este caso los eventos de "ACK Conectado" y "Desconectado" nos permiten actualizar la variable de control de conexión de la tarea decidiendo en la siguiente iteración porque rama continúa la tarea. El evento de "Publicación en temática suscrito" devuelve el mensaje que se ha enviado al tópico al que se está suscrito. El de "ACK mensaje publicado" avisa que el mensaje que se ha publicado es recibido en el Broker. Y el "ACK temática suscrito" nos avisa si nos hemos suscrito con éxito en la temática.

En el funcionamiento de la aplicación, cada vez que recibimos un evento de "ACK Conectado" el sistema realiza una petición a suscribirse al tópico de actualización de la tarjeta. Este tópico es identificado usando el IMEI [44] del nodo, que es un parámetro único procedente del modem para poder ser identificado en la red. Pero el mensaje que se incluye en el tópico es una nueva tabla de configuración para el nodo, donde están los parámetros antes mencionados. Estos nuevos parámetros son guardados en un buffer y la tarea que controla la actualización de los parámetros de la tabla es notificada.

Watchdog

En la Ilustración 17 se muestra el funcionamiento de la tarea de control del Watchdog. Esta comienza configurando el Watchdog con un periodo de refresco máximo de un segundo. Después entra en un bucle infinito en el que cada 100 ms comprueba si se impide refrescar el Watchdog desde otra parte del programa y si no, se refresca. Cuando el sistema entra en error, este se manifiesta visualmente con el estado de error de los leds y además se bloquea el Watchdog por si el error no se consigue solucionar. Una vez el estado de los leds pasa a conectado normal, el Watchdog se desbloquea.

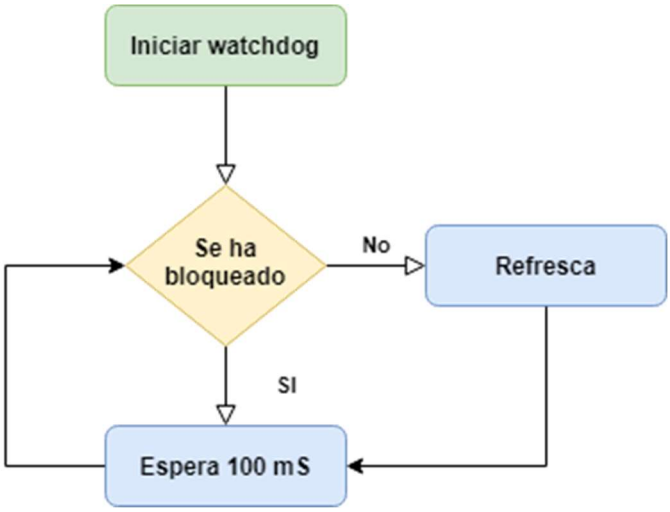

Ilustración 17 Tarea Watchdog.

State

En la Ilustración 19 se muestra el diagrama de funcionamiento de la tarea de reporte del estado. Esta tarea crea primero una instancia JSON [45] que es un formato de texto sencillo para el intercambio de datos. El uso de un formato JSON permite comunicarnos con cualquier tipo de sistema ya que es un formato estandarizado basado en caracteres alfanuméricos. Tras haber creado esta instancia, se comprueba si ha habido un error al crearla, esto es debido a que la librería utilizada para crear objetos JSON en el lenguaje de programación C, utiliza memoria dinámica. Tras comprobar este error se obtiene la hora actualizada de la red, se lee el estado del alumbrado y se encapsula en este JSON. Tras ello, se publica la instancia a través de MQTT, se borra el JSON y se espera un minuto hasta la siguiente iteración. Es necesario borrar la instancia creada debido a que las variables creadas dinámicamente en C no se borran automáticamente y es el diseñador software el encargado de gestionar su eliminación.

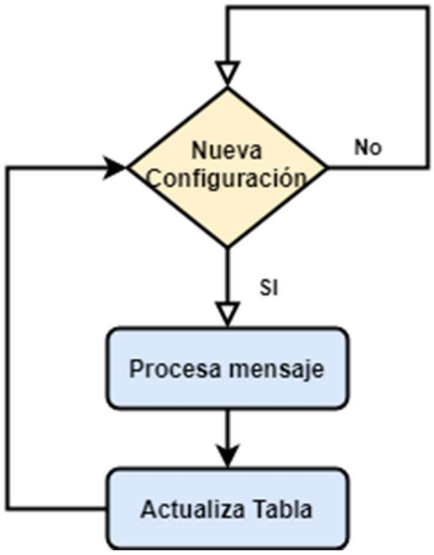

Ilustración 18 Tarea de actualización.

Update

El diagrama de funcionamiento de la tarea de actualización se muestra en la Ilustración 18. Esta tarea espera una comunicación de la función de devolución de llamada que se encarga de la gestión del evento “Publicación en tópico suscrito” de la tarea de MQTT. Cuando esta función de devolución de llamada se ejecuta, envía una notificación a esta tarea, la cual se encarga de procesar el mensaje recibido, ver si es correcto y posteriormente actualizar la tabla de configuración del sistema mediante el componente de config.

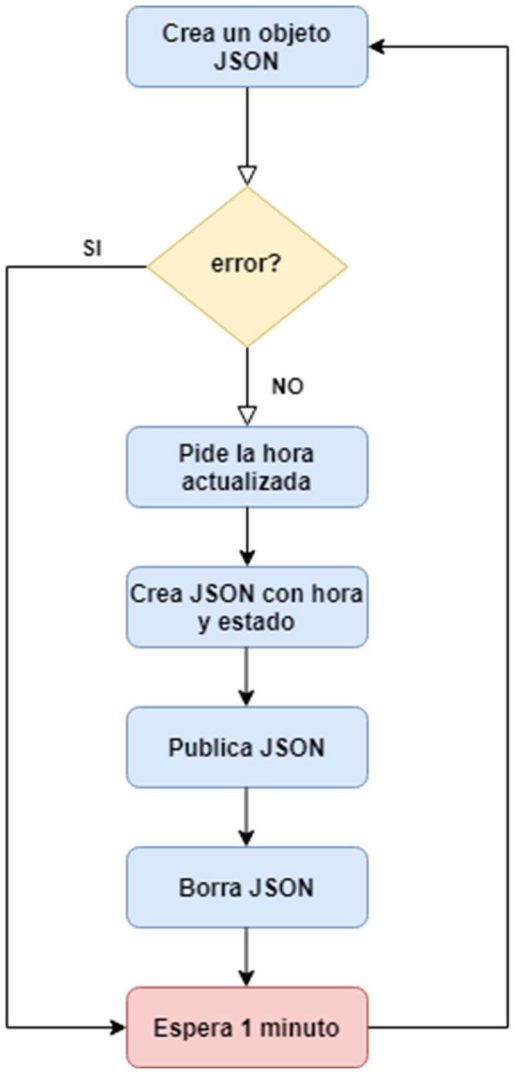

Ilustración 19 Tarea de reporte del estado.

La Ilustración 20 resume las comunicaciones entre las distintas tareas. La tarea de MQTT lee de la memoria compartida el valor del periodo del tiempo de vivo y entrega la hora actualizada a las tareas de reporte del estado y de actualización de la tabla de configuración. La Tarea de control lee de la memoria compartida la tabla de configuración para el control de la iluminación y envía el estado actual a la tarea de reporte del estado. La tarea de actualización recibe de la tarea de MQTT el nuevo mensaje de la temática de actualización y, posteriormente, actualiza la tabla de configuración de la memoria compartida. La tarea de los leds bloquea la tarea del Watchdog si un error ocurre en el sistema.

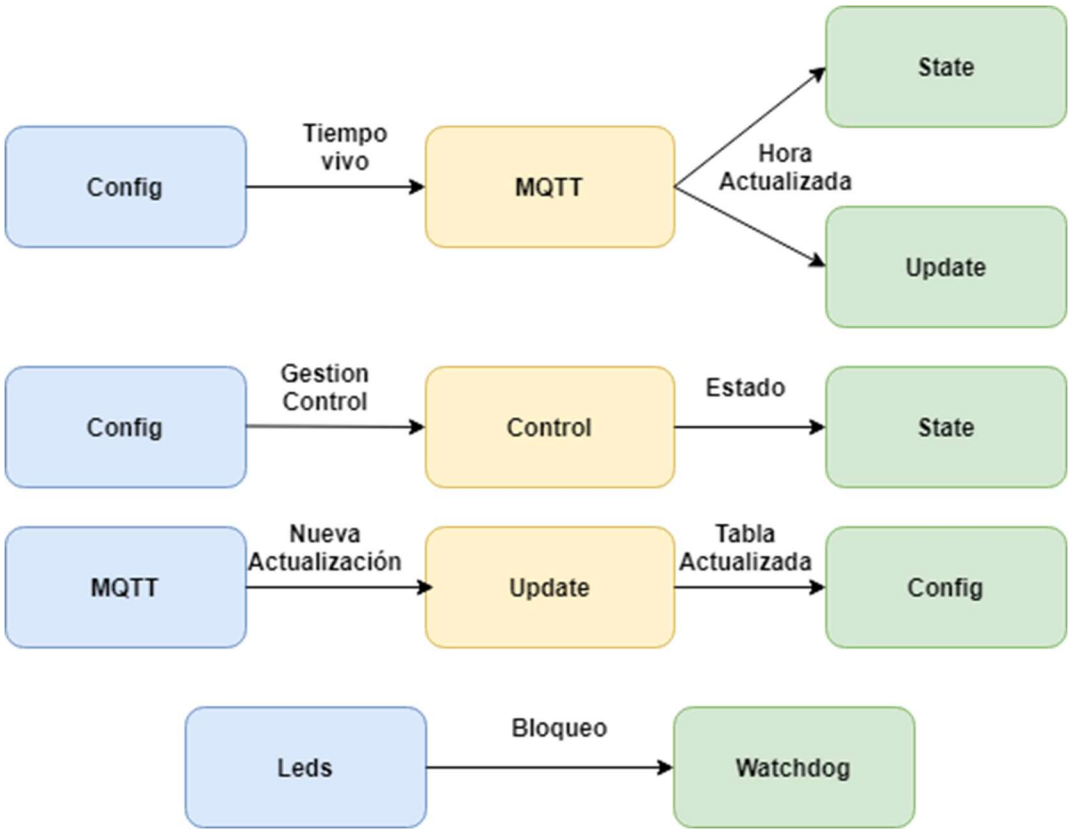

Ilustración 20 Comunicación entre tareas.

2.3. INTERFAZ MQTT Y BROKER

En la arquitectura de este TFM se utiliza el protocolo de red MQTT para la comunicación entre los nodos y la capa de persistencia de datos. En este apartado se explica este protocolo y el uso del bróker para implementar la red.

El protocolo MQTT proviene de las siglas de Message Queing Telemetry Control y es un protocolo de comunicación, máquina a máquina (M2M por sus siglas en inglés). Este protocolo fue creado en 1999 por el Dr. Andy Stanford-Clark de IBM [46] y Arlen Nipper de Arcom (ahora Eurotech) [47] como un mecanismo para monitorizar un oleoducto que circulaba a través del desierto .Es un protocolo que se ha convertido en unos de los pilares del IoT por su sencillez y ligereza, debido a que habitualmente los dispositivos IoT están limitados por la potencia de cómputo y el consumo que poseen. . El protocolo fue liberalizado en 2010 y en 2014 pasó a ser un estándar según OASIS [48], Organización para el avance de los estándares de información estructurada (por sus siglas en ingles).

Es un protocolo basado en la pila de protocolos TCP/IP como base para la comunicación. En el caso del MQTT se inicia una conexión y ésta es reutilizada para todas las transiciones de datos, no como otros protocolos que para cada transmisión utilizan una nueva conexión

En el protocolo MQTT se definen dos entidades en la red: el Broker y los clientes. Si utilizamos un paralelismo con cualquier otra tecnología web, el Broker es lo que se conoce como un servidor y el cliente como un cliente. Y la forma de comunicarse entre ellos está basada en un sistema de colas de mensajes basadas en el paradigma publicación-suscripción que se explica en el siguiente apartado.

2.4.1 PARADIGMA PUBLICACIÓN-SUSCRIPCIÓN

Para explicar el paradigma publicación-suscripción vamos a explicar primero el paradigma de comunicación más habitual en los protocolos de comunicación que es el Maestro-Esclavo o Cliente-Servidor. Las entidades que existen en este tipo de comunicaciones son un servidor y un cliente, en los cuales el servidor está siempre en espera de que el cliente inicie una conexión.

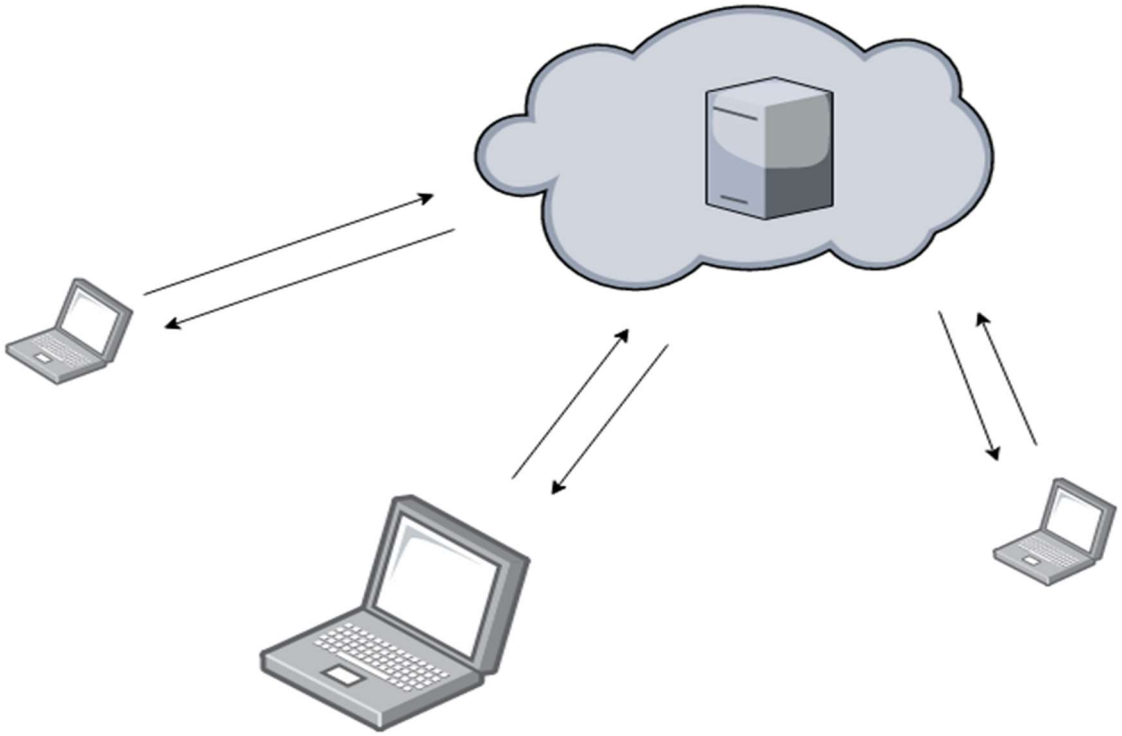

Ilustración 21 Arquitectura cliente servidor.

En la Ilustración 21 se muestra un ejemplo de este tipo de comunicación. En ella el servidor está esperando las peticiones del cliente y una vez realizada, éste devuelve la información requerida al cliente y solamente a él. Si otro cliente quiere la misma información debe hacer la misma petición.

MQTT no se basa en este tipo de arquitectura de comunicación, si no en el paradigma publicación suscripción. Para explicar este paradigma se detallan primero los conceptos que lo nutren y posteriormente su interacción. Estos conceptos son los siguientes:

- El Broker y el cliente. Son entidades de la red como ya se ha mencionado antes. Como paralelismo a la arquitectura Cliente-Servidor donde el servidor es el que posee la información y el cliente hace las peticiones, en este paradigma el cliente nutre al Broker de la información y con las acciones el Broker nutre a otros clientes de la información.
- Tópicos. Son las etiquetas que definen a las colas de mensajes. Estas se almacenan y se gestionan en el Broker pero su creación es por parte de los clientes. Lo tópicos pueden tener cualquier etiqueta, pero MQTT permite organizarlos jerárquicamente mediante el carácter “/”.
- Mensajes. Son cadenas de texto que envían los clientes a un tópico.
- Publicación. La acción de publicar la ejercen los clientes. Esta acción permite al cliente incluir un nuevo mensaje en una cola que gestiona el Broker
- Suscripción. La acción de suscribir también la ejerce el cliente. En este caso, el cliente le comunica al Broker que quiere recibir los mensajes de un tópico y el Broker cada vez que reciba un mensaje en ese tópico lo renviará a los clientes que estén suscritos a ese tópico.

Se ha comentado que los tópicos se pueden organizar jerárquicamente. Esto favorece la recopilación de los datos y su posterior procesamiento. En este TFM los tópicos se jerarquizan en función de la versión que se esté utilizando, del IMEI del nodo que es un parámetro único y la acción que realizan y siguen la siguiente estructura "RBZ/NODE4G/TOPIC_VERSION/IMEI/ACTION". Existen dos acciones "STATE" y "UPDATE". La primera se atribuye al tópico que recopila la información reportada periódicamente por cada nodo y la segunda se atribuye al tópico que incluye la tabla de configuración del nodo.

Con esta jerarquización de los tópicos se puede procesar la información individualmente de cada nodo. Pero si un cliente se suscribiera a un tópico con esta estructura "RBZ/NODE4G" se estaría suscribiendo a todas las combinaciones de tópicos existentes debajo, lo que sería en este caso todas las versiones de tópicos que existieran y a todos los nodos de la red.

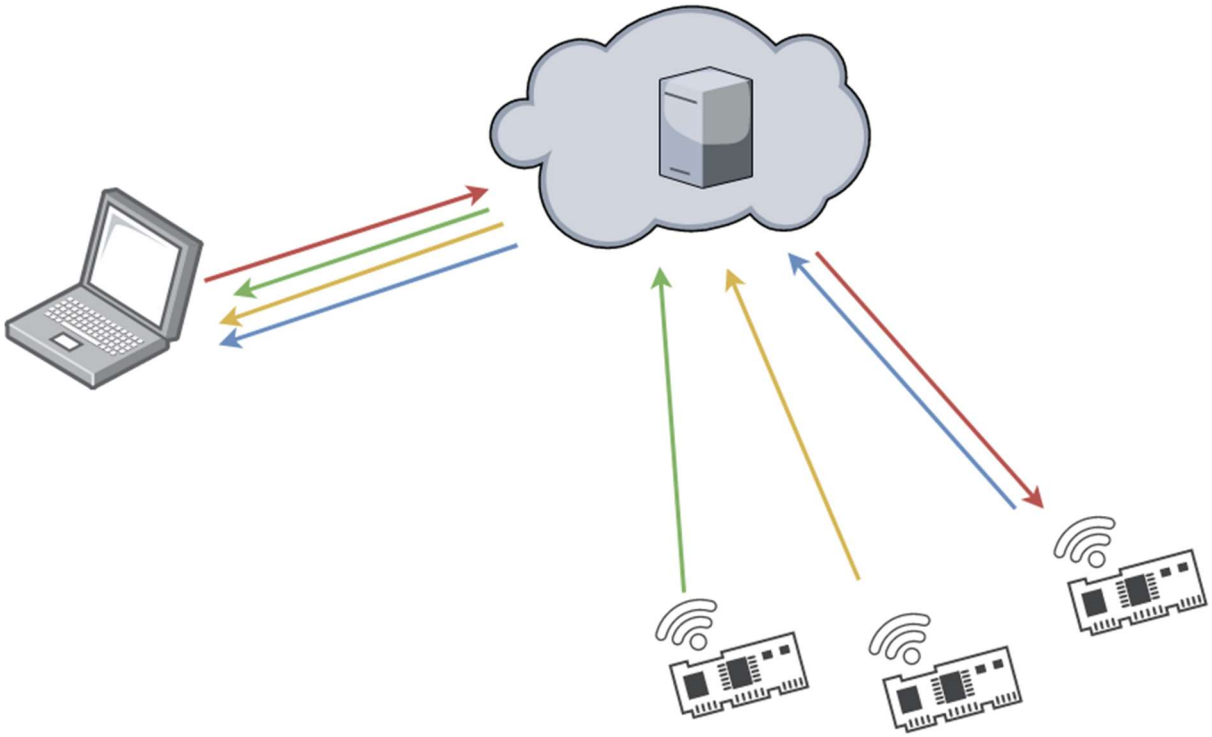

Ilustración 22 Arquitectura MQTT.

En la Ilustración 22 se muestra un ejemplo de la comunicación por el paradigma que se utiliza en este sistema. Los nodos, que se muestran abajo a la derecha, publican cada uno de ellos en un tópico donde reportan su estado y el usuario solo actualiza la tabla de configuración de un nodo. El usuario está suscrito a toda la jerarquía de tópicos y por eso recibe todos los estados de los nodos y actualiza solo un nodo.

2.4.2 BROKER

El Broker es el elemento de control en la red MQTT. Esté es el encargado de organizar las colas de mensajes entre los distintos clientes. Para ello los clientes inicializan una conexión TCP/IP con el Broker. Normalmente, se utilizan dos puertos distintos 1883 y 8883, el primero es si las conexiones no son seguras y el segundo es si las conexiones son seguras y van sobre TLS [49].

Para ello los clientes envían un mensaje CONNECT que contiene la información necesaria para iniciar una conexión. En el caso de este TFM el Broker está configurado para tener una capa de

autenticación [50], esto significa que son necesarios los parámetros de usuario y contraseña para iniciar la conexión. Esto se utiliza porque si no cualquier tipo de cliente podría comenzar una conexión con el Broker y suscribirse a la raíz de todos los tópicos y escuchar toda la información de la red.

Tras recibir el CONNECT si el cliente cumple todas las características para iniciar la conexión el Broker le devuelve un CONNACK como se muestra en la Ilustración 23.

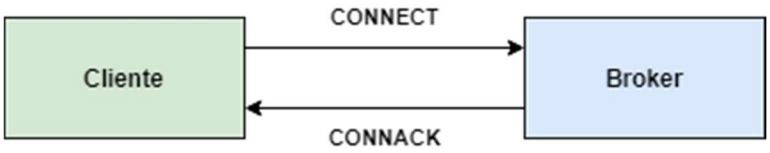

Ilustración 23 MQTT Conexión.

Cuando un cliente quiere suscribirse a un tópico envía un mensaje SUBSCRIBE y cuando recibe una respuesta del Broker con un mensaje SUBACK el cliente puede entender que la acción se ha realizado con éxito. Si, por el contrario, un cliente quiere borrarse de una lista de tópicos se utiliza el mensaje UNSUBSCRIBE. En la Ilustración 24 se muestra el flujo de información de la acción de suscripción.

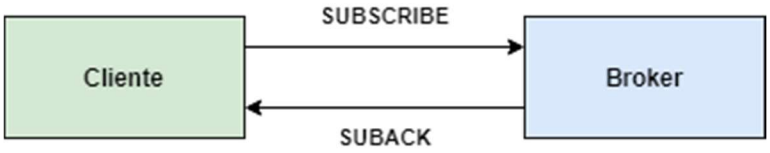

Ilustración 24 MQTT Suscripción.

En cambio, la acción de publicación que se muestra en la Ilustración 25 es más sencilla. El cliente publica el mensaje y no espera respuesta del Broker.

Ilustración 25 MQTT Publicación.

Para mantener viva la conexión entre el cliente y el Broker, los clientes envían periódicamente un mensaje PINGREQ y el Broker envía una respuesta PINGRESP. Si pasado un tiempo el Broker no recibe mensajes PINGREQ del cliente cancela la conexión enviando un mensaje DISCONNECT. Este mismo mensaje lo puede enviar el cliente si quiere cancelar la conexión en cualquier momento.

Existe otro concepto que es la calidad de servicio o QoS por sus siglas en inglés que se utiliza en MQTT. Cuando un cliente publica un mensaje a un tópico este puede elegir entre tres niveles de calidad de servicio: QoS 0, QoS 1 y QoS 2. El primero no garantiza que el mensaje haya llegado a los otros clientes suscritos a ese tópico, porque no realiza ningún mecanismo de reenvío. El segundo garantiza que el mensaje va a llegar al menos una vez a los clientes suscritos, esto puede ocasionar mensajes duplicados. Y el tercero, asegura que el mensaje llega a los clientes suscritos y solamente una vez.

2.4. PERSISTENCIA DE DATOS

La capa de persistencia de datos tiene dos objetivos: almacenar los datos recopilados de los nodos y procesar las peticiones procedentes de la capa de visualización de datos.

Estos dos objetivos tienen sus propios requisitos. Por una parte, el objetivo de almacenamiento de los datos busca organizar los datos de tal forma que los tiempos de acceso sean los menores posibles y si se produce algún cambio de los parámetros recopilados por los nodos estos no afecten a la estructura de almacenamiento.

Por otra parte, la capa de persistencia de datos debe imponer un acceso a los datos de manera restringida. También, debe ser capaz de gestionar que tipo de peticiones realiza la capa de visualización, con el fin de encapsular las acciones dirigidas a los nodos.

Para este Trabajo se decidió utilizar una Raspberry Pi 4 [51] como plataforma para esta capa de persistencia y en esta Raspberry se instaló el sistema operativo Ubuntu Server 18.04 [52]. Al utilizar el sistema operativo Ubuntu Server soluciona el problema de portabilidad de los sistemas desarrollados en esta capa. Esto es debido a que Ubuntu es una distribución de Linux y, por tanto, portable a la gran mayoría de plataformas hardware.

2.5.1 ELECCIÓN DE TECNOLOGÍA

La primera decisión que se debe tomar respecto a una tecnología de almacenamiento es si esta se basa en SQL o en NO-SQL. En el mercado existen muchas plataformas, pero todas se basan en uno o en otro modelo. Por poner en contexto, estos dos tipos de bases de datos son complementarias, debido a que las bases de datos SQL se basan en un modelo relacional de datos y las bases NO-SQL no. Por ello se analizaron los dos modelos para la elección de la tecnología.

El modelo relacional [53] es un modelo de datos basado en la lógica de predicados y en la teoría de conjuntos. Básicamente, la idea principal es el uso de relaciones entre datos.

Como ventajas de las bases de datos SQL se pueden citar:

- Madurez. Las tecnologías SQL fueron las primeras en aparecer y tienen una gran aceptación entre la comunidad de desarrolladores. Por lo tanto, existe una gran cantidad de información para realizar cualquier tipo de petición a las bases de datos, que se traduce en menor tiempo de desarrollo.
- Atomicidad. Tanto en las operaciones como en la información. Esto quiere decir que se garantiza que las operaciones se completan al 100%. Si algún error ocurre durante el proceso ningún cambio se guarda.
- Estándares bien definidos. La mayoría de las instrucciones se basan en el lenguaje SQL y por lo tanto la migración entre tecnologías es muy sencilla.
- Sencillez de sintaxis. Las peticiones a la base de datos son muy sencillas y parecidas al lenguaje humano, lo cual puede facilitar su mantenimiento por equipos de personas con menos experiencia en lenguajes de programación.

Como desventajas:

- Crecimiento. Cuando estas bases de datos tienden a crecer demasiado, el almacenamiento y mantenimiento es complicado y costoso, que se traduce en mayores tiempos de acceso a los datos.
- Cambios en la estructura. Si se producen cambios en la aplicación y el resultado no soporta el modelo relacional planteado, suele traducirse en el rediseño de la base de datos.

Como ventajas de las bases de datos No-SQL:

- Versatilidad. Es la principal ventaja que ofrecen estas bases de datos, debido a que la inclusión de nuevos parámetros en la base de datos no produce ningún contratiempo.
- Crecimiento horizontal. Estas bases de datos soportan una escalabilidad descentralizada, es decir la base de datos puede estar distribuida en varias máquinas paralelamente.
- Disponibilidad de recursos. Estas bases de datos no necesitan de grandes servidores para funcionar, por lo tanto, se pueden utilizar servidores más pequeños al principio y, posteriormente, adaptarlos a las necesidades del sistema.
- Optimización. Los sistemas NoSQL utilizan algoritmos internos para reducir el número de operaciones de la base de datos.

Como desventajas:

- Atomicidad. No todas las bases de datos NoSQL contienen esta característica.
- Documentación del software. Estos modelos de bases de datos no son tan maduros como las SQL y por lo tanto la documentación es menor.
- Estándares en los lenguajes. No se tiene un estándar que agrupe las instrucciones utilizadas en las distintas bases de datos.

Analizando las características de los dos modelos de bases de datos, la capa de persistencia de datos se diseñó en base a un modelo NoSQL. Esto es debido principalmente a la versatilidad de las redes y el crecimiento horizontal. La versatilidad es un gran factor debido a que la arquitectura del sistema debe de ser capaz de soportar cualquier tipo de entrada de datos IoT. Y el crecimiento horizontal es debido a las mejoras en tiempos de acceso que otorga esta tecnología cuando el número de datos es muy elevado, lo cual es muy típico en los sistemas IoT.

Por ello se decidió utilizar la tecnología MongoDB [54] que es ampliamente utilizada en los sistemas web.

2.5.2 ESTRUCTURA BASE DE DATOS

MongoDB organiza sus bases de datos proporcionando tres tipos de entidades: bases de datos, colecciones y documentos. Las bases de datos son la entidad más grande y está formada por colecciones. Las colecciones son agrupaciones de documentos y pueden poseer cualquier tipo de documento sin importar que sean diferentes unos de otros. Los documentos son las entradas de datos que se guardan en la base de datos. Estos documentos se basan en el formato JSON de clave valor, de tal forma que permiten a la base de datos realizar peticiones basadas en las claves.

Este TFM aprovecha este tipo de versatilidad en los datos para utilizar las colecciones relativas al identificador de cada nodo. Por lo tanto, todo el sistema guarda los datos en una misma base de datos y crea una colección de datos diferentes para cada dispositivo conectado al sistema. El nombre de la colección que se asocia a cada dispositivo es el IMEI del dispositivo. De esta forma todos los documentos relacionados con ese dispositivo están estructurados. Esto hace referencia a la Ilustración 5, donde se muestra como el Broker y la capa de aplicación se comunican con la capa de persistencia. Esto es debido a que la capa de aplicación cuando envía una actualización a un dispositivo está escribiendo en la colección correspondiente a ese dispositivo. En el apartado siguiente se explica el proceso que realiza la capa de persistencia de datos para procesar estas peticiones.

2.5.3 INTERFAZ COMUNICACIÓN BROKER Y GUI

La capa de persistencia de datos establece dos interfaces de comunicación, una con el Broker y otra con la capa de visualización. Estas dos interfaces se muestran en la Ilustración 5 donde se puede

comprobar que con el Broker se utiliza el protocolo de red MQTT y con la capa de visualización se utiliza directamente comunicaciones con MongoDB.

Para poder comunicar la capa de visualización con la base de datos se utilizaron los métodos de acceso que proporciona MongoDB. MongoDB proporciona librerías software en la mayoría de los lenguajes de programación más utilizados para permitir el acceso y la gestión de las bases de datos. Para poder utilizar estos accesos la base de datos tuvo que ser configurada. La configuración se centró en dos aspectos, el primero otorgar la capacidad de conectividad desde cualquier punto y el segundo restringir ese acceso para que solo nuestras peticiones sean aceptadas.

Configurar la base de datos para conseguir una conectividad local es trivial y desde los primeros pasos de la instalación ésta es estable. MongoDB tiene predefinida una conectividad local en el puerto 27017 sin ningún tipo de autenticación. Pero la conectividad no debe ser simplemente local, es necesario el acceso desde cualquier punto de la red a esta base de datos. Para ello es necesario obtener un dominio red que nos sirva como enlace para nuestras transacciones. Este dominio no se implementa en la base de datos MongoDB, sino que debe implementarse dentro de la plataforma Linux que aloja a la base de datos. Para ello se utilizó la herramienta No-IP [55] que permite el uso de tres dominios gratuitos con la condición de renovarlos mensualmente. La instalación de esta herramienta es sencilla y se utilizó el dominio “gafasatfmhostname.ddns.net”.

Una vez conseguido el dominio se deben reconducir las peticiones a ese dominio que interesen a la plataforma Linux. Para ello la plataforma debe tener siempre la misma dirección IP que permita identificarla unívocamente dentro de nuestra red local. Además, se tuvo que abrir los puertos del router para redirigir ese tráfico. Para aumentar la seguridad, se decidió no utilizar el puerto 27017 que utiliza por defecto MongoDB, sino que se utilizó el 40532.

Una vez configurada la base de datos, es posible acceder a ella desde cualquier punto de la red, por ello es interesante restringir ese acceso. Esto se consiguió implementando la autenticación en los accesos a la base de datos. MongoDB permite generar usuarios que tengan acceso a las bases de datos con diferentes roles. Por ello, se creó un usuario que permitiera gestionar la base de datos con permisos de lectura y escritura. De esta forma cualquier petición de acceso a la base de datos si no incluye este usuario y contraseña es rechazada.

Como se ha mencionado, la interfaz entre la capa de visualización y la base de datos se hace mediante las librerías software de MongoDB. En este TFM la capa de visualización está diseñada y desarrollada en el lenguaje de programación C# [56] y por tanto se utilizó MongoDB C#/.NET Driver [57]. Todavía queda por definir cómo se comunican el Broker con la base de datos, debido a que el Broker funciona por MQTT y la base de datos no posee ninguna librería software que convierta MongoDB a MQTT.

Esta interfaz de comunicación entre el Broker y la capa de persistencia de datos se realiza mediante el uso de dos procesos en la plataforma Linux. Estos procesos fueron diseñados y desarrollados en Python [58]. Un proceso está encargado de recibir los mensajes a través del protocolo MQTT y, si son válidos, insertarlos como documentos en la colección de la base de datos correspondiente. El otro proceso, está encargado de revisar la base de datos de manera periódica y comprobar si una nueva tabla de configuración se ha insertado y enviarla al dispositivo correcto a través de MQTT.

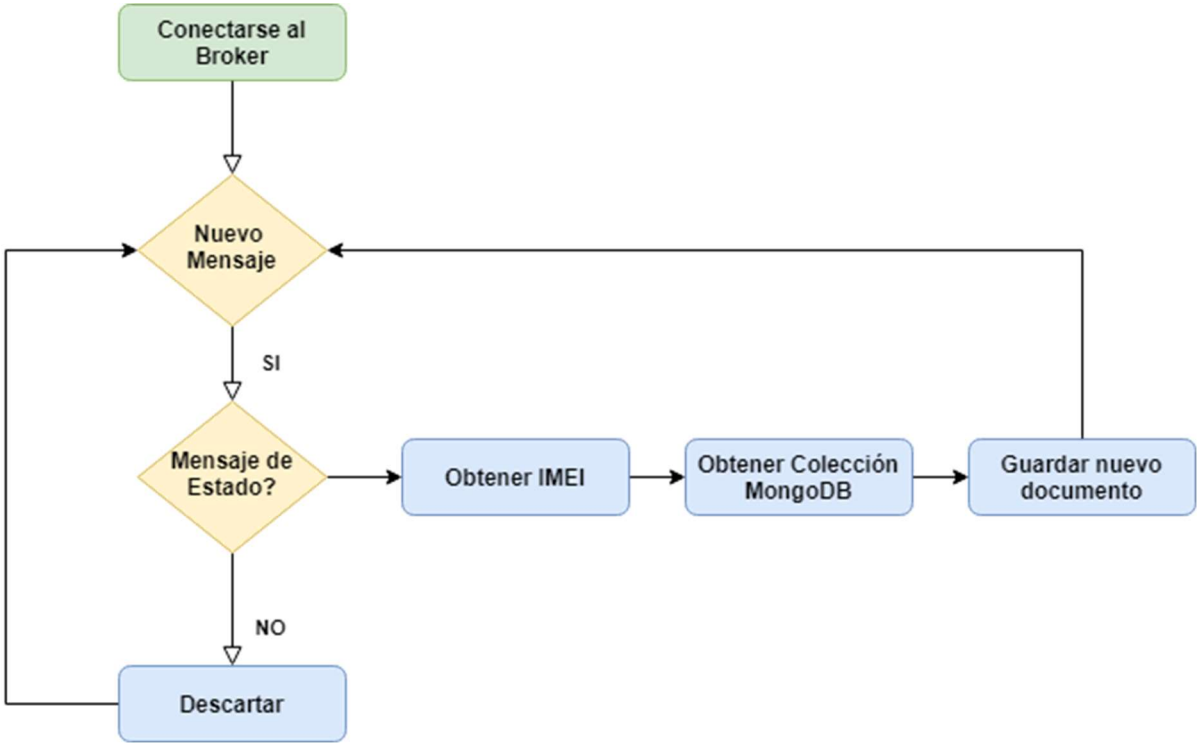

Ilustración 26 Proceso recopilar mensajes MQTT en la base de datos.

La Ilustración 26 nos muestra el primer proceso. Este proceso basado en Python se conecta el Broker mediante el uso de la librería Mosquitto [59] para Python. Tras conectarse al Broker el proceso se suscribe a los tópicos de nuestra red aprovechando el sistema jerárquico de estos. De esta forma el proceso se suscribe a todos los nodos de la red. Tras recibir un mensaje nuevo procedente de los tópicos, comprueba si éste está relacionado con la recopilación de los datos y, por lo tanto, con el estado del dispositivo. Tras ello, obtiene el IMEI identificador del dispositivo del tópico y accede a la colección correspondiente de la base de datos. Para el acceso a esta base de datos se utiliza la librería PyMongo [60]. Busca una colección con el mismo nombre del IMEI y, si no existe, la crea y guarda el mensaje recibido en esa colección como un nuevo documento.

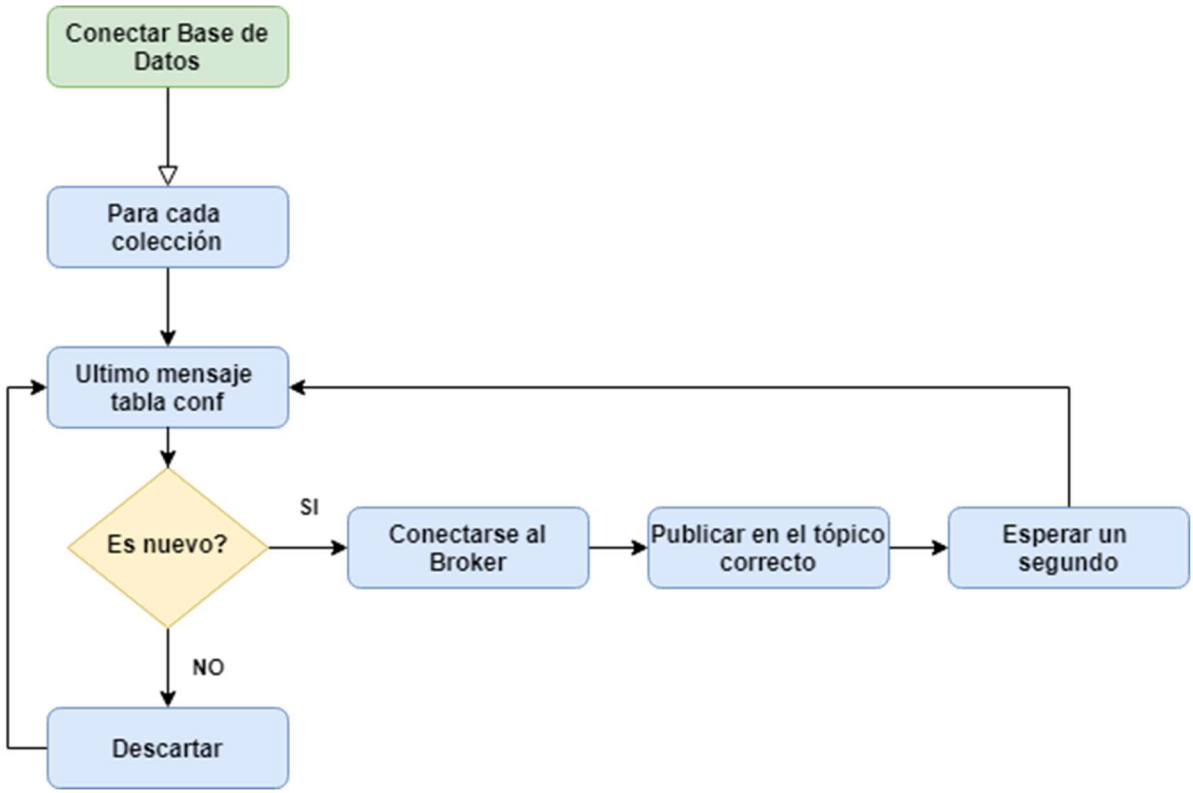

Ilustración 27 Proceso que revisa la base de datos y envíaa la tabla de actualización al correspondiente dispositivo.

La Ilustración 27 nos muestra el segundo proceso del sistema. Este proceso se dedica a acceder a cada una de las colecciones existentes en la base de datos con los nombres de los IMEI de los dispositivos buscando un nuevo documento guardado con una actualización de la tabla de configuración. Para ello utiliza una copia del último documento que se ha procesado. Si se encuentra un nuevo documento, el proceso se conecta con el Broker y publica en el tópico correcto el mensaje y vuelve a empezar.

2.5. VISUALIZACIÓN DE DATOS

La capa de visualización de datos es comúnmente denominada como la parte de la arquitectura enfocada al cliente. Con esto se hace referencia a que esta capa no solo debe mostrar los datos si no hacerlos intuitivos y abstraer al usuario de todo lo que conlleva el acceso y generación de esos datos. La aplicación de cliente debe ser rápida y eficaz, independientemente de que el proceso de adquisición de datos requiera el funcionamiento e interconexión de miles de procesos entre sí.

Esta aplicación de visualización de datos está desarrollada en el lenguaje de programación C#. Este lenguaje fue desarrollado y estandarizado por Microsoft [61] como parte de sus plataformas. Esta aplicación fue diseñada para ser de escritorio con un sistema operativo Windows [62] y ser utilizada en un ordenador personal sin especiales requisitos hardware. Por ello, el uso de C# como lenguaje de programación es un acierto. Este lenguaje está muy extendido entre la comunidad de desarrolladores y Microsoft desarrolla complejas herramientas basadas en él. Entre ellas están la herramienta Windows Presentation Foundation [63] , WPF. Esta herramienta es un marco de interfaz de usuario que crea aplicaciones de cliente de escritorio. La plataforma de desarrollo WPF permite utilizar las últimas novedades en control y gestión de recursos para el diseño de aplicaciones. Además, está basado en un patrón de diseño, que se explica en secciones posteriores y permite independizar el diseño gráfico de la aplicación de la lógica del programa. Esta independización es muy favorable porque permite la reutilización de las vistas gráficas diseñadas en otras plataformas o permite gestionar cambios en la lógica del programa sin que la interfaz gráfica se vea afectada.

La aplicación de cliente diseñada en este TFM tiene los siguientes objetivos:

- Listado de los dispositivos conectados a la red
- Visualización de los datos procedentes de un dispositivo
- Protección en el acceso a esos datos
- Ejecutar actualizaciones en los dispositivos.

Para cumplir estos objetivos la aplicación se basa en un modelo Dashboard [64]. Esto es un tipo de diseño de interfaz gráfica que permite observar los parámetros importantes de una sola mirada.

2.6.1 ARQUITECTURA GUI (PATRON MVVW)

La aplicación consta de cuatro vistas para representar distintas funcionalidades. Estas funcionalidades cumplen los objetivos marcados en el punto anterior. Por ello se puede ver en la Ilustración 28 la navegación entre las vistas.

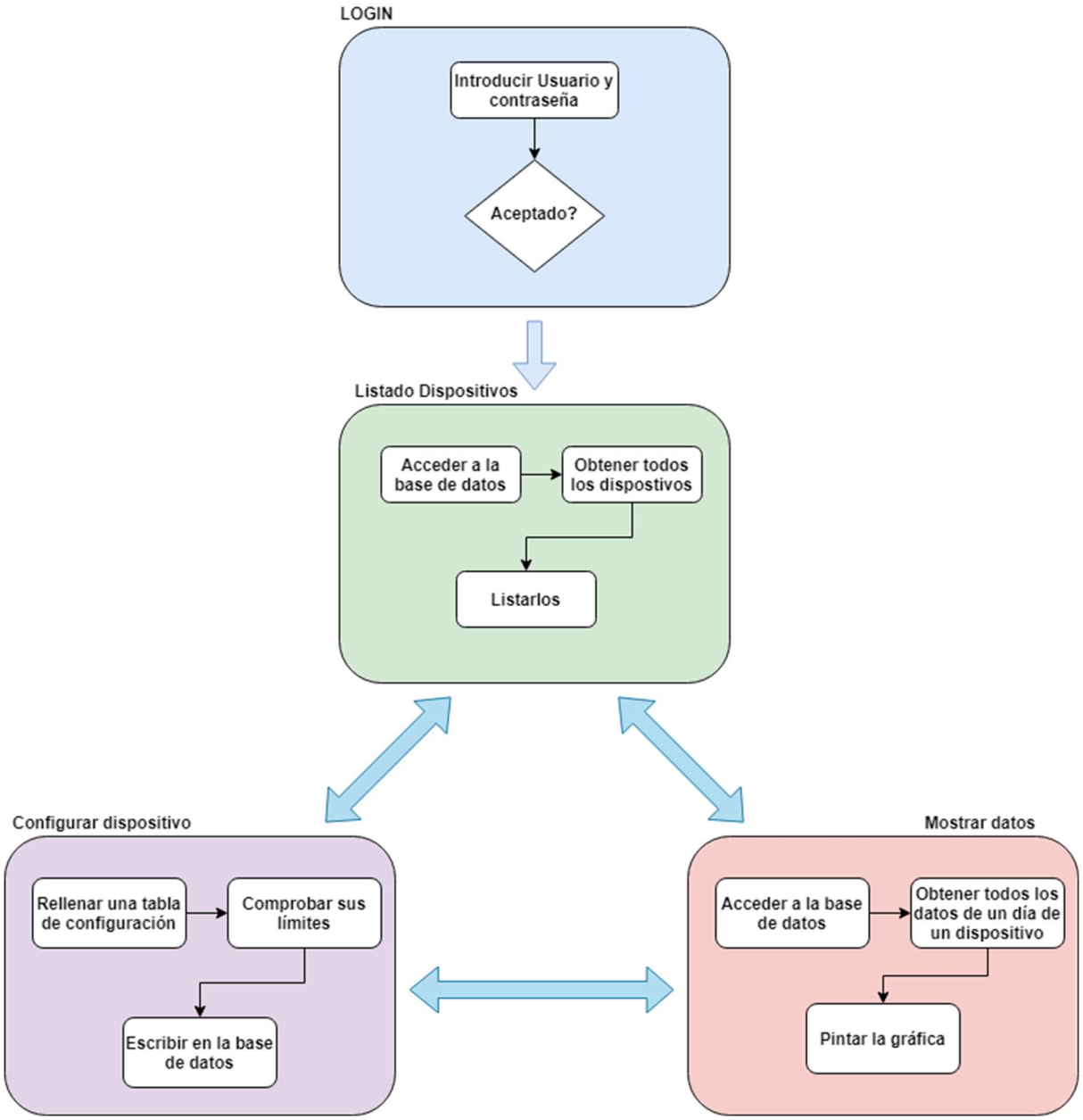

Ilustración 28 Navegación entre vistas de la aplicación de cliente.

El sistema comienza en la vista de “Login” en la cual el usuario debe introducir el usuario y contraseña de la base de datos para poder ser aceptado en la aplicación. Tras ello, la aplicación lleva a la vista del listado de los dispositivos o Dashboard. En ella, el sistema accede a todos los dispositivos conectados de la red y devuelve el último dato que ha devuelto cada uno. Tras ello se puede elegir a que vista ir, siendo las otras dos “configurar dispositivo” y “mostrar datos”. En “configurar dispositivos” se muestra una tabla con los parámetros de configuración de un dispositivo, que tras ser rellenados y antes de realizar la acción de envío, se comprueban como valores correctos. Por otro lado, está la vista de “mostrar datos”. En ella se pueden obtener los datos de un dispositivo en un día y el sistema los representa gráficamente para poder inspeccionar el funcionamiento.

Para poder realizar esta aplicación se ha seguido el patrón de diseño MVVM, que significa modelo, vista y modelo de vista (por sus siglas en inglés). El objetivo de este patrón de diseño es desacoplar lo máximo posible el diseño de la interfaz de usuario de la lógica del programa. En este patrón de diseño surgen los conceptos de modelo, vista y modelo de vista.

- Modelo. Representa la capa de datos. Un modelo contiene la información sobre lo que se está tratando, pero nunca las acciones o servicios que manipulan.
- Vista. La misión de la vista es representar la información gráficamente. En MVVM las vistas son dinámicas y poseen comportamientos y reacciones a eventos de la lógica del programa.
- Modelo de vista. Es un intermediario entre el modelo y la vista. Este contiene toda la lógica del programa y se comporta como una abstracción de la interfaz.

Utilizando este patrón de diseño se ha podido realizar una aplicación en la que las vistas son independientes de la lógica del programa. De tal forma que las peticiones a la base de datos no influyen en el funcionamiento de la interfaz gráfica de la aplicación.

En la Ilustración 29 se muestra la vista de Login de la aplicación. Es una vista muy sencilla que incluye una animación donde el rectángulo blanco entra desde la parte derecha de la pantalla. Esta vista permite introducir el usuario y contraseña. Como se puede comprobar, la contraseña no es visible desde la interfaz y, además, no es accesible por el resto de la aplicación, pues esta es directamente encapsulada en la petición a la base de datos.

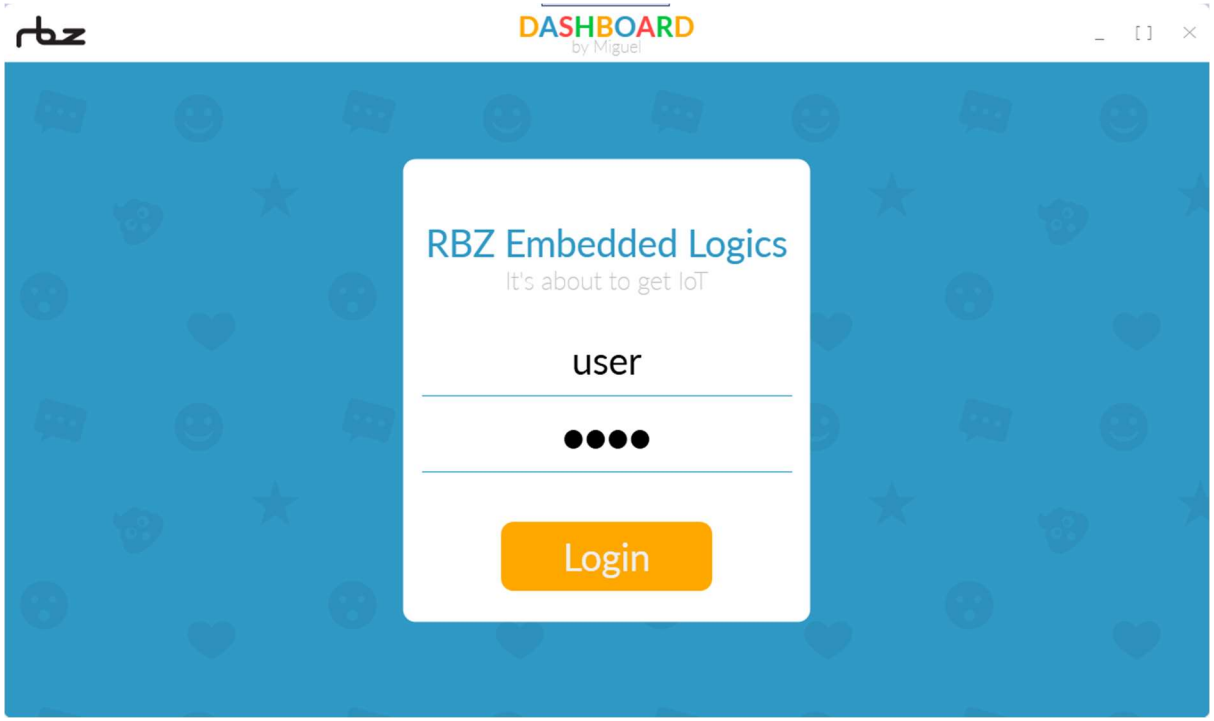

Ilustración 29 Vista de login.

Como se ha comentado anteriormente, de esta vista se salta a la vista de listado de dispositivos que se muestra en la Ilustración 30. En esta vista se muestra un esquema repetitivo en el resto de las vistas. A la izquierda aparecen tres botones: “Devices”, “Settings” y “Data”. El primer botón sirve para acceder a esta misma vista. El segundo sirve para acceder a la vista de configuración de un dispositivo. Y la tercera, sirve para visualizar los datos de un dispositivo durante un día. La parte de la derecha es la especializada en esta vista. En ella se muestra un listado de los dispositivos conectados a la red en formato Dashboard, con toda la información relevante a primera vista. En este caso se muestran tres datos: el IMEI, el estado actual del dispositivo y la hora de ese estado actual del dispositivo. La franja roja que aparece a la izquierda es debido al estado “OFF” del dispositivo, siendo verde si el estado es “ON”.

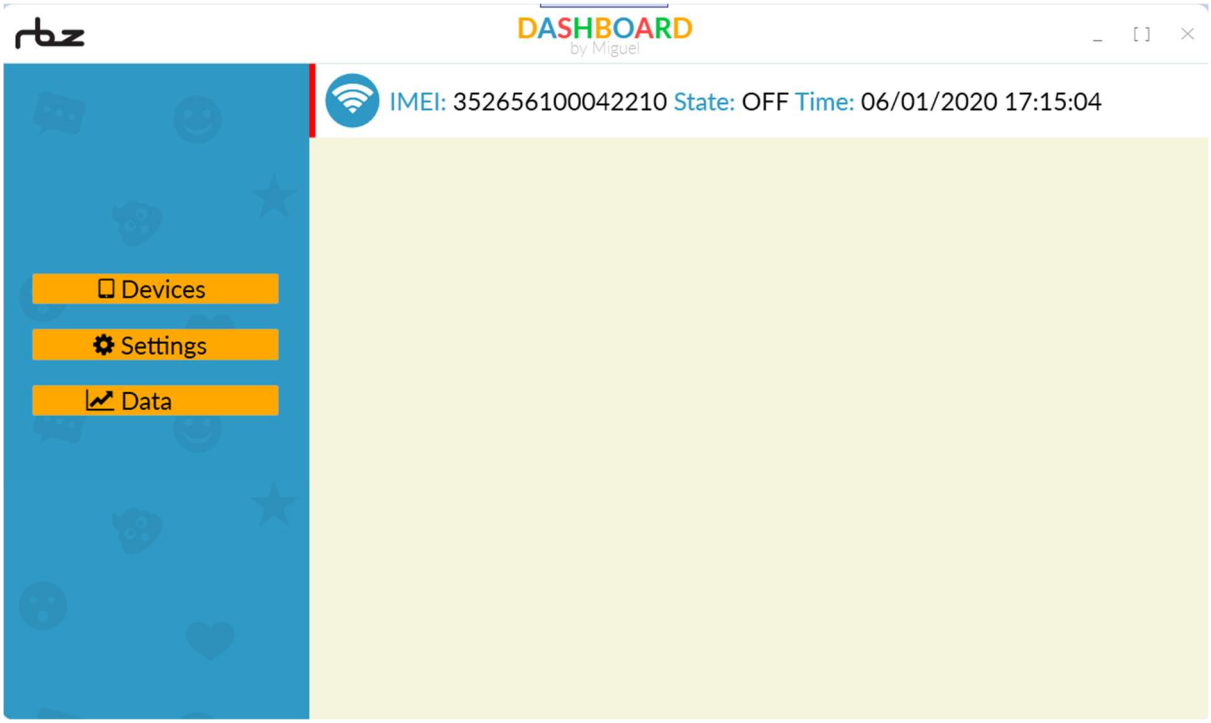

Ilustración 30 Dashboard de los dispositivos.

En la Ilustración 31 se muestra la vista que permite configurar un dispositivo. Como en la vista anterior, a la izquierda se muestran los botones de navegación entre las distintas vistas. En cambio, la parte de la derecha permite configurar un dispositivo. En primer lugar, se elige el dispositivo seleccionando su IMEI y después se configuran el resto de los parámetros. Estos son los citados en anteriores capítulos: la hora de inicio, la hora de apagado, forzar apagado, forzar encendido y el tiempo de vivo. Tras su configuración, el botón de “Send” se encarga de escribir esta nueva orden en la colección pertinente de la base de datos, donde los procesos de configuración la detectan y envían la configuración al dispositivo.

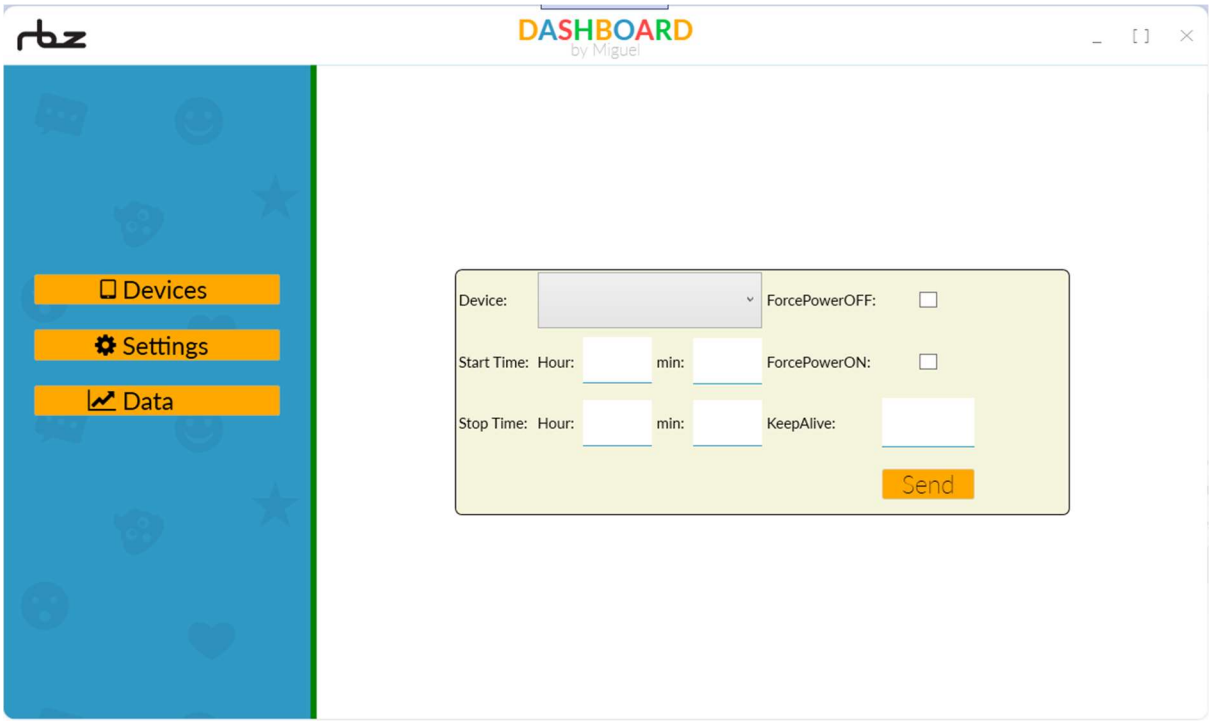

Ilustración 31 configuración de dispositivo.

En la Ilustración 32 se muestra la vista encargada de la visualización de los datos. En la parte superior se puede elegir de que dispositivo se obtienen los datos y el día. Tras ello el botón “Update” realiza las peticiones a la base de datos y rellena los campos inferiores. Estos campos son dos: una tabla y una gráfica.

La gráfica representa los datos obtenidos de la base de datos. En el eje de abscisas están representadas las horas del día y en el eje de ordenadas están representados los dos valores posibles del dispositivo “ON” y “OFF” como 1 y 0.

La tabla representa otra información distinta. En ella se muestran todas las tablas de configuración que ha recibido el dispositivo en el día que se han solicitado los datos. Si en cambio, no se registra ninguna tabla de configuración en el día solicitado, se muestra la última tabla de configuración que ha recibido el dispositivo. En el caso de esta Ilustración la fecha de los datos es del 28 de mayo de 2020, pero la última actualización de la tabla de configuración es del 12 de mayo de 2020.

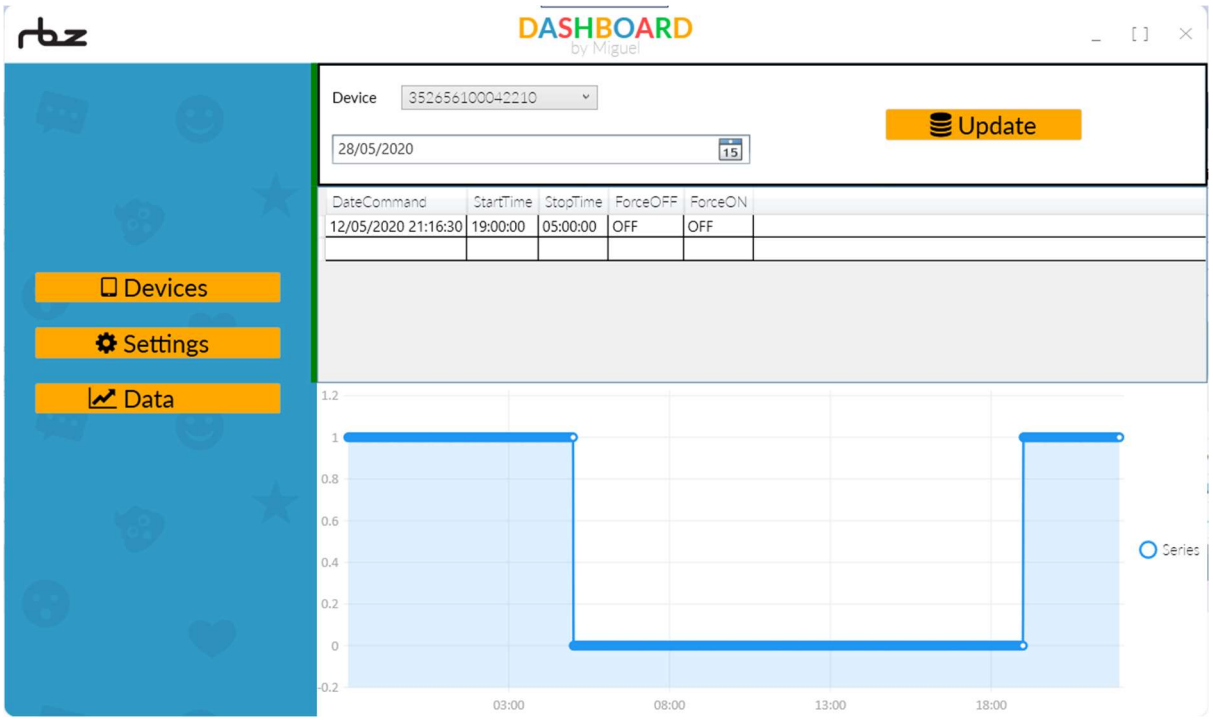

Ilustración 32 Vista de visualización de datos.

3. VALIDACIÓN Y RESULTADOS

Este capítulo se centra en la validación y muestra de resultados del sistema. Para ello se utilizará la aplicación de cliente diseñada en el proyecto y una serie de métricas para validar la recopilación de datos y respuesta del sistema. La aplicación de usuario tiene como objetivo mostrar los datos y permitir un análisis tanto del rendimiento como del funcionamiento del sistema.

Para garantizar que este funcionamiento era correcto, el sistema se mantuvo en ejecución durante varios días consecutivos en las tres pruebas realizadas. Estos datos son de las fechas 11, 12, 13 y 14 de mayo, 27, 28 y 29 de mayo y el 6 y 7 de junio de 2020. En el capítulo se mostrarán los resultados recopilados y se explicarán los procesos por los que ha ido pasando el sistema. Para ello se compararán las actualizaciones de las tablas de configuración respecto a los datos recopilados. Tras ello, se explicarán una serie de métricas para medir la capacidad de recopilación de datos y la respuesta del sistema. Se van a mostrar una serie de ilustraciones que albergan los datos de la aplicación; el resto de ellas debido a la similitud están en el Anexo A.

En la Ilustración 33 se muestran los datos recopilados durante el 11 de mayo de 2020. Como se muestra en la imagen, el último valor de la tabla de actualización es del día anterior, del 10 de mayo. Esta tabla de actualización no está habilitando ni el apagado ni el encendido forzado del sistema. En cambio, está configurando que las luces se apaguen a las 5:00 y se enciendan a partir de las 19:00. Esta gráfica muestra el funcionamiento normal del sistema de alumbrado el cual es el recopilado en la mayoría de los días.

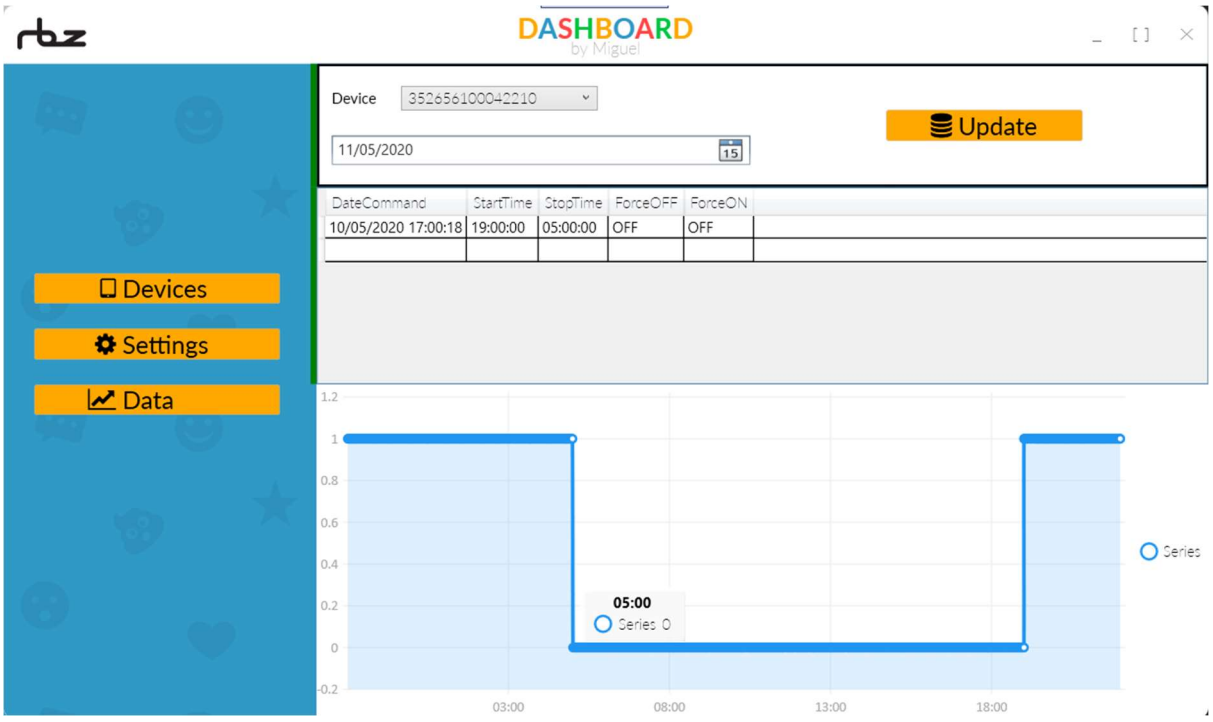

Ilustración 33 Gráfica día 11 de mayo mostrando apagado sistema.

La Ilustración 34 muestra los datos recopilados del día 12 de mayo de 2020. En este día se realizaron pruebas para comprobar el tiempo de respuesta de los dispositivos al envío de nuevas tablas de configuración. En la primera entrada de la tabla que se muestra en estas imágenes se detalla la tabla de configuración enviada. En esta primera actualización, se mantiene el horario de encendido y apagado, pero en cambio se fuerza el encendido. Como señala la tabla, la actualización es enviada a las 06:16:40, hora a la que el sistema tiene las luces apagadas. La Ilustración 34 muestra como a las

06:17:07 el sistema enciende las luces. Por lo tanto, el tiempo de respuesta del sistema es menor a un minuto y no afecta a los datos recopilados.

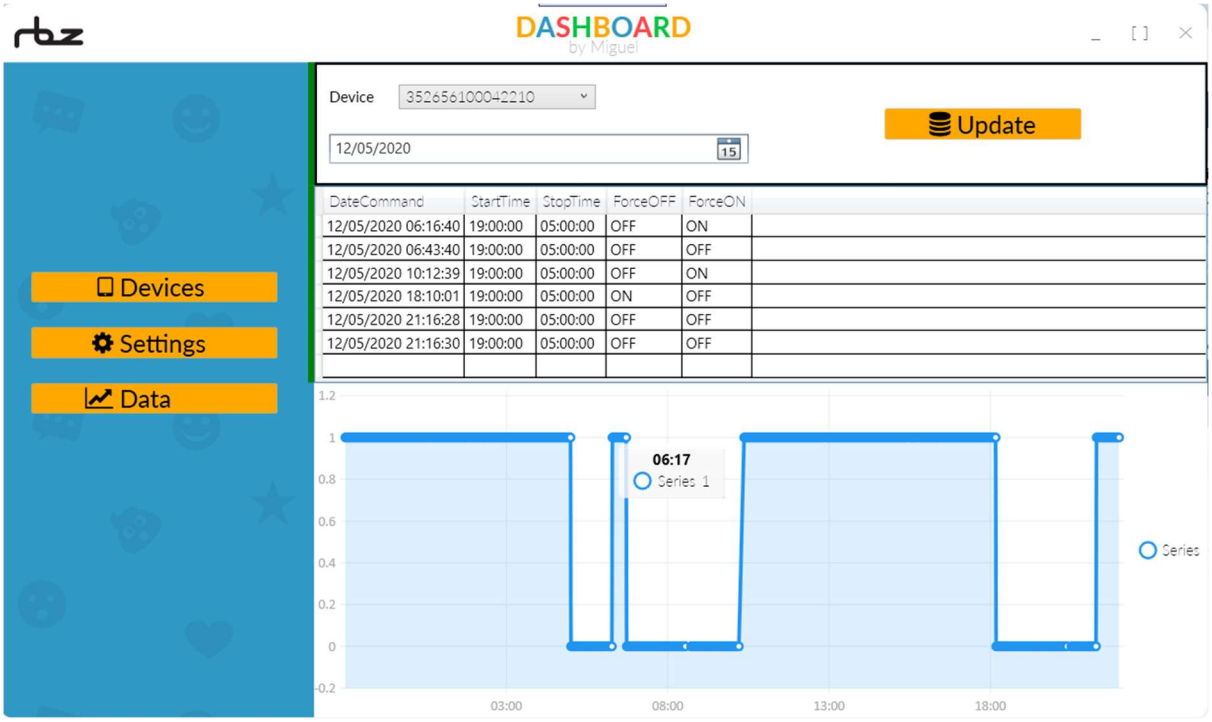

Ilustración 34 Gráfica día 12 de mayo.

La segunda entrada de la tabla muestra como a las 06:43:40 se deshabilitan los encendidos y apagados forzados, por lo que el sistema vuelve a la operativa normal. Se puede comprobar cómo a las 06:43:43 el sistema empieza a reportar datos con el sistema apagado, como especifica la tabla de configuración.

En la tercera entrada de la tabla a las 10:12:39 se fuerza un encendido del sistema. En la Ilustración 34 se muestra cómo el sistema se enciende con un ligero retraso, este es estudiado en las métricas del final del capítulo.

En la cuarta entrada de la tabla la actualización busca forzar un apagado del sistema. Esta actualización fue enviada a las 18:10:01 y es a las 18.10:48 cuando el sistema se apaga. El sistema se mantiene apagado incluso cuando las 19:00:00 son superadas y no es hasta la quinta actualización del día, como se muestra en la quinta entrada de la tabla, cuando el sistema vuelve a un funcionamiento normal. Esta quinta actualización se realiza a las 21:16:30 y el sistema vuelve al funcionamiento normal a las 21:17:27.

En la Ilustración 35 se muestran los datos recopilados del día 28 de mayo. La tabla de últimas configuraciones muestra como el sistema no ha recibido ninguna actualización desde el día 12 de mayo. Por lo tanto, el sistema funciona correctamente tras pasar un periodo de inactividad manteniendo las configuraciones.

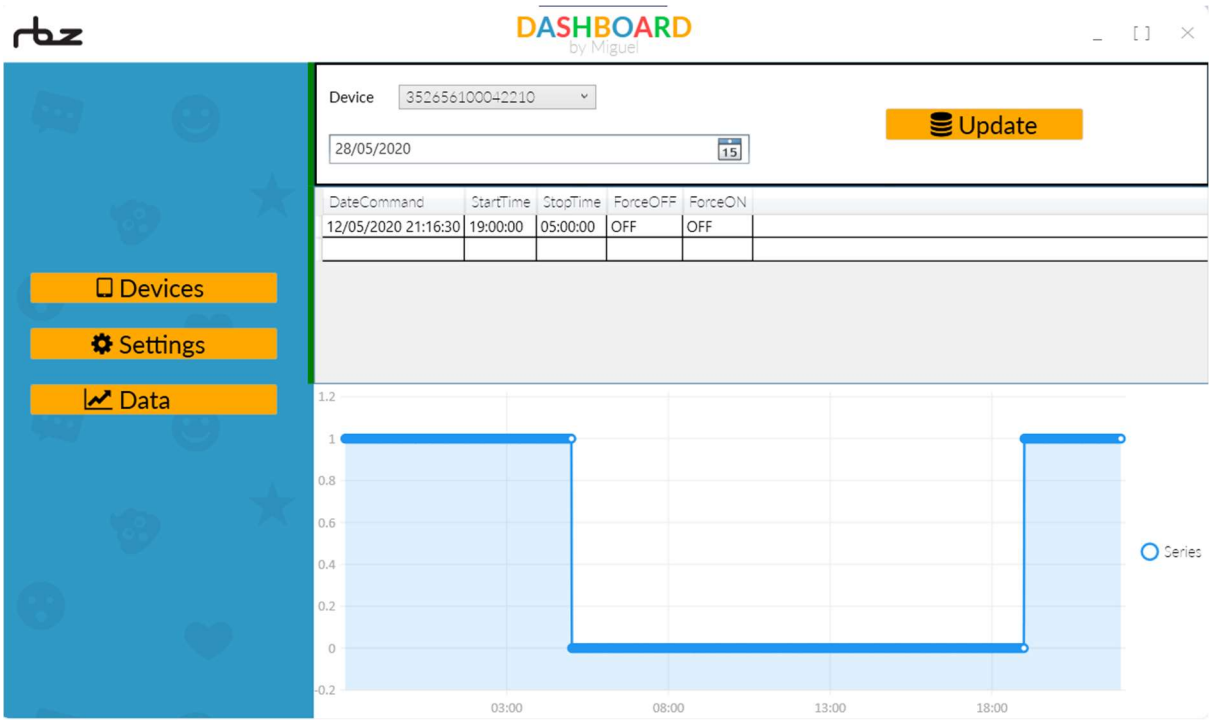

Ilustración 35 Gráfica del día 28 de mayo.

En la Ilustración 36 se muestran los datos del día 7 de junio, en el cual se hicieron pruebas de actualización de la tabla de configuración. Como se aprecia en la tabla de la Ilustración se cambiaron las horas de inicio y de apagado que se estaban usando para comprobar el funcionamiento del sistema.

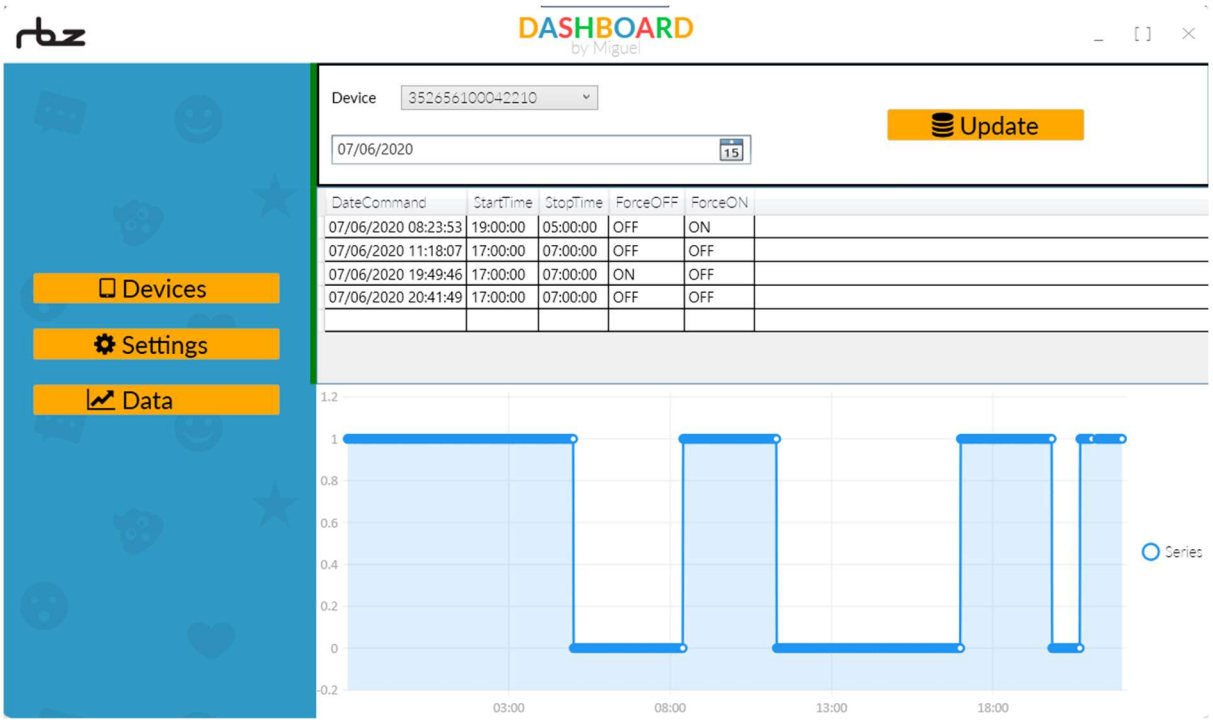

Ilustración 36 Gráfica del día 7 de junio.

Para validar el proyecto se diseñaron las siguientes métricas con el objetivo de caracterizar el funcionamiento del sistema:

- Métrica 1: Número de datos recopilados por día dividido entre el número total de paquetes esperados por día. Esta métrica busca determinar la eficiencia en el reporte de datos de nuestro sistema. Estando los rangos de la métrica entre [0,1], con 1 como el óptimo y 0 que no se ha recibido ningún dato.
- Métrica 2: Desde el envío de una actualización de configuración cuantos mensajes son reportados hasta el ajuste del sistema. Esta métrica busca calcular el tiempo de respuesta del sistema a las nuevas actualizaciones. Estando los valores de la métrica entre [1, +∞), siendo 1 que en el siguiente mensaje el sistema se ha ajustado al ajuste del sistema e infinito que el sistema no ha respondido al ajuste y sigue enviando mensajes con la anterior configuración.
- Métrica 3. Cuantos segundos han pasado entre el envío de la actualización y el primer reporte de datos que refleja esa actualización. Esta métrica también caracteriza el tiempo de respuesta del sistema. Estando el rango de la métrica en segundos y mostrando la diferencia entre el tiempo medido de la respuesta con respecto al tiempo medido de la actualización. El tiempo óptimo sería 0 [s], si tras la actualización inmediatamente se hiciera un reporte datos y el peor caso infinito, sí el sistema nunca llegará a procesar la actualización.

En la Tabla 1 se muestran los resultados de la Métrica 1 por día y la media obtenida. El sistema tiene un resultado mayor del 97% en esta métrica y por ello se considera un sistema fiable.

Tabla 1 Resultados Métrica 1.

Día	Métrica 1
11 de mayo	1425/1440 = 0.99
12 de mayo	1414/1440 = 0.98
13 de mayo	1377/1440 = 0.96
14 de mayo	1431/1440 = 0.99
27 de mayo	1405/1440 = 0.97
28 de mayo	1440/1440 = 1
29 de mayo	1423/1440 = 0.99
6 de junio	1353/1440 = 0.94
7 de junio	1425/1440 = 0.99

Resultado	0.98

En las siguientes Tablas se presentan las Métricas 2 y 3. Estas solo pueden realizarse en los días 12 de mayo y 7 de junio debido a que son los días que incluyen las pruebas de estos parámetros. Se utiliza la notación [tiempo1, tiempo2] donde "tiempo1" representa la hora en la que la actualización de la tabla de configuración fue enviada y "tiempo2" representa la hora del dato recopilado.

Tabla 2 Resultados Métricas 2 y 3 día 12 de mayo.

Actualización	Métrica 2	Métrica 3
Actualización 1	[06:16:40, 06:17:07] => 1	[06:16:40, 06:17:07] => 27 [s]
Actualización 2	[06:43:40, 06:43:43] => 1	[06:43:40, 06:43:43] => 3 [s]
Actualización 3	[10:12:39, 10:22:08] => 1	[10:12:39, 10:22:08] => 569 [s]
Actualización 4	[18:10:01, 18:10:48] => 1	[18:10:01, 18:10:48]=> 47 [s]
Actualización 5	[21:16:28, 21:17:27] => 1	[21:16:28, 21:17:27] => 59 [s]
Resultado	1	141 [s]

Tabla 3 Resultados Métricas 2 y 3 día 7 de junio.

Actualización	Métrica 2	Métrica 3
Actualización 1	[08:23:53, 08:24:11] => 1	[08:23:53, 08:24:11] => 18 [s]
Actualización 2	[11:18:07, 11:18:14] => 1	[11:18:07, 11:18:14] => 7 [s]
Actualización 3	[19:49:46, 19:50:10] => 1	[19:49:46, 19:50:10] => 24 [s]
Actualización 4	[20:41:49, 20:42:20] => 1	[20:41:49, 20:42:20]=> 31 [s]
Resultado	1	20 [s]

Tabla 4 Resultados Métricas 2 y 3.

Actualización	Métrica 2	Métrica 3
12 de mayo	1	141 [s]
7 de junio	1	20 [s]
Resultado	1	87 [s]

Analizando estos resultados se concluye que el sistema siempre recibe las actualizaciones de las tablas de configuración y responde a ellas con un bajo tiempo de respuesta. Esto es debido a que, si se analiza la Métrica 2, cuando el sistema recibe una actualización el siguiente reporte de datos siempre es acorde a ella. Con la Métrica 3, se concluye que el sistema tiene un tiempo de respuesta menor de1 minuto y medio.

4. CONCLUSIONES Y LÍNEAS FUTURAS

4.1. CONCLUSIONES

El principal objetivo de este Trabajo Fin de Máster ha sido el desarrollo de una arquitectura funcional IoT para el control del alumbrado público. Esta arquitectura es capaz de gestionar un elevado número de dispositivos conectados a la red y, a la vez, soportar el incremento de dispositivos sin pérdida de prestaciones. Además, con una alta precisión temporal, se puede determinar en qué momento cada nodo conectado a la red se enciende, manteniendo esas configuraciones incluso con una pérdida de alimentación en los dispositivos finales.

La arquitectura se ha diseñado con una gran modularidad, la cual favorece su reutilización y mantenimiento. Esta arquitectura es adaptable tanto a un sistema de control de alumbrado público como a un sistema de gestión de basuras, lo que la hace idónea para ser utilizada en las ciudades inteligentes. Esta adaptabilidad y gran capacidad de reutilización aporta un gran valor añadido al permitir incluir varias aplicaciones finales de la arquitectura sin tener que desplegar varias de ellas o diseñar un nueva. Su mantenimiento es otro valor a destacar, debido a que la modularidad otorga un gran desacoplo tecnológico. Éste facilita la implementación de nuevas tecnologías en la arquitectura minimizando los cambios que se deben realizar en el sistema.

Los nodos presentan un sistema capaz de gestionar la perdida de conectividad, garantizando el reporte del estado del sistema de manera periódica. El uso de la red de acceso LTE-M simplifica el diseño del nodo y la gestión de la conectividad. Además, el uso de esta red de acceso otorga una referencia temporal de alta precisión exigida en los sistemas de alumbrado público. Junto con el Broker y el manejo del protocolo de red MQTT el reporte de datos por parte del nodo es sencillo y jerarquizado, permitiendo una clasificación posterior muy eficiente.

La capa de persistencia de datos es capaz de clasificar los datos procedentes de los nodos gracias al protocolo de red MQTT. Esta clasificación garantiza un control individualizado de los nodos y proporciona escalabilidad en la arquitectura. El uso de la tecnología MongoDB y su modelo NoSQL garantiza la adaptabilidad del sistema, permitiendo que la inclusión de nuevos parámetros en la recopilación de datos de los nodos no afecte a su estructura ni implique un rediseño.

La aplicación de usuario consigue el objetivo de visualizar la información pertinente de manera sencilla y rápida. Esta es capaz de controlar la actualización individualizada de cada nodo de la arquitectura. Además, la visualización de los datos reportados por los nodos se muestra gráficamente permitiendo un análisis más detallado del sistema.

4.2. LÍNEAS FUTURAS

Como posibles líneas futuras de desarrollo se presentan distintas propuestas, que se clasifican en las siguientes categorías:

Arquitectura del sistema:

- Encapsular la capa de persistencia de datos en un servidor HTTP. La inclusión de este servidor en la arquitectura permite la posibilidad de migrar la tecnología de almacenamiento tras las peticiones de este protocolo de red.
- Cambiar la plataforma hardware de la capa de persistencia de datos a los servidores de algún proveedor de la nube. Este cambio aumenta el número de peticiones de acceso de la base de datos y permite implementar la capacidad de crecimiento estructural de las bases de datos NoSQL, que es extender la base de datos entre diversas plataformas hardware simultáneamente.

Seguridad:

- Establecer comunicaciones cifradas en todas las interfaces de comunicación del sistema.
- Incluir la autenticación de la aplicación de cliente en un nuevo servidor HTTP. Tras la autenticación, este nuevo servidor devolverá la dirección de la capa de persistencia de datos en función del usuario que se conecte. De esta forma, la dirección de la capa de persistencia de datos no está guardada en el código de la aplicación, sino que es dinámica en la vida del sistema.
- Incluir dispositivos hardware en los nodos que incluyan certificados de seguridad para poder asegurar que los nodos que envían los datos no han sido suplantados.

Aplicación de cliente:

- Mejorar la visualización de los datos recopilados en el sistema, posibilitando el dibujo de varios dispositivos a la vez entre otras funcionalidades.
- Inclusión de una nueva vista de localización GPS de los dispositivos.

Nodos:

- Modificar la tabla de configuración e incluir nuevos parámetros de control como la selección de la red de acceso a la que se conecta el dispositivo entre LTE-M y NB-IoT, el uso de comunicaciones seguras, el puerto del Broker al que se conecta el dispositivo y la recopilación de la geolocalización GPS.
- Implementar la funcionalidad de actualización del firmware del dispositivo mediante las comunicaciones inalámbricas.

5. BIBLIOGRAFÍA

[1] Wikipedia, «Arpanet,» [En línea]. Available: https://es.wikipedia.org/wiki/ARPANET. [Último acceso: 20 05 2020].

[2] Wikipedia, «Internet,» [En línea]. Available: https://es.wikipedia.org/wiki/Internet. [Último acceso: 27 05 2020].

[3] UIT, «Recomendación UIT-T Y.2060(06/2012),» [En línea]. Available: https://www.itu.int/rec/T-REC-Y.2060-201206-I/es. [Último acceso: 31 05 2020].

[4] L. Allince, «LoRa Alliance,» [En línea]. Available: https://lora-alliance.org/. [Último acceso: 31 05 2020].

[5] Wikipedia, «NB-IoT,» [En línea]. Available: https://es.wikipedia.org/wiki/Red_IoT. [Último acceso: 31 05 2020].

[6] Wikipedia, «LTE-M,» [En línea]. Available: https://en.wikipedia.org/wiki/LTE-M. [Último acceso: 31 05 2020].

[7] R. Journal, «RFID Journal,» [En línea]. Available: https://www.rfidjournal.com/. [Último acceso: 31 05 2020].

[8] Wikipedia, «Auto-ID Center,» [En línea]. Available: https://en.wikipedia.org/wiki/Auto-ID_Labs. [Último acceso: 31 05 2020].

[9] MIT, «MIT,» [En línea]. Available: https://www.mit.edu/. [Último acceso: 31 05 2020].

[10] C. Systems, «Cysco Systems,» [En línea]. Available: https://www.nasdaq.com/market-activity/stocks/csco. [Último acceso: 31 05 2020].

[11] N. Unidas, «Naciones Unidas,» [En línea]. Available: https://www.un.org/es/. [Último acceso: 31 05 2020].

[12] UIT, «UIT,» [En línea]. Available: https://www.itu.int/es/Pages/default.aspx. [Último acceso: 31 05 2020].

[13] UIT-T-REC-Y.2060, «Descripción general de Internet de los objetos,» 2012. [En línea].

[14] Wikipedia, «LPWAN,» [En línea]. Available: https://en.wikipedia.org/wiki/LPWAN. [Último acceso: 31 05 2020].

[15] 3GPP, «3GPP The Mobile Broadband Standard,» [En línea]. Available: https://www.3gpp.org/.

[16] Wikipedia, «VoIP,» [En línea]. Available: https://en.wikipedia.org/wiki/Voice_over_IP. [Último acceso: 31 05 2020].

[17] R. R. Design, «RBZ Embedded Logics,» 2012. [En línea]. Available: http://rbz.es/. [Último acceso: 30 05 2020].

[18] T. L. Fundation, «The Linux Fundation,» [En línea]. Available: https://www.linuxfoundation.org/. [Último acceso: 30 05 2020].

[19] MQTT, «MQTT,» [En línea]. Available: http://mqtt.org/. [Último acceso: 31 05 2020].

[20] CoAP, «CoAP Technology,» [En línea]. Available: https://coap.technology/. [Último acceso: 31 05 2020].

[21] Wikipedia, «SQL,» [En línea]. Available: https://es.wikipedia.org/wiki/SQL#:~:text=SQL%20(por%20sus%20siglas%20en,de%20bases%20de%20datos%20relacionales.. [Último acceso: 02 06 2020].

[22] Wikipedia, «NoSQL,» [En línea]. Available: https://es.wikipedia.org/wiki/NoSQL#:~:text=En%20inform%C3%A1tica%2C%20NoSQL%20(a%20veces,como%20lenguaje%20principal%20de%20consultas.. [Último acceso: 02 06 2020].

[23] STMicroelectronics, «STM,» [En línea]. Available: www.st.com. [Último acceso: 02 06 2020].

[24] NXP, «NXP,» [En línea]. Available: https://www.nxp.com/. [Último acceso: 02 06 2020].

[25] N. Semiconductor, «Nordic Semiconductor,» [En línea]. Available: https://www.nordicsemi.com/. [Último acceso: 2 06 2020].

[26] ARM, «ARM,» [En línea]. Available: https://www.arm.com/. [Último acceso: 02 06 2020].

[27] ARM, «Cortex-M,» [En línea]. Available: https://developer.arm.com/ip-products/processors/cortex-m. [Último acceso: 02 06 2020].

[28] STMicroelectronics, «AT &T IoT Starter Kit (LTE-M, STM32L4),» [En línea]. Available: https://www.st.com/en/solutions-reference-designs/at-t-iot-starter-kit-lte-m-stm32l4.html. [Último acceso: 02 06 2020].

[29] STMicroelectronics, «L4 Series,» [En línea]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l4-series.html. [Último acceso: 02 06 2020].

[30] ARM, «Cortex-M4,» [En línea]. Available: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m4. [Último acceso: 02 06 2020].

[31] Wikipedia, «Tarjeta SIM,» [En línea]. Available: https://es.wikipedia.org/wiki/Tarjeta_SIM. [Último acceso: 02 06 2020].

[32] N. Semiconductor, «nRF9160 SiP,» [En línea]. Available: https://www.nordicsemi.com/Products/Low-power-cellular-IoT/nRF9160. [Último acceso: 02 06 2020].

[33] ARM, «Cortex-M33,» [En línea]. Available: https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m33. [Último acceso: 02 06 2020].

[34] ARM, «TrustZone,» [En línea]. Available: https://developer.arm.com/ip-products/security-ip/trustzone. [Último acceso: 02 06 2020].

[35] Wikipedia, «RTOS,» [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_operativo_de_tiempo_real. [Último acceso: 02 06 2020].

[36] Wikipedia, «Bare-Metal,» [En línea]. Available: https://en.wikipedia.org/wiki/Bare-metal_server. [Último acceso: 02 06 2020].

[37] Wikipedia, «TCP/IP,» [En línea]. Available: https://es.wikipedia.org/wiki/Pila_de_protocolos. [Último acceso: 02 06 2020].

[38] Wikipedia, «Open Source,» [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_de_c%C3%B3digo_abierto. [Último acceso: 02 06 2020].

[39] Wikipedia, «Watchdog Timer,» [En línea]. Available: https://en.wikipedia.org/wiki/Watchdog_timer. [Último acceso: 02 06 2020].

[40] T. L. F. Projects, «Zephyr,» [En línea]. Available: https://www.zephyrproject.org/. [Último

acceso: 02 06 2020].

[41] Wikipedia, «UART,» [En línea]. Available: https://es.wikipedia.org/wiki/Universal_Asynchronous_Receiver-Transmitter#:~:text=UART%2C%20son%20las%20siglas%20en,la%20tarjeta%20adaptadora%20del%20dispositivo.. [Último acceso: 02 06 2020].

[42] Wikipedia, «GPIO,» [En línea]. Available: https://es.wikipedia.org/wiki/GPIO#:~:text=GPIO%20(General%20Purpose%20Input%2FOutput,usuario%20en%20tiempo%20de%20ejecuci%C3%B3n.. [Último acceso: 02 06 2020].

[43] Wikipedia, «Hayes Commands Set,» [En línea]. Available: https://en.wikipedia.org/wiki/Hayes_command_set. [Último acceso: 02 06 2020].

[44] Wikipedia, «IMEI,» [En línea]. Available: https://es.wikipedia.org/wiki/IMEI. [Último acceso: 02 06 2020].

[45] JSON, «JSON,» [En línea]. Available: https://www.json.org/json-es.html. [Último acceso: 02 06 2020].

[46] IBM, «IBM,» [En línea]. Available: https://www.ibm.com/es-es. [Último acceso: 02 06 2020].

[47] Eurotech, «Eurotech,» [En línea]. Available: https://www.eurotech.com/en/section/about-eurotech. [Último acceso: 02 06 2020].

[48] OASIS, «OASIS,» [En línea]. Available: https://www.oasis-open.org/. [Último acceso: 02 06 2020].

[49] Wikipedia, «TLS,» [En línea]. Available: https://es.wikipedia.org/wiki/Seguridad_de_la_capa_de_transporte. [Último acceso: 02 06 2020].

[50] Wikipedia, «Autenticación,» [En línea]. Available: https://es.wikipedia.org/wiki/Autenticaci%C3%B3n. [Último acceso: 02 06 2020].

[51] R. Foundation, «RaspberryPi,» [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 02 06 2020].

[52] Ubuntu, «Ubuntu,» [En línea]. Available: https://ubuntu.com/. [Último acceso: 02 06 2020].

[53] Wikipedia, «Modelo Relacional,» [En línea]. Available: https://es.wikipedia.org/wiki/Modelo_relacional. [Último acceso: 31 05 2020].

[54] MongoDB, «MongoDB,» [En línea]. Available: https://www.mongodb.com/es. [Último acceso: 31 05 2020].

[55] No-IP, «No-IP,» [En línea]. Available: https://www.noip.com/. [Último acceso: 01 06 2020].

[56] Wikipedia, «C#,» [En línea]. Available: https://en.wikipedia.org/wiki/C_Sharp_(programming_language). [Último acceso: 01 06 2020].

[57] MongoDB, «MongoDB C#./NET Driver,» [En línea]. Available: https://docs.mongodb.com/drivers/csharp. [Último acceso: 01 06 2020].

[58] P. S. Foundation, «Python,» [En línea]. Available: https://www.python.org/psf/. [Último acceso: 01 06 2020].

[59] E. Mosquitto, «Mosquitto,» [En línea]. Available: https://mosquitto.org/. [Último acceso: 01 06 2020].

[60] PyMongo, «PyMongo,» [En línea]. Available: https://pymongo.readthedocs.io/en/stable/.

[Último acceso: 01 06 2020].

[61] Microsoft, «Microsoft,» [En línea]. Available: https://www.microsoft.com/es-es. [Último acceso: 01 06 2020].

[62] Microsoft, «Windows,» [En línea]. Available: https://www.microsoft.com/es-es/windows. [Último acceso: 07 06 2020].

[63] Microsoft, «Windows Presentation Foundation,» [En línea]. Available: https://docs.microsoft.com/es-es/visualstudio/designers/getting-started-with-wpf?view=vs-2019. [Último acceso: 01 06 2020].

[64] Wikipedia, «Dashboard,» [En línea]. Available: https://en.wikipedia.org/wiki/Dashboard_(business). [Último acceso: 01 06 2020].

[65] T.-l. L. F.-Y. L. l. S. H.-Y. D. Miao Wu, «Research on the architecture of Internet of things,» *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE) ,* 2010.

ANEXO A: GRÁFICOS DE DATOS RECOPILADOS DEL SISTEMA.

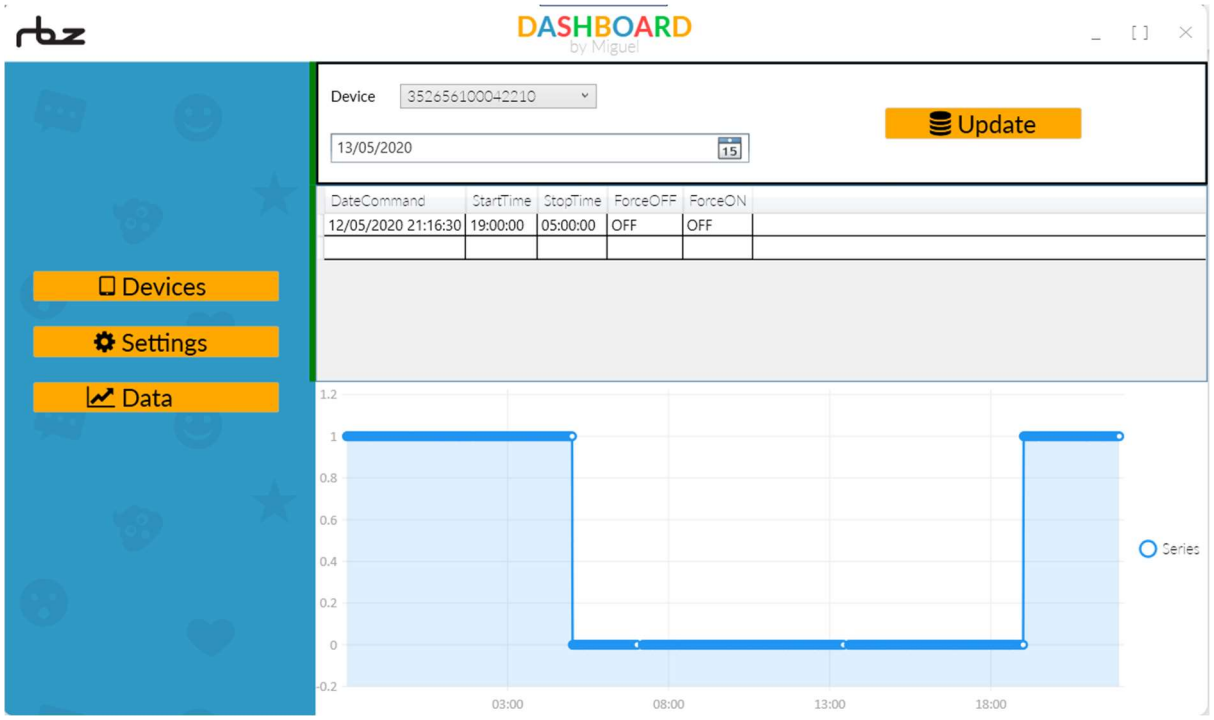

Ilustración 37 Anexo A. Gráfica datos día 13 de mayo.

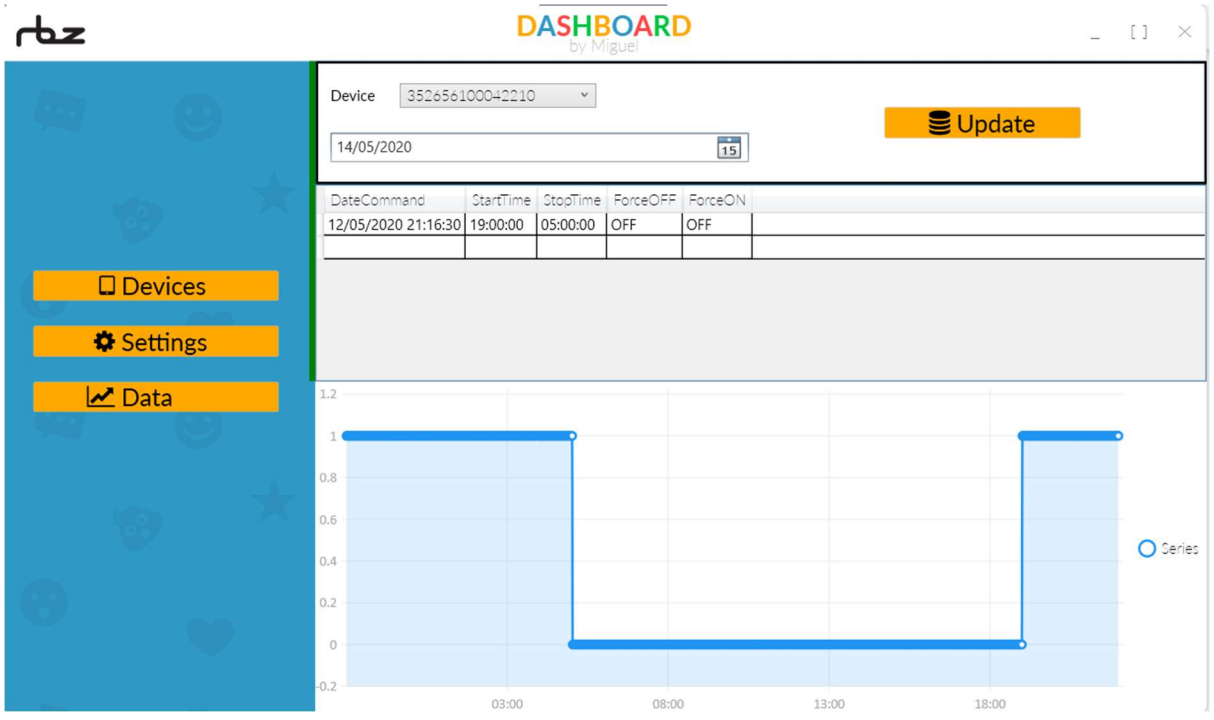

Ilustración 38 Anexo A. Gráfica datos día 14 de mayo.

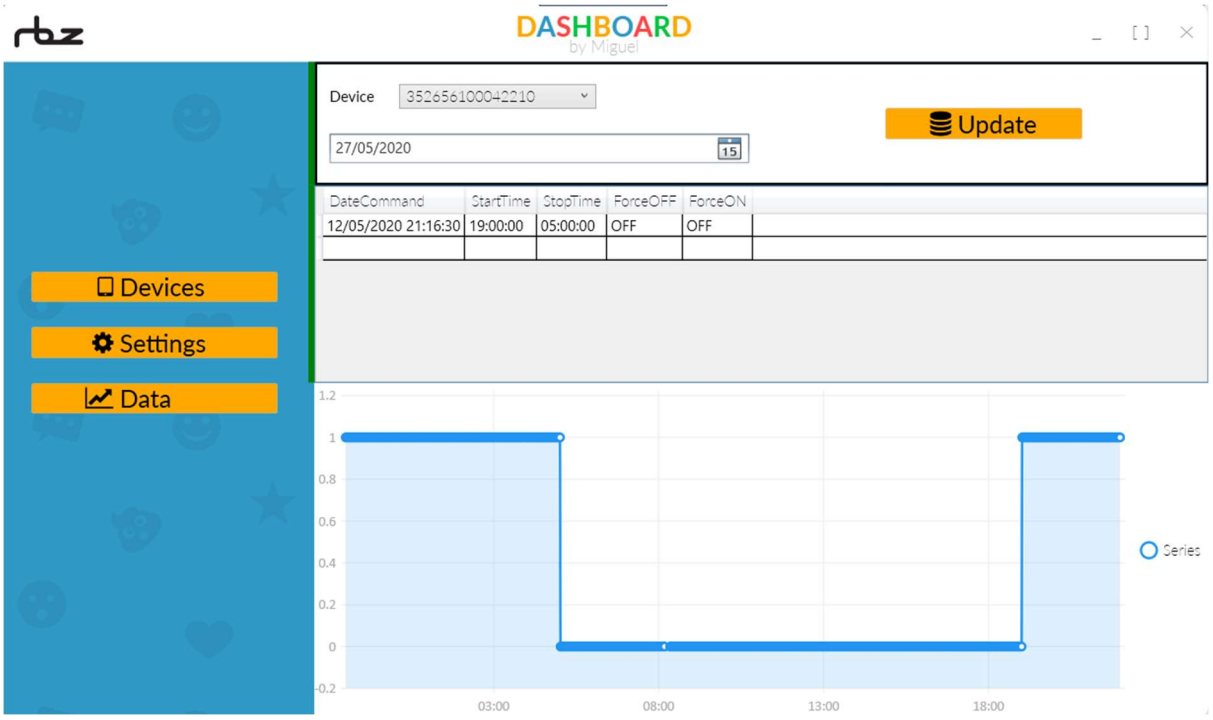

Ilustración 39 Anexo A. Gráfica datos día 27 de mayo.

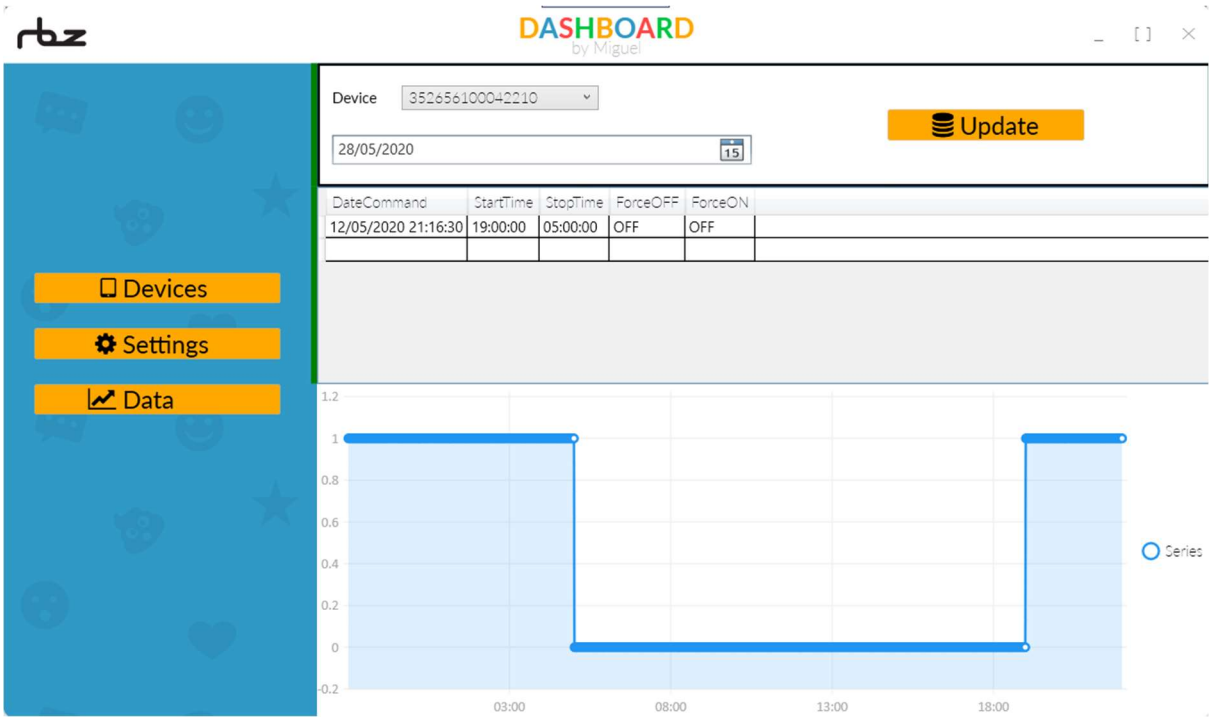

Ilustración 40 Anexo A. Gráfica datos día 28 de mayo.

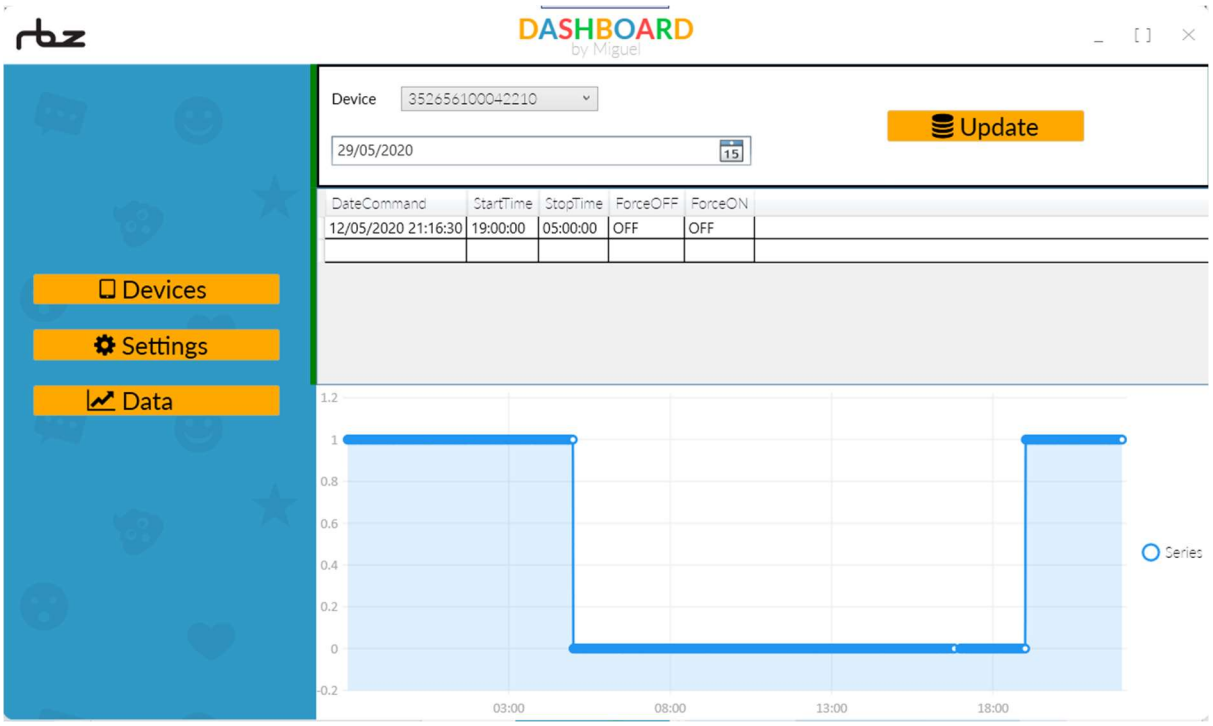

Ilustración 41 Anexo A. Gráfica datos día 29 de mayo.

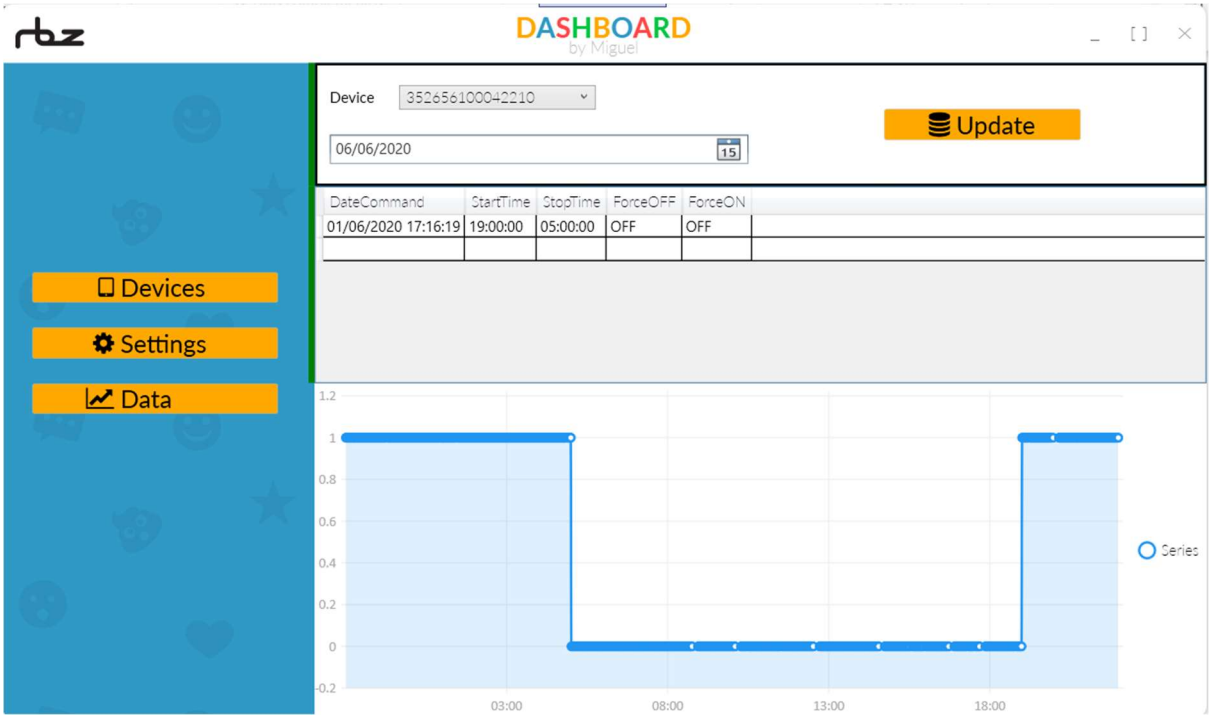

Ilustración 42 Anexo A. Gráfica datos día 6 de junio.

ANEXO B: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

B.1 INTRODUCCIÓN

Este TFM diseña y desarrolla una arquitectura IoT con el objetivo de controlar el alumbrado público de una ciudad. Esta arquitectura se incluye dentro de los proyectos de las ciudades inteligentes. Estas buscan mediante el uso de las Tecnologías de la Información y la Comunicación dotarse de infraestructuras y servicios que garanticen un desarrollo sostenible, un incremento de la calidad de vida de sus ciudadanos y una mayor eficiencia de sus recursos.

B.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

Los aspectos éticos de este TFM están directamente ligados con los sociales, económicos y medioambientales, debido a las mejoras que aporta sobre los sistemas convencionales. En el aspecto social se considera una mejora el servicio que aporta el uso de un sistema de control individualizado del alumbrado. El uso de un sistema que reporta periódicamente el estado de cada uno de los sistemas de alumbrado público mejora el mantenimiento del mismo, reduciendo los tiempos de reparación. Además, este control permite establecer nuevas actuaciones sociales que inciden directamente en la calidad de vida de los ciudadanos, como mejorar las posibles afecciones del sueño disminuyendo la luminosidad y por tanto reduciendo los índices de contaminación lumínica, o por el contrario, incrementando la luminosidad en áreas potencialmente conflictivas. También se puede incluir en los aspectos sociales que gracias a este control se puede mencionar el encendido o apagado del sistema para actividades sociales o eventos específicos que la ciudad demande.

Como aspecto económico principal, está la reducción de costes en el despliegue de estos sistemas. Si este mismo sistema se implementa sin basarse en el uso de redes de acceso inalámbricas, para conseguir el mismo control el coste es mucho más elevado. Por otra parte, la capacidad de gestionar cada una de las farolas del sistema de alumbrado público permite reducir el consumo del sistema introduciendo políticas de reducción de consumo.

En los aspectos ambientales se destaca la reducción de la contaminación lumínica y la reducción del consumo de los sistemas de alumbrado público. Estos dos aspectos vienen relacionados, gracias al control individualizado se pueden deshabilitar las zonas de alumbrado público que no se estén utilizando o disminuir la potencia lumínica un 50% para reducir el consumo hasta que esas zonas tengan tránsito.

B.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

La reducción del consumo gracias a la arquitectura de este sistema es esencial. Para obtener el mismo grado de control con una arquitectura no inalámbrica el coste adicional en el sistema es desproporcionado. Con este sistema, en el que la inclusión de los dispositivos a la red de alumbrado público es inmediata y no necesita de un mayor despliegue de medios, se posibilita la modernización de los sistemas en todos los puntos del país. Una vez implementado el sistema se pueden aplicar nuevas políticas de reducción del consumo eléctrico, las cuales pueden estar asociadas a la mejora de

la calidad del sueño de los habitantes o a disminuir la contaminación lumínica en zonas despobladas o con un bajo tránsito de personas a ciertas horas del día. Esta reducción lumínica permite poner el sistema en bajo consumo si la política aplicada no considera oportuno apagar el sistema completo, debido a que esto puede disminuir la sensación de seguridad en la zona.

B.4 CONCLUSIONES

El uso del sistema desarrollado en este TFM permite tanto reducir el consumo eléctrico de los sistemas de alumbrado público como gestionar mejor este consumo para focalizarlo en las zonas de más tránsito. Junto con esta reducción del consumo se incluye la reducción de la contaminación lumínica que es un problema muy importante en las grandes ciudades, pues este afecta a los ciclos de sueño no solo la fauna de alrededor de las ciudades, sino de los propios habitantes que viven en ellas. Todas las ventajas que aporta este TFM vienen de la mano de unos costes de despliegue muy bajos gracias a que el diseño reduce el número de recursos necesarios para el funcionamiento del sistema.

ANEXO C: PRESUPUESTO ECONÓMICO

En este anexo se presenta el presupuesto económico para la realización de este TFM.

COSTE DE MANO DE OBRA (coste directo)	**Horas**	**Precio/hora**	**Total**
	900	30 €	**27.000 €**

COSTE DE RECURSOS MATERIALES (coste directo)	**Precio de compra**	**Uso en meses**	**Amortización (en años)**	**Total**
Ordenador personal (Software incluido)	1.000,00 €	6	5	100,00 €
RaspberryPi 4B	50	6	3	8,33
Kit de desarrollo nRF9160	100	6	3	16,67

COSTE TOTAL DE RECURSOS MATERIALES	**125,00 €**

GASTOS GENERALES (costes indirectos)	15%	sobre CD	**4.068,75 €**
BENEFICIO INDUSTRIAL	6%	sobre CD+CI	**1.871,63 €**

SUBTOTAL PRESUPUESTO		**33.065,38 €**
IVA APLICABLE	21%	**6.943,73 €**
TOTAL PRESUPUESTO		**40.009,11 €**