



Facultad de
Ciencias Sociales y
Tecnologías de la Información
Talavera de la Reina. UCLM

UNIVERSIDAD DE CASTILLA-LA MANCHA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW

MQTT-inspired lightweight messaging protocol for ESP-NOW networks

Rubén Gómez Villegas

Julio, 2024



Facultad de
Ciencias Sociales y
Tecnologías de la Información
Talavera de la Reina. UCLM

UNIVERSIDAD DE CASTILLA-LA MANCHA

TRABAJO FIN DE GRADO

Departamento de Tecnologías y Sistemas de Información

Tecnología Específica de Sistemas de Información

Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW

Autor: Rubén Gómez Villegas

Tutor Académico: Rubén Cantarero Navarro

Cotutor Académico: Ana Rubio Ruiz

Julio, 2024

*Dedicado a mi familia y a todos
aquellos ...*

Declaración de Autoría

Yo, Rubén Gómez Villegas con DNI 02318379W declaro que soy el único autor del trabajo fin de grado titulado “Protocolo de mensajería ligero inspirado en MQTT para redes ESP-NOW” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Talavera de la Reina, a.....

Fdo:

Resumen

El Internet de las Cosas es muy relevante en la sociedad actual, ya que permite crear sistemas para reco-
pilar y procesar datos, así como transmitir órdenes a distintos dispositivos. Sin embargo, la implantación
de estos sistemas presenta problemas en ciertos entornos debido a diversas dificultades y costos. Estas di-
ficultades pueden surgir por la imposibilidad de instalar una red de Internet, el alto consumo de energía
de los dispositivos o la necesidad de utilizar una red inalámbrica sin una infraestructura adecuada para
procesar y transmitir mensajes. Para solucionar esta problemática, se propone el protocolo LoboMQ, que
permite gestionar mensajes transmitidos entre varios microcontroladores de bajo consumo mediante colas
de mensajes inspiradas en MQTT, y aprovecha ESP-NOW para realizar las transmisiones sin necesidad de
una red de Internet. El desarrollo de este protocolo y su implementación en una librería C++ buscan ofrecer
a la comunidad del Internet de las Cosas una solución sencilla de utilizar.

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Agradecimientos

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Desafíos durante la realización	2
1.3. Objetivos	2
1.3.1. Objetivo general	2
1.3.2. Objetivos específicos	3
1.4. Estructura del documento	3
2. Estado del arte	5
2.1. Internet de las Cosas	5
2.1.1. Ventajas y usos	7
2.1.2. Factores, limitaciones y desafíos a tener en cuenta	8
2.1.3. Primeros ejemplos de IoT	9
2.1.4. Tipos de dispositivos y redes	11
2.2. Middleware y protocolos de mensajería usados en IoT	12
2.2.1. Message Queue Telemetry Transport (MQTT)	12
2.2.2. Advanced Message Queuing Protocol (AMQP)	15
2.2.3. Extensible Messaging and Presence Protocol (XMPP)	16
2.2.4. Data Distribution Service (DDS)	17
2.2.5. Constrained Application Protocol (CoAP)	19
2.2.6. Comparación de middleware y protocolos	20
2.3. Espressif y sus dispositivos	21
2.3.1. ESP8266	21
2.3.2. ESP32	22
2.4. ESP-NOW	23
2.4.1. Tecnologías alternativas más comunes	26
2.5. Ejemplo de modelo tradicional publicador-broker-suscriptor con MQTT y ESP32	31
3. Herramientas y Metodología	33
3.1. Herramientas	33
3.1.1. Hardware	33

3.1.2. Software	35
3.1.3. Lenguajes	42
3.2. Metodología	44
3.2.1. Metodologías tradicionales y ágiles en el desarrollo de software	44
3.2.2. Scrum	46
3.2.3. OpenUP	50
3.2.4. Implementación en este proyecto	50
3.2.5. Control de versiones	57
3.2.6. Criterios de calidad y estilo del código	59
4. Resultados	61
4.1. Iteración 0	61
4.2. Iteración 1	62
4.3. Iteración 2	66
4.4. Iteración 3	70
4.5. Iteración 4	74
4.6. Iteración 5	80
4.7. Iteración 6	80
4.8. Iteración 7	81
4.9. Iteración 8	82
4.10. Iteración 9	83
4.11. Iteración 10	86
4.12. Iteración 11	87
4.13. Iteración 12	87
4.14. Iteración 13	87
4.15. Iteración 14	88
4.16. Diseño final del protocolo	89
5. Conclusiones	93
5.1. Revisión de los objetivos	93
5.2. Presupuesto	94
5.3. Competencias Específicas de Intensificación Adquiridas y/o Reforzadas	95
5.4. Propuestas de trabajo a futuro	96
Bibliografía	99
Anexo A. Pruebas	105
Anexo B. Diagrama de clases	109
Anexo C. Manual de uso y ejemplo simple	111
C.1. Broker	114

C.2. Publicador	115
C.3. Suscriptor y desuscriptor	116
Anexo D. Demostración con sensores	119

Índice de figuras

2.1. Esquema de un sistema de riego por aspersión utilizando dispositivos IoT	7
2.2. Formato de una trama MQTT (Fuente: [52])	14
2.3. Comparación de las capas del modelo OSI con las del modelo ESP-NOW TODO: REFERENCIAR	24
2.4. Formato de una trama ESP-NOW (Fuente: [64])	25
2.5. Comparación del rendimiento entre protocolos en distintos aspectos (Fuente: [41])	30
3.1. Esquema de la ejecución de Scrum (Fuente: [2])	47
3.2. Tablero Kanban utilizado durante el desarrollo	52
3.3. Vistas de las tarjetas del tablero: vista previa (izquierda) y detallada (derecha)	53
3.4. Etiquetas disponibles en el tablero	54
3.5. Reglas de automatización utilizadas en el tablero	55
3.6. Ilustración de Gitflow (Fuente: [65])	57
4.1. Diagrama de Gantt del proyecto desde febrero de 2024	61
4.2. Diagrama de casos de uso de LoboMQ	89
4.3. Formato de mensajes LoboMQ	90
4.4. Diagrama de secuencia de LoboMQ, con suscripción y publicación a un tema	90
4.5. Diagrama de secuencia de LoboMQ, con varios suscriptores y temas	91
B.1. Diagrama de clases de LoboMQ	109
C.1. Diagrama de conexión del broker	114

Índice de Tablas

2.1. Lista de estándares Wi-Fi <i>TODO: REFERENCIAR</i>	27
4.1. Lista de caracteres inválidos y su reemplazo para el nombrado de ficheros de persistencia	83
5.1. Estimación de costes del proyecto	95
A.1. Casos de prueba en la comprobación de topics	106

Índice de Listados

3.1. Ejemplo de uso de Doxygen	40
3.2. Ejemplo de estilo del código Kernighan & Ritchie con lowerCamelCase	59
4.1. Primera definición de los mensajes	63
4.2. Pseudocódigo de la primera definición del broker	64
4.3. Pseudocódigo de la primera definición del suscriptor	65
4.4. Pseudocódigo de la primera definición del publicador	66
4.5. Pseudocódigo de la modificación del broker para admitir publicaciones	66
4.6. Segunda definición de los mensajes	67
4.7. Pseudocódigo de la primera definición de la clase BrokerTopic	68
4.8. Pseudocódigo de la segunda definición del broker	69
4.9. Pseudocódigo de la primera versión de los parámetros de las funciones para publicar y suscribir definidos en la librería	71
4.10. Pseudocódigo de la modificación del broker para crear tareas según el mensaje recibido .	71
4.11. Pseudocódigo de las funciones añadidas para comprobar los topics	73
4.12. Pseudocódigo de la comprobación de compatibilidad en BrokerTopic	74
4.13. Pseudocódigo de la modificación del broker para insertar mensajes recibidos en colas .	75
4.14. Pseudocódigo de la función de desuscribir definida en la librería	77
4.15. Pseudocódigo de la adición de la desuscripción al broker	78
4.16. Pseudocódigo de la adición de la desuscripción a BrokerTopic	79
4.17. Pseudocódigo de las funciones de Logger	82
4.18. Ejemplo de fichero de persistencia	83
4.19. Pseudocódigo de las funciones de BrokerSDUtils	84
4.20. Versión final del pseudocódigo de la declaración de funciones del broker	86
A.1. Código para probar la creación de mensajes y la obtención de su contenido	105
C.1. Estructuras de PublishContent y PayloadContent	113
C.2. Código del broker de ejemplo	114
C.3. Código del publicador de ejemplo	115
C.4. Código del suscriptor y desuscriptor de ejemplo	116

Acrónimos

ACK Acknowledgement, Acuse de recibo

AES Advanced Encryption Standard, Estándar de Cifrado Avanzado

AMQP Advanced Message Queuing Protocol, Protocolo Avanzado de Colas de Mensajes

ARPANET Advanced Research Projects Agency Network, Red de la Agencia de Proyectos de Investigación Avanzada

API Application Programming Interface, Interfaz de Programación de Aplicaciones

BLE Bluetooth Low Energy, Bluetooth de Baja Energía

CoAP Constrained Application Protocol, Protocolo de Aplicación Restringida

CPU Central Processing Unit, Unidad Central de Procesamiento

CSS Cascading Style Sheets, Hojas de Estilo en Cascada

DDS Data Distribution Service, Servicio de Distribución de Datos

DTLS Data Transport Layer Security, Protocolo de Seguridad de la Capa de Transporte

ECC Elliptic Curve Cryptography, Criptografía de Curva Elíptica

ECDH Elliptic-curve Diffie–Hellman

ETS Erlang Term Storage, Almacenamiento de términos de Erlang

GB Gigabyte

GHz Gigahercios

GPIO General Purpose Input/Output, Entrada/Salida de Propósito General

HTML HyperText Markup Language, Lenguaje de Marcado de Hipertexto

HTTP HyperText Transfer Protocol, Protocolo de Transferencia de Hipertexto

ID Identificador, Identifier

IDE Integrated Development Environment, Entorno de Desarrollo Integrado

IEEE Institute of Electrical and Electronics Engineers, Instituto de Ingenieros Eléctricos y Electrónicos

IETF Internet Engineering Task Force, Grupo de Trabajo de Ingeniería de Internet

IoT Internet of Things, Internet de las Cosas

IP Internet Protocol, Protocolo de Internet

ITU International Telecommunication Union, Unión Internacional de Telecomunicaciones

JPEG Joint Photographic Experts Group

JSON JavaScript Object Notation, Notación de Objeto de JavaScript

Kbps Kilobits por segundo

KiB Kibibyte

LAN Local Area Network, Red de Área Local

mA Miliamperios

MAC Media Access Control, Control de Acceso a Medios

MAN Metropolitan Area Network, Red de Área Metropolitana

Mbps Megabits por segundo

MHz Megahercios

MIB Management Information Base, Base de Información Gestionada

MiB Mebibyte

MQTT Message Queue Telemetry Transport, Transporte de Telemetría en Colas de Mensajes

MTU Maximum Transmission Unit, Unidad de Transmisión Máxima

mW Milivatios

OASIS Organization for the Advancement of Structured Information Standards, Organización para el Avance de Estándares de Información Estructurada

OSI Open Systems Interconnection, Interconexión de Sistemas Abiertos

PDF Portable Document Format, Formato de Documento Portátil

PNG Portable Network Graphics, Gráficos de Red Portátiles

QoS Quality of Service, Calidad de Servicio

RAM Random Access Memory, Memoria de Acceso Aleatorio

REST Representational State Transfer, Transferencia de Estado Representacional

RFID Radio Frequency Identification, Identificación por Radiofrecuencia

RISC Reduced Instruction Set Computer, Computador con Conjunto de Instrucciones Reducido

RSA Rivest–Shamir–Adleman

SHA Secure Hash Algorithm, Algoritmo de Hash Seguro

SoC System on a Chip, Sistema en un Chip

SNMP Simple Network Management Protocol, Protocolo Simple de Gestión de Redes

SSL Secure Sockets Layer, Capa de Puertos Seguros

SVG Scalable Vector Graphics, Gráficos Vectoriales Escalables

TB Terabyte

TCP Transmission Control Protocol, Protocolo de Control de Transmisión

TLS Transport Layer Security, Seguridad de la Capa de Transporte

TFG Trabajo Final de Grado

UCLM Universidad de Castilla-La Mancha

UDP User Datagram Protocol, Protocolo de Datagramas de Usuario

V Voltios

VoIP Voice over Internet Protocol, Voz sobre Protocolo de Internet

WEP Wired Equivalent Privacy, Privacidad Equivalente a Cableado

WPA Wi-Fi Protected Access, Acceso Wi-Fi Protegido

XML Extensible Markup Language, Lenguaje de Marcado Extensible

XMPP Extensible Messaging and Presence Protocol, Protocolo Extensible de Mensajería y Comunicación de Presencia

μs Microsegundos

°C Grados centígrados

Capítulo 1

Introducción

En este primer capítulo se presenta el problema que se pretende resolver mediante el desarrollo del Trabajo de Fin de Grado (**TFG**), explicando el alcance de la solución, junto con la motivación y a los desafíos que han dado lugar.

A lo largo de este Trabajo de Fin de Grado se detalla el proceso de desarrollo de LoboMQ, un protocolo que se inspira en **MQTT** para tomar las bases de la comunicación publicador-suscriptor a través de colas de mensajes y adaptarlas a un entorno de trabajo con dispositivos del Internet de las Cosas. Este protocolo aprovecha las capacidades de ESP-NOW para ofrecer una comunicación sin cables ni Internet y a larga distancia, y es implementado en forma de una librería C++ para placas ESP32, la cual se encuentra disponible en el registro de PlatformIO. La librería LoboMQ ofrece al usuario las funciones necesarias para, con facilidad, desplegar un broker, publicar mensajes a un tema, y suscribirse y desuscribirse de los mismos.

1.1. Motivación

El crecimiento en la popularidad del Internet de las Cosas es un hecho [9] [61], observable tanto en la aparición de productos cotidianos como en las iniciativas de gran alcance emprendidas por organizaciones. Esta situación implica el desarrollo cada vez más frecuente de soluciones que se enfrenten y adapten a distintas restricciones, tanto del dispositivo conectado como del entorno en el que se conecta.

Uno de los dispositivos más populares en el ámbito del **IoT** son los microcontroladores ESP32, dispositivos con alta disponibilidad en el mercado, relativamente asequibles y con bastante apoyo y recursos en la comunidad de desarrolladores, siendo posible encontrar una gran cantidad de guías, ejemplos y tutoriales que permiten alcanzar las soluciones deseadas. Bajo la popularidad del **IoT** también se freqüentan protocolos de mensajería publicador-suscriptor basados en colas de mensajes, como **MQTT**, los cuales ofrecen la capacidad de comunicar bidireccionalmente muchos dispositivos entre sí. Existen una gran variedad de librerías que permiten implementar estos protocolos en microcontroladores [17] [16] [15], pero dependen de una conexión a Internet para comunicarse con el dispositivo encargado de gestionar las colas de mensajes. En algunos contextos con poco o ningún alcance de red, situaciones que son comunes en este ámbito, esta dependencia puede ser un problema que implique encarecer el uso de soluciones **IoT**, provocando la

pérdida de interés en los usuarios y desarrolladores.

Las placas de Espressif integran, junto con Wi-Fi, un protocolo de comunicación inalámbrica denominado ESP-NOW cuyo uso no requiere redes de Internet. Hasta donde alcanza el conocimiento del autor y los tutores de este TFG, el uso de ESP-NOW se ha limitado a que el desarrollador diseñe una manera propia de transmitir datos para cada proyecto, y no existe de forma pública ninguna solución que permita implementar un sistema publicación-suscripción aprovechando esta forma de comunicación. Esta es la oportunidad que el proyecto pretende explotar. Se busca aportar a la comunidad del Internet de las Cosas un protocolo utilizable en entornos restringidos por la red y el consumo energético, que combine tanto los beneficios de MQTT como los de ESP-NOW en placas ESP32.

1.2. Desafíos durante la realización

El desarrollo de la solución implica abordar una serie de desafíos, descritos a continuación.

En primer lugar, el uso de placas ESP32 requiere de conocimientos específicos para maximizar el rendimiento y la eficiencia durante la ejecución. FreeRTOS es fundamental en este contexto, ya que comprender su uso y los conceptos que lo sustentan permiten una correcta ejecución paralela de múltiples tareas en este microcontrolador, limitado por su capacidad de procesamiento. A su vez, requiere conocimientos del lenguaje C++ y de las herramientas para desarrollar código capaz de usar y liberar adecuadamente la memoria durante la ejecución. Por otro lado, el protocolo de comunicación utilizado, ESP-NOW, presenta sus propias limitaciones, como la cantidad de pares conectados simultáneamente y el tamaño máximo de los mensajes, las cuales deben ser consideradas al utilizarla.

Por otro lado, este nuevo protocolo, al basarse en el ya existente MQTT, requiere un conocimiento profundo de su funcionamiento. Esto incluye el formato y los tipos de mensajes, las comprobaciones que se realizan al enviar mensajes, y el orden de ejecución y procesamiento, entre otros aspectos. Sin embargo, esto no limita la herramienta a ser una réplica fiel, ya que se deben diseñar componentes y funcionalidades del protocolo con suficiente libertad creativa, pero manteniendo una operación similar a MQTT.

Finalmente, PlatformIO, a parte de permitir el desarrollo en las mencionadas placas, ofrece una serie de plantillas y comandos esenciales para desarrollar una librería. Esto implica realizar una lectura exhaustiva de la documentación de esta plataforma, junto con diversas pruebas, para disponer de una librería funcional.

1.3. Objetivos

En este apartado se detallan el objetivo general y los subobjetivos específicos del proyecto, utilizados para guiar su desarrollo y evaluar la adecuación de los resultados obtenidos.

1.3.1. Objetivo general

El objetivo general del proyecto es el diseño y la implementación en forma de librerías de un protocolo de mensajería publicador-subscriptor inspirado en MQTT y que haga uso del protocolo de comunicación

inalámbrica ESP-NOW.

1.3.2. Objetivos específicos

La consecución del objetivo general se descompone en una serie de objetivos específicos esenciales descritos a continuación.

- a. **Diseñar un protocolo de mensajería para redes ESP-NOW que incorpore características de publicación y suscripción similares a MQTT.** Este objetivo inicial es fundamental para los siguientes, ya que pretende definir las características necesarias del protocolo conteniendo las bases de la mensajería a través de colas y temas. Esto incluye la definición de roles de los nodos (cliente y broker), las acciones que pueden realizar (publicar, suscribirse y desuscribirse), los tipos y formatos de los mensajes intercambiados, así como la secuencia de procesamiento de los mismos.
- b. **Implementar librerías de software que permitan a los desarrolladores integrar fácilmente el protocolo propuesto en sus aplicaciones del Internet de las Cosas.** Una vez diseñado el protocolo, se debe proceder a su implementación práctica. Esto implica la creación de una librería pública que cumpla con las especificaciones de dicho protocolo, proporcionando funciones sencillas de utilizar y bien documentadas, facilitando su integración por parte de los desarrolladores.
- c. **Presentar demostraciones o casos de uso que ilustren la utilidad y aplicabilidad del protocolo en contextos reales del Internet de las Cosas.** Como parte final del proyecto, demostrar su funcionamiento y relevancia a través de ejemplos prácticos y proveer instrucciones detalladas para utilizar la librería resultante es esencial.

1.4. Estructura del documento

/TODO: redactar/

Capítulo 2

Estado del arte

En este capítulo se aborda el concepto de Internet de las Cosas, incluyendo su valor, los desafíos que presenta y los tipos de dispositivos que lo conforman. Sumado a esto, se comparan los distintos protocolos, middleware y tecnologías empleados en el transporte de datos en este ámbito, y se menciona la popular placa ESP32, ampliamente utilizada en el Internet de las Cosas.

2.1. Internet de las Cosas

El siglo XX dio lugar al desarrollo de numerosos inventos que impulsaron una revolución y un avance ágil en la sociedad, que en la actualidad son frecuentemente usados y facilitan la vida humana. Este es el caso del Internet, que desde 1969 ha permitido la comunicación entre personas y el acceso a información. Hoy en día, también millones de objetos están conectados a la red, cuyas funcionalidades dependen de esta.

Internet de las Cosas o Internet of Things (**IoT**) es un término utilizado por primera vez en un discurso realizado en septiembre de 1985, cuando Peter T. Lewis predijo que “no sólo los humanos, sino también las máquinas y otras cosas se comunicarán interactivamente a través de Internet” /**TODO: mencionar al podcast/**; también empleado por Kevin Ashton en 1999 al realizar una presentación del uso de etiquetas de identificación por radiofrecuencia, o radio frequency identification (**RFID**) en inglés, y otros sensores en los productos de la cadena de suministro. Sin embargo, es esencial tener una clara comprensión del significado del **IoT**, en mayor parte por la confusión inherente al término en sí y por las aplicaciones cotidianas de esta tecnología.

A primera vista, “Internet de las Cosas” podría dar la impresión de ser un término moderno para referirse a “conectar a Internet algo para controlarlo”, una definición bastante simple para alguien que por ejemplo controla las luces de su hogar desde su teléfono móvil. Es tanta la confusión que no existe una definición formal única, sino que se pueden encontrar una disparidad de definiciones que no todos los sistemas **IoT** cumplen. En 2012, la Unión Internacional de Telecomunicaciones (**ITU**) recomendó definir el **IoT** como una infraestructura global que permite ofrecer servicios avanzados a todo tipo de aplicaciones conectando objetos entre sí e interoperando tecnologías de la información y comunicación, aprovechando capacidades de identificación, obtención de datos, procesamiento y comunicación y cumpliendo con requisitos de segu-

ridad y privacidad. Sin embargo, por lo general, podemos entender que el IoT trata de dotar de capacidades de comunicación además de procesamiento, captura y/o análisis de datos a distintos tipos de entes, como dispositivos físicos, objetos, edificaciones, terrenos, sistemas, hardware, software, e incluso contextos y situaciones, ya sea añadiéndoles dispositivos o integrando ciertas capacidades en los propios objetos. Estos entes pueden estar compuestos de sensores, que recopilan datos, o actuadores, que controlan otros objetos, y a través de redes, privadas o públicas, pueden intercambiar información con otros dispositivos, recopilar la información en un mismo dispositivo y transferir órdenes.

Con el objetivo de ilustrar cómo funciona un sistema IoT, la Figura 2.1 muestra un sistema de riego por aspersión inteligente en un jardín. El jardín está dividido en zonas, cada una con un dispositivo compuesto por sensores de temperatura y humedad y actuadores que activan y desactivan los aspersores que riegan la vegetación. Estos dispositivos están conectados y se comunican con un único gateway o puerta de enlace, teniendo la capacidad de recibir información de los dispositivos y mandar órdenes a estos. El gateway está comunicado a través de Internet con un servidor compuesto por una base de datos, donde se almacenan los datos históricos y registros, y un servicio que le permite ser controlado desde otro dispositivo conectado a Internet en cualquier sitio y momento. Este último dispositivo, denominado cliente, puede ser un teléfono móvil o un ordenador, y puede utilizarse para ver el estado del jardín y controlar los dispositivos manualmente desde una interfaz. La siguiente arquitectura, abstraída completamente de las limitaciones y los problemas que puede ofrecer su implementación, podrá dar lugar a dos casos de uso:

- El usuario quiere activar los aspersores. El usuario, desde la interfaz el dispositivo cliente (como puede ser un botón, por ejemplo), enviará la orden de activar los aspersores al servicio, que a su vez se lo enviará al gateway. El gateway enviará una orden compatible a los dispositivos instalados en el jardín, haciendo que los aspersores comiencen a funcionar.
- Los aspersores funcionan cuando la temperatura es muy elevada y la humedad es baja. El usuario previamente, desde la interfaz del cliente, ha establecido que los aspersores funcionen de manera automática cuando, por ejemplo, el ambiente supere una temperatura de 32 °C y la humedad sea considerada baja. Estos parámetros los recibe el servidor y se los pasa al gateway, el cual los recordará. A partir de ese momento, el gateway irá recibiendo de los sensores de cada zona lecturas de temperatura y humedad, y las comparará con los parámetros establecidos. En el caso de que se superen la temperatura y humedad, se activarán los aspersores de la zona. Una vez activados, se mantendrá la lectura de temperatura y humedad, y cuando las lecturas sean inferiores, se desactivarán los aspersores. Además, con cada activación de los aspersores, el gateway notificará al servicio, que a su vez notifica al usuario, especificando la zona activada.

Cabe recalcar que el mencionado ejemplo puede volverse más complejo, por ejemplo añadiendo comprobaciones de previsión meteorológica en los próximos días, sensores de luz, control del caudal del agua, pero igualmente cumple con los requisitos para ser un sistema que implementa el IoT, ya que dota a un jardín de capacidad de comunicación, captura y análisis de datos, y se comunica con otros dispositivos y con personas para informar del estado y recibir órdenes.

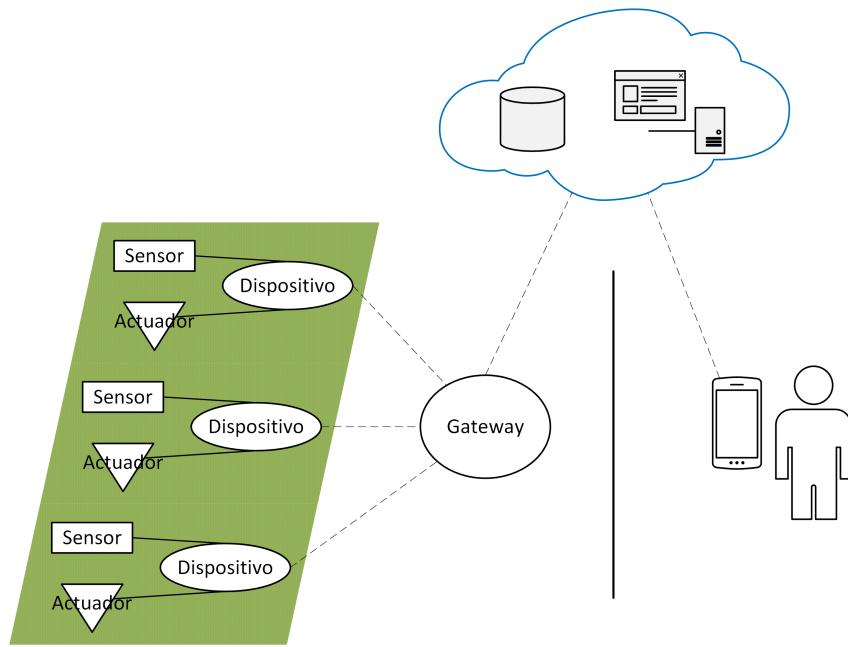


Figura 2.1: Esquema de un sistema de riego por aspersión utilizando dispositivos IoT

2.1.1. Ventajas y usos

Hoy en día vivimos en un mundo conectado, en el cual estamos rodeados de numerosos dispositivos, gran parte de ellos interconectados como sistemas, ya que la información es crucial para la sociedad. Por ejemplo, a nivel empresarial es la que permite crear nuevas posibilidades, modelos, interacciones y soluciones únicas, entender y anticiparse a tendencias e identificar nuevas oportunidades. Esta información se obtiene a partir del procesamiento y análisis de colecciones de datos, recopilados de diversas fuentes potenciales, como es el caso de los sistemas IoT.

Gracias al IoT es posible monitorear los procesos que ocurren en un negocio para controlar y optimizar los recursos utilizados, ya sean energéticos o de personal humano; además de encontrar tareas manuales o repetitivas que pueden ser automatizadas mediante el propio IoT, mejorando así la eficiencia, el tiempo y la productividad. Asimismo, permite la toma de decisiones informadas a partir de datos de negocio, tanto internos como externos, que permitan agregar valor al negocio de manera más rápida, y la mejora de experiencia y de calidad de vida de los usuarios o clientes basándose en los datos recibidos sobre estos.

En adición, tiene la capacidad de implementarse de diferentes maneras en una gran variedad de entornos y sectores, tales como:

- Sanidad: monitorizar signos vitales, hacer recomendaciones de salud adecuadas, comprobar autenticidad y dosis recomendadas en medicamentos.
- Agricultura y ganadería: controlar las condiciones del suelo y el crecimiento de los cultivos, localizar el ganado, controlar la calidad de la leche recolectada en un tanque.
- Transporte, conducción y logística: frenado automático de emergencia, cálculo de rutas óptimas, búsqueda de aparcamiento, mantener condiciones óptimas de remolques que transporten productos

perecederos.

- Fabricación: controlar temperatura y humedad de centros de fabricación, identificar áreas con necesidad de personal, crear controles de calidad a productos.
- Comercio: identificar posición óptima de los productos en las estanterías, guiar a clientes hasta determinados productos, trazabilidad en el producto desde la llegada en el almacén hasta su venta.
- Ciudades y edificios inteligentes: mejor vigilancia y actuación por los equipos de emergencia, localizar áreas de contaminación, automatizar iluminación y calefacción.

El IoT no solo se reduce a estos sectores, es prácticamente aplicable a todos, como la educación, la construcción o el turismo, y a cualquier fase, desde el diseño hasta ofrecer y consumir el servicio o producto, gracias a la popularización de esta tecnología y al aumento de dispositivos que aparecen a nuestro alrededor.

{ https://es.wikipedia.org/wiki/Internet_de_las_cosas <https://www.ibm.com/es-es/topics/internet-of-things>
<https://blog.ubisolutions.net/es/iot-tecnologias-casos-de-uso-ventajas-y-limitaciones-guia-completa>
<https://www.tokioschool.com/noticias/ventajas-desventajas-internet-cosas/> <https://kryptonsolid.com/principales-ventajas-y-desventajas-de-iot-en-los-negocios/> <https://www.campusbigdata.com/big-data-blog/item/101-relacion-iot-con-big-data> <https://www.ibm.com/es-es/topics/internet-of-things> <https://www.nougrir.com/index.php/blog-3/item/13-que-es-iot-o-internet-de-las-cosas-y-sus-aplicaciones> <https://innovayaccion.com/blog/aplicando-el-internet-de-las-cosas-a-las-empresas-2> https://cdn.ihs.com/www/pdf/IoT_ebook.pdf <https://www2.deloitte.com/content/dam/RealEstate/deloitte-nl-fsi-real-estate-smart-buildings-how-iot-technology-aims-to-add-value-for-real-estate-companies.pdf> https://www.researchgate.net/publication/325373920_Internet_of_things_IoT_a_survey_on_architecture_enabling_technologies_and_challenges }

2.1.2. Factores, limitaciones y desafíos a tener en cuenta

Al desarrollar una arquitectura del IoT, es necesario tener en cuenta una serie de factores que afectan tanto a los dispositivos como a la comunicación entre ellos.

La primera consideración se refiere a los datos, el artefacto más importante en estos sistemas. Deben ser adecuados y el personal debe estar preparado para tratarlos y analizarlos. Sin embargo, debido a la gran cantidad de datos producidos, se presenta un reto significativo al analizarlos y extraer información, especialmente en casos de falta de herramientas o experiencia en análisis.

Dependiendo del dispositivo, el uso que se le da y el entorno donde se instala, puede verse limitado por varios factores energéticos y físicos interdependientes, tales como la manera de alimentarlo, el consumo (coste de la energía y tiempo de uso), el rango de movilidad, el peso y el tamaño. Dar preferencia a un factor puede perjudicar a otro, por lo que es necesario buscar un equilibrio que satisfaga todas las necesidades o priorizar algunas sobre otras y aplicar cambios en las propias necesidades.

Las condiciones ambientales (como la lluvia, la humedad, las vibraciones y la temperatura), la disponibilidad de conexión en una zona (cobertura de red o electrificación), las interferencias electromagnéticas y otros factores del área donde se despliega el sistema pueden provocar un replanteamiento de ciertas partes. Por ejemplo, éstas pueden implicar la fiabilidad en el transporte de los datos mediante mecanismos de acuses de recibo y verificación de mensajes, o la frecuencia de transmisión y tamaño de los datos.

Debido a la heterogeneidad de los dispositivos en cuanto a fabricantes, características y capacidades, y a la falta de un estándar internacional de compatibilidad para dispositivos en IoT, es necesario dedicar un esfuerzo a identificar las tecnologías de comunicación compatibles (y que cumplan las necesidades de velocidad y alcance), configurar los dispositivos y usar dispositivos adicionales si es necesario.

Es importante además adaptar el despliegue al uso que se le vaya a dar, como puede ser tener una topología de conexión y comunicación adecuada según los dispositivos que interactúen, prepararlo para una futura escalabilidad, habilitar la gestión remota de los dispositivos para actualizarlos, reconfigurarlos, localizarlos o desconectarlos, y realizar una correcta recolección de los datos.

Como consideración final, los datos transmitidos pueden contener cierto grado de sensibilidad, por lo que es preferible mantener una buena ciberseguridad y respetar la privacidad de los datos implementando medidas que aseguren la confidencialidad, la integridad y la disponibilidad, siendo importante conocer las regulaciones de la zona en la que opera el sistema IoT para poder cumplirlas.

{ “IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things”. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Jerome Henry, Robert Barton <https://www.redeweb.com/articulos/sensores/los-sensores-inalambricos-iot-y-el-problema-de-la-corta-duracion-de-las-baterias> <https://neurona-ba.com/iot-a-tu-alcance/> <https://www.itrends.es/infraestructura/2021/12/los-problemas-de-conectividad-estan-dificultando-los-despliegues-de-iot> <https://deingenierias.com/curso/iot/1-4-desafios-y-limitaciones-de-iot/> <https://www.chakray.com/es/5-requesitos-de-una-arquitectura-iot/> <https://www.incibe.es/incibe-cert/blog/riesgos-y-retos-ciberseguridad-y-privacidad-iot> https://es.wikipedia.org/wiki/Internet_de_las_cosas <https://www.ibm.com/es-es/topics/internet-of-things> <https://blog.ubisolutions.net/es/iot-tecnologias-casos-de-uso-ventajas-y-limitaciones-guia-completa> <https://www.tokioschool.com/noticias/ventajas-desventajas-internet-cosas/> <https://kryptonsolid.com/principales-ventajas-y-desventajas-de-iot-en-los-negocios/> }

2.1.3. Primeros ejemplos de IoT

El primer dispositivo IoT conocido fue una máquina de Coca-Cola conectada a ARPANET a principios de la década de los 80 en la Universidad Carnegie Mellon de Pittsburgh, Pensilvania. Esta máquina era mantenida por los usuarios de la universidad y tenía cierta popularidad que hacía que normalmente estuviera vacía o recién cargada con botellas calientes, lo que molestaba al departamento de Ciencias de la Computación. Para resolver este problema, instalaron microinterruptores en la máquina para detectar las botellas que había en cada ranura, los conectaron al ordenador Programmed Data Processor model 10 (PDP-10) del departamento y diseñaron un programa para registrar la hora cada vez que ocurría una transacción en cada ranura, así como otro programa para recibir información de la máquina. Utilizando el último programa desde la red del departamento, se podía consultar si había botellas en las ranuras, cuánto tiempo había transcurrido desde que se recargó con una nueva botella o si estaba fría.

Más tarde, hicieron público el acceso a esta información, haciendo que cualquier persona conectada a ARPANET en cualquier parte del mundo podía consultar el estado de la máquina de Coca-Cola durante más de una década utilizando el comando `finger coke@cmua`, hasta que la máquina fue reemplazada

por ser incompatible con el nuevo diseño de las botellas de Coca-Cola.

Incluir imágenes: - Xerox Alto, ordenador fabricado en 1973, usado en la CMU para ver el estado de la maquina

- Máquina de cocacola <https://engineercom.wpenginepowered.com/wp-content/uploads/2016/02/Iiot1.png>
- PDP-10

Posteriormente, conectado a un Internet más similar al actual, apareció XCoffee o la cafetera de la Sala Trojan. En noviembre de 1991, Quentin Stafford-Fraser y Paul Jardetzky trabajaban junto a sus compañeros investigadores agrupados en el antiguo laboratorio de informática, también llamado Sala Trojan, de la Universidad de Cambridge, mientras que el resto de compañeros estaban dispersos por la universidad. Todos compartían un problema en común: una única máquina de café de goteo-filtro a la salida del laboratorio, cuya jarra estaba vacía o contenía café con un sabor horrible a excepción de cuando estaba recién hecho, siendo esta última opción solo para aquellos que se sentaban cerca. Para solucionar este problema, Stafford-Fraser y Jardetzky idearon el sistema XCoffee, y el Dr. Martyn Johnson y Daniel Gordon lo mejoraron. Fijaron una cámara apuntando a la jarra de la cafetera, la conectaron a una capturadora de fotogramas del ordenador Acorn Archimedes del rack de la sala y crearon un software para que cada vez que el servidor recibía una solicitud HTTP a través de la World Wide Web devolvía una página web con la imagen de la cafetera capturada más reciente. Desde el navegador se podía comprobar si valía la pena ir a por café (en el caso de estar en el mismo edificio que la cafetera), y esto convertía al sistema en la primera webcam de la historia, otorgándole una fama internacional. Estuvo operativa su última versión desde el 22 de noviembre de 1993 hasta el 22 de agosto de 2001 a las 09:54 UTC, cuando mostró su última imagen pocos segundos antes de su apagado debido a su difícil mantenimiento y al traslado de las instalaciones del laboratorio.

Actualmente, la máquina de café se encuentra restaurada y expuesta en el museo de informática Heinz Nixdorf MuseumsForum de Paderborn, Alemania.

<https://www.nougar.com/index.php/blog-3/item/13-que-es-iot-o-internet-de-las-cosas-y-sus-aplicaciones>

<https://informationmatters.net/internet-of-things-definitions/> (DOCUMENTO) <https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=11559&lang=es> (LIBRO)

[https://www.taylorfrancis.com/chapters/oa-edit/10.1201/9781003337584-3/internet-things-cognitive-transformation-technology-research-trends-applications-ovidiu-vermesan-markus-eisenhauer-harald-sundmaeker-patrick-guillemain-martin-serrano-elias-tragos-javier-vali %C3%B1o-arthur-van-derwees-alex-gluhak-roy-bahr](https://www.taylorfrancis.com/chapters/oa-edit/10.1201/9781003337584-3/internet-things-cognitive-transformation-technology-research-trends-applications-ovidiu-vermesan-markus-eisenhauer-harald-sundmaeker-patrick-guillemain-martin-serrano-elias-tragos-javier-vali-%C3%B1o-arthur-van-derwees-alex-gluhak-roy-bahr) (LIBRO)

[https://books.google.es/books/about/La_R_\(DOCUMENTO\).html?hl=es&gbpv=1&oi=fnd&pgis=1](https://books.google.es/books/about/La_R_(DOCUMENTO).html?hl=es&gbpv=1&oi=fnd&pgis=1) (LIBRO)

https://cdn.ihs.com/www/pdf/IoT_ebook.pdf (DOCUMENTO) <https://www.researchgate.net/publication/352544777> (DOCUMENTO)

<https://www.microsoft.com/insidetrack/blog/transforming-microsoft-buildings-with-iot-technology-and-indoor-mapping/> (LIBRO) <https://www.se.com/us/en/download/document/998-20233517/> (DOCUMENTO)

https://www.researchgate.net/publication/320135661_Efficient_IoT-based_Sensor_BIG_Data_Collection-Processing_and_Analysis_in_Smart_Buildings (DOCUMENTO)

<https://www2.deloitte.com/content/dam/Deloitte/nl/Documents/estate/deloitte-nl-fsi-real-estate-smart-buildings-how-iot-technology-aims-to-add-value-for-real-estate-companies.pdf>

<https://innovayaccion.com/blog/aplicando-el-internet-de-las-cosas-a-las-empresas-2> (DOCUMENTO)

<https://ieeexplore.ieee.org/document/7879243> <https://www.intel.com/content/www/us/en/internet-of-things/overview.html>

<https://www.fundacionbankinter.org/fif-informes/internet-de-las-cosas/> <https://www.statista.com/statistics/number-of-connected-devices-worldwide/> (VIDEO)

<https://www.youtube.com/watch?v=RnasX1bFBh8>

(LIBRO) <https://www.amazon.com/IoT-Fundamentals-Networking-Technologies-Protocols/dp/1587144565>
 (LIBRO) https://www.ra-ma.es/libro/internet-de-las-cosas_93304/ https://es.wikipedia.org/wiki/Internet_de_las_cosas
<https://www.eexcellence.es/expertos/kevin-ashton-un-tecnologo-visionario>
<https://www.redhat.com/es/topics/internet-of-things/what-is-iot>
<https://at3w.com/blog/iot-internet-of-things-tecnologia-proteccion-contra-rayo-tomas-tierra/>
<https://www.itop.es/blog/item/iot-origen-importancia-en-el-presente-y-perspectiva-de-futuro.html>
<https://www.linkedin.com/pulse/el-origen-del-internet-de-las-cosas-iot-comnet-s-a-/>
<https://www.ui1.es/blog-ui1/historia-y-origen-del-iot>
<https://blog.avast.com/es/kevin-ashton-named-the-internet-of-things>
<https://www.datacenterdynamics.com/en/podcasts/zero-downtime/episode-18-the-origin-of-the-internet-of-things-with-peter-lewis/> <https://swifftalk.net/2021/10/06/the-concept-of-iot-internet-of-things/>
<https://www.cs.cmu.edu/~coke/> https://www.cs.cmu.edu/~coke/history_long.txt <https://www.ibm.com/docs/es/aix/7.1?topic=protocol-namefinger-protocol> https://en.wikipedia.org/wiki/John_Romkey <https://romkey.com/about/> <https://blog.avast.com/the-internets-first-smart-device> https://en.wikipedia.org/wiki/Trojan_Room_coffee_pot <https://www.cl.cam.ac.uk/coffee/qsf/coffee.html> <https://www.cl.cam.ac.uk/coffee/coffee.html> https://www.youtube.com/watch?v=uF982_aRKrI <https://www.bbc.com/news/technology-20439301> <https://www.historyofinformation.com/detail.php?id=1507> <https://quentinsf.com/coffeepot/metcalfe/> <https://www.cl.cam.ac.uk/coffee/qsf/switchoff.html> <https://owl.museum-digital.de/object/3761>

2.1.4. Tipos de dispositivos y redes

Como se ha mencionado anteriormente, el **IoT** se basa en la transmisión de datos entre dispositivos. Estos datos pueden ser información recibida del entorno u órdenes de actuación, que salen o llegan a un dispositivo ubicado en un extremo de la red de conexión y que tiene conectado uno o ambos de los siguientes tipos de objetos:

- Sensores: miden una propiedad física y la representan digitalmente, siendo útiles para detectar cambios en el entorno.
- Actuadores: dispositivos que interpretan una señal eléctrica y desencadenan un efecto físico tras interpretarla, siendo útiles para producir cambios en el entorno.

Todos estos dispositivos suelen ser heterogéneos y están interconectados en una red que permite detectar, medir y actuar en relación con el entorno. Esta red, con el fin de ser efectiva y tolerante a fallos, debe pasar por un análisis de criterios como la topología de conexión entre los dispositivos, la distancia entre estos y cómo se procesan los datos.

Una red **IoT**, puede dividirse en capas según cómo se procesen los datos [49] [58]:

- Computación en la nube o *cloud*: los datos se procesan en un servidor central remoto, lejos de su origen. Requiere una conexión a Internet para su acceso, lo cual implica lidiar con la latencia de la red y el uso del ancho de banda. Ofrece una gran capacidad de procesamiento, almacenamiento y análisis, adecuada para aquellas aplicaciones que no requieren respuestas rápidas. Puede agrupar y procesar los datos de todo un sistema.
- Computación en la niebla o *fog*: los datos se procesan cerca de donde se originan, en nodos *fog* de la

red local. Estos nodos evitan el uso de Internet para enviar datos a la nube, mejorando la eficiencia y ofreciendo una respuesta más rápida que la computación *cloud*. Se pueden distribuir varios nodos *fog* para jerarquizar la red, agrupandos y procesando datos de ciertos nodos.

- Computación en el borde o *edge*: los datos se procesan en los nodos donde se originan. Ofrece un procesamiento en tiempo real sin necesidad de comunicarse con otros dispositivos, liberando el uso de ancho de banda.

{ <https://www.ibm.com/es-es/topics/internet-of-things> (LIBRO, el de los apuntes) “IoT fundamentals: Networking technologies, protocols, and use cases for the internet of things”. David Hanes, Gonzalo Salgueiro, Patrick Grossetete, Jerome Henry, Robert Barton. <https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-personal-area-network/> <https://www.cloudflare.com/es-es/learning/network-layer/what-is-a-lan/> <https://www.cloudflare.com/learning/network-layer/what-is-a-metropolitan-area-network/> <https://www.cloudflare.com/learning/network-layer/what-is-a-wan/> <https://www.gadae.com/blog/tipos-de-redes-informaticas-segun-su-alcance/> }

2.2. Middleware y protocolos de mensajería usados en IoT

El uso más común de IoT es desplegar una arquitectura compuesta por varios dispositivos IoT. En mayor parte, estos dispositivos se designarán simplemente como dispositivos IoT, ya sean sensores o actuadores, que emiten o reciben datos. Adicionalmente, habrá unos pocos dispositivos (al menos uno) con el rol de centro de mensajería, que actuarán como intermediarios encargados de retransmitir los datos a los dispositivos correspondientes.

Tras escoger una tecnología para conectar los dispositivos entre sí, en el desarrollo de aplicaciones IoT se requiere un protocolo de mensajería para intercambiar esos grandes volúmenes de datos que tratan. Un protocolo de comunicación permite que los dispositivos se comuniquen y transmitan mensajes entre los dispositivos IoT y el centro de mensajería. Además, proporciona cierta fiabilidad a la comunicación, ya que permite que los mensajes lleguen y sus datos sean entendidos y procesados correctamente.

La elección del protocolo se basa en cómo se adapta al escenario en el que se quiere implementar, considerando requisitos a tener en cuenta como la ubicación, las limitaciones físicas, el consumo y el coste. Por lo general, no cualquier protocolo de comunicación es apropiado. Los protocolos que se mencionan en este apartado se adecuan a la mayoría de escenarios IoT debido a su rapidez y su fácil implementación, y es posible escoger aquel que se adapte mejor a los requisitos.

2.2.1. Message Queue Telemetry Transport (MQTT)

El protocolo MQTT es uno de los más populares en el ámbito del IoT. Diseñado para ser ligero y adecuado para redes con ancho de banda limitado y dispositivos con pocos recursos, este estándar del comité técnico OASIS permite el transporte bidireccional de mensajes con datos entre múltiples dispositivos.

MQTT utiliza el patrón de comunicación publicación-suscripción. En este patrón, los publicadores categorizan los mensajes, y los suscriptores recibirán mensajes de las categorías de su interés, a diferencia de la

comunicación directa en la que los participantes se envían los mensajes directamente. En el caso de **MQTT**, el patrón está basado en temas o *topics*, siendo posible que los suscriptores se interesen por uno o varios.

En una red **MQTT**, se definen dos roles principales: el broker o intermediario de mensajes y los clientes. El broker **MQTT** es un servidor comparable a una oficina de correos, que recibe todos los mensajes publicados por los clientes y los dirige a los clientes de destino apropiados. Por otra parte, un cliente es cualquier dispositivo conectado al broker a través de una red, y puede producir y recibir datos al publicar y suscribirse respectivamente. Este mecanismo es útil para compartir datos, controlar y gestionar dispositivos. El enruteamiento de mensajes realizado por el broker proporciona transparencia de localización y desacoplamiento en el espacio, ya que el publicador no necesita conocer las direcciones de los suscriptores y los suscriptores no necesitan conocer al publicador, ambos interactúan únicamente con el broker.

Los mensajes están organizados en una jerarquía de temas o *topics*. Al publicar un mensaje, se publica en un tema específico, y en el caso de querer publicar en varios se deben realizar varias publicaciones. En cambio, un suscriptor puede suscribirse a un tema específico o a varios simultáneamente y recibirá una copia de todos los mensajes compatibles con los temas suscritos. La manera de indicar varios temas es mediante el uso de los siguientes caracteres comodín:

- Comodín de un nivel '+': coincide con un nivel de tema y puede utilizarse más de una vez en la especificación del tema.
- Comodín de varios niveles '#': coincide con cualquier número de niveles y debe ser el último carácter en la especificación del tema.

Por ejemplo, cuando se publica un mensaje en el tema “edificioA/sensor1/temperatura”, el broker enviará una copia del mensaje los clientes suscritos a los temas “edificioA/sensor1/temperatura”, “edificioA/+/temperatura” y “edificioA/#”, pero no a un cliente suscrito a “edificioB” o a “edificioA/+/humedad”.

La transmisión de mensajes se realiza de forma asíncrona, sin detener la ejecución de ambos componentes a la hora de publicar o recibir, y se puede realizar una comunicación uno a muchos (un publicador y varios suscriptores), muchos a uno (varios publicadores y un suscriptor), uno a uno (un publicador y un suscriptor, menos común) y muchos a muchos (varios publicadores y varios suscriptores).

En el caso de que el broker reciba una publicación de un tema en el cual no hay nadie suscrito, el broker por defecto descarta el mensaje. Es posible activar la retención de mensajes configurando un campo en el mensaje para evitar esto, consiguiendo así que el broker almacene el último mensaje retenido de cada tema y lo distribuya inmediatamente a cualquier nuevo cliente suscrito, permitiendo así que el suscriptor reciba el valor más reciente en lugar de esperar a una nueva publicación, y además añadiendo soporte a una comunicación desacoplada en el tiempo, donde publicadores y suscriptores no necesitan estar conectados simultáneamente.

El protocolo soporta un mecanismo de limpieza de sesión. Por defecto, un cliente tras desconectarse y volverse a conectar no recibe los mensajes publicados durante su desconexión y el broker olvida las suscripciones del mismo cliente. Pero al desactivar dicho mecanismo, el broker mantiene tanto las relaciones

de suscripción como los mensajes offline, enviándolos al cliente al momento de reconectarse, lo cual es útil para dispositivos que se conectan y desconectan constantemente, común en redes IoT. Además, MQTT enfrenta la inestabilidad de la red con un mecanismo *Keep Alive* que, al transcurrir un prolongado periodo sin interacción, ocurre un ping entre el cliente y el broker para evitar la desconexión. Si el ping falla y se identifica el cliente como desconectado, aplicará un mecanismo *Last Will*, que publica un último mensaje a un tema específico debido a una desconexión anormal, en el caso de estar configurado.

MQTT dispone de 14 tipos de mensajes diferentes, utilizados para establecer y cerrar una conexión con el broker, comprobar si la conexión sigue viva, publicar mensajes, y suscribirse y desuscribirse a temas.

El diseño de MQTT se basa en la simplicidad y en minimizar el ancho de banda, dejando la interpretación del contenido del mensaje en manos del desarrollador. Los mensajes transmitidos a través de la red tienen la posibilidad de configurar el QoS o calidad de servicio por cada tema, asociados con distintas garantías de entrega. Aunque MQTT funciona sobre TCP, el cual tiene su propia garantía de entrega, históricamente los niveles QoS eran necesarios para evitar la pérdida de datos en redes antiguas y poco fiables, una preocupación válida para las redes móviles actuales.

La transmisión de datos se realiza principalmente sobre la capa TCP/IP, pero existe la posibilidad de operar encima de otros protocolos de red que ofrezcan conexiones ordenadas, sin pérdidas y bidireccionales, y se transmiten en un tamaño reducido de paquetes de datos, estructurado por los siguientes campos mostrados en la Figura 2.2:

- Cabecera fija, en la que se especifica el tipo de mensaje, si el mensaje es un duplicado, el QoS, si es un mensaje que retener y el tamaño del paquete.
- Cabecera variable, no siempre presente en los mensajes, y puede transportar información adicional de control.
- Payload o carga útil.

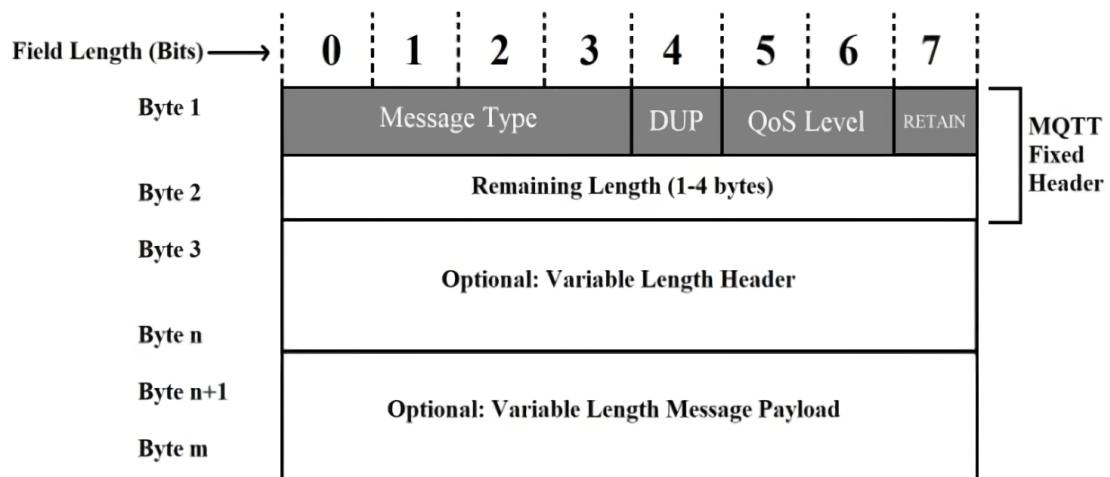


Figura 2.2: Formato de una trama MQTT (Fuente: [52])

Por defecto, este protocolo envía credenciales y mensajes en texto plano sin medidas de seguridad, pero

admite utilizar conexiones **TLS/SSL** protegidas por certificado, nombre de usuario y contraseña para cifrar y proteger la información transferida contra la intercepción, modificación o falsificación. Además, un broker **MQTT** tiene soporte para conectar dispositivos IoT a escala masiva, un factor tenido en cuenta durante su diseño y que resulta en una alta capacidad de concurrencia, rendimiento y escalabilidad, características deseables en una red **IoT**. Implementaciones como EMQX¹ y HiveMQ² han alcanzado hitos notables, con 100 y 200 millones de conexiones respectivamente, y un pico de 1 millón de mensajes gestionados por segundo. A esto se le puede sumar la capacidad de implementar múltiples brokers, para así compartir la carga de clientes y tratar la redundancia y la copia de seguridad de los mensajes en caso de fallo.

```
{ https://en.wikipedia.org/wiki/MQTT https://en.wikipedia.org/wiki/Publish_%E2%80%93subscribe_pattern
https://www.emqx.com/en/blog/what-is-the-mqtt-protocol https://www.gotiot.com/pages/articles/mqtt_intro/index.html
https://dzone.com/refcardz/getting-started-with-mqtt https://www.hivemq.com/resources/achieving-200-mil-concurrent-connections-with-hivemq/ http://www.steves-internet-guide.com/mqtt-protocol-messages-overview/ }
```

2.2.2. Advanced Message Queuing Protocol (**AMQP**)

AMQP es un protocolo binario avanzado que opera sobre la capa de aplicación, cuyo estándar abierto permite desarrollar mensajería y patrones de comunicación entre dispositivos. Facilita la comunicación entre servicios definiendo el formato de los datos enviados a través de la red como un flujo de bytes, así como la creación de mensajes, el encolamiento y enrutamiento de los mensajes producidos, y la manera de entregarlos a los consumidores.

Este protocolo se basa en el concepto de publicar y consumir de colas de mensajes a través de una conexión fiable, persistente y orientada al envío de flujos de datos. Además, es compatible con el envío de múltiples flujos de datos simultáneos mediante múltiples canales en una única conexión. Permite hacer uso de estas colas de mensajes mediante distintos tipos de comunicación, como la entrega directa punto a punto y el modelo publicación-suscripción, y, pese a no ser un protocolo diseñado originalmente para su uso en el **IoT**, funciona muy bien en este ámbito y en la gran variedad de escenarios de comunicación y mensajería posibles.

En **AMQP** se definen las dos siguientes entidades principales que interactúan entre sí:

- Cliente: del tipo suscriptor o publicador (o consumidor y productor, respectivamente), se conecta a un broker a través de credenciales y, en caso de estar autorizado, puede recibir o publicar mensajes.
- Broker: servidor de mensajería al que los clientes se conectan y que se encarga de distribuir los mensajes. Internamente, posee *exchanges* o intercambiadores, donde se conectan los productores de mensajes, y colas, vinculadas a los exchanges dependiendo de varios criterios y a las que se conectan los consumidores para extraer los mensajes producidos.

De manera sencilla, se puede resumir el funcionamiento de este protocolo como un modelo en el que los mensajes son publicados y enviados a exchanges, los cuales enrutan los mensajes a las colas apropiadas

¹<https://www.emqx.com/en>

²<https://www.hivemq.com/>

según reglas o bindings, y los consumidores reciben los mensajes a través de las mismas colas.

Los exchanges, además de recibir mensajes de los productores, se encargan de enviar los datos a las colas apropiadas, ya sean a una o a varias dependiendo del exchange y la clave de enrutamiento o routing key con la que se publica el mensaje. A este funcionamiento se le puede asociar la analogía del funcionamiento del envío de emails, ya que estos se envían a direcciones “routing key@exchange”, siendo la clave de enrutamiento la dirección de correo y el exchange el servidor. En **AMQP** existen cuatro tipos de exchanges: topic (permite una comunicación publicación-suscripción según clave y exchange), direct (permite una comunicación punto a punto), fanout (permite una comunicación similar a broadcast) y headers (permite una comunicación publicación-suscripción según cabeceras).

Las colas son fragmentos de memoria creados por el cliente suscriptor identificadas únicamente mediante un nombre, definido previamente por el cliente o automáticamente por el broker. Intrínsecamente, **AMQP** garantiza la recepción y procesamiento de mensajes, ya que dispone de un mecanismo de **ACK** o acuse de recibo que permite confirmar la recepción y procesamiento del mensaje. En el caso de no recibir el **ACK** de un mensaje por parte de un consumidor, por ejemplo, porque perdió la conexión o no se procesó correctamente, el broker encola de nuevo el mensaje para reintentar la entrega. Además, las colas admiten persistencia, para mantener la existencia de la cola incluso luego de que ocurra un reinicio en el broker.

Tras crear el exchange y la cola, se indica al broker la vinculación de ambas mediante un binding, especificando una clave de enrutamiento que, según el tipo de exchange, enruta adecuadamente las colas. El exchange entregará como máximo una copia de mensaje a una cola si corresponden las propiedades del mensaje con las propiedades del binding. Con los bindings es posible vincular varias colas a un mismo exchange, al igual que una cola a varios exchanges.

Con los tres conceptos mencionados previamente, exchange, binding y cola, se produce un desacople en tiempo y espacio entre los productores y los consumidores, ya que ambos están aislados entre sí y desconocen su existencia y ubicación.

AMQP hace uso de la capa **TCP** para la transmisión de mensajes y es extenso en cuanto a funciones. Ofrece interoperabilidad independientemente del lenguaje o del proveedor, cifrado de extremo a extremo, múltiples propiedades de mensajes y modos de entrega, fiabilidad en la entrega, persistencia de mensajes, enrutamiento basado en criterios, capacidad de escalabilidad y definición de topologías, entre otras características. Sin embargo, pese a su idoneidad para sistemas distribuidos, es un protocolo que conlleva un alto consumo de recursos, como energía y memoria.

2.2.3. Extensible Messaging and Presence Protocol (**XMPP**)

Otra manera de comunicar varios dispositivos **IoT** es utilizar el **XMPP**, anteriormente conocido como Jabber. Este protocolo se basa en la transmisión de datos estructurados en formato **XML** dentro de una red de arquitectura cliente-servidor, en la cual los dispositivos están identificados por un Jabber ID, cuyo formato es similar al de una dirección de correo electrónico (por ejemplo, “abc@example.com”). En esta red, el cliente establece una conexión TCP/IP con el servidor. Posteriormente, el cliente se autentica con el servidor, y tras una autenticación exitosa, se habilita la posibilidad de enviar y recibir mensajes.

Una característica notable de **XMPP** es que cualquiera puede tener su propio servidor **XMPP**, no restringiendo a los usuarios a conectarse únicamente con otros usuarios en el mismo servidor central. Al ser un protocolo abierto formalizado por la Internet Engineering Task Force (**IETF**)³, los desarrolladores disponen de un protocolo bien documentado y fiable, de este modo, es posible interoperar entre diferentes implementaciones de **XMPP** a través de Internet, independientemente del proveedor. En el caso de querer comunicarse con otro servidor, ambos servidores **XMPP** intercambian la información necesaria, habilitando un modelo federado.

Este protocolo está diseñado para ofrecer mensajería instantánea o casi en tiempo real a través de la red, sin importar la distancia entre los dispositivos, uno de los problemas más comunes en **IoT**. Además, permite obtener información sobre los usuarios conectados y mantener una lista de contactos para cada usuario. **XMPP** también es extensible, permitiendo a los desarrolladores añadir características y funcionalidades personalizadas, adaptando el protocolo a necesidades específicas de aplicaciones, como la transmisión de señales **VoIP**, video, ficheros, chat grupal, conferencias multiusuario, suscripción de presencia y comunicación publicación-suscripción para recibir actualizaciones sobre temas específicos de interés.

En los mensajes **XMPP** se utilizan estructuras **XML** denominadas *stanzas* para transportar datos. Existen 3 tipos principales de stanzas: de mensaje (para enviar mensajes entre clientes), de presencia (para conocer el estado y la disponibilidad de un cliente) y de info/query (para hacer consultas al servidor).

El protocolo **XMPP** es altamente escalable debido a su capacidad de manejar multitud de conexiones y mensajes simultáneos. Además, al ser descentralizado, permite implementar fácilmente más servidores para gestionar el aumento de usuarios y altos picos de uso. En cuanto a seguridad, **XMPP** es compatible con cifrado de extremo a extremo mediante **TLS** o **SSL**, garantizando así la confidencialidad de los mensajes. Por último, cuenta con una amplia comunidad de usuarios, diversas implementaciones y guías que facilitan a los desarrolladores la creación de aplicaciones que integren este protocolo.

2.2.4. Data Distribution Service (**DDS**)

DDS es un estándar de middleware y **API** máquina-máquina que facilita la comunicación y el intercambio de datos. Fue desarrollado por el Object Management Group con el fin de responder a las necesidades específicas de aplicaciones que requieren intercambios de datos fiables y de alto rendimiento en sistemas distribuidos en tiempo real, sin dejar de lado la eficiencia. Su arquitectura se basa en un modelo de publicación-suscripción sin servidor, ya que los dispositivos se conectan entre sí, y donde los datos son publicados en un dominio y los suscriptores se conectan a este para recibir la información que les interesa.

Este middleware se corresponde con la capa de software que se encuentra entre el sistema operativo y las aplicaciones, abstrayendo la comunicación entre ambos y, por tanto, los detalles del transporte de red y de los datos a bajo nivel. Permite que los distintos componentes de un sistema compartan y comuniquen datos, gestionando automáticamente el formato de los mensajes, el protocolo de transporte a usar, la fiabilidad, la calidad de servicio y la seguridad, y simplificando así el desarrollo. **DDS** se centra completamente en los datos, asegurando un buen transporte e incluyendo información contextual de los mismos, lo que lo hace

³<https://www.ietf.org/>

ideal para el IoT aplicado en entornos industriales.

DDS funciona con el concepto de espacio de datos global, un almacén de datos que a ojos del programador parece una memoria local accedida mediante APIs. Sin embargo, este espacio es solo una ilusión, ya que no existe un lugar donde residan todos los datos, se refiere a una colección de distintos almacenes locales en cada nodo por los cuales se reparten los datos. El programador piensa que está enviando mensajes a un almacén global, pero en realidad DDS envía mensajes a los almacenes locales apropiados.

Aunque este espacio global es tan característico, no se pierde compatibilidad, ya que el middleware es independiente del lenguaje de programación y la plataforma, posibilitando la interoperabilidad entre distintos sistemas y aplicaciones y una implementación que se adapte a las necesidades sin afectar a las comunicaciones entre dispositivos. Tampoco pierde efectividad, ya que su velocidad, baja latencia, baja sobrecarga en la comunicación, optimización del transporte y capacidad de transmitir millones de mensajes a multitud de receptores instantáneamente, lo convierten en una tecnología ideal para sistemas de alto rendimiento en tiempo real.

Al ser descentralizado, es decir, al no requerir un nodo central, el servicio DDS es mucho más eficiente y eficaz, ya que los mensajes no deben atravesar un dispositivo intermediario, ejecutando la comunicación punto a punto, y resultando en una mayor velocidad. Además, dispone de un servicio de descubrimiento dinámico, haciendo que las aplicaciones DDS sean extensibles y escalables. La aplicación no necesita conocer ni configurar los puntos finales de los dispositivos para la comunicación, ya que estos se descubren automáticamente en ejecución, y es capaz de identificar si el punto final se utiliza para publicar datos, para suscribirse o para ambos, el tipo de dato publicado o suscrito, y las características específicas de la comunicación.

DDS soporta principalmente el modelo de publicación-suscripción, intercambiando información basada en un tema o topic identificado por su nombre. En este modelo, cualquier nodo conectado puede publicar mensajes con el tema especificado o suscribirse a un tema, y DDS se encarga de que los datos se entreguen a los suscriptores correctos en el momento adecuado mediante comunicación peer-to-peer. Al suscribirse, es posible especificar filtros de tiempo y subconjuntos de datos para obtener solo los requeridos, y tiene la capacidad de detectar cambios para que los suscriptores reciban actualizaciones adecuadas de los datos. Por otro lado, al publicar, es DDS quien gestiona la complejidad de la transmisión y se encarga de almacenar los datos de manera segura. También ofrece compatibilidad con RPC o llamadas a procedimientos remotos.

El middleware es independiente del transporte, y puede funcionar sobre UDP, TCP y memoria compartida, entre otros. Entre las características opcionales que ofrece, como el filtrado de grandes datos, se encuentra la gestión de calidad de servicio o QoS, donde se pueden especificar requisitos de rendimiento y confiabilidad, como la latencia, el ancho de banda, la prioridad, la disponibilidad de los datos, el uso de recursos y la sincronización. Además, incluye mecanismos de seguridad que proporcionan autenticación, encriptación, confidencialidad, control de acceso e integridad en la distribución de información.

2.2.5. Constrained Application Control (CoAP)

CoAP es un protocolo de la capa de aplicación que permite a dispositivos con recursos limitados, como los que se encuentran en una red **IoT**, comunicarse entre sí. Funciona en un marco cliente-servidor, en el cual el cliente realiza una solicitud a un punto de comunicación del dispositivo servidor, y este responde, permitiendo la interoperabilidad entre los dispositivos uno a uno.

Este protocolo opera sobre el protocolo de transporte **UDP**, que, a diferencia de **TCP**, no requiere que los dispositivos establezcan una conexión de datos previa al envío de datos. Esto trae tanto consecuencias positivas como negativas. La consecuencia negativa radica en la poca fiabilidad en la comunicación, ya que el protocolo **UDP** no garantiza la entrega de los mensajes, sino que esta garantía se gestiona desde la implementación de **CoAP**. Es posible establecer garantía de entrega mediante acuses de recibo (**ACK**) y tiempos de espera. La consecuencia positiva del uso de **UDP** es la posibilidad de funcionar en redes con pérdidas o inestables, adecuado para sistemas **IoT**, ya que suelen operar en entornos de red difíciles, y la rapidez en la comunicación, pues no requiere una conexión de datos previa, enviando directamente el mensaje.

Esta comunicación utiliza una arquitectura **RESTful**, en la cual los datos y las funcionalidades se consideran recursos a los que se accede mediante una interfaz estándar y uniforme. Estos recursos se acceden y se interactúa con ellos mediante métodos **HTTP** estándar, permitiendo una interoperabilidad sencilla entre distintos tipos de dispositivos y facilitando a los desarrolladores el uso del protocolo. No es necesario que los recursos de la red sean conocidos por el dispositivo que vaya a utilizarlos, ya que **CoAP** implementa un mecanismo de descubrimiento integrado a través de un núcleo en la red, útil en redes **IoT** en las que los dispositivos constantemente se conectan y desconectan. Además, el intercambio de mensajes entre dispositivos es asíncrono, lo que significa que un dispositivo puede enviar una solicitud a otro y continuar ejecutando otras tareas mientras que la respuesta puede recibirla en cualquier momento.

CoAP se basa en el intercambio de mensajes compactos codificados en un formato binario simple, cuyo tamaño no debe superar el necesario para encapsularlos dentro de un datagrama **IP**. Los distintos tipos de mensajes son: confirmables (CON, para enviar mensajes fiables), no confirmables (NON, para enviar mensajes con menos fiabilidad), de acuse de recibo (ACK) y de reinicio (RST, para indicar que el receptor no puede procesar una solicitud).

Además es un protocolo diseñado para utilizar un bajo consumo de recursos en la transferencia, y permite hacer uso del protocolo de seguridad de la capa de transporte (Data Transport Layer Security, **DTLS**) para aumentar la seguridad, además de extender su implementación con funcionalidades adicionales. Por el contrario, es menos maduro que sus alternativas, resultando en una menor cantidad de recursos, guías y herramientas, además de una compatibilidad reducida con otros dispositivos **IoT**.

<https://webbylab.com/blog/mqtt-vs-other-iot-messaging-protocols-detailed-comparison/> <https://www.techtarget.com/iotagenda/tip-12-most-commonly-used-IoT-protocols-and-standards> <https://build5nines.com/top-iot-messaging-protocols/> <https://www.a3logics.com/blog/iot-messaging-protocols/> https://en.wikipedia.org/wiki/Advanced_Message_Queueing_Proto https://www.gotiot.com/pages/articles/amqp_intro/index.html <https://www.emqx.com/en/blog/mqtt-vs-amqp-for-iot-communications#what-is-amqp> <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core>

messaging-v1.0-os.html#section-message-format <https://en.wikipedia.org/wiki/XMPP> <https://www.pubnub.com/guides/xmpp/>
<https://blazeclan.com/blog/xmpp-for-dummies-part-3-stanzas-in-detail/> <https://slixmpp.readthedocs.io/en/latest/api/stanza/presen>
<https://slixmpp.readthedocs.io/en/latest/api/stanza/iq.html> <https://www.dds-foundation.org/omg-dds-standard/>
<https://www.utpl.edu.ec/proyectomiddleware/?q=tutorial-dds> <https://www.emqx.com/en/blog/coap-protocol0> https://www.gotiot.com/pages/articles/coap_intro/index.html (DOCUMENTO) <https://datatracker.ietf.org/doc/html/rfc>

2.2.6. Comparación de middleware y protocolos

A partir de la información de los apartados anteriores, se pueden recopilar las diferencias y similitudes en los siguientes puntos:

1. *MQTT*:

- **Descripción corta:** Basado en colas de mensajes y topics.
- **Patrón de comunicación:** Publicación-suscripción.
- **Necesita intermediario:** Sí, el broker.
- **Protocolo de transporte:** [TCP](#).
- **Ventajas:** Muy utilizado, ligero, eficiente, útil en dispositivos y redes de recursos limitados, soporta distintas calidades de servicio.
- **Desventajas:** Encriptación y enrutación limitadas, bajo soporte para tipos de datos complejos.
- **Casos de uso:** Telemetría, mensajería ligera, automatización industrial, monitorización del entorno, hogares inteligentes, soluciones energéticas.

2. *AMQP*:

- **Descripción corta:** Basado en colas de mensajes y exchanges.
- **Patrón de comunicación:** Publicación-suscripción, directa o fanout.
- **Necesita intermediario:** Sí, el broker.
- **Protocolo de transporte:** [TCP](#).
- **Ventajas:** Alto rendimiento, seguro, ampliamente usado, parecido a [MQTT](#).
- **Desventajas:** Alto consumo de recursos, difícil aprendizaje.
- **Casos de uso:** Servicios financieros, procesamiento de transacciones, envío de datos en tiempo real, comunicación segura entre entidades.

3. *XMPP*:

- **Descripción corta:** Comunicación de datos y presencia mediante mensajes [XML](#).
- **Patrón de comunicación:** Cliente-servidor.
- **Necesita intermediario:** Sí, el servidor XMPP.
- **Protocolo de transporte:** [TCP](#).
- **Ventajas:** Robusto, extensible, muy ampliable de funciones.
- **Desventajas:** No optimizado para entornos limitados, complejo de implementar.
- **Casos de uso:** Mensajería instantánea, redes sociales, plataformas de colaboración.

4. *DDS*:

- **Descripción corta:** Comunicación sin servidor.
- **Patrón de comunicación:** Publicación-suscripción.

- **Necesita intermediario:** No.
- **Protocolo de transporte:** TCP y UDP, compatible con otros.
- **Ventajas:** Alto rendimiento, fácil de escalar, comunicación centrada en datos en tiempo real, soporte de tipos de datos complejos, configurable la calidad de servicio.
- **Desventajas:** Complejo de implementar, altos recursos de dispositivos y de ancho de banda.
- **Casos de uso:** Sistemas de tiempo real, aplicaciones de misión crítica, automatización industrial, automoción, gestión de cadenas de suministro, sistemas distribuidos de gran escala.

5. *CoAP*:

- **Descripción corta:** Basada en recursos compartidos y accesibles mediante REST.
- **Patrón de comunicación:** Cliente-servidor.
- **Necesita intermediario:** No.
- **Protocolo de transporte:** UDP.
- **Ventajas:** Eficiente en redes y dispositivos de recursos limitados, soporte nativo para tecnologías web.
- **Desventajas:** Bajo soporte de clientes concurrentes.
- **Casos de uso:** Dispositivos de recursos limitados, automatización en hogares.

{ <https://webbylab.com/blog/mqtt-vs-other-iot-messaging-protocols-detailed-comparison/> <https://www.a3logics.com/blog/iot-messaging-protocols/>
}

2.3. Espressif y sus dispositivos

Espressif Systems⁴ es una multinacional de semiconductores fundada en 2008, que opera como líder mundial en el ámbito del IoT y está comprometida a proporcionar a millones de usuarios algunos de los mejores dispositivos y plataformas de software de la industria, junto con una variedad de soluciones IoT seguras, robustas, eficientes, versátiles, asequibles y orientadas al código abierto.

Los productos de Espressif se utilizan ampliamente en dispositivos de electrónica de consumo, y es conocido por sus populares series de chips, módulos y placas de desarrollo ESP8266 y ESP32, los cuales se analizarán en los siguientes apartados.

{ <https://www.espressif.com/en/company/about-espressif> <https://www.eurotronix.com/es/fabricantes/espressif>
<https://www.digikey.es/es/supplier-centers/espressif-systems> }

2.3.1. ESP8266

El ESP8266 es un SoC o sistema en un chip diseñado para dispositivos móviles, electrónica portátil y aplicaciones del IoT. Lanzado en agosto de 2014, integra un procesador mononúcleo Tensilica L106 con una arquitectura Reduced Instruction Set Computer (RISC) de 32 bits de bajo consumo y una velocidad de reloj de entre 80 y 160 MHz.

⁴<https://www.espressif.com/>

Presenta una arquitectura para el ahorro de energía, permitiendo establecer el chip en modo activo, reposo y reposo profundo, lo cual es útil para que los dispositivos diseñados para alimentarse por batería funcionen durante mucho más tiempo.

En cuanto a memoria, no dispone de una memoria flash para almacenar programas, la cual debe ser proporcionada por el módulo que implemente este chip y puede tener un tamaño máximo de 16 MiB. Integra una RAM para instrucciones de 32 KiB, una caché de instrucciones 32 KiB, 80 KiB para almacenar datos del usuario y 16 KiB para datos del sistema de ETS.

Su bajo voltaje operativo oscila entre 2,5 y 3,6 V, con una corriente de operación alrededor de los 80 mA. Cuenta con la capacidad de funcionar en entornos industriales gracias a su amplio rango de temperatura de operación, que va de -40 a 125 °C.

Admite distintos tipos de protocolos de comunicación, como IPv4, TCP, UDP y HTTP. Es un dispositivo certificado para funcionar por Wi-Fi y compatible con los protocolos 802.11 b/g/n en una frecuencia de 2,4 GHz. Tiene la capacidad de actuar como cliente en redes protegidas por claves WEP, WPA y WPA2, además de poder actuar como punto de acceso inalámbrico.

También integra en sus dimensiones compactas 16 pines GPIO para conectar dispositivos de entrada y salida, un conversor analógico de 10 bits, conmutadores de antena, un amplificador de potencia y de recepción, un balun de radiofrecuencia y módulos de gestión de potencia.

Este sistema admite varios IDEs y lenguajes de programación, como C y C++, utilizando Arduino IDE⁵ o PlatformIO; MicroPython⁶, utilizando Mu Editor, Thonny IDE o Pymakr; y Lua⁷, utilizando LuaLoader.

{ <https://www.espressif.com/en/products/socs/esp8266> https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf <https://en.wikipedia.org/wiki/ESP8266> <https://www.luisllamas.es/esp8266/> <https://programarfacil.com/podcast/esp8266-wifi-coste-arduino> <https://randomnerdtutorials.com/micropython-ides-esp32-esp8266/> <https://www.danielmartingonzalez.com/es/usando-lua-en-esp8266-nodemcu-con-lualoader-y-esplorer/> }

2.3.2. ESP32

El ESP32 es el SoC sucesor del ESP8266. Igual de apto para electrónica portátil e IoT, comparte muchas características y añade mejoras que lo convierten en un sistema muy superior.

Integra un procesador Tensilica Xtensa LX6 de doble núcleo (o de uno, dependiendo de la variante utilizada) cuya frecuencia de reloj oscila entre los 160 y 240 MHz, que trabaja en conjunto con un coprocesador de ultra baja energía.

La memoria experimenta un significativo aumento respecto a su predecesor, con un total de 520 KiB de SRAM, 448 KiB de memoria de solo lectura, 32 KiB de caché y hasta 4 MiB de memoria de almacenamiento (dependiendo del modelo).

⁵<https://www.arduino.cc/en/software>

⁶<https://micropython.org/>

⁷<https://www.lua.org/>

Este chip, lanzado en septiembre de 2016, añade en comparación con el ESP8266 una mejora de potencia, soporte de Bluetooth 4.2 y **BLE**, sensor de temperatura, sensor hall, sensor táctil, reloj de tiempo real, más pines **GPIO** (hasta 34) y varios modos de energía.

Además, incorpora arranque seguro, encriptado de la flash y soporte de aceleración por hardware para los algoritmos y protocolos de cifrado y encriptación **AES**, **SHA-2**, **RSA**, **ECC** y el generador de números aleatorios.

El ESP32 tiene la posibilidad de funcionar como un sistema autónomo o como parte de un puente e interconexiones, y tiene la capacidad de interactuar con otros sistemas para proveer funcionalidad Wi-Fi y Bluetooth a través de sus interfaces.

Desde el lanzamiento del ESP32 original, han ido apareciendo variantes con distintas capacidades y procesadores, pero gran parte del código del ESP32 es compatible. Estas variantes son:

- ESP32-S2: enfocado en el consumo, integra un procesador mononúcleo LX7, reduce la memoria disponible y no tiene soporte de Bluetooth.
- ESP32-S3: utiliza el mismo procesador que el anterior, contiene más memoria y da soporte a Bluetooth 5 y **BLE**, enfocado al soporte de inteligencia artificial.
- ESP32-C3: contiene un procesador **RISC-V** mononúcleo y admite Bluetooth 5 y **BLE**, enfocado en la seguridad.
- ESP32-C6: centrado en la conectividad, la principal diferencia con el anterior es el soporte de Bluetooth 5.3, Wi-Fi 6 (802.11ax) y conectividad de radio (802.15.4) compatible con los protocolos Thread, Zigbee y Matter.
- ESP32-C2: incorpora un procesador **RISC-V** mononúcleo y admite Bluetooth 5 y **BLE**. Es un chip de pequeñas dimensiones que mantiene una conectividad robusta y estándares de seguridad.
- ESP32-C5: es la versión más reciente con mayor velocidad de reloj y capacidad de memoria, y es el primero que soporta Wi-Fi 6 a 5 **GHz**. Su enfoque en la conectividad también proviene de la capacidad de conexión con Bluetooth 5.2.

{ https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf <https://www.espressif.com/en/products/socs>
<https://en.wikipedia.org/wiki/ESP32> <https://www.luisllamas.es/esp32/> <https://www.espboards.dev/blog/esp32-soc-options/> }

2.4. ESP-NOW

El **IoT** se forma a partir de la conectividad entre objetos, donde surge la necesidad de un protocolo que equilibre las necesidades de latencia, uso de energía, capacidad de transmisión, confiabilidad y seguridad en la transmisión de datos. Son factores determinantes en el futuro desarrollo de esta tecnología, y aparecen como candidatos un gran número de tecnologías y protocolos, destinados tanto para la comunicación en área local, como Wi-Fi, para áreas amplias, como LoRa y LoRaWAN, y para transmisión a corta distancia, como **RFID**. Recientemente, al mencionado grupo se ha añadido ESP-NOW, el cual, pese a sus limitaciones, tiene características notables y es adecuado para **IoT**.

ESP-NOW es un protocolo de comunicación inalámbrica diseñado por Espressif para su uso entre sus dispositivos, como los ESP8266 y ESP32. Con el objetivo de sustituir a Wi-Fi y a otras tecnologías, ESP-NOW es capaz de realizar transmisiones de información y control rápidas, estables y con un bajo consumo de recursos de **CPU** y memoria flash entre dispositivos inteligentes, sin necesidad de un enrutador.

Se caracteriza por la rapidez de la transmisión, lograda evitando la necesidad de establecer una conexión previa entre dispositivos. Permite a su vez poner a disposición los dispositivos para transmitir datos y recibir órdenes instantáneamente tras el encendido. Además, este protocolo está basado en la capa de enlace de datos y es capaz de omitir las capas de red, transporte, sesión, presentación y aplicación del modelo **OSI**, reduciendo el consumo de energía (mejorando la autonomía en dispositivos con batería) y el retardo en la recepción y en el procesamiento de mensajes debido a la nula necesidad de cabeceras de paquete o desempaquetadores de cada capa. En redes congestionadas, es una característica beneficiosa, ya que brinda la capacidad de respuestas rápidas que reducen el retraso causado por la pérdida de paquetes. En el modelo utilizado en ESP-NOW en comparación con el modelo **OSI** estándar de la Figura 2.3 se puede observar la ausencia de las 5 capas mencionadas anteriormente.



Figura 2.3: Comparación de las capas del modelo OSI con las del modelo ESP-NOW **TODO: REFERENCIAR**

Pese al objetivo de ESP-NOW de reemplazar Wi-Fi, en los dispositivos de Espressif es capaz de coexistir simultáneamente junto a Wi-Fi y Bluetooth. Esto es útil para utilizar un dispositivo como gateway y exportar los datos intercambiados entre ESP-NOW hacia otras redes.

Tiene la capacidad de transmitir datos de manera máquina a máquina o broadcast, y para establecer la comunicación solo se requiere la dirección **MAC** del dispositivo de destino y establecer un canal de transmisión. No obstante, dispone de una cantidad limitada de dispositivos con los que se puede emparejar. En general, el número de dispositivos emparejados no puede exceder de 20, y la cantidad de estos con los que se puede establecer una comunicación cifrada es configurable. Por defecto, este valor es 7, y admite un valor máximo de 17. Esta limitación puede ser un inconveniente en caso de necesitar una gran cantidad de dispositivos interconectados, pero una solución sería formar jerarquías de dispositivos.

Permite establecer funciones de callback que serán llamadas instantáneamente tras el envío y la recepción

de datos para poder gestionarlos adecuadamente. Cabe remarcar que el protocolo no garantiza que se reciban los datos correctamente, pero existe la posibilidad de establecer **ACKs** para confirmar la correcta recepción y procesamiento de los datos, además de números de secuencia para afrontar la duplicidad.

ESP-NOW utiliza tramas de acción específicas del proveedor para encapsular y transmitir datos de una longitud máxima de 250 bytes, con un alcance de transmisión de entre 100 y 500 metros, dependiendo de las condiciones atmosféricas, y con una tasa de velocidad de bits de 1 megabit por segundo. Esto es beneficioso para la comunicación a larga distancia debido a su gran alcance en dispositivos al aire libre o incluso separados por paredes o pisos. Sin embargo, su uso puede estar limitado por la pequeña carga útil que puede transmitir, por lo que en otros casos podría ser mejor utilizar otras tecnologías como Wi-Fi. ESP-NOW utiliza tramas de un tamaño entre 43 y 293 bytes, cuyo formato, mostrado en la Figura 2.4, está compuesto por los siguientes campos:

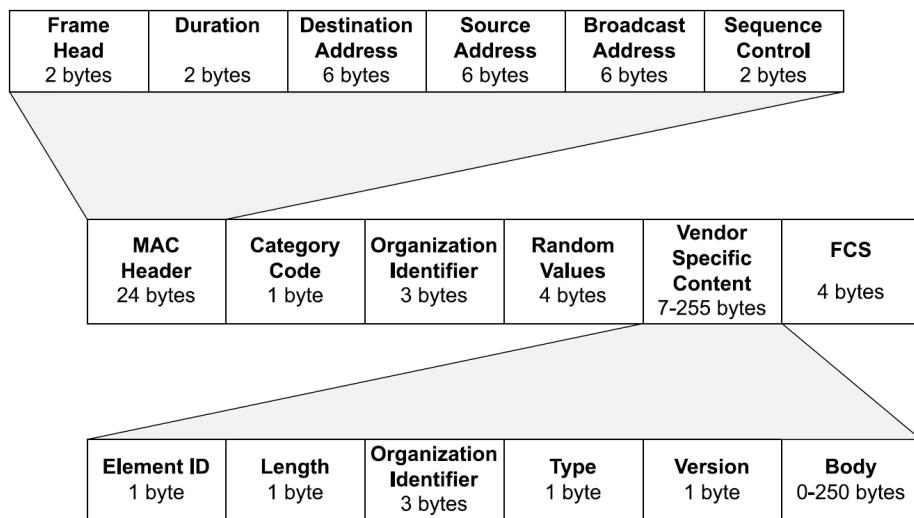


Figura 2.4: Formato de una trama ESP-NOW (Fuente: [64])

- Cabecera **MAC**, distinta de una cabecera **MAC** común debido a su funcionamiento sin conexión.
- Código de categoría, establecido a 127 para indicar la categoría específica del proveedor.
- Identificador de la organización único, que son los 3 primeros bytes de la dirección **MAC** aplicada por Espressif.
- Valores aleatorios, utilizados para prevenir ataques de retransmisión.
- Contenido específico del proveedor, que ocupa entre 7 y 257 bytes y contiene los siguientes campos específicos del proveedor:
 - ID del elemento, establecido a 221 para indicar que se trata del elemento específico del proveedor.
 - Longitud total del resto de campos.
 - Identificador de la organización, igual que el mencionado antes, los 3 primeros bytes de la dirección **MAC**.
 - Tipo, con valor 4 para indicar ESP-NOW.
 - Versión de ESP-NOW.

- Cuerpo, que contiene los datos de ESP-NOW y puede ocupar entre 0 y 250 bytes.
- Frame Check Sequence, utilizado para verificar la integridad de la información recibida.

Existe la posibilidad de asegurar la transmisión de datos a través de ESP-NOW utilizando algoritmos de encriptación **ECDH** y **AES** y el método CBC-MAC Protocol (CCMP) que protegen las tramas de acción. Esto está limitado a comunicaciones entre pares, no se admite el cifrado de tramas utilizadas para la multi-difusión.

En cuanto a la gestión de dispositivos, puede utilizarse como un protocolo que ayude al aprovisionamiento de datos y configuraciones a dispositivos, depurarlos y actualizar su firmware.

ESP-NOW no necesita ningún procedimiento especial aparte de la implementación para poder utilizarse con fines comerciales. En la actualidad, se encuentra ampliamente utilizado en electrodomésticos inteligentes, iluminación inteligente, control remoto, sensores y otros.

{ (imagen, fuente): <https://www.espressif.com/sites/all/themes/espressif/images/esp-now/model-en-mobile.png> <https://www.espressif.com/en/solutions/low-power-solutions/esp-now> <https://docs.espressif.com/projects/espressif-esp-faq/en/latest/application-solution/esp-now.html> https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html <https://emariete.com/esp8266-esp32-espnow/> }

2.4.1. Tecnologías alternativas más comunes

2.4.1.1. Wi-Fi

“Wi-Fi” es el nombre que otorga la Wi-Fi Alliance a esta tecnología de red inalámbrica basado en los estándares **IEEE 802.11**. Esta tecnología es ampliamente utilizado para enlazar dispositivos en redes **LAN** y proveer de acceso a Internet utilizando ondas de radio de 2,4 o 5 **GHz** (dependiendo de la versión) para transmitir la información, cumpliendo con la misión de ser una alternativa al envío de datos a través de cables.

En una red Wi-Fi se pueden encontrar una variedad de dispositivos cliente, que son los que aprovechan las características de la red, y dispositivos que distribuyen la red. Estos últimos consisten en routers o enruteadores, que brindan la conexión a Internet a los dispositivos y enrutan los mensajes enviados a través de la red; puntos de acceso, que transmiten la señal inalámbrica y es donde se conectan los dispositivos introduciendo las credenciales de la red; y repetidores, utilizados para extender el área de cobertura de una red. Entre los dispositivos, tanto clientes como los distribuidores de red, se utilizan adaptadores de red inalámbrica, que convierten los datos en una señal de radio y viceversa.

Es una de las tecnologías más utilizadas a nivel mundial, siendo que la mayoría de hogares y establecimientos disponen de una red Wi-Fi. Esta popularidad es beneficiosa para el **IoT**, ya que ofrece una capa de compatibilidad con una amplia gama de dispositivos sin necesidad de antenas, adaptadores de red ni otro tipo de hardware adicional. Además, Wi-Fi no es una tecnología nueva, tiene un sólido legado de interoperabilidad, y permite enviar información entre dispositivos con baja latencia.

Entre sus características adicionales se encuentra la topología flexible, que permite conectar los dispositi-

vos de distintas maneras; la seguridad, ya que es posible tener redes protegidas con contraseñas cifradas mediante distintos protocolos (**WEP**, **WPA** y **WPA2**), el bajo coste de instalación, que en comparación con la instalación de una red cableada, resulta más económico; y la capacidad de llegar a donde los cables no pueden llegar.

Existen distintos estándares de Wi-Fi que definen cómo actúa la red, y que cambian cada pocos años trayendo mejoras en el alcance, la velocidad y la conectividad. Por lo general, los dispositivos certificados para un estándar son intercomunicables con los certificados para otro estándar Wi-Fi siempre que comparten la misma banda de frecuencia, por lo que no es una preocupación tener todos los dispositivos con la versión más reciente. Los estándares Wi-Fi se muestran en la Tabla 2.1.

Tabla 2.1: Lista de estándares Wi-Fi *TODO: REFERENCIAR*

Nombre comercial	Estándar IEEE	Año	Frecuencia (GHz)	Velocidad máxima	Rango (metros)
Wi-Fi 1	802.11a	1999	5	54 Mbps	120
Wi-Fi 2	802.11b	1999	2,4	11 Mbps	140
Wi-Fi 3	802.11g	2003	2,4	54 Mbps	140
Wi-Fi 4	802.11n	2009	2,4 y 5	450 Mbps	250
Wi-Fi 5	802.11ac	2014	5	2,3 Gbps	35
WiGig	802.11ad	2016	60	7 Gbps	100
Wi-Fi 6	802.11ax	2019	2,4, 5 y 6	9,6 Gbps	240
Wi-Fi 7	802.11be	2024	2,4, 5 y 6	46 Gbps	(Por determinar)

La popularidad de Wi-Fi abarca una gran variedad de dispositivos soportados, como teléfonos inteligentes, ordenadores, televisores inteligentes, impresoras e incluso placas de desarrollo como ESP8266 y ESP32.

{ <https://en.wikipedia.org/wiki/Wi-Fi> <https://es.wikipedia.org/wiki/Wifi> <https://emariete.com/esp8266-esp32-espnow/> <https://www.wi-fi.org/discover-wi-fi/internet-things> <https://www.adslzone.net/reportajes/tecnologia/que-es-wifi-como-funciona/> <https://www.proofpoint.com/es/threat-reference/wifi#:~:text=Wifi%2C%20que%20es%20una%20contraseña>

Fuentes de tabla

<https://www.intel.es/content/www/xl/es/support/articles/000005725/wireless-legacy-intel-wireless-products.html>
<https://www.monolitic.com/cual-es-la-mejor-tecnologia-wifi-para-desarrollos-iot/>
<https://www.makeuseof.com/tag/understanding-common-wifi-standards-technology-explained/>
<https://www.netspotapp.com/es/blog/wifi-standards/>
<https://www.xataka.com/nuevo/nuevo-wifi-7-informacion>

```

https://www.geckoandfly.com/10041/wireless-wifi-802-11-abgn-router-range-and
    ↳ -distance-comparison/
https://www.business.com/articles/what-is-802-11-ax-wi-fi/
}

```

2.4.1.2. Bluetooth

Bluetooth es un estándar de tecnología que facilita el intercambio de datos entre dispositivos a través de distancias cortas, con un máximo de 10 metros.

Desde su introducción en 1998, ha pasado por múltiples revisiones, siendo las más relevantes las de la última década:

- Versión 4.0 hasta 4.2: aumentó la velocidad de transferencia de datos a 24 Mbps y el alcance hasta 100 metros, y añadió soporte al protocolo IPv6. Además, introdujo BLE o Bluetooth Low Energy, una nueva variante de esta tecnología.
- Versión 5.0 hasta 5.2: aumentó la velocidad de transferencia de datos a 50 Mbps y el alcance hasta 200 metros, y realizó mejoras en la transmisión de audio y en el consumo de energía.

Bluetooth Low Energy fue diseñado para operaciones de bajo consumo de energía, capaz de soportar diferentes tipologías de comunicación (punto a punto, difusión y malla), a diferencia del clásico que solo admite punto a punto. Mientras que el Bluetooth clásico se usa para transferir datos y sonido, BLE es capaz de utilizarse para analizar con alta precisión la presencia, distancia y dirección del dispositivo.

Bluetooth utiliza ondas de radio UHF en las bandas ISM sin licencia de 2,4 GHz, y se utiliza como alternativa a las conexiones por cable para intercambiar ficheros y conectar transmisores de audio. Gracias a su bajo consumo de energía, seguridad, capacidad contra interferencias, compatibilidad con varios sistemas operativos y facilidad de implementación, esta tecnología se convierte en una buena opción para la implementación del IoT. Además, tiene la capacidad de agregar capas de cifrado, autenticación y verificación, y de construir redes PAN entre dispositivos al interconectar varios entre sí. Es común encontrarlo en pulseras y relojes inteligentes, teléfonos inteligentes, ordenadores, reproductores de música, altavoces, auriculares, y placas de desarrollo como ESP32.

```
{
  https://www.bluetooth.com/learn-about-bluetooth/tech-overview/
  https://en.wikipedia.org/wiki/Bluetooth
  https://www.xatakahome.com/curiosidades/bluetooth-su-evolucion-estas-diferencias-distintas-versiones
  https://www.mokosmart.com/es/what-is-bluetooth-iot-and-why-choose-it/
}
```

2.4.1.3. Más tecnologías y protocolos IoT

En el ámbito del IoT, existen otras tecnologías y protocolos populares que, a diferencia de Wi-Fi y Bluetooth, los cuales están integrados en las placas ESP32, requieren modificaciones para poder ser utilizados en estas placas. Sin embargo, esto no elimina la necesidad de conocerlas, al menos a través de un análisis no tan detallado.

Zigbee: es un protocolo basado en la especificación IEEE 802.15.4 que permite formar redes PAN sencillas. Estas redes son en malla y están formadas por un nodo coordinador. Zigbee opera en la banda de 2,4 GHz y se utiliza en proyectos a pequeña escala que requieren una conexión inalámbrica de baja potencia, baja velocidad de transmisión (hasta 250 Kbps) y un rango próximo (hasta 10-100 metros) [68].

Thread: es una tecnología que utiliza 6LoWPAN y la especificación IEEE 802.15.4, por lo que opera en la banda de 2,4 GHz con una velocidad de hasta 250 Kbps. Permite crear redes malladas de bajo consumo que comunican de manera encriptada más de 250 dispositivos, los cuales deben ser certificados y creados por empresas miembros del Thread Group [67] [37].

LoRa: es una tecnología inalámbrica de conexión punto a punto que emplea tecnología de modulación en radiofrecuencia. Es ideal cuando se necesitan conexiones que cubren largas distancias con dispositivos sin fácil acceso a la corriente eléctrica de red ni cobertura, ya que ofrece un bajo consumo de uso y un largo alcance de 10-20 km, con alta tolerancia a interferencias. Opera en bandas inferiores a 1 GHz (dependiendo de la región), y transmite un máximo de 255 bytes a velocidades entre 0,3 Kbps y 50 Kbps. Esta tecnología es conocida por ser utilizada en LoRaWAN [25] [56].

2.4.1.4. Comparaciones entre Wi-Fi, Bluetooth y ESP-NOW

Es de alta relevancia comparar estas tecnologías y ESP-NOW entre sí, ya que son las más populares en el ámbito del IoT y son compatibles con las placas ESP32 que incorporan el reciente e innovador ESP-NOW. En particular, ESP32 es compatible con Bluetooth 4.2 y con Wi-Fi b, g y n de 2,4 GHz, por lo que a lo largo de este apartado se comparan estas versiones. Estos tres protocolos son similares en varios aspectos, ya que utilizan ondas de radio para transmitir datos de forma inalámbrica a una amplia gama de dispositivos, de manera rápida y fiable. Esto puede resultar en una decisión compleja para elegir entre los tres, aunque hay escenarios en los que no es necesario elegir, ya que en un ESP32 pueden trabajar en conjunto.

De manera más resumida, estas son las características teóricas que ofrecen los protocolos:

1. *ESP-NOW:*

- **Alcance:** 220 metros.
- **Cantidad de dispositivos conectables:** 20 a cada nodo.
- **Unidad de Transmisión Máxima (MTU):** 250 bytes.
- **Velocidad de transmisión:** 1 Mbps.
- **Uso:** IoT y comunicación entre dispositivos de Espressif.

2. *Wi-Fi b/g/n (2,4 GHz):*

- **Alcance:** 250 metros.
- **Cantidad de dispositivos conectables:** depende de la configuración de la red y la asignación de direcciones IP.
- **Unidad de Transmisión Máxima (MTU):** 1460 bytes, configurado en la librería de red de ESP32.
- **Velocidad de transmisión:** 54 Mbps.
- **Uso:** conexión a internet, acceso a dispositivos e IoT.

3. Bluetooth 4.2:

- **Alcance:** 50 metros.
- **Cantidad de dispositivos conectables:** 7 a cada nodo.
- **Unidad de Transmisión Máxima (MTU):** 251 bytes.
- **Velocidad de transmisión:** 1 Mbps.
- **Uso:** audio, dispositivos personales e IoT.

/TODO: referenciar/

/TODO: re-redactar/ Para detallar las comparaciones aplicando los protocolos en escenarios de uso real, se han tomado los datos de distintas pruebas realizadas y detalladas en una publicación de 2021 llevada a cabo por Dania Eridani, Adian Fatchur Rochim y Faiz Noerdiyan Cesara, miembros del Departamento de Ingeniería Informática de la Universidad de Diponegoro⁸, en Indonesia.

En esta publicación [41] se realizaron pruebas de rango, velocidad, latencia, consumo y resistencia a barreas. Para ello, se utilizaron una placa ESP32 Development Board, una ESP32-CAM y una ESP32U, además de una antena externa de 2.4GHz conectada a la última placa mencionada y utilizada en ciertas pruebas.

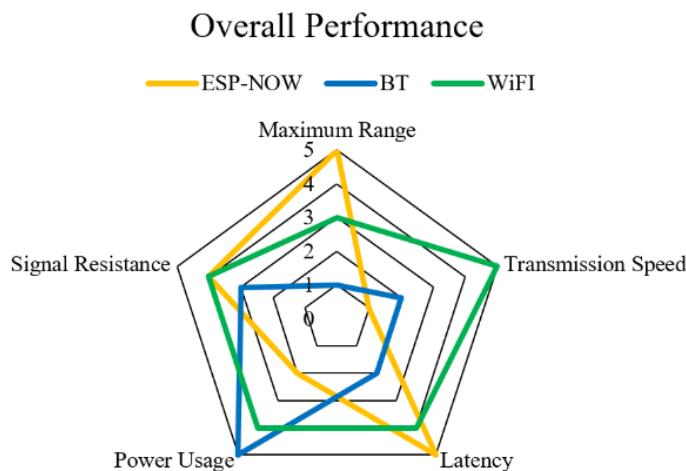


Figura 2.5: Comparación del rendimiento entre protocolos en distintos aspectos (Fuente: [41])

Las conclusiones a las que se llega en esta publicación se pueden representar en la Figura 2.5, que muestra que:

- La mayor ventaja de Bluetooth se encuentra en el consumo, ya que su rendimiento en el resto de los aspectos es muy deficiente.
- ESP-NOW es el mejor candidato cuando se requieren rangos elevados, una señal de comunicación resistente y mínima latencia en velocidades de datos muy pequeñas, pero su uso consume mucha energía relativamente.
- Wi-Fi es una tecnología muy equilibrada, y destaca por su velocidad.

{ [https://es.wikipedia.org/wiki/Bluetooth_\(especificaci%C3%B3n\)](https://es.wikipedia.org/wiki/Bluetooth_(especificaci%C3%B3n)) <https://docs.espressif.com/projects/esp->

⁸<https://tekkom.ft.undip.ac.id/language/en/home/>

faq/en/latest/application-solution/esp-now.html https://www.amarinfotech.com/differences-comparisons-bluetooth-5-vs-4-2.html https://www.symmetryelectronics.com/blog/bluetooth-5-versus-bluetooth-4-2-what-s-the-difference/ https://docs.arduino.cc/tutorials/nano-esp32/esp-now/ https://github.com/espressif/arduino-esp32/blob/master/libraries/Network/src/NetworkUdp.cpp https://www.electronicdesign.com/technologies/communications/article/v42-creating-faster-more-secure-power-efficient-designspart-1

```
(documento)https://ieeexplore.ieee.org/document/9573246
}
```

2.5. Ejemplo de modelo tradicional publicador-broker-suscriptor con MQTT y ESP32

Tras introducir una serie de conceptos sobre el IoT, en este apartado se detallará el ejemplo visto en el apartado de Internet de las Cosas, en la Figura 2.1, que trata de un sistema de riego por aspersión.

La primera parte consiste en identificar los dispositivos a utilizar. Los sensores y actuadores deben estar conectados a un dispositivo capaz de interactuar con ellos, como puede ser una placa ESP32.

El broker utilizará una comunicación basada en colas para recibir todos los datos, integrando así una implementación del protocolo de mensajería MQTT, como puede ser Mosquitto, que únicamente se puede llevar a cabo si el broker es un ordenador, como un ordenador portátil o un single board computer. Para este ejemplo, se puede suponer que el broker será una placa Raspberry Pi.

Teniendo las placas ESP32 y la Raspberry Pi con Mosquitto, es necesario establecer la comunicación entre ambos. Una de las opciones a evaluar que sea compatible con ambas placas es Bluetooth, pero no es adecuado para una comunicación a larga distancia, ni para transmitir datos en tiempo casi real debido a su baja velocidad, y tampoco es compatible con Mosquitto. Por lo tanto, se debe utilizar Wi-Fi para que las placas ESP32 publiquen los datos que generen y se suscriban a las órdenes y configuraciones mediante las colas adecuadas de Mosquitto. En este caso, se deberá desplegar una serie de puntos de acceso y routers para formar una red LAN o MAN que dote de Internet a todos los dispositivos.

Finalmente, se configuran las placas ESP32 para que, con la ayuda de alguna librería de código, puedan interactuar con las colas de Mosquitto, y también el broker para poder interactuar con el servidor.

La implementación de este sistema de riego presenta una serie de inconvenientes. En cuanto a inversión de dinero y tiempo, implica el despliegue de numerosos puntos de acceso y routers, así como su configuración y mantenimiento. Además, se debe asegurar una buena señal al aire libre, evaluar la zona donde se instalarán y proveer más baterías en los dispositivos que formen esta red.

Todos los dispositivos están conectados en la misma red, por lo que un tráfico alto de datos o una interrupción en el servicio puede provocar una congestión de la red y un mal funcionamiento de la misma.

Finalmente, en cuanto a seguridad, a todos los dispositivos de esta red se les otorga acceso a Internet, por lo que deben estar preparados para no sufrir un ciberataque que pueda invalidar el sistema por completo y

todos los dispositivos conectados. Esto supone una mayor complejidad en el despliegue y mayor mantenimiento para evitar un riesgo significativo en la red.

Aunque la implementación de este sistema de riego se puede realizar de distintas maneras, en este ejemplo se intenta demostrar la complejidad que supone un despliegue limitado por utilizar **MQTT** y **ESP32** en la actualidad. Pese a que, de manera independiente, ambos tienen grandes ventajas, su combinación supone una complejidad difícil de evitar.

/TODO: añadir un nuevo esquema/

{ <https://www.prometec.net/esp32-mqtt/> (libreria de codigo) }

Capítulo 3

Herramientas y Metodología

3.1. Herramientas

En este apartado se identifican y describen las distintas herramientas hardware y software que han permitido llevar a cabo este **TFG**.

3.1.1. Hardware

El núcleo de este **TFG** se basa en la programación de una placa de desarrollo, por lo que es necesario ciertos elementos hardware para llevarlo a cabo:

Ordenador portátil: es el componente principal para el desarrollo de este **TFG**, en el cual se han instalado las herramientas necesarias, se ha escrito el código, se ha accedido a herramientas y recursos en línea, y ha permitido subir el código a la placa. En este caso, el ordenador personal del alumno es un Lenovo Ideapad 3 15ALC6, cuyas características destacadas para este trabajo son:

- Procesador AMD Ryzen 7 5700 U, de 64 bits y lanzado en enero de 2021, que permite ejecutar aplicaciones y servicios aprovechando sus 8 núcleos, su alta frecuencia de 1,8 **GHz** hasta 4,3 **GHz** y su bajo consumo [43].
- 12 GB de **RAM** DDR4, que almacenan la información temporal generada por los programas en ejecución. Su capacidad determina cuántas tareas simultáneas se pueden ejecutar y su velocidad determina la rapidez de ejecución de estas.
- Almacenamiento interno SSD de 1 **TB**, encargado de almacenar de manera persistente información como el sistema operativo, las herramientas instaladas y los ficheros de código utilizados en el proyecto.
- 3 puertos USB, que permiten conectar distintos dispositivos simultáneamente al ordenador, como en este trabajo, las placas para subir el código y observar la salida por terminal durante la ejecución.
- Lector de tarjetas SD, utilizado para acceder al contenido de la tarjeta SD que se conecta a una de las placas (detallado posteriormente).
- Adaptador de red Bluetooth y Wi-Fi, para poder conectar el ordenador a Internet y acceder a los

recursos necesarios para el desarrollo.

Dos placas ESP32 DEVKIT V1: ambas fueron cedidas por la [UCLM](#) para el desarrollo de este [TFG](#). Este tipo de placas contienen las mismas características que el SoC ESP32 mencionado en el apartado [2.3.2](#). Ambas utilizan el módulo ESP32-WROOM-32.

Las placas están distribuidas en una placa de pruebas o breadboard, permitiendo conectar distintos elementos a los pines de las placas mediante jump wires o cables puente, sin necesidad de soldadura ni diseñar circuitos integrados y facilitando la prueba de componentes. En este caso, tienen conectados distintos módulos y sensores que permiten ampliar las funciones de estas:

- **Módulo lector de tarjeta microSD:** compuesto por un socket para insertar tarjetas microSD en un circuito impreso del cuál salen 7 pines para poder utilizar el bus SPI de las tarjetas microSD. El SPI o Serial Peripheral Interface es un estándar que se utiliza para transferir información entre circuitos integrados, como pueden ser la placa ESP32 DEVKIT y el lector de tarjetas. Los pines que tiene este módulo son los siguientes:

- VCC: entrada de energía, se conecta a una fuente de alimentación para alimentar el lector.
- GND: conexión a tierra, se conecta al terminal negativo de la fuente de alimentación.
- Data In: también conocido como MOSI o Master Output Slave Input, se utiliza para enviar datos desde la placa ESP32 hasta la tarjeta microSD.
- Data Out: también conocido como MISO o Master Input Slave Output, se utiliza para enviar datos desde la tarjeta microSD hasta la placa ESP32.
- Serial Clock: reloj SPI, se utiliza para sincronizar la transferencia de datos entre las placas.
- Chip Select: se utiliza para activar y desactivar la comunicación con el lector.
- Card Detect: se utiliza para detectar si hay una tarjeta insertada en el lector.

Para este trabajo, el lector, cedido por la UCLM, se utiliza para leer y escribir datos en una tarjeta microSD desde la ESP32 DEVKIT V1 que actúe como broker y gestione los registros o logs y las direcciones de los dispositivos suscritos contenidos.

- **Sensor DHT11:** sensor digital capaz de medir la temperatura y la humedad, cedido por la [UCLM](#). Alimentado por 3,3 o 5 voltios, es capaz de leer la humedad en el ambiente entre los rangos 20 y 95 % (con un 5 % de fallo) y la temperatura entre 0 y 50 °C (con 2 °C de fallo).
- **Potenciómetro BQ Zum Kit Advanced:** contenido originalmente en un kit junto a varios sensores y una placa controladora [31] [32], este potenciómetro de señal analógica cedido por la [UCLM](#) es capaz de devolver un valor en función a su rotación, siendo su rotación máxima 300°, al alimentarlo con 3,3 o 5 voltios [33].

Placa ESP32-2432S028R: esta placa proporcionada por el autor y popularmente conocida como Cheap-Yellow-Display o CYD para abreviar, se ha utilizado para hacer pruebas del funcionamiento de la herramienta desarrollada, y al igual que las anteriormente mencionadas placas ESP32, está potenciado por un ESP32-WROOM-32. En cuanto a componentes integra:

- Pantalla LCD de 2,8 pulgadas con resolución 320x240 píxeles, y táctil de tipo resistivo.
- LED multicolor RGB.

- Lector de tarjeta microSD.
- Sensor LDR, cuyo valor que devuelve depende de la resistencia que le otorga la luz que recibe.
- Conectores y pines adicionales para conectar un altavoz y otros módulos.

Tarjeta microSD: tarjeta proporcionada por el alumno autor y utilizada por una de las placas con lector de tarjetas para almacenar los registros o logs y las direcciones de los dispositivos suscritos al broker.

Adaptador de tarjeta microSD a SD: proporcionada por el autor, permite utilizar una tarjeta microSD en un lector de tarjetas SD normal al adaptar su tamaño y forma. Se ha utilizado para leer, modificar y eliminar contenidos de la tarjeta microSD desde el ordenador portátil, útil para hacer probar el funcionamiento de la herramienta.

```
{     https://www.sparkfun.com/products/544     https://es.wikipedia.org/wiki/Serial_Peripheral_Interface
https://tienda.bricogeek.com/sensores-temperatura/1574-modulo-sensor-dht11-humedad-y-temperatura.html
https://github.com/witnessmenow/ESP32-Cheap-Yellow-Display/ }
```

3.1.2. Software

Para poder realizar el desarrollo correcto de este **TFG**, se han requerido las siguientes piezas de software:

Windows 10¹: el sistema operativo es clave para poder hacer funcionar el Trabajo de Fin de Grado. En este caso, el autor ha utilizado la versión más reciente a fecha de la escritura de este documento, 22H2. Este sistema operativo desarrollado por Microsoft se encuentra instalado en el ordenador portátil mencionado en el apartado de hardware */TODO: mencionar/*, y debido a su alta popularidad tiene una gran compatibilidad con la mayoría de aplicaciones existentes, y en este caso es beneficioso para el resto de software mencionado posteriormente. Además, integra:

- Drivers: Ofrecen compatibilidad y rendimiento con todo el hardware contenido en el ordenador, como procesador, tarjeta gráfica, teclado y ratón, puertos USB, lectores de tarjetas y otros dispositivos externos. En este caso, para poder hacer uso de las placas, se ha necesitado instalar un driver adicional denominado *CH341SER²*.

La relevancia de usar Windows 10 es mínima, ya que es posible de adaptar el proyecto a un entorno en Linux con pocas o nulas complicaciones. El uso de Windows 10 ha sido decisión del autor por gusto.

GitHub³: es una plataforma en la nube propiedad de Microsoft que permite a los equipos de desarrolladores almacenar código, colaborar y realizar cambios en proyectos compartidos, alojados en esta plataforma en forma de repositorios de Git.

Git es un sistema de control de versiones de código abierto que permite gestionar el código fuente y el historial de cambios mediante comandos ejecutados en una terminal. Tiene la característica de ser distribuido, permitiendo usar ramas para aislar partes del código, como pueden ser nuevas funcionalidades en desarrollo, sin afectar al código final desplegado. Cada parte del equipo puede crear una rama, integrar los

¹<https://www.microsoft.com/es-es/software-download/windows10>

²https://www.wch-ic.com/downloads/CH341SER_ZIP.html

³<https://github.com/>

cambios necesarios y luego fusionarla con la rama principal para hacer efectivos estos cambios. Esto es útil para desarrollar varias funcionalidades simultáneamente que editen los mismos ficheros, comprobar las diferencias entre estos, aprobarlas y volver a una versión funcional anterior en caso de que errores.

Los repositorios en GitHub pueden ser públicos o privados, siendo en este último caso que en el repositorio solo pueden interactuar los usuarios agregados y cuya funcionalidad está limitada (a menos que se disponga de licencia de pago). Desde estos se pueden crear y modificar ramas, cargar ficheros, realizar *commits* (la unidad de trabajo que representa un cambio en el repositorio), ver su histórico para hacer un seguimiento de los cambios realizados, obtener las modificaciones realizadas por otros usuarios y hacer *pull requests* (solicitudes de cambios) para integrar estos cambios en el proyecto. GitHub va un paso más allá de alojar proyectos, ya que permite la gestión de proyectos y la interacción del equipo de desarrollo a través de *issues* que retroalimentan el proyecto y ofrecen ideas, asignación de responsabilidades, hitos, etiquetas, discusiones, y gráficos y tableros estilo Kanban que permiten observar fácilmente el trabajo realizado y por hacer. Otras funciones que permite esta plataforma son:

- Documentación de proyectos, mediante la creación de ficheros de texto “readme” o “léeme” en lenguaje Markdown en los directorios del proyecto.
- Creación de wikis.
- Automatización de pruebas, lanzamientos y despliegues con GitHub Actions, especificando los pasos a ejecutar tras una acción específica realizada en el repositorio.
- Alojamiento de páginas web estáticas con GitHub Pages como parte de un repositorio.
- Gists o fragmentos de código compatibles.
- GitHub Codespaces, un IDE online.

En el caso de este proyecto, se ha utilizado, con el plan gratuito de GitHub, un repositorio privado durante el desarrollo para alojar el código, compartirlo fácilmente con los tutores para enviar dudas e informar del estado del proyecto, y llevar un histórico de los cambios realizados. Posteriormente, se ha modificado la visibilidad del repositorio a público para compartir el proyecto a través de la plataforma de librerías de PlatformIO (mencionado en los siguientes puntos) y aprovechar la funcionalidad de GitHub Pages para incluir una página web estática que documente las distintas funciones que componen el proyecto, ambas funciones automatizadas a través de GitHub Actions /TODO: realizar/. Pese a conocer alternativas a la plataforma, como GitLab, se ha decidido utilizar GitHub por la experiencia previa del alumno y la facilidad de uso que ofrece.

Otra herramienta que ofrece GitHub para no separar la funcionalidad del escritorio local es **GitHub Desktop**⁴, que permite simplificar el flujo de trabajo del desarrollador y centrarse en su trabajo. Proporciona una interfaz gráfica que evita interactuar directamente con Git para clonar proyectos, hacer *commits*, cambiar de rama y ver los cambios y diferencias en los archivos.

{ <https://docs.github.com/es/get-started/start-your-journey/about-github-and-git> <https://www.hostinger.es/tutoriales/que-es-github> <https://en.wikipedia.org/wiki/GitHub> <https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/about-commits> <https://docs.github.com/es/issues/tracking->

⁴<https://desktop.github.com/>

your-work-with-issues/about-issues <https://desktop.github.com/> }

Visual Studio Code⁵: es un **IDE** ligero, potente, multiplataforma, personalizable y de código abierto, creado por Microsoft. Permite abrir directorios como espacios de trabajo capaces de contener múltiples lenguajes de programación, ofreciendo soporte nativo para JavaScript, TypeScript y Node.js. Visual Studio Code integra funcionalidades clave para incrementar la productividad del desarrollador, tales como el subrayado de sintaxis y errores, la sangría automática, la refactorización, el autocompletado, las sugerencias, la compilación y ejecución mediante clics, el depurador interactivo y la compatibilidad con repositorios Git. Tanto la funcionalidad como la compatibilidad con lenguajes son ampliables mediante extensiones.

Este ha sido el **IDE** de preferencia para el desarrollo de este trabajo, y se ha complementado con las siguientes extensiones:

- PlatformIO IDE⁶: habilita el desarrollo de software embebido para distintas plataformas y frameworks utilizando el sistema de PlatformIO (explicado posteriormente) [54].
- C/C++ Extension Pack⁷: facilita el desarrollo de proyectos en C/C++ añadiendo compatibilidad con estos lenguajes.
- GitHub Copilot⁸: una herramienta de programación basada en inteligencia artificial entrenada a partir de repositorios públicos y que ayuda a escribir código de forma rápida e inteligente [44]. Ofrece sugerencias de código inteligentes mientras se escribe y proporciona un chat para realizar consultas acerca del código, como pedir explicaciones, informarse acerca de conceptos de programación y guiar al usuario para mejorar su código o implementar nuevas funcionalidades [44] [8]. El uso de esta herramienta está limitado a los usuarios de pago o a quienes disponen de un GitHub Student Developer Pack [3], como ocurre en este proyecto.
- Better Comments⁹: añade diferencias de estilo y enfatizaciones a comentarios en el código destinados a alertar, hacer peticiones, indicar acciones por realizar o TODOs, o remarcar información importante, logrando así comentarios más comprensibles [30].
- TODO Highlight¹⁰ y Todo Tree¹¹: ambas extensiones se utilizan en conjunto para llevar un seguimiento de las tareas que se deben realizar en el código, permitiendo resaltar líneas que contienen el texto “TODO” y “FIXME” [48] y mostrarlas de manera agrupada en forma de árbol [45].
- Code Spell Checker¹²: un corrector ortográfico que marca los errores ortográficos detectados en los comentarios y en el código, y que ayuda a solucionarlos a través de sugerencias [62].
- Doxygen Documentation Generator¹³: permite generar automáticamente en formato Doxygen los comentarios junto a parámetros como las descripciones, los parámetros y los valores retornados para su uso en la documentación. Además, ofrece soporte para autocompletado y sugerencias de comandos

⁵<https://code.visualstudio.com/>

⁶<https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide>

⁷<https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools-extension-pack>

⁸<https://marketplace.visualstudio.com/items?itemName=GitHub.copilot>

⁹<https://marketplace.visualstudio.com/items?itemName=aaron-bond.better-comments>

¹⁰<https://marketplace.visualstudio.com/items?itemName=wayou.vscode-todo-highlight>

¹¹<https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree>

¹²<https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker>

¹³<https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen>

Doxygen [59].

```
{ https://code.visualstudio.com/docs/editor/whyvscode https://en.wikipedia.org/wiki/Visual_Studio_Code
}
```

PlatformIO¹⁴: es una herramienta de código abierto y multiplataforma, destinada a ingenieros y desarrolladores de software de sistemas embebidos, ya sean aficionados y profesionales, que incluye lo necesario para compilar, ejecutar, subir y escribir el código. Ofrece una amplia compatibilidad, con soporte para más de mil placas embebidas diferentes, más de 40 plataformas de desarrollo y más de 20 marcos de trabajo o frameworks.

PlatformIO aloja uno de los mayores registros de bibliotecas embebidas en PlatformIO Registry, lo que permite explorar e instalar de manera sencilla distintas bibliotecas de código, plataformas y herramientas, listadas junto a ejemplos e instrucciones de uso. Este registro se puede utilizar desde la interfaz gráfica de la herramienta, por línea de comandos y desde su página web¹⁵.

Una de sus características más importantes es la gestión de dependencias integrada. Es común que los proyectos aprovechen funcionalidades ofrecidas en bibliotecas, por lo que, tras ser referenciadas por el desarrollador, PlatformIO se encarga de resolver las dependencias al compilar el código. Además, contiene un depurador de código, un analizador estático de código, un monitor de puerto serial y soporte para pruebas unitarias. Esta herramienta se pone a disposición del usuario mediante la integración con otros IDEs o editores de texto a través de extensiones, siendo Visual Studio Code el más recomendado.

El código de este proyecto se ha creado con PlatformIO y se ha configurado para utilizar la placa ESP32 DEVKIT V1 con la plataforma Espressif32. En cuanto al framework, se ha utilizado el de Arduino, pese a ser posible también utilizar el de Espressif (ESP-IDF), debido a que cumple las necesidades del desarrollador, a la experiencia previa con placas de desarrollo Arduino por parte del autor y a la facilidad de uso. Las diferencias entre ambos son las siguientes:

- El framework ESP-IDF ofrece un soporte completo de los lenguajes C y C++, permitiendo escribir código eficiente y de alto rendimiento. Por otro lado, Arduino utiliza una implementación simplificada y adaptada a los microcontroladores, limitando la flexibilidad y funcionalidad del código.
- Las aplicaciones desarrolladas con ESP-IDF están preparadas para hacer uso de los núcleos disponibles en la placa y su estructura se basa en tareas, mientras que en Arduino por defecto solo se aprovecha de un núcleo y las aplicaciones siguen una estructura en la que se debe declarar una función `setup` y otra `loop`.
- El framework de Arduino es útil si previamente el desarrollador lo ha utilizado para desarrollar en otras placas, además de ser fácil de usar para quienes no tienen mucha experiencia, y contiene un gran rango de librerías y APIs por defecto que facilitan el desarrollo. En cambio, ESP-IDF se puede utilizar para desarrollar software que requiera controlar funciones avanzadas del hardware, como el consumo de energía y recursos, e incluye un mayor conjunto de herramientas para depurar la placa y gestionar el uso de la memoria.

¹⁴<https://platformio.org/>

¹⁵<https://registry.platformio.org/>

- En cuanto a la comunidad, debido a la implementación de Arduino en una gran variedad de placas, es la que mayor comunidad tiene en comparación con ESP-IDF.

{ <https://docs.platformio.org/en/latest/what-is-platformio.html> <https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide> <https://docs.platformio.org/en/latest/platforms/index.html> <https://docs.platformio.org/en/latest/librarymanager/dependencies.html> <https://docs.platformio.org/en/latest/integration/ide/index.html> <https://registry.platformio.org/> <https://www.espboards.dev/blog/esp-idf-vs-arduino-core/> }

Trello¹⁶: desarrollado por Atlassian, es una aplicación web que facilita la gestión de tareas utilizando tableros visuales al estilo Kanban. Los componentes clave son:

- Listas: columnas que representan las fases de una tarea.
- Tarjetas: tareas o ideas que se mueven entre listas según su estado. Pueden incluir miembros asignados, fechas de vencimiento, archivos adjuntos, descripciones, comentarios y checklists.
- Miembros: usuarios con tareas asignadas y permisos en el tablero.

Trello tiene la capacidad de ampliar la funcionalidad gracias a las integraciones con otras aplicaciones, a las automatizaciones sin código para centrarse únicamente en el trabajo, y a los powerups, que actúan como extensiones de la funcionalidad básica de Trello.

Para el proyecto, el alumno ha creado un tablero adaptado a sus necesidades (detallado posteriormente) usando el plan gratuito de Trello. Este tablero permite llevar un seguimiento de todas las tareas, tanto de código como de la memoria, realizadas durante el desarrollo, y ha facilitado las reuniones de seguimiento al poder mostrar a los tutores el estado de las tareas. Utiliza los siguientes *power-ups*:

- GitHub¹⁷: sirve para llevar un seguimiento de lo sucedido en GitHub desde el tablero, como adjuntar a tarjetas ramas, commits, incidencias y pull requests y asociar repositorios a tarjetas [7].
- Smart Fields¹⁸: permite crear campos personalizados en las tarjetas, como campos de texto, de número o de fecha, y soporta el uso de fórmulas para calcular el valor del campo. Estos campos se pueden mostrar desde la vista general del tablero, sin necesidad de entrar a ver los detalles de la tarjeta [21].

{ <https://en.wikipedia.org/wiki/Trello> <https://trello.com/es> <https://trello.com/es/about> <https://trello.com/es/tour> }

Doxygen¹⁹: utilizada para la documentación de este trabajo, esta herramienta de código abierto, muy empleada en el desarrollo de software, permite obtener documentación a partir del código de forma sencilla. Posibilita la generación automática de documentación en distintos formatos, como **HTML**, **PDF**, **Word** y **XML**, a partir de los comentarios insertados en el código durante el desarrollo, analizando la información de las distintas clases, funciones y variables. Gracias a esta automatización, se agiliza y estandariza el proceso de documentación de proyectos, lo cual es beneficioso para entender el proyecto y el código que lo compone, además de mejorar la colaboración entre los miembros del equipo de desarrollo y el mantenimiento del propio código.

¹⁶<https://trello.com/es>

¹⁷<https://trello.com/power-ups/55a5d916446f517774210004/github>

¹⁸<https://trello.com/power-ups/5e2212c3ba57415ef2ef9352/smart-fields>

¹⁹<https://doxygen.nl/>

Soporta C++, C, Python, Java, PHP y otros lenguajes. La documentación se realiza a partir de comentarios, como se muestra en el Listado 3.1.

Listado 3.1: Ejemplo de uso de Doxygen

```

1 /**
2 * <Descripción corta de la función>
3 *
4 * <Descripción larga de la función>
5 *
6 * @param parametro1 <Descripción del primer parámetro de entrada>
7 * @param parametro2 <Descripción del segundo parámetro de entrada>
8 * @param <Descripción del resto de parámetros>
9 * @return <Descripción del valor o valores que retorna>
10 */
11 public int ejemploFuncion(int parametro1, bool parametro2, ...)
```

Entre las capacidades adicionales se encuentran las referencias cruzadas entre distintas partes de la documentación, soporte de Markdown en los comentarios, dibujo de diagramas para representar gráficamente clases, herencias y relaciones entre partes del código, personalización de la documentación resultante y configuración mediante un fichero Doxyfile con distintos parámetros establecidos por el usuario.

En el caso de este trabajo, se ha generado documentación en formato **HTML** para crear una página web con toda la información de las distintas funciones que componen el proyecto, y se ha personalizado con Doxygen Awesome²⁰, un tema **CSS** aplicable a la página para disponer de una página con un aspecto moderno, limpio y compatible con la interfaz móvil.

{ <https://doxygen.nl/> <https://en.wikipedia.org/wiki/Doxygen> <https://jothepro.github.io/doxygen-awesome-css/> }

Plantilla de TFG²¹: fue desarrollada por Félix Albertos Marco, profesor del Grado de Ingeniería Informática de la sede de la **UCLM** de Talavera de la Reina, ofreciendo una alternativa para desarrollar la memoria del Trabajo de Fin de Grado. Esta plantilla, utilizada en este **TFG**, se caracteriza por emplear Markdown para redactar todos los apartados, evitando las complicaciones del formato del documento (como el interlineado, fuentes, cabeceras y pie de página y saltos de página), además de proporcionar la portabilidad del trabajo, ya que estos ficheros son de marcado ligero y no necesitan editores ni sistemas operativos específicos. El soporte de Markdown también incluye todos los elementos que este lenguaje ofrece, como tablas, listados, fragmentos de código o formateado del texto.

{ https://www.felixalbertos.com/resources/downloads/tfg_template.html }

Teams²²: es una aplicación multiplataforma de colaboración en equipo, disponible de forma gratuita con

²⁰<https://jothepro.github.io/doxygen-awesome-css/>

²¹https://www.felixalbertos.com/resources/downloads/tfg_template.html

²²<https://www.microsoft.com/es-es/microsoft-teams/group-chat-software/>

un plan personal limitado o mediante una licencia Microsoft 365.

Microsoft 365 es una suscripción que incluye las aplicaciones de Microsoft Office (Word, PowerPoint y Excel), almacenamiento en la nube, copias de seguridad y correo electrónico, que habilita el uso en la nube de éstas y permite mantener siempre la última versión con los parches de seguridad correctos mediante un pago mensual o anual. La **UCLM** provee a sus usuarios de cuentas de Microsoft 365 Empresa, la cual se ha aprovechado para este desarrollo.

Teams centraliza la comunicación en los equipos al permitir la interacción a través de mensajes instantáneos y reuniones. Su funcionamiento se basa en el concepto de equipos o *teams*, espacios de trabajo compartidos donde se intercambian mensajes, contenido y herramientas entre los miembros. Dentro de los equipos existen los canales, que son áreas para conversaciones sobre temas específicos en los que es posible restringir el acceso únicamente a ciertos miembros del equipo, permitiendo crear subdivisiones en el equipo. Por otro lado, para comunicaciones directas o en grupos pequeños, se pueden utilizar chats normales en lugar de equipos.

Además, Teams dispone de otras funciones, como aplicaciones instalables dentro del propio software para ampliar la funcionalidad, y un calendario con capacidad para programar reuniones.

Para este **TFG**, Microsoft Teams ha permitido mantener un contacto directo entre los tutores y el alumno a través de un equipo organizado por canales destinados a las partes de desarrollo y memoria. Estos han agilizado la consulta de dudas y de disponibilidad y organizado archivos compartidos. Por ejemplo, en el canal destinado a la memoria se alojaron las distintas versiones del documento de la memoria, mientras que en el canal de desarrollo se encuentran enlaces, manuales y códigos de ejemplo. Además, se han programado y celebrado reuniones de seguimiento en esta aplicación, de la cual tanto el alumno como los tutores tenían experiencia de uso.

{ <https://teamsdemo.office.com> https://en.wikipedia.org/wiki/Microsoft_Teams <https://www.microsoft.com/es-es/microsoft-teams/compare-microsoft-teams-business-options> <https://www.microsoft.com/es-ES/microsoft-365/buy/compare-all-microsoft-365-products> }

Visio²³: es un programa para crear diagramas y vectores profesionales de manera fácil gracias a su incluida y extensa librería de objetos. Su uso está limitado a los usuarios con licencia de Microsoft 365, servicio mencionado anteriormente y del cual la **UCLM** distribuye la licencia a todos los usuarios de la universidad, como es el caso del autor, quien ha destinado el uso de la aplicación al dibujo de algunos esquemas de la memoria. Este programa facilita la visualización de datos e ideas de manera atractiva, lo cual resulta útil en equipos y en la documentación para asegurar la comprensión sencilla de los conceptos.

{ <https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software#x68bca46524744e268ea489ad8cc29bbb> https://en.wikipedia.org/wiki/Microsoft_Visio }

Visual Paradigm²⁴: es una herramienta que soporta los diagramas resultantes de todas las fases del ciclo de vida del desarrollo de software, entre otros, los diagramas de casos de uso, de clases, de secuencia, de

²³<https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software>

²⁴<https://www.visual-paradigm.com/>

flujo y de arquitectura. Este programa facilita al usuario la realización de diagramas que cumplen con el Lenguaje Unificado de Modelado, ya sea de manera manual o a través de ingeniería inversa, es decir, a partir de código. Además, tiene la capacidad de generar código a partir de los diagramas creados, ayudando al desarrollo [36]. En este caso, se ha utilizado esta herramienta para ofrecer una comprensión visual de la solución mediante el modelado de una serie de diagramas integrados en esta memoria.

Navegador web, como Firefox²⁵ y Microsoft Edge²⁶: utilizados en este proyecto para acceder a las herramientas web mencionadas anteriormente y realizar búsquedas de información acerca de partes del código en desarrollo y del contenido de esta memoria.

/TODO: poner lo que vaya a utilizar para diagramas/

3.1.3. Lenguajes

C++: es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup para extender el ya existente C y añadir mecanismos de manipulación de objetos. Es un lenguaje que requiere una compilación para que el programa pueda ser ejecutado, además de ser multiparadigma, abarcando programación estructurada, orientada a objetos, genérica e imperativa (es decir, las instrucciones indican cómo realizar una tarea y se conoce el estado del programa durante la ejecución).

La orientación a objetos permite descomponer los proyectos en distintos archivos que contienen tipos de datos abstractos o clases. Estas son estructuras de tipos de datos concretos con un nombre definido. Además, permite asignar propiedades y funciones ejecutables a un objeto de esa clase y relacionar las propias clases con otras, por ejemplo, para que una clase pueda heredar de otra.

Tanto C++ como su predecesor C son lenguajes que requieren que el programador tenga claro qué hacer y cómo hacerlo, ya que permiten al programador expresar lo que quiere hacer sin restringir lo que está permitido. Son lenguajes simples, concisos y rápidos, pero, por otro lado, la compilación del código no comprueba las conversiones incorrectas de tipos, los índices erróneos de arrays ni el mal uso de punteros. Además, no cuentan con un recolector de basura que gestione la memoria automáticamente, por lo que el programador debe realizarlo manualmente. Esto convierte a C++ en un lenguaje frágil y exigente en la gestión de recursos.

C++ fue diseñado teniendo en cuenta la programación de sistemas, tanto grandes como embebidos y con recursos limitados, utilizando el rendimiento, la eficiencia y la flexibilidad como puntos clave de diseño.

```
{
    https://www2.eii.uva.es/fund_inf/cpp/temas/1_introduccion/introduccion.html      https://academia-lab.com/enciclopedia/programacion-imperativa/          http://cslibrary.stanford.edu/101/EssentialC.pdf
    https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-c/ https://en.wikipedia.org/wiki/C%2B%2B
}
```

En el caso de este trabajo, se utiliza el framework de Arduino, una implementación de este lenguaje que limita las funciones que C++ trae por defecto y se puede usar, e incluye algunas propias. Junto a este lenguaje

²⁵<https://www.mozilla.org/es-ES/firefox/new/>

²⁶<https://www.microsoft.com/es-es/edge/>

je, se han utilizado las siguientes librerías compatibles con la placa ESP32 y que la añaden funcionalidades y mejoran el desarrollo de código:

- FreeRTOS²⁷: esta biblioteca permite utilizar FreeRTOS como un sistema operativo en el proyecto, encargándose del acceso al hardware y de la gestión de las tareas. Es un sistema operativo en tiempo real, centrado en tener un control preciso del tiempo, y está destinado a ser utilizado en dispositivos embebidos.

FreeRTOS se basa en tareas definidas por el desarrollador, cada una con su frecuencia de ejecución. Estas tareas se ejecutan dependiendo de su estado, los cuales pueden ser: disponible, en ejecución, suspendida o bloqueada. El núcleo de FreeRTOS es el scheduler o planificador, que se encarga de gestionar y ejecutar las tareas. Este distribuye el tiempo de ejecución del procesador entre las tareas y asigna las tareas a los núcleos del procesador disponibles (solo se puede ejecutar una tarea por núcleo), permitiendo la ejecución en paralelo o alternando la ejecución entre varias tareas. Además, es el responsable de cambiar los estados entre las tareas.

FreeRTOS también ofrece otras funcionalidades y herramientas, como semáforos para sincronizar tareas, colas para compartir datos entre tareas, temporizadores y un bus de notificaciones.

```
{ https://es.wikipedia.org/wiki/FreeRTOS https://www.luisllamas.es/como-usar-freertos-en-arduino/
}
```

- bblanchon/ArduinoJson²⁸: es capaz de abstraer documentos JSON y las herramientas para serializarlos y deserializarlos, añadiendo una inexistente compatibilidad de C++ con este formato de datos estructurado. Este proyecto utiliza ArduinoJson 7.0.4 para crear ficheros en los que almacenar propiedades de objetos, como una lista de placa suscriptoras, y almacenarlos en una tarjeta microSD, para poder recuperarlos durante el arranque de la placa ESP32.
- x385832/Elog²⁹: creada para manejar eficientemente los logs o registros sin que impacte en el rendimiento de la ejecución, añade la capacidad de mostrar los registros por terminal serial, agregarlos a un fichero en una tarjeta SD y almacenarlos en la memoria flash. Admite distintos tipos de registros, dependiendo de cómo de crítico sea el mensaje, diferenciar mensajes por clases especificadas y mostrar marcas de tiempo. En el caso de este trabajo, se utiliza en la versión 1.1.5 para añadir mensajes de registro, como avisos o errores, para informar del estado de la ejecución del código, y ofrecer la posibilidad de almacenar los registros en una tarjeta microSD.

```
{ https://arduinojson.org/v7/faq/automatically-serialize-an-object/ https://www.luisllamas.es/en/arduinojson/ https://registry.platformio.org/libraries/x385832/Elog }
```

Markdown: es un lenguaje de marcado ligero utilizado en este TFG para redactar los distintos apartados de la memoria. Markdown permite escribir en documentos de texto plano, utilizando su propia sintaxis para indicar formatos especiales y el aspecto que debe tener (como negrita, cursiva o títulos), con la característica de mantener una lectura natural del documento en casos en los que no sea posible previsualizar el formato.

²⁷<https://www.freertos.org/>

²⁸<https://registry.platformio.org/libraries/bblanchon/ArduinoJson>

²⁹<https://registry.platformio.org/libraries/x385832/Elog>

A pesar de ser un lenguaje ligero, no limita el uso a únicamente texto, ya que permite insertar imágenes, tablas, listados y otros tipos de elementos.

{ <https://www.markdownguide.org/getting-started/> }

3.2. Metodología

En este apartado se tratan conceptos básicos de la metodología junto a la forma en la que se aplica a este TFG.

3.2.1. Metodologías tradicionales y ágiles en el desarrollo de software

Las metodologías de desarrollo de software son marcos, compuestos de técnicas y métodos, utilizados con el fin de estructurar, planear y controlar el proceso para aumentar la productividad y la calidad del software. Estas se deben elegir con precaución, ya que, si son apropiadas para el equipo y para el proyecto y su implementación se lleva a cabo correctamente, benefician al desarrollo con estimaciones superiores de tiempo y esfuerzo, priorización de tareas, clara comprensión de los esfuerzos futuros y previos, tiempo adicional suficiente para hacer ajustes, reportes al cliente el estado del proyecto y, además, entregas de sistemas y productos estables y con calidad [19] [50].

Las metodologías se organizan en dos grandes bloques: las tradicionales y las ágiles.

3.2.1.1. Metodología tradicional

Las metodologías tradicionales en el desarrollo de software se caracterizan por su enfoque estructurado y secuencial en la gestión de proyectos. Enfatizan la planificación exhaustiva y la fijación de los requisitos a lograr en las primeras etapas del proyecto, y siguen de manera lineal unas fases bien definidas, abarcando el inicio del proyecto, la planificación, la implementación, la verificación y el mantenimiento, durante las cuales se genera el producto junto a la documentación clara [50] [34].

Por un lado, ofrecen unos objetivos claramente definidos y sus procesos son controlables, pero, por otro lado, el desarrollo requiere más responsabilidad por parte de los involucrados y es poco tolerante a cambios tardíos, ya sean de requisitos o para resolver problemas, influyendo negativamente en el presupuesto y los plazos de entrega [50] [34] [63].

Entre la variedad de esta clase de metodologías se encuentran:

- Desarrollo en cascada: planificación y desarrollo lineal de etapas que entregan un único producto que cumple con los requisitos especificados. Es adecuado en proyectos con requisitos estables y entornos regulados por normativas [63].
- Proceso Unificado de Desarrollo: ciclos iterativos e incrementales centrados en casos de uso y en la gestión de riesgos. Su uso resulta adecuado en proyectos complejos y que necesitan una gestión de riesgos cuidadosa [63].

- Desarrollo en espiral: ciclos iterativos e incrementales formados por fases. Es utilizada en proyectos complejos con requisitos cambiantes y un alto nivel de incertidumbre o riesgos [63].

3.2.1.2. Metodología ágil

La metodología ágil es una metodología de gestión de proyectos que impulsa el desarrollo iterativo mediante entregas incrementales, la colaboración en equipo y la planificación y el aprendizaje continuos [47] [46] [66]. Está basada en el Manifiesto Ágil, acordado en 2001 por 17 desarrolladores de software agrupados como la Agile Alliance [66], y valora [26]:

- Los individuos e interacciones sobre los procesos y las herramientas.
- El software funcionando sobre la documentación extensiva.
- La colaboración con el cliente sobre la negociación contractual.
- La respuesta ante el cambio sobre seguir un plan.

En esta metodología, el desarrollo se realiza en ciclos iterativos breves o sprints de una duración entre 1 y 4 semanas, cuyo resultado es un pequeño incremento de la funcionalidad del software. En los sprints, el equipo realiza la codificación, las pruebas y la comprobación de calidad, fases regidas por requisitos bien definidos y priorizados de trabajo pendiente. En el desarrollo existe el rol del propietario del producto, [46] [66] quien se encarga de representar a las partes interesadas del proyecto y se compromete a responder a las preguntas de los desarrolladores a lo largo de las iteraciones [66]. El propietario del producto añade, modifica y reprioriza las tareas pendientes dentro de un listado o backlog al principio de cada sprint, en función de las necesidades del cliente [46]; y al final de cada iteración es quien se reúne con las partes interesadas para revisar el progreso del desarrollo y reevaluar la priorización de las tareas, con la intención de satisfacer a los interesados [66]. Además, las tareas con las que se trabaja deben ser claras para el equipo de desarrollo, por lo que requieren una continua refinación por parte del propietario y el equipo hasta que el propio equipo decida que están listas para trabajarlas [46].

Los incrementos permiten corregir constantemente errores que surjan y adaptarse a cambios, en vez de tener que enfrentarlos al final del proyecto o en etapas en las que puede resultar muy costoso [66].

Una característica fundamental del desarrollo ágil es la comunicación directa y eficaz entre los involucrados, lo que se traduce en la capacidad de comunicar de forma efectiva las necesidades del cliente y las tareas con el equipo de desarrollo. Es común apoyarse en un radiador de información que muestra el estado del proyecto, como un tablero con post-its o una pantalla grande. Esto se aplica a otra característica del desarrollo: la realización de sesiones de retroalimentación diarias para revisar el progreso, las tareas a completar en el día, los impedimentos y los riesgos. Son sesiones breves, de unos 15 minutos y sin entrar en detalle, ya que tienen el fin de reforzar la comunicación y la coordinación del equipo de desarrollo [66].

En esta metodología, es común utilizar herramientas y técnicas de integración continua y entrega continua que mejoren la calidad del proyecto y agilicen el desarrollo. Por ejemplo, automatizar la compilación, la ejecución de pruebas y el despliegue. Esto evita procesos lentos y propensos a errores, y permite demostrar un producto de calidad al final de cada iteración [46] [66].

La metodología ágil es adaptativa, es decir, se centra en adaptarse rápidamente a las necesidades cambiantes.

Durante la planificación se identifican hitos, pero estos son flexibles, además de que difícilmente se describe el futuro del desarrollo. Esta característica es muy importante, ya que es la que la separa de las metodologías tradicionales predictivas. Las metodologías predictivas planifican las tareas y características previstas en detalle y tienen en cuenta los riesgos conocidos, resultados de un primer análisis exhaustivo. En el caso de no haber tenido en cuenta algún riesgo o que ocurra algún otro imprevisto, puede ser difícil realizar cambios [66]. Esta comparación no intenta demostrar que una es superior a la otra, ya que ambas son adecuadas dependiendo de las necesidades específicas del proyecto, e incluso existe la posibilidad de realizar un enfoque híbrido combinándolas [42].

En adición, los principios del Manifiesto Ágil también enfatizan la colaboración continua entre los responsables de negocio y un equipo de desarrollo motivado y confiado a lo largo de todo el proyecto. Reflexionan y perfeccionan constantemente la efectividad técnica, con el fin de mantener un ritmo de desarrollo constante para alcanzar la principal medida de progreso: el software funcionando [27].

Dentro de este tipo de metodología se pueden encontrar, entre otros:

- Scrum: define de forma flexible roles, eventos, artefactos y reglas que los equipos deben seguir para mejorar la productividad, la calidad del trabajo y la comunicación. Los eventos principales que ocurren son los sprints iterativos e incrementales en los cuales existe un control continuo de la calidad del producto. Debe utilizarse por equipos de menos de 10 personas que sepan autoorganizar su trabajo. En grandes proyectos, el uso de esta metodología puede resultar en una pérdida de la perspectiva general durante el desarrollo. Además, aunque es compatible con todo tipo de proyectos, no es sencillo de integrar en todas las organizaciones [40].
- Kanban: se centra en la mejora del flujo de trabajo, de la productividad y de la calidad a través del uso de un tablero. El tablero muestra las tareas como tarjetas que se van desplazando entre las columnas para representar si están pendientes, en curso o concluidas. Estas columnas admiten priorización y limitación, evitando tener equipos sobrecargados. Las tareas se completan antes de comenzar otras, y ocurren reuniones regulares para obtener retroalimentación. Es una metodología fácil de integrar y que muestra los avances del proyecto de manera sencilla, pero requiere que el trabajo sea divisible en fases y que los miembros se adapten a trabajar en distintas etapas del proceso [39].
- Programación Extrema (eXtreme Programming): se basa en un entorno de comunicación constante entre desarrolladores y cliente, y en el cual existe respeto para tratar errores y críticas. Es utilizada en proyectos en los cuales el cliente no tiene una idea clara del producto final, por lo que ocurren procesos iterativos para entregar una versión y revisarla. Sin embargo, requiere una gran inversión de tiempo y disciplina para llevarla a cabo [38].

3.2.2. Scrum

Como está definido en la Guía de Scrum [60], Scrum es un marco de trabajo ligero para el desarrollo ágil de software que ayuda a personas, equipos y organizaciones a generar valor a través de soluciones adaptativas. Busca conseguir un equipo que trabaje en colaboración y obtener el mejor resultado posible de los proyectos, pero en vez de detallar instrucciones específicas a seguir, ofrece una guía de relaciones e

interacciones en el equipo. Esta libertad permite usar varios procesos, técnicas y métodos que visibilicen la eficacia de la gestión, el entorno y las técnicas de trabajo para poder mejorarlas.

Se basa en el conocimiento adquirido a partir de la experiencia, en la toma de decisiones basada en observaciones, y en el pensamiento centrado en lo esencial. Utiliza un enfoque iterativo e incremental en el cual un equipo con las habilidades necesarias para el proyecto celebra ciertos eventos que previenen y controlan riesgos, inspeccionan, adaptan y desarrollan el trabajo. Estos eventos están envueltos en uno principal, el sprint.

Los sprints son etapas de desarrollo que duran entre 1 y 4 semanas de trabajo (según se establezca en el equipo). En estos se convierten las ideas en valor para el producto, llamados incrementos, los cuales se producen con cada sprint y avanzan el desarrollo hacia el objetivo del producto final. Los incrementos pasan por una revisión con los interesados del proyecto, y se ajusta lo necesario para poder comenzar el siguiente sprint, uno detrás de otro.

Tanto el sprint como los otros eventos tienen utilidad, ya que se puede visibilizar el trabajo realizado en los artefactos, inspeccionar los mismos para detectar variaciones o problemas y adaptar los procesos y los materiales en caso de que el desarrollo se desvíe fuera de los límites aceptables o si el producto resultante es inaceptable.

El proceso de llevar a cabo Scrum se define por sus eventos, roles y artefactos, resumidos en inglés en la Figura 3.1 y que se detallan a lo largo de este apartado.

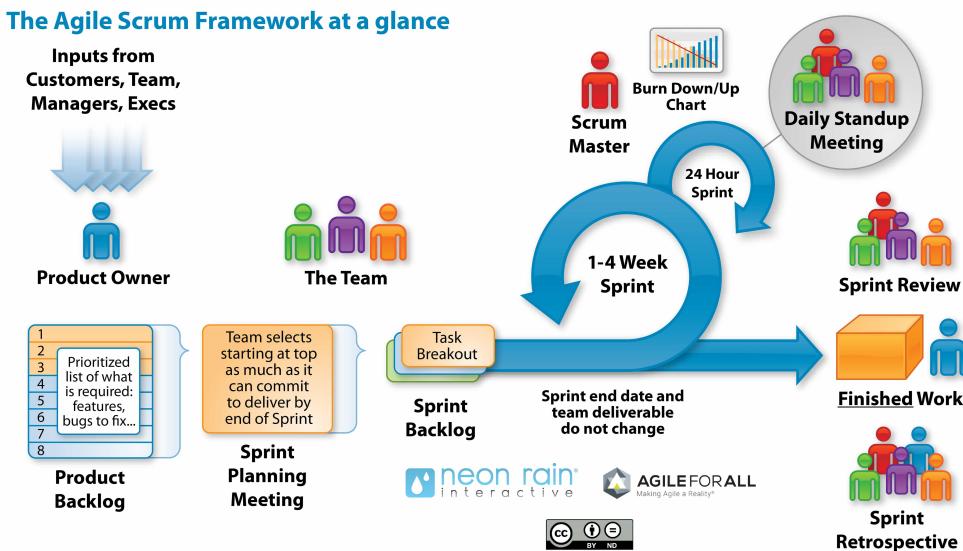


Figura 3.1: Esquema de la ejecución de Scrum (Fuente: [2])

<https://arrizabalagauriarte.com/en/10-principios-clave-metodologias-agile-scrum/>

Durante los sprints y en toda la implementación de Scrum, se encuentra una pequeña unidad de personas responsables: el equipo Scrum. Este equipo es multifuncional, y sus miembros poseen las habilidades necesarias para generar valor en cada sprint. Es autogestionado, asignándose internamente las tareas y la forma

de realizarlas, y está enfocado en el objetivo del producto. El equipo Scrum es responsable de llevar a cabo todas las actividades relacionadas con el producto (colaboración con interesados, mantenimiento, desarrollo, investigación y otras), además de crear un incremento útil y valioso en cada sprint. No tiene jerarquías ni subequipos, todos sus miembros están al mismo nivel. El tamaño ideal del equipo es de 10 personas o menos, pero no implica que no pueda haber más personas, en dicho caso se organizarán varios equipos Scrum. En el equipo existen tres roles principales:

- Equipo de desarrollo: grupo de personas comprometidas a crear cualquier aspecto de un incremento de calidad en cada sprint. Se encargan de crear el Sprint Backlog con las tareas a realizar durante el sprint y de adaptar su plan de trabajo diario para cumplir con el objetivo del sprint.
- Propietario del Producto o Product Owner: única persona responsable de maximizar el valor del producto resultante. Se encarga o se responsabiliza de que el Product Backlog contenga las tareas necesarias para alcanzar el objetivo del producto, representen las necesidades de los interesados, estén ordenadas y priorizadas, y sean comprendidas. Toda la organización debe respetar sus decisiones, y cualquier posible Product Backlog debe ser justificado ante él.
- Scrum Master: responsable de asegurar que Scrum sea comprendido y ejecutado tal y como está definido en la Guía de Scrum, tanto por el equipo de desarrollo como por la organización en su conjunto, por lo que puede ser común a toda la organización. Ayuda a comprender el trabajo, facilita su realización, elimina barreras y mejora las prácticas utilizadas.

Estos miembros, junto a los interesados, tienen una idea clara del estado del desarrollo y de las tareas realizadas a partir de los siguientes artefactos, tratados a lo largo de los sprints:

- Pila del Producto o Product Backlog: lista ordenada y emergente de las tareas a realizar necesarias para mejorar el producto. Estas tareas se refinan mediante su detallado y división en partes más pequeñas y precisas, listas para ser seleccionadas en la planificación del siguiente sprint. Esta lista se basa en el cumplimiento del objetivo del producto, que es el objetivo a largo plazo.
- Pila del Sprint o Sprint Backlog: lista de tareas creada por los desarrolladores, que representa el plan de trabajo para el sprint con el fin de lograr el objetivo del sprint. No es fija y se actualiza a lo largo del sprint.
- Incremento: representa un avance hacia el objetivo del producto. Se generan uno o varios incrementos al final del sprint, los cuales se añaden a los anteriores y se verifican para garantizar su correcto funcionamiento en conjunto. Un incremento se considera completo cuando cumple con los estándares de calidad definidos en la definición de terminado.

Una característica importante de Scrum es la celebración de eventos que permiten inspeccionar y adaptar los artefactos de manera formal. Estos eventos se utilizan para aplicar regularidad a las reuniones y minimizar la necesidad de reuniones no planificadas. La Guía de Scrum recomienda que siempre se realicen en el mismo lugar y a la misma hora para reducir la complejidad. Los distintos tipos de eventos son los siguientes:

- Sprint: además de lo mencionado anteriormente, cabe detallar algunas características de este evento. El sprint es una etapa en la que se procura mantener constante la calidad del desarrollo y no se realizan cambios que puedan comprometer el objetivo del sprint. Permite modificaciones en el Product

Backlog para refinar los elementos, así como aclaraciones y negociaciones con el Product Owner. La ejecución del sprint garantiza la inspección y adaptación del progreso hacia el objetivo del producto al menos una vez al mes. Su duración se justifica porque períodos más largos pueden hacer que el objetivo del sprint se vuelva inválido e incrementen la complejidad y los riesgos. Los sprints cortos limitan el riesgo de coste y esfuerzo a intervalos pequeños, comparables a proyectos de menor envergadura. Los sprints anteriores son útiles para la toma de decisiones y para pronosticar el progreso del sprint, mediante prácticas como el análisis de trabajo pendiente (burn-downs), de trabajo completado (burn-ups) y los flujos acumulativos. Existe la posibilidad de cancelar el sprint si su objetivo se vuelve obsoleto, y solo el Product Owner tiene la autoridad para hacerlo.

- Planificación del Sprint: sirve para establecer el trabajo que se realizará durante el sprint. Es el primer evento que se ejecuta antes de comenzar el sprint, en el cual el equipo Scrum colabora para crear un plan. El Product Owner debe asegurarse de que los asistentes estén preparados para discutir los elementos importantes del Product Backlog y su relación con el objetivo del producto. Los temas que se tratan son:
 - El valor que aporta el sprint y el incremento resultante al producto y a los interesados, definiendo un objetivo del sprint.
 - Las tareas a realizar durante el sprint, seleccionadas por los desarrolladores a partir del Product Backlog. El equipo Scrum puede refinar estos elementos para aumentar su comprensión y confianza.
 - El plan del trabajo necesario para crear un incremento que cumpla con la definición de terminado, elaborado por los desarrolladores.

Este evento no se limita únicamente al equipo Scrum, se pueden invitar a otras personas que asesoren al equipo.

- Scrum diario: es un evento diario de unos 15 minutos realizado durante el sprint en el que los desarrolladores inspeccionan el progreso hacia el objetivo del sprint y adaptan el Sprint Backlog según sea necesario. Los participantes informan brevemente sobre el progreso y generan un plan viable para el siguiente día laboral, identificando impedimentos, promoviendo una toma de decisiones rápida y eliminando la necesidad de reuniones no planificadas. Esto se traduce en un equipo bien comunicado, enfocado y autogestionado. El Scrum diario no tiene una estructura fija y no excluye la posibilidad de realizar discusiones más detalladas a lo largo del día.
- Revisión del Sprint: es una sesión de trabajo colaborativo que se realiza con el fin de inspeccionar los resultados al final del sprint y determinar futuras adaptaciones del proyecto. Durante este evento, el equipo Scrum presenta los resultados de su trabajo a los interesados y se discute el progreso hacia el objetivo del producto, incluyendo posibles ajustes al Product Backlog para satisfacer nuevas oportunidades.
- Retrospectiva del Sprint: es la última reunión del sprint, con el fin de planificar formas de aumentar la calidad y efectividad del desarrollo. El equipo Scrum analiza las interacciones entre personas, los procesos y herramientas, y la definición de terminado para identificar qué salió bien durante el sprint,

los problemas encontrados y la forma de resolverlos (en el caso de hacerlo). Tras ello, identifican cambios útiles para mejorar, y se abordan al momento o se añaden al Sprint Backlog.

3.2.3. OpenUP

OpenUP es definido [13] como una metodología de desarrollo de software iterativa e incremental que manifiesta el proceso de construir un sistema mediante el desarrollo colaborativo de software, con un enfoque pragmático y una filosofía ágil. A su vez, permite centrar el desarrollo en los casos de uso del usuario final, en la identificación y mitigación de riesgos, y en la arquitectura del sistema desde las primeras fases. Esta metodología está concebida para ser utilizada como una base, conteniendo los elementos fundamentales para el desarrollo, y se le puede añadir o adaptar el contenido de otros procesos según las necesidades del proyecto.

OpenUP se apoya en equipos saludables, colaborativos y con conocimiento compartido acerca del proyecto, en maximizar los beneficios y cumplir con los requisitos, y en utilizar prácticas que permitan obtener retroalimentación temprana y continua a partir de incrementos en la funcionalidad del producto.

Los proyectos que implementan OpenUP se dividen en iteraciones cortas planificadas. El uso de las iteraciones evita la generación innecesaria de documentación, diagramas e iteraciones excesivas, y facilita la detección temprana de errores, haciendo más eficiente el uso de recursos y tiempo. Durante las iteraciones, el equipo se organiza y se compromete para alcanzar los objetivos. Se define una lista de tareas detalladas que resultan en microincrementos y en la entrega de valor a las partes interesadas. Los microincrementos se producen a un ritmo constante y permiten comprender el valor que aporta la iteración al proyecto y el progreso del mismo.

OpenUP organiza las iteraciones en cuatro fases del ciclo de vida del proyecto, compuestas por objetivos y actividades que permiten gestionar eficientemente el proyecto:

- Inicio: se determina el alcance del proyecto y los objetivos a cumplir. Tras esta fase, si se considera viable, el proyecto se continúa.
- Elaboración: trata la mitigación de riesgos, la arquitectura del sistema, la validación de requisitos, el diseño, los costos y los cronogramas.
- Construcción: detalla los requisitos y su cumplimiento, el diseño, la implementación de funcionalidades y las pruebas del software.
- Transición: se centra en la transferencia del software al entorno del cliente, en la aceptación y conformidad del producto por parte de las partes interesadas, en la corrección de defectos y otros ajustes finales.

3.2.4. Implementación en este proyecto

Tras la definición de los conceptos necesarios acerca de las metodologías, esta sección desarrolla la manera en la que se aplicaron Scrum y OpenUP. El desarrollo de este proyecto se ha realizado teniendo OpenUP en mente, mientras que el esfuerzo de este proyecto se ha gestionado utilizando la metodología Scrum. Pese a conocer que la aplicación no es la más adecuada al ser un equipo más pequeño de lo normal y

en el que existe un único desarrollador, la aplicación de esta metodología procura ser fiel con todos los elementos esenciales que la componen. Se ha optado por metodologías ágiles debido a que es un proyecto cuyo desarrollo comienza con unos requisitos básicos y atraviesa fases en las que se identifican fallos de rendimiento, posibles mejoras y nuevos requisitos. Una característica importante de estas metodologías es la constante retroalimentación de los interesados, que permiten redirigir el proyecto según transcurra, y la baja formalidad en cuanto a documentación permitía centrar el esfuerzo en el funcionamiento del proyecto. Por otro lado, la elección ha sido influida por la preferencia y la previa experiencia del autor con estas metodologías.

3.2.4.1. Roles y artefactos

En este proyecto han participado los tutores y el alumno con los roles distribuidos de la siguiente manera:

- Interesados: los tutores Rubén Cantarero Navarro y Ana Rubio Ruiz.
- Equipo de desarrollo: compuesto únicamente por el alumno Rubén Gómez Villegas.
- Product Owner: los tutores Rubén Cantarero Navarro y Ana Rubio Ruiz.
- Scrum Master: tanto el alumno como los tutores. Debido a que ambas partes tienen conocimiento acerca de Scrum y llegaron a un acuerdo acerca de como gestionar este proyecto, no ha sido necesaria la intervención de este rol.

Los distintos artefactos Scrum se han representado en un tablero Kanban creado con Trello para facilitar el seguimiento de la metodología. Un tablero Kanban funciona como una herramienta ágil para gestionar proyectos, ayudando a visualizar el trabajo, limitar el trabajo en curso y maximizar la eficiencia del mismo [57] [24], además de tener equipos comprometidos con una cantidad de trabajo adecuada [57]. Está compuesto por tarjetas y otras señales visuales que describen el trabajo, permitiendo que los compañeros y los interesados conozcan su estado [57]. Las tarjetas están agrupadas en columnas que representan etapas del flujo de trabajo, y se van moviendo entre columnas según avanza la tarea hasta su finalización [57] [24].

El tablero de Trello utilizado, mostrado en la Figura 3.2, es una representación digital de Kanban, creado por y para el alumno y las necesidades del desarrollo, y contiene las siguientes columnas:

- *Info, templates and utils*: contiene información del tablero y una tarjeta que actúa como plantilla para representar tareas.
- *Product backlog*: contiene las tareas propias de un Product Backlog de Scrum.
- *Sprint backlog*: contiene las tareas propias de un Sprint Backlog de Scrum.
- *In progress*: lista las tareas que se están ejecutando a lo largo del sprint. Son tareas que aún no están finalizadas.
- *Review/Testing*: representa las tareas que se han terminado de desarrollar pero que deben pasar por una revisión de su correcto funcionamiento y otra del cumplimiento con las necesidades de los interesados, siendo esta última ejecutada durante la revisión del sprint.
- *Done (Sprint X)*: cuando las tareas están terminadas, acaban en esta lista, en la cual se marcan apropiadamente como finalizadas. Existe una lista por cada sprint, para poder diferenciar qué tarea se hizo cuándo, siendo “X” el número del sprint.

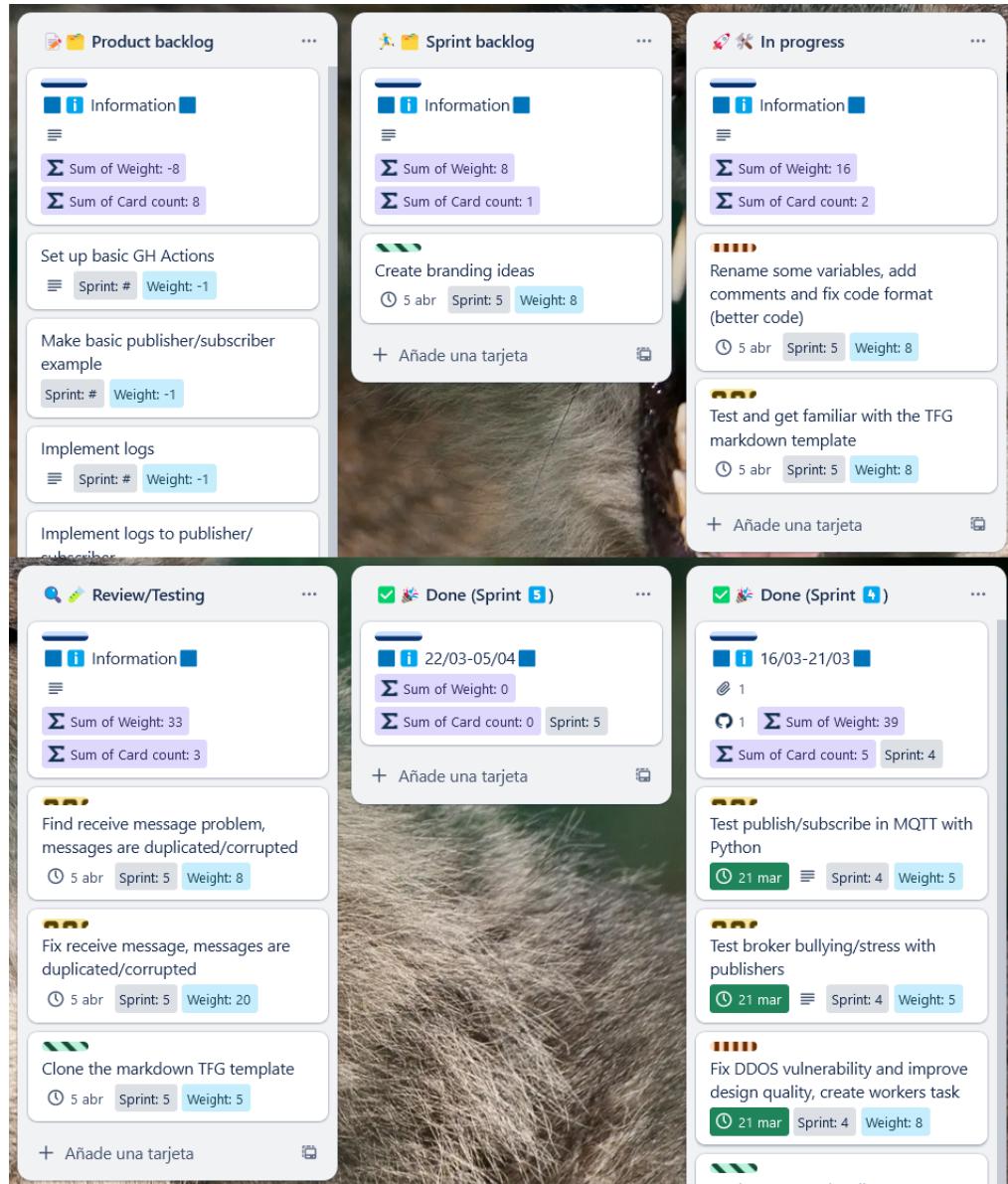


Figura 3.2: Tablero Kanban utilizado durante el desarrollo

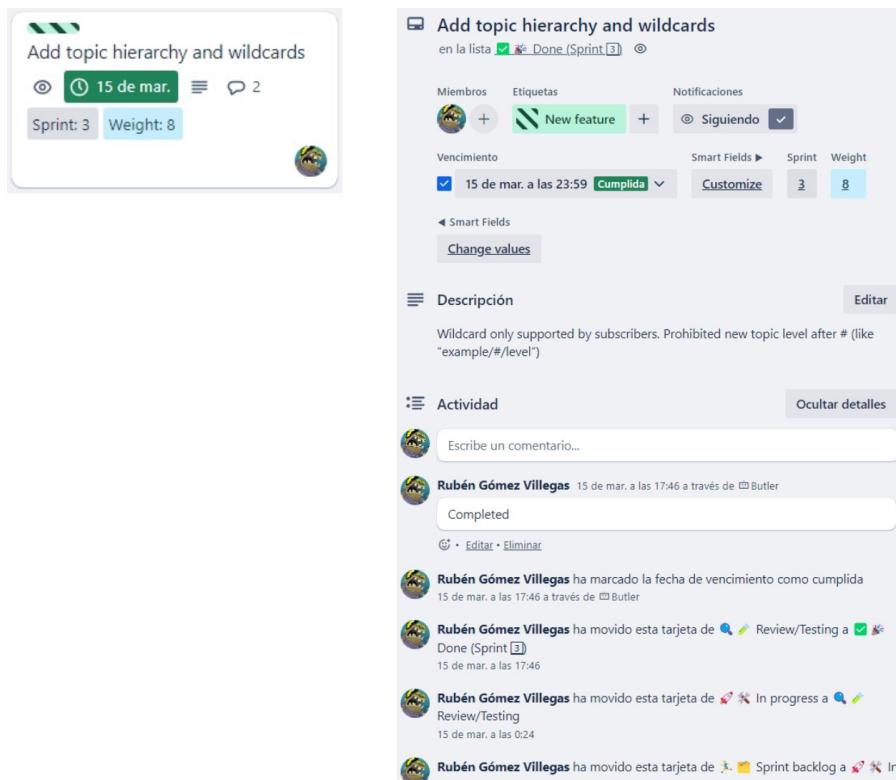


Figura 3.3: Vistas de las tarjetas del tablero: vista previa (izquierda) y detallada (derecha)

Las tarjetas representan tareas, y se muestran en dos vistas, mostradas en la Figura 3.3: la vista previa, que muestra la tarjeta resumida en una columna, y la vista detallada, la cual se abre al hacer clic en la tarjeta resumida. Las tarjetas tienen las siguientes propiedades configuradas:

- Miembros: quienes se encargan de la tarea. Son usuarios registrados en Trello y miembros del tablero que pueden autoasignarse o ser asignados por otros.
- Etiquetas: se utilizan para clasificar de manera visual el tipo de tarjeta. Existen 6 tipos distintos, cada uno con un color y patrón (útil para las personas con daltonismo) como se puede ver en la Figura 3.4, siendo posible aplicar varios simultáneamente. Son los siguientes:
 - *New feature*, verde: la tarjeta representa una tarea cuyo resultado es una nueva funcionalidad en el proyecto.
 - *Test*, amarillo: la tarjeta representa una tarea que prueba funciones y elementos ya existentes en el proyecto.
 - *Fix*, naranja: la tarjeta representa una tarea que arregla funciones y elementos ya existentes en el proyecto.
 - *Blocked*, rojo: la tarjeta representa una tarea pausada debido a un impedimento u obstáculo, como una dependencia de otra tarea.
 - *Details card*, azul: la tarjeta muestra información, no representa ninguna tarea.
 - *Report*, rosa: la tarjeta representa una tarea relacionada con esta memoria del proyecto.



Figura 3.4: Etiquetas disponibles en el tablero

- Vencimiento: fecha límite de realización de la tarea.
- Sprint: un campo personalizado que representa el número del sprint de la tarea.
- Weight: un campo personalizado numérico que representa el número del sprint de la tarea. El weight o peso es un número subjetivo que representa la dificultad, el tiempo a dedicar y la experiencia o falta de esta por parte del desarrollador respecto a una tarea.
- Card count: un campo personalizado numérico utilizado para contabilizar la cantidad de tarjetas en una lista. Siempre tiene el valor 1 y está oculto.
- Descripción: en el caso de necesitar detallar o añadir información adicional a las tareas, existe este campo.
- Actividad: muestra las acciones que han ocurrido (como el paso de una lista a otra o la asignación de miembros) y permite a los miembros del tablero hacer comentarios.

En este tablero se pueden identificar dos tipos de tarjetas: las tarjetas de información y las de tareas. Las de tareas se crean a partir de una plantilla y se mueven entre las listas durante los sprints. En cambio, las de información son fijas y sirven para detallar el uso de una lista y resumir su contenido. Éstas se encuentran en la primera posición de las listas, tienen la etiqueta azul y dos campos que se pueden observar desde la vista previa:

- Sum of Weight: calcula la suma de los pesos de una lista de tareas. Útil para calcular el peso total de un sprint.
- Sum of Card count: calcula la cantidad de tareas en una lista. Debido a que Trello por defecto no proporciona una forma de realizar esto, el campo es útil en las listas en las que no caben todas las tarjetas en pantalla, además que sirve para comparar entre sprints la cantidad de tareas hechas.

Además, las tarjetas de información en los sprints indican la fecha de inicio y fin del sprint. En el caso de ser un sprint finalizado, se le adjunta el pull request realizado en GitHub para conocer fácilmente el incremento del producto al que dio lugar.

En el tablero se ha habilitado la automatización que ofrece Trello a través de la selección de varios triggers

o disparadores y de acciones consecuentes relacionadas con las propiedades de las tarjetas y las acciones realizables sobre ellas. En este proyecto, las reglas activadas son las siguientes, mostradas en la Figura 3.5:

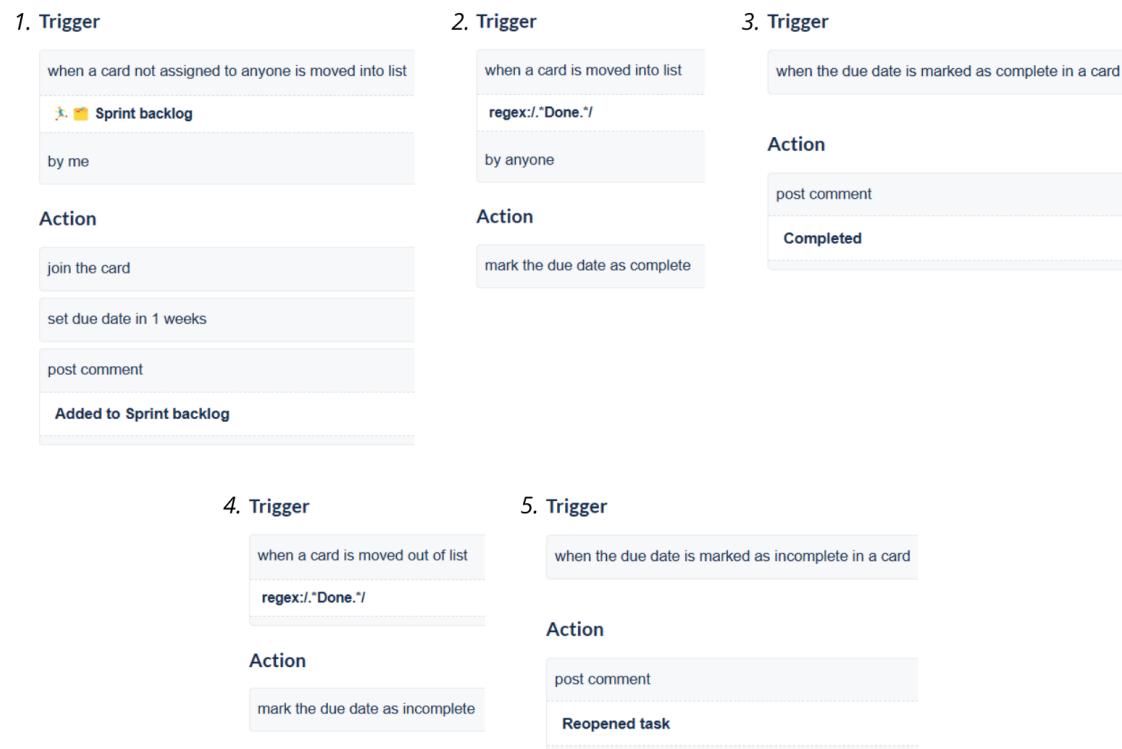


Figura 3.5: Reglas de automatización utilizadas en el tablero

1. *Cuando una tarjeta sin miembro asignado se mueve al Sprint backlog por un miembro X, el miembro X se le asigna, se establece la fecha de vencimiento a 1 semana y se añade el comentario “Añadido al Sprint backlog”:* automatiza la selección de una tarea para ejecutarla durante el sprint.
2. *Cuando una tarjeta se mueve a cualquier lista “Done” por cualquiera, marcar la fecha de vencimiento como completa:* automatiza el completado de una tarea.
3. *Cuando la fecha de vencimiento de una tarjeta se marca como completa, se añade el comentario “Completado”:* automatiza el registro del completado de una tarea.
4. *Cuando una tarjeta se mueve fuera de cualquier lista “Done” por cualquiera, marcar la fecha de vencimiento como incompleta:* automatiza la reversión del completado de una tarea.
5. *Cuando la fecha de vencimiento de una tarjeta se marca como incompleta, se añade el comentario “Tarea reabierta”:* automatiza el registro de la reapertura de una tarea.

Cabe recalcar que este tablero lo ofrece el alumno para su uso en cualquier desarrollo de software, y que en este caso su contenido está en inglés debido a que su uso ha sido en gran parte para gestionar la parte codificada, también en inglés, de este TFG.

3.2.4.2. Ejecución de los eventos

Los eventos definidos en **Scrum** se han realizado mediante Teams, donde se programan las reuniones para crear recordatorios y llevarlas a cabo, evitando el consumo de tiempo que supone para el alumno y los tutores el desplazamiento, y facilitando la manera de mostrar código y otros aspectos del proyecto. Se han ejecutado de la siguiente forma:

Planificación del sprint: antes de iniciar el sprint, el desarrollador y los Product Owners comparan los objetivos del proyecto y las necesidades con las tareas ya realizadas, y debaten sobre qué tareas nuevas añadir en forma de tarjetas al product backlog para realizar a lo largo del desarrollo. Estas tareas también se pueden basar en ideas surgidas tanto por el desarrollador como por los propietarios. Asimismo, se planifica el fin del sprint, limitando el tiempo a 1 semana si el incremento se basa en el código del producto, o a 2 semanas si el incremento se basa en la documentación y la memoria de este proyecto; y se consideran las tareas para añadirlas al sprint backlog y así realizarlas en el sprint entrante, con el suficiente refinamiento y que resulten en uno o varios incrementos previamente debatidos. Estas tareas son añadidas a la lista correspondiente del tablero y el desarrollador rellena los campos de manera apropiada, incluyendo el peso de la tarea, siendo este un valor útil para conocer si es demasiada tarea a realizar en un sprint. Normalmente, si el equipo de desarrollo fuera de mayor tamaño habría que llevar a cabo procesos como el *planning poker* para estimar un peso con el que el equipo esté de acuerdo. Pero como no es el caso, el desarrollador elige un valor de la secuencia modificada de Fibonacci: 0.5, 1, 2, 3, 5, 8, 13, 20, 40 o 100. Como parte final del sprint, se programan las demás reuniones y la próxima planificación del sprint. Finalmente, en el tablero Kanban se prepara una nueva lista *Done* con el número del sprint.

Sprint: a lo largo del sprint, el desarrollador ejecuta las tareas establecidas en el sprint backlog haciendo uso de las herramientas correspondientes. En cuanto comienza una tarea, la tarjeta es añadida a la lista *In progress* y se moverá hacia la lista *Review/Testing* cuando se finalice. Dentro de esta fase, la tarea puede pasar por pruebas realizadas dentro del sprint, y si se reconsidera como no completada, se mueve de vuelta a *In progress*. Dichas tareas se cumplen siguiendo la calidad definida en los siguientes apartados, y durante su progreso se irán realizando *commits* o publicaciones con los cambios al repositorio, concretando más, en la rama *develop* (se explica acerca de esta rama en posteriores apartados). A modo de añadir métricas de tiempo invertido en el desarrollo, los sprints se han realizado a lo largo de los días de la semana, con una jornada no fija de entre 1 y 8 horas diarias.

Scrum diario: no se ha realizado debido a que el equipo de desarrolladores está formado únicamente por el autor y a que los otros roles de Scrum ya conocen la tarea a realizar a lo largo del sprint por la reunión de planificación. Por otro lado, los tutores han mostrado la predisposición de atender lo antes posible cualquier consulta, ya sea a través de llamada rápida o mensaje de Teams, siendo esto un buen sustitutivo del Scrum diario.

Revisión del sprint: el desarrollador y los product owners revisan el estado final de las tareas establecidas en el sprint, además de probar la nueva funcionalidad del incremento. Tras una posible modificación rápida, las tarjetas de tareas del tablero Kanban son movidas a la columna *Done* del correspondiente sprint, marcándose así como completadas, y se realiza un pull request desde la rama *develop* a *main*, aumentando

la versión del producto (ramas y versionado explicados en posteriores apartados). En el caso de ser una versión superior a la 1.0.0, se ejecuta el proceso de crear un release en el repositorio, subir la nueva versión del producto al registro de PlatformIO y generar la página web de documentación.

Retrospectiva del sprint: utilizadas para comentar el rendimiento de los sprints, la idoneidad de la cantidad de tareas, mejoras en la gestión del proyecto y en las propias reuniones. Por ejemplo, en una reunión de este tipo se debatió el paso de sprints de 1 semana a 2 semanas cuando las tareas están relacionadas con la redacción y no con el código, debido a que las primeras consumían más tiempo. No existía un procedimiento para el desarrollo de esta reunión, simplemente el tema se exponía y se resolvía en consecuencia.

3.2.5. Control de versiones

En este apartado se detalla la forma en la que se ha implementado el control de versiones, una característica beneficiosa en el desarrollo de software, ya que permite a los equipos de desarrollo trabajar en paralelo evitando conflictos en el código. Además, facilita el rastreo y la gestión de los cambios realizados a lo largo del desarrollo, y permite comparar el estado actual del código fuente con versiones previas para resolver errores [29]. Para este TFG, se han aprovechado las herramientas que ofrece GitHub, como las ramas, los pull requests y las liberaciones o releases.

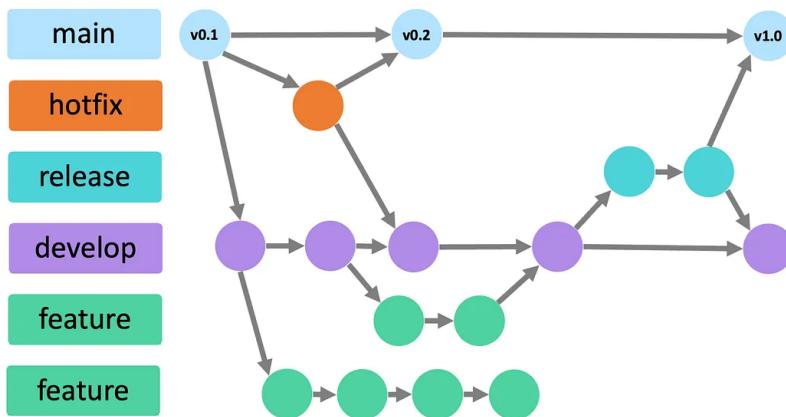


Figura 3.6: Ilustración de Gitflow (Fuente: [65])

En cuanto a la ramificación del repositorio, se utiliza una adaptación del flujo de trabajo Gitflow, un modelo de creación de ramas publicado en 2010 por Vincent Driessen, útil para proyectos de entrega continua [28]. Según la publicación [35], y para detallar la Figura 3.6, existen dos ramas principales que perduran a lo largo del tiempo: `master` o `main`, que registra el historial de publicación oficial del proyecto, y `develop`, en la cual se van integrando las funciones para la siguiente liberación del código. Cuando el código de la rama `develop` se encuentra en un punto estable y listo para la liberación, todos los cambios se fusionan hacia la rama `master`, lo que se traduce en una nueva versión del producto que se etiqueta. Junto a estas ramas, se utilizan otras con un tiempo de vida limitado y que en algún momento se eliminarán:

- Ramas de características o `feature`: se utilizan para desarrollar nuevas características para una fu-

tura versión. Se crean a partir de `develop`, existen mientras la característica está en desarrollo, y finalmente se fusionan de nuevo hacia `develop` añadiendo la característica, o se descartan.

- Ramas de lanzamiento o *release*: nombradas `release-<next-version>`, se utilizan para preparar una nueva versión de producción. Se crea una a partir de `develop` cuando este último refleja las características destinadas a la versión a construir, se le asigna un número de versión, se mantiene la rama para añadir cambios ligeros y metadatos, hasta que finalmente está lista para el lanzamiento y se fusiona en `master`, representando una nueva versión. También se fusiona hacia `develop`, para mantener esos pequeños cambios realizados a esta rama.
- Ramas de parche rápido o *hotfix*: nombradas `hotfix-<next-version>`, se utilizan para resolver un fallo crítico en una versión de producción. Se crea a partir de `master`, se solventa el problema y finalmente se fusiona en `master`, aumentando la versión del producto, y en `develop`.

El uso de Gitflow por parte del alumno proviene de la previa experiencia utilizándolo y la compatibilidad que tiene con las metodologías del proyecto, pero en este caso difiere respecto a lo definido anteriormente:

- No se utilizan ramas de características. Al ser un equipo de desarrollo de un solo miembro y al no haberse planeado el desarrollo de distintas características en paralelo (la complejidad del proyecto no lo requiere), se ha considerado innecesaria la creación de este tipo de ramas, por lo que todos los cambios se van publicando a la rama `develop`.
- No se utilizan ramas de lanzamiento. Para simplificar el desarrollo, la única rama que contiene los lanzamientos es `main`, y en caso de realizar un ligero cambio en el código se crean ramas *hotfix*. `develop` se fusiona directamente hacia `main`, y si luego ocurre un *hotfix* en `main`, `main` se fusiona hacia `develop`.

Para etiquetar las versiones se ha utilizado un versionado semántico, que sirve para lanzar versiones que puedan indicar a los sistemas dependientes si dicha versión puede romper el funcionamiento del sistema. En su definición [55] se trata el formato de la versión, que es X.Y.Z, formado por números enteros no negativos ni precedidos por ceros, y el incremento de versiones está definido de la siguiente forma:

- X, versión mayor: se han realizado cambios incompatibles. Cada incremento de X provoca establecer Y y Z a 0.
- Y, versión menor: se ha añadido funcionalidad compatible con versiones anteriores. Cada incremento de Y provoca establecer Z a 0.
- Z, versión parche: repara errores compatibles con versiones anteriores.

La primera versión lanzada al público se define como 1.0.0, y es posible agregar identificadores y metadatos a versiones prelanzamiento. Por ejemplo, 1.0.0-alpha+001.

Se ha preferido el versionado semántico sobre otros, como el versionado de calendario (usado por Ubuntu, cuya versión 24.04 muestra que salió en abril de 2024). Por ejemplo, ante el de calendario, es preferible este uso para que la versión no muestre una sensación de obsolescencia al utilizar una versión lanzada hace años ni una falta de madurez al utilizar una versión lanzada recientemente. De manera general, su uso se debe a la facilidad, la comprensión y a la previa experiencia del alumno.

3.2.6. Criterios de calidad y estilo del código

A lo largo del desarrollo del código, se han seguido unos criterios de calidad establecidos por el autor con el fin de obtener un código comprensible tanto para desarrolladores como para usuarios, fácil de mantener y retomar en el futuro. Además, se ha procurado mantener un estilo homogéneo del código, también definido en este apartado. Los criterios son los siguientes:

- El código está escrito en inglés, incluyendo variables, funciones, comentarios y mensajes de registro. Al utilizar el idioma más hablado en el mundo [11], la base de usuarios puede ser mucho más amplia que si se utilizara el idioma local.
- El código es autoexplicativo. En caso de ser complejo de comprender, hay comentarios que facilitan su entendimiento.
- Gran parte de las funciones están documentadas utilizando Doxygen, especificando una descripción, parámetros (en caso de existir) y valores de retorno (en caso de tenerlos). También se documentan otras partes del código, como las enumeraciones y los valores configurables.
- Las importaciones, enumeraciones, definiciones de valores, documentación y otros elementos considerados necesarios se muestran en archivos de cabecera (.h), evitando sobrecargar los archivos de código fuente (.cpp) con elementos no funcionales.
- El funcionamiento del código es eficiente, aprovechando las capacidades de la placa de desarrollo y del lenguaje de programación. Para ello, se realizan pruebas y consultas con los tutores y con herramientas de inteligencia artificial.
- Las líneas de comentarios no superan los 80 caracteres de longitud y las de código no superan los 120. En caso de necesitar más espacio, se introducen saltos de línea. Estos valores se originan a partir del primer almacenamiento digital de la historia de la informática, las tarjetas perforadas de 80x12 agujeros, y en la actualidad se limitan los caracteres por fila para mejorar la legibilidad del código sin necesidad de hacer scroll horizontal [53].
- El estilo de programación utilizado es el *Kernighan & Ritchie*, que evita líneas sin sentido y el consumo innecesario de espacio vertical en la sintaxis. Junto a este estilo, se emplea la indentación mediante tabulaciones y la convención de nombrado *lowerCamelCase*, útil para identificar distintas palabras al nombrar elementos del código sin necesidad de ampliar horizontalmente el código. El Listado 3.2 muestra un ejemplo de este estilo.

Listado 3.2: Ejemplo de estilo del código Kernighan & Ritchie con lowerCamelCase

```

1 public int main(int argc, char *argv[]) {
2     while (x == y) {
3         doSomething();
4         doSomethingElse();
5         if (error)
6             fixError();
7         else
8             continueAsNormal();

```

9 }

10 }

- La ejecución muestra registros o *logs* si el usuario lo desea, los cuales se distribuyen por el código en puntos clave. Estos registros permiten realizar un seguimiento de lo que realiza la ejecución del código del proyecto e identificar fácilmente los fallos.

También, como parte de la calidad del código, se han establecido pruebas para comprobar el correcto funcionamiento del código, las cuales están detalladas en el Anexo /TODO: insertar numero/.

Capítulo 4

Resultados

En este capítulo se describe el avance del desarrollo del proyecto a través de iteraciones o sprints, mencionando los logros y problemas identificados en cada uno. Además, se describe y muestra el funcionamiento de la herramienta desarrollada en su última versión a fecha de redacción de esta memoria. Este proyecto, nombrado “*LoboMQ*” durante el desarrollo, se puede encontrar en el siguiente repositorio:

<https://github.com/rubnium/LoboMQ>

Las iteraciones se comenzaron el 28 de septiembre de 2023, luego de evaluar la viabilidad del proyecto y de recopilar los requisitos y objetivos del mismo entre los tutores y el alumno, así como de establecer las metodologías y herramientas a utilizar. Estas iteraciones se desarrollaron en orden y se representan en la Figura 4.1, mostrada de manera optimizada a partir de febrero de 2024.

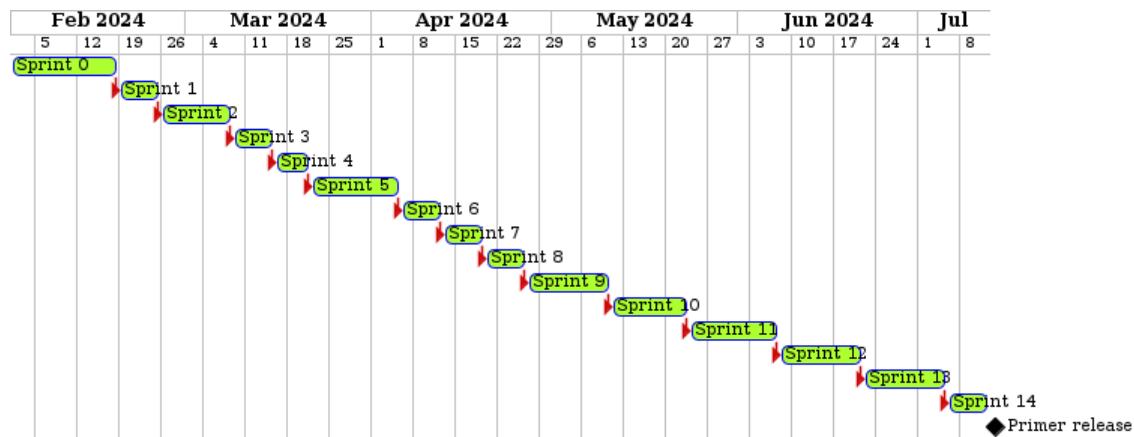


Figura 4.1: Diagrama de Gantt del proyecto desde febrero de 2024

4.1. Iteración 0

Periodo: 28/09/2023 - 18/02/2024 (20,43 semanas). Peso total: 143.

El objetivo principal de este sprint fue la preparación previa al desarrollo y la obtención de conocimientos

necesarios. Esta es la iteración más larga y con menor progreso por semana debido a la necesidad de compaginar las asignaturas del grado con este desarrollo y a la lenta pero eficaz captación de conocimientos por parte del alumno.

Tareas:

- Preparar el entorno de desarrollo (peso: 3). Instalar las herramientas y crear el repositorio.
- Familiarizarse con ESP32 y PlatformIO (peso: 3). Aprender a utilizar PlatformIO para desarrollar programas y subirlos a la placa de desarrollo.
- Escribir el anteproyecto del **TFG** (peso: 5).
- Comunicar placas ESP32 a través de ESP-NOW (peso: 2). Aprender a enlazar las placas a través de sus direcciones **MAC** y comprender el uso de las funciones para formar y emitir un mensaje (procurando no sobrepasar el tamaño máximo del payload) y para extraer los datos al recibirlo.
- Entender y realizar ejemplos de FreeRTOS (peso: 100). Leer la documentación de FreeRTOS y comprender los distintos conceptos que abarca, como las tareas (con sus estados y prioridades), las colas y los semáforos.
- Realizar ejemplos emisor-receptor con ESP-NOW y FreeRTOS (peso: 5). Aplicar el conocimiento de ambos aspectos para crear tareas que envíen y reciban datos aprovechando los hilos de ejecución disponibles.
- Realizar ejemplos publicador-suscriptor con **MQTT** (peso: 5). Desplegar un broker Mosquitto en Docker y crear un código para que la placa ESP32 publique mensajes, mientras que otro permite obtenerlos. Esto permite conocer los tipos y parámetros al trabajar con mensajes y otras capacidades en **MQTT**.
- Preparar tablero Kanban (peso: 20). Crear el tablero, las columnas y las plantillas, junto a la configuración necesaria para la gestión del proyecto a lo largo de todo el desarrollo.

4.2. Iteración 1

Periodo: 19/02/2024 - 25/02/2024 (1 semana). Peso total: 26.

El objetivo principal de este sprint fue realizar una primera versión funcional centrada en los roles de broker y suscriptor, sin tener en cuenta los topics.

Tareas:

- Crear el esqueleto de la librería (peso: 5). Preparar los ficheros necesarios para definir una librería en PlatformIO. En esta iteración, la librería contenía el fichero de configuración y el de código para definir un broker vacío.
- Definir la estructura de los mensajes (peso: 8). La primera definición de los mensajes se realizó fuera de la librería, en ficheros .h que formaban parte de un ejemplo. En esta definición se distinguen tres tipos de mensajes: suscripción, desuscripción y publicación, que eran contenidos en un mensaje genérico con un campo de byte `msgType` para identificar el tipo de mensaje. Tanto los mensajes de suscripción como los de publicación contenían únicamente un campo `topic`, definido con capacidad

para 10 caracteres, y los de publicación además contenían un campo `content` con capacidad para 20 bytes. El código de la definición se muestra en el Listado 4.1.

Listado 4.1: Primera definición de los mensajes

```

1 #definir MSGTYPE_SUBSCRIBE = 1
2 #definir MSGTYPE_UNSUBSCRIBE = 2
3 #definir MSGTYPE_PUBLISH = 3
4
5 typedef struct {
6     char topic[10];
7 } SubscribeAnnouncement;
8
9 typedef struct {
10    char topic[10];
11 } UnsubscribeAnnouncement;
12
13 typedef struct {
14    char topic[10];
15    uint8_t content[20];
16 } PublishContent;
17
18 typedef union {
19     SubscribeAnnouncement subscribeAnnouncement;
20     UnsubscribeAnnouncement unsubscribeAnnouncement;
21     PublishContent publish;
22 } PayloadUnion;
23
24 typedef struct {
25     uint8_t msgType;
26     PayloadUnion payload;
27 } Message;

```

- Crear un ejemplo de suscriptor (peso: 13). Sin utilizar la librería, se crearon dos códigos de ejemplo para los roles broker y suscriptor a partir de la definición de mensajes anterior. El broker utilizaba FreeRTOS para crear dos tareas que se ejecutan en paralelo: la primera generadora de mensajes de prueba en una cola de mensajes, y la otra para enviarlos a las placas que se hayan suscrito al broker (sin tener en cuenta los topics en esta iteración). Esta suscripción es controlada al añadir las direcciones de los suscriptores a un vector, y por cada nuevo suscriptor se comprueba si ya existía. El suscriptor, por su parte, crea mensajes de suscripción y se los envía al broker cada 2 segundos, además de controlar los mensajes de publicación recibidos. El pseudocódigo de estos ejemplos se

muestra en los Listados 4.2 y 4.3.

Listado 4.2: Pseudocódigo de la primera definición del broker

```

1 struct PayloadStruct: //Estructura de mensajes enviados
2     int number
3
4 vector<uint8_t> subscribers //Direcciones MAC de las placas suscriptoras
5 QueueHandle_t messagesQueue //Cola de mensajes para enviar
6
7 //Cada vez que se reciba un mensaje, se llama a esta función
8 Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
9     Message recvMessage = Copia de los datos recibidos
10    Si recvMessage.msgType es un mensaje de suscripción:
11        Si mac no estaba en subscribers: //Si es un suscriptor nuevo
12            Imprimir mensaje de aviso de suscripción con topic
13            Agregar mac a subscribers
14
15 //Tarea encargada de gestionar la cola de mensajes
16 Función DispatchMessagesTask():
17     Message message
18     Por siempre:
19         message = Recibir un mensaje de messagesQueue
20         Por cada subscriber en subscribers:
21             Registrar el subscriber como par
22             Enviar message a subscriber
23
24 //Tarea encargada de crear mensajes de prueba en la cola de mensajes
25 Función ProduceMessageTask():
26     Por siempre:
27         PayloadStruct payload
28         payload.number = Número aleatorio
29         Message sendMessage //Nuevo mensaje
30         sendMessage.msgType = Definir mensaje como publicación
31         sendMessage.payload.subscribeAnnouncement.topic = "mock" //Asignar
32             ↛ topic al payload SubscribeAnnouncement
33         sendMessage.payload.content = payload
34         Enviar sendMessage a la cola messagesQueue
35         Esperar 1 segundo

```

```

36 Función setup():
37     Inicializar ESP-NOW
38     Registrar la función OnDataRecv como callback al recibir un mensaje
39     Inicializar la cola messagesQueue
40     Crear tarea DispatchMessagesTask
41     Crear tarea ProduceMessageTask

```

Listado 4.3: Pseudocódigo de la primera definición del suscriptor

```

1 struct PayloadStruct: //Estructura de mensajes recibidos
2     int number
3
4     uint8_t destBoardAddr[] //Dirección MAC de la placa broker
5
6 //Cada vez que se reciba un mensaje del broker, se llama a esta función
7 Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
8     Message recvMessage = Copia de los datos recibidos
9     PayloadStruct messagePayload = recvMessage.payload.publish.content // 
    ↪ Contenido recibido
10    Imprimir messagePayload.number
11
12 Función subscribe(char* topic):
13     Message subMsg //Nuevo mensaje
14     subMsg.msgType = Definir mensaje como suscripción
15     subMsg.payload.subscribeAnnouncement.topic = topic //Asignar topic al
    ↪ payload SubscribeAnnouncement
16     Enviar subMsg al broker
17
18 Función setup():
19     Inicializar ESP-NOW
20     Registrar la función OnDataRecv como callback al recibir un mensaje
21     Registrar el broker como par
22
23 Función loop():
24     subscribe("topic1")
25     Esperar 2 segundos

```

/TODO: preguntar si mencionar los ejemplos que utilicé/

4.3. Iteración 2

Periodo: 26/02/2024 - 08/03/2024 (1,71 semanas). Peso total: 49.

El objetivo principal de este sprint fue el soporte agregado a la publicación de mensajes y al uso de los temas.

Tareas:

- Probar la subida de la librería a PlatformIO (peso: 5).
- Crear ejemplo publicador (peso: 8). En adición al ejemplo del suscriptor realizado en la [iteración 1](#), teniendo en común la desatención del topic y el desarrollo fuera de la librería, el publicador crea mensajes que publica en el broker. El Listado 4.4 muestra el pseudocódigo de este ejemplo. Asimismo, se añadió al broker la capacidad de gestionar estos mensajes, modificando la función OnRecv como se observa en el Listado 4.5.

Listado 4.4: Pseudocódigo de la primera definición del publicador

```

1 struct PayloadStruct: //Estructura de mensajes enviados
2     int number
3
4     uint8_t destBoardAddr[] //Dirección MAC de la placa broker
5
6     Función generateAndPublish(char* topic):
7         PayloadStruct payload
8         payload.number = Número aleatorio
9         Message sendMessage //Nuevo mensaje
10        sendMessage.msgType = Definir mensaje como publicación
11        sendMessage.payload.publish.topic = topic //Asignar topic al payload
12        ↳ PublishContent
13        sendMessage.payload.content = payload
14        Enviar sendMessage al broker
15
16     Función setup():
17         Inicializar ESP-NOW
18         Registrar el broker como par
19
20     Función loop():
21         generateAndPublish("topic1")
22         Esperar 5 segundos

```

Listado 4.5: Pseudocódigo de la modificación del broker para admitir publicaciones

```

1 //Cada vez que se reciba un mensaje, se llama a esta función
2 Función OnDataRecv(uint8_t *mac, uint8_t *incomingData):
3     Message recvMessage = Copia de los datos recibidos
4     Si recvMessage.msgType es un mensaje de suscripción:
5         Si mac no estaba en subscribers: //Si es un suscriptor nuevo
6             Agregar mac a subscribers
7     Sino si recvMessage.msgType es un mensaje de publicación:
8         PublishContent pubMsg = recvMessage.payload.publish
9         PayloadStruct pubContentPayload = pubMsg.content
10        Enviar recvMessage a la cola messagesQueue

```

- Mejorar la estructura de los mensajes (peso: 8). La estructura original de la [iteración 1](#) tenía un problema, y es que al utilizar un `typedef union` (utilizado para crear un mensaje genérico), todos los mensajes tenían el mismo tamaño, independientemente de su contenido, provocando un consumo de memoria innecesario. Para solucionarlo, se cambió la estructura de los mensajes, utilizando el primer byte para identificar el tipo de mensaje, y dependiendo del tipo se conoce la cantidad de bytes que habrá que leer del mensaje recibido. Este cambio se puede ver en el [Listado 4.6](#). A su vez, se realizaron ligeros cambios en el publicador, suscriptor y broker para gestionar los mensajes, añadiendo el mencionado método para identificarlos.

Listado 4.6: Segunda definición de los mensajes

```

1 typedef enum {
2     MSGTYPE_SUBSCRIBE = 0x00,
3     MSGTYPE_UNSUBSCRIBE,
4     MSGTYPE_PUBLISH
5 } MessageType;
6
7 typedef struct {
8     MessageType type;
9 } MessageBase;
10
11 typedef struct : public MessageBase {
12     char topic[10];
13 } SubscribeAnnouncement;
14
15 typedef struct : public MessageBase {
16     char topic[10];
17 } UnsubscribeAnnouncement;
18

```

```

19 | typedef struct : public MessageBase {
20 |     char topic[10];
21 |     uint8_t content[20];
22 | } PublishContent;

```

- Hacer ejemplos con tarjeta SD (peso: 8). Primer acercamiento a este tipo de memoria externa para aprender a trabajar con ficheros y carpetas.
- Implementar el uso de más topics (peso: 20). Para añadir esta posibilidad, se decidió crear una clase **BrokerTopic**, siendo cada una un topic con un listado de suscriptores y una cola de mensajes, y las funciones apropiadas para gestionar la lista y la cola. Se crea una clase **BrokerTopic** en el broker por cada mensaje (de publicación o suscripción) recibido con un nuevo topic. El Listado 4.7 muestra el pseudocódigo de la primera versión de esta clase, y el 4.8 muestra la implementación de **BrokerTopic**.

Listado 4.7: Pseudocódigo de la primera definición de la clase **BrokerTopic**

```

1 Clase BrokerTopic:
2 Privado:
3     char topic[10]
4     vector<uint8_t[6]> subscribers //Direcciones MAC de los suscriptores
5     QueueHandle_t messagesQueue //Cola que almacena mensajes para enviar
6
7 Público:
8     Función constructora
9     Función getTopic()
10    Función getSubscribersAmount() //Cuenta la cantidad de suscriptores
11    Función getSubscribers()
12    Función subscribe(uint8_t mac):
13        Si no isSubscribed(mac):
14            Registrar mac como par
15            Agregar mac a subscribers
16        Función isSubscribed(uint8_t mac): //Comprueba si la MAC ya estaba
17            ↵ suscrita
18            Recorrer subscribers
19            Si encuentra mac en subscribers, retornar true //Suscrita
20            Sino, retornar false //No suscrita
21        Función sendToQueue(PublishContent pubContent) //Enviar mensaje a la cola
22            ↵ messagesQueue
23        Función dispatchMessages(): //Enviar todos los mensajes de la cola
24            ↵ messagesQueue a los suscriptores

```

```
22     PublishContent message = Recibir un mensaje de messagesQueue  
23     Por cada subscriber en subscribers:  
24         Registrar subscriber como par  
25         Enviar message a subscriber
```

Listado 4.8: Pseudocódigo de la segunda definición del broker

```

1 struct PayloadStruct: //Estructura de mensajes enviados
2     int number
3
4 vector<BrokerTopic> topicsVector //Contiene los topics
5
6 //Cada vez que se reciba un mensaje, se llama a esta función
7 Función OnDataRecv(uint8_t mac, uint8_t incomingData):
8     Message msgType = Primer byte de incomingData
9     Según msgType:
10         Si es un mensaje de suscripción:
11             SubscribeAnnouncement subAnnounce = Copia de los datos recibidos
12             Por cada topicObject de topicsVector con topicObject.topic igual
13                 ↪ al topic de subAnnounce:
14                     Si mac no está suscrita al topicObject:
15                         topicObject.subscribe(mac)
16                     Si no hubo ningún topicObject con el mismo topic:
17                         BrokerTopic newTopic = Nuevo topic creado con el mismo topic
18                             ↪ que el de subAnnounce
19                             newTopic.subscribe(mac)
20                         Agregar newTopic a topicsVector
21
22         Si es un mensaje de publicación:
23             PublishContent pubContent = Copia de los datos recibidos
24             Por cada topicObject de topicsVector con topicObject.topic igual
25                 ↪ al topic de pubContent:
26                     topicObject.sendToQueue(pubContent)
27             Si no hubo ningún topicObject con el mismo topic:
28                 BrokerTopic newTopic = Nuevo topic creado con el mismo topic
29                     ↪ que el de subAnnounce
30                     newTopic.sendToQueue(pubContent)
31                 Agregar newTopic a topicsVector

```

```

29 //Tarea encargada de enviar cada mensaje almacenado en los topics
30 Función DispatchMessagesTask():
31     Por siempre, cada 1 segundo:
32         Por cada topicObject en topicsVector:
33             topicObject.dispatchMessage()
34
35 //Tarea encargada de crear mensajes de prueba
36 Función ProduceMessageTask():
37     Por siempre, cada 1 segundo:
38         PayloadStruct payload
39         payload.number = Número aleatorio
40         Message sendMessage //Nuevo mensaje
41         sendMessage.msgType = Definir mensaje como publicación
42         sendMessage.payload.subscribeAnnouncement.topic = "mock" //Asignar
43             ↛ topic al payload SubscribeAnnouncement
44         sendMessage.payload.content = payload
45         Por cada topicObject de topicsVector con topicObject.topic igual al
46             ↛ topic de sendMessage:
47                 topicObject.sendToQueue(sendMessage)
48         Si no hubo ningún topicObject con el mismo topic:
49             BrokerTopic newTopic = Nuevo topic creado con el mismo topic que
50                 ↛ el de sendMessage
51                 topicObject.sendToQueue(sendMessage)
52                 Agregar newTopic a topicsVector
53
54 Función setup():
55     Inicializar ESP-NOW
56     Registrar la función OnDataRecv como callback al recibir un mensaje
57     Inicializar la cola messagesQueue
58     Crear tarea DispatchMessagesTask
59     Crear tarea ProduceMessageTask

```

4.4. Iteración 3

Periodo: 09/03/2024 - 15/03/2024 (1 semana). Peso total: 57.

El objetivo principal de este sprint fue la implementación de paralelismo al procesar los tipos de mensajes y la compatibilidad con caracteres comodín en los topics.

Tareas:

- Mover funciones de publicar/suscribir de los ejemplos al código de la librería (peso: 8). Las funciones desarrolladas como ejemplos en las iteraciones 1 y 2 fueron movidas tras considerarse versiones funcionales adecuadas, quedando declaradas como en el Listado 4.9. Los códigos anteriores fueron modificados para hacer llamadas a la librería.

Listado 4.9: Pseudocódigo de la primera versión de los parámetros de las funciones para publicar y suscribir definidos en la librería

```
1 Función publish(uint8_t mac, char topic, void payload)
2 Función subscribe(uint8_t mac, char topic)
```

- Modificar callback del broker para crear tareas (peso: 20). Para aprovechar la capacidad del paralelismo, se implementaron tareas capaces de gestionar los mensajes de suscripción y publicación, que recorren los objetos BrokerTopic para determinar en cuál es más adecuado publicar o suscribir. En este primer acercamiento, cada vez que el broker recibe uno de estos mensajes, se crea una tarea. La modificación se llevó a cabo como se muestra en el Listado 4.10. Junto a esto, se eliminó la tarea DispatchMessagesTask del broker y se creó la función publish(PublishContent ↛ pubContent) en BrokerTopic, que envía directamente el mensaje de publicación a todos los suscriptores del objeto. Esto se debe a la nueva solución, en la que la carga del procesamiento de los suscriptores se realiza en cada tarea PublishTask creada.

Listado 4.10: Pseudocódigo de la modificación del broker para crear tareas según el mensaje recibido

```
1 struct SubscribeTaskParams: //Estructura de parámetros para suscripción
2     SubscribeAnnouncement subAnnounce
3     uint8_t mac
4
5 struct PublishTaskParams: //Estructura de parámetros para publicación
6     PublishContent pubContent
7     uint8_t mac
8
9 Función SubscribeTask(void parameter):
10     SubscribeTaskParams params = parameter //Extraer los parámetros
11     SubscribeAnnouncement subAnnounce = params.subAnnounce
12     uint8_t mac = params.mac
13     Por cada topicObject de topicsVector con topicObject.topic igual al topic
14         ↛ de subAnnounce:
15             Si mac no está suscrita al topicObject:
16                 topicObject.subscribe(mac)
17             Si no hubo ningún topicObject con el mismo topic:
```

```

17     BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el
         ↪ de subAnnounce
18     newTopic.subscribe(mac)
19     Agregar newTopic a topicsVector
20
21 Función PublishTask(void parameter):
22     PublishTaskParams params = parameter //Extraer los parámetros
23     PublishContent pubContent = params.pubContent
24     uint8_t mac = params.mac
25     Por cada topicObject de topicsVector con topicObject.topic igual a
         ↪ pubContent.topic:
26         topicObject.sendToQueue(pubContent)
27     Si no hubo ningún topicObject con el mismo topic:
28         BrokerTopic newTopic = Nuevo topic creado con el mismo topic que el
         ↪ de subAnnounce
29         topicObject.sendToQueue(pubContent)
30         Agregar newTopic a topicsVector
31
32
33 //Cada vez que se reciba un mensaje, se llama a esta función
34 Función OnDataRecv(uint8_t mac, uint8_t incomingData):
35     Message msgType = Primer byte de incomingData
36     Según msgType:
37         Si es un mensaje de suscripción:
38             SubscribeTaskParams subTaskParams
39             subTaskParams.subAnnounce = Copia de los datos recibidos
40             subTaskParams.mac = mac
41             Crear tarea SubscribeTask con parámetro subTaskParams
42         Si es un mensaje de publicación:
43             PublishTaskParams pubTaskParams
44             pubTaskParams.pubContent = Copia de los datos recibidos
45             pubTaskParams.mac = mac
46             Crear tarea PublishTask con parámetro pubTaskParams

```

- Añadir comprobador, agregador y eliminador de pares (peso: 13). ESP-NOW tiene un límite de pares que se pueden agregar, por lo que es necesario implementar un comprobador de pares para no agregar pares repetidos. Con este objetivo, se creó la función addPeer(uint8_t *mac) en BrokerTopic, que es llamada antes de enviar un mensaje a los suscriptores. Además, se creó la función removePeer(uint8_t *mac) para eliminar un par registrado en la futura implementa-

ción de la desuscripción.

- Hacer ejemplos con formatos CSV y **JSON** en tarjeta SD (peso: 8). Crear ejemplos para serializar y deserializar objetos en ficheros contenidos en una tarjeta SD con el fin de aprender las capacidades de las librerías que lo permiten para implementar la persistencia de topics en futuras iteraciones.
- Añadir soporte a jerarquía de topics y caracteres comodín (peso: 8). La librería que se desarrolla en este proyecto se basa en **MQTT**, y está influenciada por sus capacidades, como la jerarquía y los caracteres comodín o wildcards + y #. Esto implica tener en cuenta el topic que inserta el usuario al publicar o suscribirse mediante dos funciones creadas en la librería. Definidas en el Listado 4.11, `pubTopicCheck` y `subTopicCheck` ambas arreglan el mal uso de los caracteres /, comprueban que el tamaño del topic no sea mayor del permitido y recorren todos los caracteres comprobando que sean ASCII y estén bien utilizados, como por ejemplo, la prohibición del publicador para usar wildcards. Son funciones llamadas antes del envío de los mensajes, es decir, desde los roles publicador y suscriptor/descriptador.

En cuanto a la jerarquía de suscripción, se refiere a la suscripción a un tema como `cocina/+`, que significa la suscripción a publicaciones con temas como `cocina/nevera` o `cocina/grifo`. Esto se ha resuelto implementando una nueva estrategia para crear BrokerTopics. Primero, se sigue creando un `BrokerTopic` por cada suscripción, incluso si el topic tiene un wildcard. Para cada publicación, en cambio, se elimina esta capacidad, evitando un consumo de memoria por publicaciones sin receptores, que en este caso se descartarán. Una publicación a un topic pasa por la comprobación del topic con los `BrokerTopics` ya creados, y si es compatible o “publicable”, envía mensajes a los suscriptores. Por ejemplo, una publicación con topic `cocina/grifo` es “publicable” para los suscriptores de `cocina/+`, pero, por el contrario, una publicación `banio/grifo` no lo es, y si no existen `BrokerTopics` compatibles se descarta el mensaje. La implementación se ha llevado a cabo agregando la función para comprobar si es “publicable” junto a un atributo que facilita dicha comprobación, como se muestra en el Listado 4.12.

Listado 4.11: Pseudocódigo de las funciones añadidas para comprobar los topics

```

1 #definir MAXTOPICLENGTH = 10 //Longitud máxima del topic
2
3 Función fixTopicAndCheckLength(char topic):
4   Si topic es null:
5     Retornar error
6   Eliminar caracteres / añadidos al principio o al final de topic
7   Si topic es demasiado grande:
8     Retornar error
9     Retornar éxito
10
11 Función pubTopicCheck(char topic):
12   Si fixTopicAndCheckLength(topic) retorna error:

```

```

13   Retornar error
14   Si existe algún carácter '+', '#' o no ASCII en topic:
15     Retornar error
16   Retornar éxito
17
18 Función subTopicCheck(char topic):
19   Si fixTopicAndCheckLength(topic) retorna error:
20     Retornar error
21   Por cada carácter en topic:
22     Si no es ASCII:
23       Retornar error
24     Si es '+' o es '#' en una posición incorrecta:
25       Retornar error
26   Retornar éxito

```

Listado 4.12: Pseudocódigo de la comprobación de compatibilidad en BrokerTopic

```

1 Clase BrokerTopic:
2 Privado:
3   bool hasWildcards //Si el topic contiene wildcards
4   char topic[MAXTOPICLENGTH]
5   ...
6
7 Público:
8   Función constructora
9   ...
10  Función publish(PublishContent pubContent)
11  Función isPublishable(char publishTopic):
12    Si publishTopic es igual a topic:
13      Retornar true
14    Si hasWildcards es true y se recorrieron los wildcards de topic
15      ↢ probando su compatibilidad con publishTopic:
16      Retornar true
17    Sino:
18      Retornar false

```

4.5. Iteración 4

Periodo: 16/03/2024 - 21/03/2024 (1 semana). Peso total: 39.

El objetivo principal de este sprint fue añadir soporte para la desuscripción y arreglar varios errores y vulnerabilidades.

Tareas:

- Probar publicador/suscriptor MQTT con Python (peso: 5). Parecido a la **iteración 0**, trata de desarrollar un código en Python para explorar las capacidades de **MQTT**. En concreto, en esta iteración se comprobó que si un usuario se suscribe a dos temas como **cocina/nevera** y **cocina/+**, y se publica un mensaje en **cocina/nevera**, cuántas veces recibe el mensaje el suscriptor. El resultado fue 1.
- Realizar pruebas de estrés a la solución actual (peso: 5). La solución desarrollada en la **iteración 3** presentó una vulnerabilidad revisada al final del sprint anterior. Anteriormente, se creaban tareas cada vez que se recibía un mensaje, lo que significaba que un nodo malicioso que publicara o se suscribiera constantemente al broker podría realizar ataques de denegación de servicio, provocando sobrecargas en la memoria o tareas mal ejecutadas. La tarea consistió en desarrollar un publicador malicioso que enviara mensajes constantemente a un broker desplegado y modificado para que las tareas demoren más tiempo, generando tres tipos de errores:
 - El broker no podía crear tareas, posiblemente debido a que la memoria de la placa estaba saturada con demasiadas tareas encoladas.
 - Error de “núcleo entró en pánico” y “carga prohibida”.
 - Los mensajes no se trataban correctamente, el contenido era interpretado con caracteres erróneos.
- Arreglar vulnerabilidad DDOS (peso: 8). A partir de la prueba anterior, se vio necesario idear una nueva solución para gestionar los mensajes. En esta, el broker ejecuta constantemente una tarea por cada tipo de mensaje: una para publicar, otra para suscribir y otra para desuscribir. Estas tareas leen constantemente de una cola de mensajes, en las cuales los mensajes se insertan tras su recepción, como se muestra en el Listado 4.13.

Listado 4.13: Pseudocódigo de la modificación del broker para insertar mensajes recibidos en colas

```

1 //Cantidad de tareas ejecutadas por tipo de mensaje
2 #definir SUBSCRIBETASKS = 1
3 #definir UNSUBSCRIBETASKS = 1
4 #definir PUBLISHTASKS = 1
5
6 struct SubscribeTaskParams: //Estructura de parámetros para suscripción
7   SubscribeAnnouncement subAnnounce
8   uint8_t mac
9
10 struct PublishTaskParams: //Estructura de parámetros para publicación
11   PublishContent pubContent

```

```
12     uint8_t mac
13
14 vector<BrokerTopic> topicsVector //Contiene los topics
15
16 //Colas de mensajes
17 QueueHandle_t subMsgQueue;
18 QueueHandle_t pubMsgQueue;
19
20 //Tarea encargada de procesar mensajes de suscripción
21 Función SubscribeTask(void parameter):
22     SubscribeTaskParams params
23     Por siempre, cada 1 segundo:
24         Recibir de la cola subMsgQueue y copiar en params
25         //Extraer los parámetros
26         SubscribeAnnouncement subAnnounce = params.subAnnounce
27         uint8_t mac = params.mac
28         topicObject = Primer elemento de topicsVector con un topic igual al
29             ↪ de subAnnounce
30         Si topicObject existe:
31             Si mac no está suscrita al topicObject:
32                 topicObject.subscribe(mac)
33             Sino:
34                 BrokerTopic newTopic = Nuevo topic creado con el mismo topic que
35                     ↪ el de subAnnounce
36                 newTopic.subscribe(mac)
37                 Agregar newTopic a topicsVector
38
39 //Tarea encargada de procesar mensajes de publicación
40 Función PublishTask(void parameter):
41     PublishTaskParams params = parameter
42     Por siempre, cada 1 segundo:
43         Recibir de la cola pubMsgQueue y copiar en params
44         //Extraer los parámetros
45         PublishContent pubContent = params.pubContent
46         uint8_t mac = params.mac
47         Por cada topicObject en topicsVector:
48             Si topicObject.isPublishable(pubContent.topic): //Compatibilidad
49                 ↪ entre topic de la publicación y el del BrokerTopic
```

```

47     topicObject.publish(pubContent)
48
49 //Cada vez que se reciba un mensaje, se llama a esta función
50 Función OnDataRecv(uint8_t mac, uint8_t incomingData):
51     Message msgType = Primer byte de incomingData
52     Según msgType:
53         Si es un mensaje de suscripción:
54             SubscribeTaskParams subTaskParams
55                 subTaskParams.subAnnounce = Copia de los datos recibidos
56                 subTaskParams.mac = mac
57                 Enviar subTaskParams a la cola subMsgQueue
58         Si es un mensaje de publicación:
59             PublishTaskParams pubTaskParams
60                 pubTaskParams.pubContent = Copia de los datos recibidos
61                 pubTaskParams.mac = mac
62                 Enviar pubTaskParams a la cola pubMsgQueue
63
64 Función setup():
65     Inicializar ESP-NOW
66     Registrar la función OnDataRecv como callback al recibir un mensaje
67     Inicializar la cola subMsgQueue
68     Inicializar la cola pubMsgQueue
69     Crear tareas SubscribeTask según SUBSCRIBETASKS
70     Crear tareas PublishTask según PUBLISHTASKS

```

- Implementar la desuscripción (peso: 13). Crear las funciones de desuscripción en la librería y en BrokerTopic, y la tarea y cola necesarias en el broker para gestionar este tipo de mensajes recibidos. La desuscripción elimina la dirección **MAC** de la lista de suscriptores de un topic, en este caso se realiza aprovechando las funciones integradas en C++ para buscar en un vector y eliminar el elemento, junto con la función `removePeer` creada anteriormente. En adición a eliminar de la lista de suscriptores, para evitar malgasto de memoria, si al desuscribir el BrokerTopic se queda sin suscriptores, se elimina. La implementación se muestra en los Listados 4.14, 4.15 y 4.16.

Listado 4.14: Pseudocódigo de la función de desuscribir definida en la librería

```

1 Función unsubscribe(uint8_t *mac, char *topic):
2     Inicializar ESP-NOW
3     Registrar el broker como par
4     Si subTopicCheck(topic) retorna error:
5         Retornar error

```

```

6
7 UnsubscribeAnnouncement unsubMsg //Nuevo mensaje
8 unsubMsg.type = Definir mensaje como desuscripción
9 unsubMsg.topic = topic
10 Enviar unsubMsg al broker

```

Listado 4.15: Pseudocódigo de la adición de la desuscripción al broker

```

1 //Cantidad de tareas ejecutadas por tipo de mensaje
2 #definir SUBSCRIBETASKS = 1
3 #definir UNSUBSCRIBETASKS = 1
4 #definir PUBLISHTASKS = 1
5
6 struct UnsubscribeTaskParams: //Estructura de parámetros para desuscripción
7     UnsubscribeAnnouncement *unsubAnnounce
8     uint8_t *mac
9
10 vector<BrokerTopic> topicsVector //Contiene los topics
11
12 //Colas de mensajes
13 QueueHandle_t unsubMsgQueue;
14
15 //Tarea encargada de procesar mensajes de desuscripción
16 Función UnsubscribeTask(void parameter):
17     UnsubscribeTaskParams params
18     Por siempre, cada 1 segundo:
19         Recibir de la cola unsubMsgQueue y copiar en params
20         //Extraer los parámetros
21         UnsubscribeAnnouncement unsubAnnounce = params.subAnnounce
22         uint8_t mac = params.mac
23         it = Primer elemento de topicsVector con un topic igual al de
24             ↛ unsubAnnounce
25         Si it existe:
26             Si mac está suscrita al it:
27                 it.unsubscribe(mac)
28             Si it tiene 0 suscriptores:
29                 Eliminar it de topicsVector
30 //Cada vez que se reciba un mensaje, se llama a esta función

```

```

31 Función OnDataRecv(uint8_t mac, uint8_t incomingData):
32     Message msgType = Primer byte de incomingData
33     Según msgType:
34         ...
35         Si es un mensaje de desuscripción:
36             UnsubscribeTaskParams unsubTaskParams
37                 unsubTaskParams.unsubAnnounce = Copia de los datos recibidos
38                 unsubTaskParams.mac = mac
39                 Enviar unsubTaskParams a la cola unsubMsgQueue
40
41 Función setup():
42     Inicializar ESP-NOW
43     Registrar la función OnDataRecv como callback al recibir un mensaje
44     Inicializar la cola subMsgQueue
45     Inicializar la cola unsubMsgQueue
46     Inicializar la cola pubMsgQueue
47     Crear tareas SubscribeTask según SUBSCRIBETASKS
48     Crear tareas UnsubscribeTask según UNSUBSCRIBETASKS
49     Crear tareas PublishTask según PUBLISHTASKS

```

Listado 4.16: Pseudocódigo de la adición de la desuscripción a BrokerTopic

```

1 Clase BrokerTopic:
2 Privado:
3     ...
4     char topic[MAXTOPICLENGTH]
5     vector<uint8_t[6]> subscribers //Direcciones MAC de los suscriptores
6
7 Público:
8     Función constructora
9     ...
10    Función removePeer(uint8_t mac)
11    Función isSubscribed(uint8_t mac)
12    Función unsubscribe(uint8_t mac):
13        Recorrer subscribers con un valor igual a mac
14        Si lo encuentra:
15            Borrar de subscribers
16            removePeer(mac)
17            Retornar true

```

18

```
    Retornar false //Si no se encontró
```

- Arreglar el envío de mensajes duplicados al suscribirse múltiples veces al mismo topic (peso: 8). Añadir un comprobador de los destinatarios a quienes ya se les envió el mensaje. Esto se realizó creando, al recibir un mensaje de publicación, un vector `alreadySentMacs` en el broker, el cual es compartido por todos los `BrokerTopics` compatibles con el topic publicado y contiene todos los destinatarios a los que ya se le envió el mensaje. Un objeto `BrokerTopic` de los compatibles primero comprueba uno a uno si sus suscriptores ya están en esta lista y, si no es el caso, se le envía el mensaje y se le añade al vector, para que otros objetos `BrokerTopic` compatibles no le envíen el mensaje.

4.6. Iteración 5

Periodo: 16/03/2024 - 21/03/2024 (1 semana). Peso total: 57.

El objetivo principal de este sprint fue la mejora de la calidad del código, la corrección de errores y el primer paso para redactar la memoria.

Tareas:

- Encontrar (peso: 8) y arreglar (peso: 20) la recepción de mensajes corruptos o duplicados. Tras la revisión del sprint anterior, se encontró el fallo de que los mensajes de suscripción y desuscripción se duplicaban en las colas. El primer paso para resolverlo fue aislar el error, creando un broker dedicado únicamente a atender peticiones de suscripción. Luego de realizar pruebas modificando este código, se descubrió que el fallo se encontraba en la mala gestión de la memoria con las variables que almacenan los parámetros de los mensajes y se insertan en las colas, por lo que se solucionó haciendo un mejor uso de los punteros y de las liberaciones de memoria.
- Clonar la plantilla de la memoria del **TFG** (peso: 5).
- Renombrar variables, añadir comentarios y arreglar el formato del código (peso: 8). Junto con eliminar código sin usar, comentarios innecesarios e implementar `EditorConfig`¹ y cumplir con los criterios de calidad, con el fin de mejorar la calidad del código.
- Crear ideas para nombrar la herramienta (peso: 8). En gran parte del desarrollo, la librería no ha tenido un nombre definido, siendo este paso esencial para poder publicarla y darle una identidad.
- Probar y familiarizarse con la plantilla de la memoria (peso: 8). Aprender a utilizar la plantilla para empezar a redactar en posteriores iteraciones.

4.7. Iteración 6

Periodo: 06/04/2024 - 12/04/2024 (1 semana). Peso total: 36.

El objetivo principal de este sprint fue la integración del broker en el código de la librería.

¹<https://editorconfig.org/>

Tareas:

- Planificar el contenido de la memoria (peso: 5). Diseñar una estructura inicial con los distintos temas que se detallan en la memoria.
- Crear ejemplos con las funciones integradas de log y la librería Elog (peso: 13). Dos ejemplos sencillos probando el registro o log en diferentes niveles y en la tarjeta SD, con el fin de comparar las capacidades de cada uno y decidir cuál es considerado más adecuado para las necesidades del alumno. Se descubrió poca libertad de uso con las funciones integradas, ya que requieren una configuración del proyecto que vaya a utilizarlas, traduciéndose negativamente en trabajo adicional para el usuario final.
- Mover el código fuente del broker a la librería (peso: 13). Al igual que lo realizado previamente con las funciones de publicar, suscribir y desuscribir, tras haberse probado en profundidad el funcionamiento del código del broker, se movió al código final de la librería, resultando en la función `setupBroker()`.
- Probar librería refactorizada (peso: 5). El proyecto pasó por una refactorización, incluyendo una jerarquía de ficheros, variables y definiciones organizadas, y la modificación y creación de cabeceras apropiadas para los usuarios finales que únicamente requieran de las funciones de cliente, solo las del broker, o ambas simultáneamente. La prueba implementó la librería y confirmó su correcto funcionamiento.

4.8. Iteración 7

Periodo: 13/04/2024 - 19/04/2024 (1 semana). Peso total: 42.

El objetivo principal de este sprint fue la implementación de logs en la librería.

Tareas:

- Crear ejemplos con las librerías ArduinoLog² y DebugLog³ (peso: 13). Siguiendo con la iteración anterior, se crearon ejemplos para probar las librerías.
- Elegir un sistema de registro/log (peso: 8). Comparar las funciones integradas de log y las librerías Elog, ArduinoLog, DebugLog, en base a requisitos establecidos por el alumno:
 - Simple de implementar y modificar o personalizar.
 - Diferentes niveles de registro.
 - Mensajes mostrados junto a marcas de tiempo y el nivel de log.
 - Permite guardar registros en la tarjeta SD.

La comparación resultó en la decisión de utilizar Elog para implementar los registros.

- Crear la primera versión de la estructura de la memoria del TFG (peso: 8). Idear una estructura adecuada para debatirla con los tutores y empezar a redactar en siguientes iteraciones.

²<https://registry.platformio.org/libraries/thijse/ArduinoLog>

³<https://registry.platformio.org/libraries/hideakitai/DebugLog>

- Implementar mensajes de registro (peso: 13). La implementación de Elog se basó en ofrecer la facilidad al usuario final de decidir qué tipo de mensajes mostrar en pantalla (en base al nivel de log) o si prefiere guardarlos en una tarjeta SD. Para ello, se crearon los ficheros `Logger.h` y `Logger.c`, que definen las tres funciones mostradas en el Listado 4.17. La idea es que el usuario primero cree un objeto `Elog` configurado a su gusto y se lo pase como parámetro a las funciones principales de la librería. Además, se modificó el broker, cambiando la función principal a `setupBroker(Elog ↳ logger)` para tomar el objeto configurado y utilizarlo, y añadiendo logs en el código para imprimir mensajes en puntos clave.

Listado 4.17: Pseudocódigo de las funciones de Logger

```

1 Enumeración LoggerClass:
2     BROKER, PUBLISHER, SUBSCRIBER, UNKNOWN
3
4 Función initializeSerialLogger(LoggerClass className, LogLevel level = INFO)
    ↳ //Muestra logs por terminal
5     Retorna objeto Elog configurado
6 Función initializeSDLogger(LoggerClass className, int cs, int sck, int miso,
    ↳ int mosi, LogLevel level = INFO) //Guarda logs en la tarjeta SD
7     Retorna objeto Elog configurado a los pines de la tarjeta SD
8 Función disableLogger(): //No muestra ni guarda logs
9     Retorna objeto Elog configurado

```

4.9. Iteración 8

Periodo: 20/04/2024 - 26/04/2024 (1 semana). Peso total: 41.

El objetivo principal de este sprint fue crear y probar una estructura de ficheros para persistir los temas y suscriptores del broker.

Tareas:

- Implementar logs en las funciones de publicador/suscriptor (peso: 13). Al igual que en la función del broker, añadir mensajes de log clave e incluir por parámetro el objeto `Elog`, quedando las funciones `publish(uint8_t mac, char topic, void payload, Elog logger = ↳ disableLogger())`, `subscribe(uint8_t mac, char topic, Elog logger = ↳ disableLogger())` y `unsubscribe(uint8_t mac, char topic, Elog logger = ↳ disableLogger())`.
- Idear una estructura de ficheros para persistir los topics y sus suscriptores (peso: 8). Se ideó un sistema de almacenamiento para los objetos `BrokerTopic`, guardándolos dentro del directorio / ↳ LoboMQ/topics en ficheros JSON debido a la sencillez de manipulación. Su contenido, como se observa en el Listado 4.18, tiene dos campos: el topic y la lista de suscriptores en un formato de

dirección MAC legible para el usuario, útil en caso de necesitar editarla manualmente. El nombre de los ficheros se basa en el nombre del topic, aunque de manera limitada, ya que existen caracteres que el sistema operativo no permite utilizar para nombrar ficheros. Debido a que los nombres de los topics son siempre caracteres ASCII, y para no sustituir los caracteres inválidos por letras y símbolos que el usuario pueda usar, se ha optado por utilizar el juego de caracteres ASCII extendido, compatibles con el nombrado de ficheros. Los caracteres inválidos son sustituidos por caracteres con cierta semejanza según decidió el alumno, mostrados en la Tabla 4.1. Esto resulta en, por ejemplo, que la lista de suscriptores del topic `cocina/+` se guarde en el fichero `cocina√+.json`.

Listado 4.18: Ejemplo de fichero de persistencia

```

1 {
2   "topic": "cocina/+",
3   "subscribers": [
4     "C0:49:EF:CB:99:10",
5     "29:BB:91:73:38:44"
6   ]
7 }
```

Tabla 4.1: Lista de caracteres inválidos y su reemplazo para el nombrado de ficheros de persistencia

Carácter ASCII inválido	Reemplazo extended ASCII
<	«
>	»
:	÷
”	“
/	√
\	ì
?	¿
*	º

- Crear y probar la lectura y escritura de ficheros que almacenan topics y suscriptores (peso: 20). A partir de lo anterior, crear un ejemplo capaz de crear y leer ficheros en la tarjeta SD, y realizar las conversiones adecuadas para las direcciones MAC y el nombre del fichero.

4.10. Iteración 9

Periodo: 27/04/2024 - 10/05/2024 (2 semanas). Peso total: 65.

El objetivo principal de este sprint fue la adición de la persistencia y de códigos de error a la librería.

Tareas:

- Añadir persistencia al broker para los temas y suscriptores (peso: 20). La persistencia permite almacenar en la tarjeta SD los temas y suscriptores que se están utilizando al momento de la ejecución del broker, permitiendo que el sistema se pueda reiniciar y continuar funcionando sin perder información. A partir de la estructura y las pruebas de la [iteración 8](#), se ha implementado en el código del broker, comenzando por ofrecer la opción al usuario mediante los parámetros de `setupBroker(Elog logger = initializeSerialLogger(BROKER), ↵ bool persistence = false, int csSdPin = -1)`. Si el usuario activa la persistencia, se almacena el pin CS de la tarjeta SD y se inicializa un semáforo mutex, que controla los accesos a la tarjeta. El broker lee los ficheros almacenados al iniciarse y escribe el fichero correspondiente al topic cada vez que ocurre una suscripción o desuscripción, gracias a las funciones probadas en la iteración, incluidas en los ficheros `BrokerSDUtils` y recopiladas en el [Listado 4.19](#). Para evitar la constante conversión del topic al nombre de fichero compatible, se ha creado un atributo `filename` en la clase `BrokerTopic`.
- Redactar Estado del arte (peso: 40). Primera redacción de la memoria.
- Implementar códigos de error en las funciones (peso: 5). Con la intención de ofrecer al usuario informado acerca del resultado de la ejecución de las funciones, se han modificado para que retornen si esta fue correcta u ocurrió algún error, mediante los siguientes códigos definidos:
 - `LMQ_ERR_SUCCESS`: ejecución correcta, sin errores.
 - `LMQ_ERR_INVAL_TOPIC`: el tema utilizado para publicar, suscribirse o desuscribirse no cumple con los requisitos de formato.
 - `LMQ_ERR_BAD_ESP_CONFIG`: no se pudo inicializar ESP-NOW.
 - `LMQ_ERR_ESP_SEND_FAIL`: no se pudo enviar el mensaje de publicación, suscripción o desuscripción.
 - `LMQ_ERR_XQUEUECREATE_FAIL`: el broker no pudo crear las colas necesarias.
 - `LMQ_ERR_XTASKCREATE_FAIL`: el broker no pudo crear las tareas necesarias.

[Listado 4.19: Pseudocódigo de las funciones de BrokerSDUtils](#)

```

1 #definir FILE_PATH = "/LoboMQ/topics" //Directorio de ficheros
2 #definir FILE_FORMAT = ".json" //Formato de ficheros
3
4 Función replaceChars(char str): //Utilizado para los nombres de ficheros
5   Retorna un string con los caracteres inválidos reemplazados
6 Función initializeSDCard(int csPin, Elog logger, SemaphoreHandle_t mutex,
7   ↵ TickType_t delay):
8   Inicializar conexión con la tarjeta SD
9   Si falla, retornar false

```

```

9   Tomar el mutex con una espera máxima indicada en delay
10    Si pasa el tiempo, retornar false
11   Crear las estructura de carpetas FILE_PATH
12    Si falla, devolver mutex y retornar false
13   Devolver el mutex
14   Retornar true
15
16 Función restoreBTs(vector<BrokerTopic> topicsVector, Elog logger,
17   ↘ SemaphoreHandle_t mutex, TickType_t delay):
18   Tomar el mutex con una espera máxima indicada en delay
19   Si falla, retornar
20   Por cada file en los ficheros del directorio FILE_PATH:
21     Si file no es una carpeta y termina en FILE_FORMAT:
22       JsonDocument doc
23       Deserializar file en doc
24       Si falla, continuar bucle
25       topic = doc["topic"]
26       JSONArray subscribersArray = doc["subscribers"]
27       Si subscribersArray tiene tamaño igual a 0: //Si no tiene
28         ↗ suscriptores
29       Continuar bucle
30
31       BrokerTopic newTopic = Creado con topic asignado
32       Suscribir cada suscriptor de subscribersArray a newTopic
33       newTopic.setFileName(file) //Guardar el nombre del fichero
34   Devolver el mutex
35
36 Función writeBTToFile(BrokerTopic brokerTopic, Elog logger, SemaphoreHandle_t
37   ↗ mutex, TickType_t delay):
38   JsonDocument doc
39   doc["topic"] = brokerTopic.topic
40   doc["subscribers"] = brokerTopic.subscribers
41   fullFilepath = FILE_PATH + "/" + brokerTopic.filename + FILE_FORMAT
42   Tomar el mutex con una espera máxima indicada en delay
43   Si pasa el tiempo, retornar
44   file = Abrir fichero fullFilepath en modo escritura
45   Si file existe:
46     Serializar doc en file

```

```

44 Devolver el mutex
45
46 Función deleteBTFile(char filename, Elog logger, SemaphoreHandle_t mutex,
47   ↳ TickType_t delay):
48     Tomar el mutex con una espera máxima indicada en delay
49     Si pasa el tiempo, retornar
50     fullfilepath = FILE_PATH + "/" + filename + FILE_FORMAT
51     file = Abrir fichero fullfilepath en modo escritura
52     Si file existe:
53       Eliminar fichero
      Devolver el mutex

```

4.11. Iteración 10

Periodo: 11/05/2024 - 23/05/2024 (2 semanas). Peso total: 106.

El objetivo principal de este sprint fue añadir una capa de seguridad adicional a través de un whitelist.

Tareas:

- Crear clase para almacenar una lista de direcciones MAC (peso: 20). La creación de la nueva clase `MACAddrList` facilita al usuario la creación de listas de direcciones MAC. Esta clase hereda de la clase `vector` y añade funciones para verificar si una dirección MAC ya está incluida, para añadir una o varias direcciones a la vez (pasándolas como un `string` o un `uint8_t [6]`), para eliminarlas y para limpiar la lista.
- Añadir autenticación mediante una whitelist de MACs (peso: 20). A partir de la clase `MACAddrList` creada recientemente, el usuario puede crear una lista de direcciones al iniciar el broker, la cual que será utilizada para comprobar quién tiene acceso. Esta lista es pasada a la función principal del broker, mostrada en el Listado 4.20. Por defecto, la whitelist es un puntero nulo, utilizado para indicar que no se ha especificado ninguna. En caso de estar especificada la whitelist, el broker, al recibir mensajes, primero verifica si el emisor se encuentra en dicha lista, y si no es así no procesa el mensaje.

Listado 4.20: Versión final del pseudocódigo de la declaración de funciones del broker

```

1 #definir BRO_DEFAULT_WHITELIST = nullptr
2 #definir BRO_DEFAULT_LOGGER = initializeSerialLogger(BROKER)
3 #definir BRO_DEFAULT_PERSISTENCE = false
4 #definir BRO_DEFAULT_CS_SD_PIN = -1
5
6 Función initBroker(MACAddrList whitelist = BRO_DEFAULT_WHITELIST, Elog
    ↳ logger = BRO_DEFAULT_LOGGER, bool persistence =
    ↳ BRO_DEFAULT_PERSISTENCE, int csSdPin = BRO_DEFAULT_CS_SD_PIN):

```

7

Levantar el broker o retornar un código de error

- Arreglar y probar error al crear logs en la tarjeta SD (peso: 13).
- Arreglar y probar bug de creación de ficheros erróneos para la persistencia (peso: 13).
- Redactar Estado del Arte (peso: 40).

4.12. Iteración 11

Periodo: 24/05/2024 - 07/06/2024 (2 semanas). Peso total: 106.

El objetivo principal de este sprint fue redactar la memoria y preparar la documentación Doxygen.

Tareas:

- Redactar Estado del arte (peso: 40) y Herramientas y metodología (peso: 40).
- Añadir descripciones Doxygen y corregir las ya existentes (peso: 13). Mejorar la documentación de las funciones que el usuario puede utilizar con descripciones detalladas, que serán mostradas en la página Doxygen.
- Migrar la librería al nuevo nombre (peso: 8). Finalmente, se decidió por el nombre “LoboMQ”, por lo que se reemplazaron todos los nombres provisionales de los ficheros, las variables y cualquier otra referencia a la herramienta.
- Configurar Doxygen y Doxygen Awesome (peso: 13). Establecer las propiedades necesarias en el fichero Doxyfile y descargar las hojas de estilo Doxygen Awesome para generar un HTML que muestre la documentación del proyecto.

4.13. Iteración 12

Periodo: 08/06/2024 - 21/06/2024 (2 semanas). Peso total: 100.

El objetivo principal de este sprint fue redactar la memoria y arreglar Doxygen.

Tareas:

- Redactar Estado del arte (peso: 40) y Herramientas y metodología (peso: 40).
- Corregir los comentarios y la generación de Doxygen (peso: 20). Realizado con el fin de mostrar la documentación de forma correcta.

4.14. Iteración 13

Periodo: 22/06/2024 - 05/07/2024 (2 semanas). Peso total: 116.

El objetivo principal de este sprint fue redactar la memoria, realizar pequeñas correcciones, personalizar la herramienta y crear un test para probar el funcionamiento de los clientes.

Tareas:

- Añadir comprobación ASCII del topic al publicar (peso: 5). Arreglar esta opción ya existente en el caso de la suscripción.
- Eliminar el parámetro `csPin` no utilizado al recuperar objetos `BrokerTopic` de la tarjeta SD (peso: 5). Mejorar el código evitando la asignación innecesaria de parámetros.
- Actualizar y probar la librería Elog (peso: 8). Hasta el momento, en el desarrollo se ha utilizado la versión 1.1.4. Para asegurar que el uso de la librería siga siendo correcto, se realizan pruebas con la reciente versión 1.1.6.
- Añadir `readme` y un logo (peso: 5). Mejorar la documentación del proyecto mediante un fichero que lo describa, y crear un logo que añada identidad a la herramienta.
- Mejorar la generación de la documentación Doxygen y añadir comentarios (peso: 8).
- Crear test de publicador y suscriptor (peso: 13). Asegurar que las funciones de crear mensaje, extraer contenido y comprobar el topic funcionan correctamente, siendo además útil para comprobar estas funciones luego de que surja alguna modificación.
- Redactar Resultados (peso: 40), Estado del arte y Herramientas y metodología (peso: 40).

4.15. Iteración 14

Periodo: 06/07/2024 - 12/07/2024 (1 semana). Peso total: 88.

El objetivo principal de este sprint fue arreglar algunos fallos, ampliar el tamaño de los mensajes y los topics, y finalizar el proyecto.

Tareas:

- Cambiar el nombre del repositorio (peso: 3). Renombrar el repositorio a “LoboMQ”, completando así el traspaso de la librería a dicho nombre.
- Añadir tamaño como parámetro de `publish` (peso: 3). Tras descubrirse un fallo en el procesamiento de los mensajes, debido a que siempre se publicaban mensajes de 4 bytes, se identificó que la causa residía en el empaquetado, el cual tenía la capacidad de obtener el tamaño del parámetro `void *payload`. La solución consistió en añadir el nuevo parámetro `size_t payloadSize` y asignarlo correctamente.
- Ampliar el tamaño de `topic` y `payload` (peso: 8). Hasta el momento, el tamaño de ambos era de 10 y 64 bytes respectivamente, lo cual no aprovechaba el espacio disponible en un mensaje ESP-NOW. Ambos fueron ampliados hasta 24 y 120 bytes.
- Crear ejemplo básico publicador-suscriptor (peso: 8). El resultado de esta tarea se encuentra en el Anexo /TODO: *incluir numero*.
- Preparar código de demostración (peso: 13). /TODO: *redactar, diciendo que lo tengo en X anexo. Sino, decir que lo hice y ya/*
- Configurar GitHub Actions para desplegar la página Doxygen y publicar la librería en el registro de PlatformIO (peso: 13). Automatizar estos procesos permitirá trabajar en el repositorio en el futuro sin preocuparse de dichas tareas.
- Finalizar memoria (peso: 40).

4.16. Diseño final del protocolo

Una vez finalizado el desarrollo, los artefactos del protocolo quedan definidos. En LoboMQ existen dos tipos de nodos: el broker y los clientes, que interactúan con el sistema según se muestra en el diagrama de casos de uso de la Figura 4.2. El broker gestiona las suscripciones y actúa como un centro de mensajería, redirigiendo las publicaciones a los suscriptores interesados en el mismo tema. Los clientes, por su parte, publican mensajes en temas y se suscriben o desuscriben de estos. Tanto la publicación como la suscripción o desuscripción se transmiten de forma asíncrona, permitiendo la ejecución de tareas sin estar bloqueados esperando una respuesta.

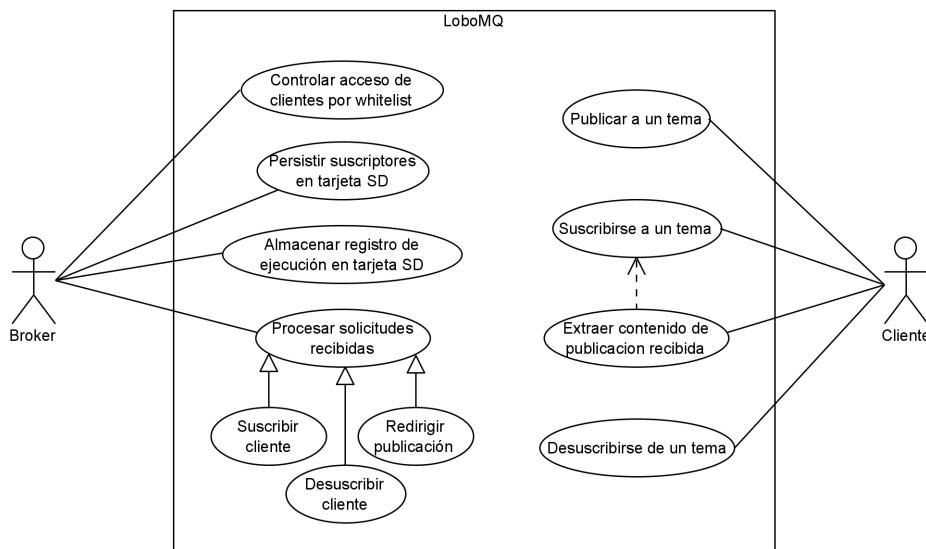


Figura 4.2: Diagrama de casos de uso de LoboMQ

Se identifican tres tipos de mensajes: `SubscribeAnnouncement`, `UnsubscribeAnnouncement` y `PublishContent`, cuyos campos se muestran en la Figura 4.3. Estos mensajes tienen en común los campos `type`, que identifica el tipo de mensaje, y `topic`, el tema al cual se relaciona el mensaje. El `topic`, al igual que en MQTT, admite el uso de wildcards, pero en este caso está limitado a 24 caracteres. Además, el contenido a publicar no puede superar los 120 bytes, ya que se ha ideado que el mensaje ocupe 152 bytes para no sobrecargar la memoria del microcontrolador. La estructura de este contenido debe ser conocida tanto por el publicador como por el suscriptor para poder procesarla correctamente.

Los diagramas de secuencia mostrados en las Figuras 4.4 y 4.5 demuestran el procesamiento de los mensajes en este protocolo de la siguiente manera:

- Se comprueba si el emisor del mensaje tiene acceso. Si es así, el mensaje se procesa.
- Con cada suscripción a un tema, el broker registra dicho tema en caso de no existir, y le asocia el suscriptor. En la librería LoboMQ, se utiliza la clase `BrokerTopic` para representar un tema.
- Con cada desuscripción de un tema, el broker desasocia el suscriptor del tema y, en el caso de estar vacío, elimina el tema del registro.

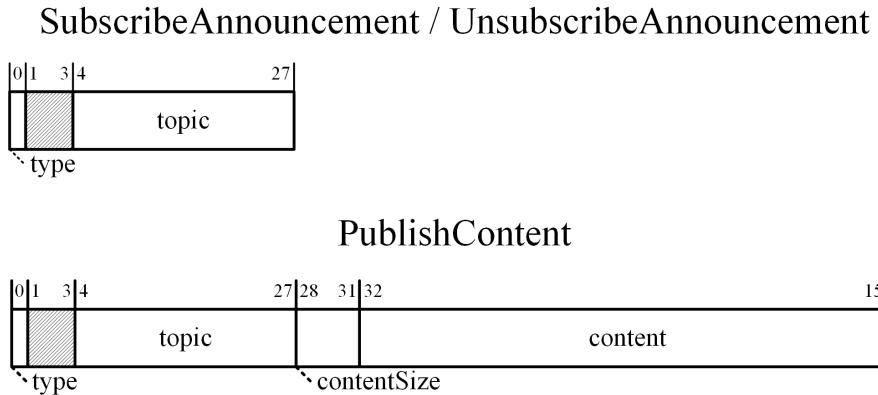


Figura 4.3: Formato de mensajes LoboMQ

- Con cada publicación a un tema, se comprueba si dicho tema es compatible con cada tema del registro. Por cada uno que sí lo sea, se les envía la publicación a sus suscriptores.

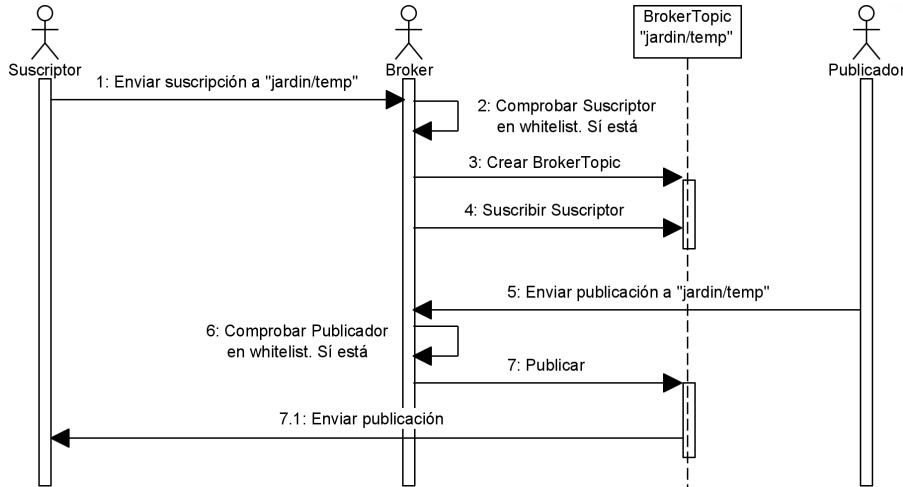


Figura 4.4: Diagrama de secuencia de LoboMQ, con suscripción y publicación a un tema

La implementación de este protocolo se realiza mediante la creación de código en C++ que utiliza la homónima librería LoboMQ, la cual está disponible tanto en el anteriormente mencionado repositorio como en el registro de PlatformIO /TODO: insertar enlace/. El manual de uso de la librería está disponible en el Anexo C, y el diagrama de clases completo se encuentra en el Anexo B. Por otro lado, la documentación completa de la librería está disponible en la página /TODO: insertar enlace/.

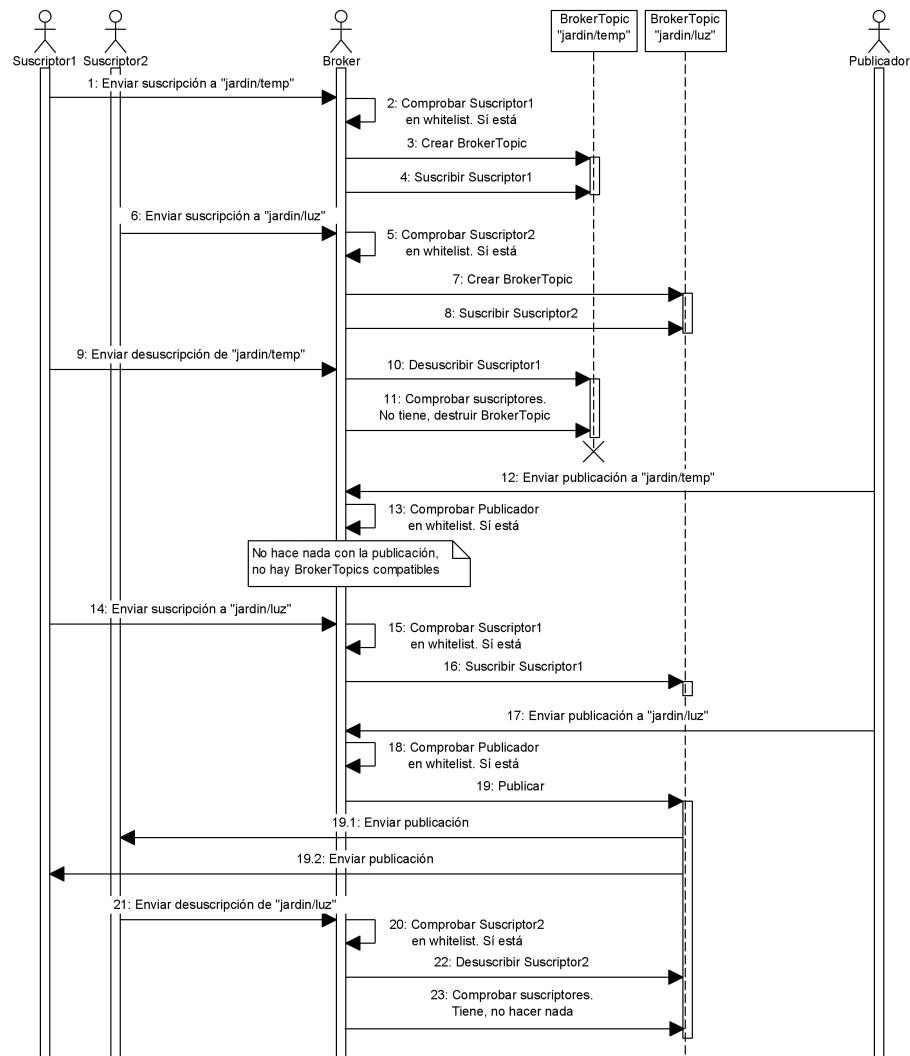


Figura 4.5: Diagrama de secuencia de LoboMQ, con varios suscriptores y temas

Capítulo 5

Conclusiones

En este capítulo final se presentan las conclusiones respecto a cómo se ha abordado el problema principal, analizando los objetivos cumplidos y el coste total del proyecto, y revisando las competencias de la formación académica adquiridas. Finalmente, se realiza una autocritica del proyecto, señalando las posibles mejoras que se podrían implementar en el futuro.

5.1. Revisión de los objetivos

En esta sección se analiza la consecución del objetivo principal del proyecto, el diseño y la implementación de un protocolo de mensajería publicador-subscriptor inspirado en **MQTT** y que haga uso del protocolo de comunicación inalámbrica **ESP-NOW**. Este objetivo se alcanzó a través de los siguientes objetivos específicos:

- a. **Diseñar un protocolo de mensajería para redes ESP-NOW que incorpore características de publicación y suscripción similares a MQTT.** El diseño se ha alcanzado observando el funcionamiento y las características de MQTT y los límites de ESP-NOW. En base a esto, se definieron los tipos de mensaje intercambiados, los roles y las acciones realizables.
- b. **Implementar librerías de software que permitan a los desarrolladores integrar fácilmente el protocolo propuesto en sus aplicaciones del Internet de las Cosas.** Este objetivo ha sido cumplido al crear una librería basada en la especificación anterior del protocolo que contiene las funciones necesarias para desplegar el broker, publicar, suscribirse o desuscribirse, y otras adicionales que complementan el uso del protocolo. Estas funciones fueron creadas con empatía hacia el usuario novato, quitándole en la mayor cantidad posible la responsabilidad de manejar la comunicación del protocolo mediante funciones que abstraen la creación de tareas y colas, la conexión con los nodos y la creación de los mensajes, y cuyos parámetros contienen lo justo y necesario. El código fuente ha sido desarrollado siguiendo criterios de calidad (listados en el Apartado [3.2.6](#)), permitiendo al usuario comprender la ejecución de las funciones y el motivo de la existencia de las variables y estructuras. A esta característica se le suma el estar disponible la librería de manera pública en GitHub, posibilitando a que otros desarrolladores tomen la librería y creen su propia versión o aporten a la ya existente con

funcionalidades nuevas.

- c. **Presentar demostraciones o casos de uso que ilustren la utilidad y aplicabilidad del protocolo en contextos reales del Internet de las Cosas.** El desarrollo del manual de uso y los ejemplos mostrados en el Anexo C permitieron cumplir este objetivo, listando y demostrando el uso de las diferentes funcionalidades integradas en LoboMQ.

Adicionalmente, la lista de objetivos se ha desviado desde la original según se han progresado y descubierto nuevas oportunidades para mejorar el proyecto. Estas desviaciones se listan a continuación:

- **Proteger el uso de la solución mediante control de acceso.** Se ha logrado añadir una medida de seguridad a través de una lista blanca que lista los remitentes aceptados por el broker, evitando un posible robo de información o manipulación de mensajes.
- **Crear un mecanismo de recuperación de información tras un reinicio en el broker.** Los suscriptores y los temas se almacenan como objetos en la memoria de la placa, y además se ha añadido la posibilidad de almacenarse en la tarjeta micro SD en formato **JSON**. Cuando ocurre un reinicio en la placa, la información almacenada en la RAM es eliminada, pero con la persistencia es posible recuperar los suscriptores y temas que estaban en uso.
- **Permitir al usuario adaptar el protocolo a sus necesidades.** Se han parametrizado las funciones, más en concreto la de levantar el broker, para no forzar al usuario a utilizar todas las funcionalidades o utilizar únicamente las esenciales. Estos parámetros dan al usuario la capacidad de elección si su sistema necesita persistencia y control de acceso. Por otro lado, el usuario también puede decidir sobre el uso del logger o gestor de mensajes de registro y los tipo de mensajes que imprime y donde los imprime.
- **Mantener al usuario desarrollador informado y ofrecer la capacidad de adaptarse al estado de ejecución.** El ya mencionado logger es una funcionalidad importante en este caso, ya que establecerlo al nivel DEBUG permite observar los tamaños de mensajes intercambiados, en el caso de los nodos cliente, y la gestión de temas y mensajes en el caso de los nodos broker. Por otro lado, la implementación de códigos de error permite al desarrollador utilizar una función y controlar el resultado de esta para, por ejemplo, realizar determinadas acciones en caso de error.
- **Facilitar el acceso a la documentación de la librería.** El uso de Doxygen y GitHub Pages permite al usuario acceder a la documentación de la librería desde cualquier dispositivo, sin necesidad de utilizar el IDE ni revisar ficheros de código fuente manualmente.

5.2. Presupuesto

Los costes del proyecto son estimados y listados en la Tabla 5.1, resultando en un total de 15.599,54 €. Es importante destacar que este total no refleja costes adicionales como la electricidad y el Internet. Por otro lado, la licencia de Visual Paradigm utilizada es de un mes, ya que esta herramienta se empleó en las fases finales del proyecto y no a lo largo del mismo. El salario del desarrollador se calcula en base al salario

medio de un ingeniero de software en España, pero aplicado a tiempo parcial, debido a que no hubo un horario fijo para la realización del proyecto, variando entre 1 y 8 horas diarias.

Tabla 5.1: Estimación de costes del proyecto

Concepto	Precio/unidad	Cantidad	Precio total
Ordenador Lenovo Ideapad 3 15ALC6	699,00 € [10]	1	699,00 €
Placa ESP32 DEVKIT V1	8,00 € [14]	2	16,00 €
Placa ESP32-2432S028R	11,51 € [6]	1	11,51 €
Placa de pruebas	1,79 € [18]	2	3,58 €
20 cables puente	0,90 € [1]	1	0,90 €
Sensor DHT11	1,99 € [5]	1	1,99 €
Potenciómetro BQ Zum Kit	3,55 € [31]	1	3,55€
Módulo lector de tarjeta microSD	6,21 € [22]	1	6,21 €
Tarjeta microSD	5,50 € [12]	1	5,50 €
Licencia Windows 10	145,00 € [4]	1	145,00 €
Licencia Microsoft 365, 10 meses	7,00 €/mes [51]	1	70,00 €
Licencia Visual Paradigm, 1 mes	6,00 \$/mes [23]	1	5,50 €
Licencia Smart Fields, 10 meses	4,99 \$/mes [21]	1	45,80 €
Salario desarrollador, 10 meses	1.458,50 € / mes [20]	1	14.585,00 €
TOTAL			15.599,54 €

5.3. Competencias Específicas de Intensificación Adquiridas y/o Re-forzadas

El desarrollo de este TFG ha permitido adquirir, junto al resto de competencias del Grado en Ingeniería Informática, las competencias específicas de intensificación en Sistemas de Información, listadas y justificadas a continuación.

- **[SI1] Capacidad de integrar soluciones de Tecnologías de la Información y las Comunicaciones y procesos empresariales para satisfacer las necesidades de información de las organizaciones, permitiéndoles alcanzar sus objetivos de forma efectiva y eficiente, dándoles así ventajas competitivas.** El protocolo LoboMQ ofrece un método de comunicación indirecta entre microcontroladores comúnmente utilizados en IoT, posibilitando su uso en el ámbito empresarial para desarrollar soluciones que aprovechen el valor y las ventajas del IoT.
- **[SI2] Capacidad para determinar los requisitos de los sistemas de información y comunicación de una organización atendiendo a aspectos de seguridad y cumplimiento de la normativa y la legislación vigente.** El alumno y los tutores realizaron una recogida de requisitos para planificar la implementación de LoboMQ, incluyendo roles, mensajes y acciones a realizar en el proceso de

comunicación. Durante esta recogida, se mencionó la conveniencia de implementar un mecanismo de seguridad mediante usuario y contraseña, encriptación y verificación de mensajes y control de acceso, ya que una vulnerabilidad podría resultar en la pérdida de información, manipulación de mensajes y caídas en el sistema. Debido al limitado tiempo de desarrollo, únicamente se implementó un control de acceso mediante whitelist.

- **[SI3] Capacidad para participar activamente en la especificación, diseño, implementación y mantenimiento de los sistemas de información y comunicación.** El proyecto ha implicado la ejecución de las fases comunes del desarrollo de software por parte del autor, distribuidas en varias etapas iterativas y cuyo éxito se refleja en el cumplimiento de los objetivos establecidos.
- **[SI4] Capacidad para comprender y aplicar los principios y prácticas de las organizaciones, de forma que puedan ejercer como enlace entre las comunidades técnica y de gestión de una organización y participar activamente en la formación de los usuarios.** Un principio muy activo durante el desarrollo de LoboMQ ha sido la comprensión del código de la librería, el cual abarca los comentarios, la documentación, la organización del código fuente y los ejemplos. Esto permite entender sus capacidades, limitaciones y funcionamiento, pudiendo así comprobar si es adecuado para implementarlo en un proyecto de una organización y formar a desarrolladores que conozcan la manera de implementarlo y adaptarlo a situaciones específicas.
- **[SI5] Capacidad para comprender y aplicar los principios de la evaluación de riesgos y aplicarlos correctamente en la elaboración y ejecución de planes de actuación.** En cuanto al desarrollo, la elección de Scrum como metodología de gestión de proyectos se justifica, entre otros motivos, por su capacidad de adaptación a requisitos cambiantes, como los que pueden surgir al encontrar fallos. Esto abre la posibilidad de crear parches rápidos o hotfixes en casos de errores críticos, o tareas para futuras iteraciones. En cambio, en LoboMQ se evaluó el riesgo de posibles reinicios de una placa broker durante su ejecución (debido a cortes de energía) y la pérdida de temas y suscriptores, que justificó la creación de un mecanismo de persistencia para almacenar y recuperar los temas y suscriptores en una tarjeta micro SD.
- **[SI6] Capacidad para comprender y aplicar los principios y las técnicas de gestión de la calidad y de la innovación tecnológica en las organizaciones.** El proyecto se ha realizado siguiendo buenas prácticas y estándares, como el uso de metodologías ágiles que permiten mantener una constante comunicación entre los miembros del equipo. Por otro lado, el código resultante ha experimentando un constante incremento de calidad siguiendo unos criterios establecidos para mejorar su uso y comprensión.

5.4. Propuestas de trabajo a futuro

LoboMQ se encuentra en una versión base, pero con una amplia posibilidad de mejorar la comunicación, explotar las capacidades de las placas y, en general, potenciar las prestaciones del protocolo a través de un desarrollo más extenso. Algunas de estas posibilidades se presentan a continuación en forma de propuestas de mejora.

En primer lugar, debido a las tecnologías utilizadas por las placas ESP32 para este protocolo, existe la

posibilidad de ampliar la compatibilidad con varios tipos de placas con similares características, como las ESP8266. Aunque estas placas disponen de un único núcleo y son relativamente antiguas, podría ser interesante implementar una versión de la librería para permitir la interoperabilidad entre distintos tipos de placas.

Un aspecto muy importante en la comunicación es la fiabilidad de la entrega, la cual puede ser mejorada implementando la calidad de servicio o **QoS**, una característica relevante de **MQTT**. A su vez, el procesamiento de los mensajes podría mejorarse añadiendo una opción de persistencia de los mismos, permitiendo así reiniciar el broker sin perder ningún mensaje y entregarlos a las nuevas placas conectadas.

Otra mejora tomada de **MQTT** es la adición del mecanismo *Keep Alive*, que puede permitir tanto al broker como a los suscriptores comprobar la existencia de ambos cada cierto tiempo. Con este mecanismo, el broker puede beneficiarse en cuanto a rendimiento, ya que, si detecta que un suscriptor está desconectado, puede desuscribirlo, evitando así enviar mensajes que nunca llegarán.

El uso de una whitelist de direcciones **MAC** que identifican un único dispositivo puede no resultar óptimo en redes con cambios frecuentes en su configuración. Por ello, se propone mejorar este aspecto mediante un mecanismo de autenticación con usuario y contraseña, permitiendo utilizar ambos para controlar el acceso. Asimismo, encriptar la comunicación puede asegurar los datos, y su implementación es posible utilizando las capacidades integradas en ESP-NOW o desarrollando un mecanismo propio de LoboMQ.

La capacidad de procesamiento de las placas ESP32 puede ser insuficiente en contextos donde la comunicación es constante y se realiza entre una gran cantidad de dispositivos. Añadir la posibilidad de distribuir un sistema que haga uso de LoboMQ puede beneficiar estos contextos, ya que al agregar el uso de varias placas broker se pueden distribuir las tareas (por ejemplo, una placa se encarga de la autenticación y la publicación, mientras que otra realiza las suscripciones) y los suscriptores, posibilitando superar la limitación de 20 dispositivos en redes ESP-NOW. Además, se podría designar un broker de respaldo o backup, manteniendo el sistema en continua operación incluso si un broker se detiene.

Finalmente, no se puede obviar la mejora de rendimiento. Todo software es susceptible de mejoras y este protocolo no es la excepción, especialmente cuando se utilizan placas de baja potencia. La implementación de funciones más eficientes y una adecuada gestión de la memoria son tareas esenciales para esta mejora.

Bibliografía

- [1] 20 cables jumper protoboard de 10cm - Macho/Macho cable jumpers. <https://www.electrocomponentes.es/cable-jumper/229-20-cables-jumper-protoboard-de-10cm-machomacho.html>.
- [2] Agile scrum for Web Development | Neon Rain Interactive. <https://www.neonrain.com/blog/agile-scrum-for-web-development/>.
- [3] Build software better, together. https://education.github.com/discount_requests/application.
- [4] Comprar Windows 11 Home - Microsoft Store es-ES. <https://www.microsoft.com/es-es/d/windows-11-home/dg7gmgf0krt0/000P>.
- [5] DHT11 - Modulo Sensor Temperatura y Humedad relativa DHT 11 3,3v 5v. <https://www.electrocomponentes.es/temperatura/530-modulo-sensor-temperatura-y-humedad-relativa-dht11-33v-5v-con-led.html>.
- [6] ESP32 WIFI y Bluetooth Placa de Desarrollo 2,8 "Módulo de Pantalla Inteligente de 2,8 pulgadas TFT LCD Con Táctil. <https://es.aliexpress.com/item/1005006135387060.html>.
- [7] GitHub | Trello Power-Ups. <https://trello.com/power-ups/55a5d916446f517774210004/github>.
- [8] 2024. GitHub Copilot overview. <https://code.visualstudio.com/docs/copilot/overview>.
- [9] 2023. Internet de las cosas (IoT): dispositivos conectados en el mundo 2015-2027 | Statista. <https://es.statista.com/estadisticas/517654/prevision-de-la-evolucion-de-los-dispositivos-conectados-para-el-internet-de-las-cosas-en-el-mundo/>.
- [10] 2023. Lenovo IdeaPad 3 15ALC6 AMD Ryzen 7 5700U/16 GB/512 GB SSD/15.6" | PcComponentes.com. <https://www.pccomponentes.com/lenovo-ideapad-3-15alc6-amd-ryzen-7-5700u-16-gb-512-gb-ssd-156>.
- [11] 2023. Los idiomas más hablados en el mundo en 2023 | Statista. <https://es.statista.com/estadisticas/635631/los-idiomas-mas-hablados-en-el-mundo/>.
- [12] Magix Micro SD Card HD Series Class10 V10 + SD Adapter Up to 80MB/s (16GB) : Amazon.es: Informática. <https://www.amazon.es/Magix-Micro-Card-MAGIX-Class10-Adapter/dp/B07LGCHSNC/>.
- [13] OpenUP - UTM. <https://www.utm.mx/~caff/doc/OpenUPWeb/>.

- [14] Placa de desarrollo Bluetooth WIFI ESP32 CP2102. <https://robotland.es/arduino-compatible/964-placa-de-desarrollo-bluetooth-wifi-esp32-cp2102.html>.
- [15] PlatformIO Registry - ajlennon/mqtt-sn. <https://registry.platformio.org/libraries/ajlennon/mqtt-sn>.
- [16] PlatformIO Registry - alexcajas/EmbeddedMqttBroker. <https://registry.platformio.org/libraries/alexcajas/EmbeddedMqttBroker>.
- [17] PlatformIO Registry - knolleary/PubSubClient. <https://registry.platformio.org/libraries/knolleary/PubSubClient>.
- [18] Protoboard MB-102 830 puntos enlazable con lineas 830p - Blanca. <https://www.electrocomponentes.es/placas/279-protoboard-mb-102-830-puntos-enlazable-con-lineas-blanca.html>.
- [19] ¿Qué es y para qué sirve una metodología de desarrollo de software? - desarrollodesoftware. <https://desarrollodesoftware.dev/metodologia>.
- [20] Salario para Ingeniero De Software en España - Salario Medio. <https://es.talent.com/salary?job=ingeniero+de+software>.
- [21] Smart Fields | Trello Power-Ups. <https://trello.com/power-ups/5e2212c3ba57415ef2ef9352/smart-fields>.
- [22] SparkFun Electronics BOB-00544. <https://www.digikey.es/es/products/detail/sparkfun-electronics/BOB-00544/5824094>.
- [23] Subscribe Visual Paradigm: BEST UML, BPMN, PMBOK tools! <https://www.visual-paradigm.com/shop/vp.jsp>.
- [24] Tablero Kanban: Qué es, cómo hacerlo y ejemplos | Miro. <https://miro.com/es/agile/que-es-tablero-kanban/>.
- [25] Tecnología LORA y LoRAWAN - CatSensors. <https://www.catsensors.com/es/lorawan/tecnologia-lora-y-lorawan>.
- [26] Agile Alliance Manifiesto por el Desarrollo Ágil de Software. <https://agilemanifesto.org/iso/es/manifesto.html>.
- [27] Agile Alliance Principios del Manifiesto Ágil. <https://agilemanifesto.org/iso/es/principles.html>.
- [28] Atlassian Flujo de trabajo de Gitflow | Atlassian Git Tutorial. <https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow>.
- [29] Atlassian Qué es el control de versiones | Atlassian Git Tutorial. <https://www.atlassian.com/es/git/tutorials/what-is-version-control>.
- [30] Bond, A. Better Comments - Visual Studio marketplace. <https://marketplace.visualstudio.com/items?itemName=aaron-bond.better-comments>.
- [31] BQ Educación Componentes zum Kit Advanced. <https://tienda.bq.com/products/componentes-zum-kit-advanced?variant=37589957935292>.

- [32] BQ Educación Zum Kit Advanced: kit de robótica para niños | BQ Educación. <https://educacion.bq.com/zum-kit-advanced/>.
- [33] Centro Gallego de Robótica Educativa Potenciómetro para proyectos. - Centro Gallego de Robótica Educativa. <https://centroderobotica.com/producto/potenciometro-para-proyectos/>.
- [34] De Negocios Feda, E. 2019. GESTIÓN ÁGIL vs GESTIÓN TRADICIONAL DE PROYECTOS ¿CÓMO ELEGIR? - Escuela de Negocios FEDA. <https://www.escueladenegociosfeda.com/blog/50-la-huella-de-nuestros-docentes/471-gestion-agil-vs-gestion-tradicional-de-proyectos-como-elegir>.
- [35] Driessen, V. 2010. A successful Git branching model. <https://nvie.com/posts/a-successful-git-branching-model/>.
- [36] EcuRed Visual Paradigm - ECUREd. https://www.ecured.cu/Visual_Paradigm.
- [37] Editorial Team - everything RF 2022. What is Thread? - everything RF. <https://www.everythingrf.com/community/what-is-thread>.
- [38] Equipo editorial de IONOS 2019. Extreme Programming: desarrollo ágil llevado al extremo. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/extreme-programming/>.
- [39] Equipo editorial de IONOS 2020. Kanban. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-kanban/>.
- [40] Equipo editorial de IONOS 2019. Scrum: gestión de proyectos con un enfoque flexible. <https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/scrum/>.
- [41] Eridani, D. et al. 2021. Comparative Performance Study of ESP-NOW, Wi-Fi, Bluetooth Protocols based on Range, Transmission Speed, Latency, Energy Usage and Barrier Resistance. *2021 International Seminar on Application for Technology of Information and Communication (iSemantic)* (Sep. 2021), 322–328.
- [42] Esei 2023. Agile Methodology vs. Traditional Project Management. <https://www.eseibusinessschool.com/es/agile-vs-traditional-project-management/>.
- [43] Expósito, D.B. AMD Ryzen 7 5700U: características, especificaciones y precios. <https://www.geektopia.es/es/product/amd/ryzen-7-5700u/>.
- [44] GitHub GitHub Copilot - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=GitHub.copilot>.
- [45] Gruntfuggly Todo Tree - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=Gruntfuggly.todo-tree>.
- [46] Jacobs, M. et al. 2022. What is Agile development? - Azure DevOps. <https://learn.microsoft.com/en-us/devops/plan/what-is-agile-development>.
- [47] Jacobs, M. et al. 2022. What is Agile? - Azure DevOps. <https://learn.microsoft.com/en-us/devops/plan/what-is-agile>.

- [48] Liu, W. TODO Highlight - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=wayou.vscode-todo-highlight>.
- [49] Manole, L. 2023. Edge, Fog, and Cloud Computing: What's the Difference? <https://blog.isa.org/edge-fog-and-cloud-computing-whats-the-difference>.
- [50] Meno, A. 2021. Metodologías de desarrollo de software (tradicionales vs ágiles) – Tecnitium. <https://tecnitium.com/metodologias-de-desarrollo-de-software/>.
- [51] Microsoft Compara todos los planes de Microsoft 365 (anteriormente Office 365): Microsoft Store. <https://www.microsoft.com/es-es/microsoft-365/buy/compare-all-microsoft-365-products?culture=es-es&country=es#M365Personalmonthly>.
- [52] Mishra, B. 2018. TMCAS: An MQTT based Collision Avoidance System for Railway networks. (Jul. 2018), 1–6.
- [53] Nabheetscn 2022. The Magic 80 Char limit which used to exist for developers. <https://community.sap.com/t5/technology-blogs-by-members/the-magic-80-char-limit-which-used-to-exist-for-developers/ba-p/13530175>.
- [54] PlatformIO PlatformIO IDE - Visual Studio marketplace. <https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide>.
- [55] Preston-Werner, T. Semantic Versioning 2.0.0. <https://semver.org/spec/v2.0.0.html>.
- [56] Rcarillo 2023. Qué es LoRa, cómo funciona y características principales. <https://www.vencel.com/que-es-lora-como-funciona-y-caracteristicas-principales/>.
- [57] Rehkopf, M. ¿Qué es un tablero kanban? | Atlassian. <https://www.atlassian.com/es/agile/kanban/boards>.
- [58] Rodal, E. 2021. Diferencias entre fog computing, edge computing y cloud computing. <https://www.podcastindustria40.com/fog-computing/>.
- [59] Schlosser, C. Doxygen Documentation Generator - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=cschlosser.doxdocgen>.
- [60] Schwaber, K. and Sutherland, J. 2020. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>.
- [61] Statista 2024. Gasto mundial en IoT 2018-2023. <https://es.statista.com/estadisticas/1117893/internet-de-las-cosas-gasto-a-nivel-mundial/>.
- [62] Street Side Software Code Spell Checker - Visual Studio Marketplace. <https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker>.
- [63] TecnoDigital 2024. Metodologías de desarrollo de software clásicas: enfoques tradicionales. <https://informatedigital.com/software/metodologias-de-desarrollo-de-software-clasicas/>.
- [64] Urazayev, D. et al. 2023. Indoor Performance Evaluation of ESP-NOW. (May 2023), 1–6.

- [65] Wen, R. 2023. From Git-flow to GitHub-flow. <https://blog.kinto-technologies.com/posts/2023-03-07-From-Git-flow-to-GitHub-flow/>.
- [66] Wikipedia contributors 2024. Agile software development. https://en.wikipedia.org/wiki/Agile_software_development.
- [67] Wikipedia contributors 2024. Thread (network protocol). [https://en.wikipedia.org/wiki/Thread_\(network_protocol\)](https://en.wikipedia.org/wiki/Thread_(network_protocol)).
- [68] Wikipedia contributors 2024. Zigbee. <https://en.wikipedia.org/wiki/Zigbee>.

Anexo A. Pruebas

Se han desarrollado pruebas para comprobar la correcta ejecución del código en la librería. Debido a que PlatformIO no proporciona una manera sencilla de verificar la correcta comunicación entre dos placas, en este capítulo solo se abordan pruebas que evalúan funciones de la librería que no están relacionadas directamente con el envío de datos.

En el caso de los ficheros PubSub, estas pruebas unitarias de caja negra tienen como objetivo probar la correcta identificación de los mensajes y los temas restringidos para publicar o suscribirse.

La primera prueba consiste en crear un mensaje con un número y almacenarlo como bytes, tal como lo haría un publicador. Luego, se llama a la función `isLMQMessage()` para comprobar que se trata de un mensaje compatible con LoboMQ, y posteriormente se extraen los datos creando los correspondientes objetos, llamando a la función `getLMQPayload()` para obtener el contenido de los bytes, al igual que lo haría un suscriptor. Finalmente, se verifica si el número empaquetado sigue siendo el mismo. Este código se detalla en el Listado A.1

Listado A.1: Código para probar la creación de mensajes y la obtención de su contenido

```
1 //Payload structure
2 typedef struct {
3     int number;
4 } CustomPayloadStruct;
5
6 TEST_CASE("Create Message And Check And Get Content") {
7     int number = 666;
8
9     //Create and fill publish message
10    CustomPayloadStruct payloadSend;
11    payloadSend.number = number;
12    PublishContent pubMsg;
13    pubMsg.type = MSGTYPE_PUBLISH;
14    strcpy(pubMsg.topic, "test");
15    pubMsg.contentSize = sizeof(payloadSend);
```

```

16    memcpy(&pubMsg.content, &payloadSend, sizeof(payloadSend));
17
18    //Transform to bytes
19    const uint8_t *data = (const uint8_t *) &pubMsg;
20
21    CHECK(isLMQMessage(data) == true);
22
23    //Extract payload from bytes
24    PayloadContent contentRecv = getLMQPayload(data);
25    CustomPayloadStruct payloadRecv;
26    memcpy(&payloadRecv, &contentRecv.content, contentRecv.contentSize);
27
28    CHECK(payloadRecv.number == number);
29 }

```

El resto de los tests de estos ficheros prueban la compatibilidad de los temas con la publicación y la suscripción. A partir de las verificaciones que se realizan al publicar o suscribirse, se han identificado una serie de casos de prueba, mostrados en la Tabla A.1 junto al parámetro que recibe la función y el resultado esperado. Estos buscan cubrir todas las condiciones y decisiones posibles. Se prueba la comprobación de temas nulos o vacíos, la presencia de caracteres ASCII, el tamaño excesivo de los temas y el uso de caracteres comodín en posiciones prohibidas.

Tabla A.1: Casos de prueba en la comprobación de topics

Función	Parámetros	Resultado
pubTopicCheck	topic = "/aaa/"	LMQ_ERR_SUCCESS
pubTopicCheck	topic = "aaa"	LMQ_ERR_SUCCESS
pubTopicCheck	topic = "aaa/+	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "aaa/+/aaa"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "+/aaa"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "+"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "aaa/#"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "aaa/#/aaa"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "#/aaa"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "#"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "/"	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = ""	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = null	LMQ_ERR_INVAL_TOPIC
pubTopicCheck	topic = "ááá"	LMQ_ERR_INVAL_TOPIC

Función	Parámetros	Resultado
pubTopicCheck	topic = "aaaaaaaaaaaaaaaaaaaaaaaaaaa"	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = "/aaa/"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "aaa"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "aaa/+	LMQ_ERR_SUCCESS
subTopicCheck	topic = "aaa/+/aaa"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "+/aaa"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "+"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "aaa/#"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "aaa/#/aaa"	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = "#/aaa"	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = "#"	LMQ_ERR_SUCCESS
subTopicCheck	topic = "/"	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = ""	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = null	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = "ááá"	LMQ_ERR_INVALID_TOPIC
subTopicCheck	topic = "aaaaaaaaaaaaaaaaaaaaaaaaaaa"	LMQ_ERR_INVALID_TOPIC

Estas pruebas se encuentran en la carpeta `test` de la librería. Han sido desarrolladas con el framework de pruebas Doctest¹, y para su ejecución se requiere de placas compatibles con el modo de depuración y pruebas, conectadas al ordenador encargado de compilar y subir el código del test desde PlatformIO.

¹<https://docs.platformio.org/en/latest/advanced/unit-testing/frameworks/doctest.html>

Anexo B. Diagrama de clases

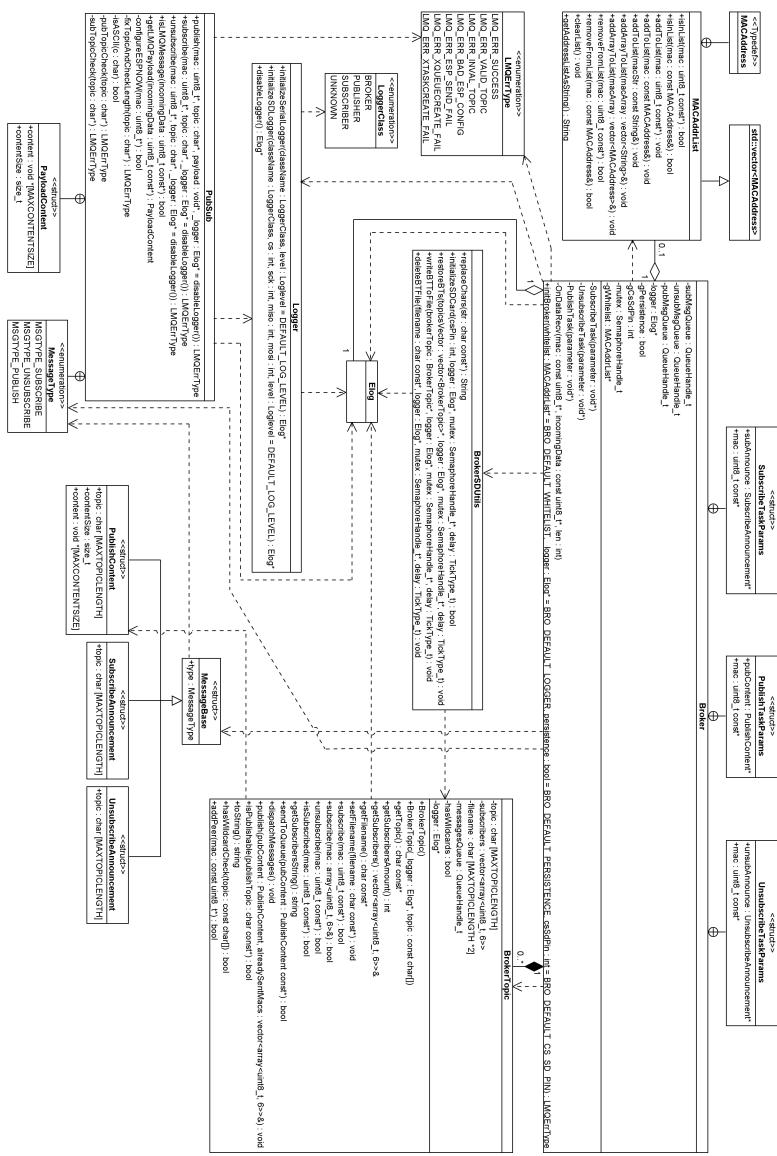


Figura B.1: Diagrama de clases de LoboMQ¹

¹Disponible con mayor resolución en <https://github.com/rubnium/LoboMQ/ETC> /TODO: cambiar enlace/

Anexo C. Manual de uso y ejemplo simple

Este anexo demuestra el correcto uso de la librería LoboMQ por parte del usuario final, explicando sus componentes y proporcionando ejemplos.

Tras la correcta instalación de la dependencia de la librería, la cual está fuera del alcance de este anexo, es posible comenzar a integrar la librería en el código de cualquier proyecto. Dependiendo de las funciones a utilizar, será necesario incluir los ficheros de cabecera correspondientes, listados a continuación.

- `#include <LoboMQ/Broker.h>`:
 - Función `initBroker`: configurar las tareas y colas adecuadas para gestionar los mensajes recibidos y otorgar a la placa/nodo el rol de broker.
 - Parámetro `whitelist`: lista con las direcciones MAC permitidas.
 - Parámetro `logger`: objeto de tipo `Elog` para la salida de mensajes de registro.
 - Parámetro `persistence`: indica si se desea activar la persistencia.
 - Parámetro `csSdPin`: pin *Chip Select* del lector de tarjeta SD.
 - Retorna un código de error.
 - Clase `MACAddrList`: lista de direcciones MAC. Hereda de `std::vector`, añadiendo nuevas funciones:
 - `isInList`: comprobar si una dirección MAC se encuentra en la lista.
 - Parámetro `mac`: dirección MAC a comprobar.
 - Retorna `true` o `false`.
 - `addToList`: añadir una dirección MAC a la lista. No inserta duplicados.
 - Parámetro `mac`: dirección MAC a añadir.
 - `addArrayToList`: añadir múltiples direcciones MAC de un array a la lista. No inserta duplicados.
 - Parámetro `macArray`: array de direcciones MAC.
 - `removeFromList`: eliminar una dirección MAC de la lista.
 - Parámetro `mac`: dirección MAC a eliminar.
 - Retorna `true` si se eliminó correctamente, o `false`.
 - `clearList`: vaciar la lista.
 - `getAddressListAsString`: obtener un string con todas las direcciones MAC de la lista.
- `#include <LoboMQ/PubSub.h>`. Funciones incluidas:

- **publish:** tomar el payload, construir un mensaje y publicarlo en el tema especificado en el broker.
 - Parámetro **mac**: dirección MAC del broker.
 - Parámetro **topic**: tema al que se publicará el mensaje.
 - Parámetro **payload**: contenido del mensaje.
 - Parámetro **payloadSize**: tamaño del payload en bytes.
 - Parámetro **logger**: objeto de tipo Elog para la salida de mensajes de registro.
 - Retorna un código de error.
- **subscribe:** enviar un mensaje al broker anunciando interés en recibir todos los mensajes compatibles con el tema especificado.
 - Parámetro **mac**: dirección MAC del broker.
 - Parámetro **topic**: tema al que se suscribe el emisor.
 - Parámetro **logger**: objeto de tipo Elog para la salida de mensajes de registro.
 - Retorna un código de error.
- **unsubscribe:** enviar un mensaje al broker anunciando desinterés en recibir todos los mensajes compatibles con el tema especificado.
 - Parámetro **mac**: dirección MAC del broker.
 - Parámetro **topic**: tema del cual se desuscribe el emisor.
 - Parámetro **logger**: objeto de tipo Elog para la salida de mensajes de registro.
 - Retorna un código de error.
- **isLMQMessage:** comprobar si los bytes representan un mensaje de publicación LoboMQ.
 - Parámetro **incomingData**: bytes a comprobar.
 - Retorna **true** o **false**.
- **getLMQPayload:** extraer el payload de los bytes de un mensaje de publicación LoboMQ.
 - Parámetro **incomingData**: bytes del mensaje.
 - Retorna un objeto **PayloadContent**.
- **#include <LoboMQ.h>**. Incluye todas las funciones mencionadas anteriormente.

Estas cabeceras comparten el objeto **Elog logger**, el cual hereda características relevantes, como la función **log** para mostrar mensajes de registro y el listado de niveles de registro: **ALERT**, **CRITICAL**, **ERROR**, **WARNING**, **NOTICE**, **INFO** y **DEBUG**. Esta librería incluye funciones para crear este objeto fácilmente:

- **initializeSerialLogger:** crear un objeto Elog que imprime mensajes en el monitor serial.
 - Parámetro **className**: clase de registro.
 - Parámetro **level**: nivel más crítico de registro interesado.
 - Retorna un objeto Elog.
- **initializeSDLogger:** crear un objeto Elog que imprime mensajes en la tarjeta SD.
 - Parámetro **className**: clase de registro.
 - Parámetros **cs**, **sck**, **miso** y **mosi**: pines del módulo lector de la tarjeta SD.
 - Parámetro **level**: nivel más crítico de registro interesado.
 - Retorna un objeto Elog.

- `disableLogger`: crear un objeto Elog sin la habilidad para mostrar mensajes de registro.
 - Retorna un objeto Elog.

El valor de `className` debe ser BROKER, PUBLISHER, SUBSCRIBER o UNKNOWN, dependiendo del rol del nodo que cree el objeto. Esto es importante, ya que al ver el registro, permite conocer quién lo está emitiendo.

Cabe aclarar que cada vez que se mencionan los códigos de error en las funciones principales de LoboMQ, se refiere a los mostrados en el apartado [4.10 Iteración 9](#). Por otro lado, el uso de temas o topics está restringido de la siguiente manera:

- Existe un límite de 24 caracteres.
- Solo se admiten caracteres ASCII.
- El uso de caracteres comodín es similar al del protocolo MQTT, explicado en el apartado [2.2.1. Message Queue Telemetry Transport \(MQTT\)](#).
- No se puede publicar en temas con caracteres comodín.
- No se puede utilizar el carácter comodín # en un nivel intermedio. Por ejemplo, `sensor#/temp`.

Además, resulta ideal conocer las estructuras de los mensajes de publicación, ya que de esta manera el desarrollador puede extraer otros campos que esta librería no aporta con facilidad. Los mensajes publicados y los que reciben los suscriptores utilizan la estructura `PublishContent`, formada por el topic, el contenido en bytes y el tamaño del contenido. Por otro lado, este mensaje es filtrado y extraído en una estructura `PayloadContent` al utilizar la función `getLMQPayload`. Ambas estructuras mencionadas se muestran en el Listado C.1.

Listado C.1: Estructuras de `PublishContent` y `PayloadContent`

```

1 typedef struct : public MessageBase {
2     char topic[MAXTOPICLENGTH];
3     size_t contentSize;
4     void* content[MAXCONTENTSIZE];
5 } PublishContent;
6
7 typedef struct {
8     size_t contentSize;
9     void* content[MAXCONTENTSIZE];
10 } PayloadContent;

```

Una vez puesta a disposición del usuario final la funcionalidad de la librería, se procede a ver cómo se utiliza para los distintos tipos de nodos: broker y cliente, teniendo este último la capacidad de publicar y suscribirse o desuscribirse.

C.1. Broker

Este es el nodo más sencillo de desplegar, ya que únicamente es necesario llamar a la función `initBroker` ↵ . Si es necesario aprovechar las funcionalidades disponibles, se debe seguir la siguiente secuencia de ejecución:

1. Crear la whitelist de direcciones MAC. Si no es necesario, se debe utilizar su valor por defecto, el cual indica que no se usa este método de control de acceso.
2. Crear el logger. Su valor por defecto imprime los mensajes por el monitor serial.
3. Llamar a la función `initBroker`. Si es necesario, se debe indicar la persistencia.

El siguiente ejemplo muestra un broker con persistencia y control de acceso activado. Para ello, primero se han realizado las conexiones adecuadas entre la placa ESP32 y el lector de tarjetas SD, como se muestra en la Figura C.1. Luego, se presenta el desarrollo del código de este ejemplo, representado en el Listado C.2.

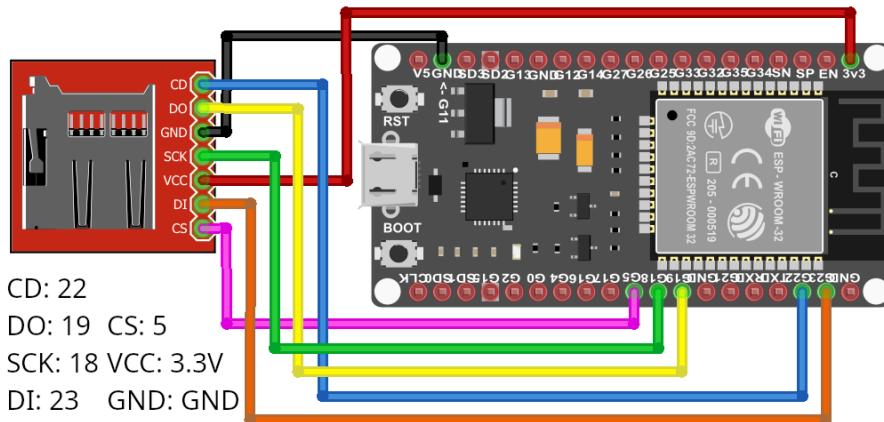


Figura C.1: Diagrama de conexión del broker

Listado C.2: Código del broker de ejemplo

```

1 #include <LoboMQ/Broker.h>
2
3 void setup() {
4     Serial.begin(9600);
5     MACAddrList *whitelist = new MACAddrList();
6     whitelist->addArrayToList((std::vector<String>){
7         "08:B6:1F:BA:04:F8",
8         "C0:49:EF:CB:99:10"
9     });
10
11 //Choose logger
12 Elog *logger = initializeSDLogger(BROKER, 5, 18, 19, 23, DEBUG);

```

```

13
14 //Initialize broker
15 initBroker(whitelist, logger, true, 5);
16 }
17
18 void loop() { }
```

Este ejemplo se encuentra en la carpeta `examples` de la librería.

C.2. Publicador

En el caso de un nodo que publique contenido, la secuencia de ejecución es la siguiente:

1. Crear el logger. Su valor por defecto imprime los mensajes por el monitor serial.
2. Crear la estructura del contenido a publicar. Debe ser la misma estructura en el suscriptor.
3. Crear el contenido a publicar.
4. Llamar a la función `publish`. Uno de los parámetros es la dirección del broker, que debe de obtenerse previamente. Una manera de hacerlo es revisando los logs al iniciar el broker.

El siguiente código de ejemplo, mostrado en el Listado C.3, contiene un publicador que genera un número aleatorio y lo publica cada dos segundos en el tema “topic1”.

Listado C.3: Código del publicador de ejemplo

```

1 #include <LoboMQ/PubSub.h>
2
3 uint8_t brokerAddr[] = {0x24, 0xDC, 0xC3, 0x9C, 0x7E, 0x40}; //Broker MAC
4   ↛ destination
5
6 Elog *_logger;
7
8 void setup() {
9   Serial.begin(9600);
10  randomSeed(analogRead(0)); //to generate random numbers
11
12 //Choose logger:
13 _logger = initializeSerialLogger(PUBLISHER, DEBUG);
14
15 _logger->log(INFO, "Started publisher board on %s!", WiFi.macAddress().
16   ↛ c_str()); }
```

```

17 void loop() {
18     int numberExample = random(101); //generates random number
19     publish(brokerAddr, "topic1", &numberExample, sizeof(numberExample),
20             ↪ _logger);
21     sleep(2);
22 }
```

Este ejemplo se encuentra en la carpeta examples de la librería.

C.3. Suscriptor y desuscriptor

A diferencia de los ejemplos previos, el suscriptor requiere aplicar más lógica, ya que estos nodos procesan los mensajes recibidos. La secuencia de ejecución es la siguiente:

1. Crear el logger. Su valor por defecto imprime los mensajes por el monitor serial.
2. Crear la estructura del contenido a publicar. Debe ser la misma estructura en el publicador.
3. Configurar ESP-NOW junto a la función que procesa los mensajes.
4. Llamar a la función `subscribe`. Uno de los parámetros es la dirección del broker, que debe de obtenerse previamente. Una manera de hacerlo es revisando los logs al iniciar el broker.

Junto a esta secuencia, en la función utilizada para procesar los mensajes, resulta esencial llamar a las funciones de LoboMQ `isLMQMessage` y `getLMQPayload` para comprobar que el mensaje recibido forma parte de LoboMQ y obtener sus bytes de contenido. Estos bytes luego pueden ser copiados a un objeto de la estructura definida en el paso 2.

En cambio, la secuencia de ejecución de la desuscripción es más simple:

1. Crear el logger. Su valor por defecto imprime los mensajes por el monitor serial.
2. Configurar ESP-NOW.
3. Llamar a la función `unsubscribe`. Uno de los parámetros es la dirección del broker, que debe de obtenerse previamente. Una manera de hacerlo es revisando los logs al iniciar el broker.

El siguiente código de ejemplo muestra el uso de estas características, suscribiéndose a cualquier tema y, 20 segundos más tarde, desuscribiéndose. Además, muestra una forma de procesar los mensajes recibidos, en este caso un entero. Todo ello es observable en el Listado C.4.

Listado C.4: Código del suscriptor y desuscriptor de ejemplo

```

1 #include <LoboMQ/PubSub.h>
2
3 uint8_t brokerAddr[] = {0x24, 0xDC, 0xC3, 0x9C, 0x7E, 0x40}; //Broker MAC
4             ↪ destination
5 Elog *_logger;
```

```
6
7 //Function called on every ESP-NOW message reception
8 void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len) {
9     if (isLMQMessage(incomingData)) {
10         PayloadContent content = getLMQPayload(incomingData);
11
12         int numberExample;
13         memcpy(&numberExample, &content.content, content.contentSize);
14
15         _logger->log(INFO, "Received LMQ message of %dB", content.contentSize);
16         _logger->log(INFO, "Number: %d", numberExample);
17     } else {
18         _logger->log(ERROR, "Invalid message type received");
19     }
20 }
21
22 void setup() {
23     Serial.begin(9600);
24
25     //Choose logger:
26     _logger = initializeSerialLogger(SUBSCRIBER, DEBUG);
27
28     WiFi.mode(WIFI_STA);
29
30     //Initialize ESP-NOW and set up receive callback
31     if (esp_now_init() != ESP_OK || esp_now_register_recv_cb(OnDataRecv) !=
32         ↪ ESP_OK) {
33         _logger->log(ERROR, "Couldn't initialize ESP-NOW");
34         exit(1);
35     }
36
37     _logger->log(INFO, "Started subscriber board on %s!", WiFi.macAddress().
38         ↪ c_str());
39     subscribe(brokerAddr, "#", _logger);
40     sleep(20);
41     unsubscribe(brokerAddr, "#", _logger);
42 }
```

42 | **void loop()** { }

Este ejemplo se encuentra en la carpeta `examples` de la librería.

Anexo D. Demostración con sensores

No hace falta que me lo reviseis. Solo va a ser explicar la demo y poner pseudocodigos.

/TODO: hacer/

