ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Máster

Análisis y estudio de diferentes tecnologías para aplicaciones de telecontrol

(Analysis and study of different technologies for remote control applications)

Para acceder al Título de

Máster Universitario en

Ingeniería de Telecomunicación

Autora: Lara Martínez Fernández

Julio – 2025

MASTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE MASTER

Realizado por: Lara Martínez Fernández

Director del TFM: Roberto Sanz Gil

Título: "Análisis y estudio de diferentes tecnologías para aplicaciones de telecontrol "

Title: "Analysis and study of different technologies for remote control applications "

Presentado a examen el día: 17 de julio de 2025

para acceder al Título de

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Composición del Tribunal:

Presidente (Apellidos, Nombre): Fernández Ibáñez, Tomás

Secretario (Apellidos, Nombre): García García, José Ángel

Vocal (Apellidos, Nombre): Quintela Incera, María Ángeles

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo: El Presidente

Fdo: El Secretario

Fdo: El Vocal

Fdo: El Director del TFM
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Trabajo Fin de Máster Nº
(a asignar por Secretaría)

Resumen

A lo largo del presente trabajo se ha explorado en profundidad la integración de dispositivos IoT con la plataforma Home Assistant, empleando protocolos como MQTT y Matter para la comunicación eficiente entre sensores, actuadores y el entorno de control. Se ha demostrado la viabilidad de implementar soluciones domóticas personalizadas, partiendo de hardware accesible como las placas ESP32, y aplicando metodologías tanto de configuración local como remota, que permiten la supervisión y gestión del sistema desde cualquier ubicación.

Las pruebas realizadas han evidenciado el correcto funcionamiento del sistema, tanto en términos de comunicación como en la visualización de datos en el dashboard de Home Assistant. Asimismo, se ha puesto de relieve la importancia de aspectos como la seguridad en la red, la interoperabilidad entre protocolos, y la necesidad de estructuras jerárquicas bien definidas para la escalabilidad del sistema.

El enfoque adoptado, basado en plataformas open source, ha permitido una mayor flexibilidad, control y capacidad de adaptación, a la vez que se han presentado diversas vías de acceso remoto, evaluando sus ventajas e inconvenientes en términos de facilidad de implementación, fiabilidad y coste.

Abstract

Throughout this work, the integration of IoT devices with the Home Assistant platform has been explored in depth, using protocols such as MQTT and Matter for efficient communication between sensors, actuators and the control environment. The feasibility of implementing customised home automation solutions has been demonstrated, starting from accessible hardware such as ESP32 boards, and applying both local and remote configuration methodologies, which allow the supervision and management of the system from any location.

The tests carried out have shown the correct functioning of the system, both in terms of communication and in the display of data on the Home Assistant dashboard. It has also highlighted the importance of aspects such as network security, interoperability between protocols, and the need for well-defined hierarchical structures for the scalability of the system.

The approach adopted, based on open source platforms, has allowed for greater flexibility, control and adaptability, while various remote access routes have been presented, evaluating their advantages and disadvantages in terms of ease of implementation, reliability and cost.

Índice de contenidos

Lista de figuras

Lista de acrónimos

6LOWPAN	IPv6 Over Low Power Wireless Personal Area Network
ADC	Analog-to-Digital Converter
API	Application Programming Interface
BLE	Bluetooth Low Energy
CSA	Connectivity Standars Alliance
CASE	Certificate Authenticated Session Establishment
COAP	Constrained Application Protocol
DAC	Device Attestation Certificate
DNS	Domain Name System
EDCH	Elliptic Curve Difffie-Hellman
GPIO	General-Purpose Input/Output
HA	Home Assistant
HMAC	Hash-Based Message Authentication Code
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrussion Detection System
IP	Internet Protocol
ISP	Internet Service Provider
IoT	Internet of Things
LAN	Local Area Network
MQTT	Message Queuing Telemetry Transport
NAT	Network Address Translation
NUC	Next Unit of Computing
PASE	Password Authenticated Session Establishment
QoS	Quality of Service
RoT	Root of Trust
SLAAC	Stateless Address Autoconfiguration
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TLV	Type Lenght Value
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
WI-FI	Wireless Fidelity

1

Capítulo 1 Introducción

La transformación digital ha dado lugar a un notable avance en el desarrollo de sistemas inteligentes orientados al confort, la eficiencia y la automatización de tareas cotidianas. Uno de los ámbitos donde esta evolución ha tenido un mayor impacto es en el entorno doméstico, dando lugar al concepto de *smart home* o vivienda inteligente. En este contexto, los sistemas de automatización del hogar han adquirido una relevancia creciente gracias a su capacidad para integrar sensores, actuadores y plataformas de control que permiten monitorizar y gestionar dispositivos de manera remota o autónoma [1].

La disponibilidad de dispositivos de bajo coste, como las placas de desarrollo ESP32, ha contribuido de forma significativa a la expansión de soluciones domóticas personalizables y accesibles.

Paralelamente, plataformas como Home Assistant (HA) han emergido como soluciones *open source* altamente adaptables y con una amplia comunidad de desarrollo, lo que ha favorecido su uso en proyectos tanto académicos como profesionales [1].

Este trabajo se enmarca dentro de esa evolución tecnológica, abordando el análisis y la integración de diversos componentes de hardware y software en un entorno doméstico inteligente. Se hace especial énfasis en la comunicación eficiente entre dispositivos, explorando distintas estrategias de conectividad como MQTT (Message Queueing Telemetry Transport) y Matter [2][4], así como en la incorporación de herramientas de interacción remota como Telegram.

1.1 Contexto y motivación

En los últimos años, el ecosistema de la Internet de las Cosas (IoT) ha crecido exponencialmente, impulsando el desarrollo de infraestructuras más inteligentes y eficientes. Este crecimiento ha venido acompañado por la necesidad de crear entornos capaces de gestionar múltiples dispositivos de forma centralizada, segura y escalable. La automatización del hogar se ha beneficiado particularmente de esta tendencia, integrando sensores, actuadores, asistentes virtuales y servicios de notificación para mejorar la calidad de vida de los usuarios [2][4].

Frente a soluciones comerciales cerradas y de elevado coste, ha surgido una alternativa basada en el uso de hardware accesible y plataformas de código abierto. En este sentido, HA se ha consolidado como una de las plataformas más robustas, permitiendo la personalización total del sistema domótico y la integración con un amplio abanico de dispositivos y servicios en la nube [1].

El presente trabajo surge de la necesidad de explorar estas posibilidades, evaluando el potencial de integración de tecnologías como ESPHome, MQTT, Matter y herramientas de mensajería instantánea para la creación de un sistema domótico completo y funcional. Además, se pretende estudiar la viabilidad de la comunicación entre dispositivos dentro y fuera de la red local, abriendo la puerta a futuras aplicaciones más avanzadas, como el acceso remoto seguro, el uso de protocolos emergentes y la aplicación de algoritmos de automatización inteligente [2][4].

1.2 Objetivos del trabajo

El objetivo general de este Trabajo Fin de Máster es diseñar, implementar y evaluar un sistema de automatización del hogar basado en tecnologías de código abierto, empleando como núcleo la plataforma proporcionada por Home Assistant (HA). Para ello, se integrarán distintos dispositivos basados en microcontroladores ESP32, así como una serie de sensores y actuadores que permitirán realizar tareas de monitorización y control en tiempo real.

Como objetivos específicos, se plantean los siguientes:

- Estudiar el funcionamiento interno de HA y su arquitectura de integración.
- Configurar dispositivos ESP mediante ESPHome y, posteriormente, mediante código desarrollado con Arduino.

- Establecer la comunicación de los sensores con HA utilizando tanto la API nativa como el protocolo MQTT.
- Evaluar las capacidades de integración del protocolo Matter en el entorno propuesto.
- Realizar una serie de pruebas funcionales que permitan validar el comportamiento del sistema tanto en red local como desde el exterior.
- Analizar el tráfico de red generado por el sistema utilizando herramientas como Wireshark y TCPDump, con el fin de obtener una visión profunda de las comunicaciones subyacentes.
- Implementar mecanismos de interacción y notificación remota a través de plataformas como Telegram.

1.3 Estructura del documento

La estructura del documento se organiza de la siguiente manera:

- **Capítulo 2**: se dedica al estudio de las plataformas de automatización existentes, tanto desde una perspectiva histórica como técnica. Se analizan sus características, fortalezas y limitaciones, prestando especial atención a aquellas soluciones que han tenido un mayor impacto en el ámbito doméstico.

- **Capítulo 3**: aborda el software desde una perspectiva teórica, centrándose en los protocolos y estándares de comunicación relevantes, como MQTT y Matter. Se examinan sus fundamentos, principios de diseño y funcionamiento, así como su impacto en la interoperabilidad y eficiencia de los sistemas domóticos.

- **Capítulo 4**: está orientado al análisis del hardware empleado, incluyendo las placas de desarrollo utilizadas y su interacción con sensores y módulos complementarios. Además, se detallan los procesos de instalación, configuración e integración en el ecosistema domótico propuesto.

- **Capítulo 5**: se documenta el proceso de integración y las pruebas realizadas. Se detallan los procedimientos de implementación, los escenarios de evaluación y los resultados obtenidos, con el fin de valorar la efectividad del sistema desarrollado y su potencial para ser ampliado en futuras fases.

- **Capítulo 6**: se presentan las conclusiones generales del trabajo, una valoración crítica de los resultados obtenidos y diversas líneas de trabajo futuro que podrían derivarse de este proyecto.

2

Capítulo 2 Estado del arte

El desarrollo de sistemas inteligentes para el control del entorno doméstico ha experimentado una evolución significativa en los últimos años, impulsada por el auge del IoT, la expansión del acceso a redes de datos y la creciente demanda de soluciones automatizadas en el hogar. En este contexto, resulta esencial realizar un análisis del estado actual de las tecnologías y estándares implicados en estos entornos, con el objetivo de identificar los fundamentos técnicos sobre los que se sustentan las soluciones actuales y futuras. A lo largo de este capítulo, se presentarán los antecedentes más relevantes en la evolución de la domótica, así como los protocolos de comunicación que permiten la interconexión y control de dispositivos. Finalmente, se abordarán las consideraciones de seguridad asociadas a este tipo de sistemas, con el fin de destacar los riesgos, las medidas de protección existentes y los enfoques más recomendados para garantizar la privacidad e integridad de los datos.

2.1 Antecedentes

En los últimos años, el desarrollo de tecnologías orientadas a la automatización del hogar ha experimentado un notable crecimiento, impulsado por el auge del IoT y la demanda de soluciones que mejoren la comodidad, eficiencia energética y seguridad de los entornos domésticos. Estas tecnologías han permitido la aparición de plataformas de control que gestionan múltiples dispositivos inteligentes desde un único entorno centralizado lo que ha favorecido la integración de luces, sensores, termostatos, cámaras y otros sistemas en infraestructuras domóticas cada vez más sofisticadas [1].

El ecosistema actual de plataformas de automatización del hogar puede clasificarse de forma general en dos grandes categorías: soluciones *open source* y soluciones comerciales de pago. Las plataformas de código abierto, como HA, *openHAB* o *Domoticz*, se caracterizan por su alta flexibilidad, la posibilidad de ser adaptadas a distintos entornos de hardware y su enfoque en la privacidad del usuario. Su naturaleza abierta fomenta el desarrollo comunitario y la constante evolución del software, lo que permite incorporar nuevas funcionalidades y compatibilidades con rapidez [1]. Estas soluciones, requieren conocimientos técnicos más avanzados para su implementación, ofreciendo un control detallado sobre todos los aspectos del sistema.

En contraposición, las soluciones de pago como *Google Nest*, *Amazon Alexa* o *Apple HomeKit* presentan una aproximación más orientada al consumidor general, con interfaces simplificadas y procesos de configuración guiados. Estas plataformas suelen ofrecer una experiencia más amigable, especialmente para usuarios sin conocimientos técnicos, y una integración directa con servicios de nube, asistentes virtuales y ecosistemas cerrados. Sin embargo, su dependencia de infraestructuras externas y la menor capacidad de personalización representan limitaciones importantes para ciertos perfiles de usuario [1].

En ambos casos, la interoperabilidad se ha convertido en un aspecto clave. La capacidad de una plataforma para comunicarse con dispositivos que emplean diferentes protocolos como *Zigbee*, *Z-Wave*, *Bluetooth* o *Wi-Fi* determina en gran medida su viabilidad como sistema domótico. A su vez, esta interoperabilidad plantea desafíos técnicos que deben ser abordados desde el diseño del sistema y la elección de la plataforma.

Adicionalmente, las plataformas modernas no se limitan a permitir el control básico de dispositivos, sino que habilitan funciones avanzadas como la creación de rutinas automatizadas, el monitoreo en tiempo real del consumo energético, el análisis del comportamiento del usuario y la implementación de respuestas adaptativas mediante algoritmos de inteligencia artificial. Todo ello con el objetivo de desarrollar entornos inteligentes que no solo respondan a comandos directos, sino que también sean capaces de anticipar las necesidades del usuario.

En este contexto, el presente trabajo se fundamenta en el estudio y despliegue de soluciones domóticas basadas en plataformas de código abierto, con especial énfasis en Home Assistant, por su alta modularidad, compatibilidad y enfoque en la privacidad local [1].

2.1.1 Home Assistant

Home Assistant se ha consolidado como una de las plataformas de automatización del hogar de código abierto más completas y adaptadas a nivel global. Desarrollada en Python y mantenida por una comunidad activa de colaboradores, su diseño modular y flexible permite la integración de una amplia variedad de dispositivos y protocolos, ofreciendo así una solución altamente personalizable para la gestión inteligente del hogar. Esta plataforma no solo permite el control y monitorización de sensores, luces, enchufes o cámaras, sino también la creación de automatismos complejos y rutinas adaptativas basadas en eventos, condiciones o comportamientos del usuario [1].

Una de las principales ventajas de HA radica en su enfoque en la privacidad y el procesamiento local. A diferencia de otras soluciones comerciales, HA se ejecuta de forma local, sin requerir necesariamente servicios en la nube, lo que garantiza que los datos del usuario permanezcan en su red doméstica. Esta característica ha sido valorada positivamente por la comunidad técnica, ya que reduce el riesgo de exposición de datos sensibles y mejora la capacidad de respuesta del sistema [1].

La arquitectura de Home Assistant se compone de varios elementos clave. El núcleo del sistema es el Home Assistant Core, que gestiona las entidades, estados y servicios disponibles en el sistema. Sobre él se encuentra el Supervisor, encargado de gestionar el sistema operativo, actualizaciones, complementos y copias de seguridad. Esta arquitectura modular es compatible con diversos entornos de instalación, desde máquinas virtuales y contenedores Docker hasta dispositivos como Raspberry Pi o NUCs [1].

Además, HA se apoya en el concepto de *integraciones*, que permiten conectar dispositivos o servicios externos. Cada integración se asocia con una o varias plataformas, y puede representar dispositivos físicos, servicios en la nube, o entidades virtuales. La flexibilidad en el desarrollo de integraciones ha contribuido a una rápida expansión del ecosistema, permitiendo que Home Assistant sea compatible con miles de dispositivos y marcas diferentes [1].

En cuanto a la experiencia de usuario, Home Assistant ofrece un panel de control denominado *dashboard*, completamente personalizable, desde el cual se pueden visualizar estados, ejecutar acciones o acceder a históricos de sensores. Este entorno visual facilita tanto la supervisión como la interacción con el sistema. Adicionalmente, dispone de soporte nativo para plataformas de mensajería como Telegram y servicios de voz como Google Assistant o Alexa, ampliando así las posibilidades de control remoto y notificaciones [1].

Home Assistant ha sido objeto de múltiples estudios en el ámbito académico por su relevancia en proyectos de IoT, domótica y ciberseguridad. Su código abierto y documentación extensa permiten que sea una plataforma idónea tanto para entornos domésticos como para escenarios de investigación, desarrollo y prototipado de soluciones avanzadas de automatización.

2.2 Protocolos y estándares de comunicación

El desarrollo de sistemas de automatización del hogar y entornos IoT se apoya en una diversidad de protocolos y estándares de comunicación que permiten la interoperabilidad entre dispositivos, la transmisión eficiente de datos y el control remoto en tiempo real. Estos protocolos, tanto en su vertiente de red como en la de aplicación, han sido diseñados para operar en entornos con limitaciones de potencia, ancho de banda y procesamiento, características comunes en los dispositivos integrados en soluciones domóticas.

Uno de los protocolos más ampliamente utilizados es MQTT un protocolo ligero basado en el modelo *publish/subscribe*, ideal para redes con conectividad intermitente o baja latencia. MQTT se caracteriza por su bajo consumo de ancho de banda, lo que lo hace especialmente adecuado para sensores y actuadores de bajo coste. Además, permite establecer distintos niveles de calidad de servicio (QoS), ajustando la fiabilidad de entrega en función de las necesidades de la aplicación [3][4].

CoAP (Constrained Application Protocol) es otro protocolo relevante en entornos IoT. Diseñado por la IETF, CoAP está basado en el modelo REST y utiliza UDP en lugar de TCP, lo que reduce la sobrecarga de la conexión. CoAP permite funcionalidades similares a HTTP, como operaciones *GET*, *POST*, *PUT* y *DELETE*, pero adaptadas a dispositivos con recursos limitados. Asimismo, permite la integración con redes 6LoWPAN y protocolos como Thread, fundamentales en la evolución hacia estándares como Matter [3].

En lo que respecta a la comunicación a nivel físico y de enlace, destacan protocolos como Zigbee y Z-Wave, ampliamente adoptados por su eficiencia energética y capacidad de formar redes malladas (*mesh networks*). Zigbee, basado en el estándar IEEE 802.15.4, opera en la banda ISM no regulada de 2.4 GHz y se caracteriza por su alta capacidad de integración con múltiples dispositivos. Z-Wave, por su parte, opera en bandas en torno a 800-900 MHz, lo que proporciona un mejor alcance y menor interferencia, aunque con menor velocidad de transmisión [3].

Más recientemente, el estándar Matter, desarrollado por Connectivity Standards Alliance (CSA), ha ganado protagonismo por su enfoque unificador. Matter busca resolver los problemas de fragmentación en el ecosistema IoT mediante un protocolo común basado en IP, que permita la interoperabilidad entre dispositivos de distintos fabricantes. Matter utiliza protocolos subyacentes como UDP, IPv6 y TLS, y se apoya en tecnologías como Thread y Wi-Fi para el transporte, integrando mecanismos de seguridad desde su diseño [4][5][7].

Por otro lado, el uso de protocolos tradicionales como HTTP y WebSocket también tiene cabida en plataformas domóticas más complejas, especialmente cuando se utilizan interfaces web o comunicaciones bidireccionales entre el usuario y el sistema. WebSocket permite mantener una conexión persistente entre cliente y servidor, optimizando la latencia frente a modelos basados en peticiones individuales como HTTP [4].

La coexistencia de múltiples protocolos en un mismo sistema es habitual, ya que cada uno cumple funciones específicas. Esta diversidad exige una arquitectura flexible y modular que permita integrar distintos estándares sin comprometer la eficiencia o la seguridad. La elección de los protocolos adecuados depende de factores como el tipo de dispositivo, la topología de la red, los requisitos de latencia y la necesidad de interoperabilidad.

La tendencia actual se orienta hacia soluciones basadas en IP que favorecen una mayor escalabilidad, seguridad y compatibilidad entre dispositivos de distintos fabricantes, siendo Matter un claro ejemplo de esta evolución tecnológica [4].

2.3 Consideraciones de seguridad

En el contexto de la automatización del hogar y los entornos IoT, la seguridad representa un pilar fundamental que debe ser abordado desde las primeras fases de diseño del sistema. A medida que se incrementa la conectividad y la complejidad de los dispositivos integrados, también aumentan los vectores de ataque potenciales, por lo que garantizar la confidencialidad, integridad y disponibilidad de los datos y servicios se convierte en una prioridad.

Uno de los principales riesgos asociados a la domótica es el acceso no autorizado a la red local o a dispositivos conectados. La presencia de sensores, actuadores y sistemas de videovigilancia conectados a Internet introduce una superficie de ataque considerable, que puede ser explotada por ciberdelincuentes para obtener información sensible, manipular el funcionamiento del sistema o, incluso, comprometer la seguridad física de los usuarios [4]. Por este motivo, se recomienda implementar medidas de protección perimetral como cortafuegos, sistemas de detección de intrusiones (IDS) y segmentación de redes.

Los protocolos utilizados también deben contemplar mecanismos de autenticación y cifrado. En el caso de MQTT, por ejemplo, la autenticación mediante usuario y contraseña puede verse reforzada con el uso de certificados TLS para cifrar las comunicaciones. Asimismo, la elección del nivel de calidad de servicio afecta la fiabilidad de los mensajes transmitidos y su protección frente a pérdida o duplicación [2][3]. En protocolos como CoAP, el uso de DTLS (Datagram Transport Layer Security) permite implementar cifrado y autenticación de forma ligera para dispositivos con recursos limitados [4].

Por otro lado, las plataformas domóticas como Home Assistant disponen de mecanismos específicos para mejorar la seguridad del sistema. Entre ellos se incluyen el uso de contraseñas seguras, autenticación en dos factores (2FA) y la posibilidad de restringir el acceso a través de listas blancas de direcciones IP. Además, mediante el uso del fichero *secrets.yaml*, es posible aislar credenciales y datos sensibles del archivo principal de configuración, reduciendo así el riesgo en caso de acceso no autorizado [1].

El uso de servicios en la nube, muy común en soluciones comerciales como *Google Nest*, *Alexa* o *HomeKit*, introduce un nuevo conjunto de desafíos relacionados con la protección de los datos. Aunque muchas de estas plataformas implementan estándares avanzados de cifrado, la dependencia de servidores externos implica que parte del control sobre la seguridad escapa al usuario, lo que ha impulsado la búsqueda de alternativas basadas en el control local [4].

En esta línea, los nuevos estándares como Matter han adoptado un enfoque *secure by design*, integrando cifrado extremo a extremo, verificación de identidad entre dispositivos y un proceso de emparejamiento seguro basado en certificados digitales. Matter también impone requisitos estrictos sobre el almacenamiento seguro de claves, el aislamiento de procesos y la actualización segura del firmware, lo que reduce notablemente la probabilidad de explotación de vulnerabilidades conocidas [5][7].

Por último, no debe ignorarse el factor humano como vector de riesgo. Prácticas como la configuración por defecto de contraseñas, la falta de actualizaciones o la exposición de servicios en puertos inseguros son errores comunes que pueden comprometer un sistema por completo. En este sentido, la educación del usuario y el mantenimiento continuo del sistema son elementos clave para preservar un entorno domótico seguro y confiable [4].

3

Capítulo 3 Software

La comunicación entre dispositivos en un sistema domótico depende de protocolos eficientes y seguros. En este trabajo se han empleado dos soluciones ampliamente utilizadas en el ámbito del IoT: MQTT, por su ligereza y simplicidad, y Matter, un estándar emergente que promueve la interoperabilidad local con alta seguridad. Ambas tecnologías permiten el intercambio de información entre sensores, actuadores y plataformas de control, y serán analizadas en detalle a continuación, considerando su arquitectura, funcionamiento y aplicación práctica posteriormente.

3.1 MQTT

El protocolo MQTT (Message Queuing Telemetry Transport) se ha consolidado como uno de los pilares fundamentales en el ecosistema del Internet de las Cosas, destacando por su simplicidad, bajo uso de recursos y eficacia en entornos con conectividad limitada. Fue desarrollado inicialmente por IBM en 1999 para la monitorización de oleoductos mediante conexiones vía satélite, pero ha sido ampliamente adoptado en aplicaciones industriales, domésticas y a través de la nube [2][3].

MQTT se fundamenta en un modelo *publish/subscribe*, que permite una comunicación asíncrona entre múltiples dispositivos sin necesidad de que estén conectados directamente entre sí. Este enfoque está compuesto por tres elementos principales:

- **Publicador (Publisher):** envía mensaje a un tema o *topic*.
- **Suscriptor (Subscriber):** recibe los mensajes publicados en los *topic* a los que se ha suscrito.
- **Bróker:** servidor intermedio que gestiona la distribución de mensajes entre publicadores y suscriptores.

Este modelo permite desacoplar emisores y receptores en tiempo, espacio y sincronización, lo que lo hace ideal para sistemas distribuidos con topologías dinámicas y múltiples nodos [3].

Figura 3.1: Esquema modelo publish/subscribe

Los mensajes se envían a través de *topics*, que son cadenas de texto jerárquicas utilizadas para clasificar los mensajes publicados por los dispositivos. Estos *topics* funcionan como direcciones virtuales que permiten a los dispositivos suscribirse únicamente a los datos relevantes para ellos. Puede tener una estructura del tipo *home/salón/temperatura.* Esta estructura es flexible y puede tener tantos niveles como se necesite. La barra inclinada (/) actúa como delimitador entre niveles. La flexibilidad de este sistema permite que se puedan crear temas específicos por habitación, tipo de sensor o cualquier otra categoría relevante para la arquitectura del sistema.

MQTT admite el uso de comodines para facilitar las suscripciones dinámicas. "+" remplaza exactamente un nivel del topic, por ejemplo, *home/+/temperatura* suscribirá a *home/salón/temperatura* y *home/cocina/temperatura.* "#" reemplaza todos los niveles restantes del topic, por ejemplo, *home/#* suscribirá a cualquier mensaje que comience con *home/*. Esta jerarquía, junto con el uso de comodines, ofrece una gran escalabilidad en arquitecturas distribuidas, donde decenas de dispositivos pueden organizarse y controlarse de manera eficiente.

Se definen diferentes tipos de mensaje que permiten el establecimiento, mantenimiento y cierre de sesiones de comunicación, así como la transmisión de los datos.

Los principales mensajes de control son los siguientes:

- **CONNECT:** se utiliza por el cliente para iniciar una conexión con el broker. Incluye el nombre de usuario y contraseña (si están habilitados), opcionalmente el mensaje LWT permite a los clientes especificar un mensaje que el bróker publicará automáticamente en su nombre si se produce una desconexión inesperada o parámetros como el ID de cliente o el tiempo de espera.

- **CONACK:** es la respuesta del broker al mensaje CONNECT. Indica si la conexión ha sido aceptada o rechazada, y en caso de error, proporciona un código de retorno.

- **PUBLISH:** es el mensaje que transporta el dato o payload, publicado en un topic determinado. Puede tener un QoS que determina el nivel de garantía de entrega.

- **PUBACK / PUBREC /PUBREL /PUBCOMP:** estos cuatro tipos de mensajes se utilizan para la entrega garantizada de mensajes cuando se utiliza QoS 1 o 2.
  - **PUBACK:** confirmación para QoS 1.
  - **PUBREC / PUBREL / PUBCOMP:** intercambio de confirmaciones en QoS 2 para evitar duplicados.

- **SUBSCRIBE:** permite al cliente expresar interés por uno o más topics. Es enviado tras la conexión para comenzar a recibir datos.

- **SUBACK:** respuesta del bróker al mensaje SUBSCRIBE. Confirma si la suscripción ha sido aceptada.

- **UNSUSCRIBE / UNSUBACK:** utilizado para dejar de recibir datos de uno o más topics.

- **PINGREQ / PINGRESP:** se usan como mensaje de latido o *keep alive*. El cliente envía PINGREQ y el bróker responde con PINGRESP, asegurando que la conexión sigue viva.

- **DISCONNECT:** el cliente lo envía para indicar que desea cerrar la sesión de forma voluntaria.

Estos mensajes permiten no solo enviar datos, sino también mantener la comunicación viva, gestionar suscripciones y asegurar fiabilidad de transmisión. MQTT incluye varios mecanismos que permiten mejorar su funcionamiento en escenarios donde la fiabilidad y la eficiencia son críticas. Entre ellos, destacan los distintos tipos de mensajes persistente. Cuando se publica un mensaje con la opción *retained* activada, el bróker guarda el último valor enviado en ese topic.

Así, cualquier cliente que se conecte más tarde y se suscriba a ese topic, recibirá inmediatamente el valor más reciente, sin necesidad de esperar a una nueva publicación [2]. LWT (Last Will and Testament) es una funcionalidad critica para saber si un dispositivo se ha desconectado de forma inesperada. Cuando un cliente se conecta, puede configurar un mensaje que el bróker enviará a un topic concreto si detecta que la conexión se ha cerrado de forma abrupta. Esto es útil para notificar fallos de dispositivos en sistemas domóticos o industriales donde la fiabilidad es clave.

Por último, MQTT permite mantener sesiones persistentes. Si un cliente se desconecta temporalmente, el bróker puede almacenar los mensajes que se publiquen en los topics a los que esta suscrito y enviarlos una vez el cliente se reconecte (dependiendo de la configuración y QoS). Esto garantiza que no se pierdan datos en caso de caídas temporales de la red [3].

Uno de los aspectos clave del protocolo es su soporte para distintos niveles de calidad de servicio, que permite equilibrar la fiabilidad de la entrega de los mensajes frente a los recursos consumidos [2]. Existen tres niveles:

- **QoS 0 (At most once):** los mensajes se envían una única vez, sin confirmación. Es rápido y eficiente, pero no garantiza la entrega.
- **QoS 1 (At least once):** se garantiza que el mensaje será entregado al menos una vez, aunque puede llegar duplicado.
- **QoS 2 (Exactly once):** garantiza que el mensaje se entregará una única vez sin duplicados, aumentando la latencia.

Aunque MQTT no incluye cifrado ni autenticación por defecto, su seguridad se implementa habitualmente mediante diferentes mecanismos como la autenticación mediante usuario y contraseña, incluidos en el paquete de conexión del bróker; cifrado TLS/SSL para proteger la integridad y confidencialidad de los datos; control de acceso en *topics* mediante listas ACL (Access Control Lists), que determinan qué clientes pueden publicar o suscribirse a determinados *topics* [4].

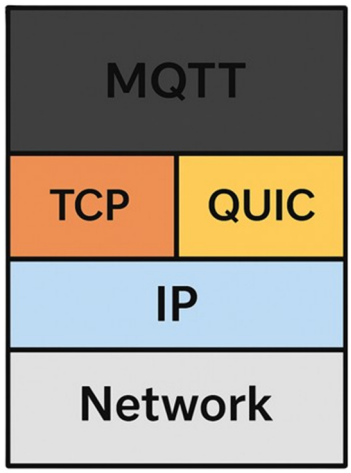

Figura 3.2: Pila de protocolos de MQTT

La Figura 3.2 muestra una representación de la pila de protocolos sobre la que opera MQTT, destacando sus capas fundamentales en el contexto de redes IP. En la parte superior se encuentra MQTT, que actúa como un protocolo de capa de aplicación diseñado para comunicaciones eficientes entre dispositivos IoT. MQTT no depende de un protocolo de transporte específico, aunque tradicionalmente ha utilizado TCP como su principal protocolo subyacente. Sin embargo, versiones más recientes como MQTT 5.0 también permiten su implementación sobre otros protocolos, como QUIC [3].

En cualquier caso, el protocolo de transporte más comúnmente utilizado es TCP (Transmission Control Protocol), que garantiza una comunicación fiable y ordenada entre cliente y bróker, incluyendo mecanismos de retransmisión y control de flujo. Alternativamente, QUIC, un protocolo más moderno desarrollado por Google y estandarizado por IETF, ofrece ventajas en

términos de velocidad y seguridad y combina características de transporte y cifrado en una sola capa. Su integración con MQTT está aún en fases tempranas, pero ya se considera una opción prometedora para entornos con alta latencia o limitaciones de ancho de banda.

En la capa inmediatamente inferior se encuentra el protocolo IP (Internet Protocol), tanto en su versión IPv4 como IPv6, los cuales proporcionan direccionamiento lógico y encaminamiento de paquetes tanto en redes locales como globales.

Finalmente, el nivel más bajo representa la interfaz con las capas inferiores, estas son, la capa de enlace y la capa física, que pueden incluir tecnologías como Ethernet, Wi-Fi, LTE o LoRA, entre otras. Ambas capas son las responsables de la transmisión efectiva de los datos entre dispositivos a través de medios físicos o inalámbricos.

3.2 Matter

Matter es un protocolo de conectividad de código abierto diseñado para garantizar la interoperabilidad, seguridad y fiabilidad en la comunicación entre dispositivos inteligentes dentro del hogar. Impulsado por la CSA (Connectivity Standards Alliance), Matter surge como una iniciativa colaborativa entre grandes actores del sector como Apple, Google, Amazon y la antigua Zigbee Alliance (hoy CSA), con el objetivo de unificar los estándares fragmentados que tradicionalmente han caracterizado al ecosistema domótico [4].

Este estándar fue introducido originalmente en 2010 bajo el nombre de *Project CHIP* (Connected Home over IP) y su desarrollo se ha caracterizado por una fuerte orientación hacia la accesibilidad técnica, la transparencia del código y la eliminación de barreras entre fabricantes. La idea principal es que, independientemente del fabricante o del sistema operativo, los dispositivos certificados por Matter puedan interoperar sin necesidad de gateways propietarios, aplicaciones específicas o protocolos exclusivos [4][5].

Uno de los elementos distintivos de Matter es que no crea un nuevo protocolo de red desde cero, sino que se basa en tecnologías ya existentes y probadas, tales como IPv6, TCP, UDP, BLE, Wi-Fi, Thread y Ethernet. De esta manera, Matter se posiciona como un protocolo de aplicación que opera sobre capas de red bien establecidas, facilitando la adopción masiva por parte de la industria y garantizando la compatibilidad con infraestructuras actuales [5][7].

La arquitectura de Matter está organizada de forma modular y jerárquica, permitiendo encapsular responsabilidades en distintas capas funcionales. Este diseño facilita el mantenimiento, la evolución del estándar y la integración de dispositivos con capacidades limitadas. La pila de protocolos de Matter se basa principalmente en el modelo OSI, adoptando sus principios de separación de capas, pero optimizándolo para el entorno del IoT.

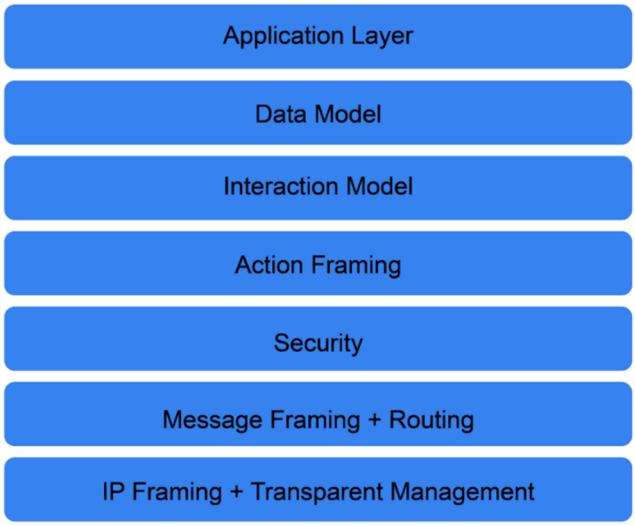

Figura 3.3: Arquitectura de Matter [6]

Entre los elementos clave de esta arquitectura se destacan [5][6]:

- **Capa de aplicación**: esta capa define las funcionalidades que debe cumplir el dispositivo, como encender una luz, regular la temperatura o medir un nivel de humedad. Está estrechamente relacionada con el modelo de datos, que estructura y define cómo se representan estos atributos funcionales.

- **Modelo de datos**: contiene la información estructurada que respalda las funciones de la aplicación. La interacción con el dispositivo se basa en estos datos.

- **Modelo de interacción:** Matter define cinco tipos de interacción entre clientes y servidores: lectura de atributos, escritura de atributos, invocación de comandos, suscripción y eventos. Estas interacciones se transforman en una representación binaria comprimida, mediante el mecanismo conocido como TLV (Type-Length-Value), para reducir el tamaño del mensaje y optimizar la transmisión [7].

- **Enmarcado y transporte:** una vez serializados, los datos se encapsulan en una capa de transporte basada en protocolos IP. Aquí, Matter puede utilizar TCP para garantizar la entrega fiable de los datos o UDP para comunicaciones más ligeras. En ambos casos, se añade información de encabezado que permite dirigir el paquete adecuadamente a su destino.

- **Seguridad:** se implementa cifrado de extremo a extremo y autenticación mutua entre dispositivos. Para ello, utiliza certificados X.509, claves públicas derivadas del modelo PASE/CASE (Password Authenticated Session Establishment / Certificate Authenticated Session Establishment) y el cifrado mediante algoritmos como AES-CCM o HMAC [37]. Estos mecanismos garantizan que los datos no pueden ser interceptados ni manipulados por terceros [7].

- **Red y conectividad:** Matter no define una nueva capa de enlace físico, sino que utiliza tecnologías como Wi-Fi, Ethernet y Thread, dependiendo del tipo de dispositivo y sus requisitos energéticos. La conectividad entre diferentes medios se gestiona mediante *Border Routers*, que permiten la interoperabilidad entre dispositivos Thread y redes IP.

El funcionamiento típico de Matter se basa en una comunicación local, es decir, no necesita de la nube para que los dispositivos interactúen. Esto ofrece beneficios claros en términos de latencia, privacidad y fiabilidad. No obstante, se permite la integración con servicios cloud.

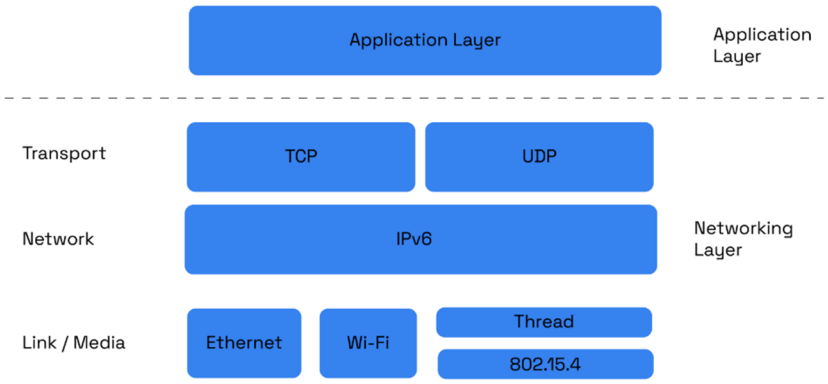

Figura 3.4: Pila de protocolos de Matter [6]

El diseño de Matter se apoya en un conjunto de tecnologías consolidadas que aportan robustez, escalabilidad y eficiencia al sistema:

- **IPv6:** permite asignar una dirección única y global a cada dispositivo, eliminando la necesidad de gateways propietarios. Matter se basa completamente en IP, lo que facilita su integración con redes existentes y su escalabilidad a largo plazo [4][5]. Proporciona una cantidad inmensa de direcciones IP, lo cual es esencial para el crecimiento del IoT.

- **Wi-Fi:** es utilizado por dispositivos con alimentación continua y que requieren transferencia de datos. Su uso permite una integración sencilla con routers convencionales y una cobertura amplia dentro del hogar. En contraposición, no es óptimo para dispositivos alimentados por batería ya que el consumo energético es relativamente alto [5].

- **Thread:** protocolo de malla basado en IPv6 sobre 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), especialmente diseñado para dispositivos de bajo consumo energético. Matter lo utiliza como su solución predilecta para sensores, cerraduras y otros dispositivos con batería. Permite la autorreparación de la red y asegura una baja latencia [4][7].

- **BLE (Bluetooh Low Energy):** utilizado exclusivamente durante el proceso de configuración inicial, permite transferir credenciales de red y escanear el código QR del dispositivo sin necesidad de intervención manual compleja.

- **Ethernet:** en dispositivos estáticos, como hubs o paneles de control, Matter permite utilizar redes cableadas para garantizar mayor estabilidad y rendimiento, ofreciendo una comunicación fiable sin interferencias inalámbricas.

Matter adopta un enfoque abierto y transparente en su desarrollo, permitiendo que tanto miembros como no miembros del grupo de trabajo accedan a su diseño y procesos técnicos. Como parte de este compromiso, su código fuente está disponible en un repositorio de GitHub

bajo la licencia Apache V2, facilitando así su implementación y adopción por parte de la comunidad.

El desarrollo de Matter se basa en la integración de tecnologías probadas en el mercado, consolidándolas en un sistema unificado que optimiza la interoperabilidad entre dispositivos inteligentes. Este enfoque permite acelerar su implementación y extender sus beneficios tanto a fabricantes como a consumidores.

Los principios fundamentales de Matter son los siguientes [6]:

- **Interoperabilidad:** cualquier dispositivo certificado por Matter puede interactuar con cualquier otro, sin importar el fabricante.

- **Seguridad por diseño:** todos los dispositivos deben cumplir estrictos requisitos de cifrado, autenticación y actualización segura.

- **Conectividad local:** aunque puede usarse la nube, está diseñado para funcionar completamente dentro de la red local.

- **Simplicidad de configuración:** se priorizan mecanismos intuitivos como el escaneo de códigos QR, BLE o métodos preconfigurados para facilitar la incorporación de nuevos dispositivos.

4

Capítulo 4 Hardware

El diseño de un sistema domótico eficiente y fiable requiere la selección cuidadosa de componentes de hardware que garanticen conectividad, procesamiento adecuado y compatibilidad con los protocolos de comunicación empleados. En este proyecto se han utilizado los módulos ESP-WROOM-32, su variante ESP-WROOM-32D y la cámara ESP32-CAM, desarrollados por Espressif Systems, debido a su versatilidad y amplio soporte en aplicaciones de IoT.

4.1 Dispositivos utilizados

El núcleo funcional de cualquier sistema domótico basado en IoT lo constituyen los dispositivos de control y adquisición, que deben ser lo suficientemente versátiles, eficientes y económicos como para integrarse sin dificultad en entornos residenciales. En este contexto, las placas basadas en el chip ESP32 han adquirido una gran relevancia gracias a su equilibrio entre prestaciones técnicas, bajo consumo energético y amplia disponibilidad en el mercado. Su diseño las hace especialmente adecuadas para proyectos de automatización, permitiendo tanto la recogida de datos, desde sensores como el control de actuadores a través de múltiples interfaces de comunicación.

Dentro de esta familia, los módulos ESP-WROOM-32 y ESP-WROOM-32D, desarrollados por Espressif Systems, han sido seleccionados como plataformas de hardware para este proyecto por su alta compatibilidad con protocolos estándar, como Wi-Fi y Bluetooth, su facilidad de integración con entornos de desarrollo como Arduino IDE y su excelente relación coste-beneficio [8][9]. Ambos módulos incorporan el microcontrolador ESP32, una unidad de procesamiento dual-core que combina potencia de cálculo con múltiples opciones de conectividad, características que los convierten en componentes esenciales para el despliegue de nodos inteligentes dentro de una red domótica.

Estos módulos no solo permiten la ejecución de tareas de control y monitorización, sino que también facilitan la comunicación directa con brókers MQTT o controladores Matter, sin necesidad de elementos intermedios. Su arquitectura permite ejecutar código nativo para recopilar datos de sensores, ejecutar automatizaciones y enviar notificaciones a través de diversos protocolos, garantizando así una experiencia de usuario fiable y personalizable [5].

Además, la disponibilidad de distintas versiones, como la variante 32D que presenta diferencias en el diseño de la antena, proporciona al desarrollador una mayor flexibilidad para adaptarse a los requisitos físicos y ambientales del entorno donde se desplegará el sistema. Por todo ello, las placas ESP32 constituyen una solución sólida, ampliamente documentada y soportada por una comunidad activa, lo que garantiza su aplicabilidad.

4.1.1 ESP-WROOM-32

El módulo ESP-WROOM-32 es una placa basada en el microcontrolador ESP32-D0WDQ6. Este chip cuenta con un procesador de doble núcleo Xtensa® de 32 bits, con una frecuencia ajustable entre 80 MHz y 240 MHz. Además, incorpora una memoria flash SPI de 4 MB y una antena PCB integrada, lo que facilita su implementación en diseños compactos. Se trata de una solución completa e independiente que Wi-Fi 802.11 b/g/n, Bluetooth v4.2 BR/EDR y BLE (Bluetooth Low Energy), lo que permite operar en múltiples escenarios de conectividad [8]. Cuenta con soporte para modos de bajo consumo, incluyendo deep-sleep, con un consumo inferior a 5 µA.

Entre sus principales características técnicas se encuentran [8]:

- **CPU:** dual-core Tensilica LX6 a 240 MHz.
- **Memoria:** 520 KB de SRAM y 4 MB de Flash externa.
- **Conectividad:** Wi-Fi y Bluetooth integrados.
- **GPIO:** hasta 34 pines programables de entrada/salida.

- **Interfaz:** UART, SPI, I2C, I2S, PWM, ADC y DAC.
- **Tensión de operación:** 3.3 V.

Su versatilidad lo hace adecuado para entornos donde se requieren comunicaciones inalámbricas rápidas y estables, como la publicación de datos mediante MQTT o la integración con plataformas como HA a través de ESPHome. Además, puede ejecutarse sin necesidad de un sistema operativo, utilizando entornos de desarrollo ligeros como IDE Arduino que facilita su programación.

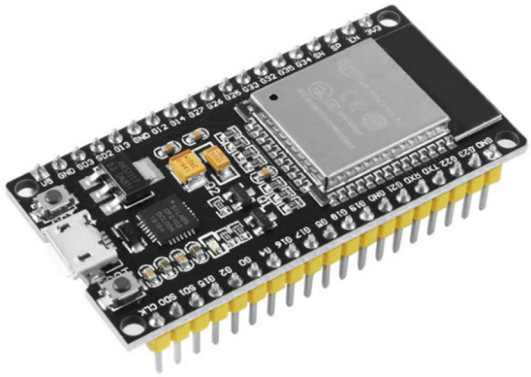

Figura 4.1: Estructura física del ESP-WROOM-32

4.1.2 ESP-WROOM-32D

El módulo ESP-WROOM-32D es una evolución del anterior, basado en el mismo controlador ESP32-D0WD, pero optimizado para entornos donde la estabilidad de señal es crítica. La principal diferencia radica en el diseño de la antena: esta versión incorpora una antena tipo PCB en lugar de antena integrada o conectores externos, lo que permiten una mejor propagación de la señal en entornos cerrados o con interferencias [9].

Figura 4.2: Estructura física del ESP-WROOM-32D

Sus características destacadas son las siguientes [9]:

- **CPU:** ESP32-D0WD, dual-core
- **Memoria:** SRAM de 520 KB y Flash de 4 MB.
- **Antena:** PCB integrada, diseñada para optimizar el rendimiento RF.
- **Certificaciones:** cumple las normal CE, FCC y otras regulaciones globales.
- **Tamaño compacto:** ideal para integrarse en dispositivos embebidos o proyectos con restricciones físicas.

El módulo ESP-WROOM-32D mantiene todas las funcionalidades del WROOM-32 original, pero su rendimiento RF mejorado lo convierte en una opción preferente en aplicaciones criticas como el control de luces, cámaras o sensores a través del estándar Matter, donde la calidad del enlace inalámbrico es un factor decisivo para asegurar la comunicación local entre dispositivos [9].

4.1.3 ESP32-CAM

El módulo ESP32-CAM representa una de las variantes más versátiles del microcontrolador ESP32, al integrar conectividad Wi-Fi y Bluetooth junto con una cámara OV2640 y ranura para tarjeta microSD en un formato compacto. Esta integración lo convierte en una solución idónea para sistemas de videovigilancia, monitorización de espacios y aplicaciones relacionadas con visión artificial dentro de entornos domóticos.

A nivel técnico, este módulo cuenta con un procesador de doble núcleo Tensilica LX6 que opera a 240 MHz, 520 KB de SRAM y una memoria flash de 4 MB. La cámara incorporada, el modelo OV2640, ofrece una resolución de hasta 2 megapíxeles (1600×1200), siendo suficiente para tareas básicas de reconocimiento visual y supervisión. El módulo también dispone de varios pines GPIO, aunque su uso se encuentra limitado debido a la ocupación de algunos de ellos por funciones internas, como el manejo de la cámara o la tarjeta microSD.

Figura 4.3: Estructura física del ESP32-CAM

En el contexto de HA, el ESP32-CAM ha sido integrado utilizando el firmware ESPHome, el cual permite configurar y controlar el módulo a través de YAML. Además, mediante la utilización de automatizaciones y notificaciones por Telegram, se ha implementado un sistema que captura y envía imágenes ante la detección de eventos específicos, como movimientos o accesos no autorizados.

4.1.4 Sensor KY-015

El sensor KY-015 es un módulo que permite la medición conjunta de temperatura y humedad ambiental mediante el uso de un componente DHT11. Este sensor digital se basa en un termistor de tipo NTC para detectar la temperatura, y un sensor capacitivo para la humedad relativa, proporcionando datos discretos en intervalos regulares.

Las especificaciones técnicas del sensor incluyen un rango de temperatura de 0 °C a 50 °C con una precisión de ±2 °C, y un rango de humedad de 20% a 90% con una precisión de ±5%. Aunque estas cifras lo sitúan por debajo de sensores más avanzados como el DHT22, su bajo coste y facilidad de uso lo convierten en una opción adecuada para proyectos educativos o de automatización básica.

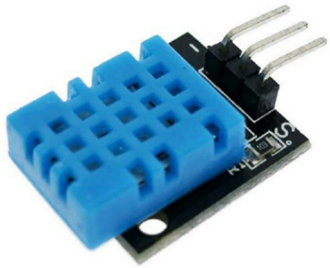

Figura 4.4: Sensor de temperatura y humedad KY-015

4.2 Programación de los dispositivos

La programación de los dispositivos constituye una etapa fundamental en el desarrollo de sistemas domóticos, ya que permite definir el comportamiento, la lógica de control y los mecanismos de comunicación de los distintos nodos que forman parte de la infraestructura inteligente. En función del protocolo utilizado, los dispositivos requieren un entorno de desarrollo, una configuración de red y una codificación específica para establecer correctamente el flujo de información con la plataforma central o entre dispositivos.

En este contexto, se han implementado dos enfoques diferenciados en función del estándar de comunicación utilizado, por un lado, el protocolo MQTT, ampliamente adoptado por su ligereza y eficacia en redes locales, cuya programación se ha llevado a cado utilizando el entorno Arduino IDE; y por otro, el estándar Matter, más reciente y con una arquitectura más compleja, cuyo despliegue se ha realizado a través de herramientas especificas provistas por el fabricante.

Cada enfoque presenta particularidades en cuanto a la estructura del código, el proceso de configuración y los recursos necesarios, por lo que en los apartados siguientes se describe de forma detallada la programación empleada para cada protocolo, junto con las herramientas y procedimientos necesarios para su implementación.

4.2.1 Programación MQTT

La implementación del protocolo MQTT en dispositivos como el ESP32 puede realizarse a través de diferentes enfoques, en función del entorno de desarrollo utilizado y del nivel de control requerido sobre el dispositivo. Las dos opciones más comunes son el uso de Arduino IDE, que permite una programación directa y personalizada en lenguaje C++; y la integración mediante ESPHome dentro de la plataforma Home Assistant, que ofrece una configuración más simplificada basada en ficheros YAML.

La elección de uno u otro método depende en gran medida del grado de flexibilidad deseado, así como del tipo de aplicación. Mientras que ESPHome permite una rápida configuración sin necesidad de conocimientos avanzados de programación, el uso de Arduino IDE ofrece un mayor control sobre el comportamiento del dispositivo, permitiendo el uso de bibliotecas específicas.

En este trabajo se ha optado por la vía de Arduino IDE, debido a que permite una interacción más directa con las funciones del sensor. Esta opción ha sido especialmente útil para comprender en detalle el funcionamiento del protocolo MQTT y su estructura de publicación/suscripción, así como para validar la conexión y transmisión de datos entre el ESP32 y el bróker MQTT alojado en HA.

El primer paso consiste en asegurar que la placa ESP32 esté correctamente añadida al entorno de Arduino. Para ello, se accede a la sección de *Preferencias* del software y se incorpora la siguiente URL correspondiente al gestor de tarjetas, lo que permitirá posteriormente instalar los recursos necesarios para esta familia de microcontroladores.

https://dl.espressif.com/dl/package_esp32_index.json

A través del *Gestor de placas*, se instala el soporte para ESP32, seleccionando posteriormente el modelo exacto con el que se va a trabajar desde el menú de selección de placa. En este caso, se ha empleado el modelo *ESP32 Dev Module*, compatible con variantes como ESP32-WROOM-32 y ESP32-WROOM-32D.

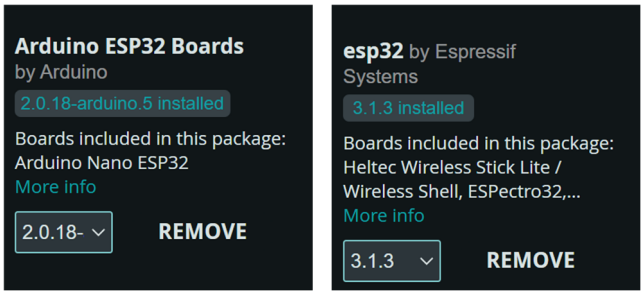

Figura 4.5: Módulos instalados para la gestión de la placas ESP32

Una vez seleccionada la placa, se procede a instalar las librerías necesarias que permitirán la comunicación mediante el protocolo MQTT y el manejo de sensores genéricos. Estas librerías, como *PubSubClient* para MQTT y aquellas específicas para sensores, por ejemplo, los basados en DHT o similares que se utilizarán posteriormente, pueden ser añadidas desde el *Gestor de bibliotecas* de Arduino.

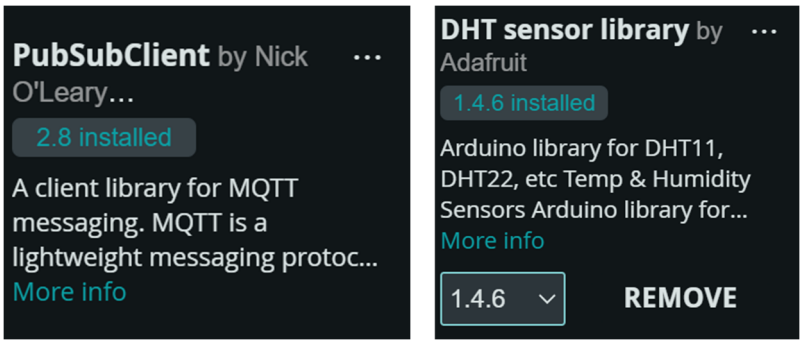

Figura 4.6: Librerías instaladas

La siguiente etapa implica la conexión física del sensor al ESP32, considerando los pines disponibles en el microcontrolador y las características del sensor empleado. La elección del pin adecuado es fundamental para asegurar la lectura correcta de datos. Una vez realizada la conexión, se desarrolla el código fuente que permitirá al microcontrolador conectarse a una red Wi-Fi, establecer comunicación con el bróker MQTT y publicar los datos procedentes del sensor en uno o varios topics definidos. Este código es posteriormente cargado en la placa mediante el puerto USB, asegurando la correcta configuración del puerto serie y la velocidad de transmisión, en este caso de 115200 baudios.

Una vez instalado el entorno de desarrollo y configurado las bibliotecas necesarias, el dispositivo puede ser programado para que actúe como un cliente MQTT. En el ejemplo implementado, se ha utilizado un sensor DHT11 conectado a una placa ESP32, cuya función consiste en leer la temperatura y la humedad, y posteriormente publicar los valores obtenidos mediante mensajes MQTT.

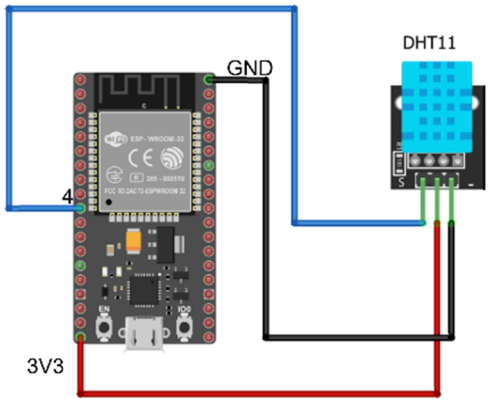

Figura 4.7: Esquema de conexión del ESP32 y el sensor

A continuación, se muestra el código empleado que será comentado:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// Configuración WiFi
const char* ssid = "wifi_name";
const char* password = "wifi_pass";

// Configuración MQTT
const char* mqtt_server = "192.168.1.110";
const char* mqtt_user = " usuario_mqtt ";
const char* mqtt_password = "contraseña_mqtt";
const char* temp_topic = "esp32/temp";
const char* hum_topic = "esp32/hum";
const char* output_topic = "esp32/output";

// Configuración del sensor DHT11
#define DHTPIN 4   // Pin donde está conectado el DHT11
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// LED Pin
const int ledPin = 2;  // GPIO 2 en ESP32

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;

void setup() {
    Serial.begin(115200);
    dht.begin();
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);

    pinMode(ledPin, OUTPUT);
}

void setup_wifi() {
    Serial.print("Conectando a ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nWiFi conectado. IP:"+ WiFi.localIP().toString());
}

void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Mensaje recibido en el topic: ");
    Serial.print(topic);
    Serial.print(". Mensaje: ");

    String messageTemp;
    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();

    if (String(topic) == output_topic) {
        Serial.print("Cambiando salida a ");
        if (messageTemp == "on") {
```

```
            Serial.println("ON");
            digitalWrite(ledPin, HIGH);
        } else if (messageTemp == "off") {
            Serial.println("OFF");
            digitalWrite(ledPin, LOW);
        }
    }
}

void reconnect() {
    while (!client.connected()) {
        Serial.print("Intentando conexión MQTT...");
        if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
            Serial.println("Conectado");
            client.subscribe(output_topic);
        } else {
            Serial.print("Fallo, rc=");
            Serial.print(client.state());
            Serial.println(". Reintentando en 5 segundos...");
            delay(5000);
        }
    }
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    long now = millis();
    if (now - lastMsg > 5000) {
        lastMsg = now;

        float temperature = dht.readTemperature();
        float humidity = dht.readHumidity();

        if (!isnan(temperature) && !isnan(humidity)) {
            char tempString[50];
            char humString[50];

            snprintf(tempString, 50, "{\"temperature\": %4.1f}", temperature);
            snprintf(humString, 50, "{\"humidity\": %4.1f}", humidity);

            Serial.print("Temperatura: ");
            Serial.println(tempString);
            client.publish(temp_topic, tempString);

            Serial.print("Humedad: ");
            Serial.println(humString);
            client.publish(hum_topic, humString);
        } else {
            Serial.println("Error leyendo el sensor DHT");
        }
    }
}
```

El código se estructura en varias secciones. En primer lugar, se realiza la inclusión de las bibliotecas necesarias: *WiFi.h* para la conexión inalámbrica, *PubSubCliente.h* para gestionar la comunicación MQTT y *DHT.h* para interactuar con el sensor de temperatura y humedad.

Seguidamente, se definen las credenciales de red Wi-Fi y los parámetros del bróker MQTT, incluyendo la dirección IP del servidor, el puerto (1883), y las credenciales de autenticación. También se definen los topics utilizados para la publicación de los datos, en este caso

relacionados con la temperatura y humedad, y sus respectivos *topics*: *esp32/temp* y *esp32/hum* y la recepción de datos: *esp32/output.*

La función *setup()* inicializa el puerto serie para su monitorización, el sensor DHT, la conexión Wi-Fi y configura el cliente MQTT. El dispositivo permanece conectado en todo momento al *bróker* mediante la función *loop(),* y si se detecta una desconexión la función *reconnect()* se encarga de restablecerla automáticamente.

Los datos recogidos por el sensor se procesan cada cinco segundos. Si la lectura es válida, se convierte a una cadena JSON y se publica en los topics correspondientes mediante *client.publish().* Este formato facilita su integración con HA, que puede interpretar directamente los valores en formato *value_json.* Además, el dispositivo también puede recibir mensajes a través del *topic esp32/output*. Estos mensajes son gestionados en la función *callback().*

Este enfoque permite disponer de un sistema bidireccional donde el sensor actúa como publicador de datos y al mismo tiempo como receptor de comandos, todo ello a través del protocolo MQTT.

Previo a la configuración de los sensores, es imprescindible habilitar el bróker MQTT en HA. Para ello, se instala el complemento *Mosquito broker* desde la sección de *Add-ons,* disponible en el gestor de complementos del sistema. Este bróker actuará como intermediario entre los dispositivos que publican datos y HA, que se encargará de suscribirse a los temas correspondientes.

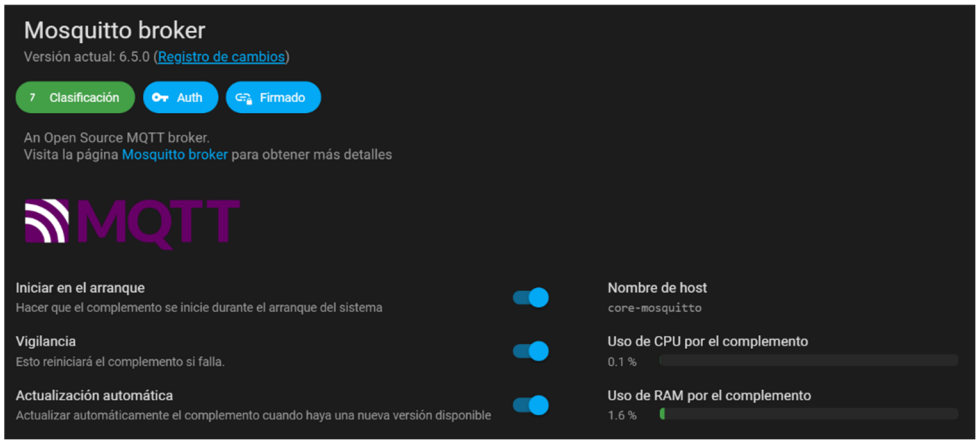

Figura 4.8: Complemento Mosquito bróker añadido

Una vez instalado el complemento, debe modificarse el fichero de configuración de HA para incluir la configuración básica del bróker. En esta configuración se define la dirección IP local del sistema, el puerto por defecto (1883), así como las credenciales de acceso que se utilizarán posteriormente en los dispositivos que se conecten al bróker. La estructura que se debe añadir es la siguiente:

```
mqtt:
  broker: 192.168.1.110  # IP de HA en la red local
  port: 1883
  username: "usuario_mqtt"
  password: "contraseña_mqtt"
  discovery: true
  discovery_prefix: homeassistant
```

Los campos *username* y *password* hacen referencia a las credenciales que deberán utilizarse en la programación de los dispositivos clientes para poder autenticarse correctamente ante el broker. Tras guardar los cambios y reiniciar Home Assistant, se procede a iniciar el complemento *Mosquitto* desde su panel de control. En los registros (logs) del *add-on* se puede verificar que el servicio se ha iniciado correctamente y que está a la espera de nuevas conexiones.

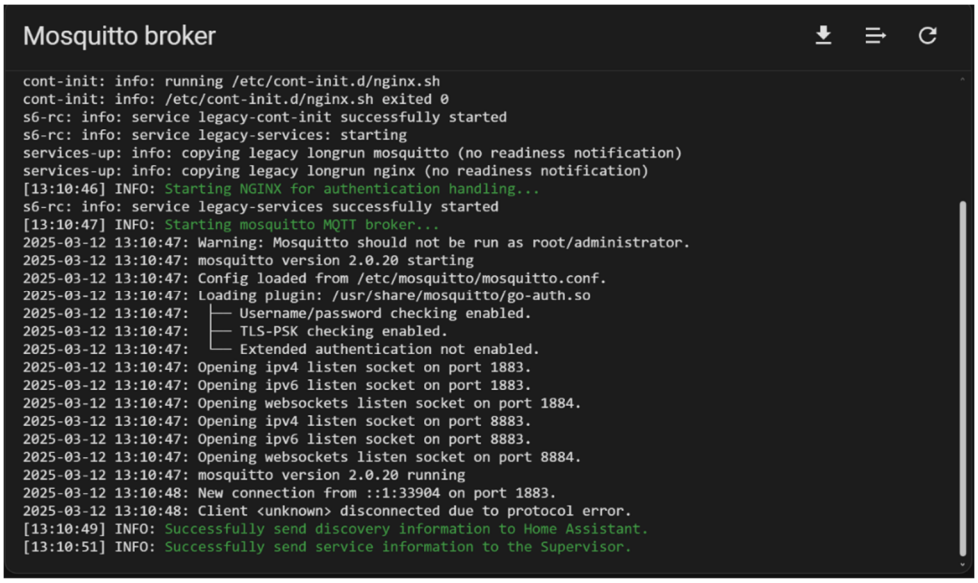

Figura 4.9: Verificación del correcto funcionamiento del bróker

Tras la carga del programa, se procede a la configuración de HA. Para ello, se modifica el fichero *configuration.yaml*, incorporando la definición de los sensores y los topics utilizados. De este modo, HA será capaz de suscribirse a los temas correspondientes y recibir la información publicada por el ESP32.

Realizando una configuración típica, se incluirían entradas para sensores genéricos, donde se especifica el *topic*, el nombre del sensor, su identificador único y una plantilla para interpretar el valor recibido:

```yaml
mqtt:
  sensor:
    - state_topic: "esp32/sensor1"
      name: "Sensor Genérico 1"
      unique_id: sensor1_esp32
      value_template: "{{ value_json.valor1 }}"
    - state_topic: "esp32/sensor2"
      name: "Sensor Genérico 2"
      unique_id: sensor2_esp32
      value_template: "{{ value_json.valor2 }}"
```

Una vez reiniciado HA, las nuevas entidades aparecerán dentro del apartado *Dispositivos y Servicios > MQTT*, y podrán añadirse al panel principal para visualizar sus valores en tiempo real. Además de la configuración y programación del dispositivo, es posible verificar la conectividad y el correcto envío de datos a través de los *logs* del complemento *Mosquitto broker.* Estos *logs* pueden consultarse desde la propia interfaz de HA una vez iniciado el servicio, y permiten comprobar si el cliente MQTT, en este caso el ESP32, ha logrado establecer la conexión correctamente.

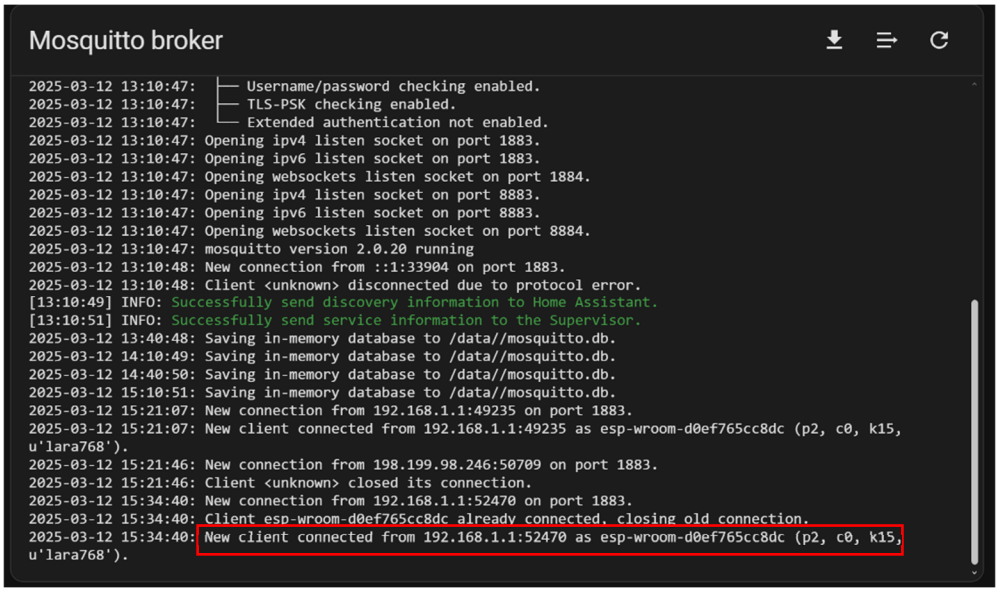

Figura 4.10: Logs del bróker donde se puede ver la conexión del ESP32

Adicionalmente, se puede utilizar el comando *mosquitto_sub* para suscribirse a los mensajes enviados al bróker y visualizar en tiempo real toda la información publicada por los dispositivos conectados. Este procedimiento requiere acceso por SSH a la máquina que ejecuta HA o a un terminal con acceso al bróker. El comando utilizado es el siguiente:

```
mosquitto_sub -h 192.168.1.110 -t "#" -u "usuario_mqtt" -P "contraseña_mqtt"
```

donde:

- ***mosquito_sub*** es la utilidad cliente utiliza para recibir mensajes desde un bróker MQTT
- **-h 192.168.1.110** indica la dirección IP del bróker, que en este caso corresponde a la máquina donde se encuentra instalado HA.
- **-t "#"** permite suscribirse a todos los temas disponibles.
- **-u** y **-P** permiten especificar el nombre de usuario y contraseña configurados en el bróker para autenticar la conexión.

Una vez ejecutado, este comando mostrará en tiempo real los mensajes publicados por el ESP32, lo que permite verificar que los valores de los sensores están siendo enviados de forma correcta y en el formato esperado. Tras completar la configuración tanto en el ESP32 como en Home Assistant, se ha podido comprobar cómo los valores medidos se reflejan correctamente en el dashboard, demostrando que la comunicación entre dispositivos mediante MQTT se ha establecido con éxito.

Figura 4.11: Comprobación de datos mediante conexión SSH.

4.2.2 Programación Matter

El proceso de implementación de dispositivo compatibles con Matter en placas ESP32 se ha visto ampliamente facilitado gracias a herramientas oficiales como *ESP Launchpad*, una plataforma desarrollada por Espressif que permite la instalación y configuración del firmware de Matter de forma intuitiva y accesible. Esta herramienta permite flashear directamente el firmware desde un navegador web, evitando así la necesidad de configurar entornos de desarrollo más complejos como ESP-IDF o PlatformIO.

Antes de comenzar, es necesario identificar correctamente el modelo especifico de placa ESP32 que se va a utilizar, ya que esta elección determinará la compatibilidad y el funcionamiento adecuado del firmware. Una vez Identificada la placa conectada mediante USB, se procede a borrar la memoria del dispositivo. Esta operación es recomendable para asegurar que no existan residuos de configuraciones anteriores que puedan interferir con la instalación del nuevo firmware.

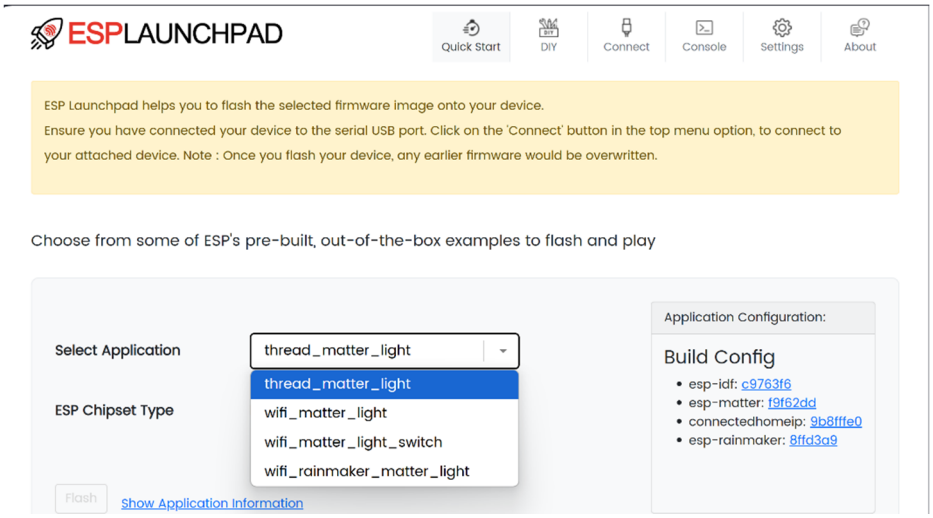

Figura 4.12: Interfaz principal ESP LaunchPad

Posteriormente, desde la interfaz de ESP Launchpad, se selecciona la aplicación de ejemplo basada en Matter que se desea desplegar. Dicha aplicación se puede comunicar vía Wi-Fi, siendo ésta la opción más común para dispositivos que disponen de alimentación continua. Una vez seleccionada, se procede al flasheo del firmware, proceso que finaliza con la visualización de un mensaje de confirmación y la generación automática de un código QR. Este código será necesario durante la puesta en servicio del dispositivo, ya que permite su vinculación directa con plataformas compatibles mediante escaneo.

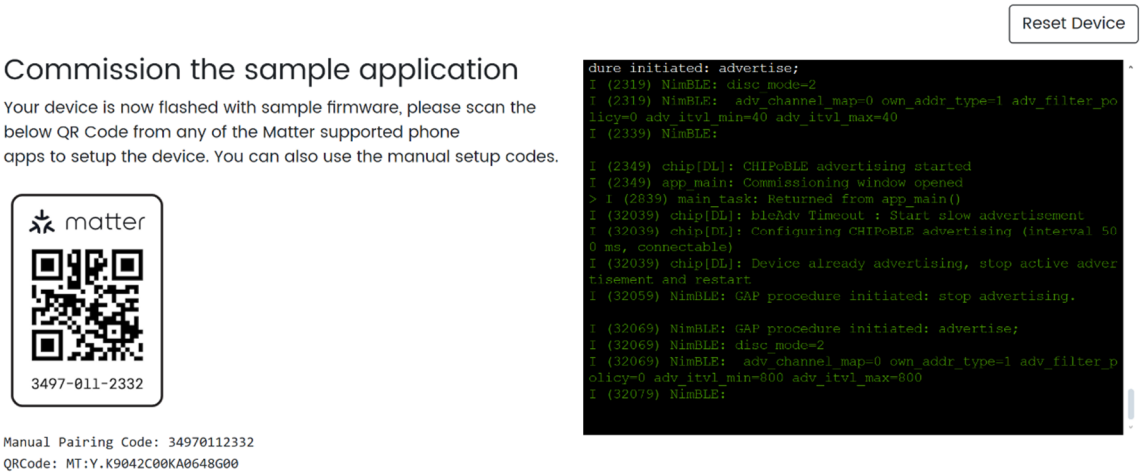

Figura 4.13: Código QR generado tras el flasheo del firmware.

Una vez reiniciado el dispositivo, se encuentra listo para la fase de comisionado, donde se integra al ecosistema domótico mediante aplicaciones que soporten Matter. En este trabajo, se ha utilizado HA, que dispone de soporte nativo para la incorporación de dispositivos Matter. Para ello, y de forma práctica, se emplea la aplicación móvil oficial de Home Assistant, disponible para sistemas Android e iOS. Para ello, se accede a la ruta *Configuración > Dispositivos y Servicios > Añadir Dispositivo > Añadir dispositivo Matter*. Durante el proceso, se solicita escanear el código QR generado previamente, lo que habilita la comunicación entre el dispositivo físico y la plataforma.

El uso de la app móvil de HA simplifica el proceso frente a métodos manuales, como comandos CLI, permitiendo que el proceso se realice desde cualquier lugar sin necesidad de un ordenador.

Durante el proceso de incorporación de un dispositivo Matter mediante la aplicación de Home Assistant, el escaneo del código QR desempeña un papel fundamental en el establecimiento de una sesión segura entre el controlador (en este caso, HA) y el dispositivo comisionado. Este código QR contiene una serie de parámetros críticos definidos por el estándar Matter, tales como el *discriminator*, el *setup PIN code*, así como los identificadores de fabricante y producto. Estos valores no solo identifican al dispositivo, sino que también sirven para iniciar el establecimiento de una sesión cifrada.

El proceso comienza con la apertura de un canal de comunicación utilizando Bluetooth BLE, que actúa como tecnología temporal para transferir las credenciales de red (por ejemplo, el nombre de la red inalámbrica y la contraseña Wi-Fi) al dispositivo. A continuación, se inicia el procedimiento de PASE (Password Authenticated Session Establishment), mediante el cual ambas partes generan una clave compartida derivada del PIN introducido, empleando algoritmos criptográficos seguros. Esta clave permite establecer una sesión cifrada punto a punto, que asegura tanto la confidencialidad como la autenticación mutua [6].

Una vez validado el dispositivo y transferidas las credenciales de red, el nodo se conecta a la red IP local mediante Wi-Fi o Thread, en función de su configuración. Posteriormente, se completa el proceso de comisionado y el controlador (HA) puede descubrir los servicios y clústeres disponibles en el dispositivo a través de protocolos como mDNS y el modelo de datos de Matter. Esto habilita la representación del dispositivo como una entidad funcional dentro del entorno domótico, completamente integrada y lista para ser controlada, automatizada o monitorizada.

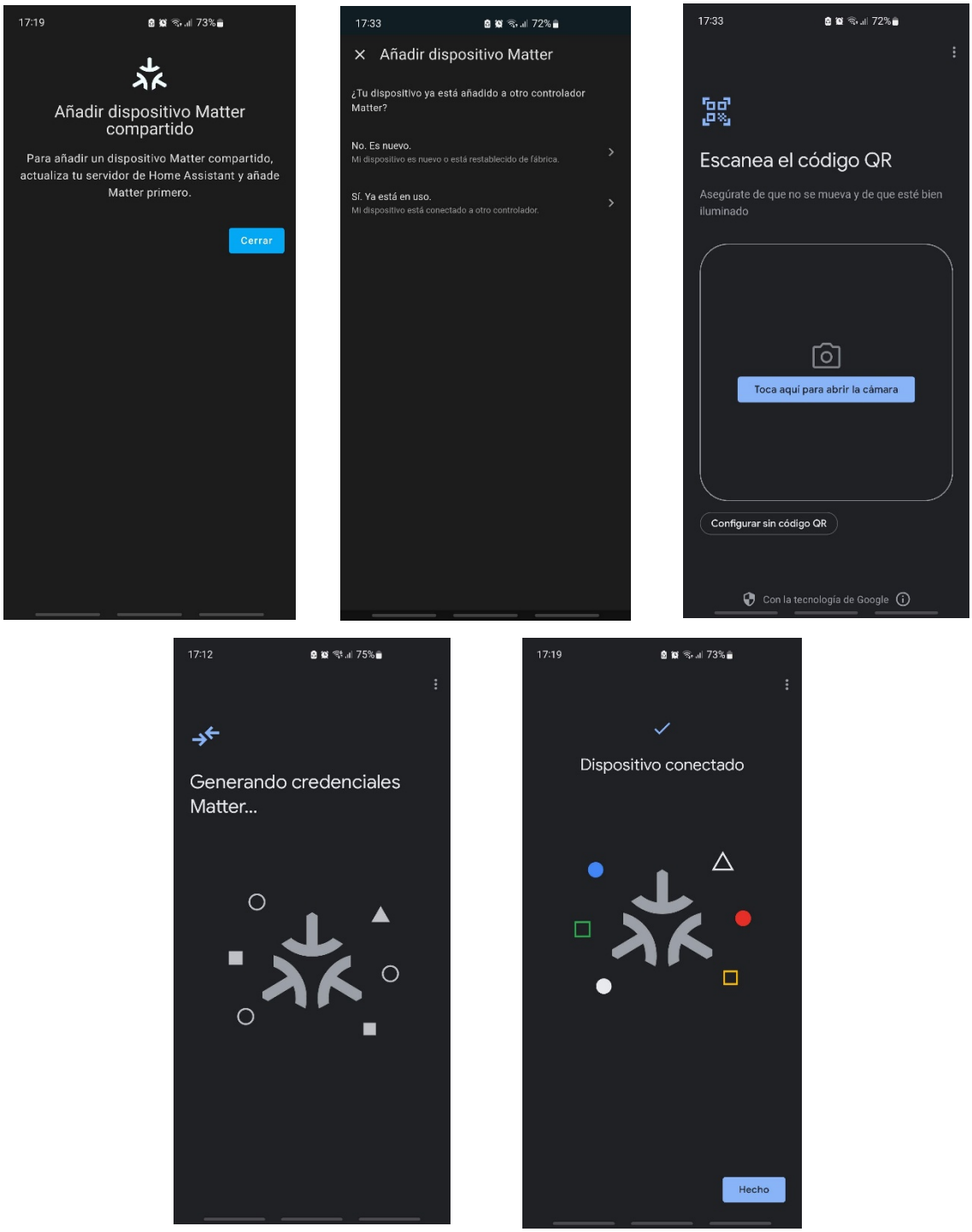

Figura 4.14: Proceso de incorporación de un dispositivo Matter en la app de HA

Una vez finalizado el proceso de establecimiento de la sesión segura y transferidas las credenciales de red, se procede a la fase de confirmación del emparejamiento, también denominada *attestation*. Esta etapa es crítica para validar que el dispositivo que se ha conectado a la red es auténtico y cumple con las especificaciones definidas por el estándar Matter.

Durante este proceso, el dispositivo comisionado envía un certificado digital firmado por el fabricante, conocido como DAC (Device Attestation Certificate). Este certificado contiene información sobre el modelo, número de serie y las claves públicas del dispositivo. El controlador, en este caso HA, utiliza esta información para verificar la identidad del dispositivo mediante una cadena de confianza, que se apoya en una RoT (Root of Trust) almacenada localmente o proporcionadas por la CSA [6].

Si la validación del certificado es exitosa, se procede con la confirmación definitiva de la incorporación del dispositivo a la red Matter. En este punto, se establece una relación de confianza entre el dispositivo y el contralor, y se registran las claves necesarias para futuras sesiones seguras. Además, el controlador consulta el modelo de datos del dispositivo para descubrir las funcionalidades disponibles (clústeres, atributos y comandos).

Una vez completado este procedimiento, el dispositivo queda registrado e integrado en HA, siendo posible su visualización, supervisión y control desde el panel principal dashboard. Ese método de implementación permite una incorporación rápida, segura y eficiente de dispositivos Matter en entornos domésticos y representa una solución altamente escalable para proyectos de automatización basados en estándares abiertos y conectividad local.

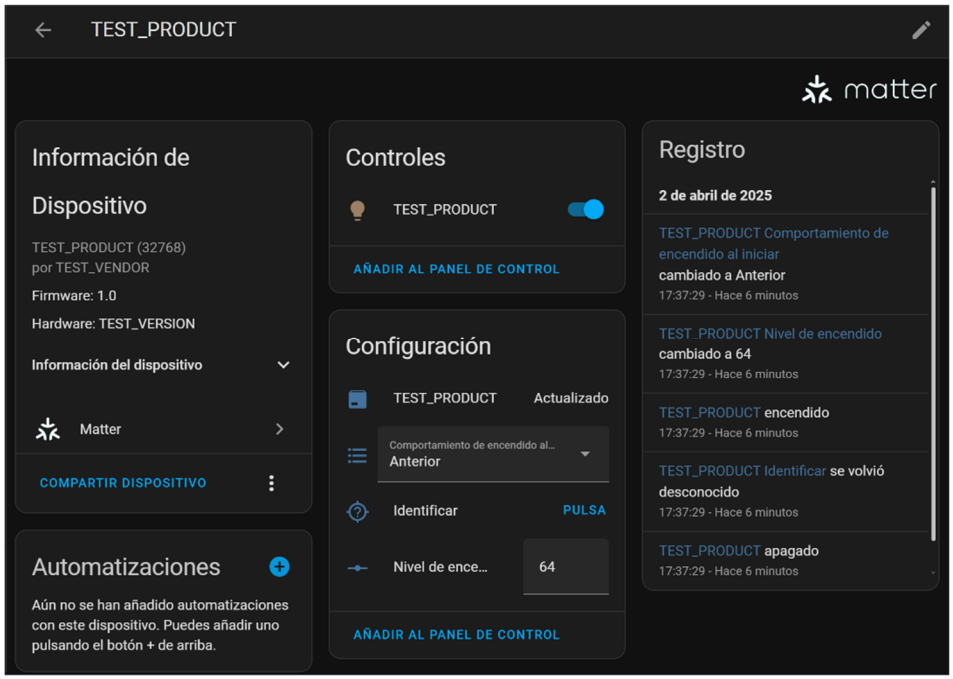

Figura 4.15: Información genérica proporcionada por HA sobre el dispositivo añadido

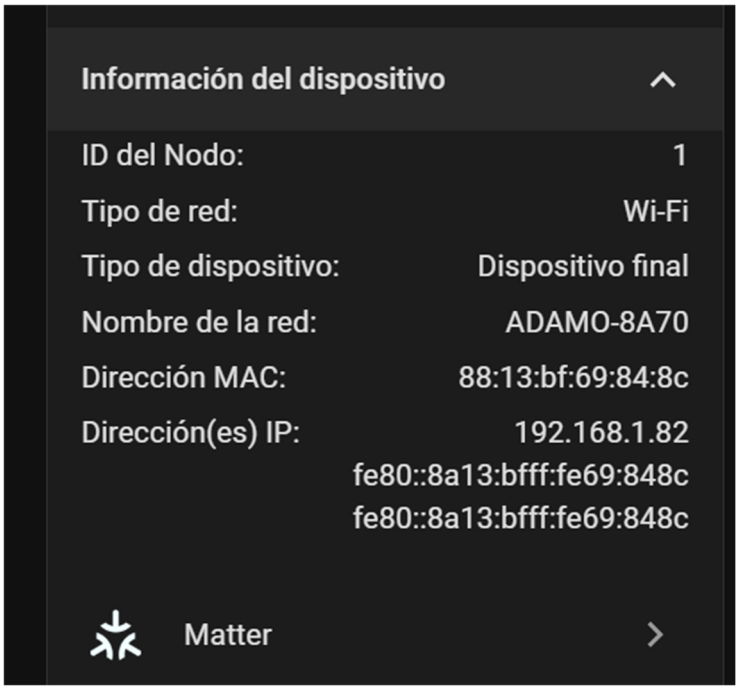

Figura 4.16: Información de red del dispositivo Matter emparejado en HA

En la Figura 4.16 se muestra la información detallada de un dispositivo integrado mediante el protocolo Matter en Home Assistant. Se puede observar que se trata de un *dispositivo final* conectado por red Wi-Fi, cuyo identificador de nodo es 1. También se incluye su dirección MAC y sus direcciones IP, tanto en formato IPv4 (192.168.1.82) como en IPv6 (fe80::8a13:bfff:fe69:848c), característica habitual de los dispositivos Matter. La presencia de ambas direcciones demuestra la compatibilidad con el protocolo IP sobre el que se basa Matter, así como su interoperabilidad dentro de la red local.

La aparición de una dirección IPv6 link-local es especialmente significativa, ya que Matter se basa en IPv6 como pilar fundamental de su arquitectura [4]. A diferencia de otros protocolos domóticos que requieren gateways o conversiones, Matter utiliza una pila IP completa, lo que permite que los dispositivos se comuniquen directamente en la red sin intermediarios. En concreto, la dirección fe80::/10 corresponde al espacio de direcciones link-local, que se utilizan para la comunicación entre dispositivos en la misma red local sin necesidad de un router.

4.2.3 Programación ESP32-CAM

Para incorporar la cámara al sistema domótico, se procedió a añadir un componente denominado *CAM* dentro del panel de ESPHome. Durante el asistente de creación se seleccionó como tipo de dispositivo la opción genérica ESP32, paso que resulta idéntico en cualquier instalación, variando únicamente el modelo concreto de placa empleado.

Uno de los aspectos críticos de la configuración radica en la declaración correcta de los pines que la cámara utiliza; dicha asignación se extrae de la documentación oficial de ESPHome y garantiza que el sensor OV2640 quede inicializado de forma adecuada. Aunque en este caso no se conectaron sensores adicionales al mismo módulo, se dejaron definidas secciones vacías en el archivo YAML para facilitar futuras ampliaciones.

Una vez compilado y flasheado el firmware, el dispositivo quedó visible en el panel de ESPHome con el estado ONLINE, lo que confirmó la correcta conexión a la red y la disponibilidad de la API. En relación con la configuración de la cámara, el archivo YAML permite ajustar parámetros tales como resolución, calidad JPEG, brillo, contraste y autoenfoque. A modo de ejemplo, el bloque principal correspondiente a la cámara quedó definido del siguiente modo (modelo *AI-Thinker*).

```yaml
esphome:
  name: esp-cam
  friendly_name: ESP-CAM

esp32:
  board: esp32dev
  framework:
    type: arduino

# Enable logging
logger:

# Enable Home Assistant API
api:
  encryption:
    key: "SUEk/0W89x36MwhxdOepSJrBjnplT5Qmh2Q76MJczN4="

ota:
  - platform: esphome
    password: "b25be16129d2c74c85a8e51108b1003d"

wifi:
  ssid: !secret wifi_ssid
  password: !secret wifi_password

  manual_ip:
    static_ip: 192.168.1.52   # Cambia esto por la IP deseada para tu ESP32-CAM
    gateway: 192.168.1.1       # Cambia esto por la IP de tu router
    subnet: 255.255.255.0      # La máscara de subred, usualmente 255.255.255.0

  # Enable fallback hotspot (captive portal) in case wifi connection fails
  ap:
    ssid: "Esp-Cam Fallback Hotspot"
    password: "sR1xHwtUgl1t"

captive_portal:

esp32_camera:
  external_clock:
    pin: GPIO0
```

```
  frequency: 20MHz
i2c_pins:
  sda: GPIO26
  scl: GPIO27
data_pins: [GPIO5, GPIO18, GPIO19, GPIO21, GPIO36, GPIO39, GPIO34, GPIO35]
vsync_pin: GPIO25
href_pin: GPIO23
pixel_clock_pin: GPIO22
power_down_pin: GPIO32

# Image settings
name: My Camera
```

En dicho bloque se estableció una IP estática, con dirección IP 192.168.1.52 mediante la opción *manual_ip*, lo que garantiza que Home Assistant localice siempre la cámara en la misma dirección sin depender de asignaciones DHCP dinámicas. Tras la primera conexión, se añadió al *dashboard* de Home Assistant la tarjeta de cámara, permitiendo la visualización en directo de las imágenes captadas. Con esta integración, el módulo ESP32-CAM quedó plenamente operativo dentro del ecosistema.

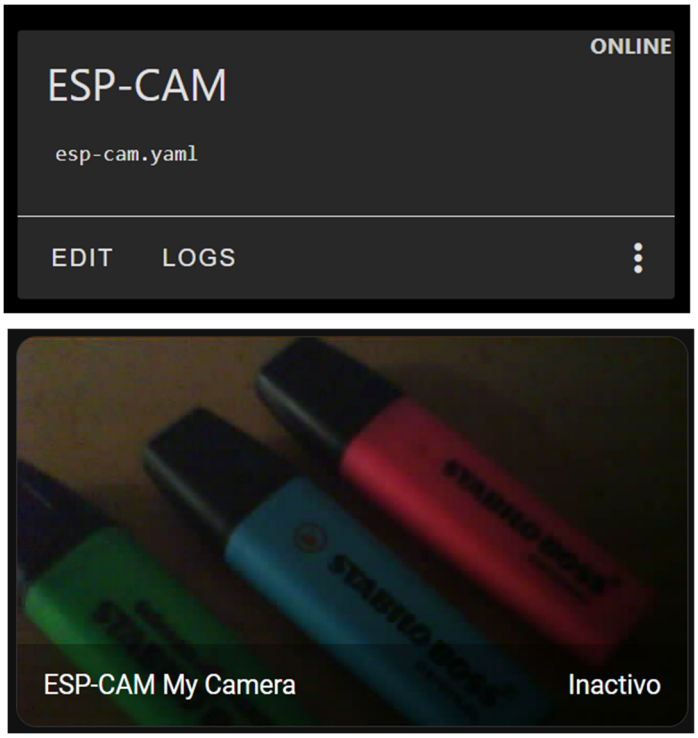

Figura 4.17: Demostración del correcto funcionamiento del ESP32-CAM

5

Capítulo 5 Integración y pruebas del sistema

Una vez desarrolladas las distintas piezas del sistema domótico, tanto a nivel de hardware como de software, resulta esencial validar su integración y funcionamiento conjunto. En este capítulo se describe el proceso de incorporación de los dispositivos al entorno HA, así como las pruebas realizadas para comprobar la correcta comunicación entre los sensores, protocolos y plataformas externas. Asimismo, se analizarán los mecanismos de control remoto, visualización de datos en tiempo real y respuesta del sistema ante distintas situaciones, evaluando así su fiabilidad, escalabilidad y facilidad de uso.

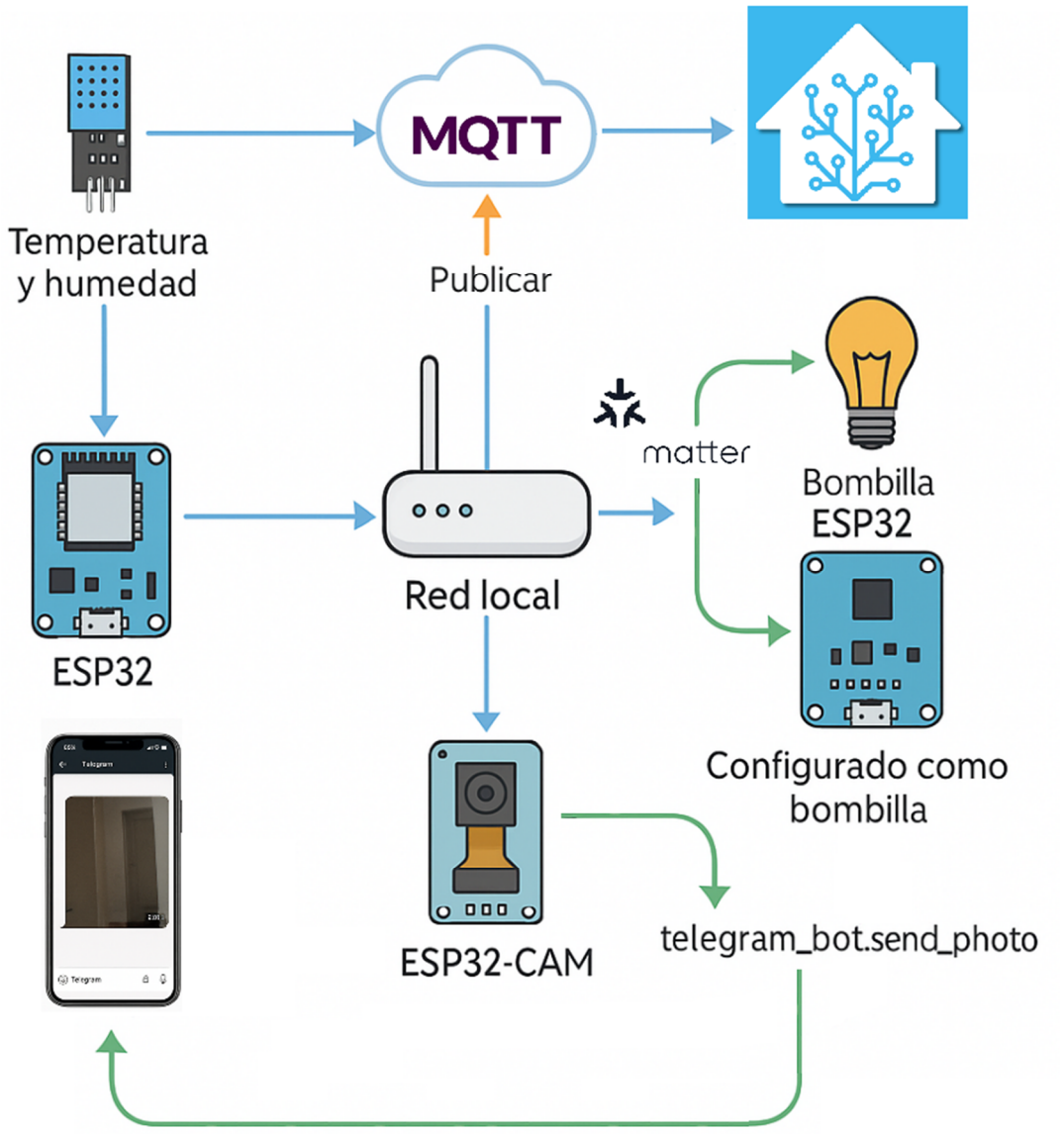

Figura 5.1: Despliegue realizado aplicando diferentes tecnologías

La Figura anterior muestra una representación esquemática del sistema de automatización desarrollado, basado en la integración de múltiples placas ESP32 y coordinado a través de la plataforma HA. En la parte superior izquierda se observa un sensor de temperatura y humedad (KY-015), el cual se encuentra conectado a una primera placa ESP32. Esta placa publica los datos obtenidos mediante el protocolo MQTT, permitiendo su visualización remota desde Home Assistant.

En paralelo, el mismo tipo de sensor ha sido conectado a una segunda placa ESP32 configurada con el estándar Matter, lo que permite su integración directa en el ecosistema de HA como si se tratase de una bombilla inteligente, facilitando así el control nativo y sin necesidad de brókers intermedios.

Por otro lado, un módulo ESP32-CAM se encarga de generar imágenes en tiempo real que son almacenadas localmente en el sistema. Estas imágenes son posteriormente enviadas de forma automatizada mediante el bot de Telegram previamente configurado, lo que aporta una capa adicional de gestión, supervisión y notificación visual.

5.1 Acceso a Home Assistant desde fuera de la LAN

La capacidad de acceder y gestionar un sistema domótico desde fuera de la red local es un requisito esencial para garantizar la flexibilidad, supervisión y control remoto de los dispositivos inteligentes. Home Assistant ofrece diversas formas de habilitar este acceso externo, cada una con sus propias características en términos de configuración, seguridad y dependencia de servicios externos. En esta sección, se analizarán las principales opciones disponibles para acceder a HA desde redes externas, incluyendo Home Assistant Cloud, la apertura de puertos en el router y el uso de la API nativa de ESPHome.

5.1.1 Home Assistant Cloud

Home Assistant Cloud es un servicio de suscripción desarrollado por Nabu Casa, empresa fundada por los propios desarrolladores de HA con el propósito de simplificar el acceso remoto a la plataforma, así como la integración con asistentes de voz como Google Assistant o Amazon Alexa. Su principal ventaja radica en ofrecer una solución segura y sin complicaciones, eliminando la necesidad de realizar configuraciones avanzadas como la apertura de puertos en el router, el uso de servicios DNS dinámicos o la instalación manual de certificados SSL [10].

Previo a la existencia de este servicio, los usuarios que deseaban acceder a HA desde fuera de su red local debían implementar soluciones alternativas, como DuckDNS o VPNs, que, aunque funcionales, implicaban una carga técnica considerable y un mayor riesgo si no se configuraban adecuadamente. En este contexto, Nabu Casa proporciona una solución oficial y mantenida, que no solo mejora la experiencia del usuario, sino que además contribuye directamente al desarrollo sostenible del proyecto HA, al financiarse mediante las suscripciones de usuarios [10][11].

El servicio Home Assistant Cloud incluye las siguientes funcionalidades [11]:

- **Acceso remoto cifrado:** permite acceder de forma segura a la instancia local de HA desde cualquier ubicación, utilizando una URL personalizada del tipo https://nombre-ui.nabucasa.com y comunicación cifrada mediante HTTPS.
- **Integración con asistentes de voz:** facilita la vinculación directa de HA con servicios como Google Assistant o Alexa, evitando configuraciones complejas garantizando una experiencia de usuario unificada.
- **Soporte al desarrollo del proyecto:** la suscripción, con un coste aproximado de 7,70 € mensuales en la Unión Europea, sirve para financiar el mantenimiento del software y sus funcionalidades futuras, sin recurrir a financiación externa ni a modelos basados en publicidad.

Para activar el servicio, el usuario debe acceder al apartado *Home Assistant Cloud* en el menú de configuración. Es importante que el archivo *configuration.yaml* contenga las integraciones *default_config:* y *cloud:* para que esta opción esté disponible. Posteriormente, se podrá iniciar sesión con una cuenta de Nabu Casa o bien crear una nueva, disponiendo de una prueba gratuita de un mes. Una vez autenticado, se habilita la opción *Acceso Remoto*, tras lo cual se genera automáticamente una URL segura para el acceso externo. La activación inicial puede requerir algunos minutos debido a la validación y emisión del certificado HTTPS.

Unión Europea	Reino Unido	Canadá	EE. UU.
7.50 €	6.50 GBP	8.70 CAD	6.50 USD

Figura 5.2: Precios mensuales de Home Assistant Cloud según ubicación

HA Cloud constituye una de las formas más cómodas y seguras para habilitar el acceso remoto a la plataforma. Sin embargo, al tratarse de un servicio de pago, puede no ser la opción preferida en todos los escenarios, especialmente en contextos experimentales o académicos. En este trabajo, no se profundiza en su implementación con sensores remotos debido a esta limitación económica, aunque se deja constancia de su existencia y viabilidad para aquellos usuarios que deseen priorizar la simplicidad y el soporte oficial.

5.1.2 Acceso remoto mediante redirección de puertos y DDNS

En el contexto de una solución domótica autogestionada, establecer un acceso remoto seguro y fiable a HA sin recurrir a servicios de terceros de pago es una prioridad para muchos usuarios avanzados. Una de las alternativas más comunes consiste en configurar manualmente la redirección de puertos en el router doméstico y vincular la dirección IP pública del hogar a un nombre de dominio mediante servicios de DNS dinámico (DDNS). Este enfoque permite mantener el control completo de la infraestructura, evitando la dependencia de plataformas externas como Nabu Casa, adaptándose a distintos entornos de red.

Para comenzar, se debe identificar la dirección IP pública asignada por el proveedor de servicios de internet. Esta información puede consultarse desde la interfaz del propio router o accediendo a plataformas como *What is my ip address.* En caso de disponer de una IP dinámica, es decir que cambia con el tiempo, se recomienda emplear un servicio de DDNS, que se encargue de mantener un dominio actualizado y vinculado con la IP actual, asegurando así la accesibilidad remota al sistema.

La configuración de redirección de puertos se realiza accediendo a la interfaz de administración del router, habitualmente disponible en las direcciones 192.168.1.1 o 192.168.0.1. Desde la sección correspondiente a *Port Forwarding*, se crean reglas que redirigen las solicitudes entrantes hacia la dirección IP local del servidor HA.

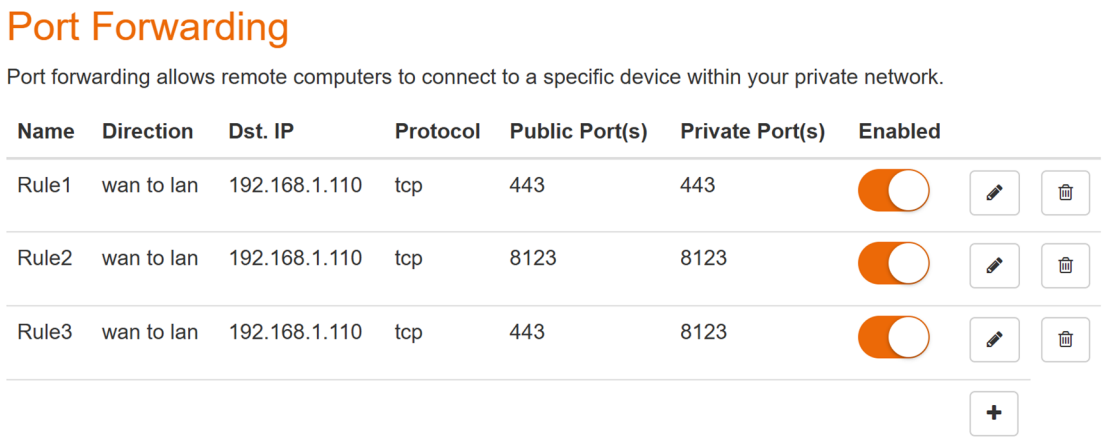

Figura 5.3: Redirección de puertos realizada en el router

En la Figura anterior se muestra un ejemplo de configuración de redirección de puertos en el router, necesaria para permitir el acceso externo de HA desde fuera de la red local. En este caso, se han definido tres reglas que redirigen el tráfico entrante desde la red WAN hacia el servidor donde se encuentra instalado HA, en este caso, con IP local 192.168.1.110.

El puerto 8123 es el puerto por defecto utilizado por HA para su interfaz web. La regla 2 configura la redirección del puerto 8123 externo (público) al mismo puerto interno en el servidor local, permitiendo el acceso mediante la URL *http://IP_PUBLICA:8123* o mediante un dominio personalizado como DuckDNS, escogido en este caso.

El puerto 443 corresponde al puerto estándar para conexiones seguras mediante el protocolo HTTPS. En la regla1, se realiza una redirección del puerto 443 público al mismo puerto en la red local, lo que indica que el servidor de HA está configurado internamente para aceptar conexiones cifradas directamente. Esta opción se utiliza cuando HA gestiona certificados SSL, ya sea de forma nativa o mediante un proxy inverso como NGINX.

Además, se ha definido una tercera regla, que redirige el puerto 443 de la red pública al puerto 8123 del servidor local. Esta configuración permite acceder de forma cifrada, es decir, mediante HTTPS, a HA sin necesidad de exponer directamente el puerto 8123 al exterior. Es especialmente útil cuando se integran servicios como Let's Encrypt o DuckDNS, ya que permite acceder a la plataforma con una URL segura sin tener que especificar el número de puerto.

En paralelo, muchos routers incluyen soporte integrado para servicios de DDNS como No-IP, DynDNS, o soluciones propias del fabricante. Para su configuración se requiere generalmente el registro en un servicio DDSNS compatible, la introducción de las credenciales en el router y la asociación del dominio generado a la IP pública del hogar.

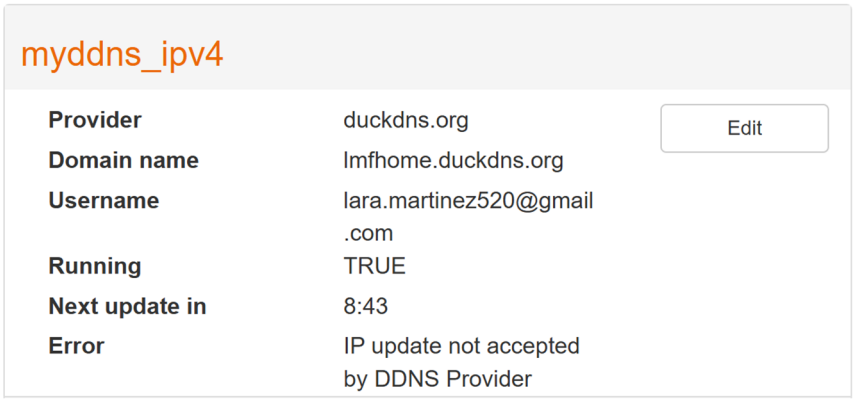

Figura 5.4: Configuración DDNS en el router

Este procedimiento garantiza que incluso ante cambios en la dirección IP pública, el dominio continuará apuntando correctamente a la instalación HA. Una vez configurada la redirección de puertos, se debe modificar el archivo *configuration.yaml* de HA para permitir conexiones externas y aplicar medidas de protección básicas.

Con esta configuración aplicada y el sistema reiniciado, se puede probar el acceso desde otra red empleando la URL ya con la dirección IP pública o, preferentemente mediante el dominio asignado por el servicio DDNS como es, en este caso, *http://lmfhome.duckdns.org:8123*.

```yaml
http:
  use_x_forwarded_for: true
  trusted_proxies:
    - 192.168.1.1  # IP del router
  ip_ban_enabled: true
  login_attempts_threshold: 5
```

DuckDNS es una de las opciones gratuitas más populares para implementar esta funcionalidad. Este servicio asigna un subdominio al usuario, el cual se asocia a su dirección IP pública y se actualiza automáticamente cuando ésta cambia. Además, permite generar certificados SSL gratuitos mediante Let's Encrypt, ofreciendo así cifrado HTPPS para asegurar las comunicaciones.

La implementación de DuckDNS sigue una serie de pasos bien definidos. Primero, se crea una cuenta en DuckDNS y se registra un subdominio, en este caso lmfhome.duckdns.org. A continuación, se obtiene el *token* de autenticación, que será necesario para configurar el complemento desde la interfaz de HA. Este complemento se encarga de mantener el subdominio actualizado con la IP actual y puede integrarse con Let's Encrypt para generar y renovar automáticamente certificados de seguridad.

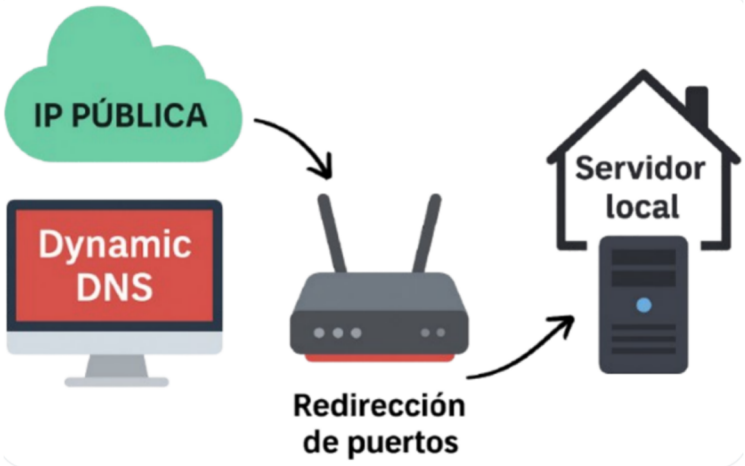

Figura 5.5: Funcionamiento redireccionamiento de puertos

5.1.3 ESPHome API

Además de MQTT, otra opción válida para permitir que un ESP32 se comunique directamente con HA es mediante el uso de ESPHome API, un protocolo nativo diseñado específicamente para facilitar la integración directa de microcontroladores con HA. Este método es especialmente útil para aquellos usuarios que buscan una configuración más sencilla, sin necesidad de brókers intermedios, manteniendo además la capacidad de comunicación bidireccional entre dispositivos y la plataforma de automatización [12].

ESPHome API utiliza un protocolo propietario basado en TCP, que permite al dispositivo intercambiar datos estructurados directamente con HA, incluyendo estados, logs y comandos de control. La principal ventaja de este enfoque es su simplicidad de uso, dado que la integración se realiza automáticamente una vez detectado el dispositivo en la red. No obstante, esta funcionalidad suele estar limitada al entorno de red local a menos que se tomen medidas específicas para permitir el acceso remoto [12].

En este caso, la configuración inicial del ESP32 no difiere de la habitual, excepto por un detalle esencial: si se desea situar el microcontrolador en una red diferente a la de HA, por ejemplo, en una ubicación remota, será necesario modificar las credenciales de red en el archivo de configuración del dispositivo para conectarlo a la nueva red Wi-Fi.

Para que esta integración funcione correctamente desde fuera de la LAN, es imprescindible que el ESP32 pueda alcanzar el servidor de HA a través de Internet. Para ello, se requiere una correcta configuración de red y la apertura de puertos específicos en el router. Concretamente [12]:

- **Puerto 8123:** utilizado por HA para el acceso a su interfaz principal.
- **Puerto 443:** necesario para conexiones seguras mediante HTTPS si se emplean certificados SSL.
- **Puerto 6053:** puerto especifico utilizado por ESPHome API para establecer comunicación entre el dispositivo y HA.

Este último es fundamental. Sin una redirección correcta del puerto 6053 hacia el servidor de HA, el ESP32 no podrá establecer una conexión remota utilizando ESPHome, aunque el resto de los puertos estén abiertos.

En las pruebas realizadas, el dispositivo fue configurado para conectarse a una red Wi-Fi externa, y los puertos necesarios fueron revisados y ajustados en el router. Sin embargo, la conexión no se completó con éxito. Esto podría deberse a múltiples factores, entre ellos: restricciones del ISP, configuración incorrecta del reenvío NAT o problemas de firewall. Dado que ESPHome utiliza una conexión TCP persistente para mantener actualizados los datos y recibir órdenes, cualquier bloqueo o inestabilidad en el puerto 6053 impedirá la comunicación.

Este tipo de integración remota puede considerarse experimental o limitada en escenarios reales, ya que ESPHome está diseñado principalmente para operar en redes locales. Por ello, en entornos donde se requiere una integración remota más fiable, protocolos como MQTT o soluciones en la nube pueden resultar más adecuados.

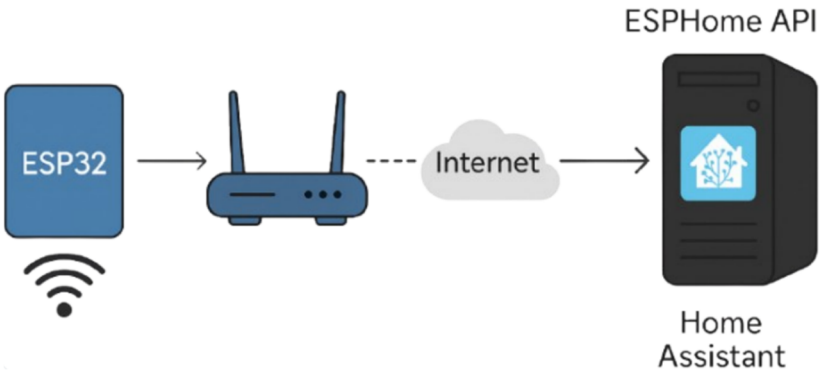

Figura 5.6: Conexión remota de un ESP32 a Home Assistant mediante ESPHome API

5.2 MQTT

Una vez habilitado el acceso remoto a HA mediante la configuración del router y la correspondiente redirección de puertos, se procede a analizar el comportamiento y la eficacia del protocolo MQTT cuando opera desde fuera de la red local. Esta prueba resulta especialmente relevante para validar la interoperabilidad del sistema en entornos distribuidos, donde los sensores o dispositivos conectados no comparten la misma LAN que el servidor domótico.

En este apartado se profundizará en el flujo de comunicación entre un ESP32 configurado como cliente MQTT y el bróker alojado en HA. El objetico es comprobar que el envío de datos desde el exterior hacia el bróker, así como la recepción de comandos por parte del dispositivo se realiza de manera correcta, segura y sin pérdidas de paquetes.

Para realizar esta tarea, se emplea la utilidad tcpdump, una herramienta ligera y eficaz para capturar tráfico en tiempo real. En primer lugar, es necesario instalarla en el entorno del sistema operativo sobre el que se ejecuta HA, en este caso la máquina virtual. Esto se realiza mediante el comando *apk add tcpdump.* Una vez instalada, se ejecuta la captura con el comando *tcpdumo -i enp2s1 -1 captura_mqtt.pcap.* Analizando el comando, *-i enp2s1* indica la interfaz de red por la que se capturará el tráfico, en este caso la correspondiente a la red interna de la máquina, mientras que *-w* especifica el nombre del archivo donde se almacenará el volcado de datos.

Finalizada la captura, se procede a verificar que el archivo ha sido generado correctamente mediante *ls -ila.* Con el archivo presente, se mueve a una ubicación accesible como la carpeta *homeassistant*, mediante el comando *mv captura_mqtt.pcap homeassistant*. Esto permite descargar posteriormente el archivo desde el editor de archivos integrados en HA y abrirlo en Wireshark, una herramienta gráfica especializa en el análisis de paquetes de red.

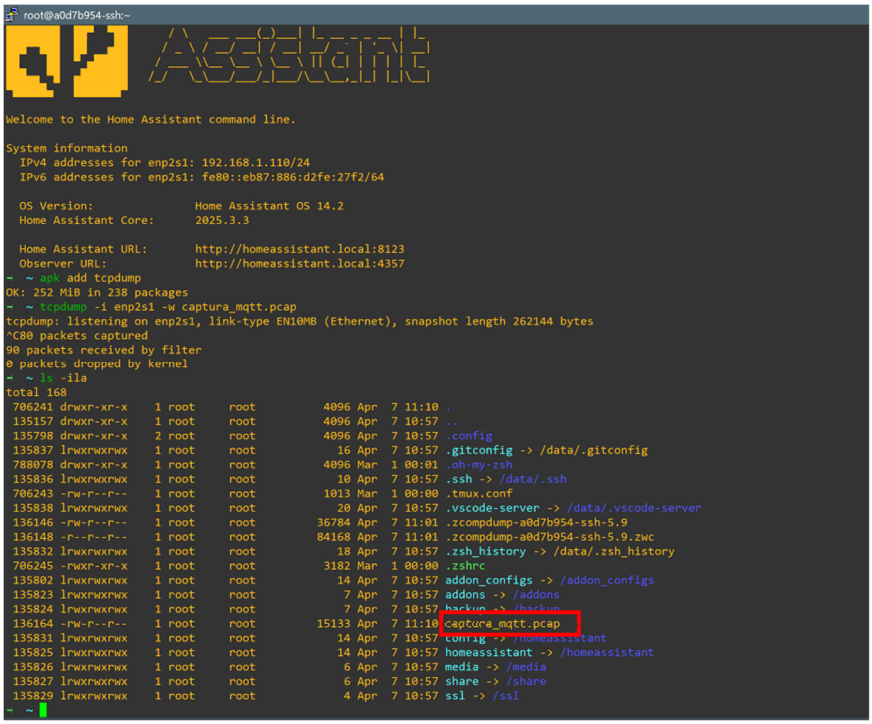

Figura 5.7: Interfaz de la máquina virtual donde se realiza la captura de tráfico

Dentro de Wireshark, se emplea un filtro para visualizar exclusivamente los paquetes relacionados con el protocolo MQTT, lo que facilita enormemente su análisis. A través de esta herramienta es posible comprobar en tiempo real:

- La conexión del cliente (ESP32) al bróker (Mosquitto).
- Las suscripciones y publicaciones realizadas.
- Los topics empleados y su estructura jerárquica.
- El contenido del mensaje, codificado en formato JSON.
- Los niveles de calidad de servicio utilizados.

En este experimento concreto, el dispositivo ESP32, configurado como cliente MQTT, publica datos generados por un sensor KY-015 que mide temperatura y humedad. El bróker MQTT recibe estas publicaciones en topics previamente definidos. Esta comunicación puede observarse claramente en la captura, permitiendo no solo verificar el correcto funcionamiento del sistema, sino también detectar posibles errores en el formato del mensaje, la autenticación o el establecimiento de la sesión.

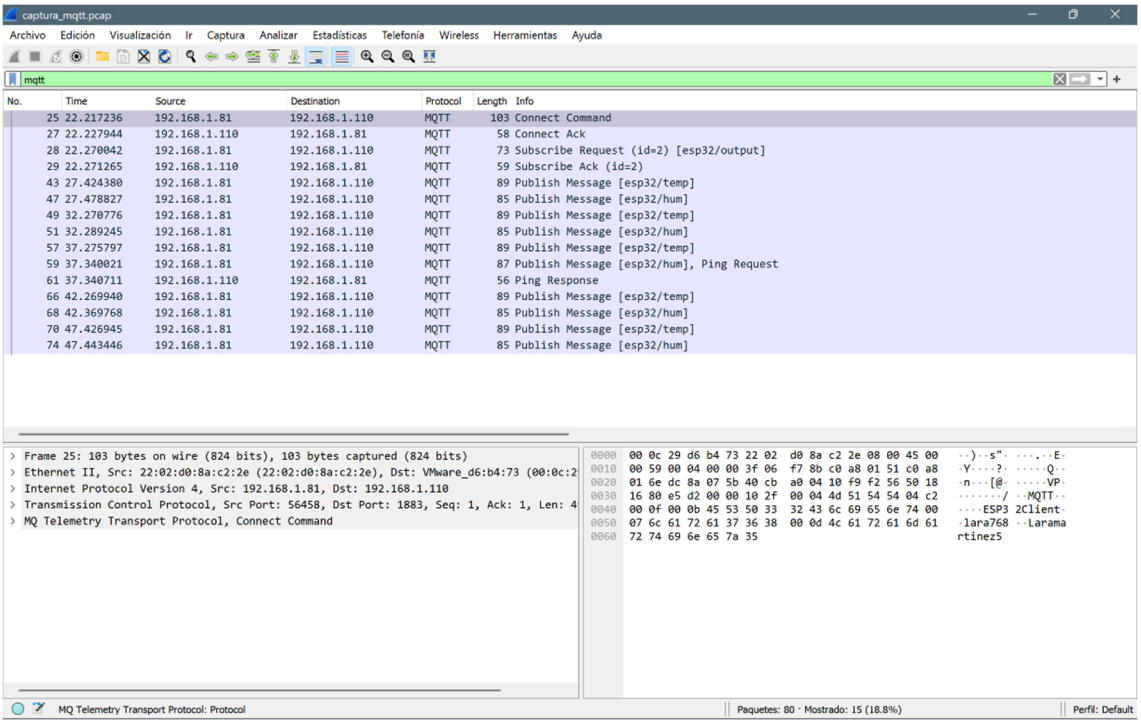

Figura 5.8: Paquetes encontrados tras aplicar el filtro MQTT

Durante el análisis de la captura de tráfico de red correspondiente al protocolo MQTT, se ha podido observar el proceso completo de comunicación entre el cliente y el bróker, alojado en la máquina virtual. La dirección IP correspondiente al bróker es 192.168.1.110. La secuencia de comunicación se inicia con el paquete CONNECT, enviado por el ESP32, en el cual se solicita establecer una nueva conexión con el bróker. Esta trama contiene información esencial para la autenticación y mantenimiento de la sesión como el identificador de cliente (Client ID), las credenciales de acceso (nombre de usuario y contraseña) y el valor de *keep-alive*, que indica cada cuánto tiempo el cliente debe enviar un mensaje de latido para mantener la conexión activa. En respuesta a este intento de conexión, el bróker remite un paquete CONNACK. En la captura analizada, este mensaje de respuesta incluye el parámetro *return code***,** que indica si la conexión ha sido aceptada. En este caso, se observa return code = 0, lo que confirma que la conexión ha sido aceptada de forma satisfactoria.

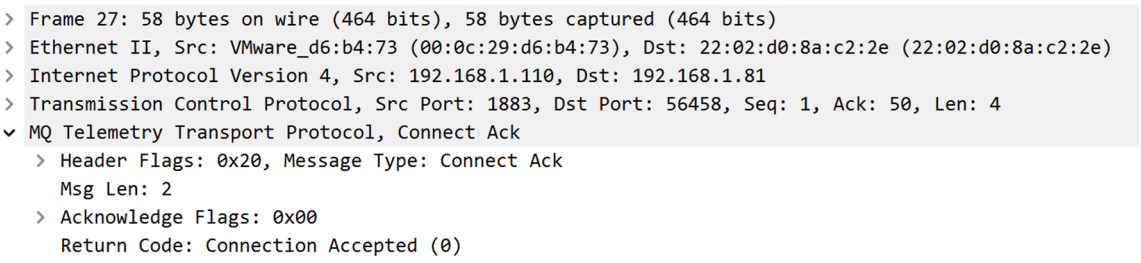

Figura 5.9: Paquete CONNACK

Una vez establecida la conexión, el cliente procede a la suscripción de topics específicos, lo cual se realiza mediante un paquete SUBSCRIBE. En la traza capturada se observa que el cliente se suscribe, por ejemplo, al topic *esp32/output.* Esta acción es reconocida por el bróker mediante el envío de un paquete SUBACK, donde se confirma el nivel de calidad de servicio que ha sido aceptado para esa suscripción. En este caso, se ha utilizado QoS 0, lo que implica que los mensajes serán enviados una única vez, sin necesidad de confirmación, optimizando así el tiempo de respuesta, aunque con menos fiabilidad en caso de pérdida de paquetes.

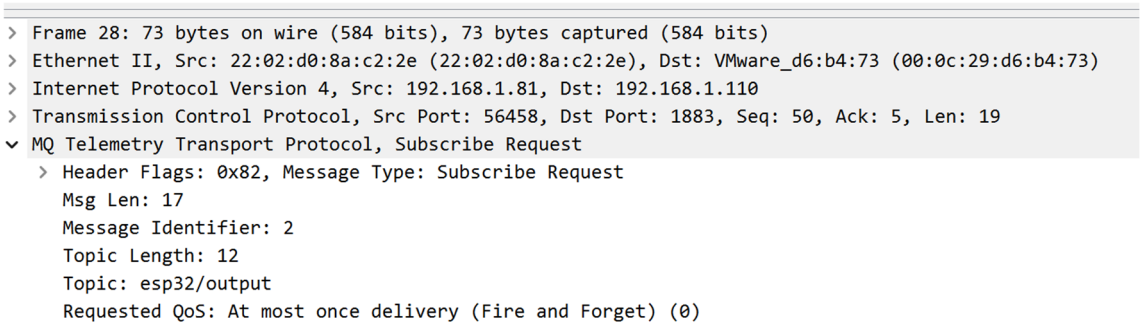

Figura 5.10: Paquete SUBSCRIBE

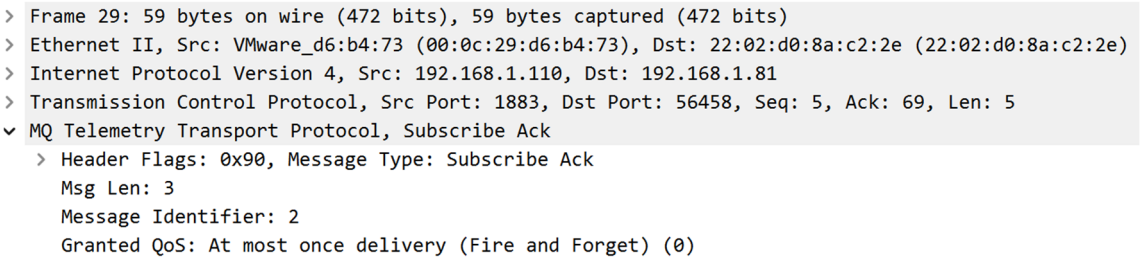

Figura 5.11: Paquete SUBACK

A continuación, se identifican múltiples tramas PUBLISH, enviadas periódicamente con una frecuencia aproximada de cinco segundos, que contienen los datos recopilados por el sensor de temperatura y humedad conectado al ESP32. El contenido de estos mensajes se presenta en formato JSON, de forma estructurada como se puede ver en la Figura 5.12. Este mensaje se publica sobre el topic *esp32/temp*, coincidiendo con la configuración especificada previamente en el fichero *configuration.yaml* de HA. Gracias a la estructura jerárquica de los topics, basada en delimitadores por barras, se logra una organización modular del sistema, facilitando tanto su escalabilidad como el mantenimiento. Esta organización resulta especialmente útil cuando se integran múltiples sensores o nodos distribuidos.

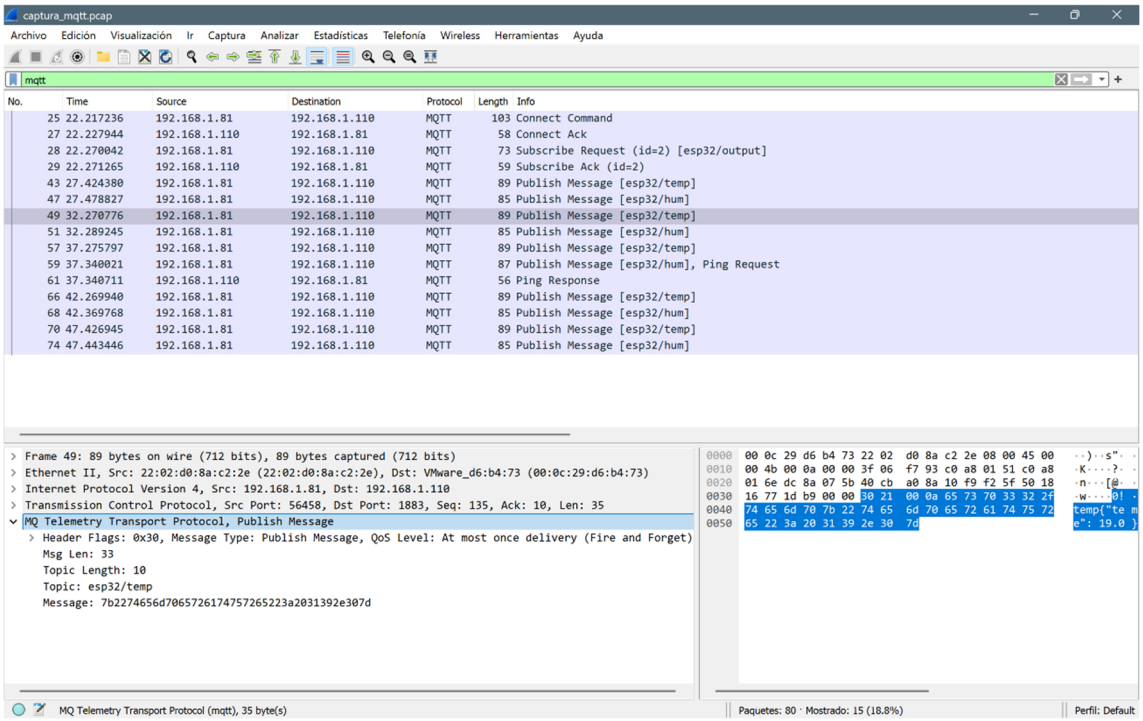

Figura 5.12: Paquete PUBLISH

Cada paquete PUBLISH observado en la captura se transmite con un nivel de calidad de servicio QoS 0, lo que implica que no se requiere confirmación por parte del receptor. Como resultado, no se generan tramas de tipo PUBACK, lo que minimiza la sobrecarga de la red y favorece la eficiencia del sistema, aunque se asume el riesgo de pérdida de mensaje en condiciones adversas de conectividad.

Mediante el análisis de las marcas de tiempo (*timestamps*) de las tramas capturadas, se ha verificado que el ESP32 publica los datos de forma periódica y regular, con un intervalo aproximado de cinco segundos. Esto demuestra el correcto funcionamiento del bucle de envío programado en el firmware del dispositivo, conforme a los parámetros definidos.

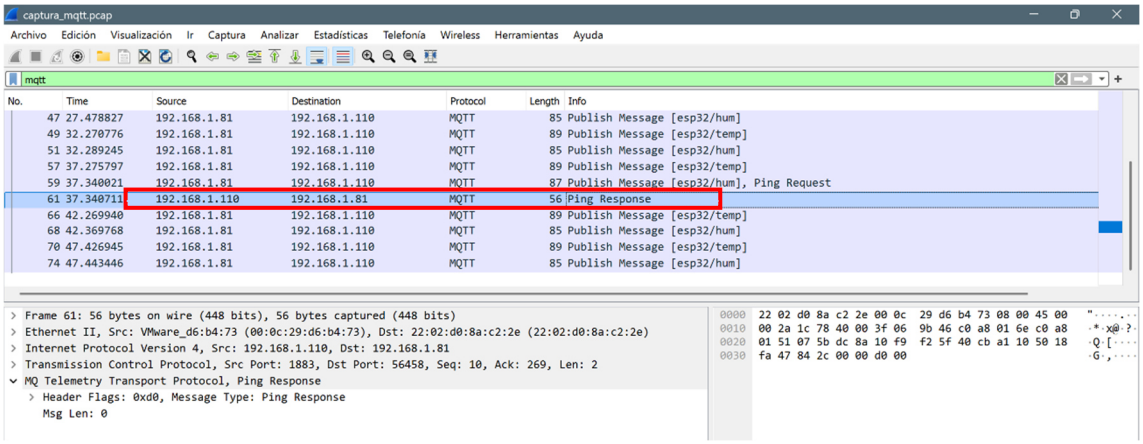

Figura 5.13: Paquete Ping Response

Desde la interfaz gráfica de HA, los datos transmitidos por el ESP32 se visualizan correctamente gracias a la opción de descubrimiento automático habilitada en el bróker MQTT mediante la directiva *Discovery: true*. Esta funcionalidad permite que HA detecte nuevos dispositivos o sensores y cree entidades asociadas a los topics relevantes de manera automática, sin necesidad de intervención manual.

Como aspecto adicional del análisis, se ha verificado que el sistema MQTT mantiene activa la sesión mediante tramas del tipo PINGREQ y su correspondiente respuesta PINGRESP. Estas tramas son intercambiadas en ausencia de datos con el fin de asegurar la persistencia de la conexión TCP entre cliente y bróker, según los parámetros definidos por el valor de *keep-alive* en el paquete CONNECT. Este mecanismo forma parte de las especificaciones del protocolo y es esencial para detectar posibles desconexiones o fallos de red.

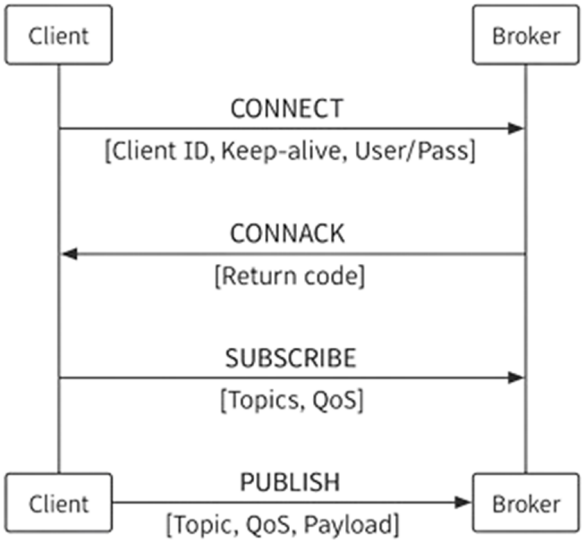

Figura 5.14: Interacción del protocolo MQTT

El esquema anterior proporciona una visión clara del flujo de comunicación entre los distintos componentes del sistema, ilustrando cómo el cliente establece la conexión con el bróker, se suscribe a los topics configurados y comienza a publicar datos de manera automatizada y en tiempo real. Se trata de una representación del comportamiento típico de un dispositivo IoT utilizando el protocolo MQTT, en este caso un ESP32 actuando como cliente publicador.

La captura de la Figura 5.15 refleja una secuencia completa de mensajes entre el cliente MQTT, con dirección 192.168.1.81 correspondiente al ESP32, y el bróker MQTT, ejecutándose en una máquina virtual con HA. El intercambio de información comienza con el establecimiento de una conexión TCP, requisito previo para utilizar el protocolo MQTT. Este procedimiento se lleva a cabo mediante el mecanismo *three-way-hadshake,* en el que el cliente envía un paquete SYN, el servidor responde con un SYN-ACK y finalmente el cliente confirma con un ACK. Esta fase garantiza una conexión fiable sobre la cual se desarrollará la sesión MQTT.

Una vez establecida la capa de transporte, el ESP32 envía un paquete MQTT CONNECT, solicitando la apertura de sesión con el bróker. Este mensaje incluye parámetros como el identificador del cliente, credenciales de autenticación y el valor de *keep-alive*. El bróker responde con un CONNACK, confirmando la aceptación de la conexión y el inicio de la sesión.

Posteriormente, el ESP32 procede a la suscripción a los topics configurados, como *esp32/output*, a través del comando SUBSCRIBE. La confirmación se produce mediante el mensaje SUBACK emitido por el bróker. En este punto se ha detectado un evento de TCP Retransmission, lo que indica una posible pérdida de paquetes o un retardo momentáneo en la red. No obstante, el protocolo TCP garantiza la entrega íntegra de los datos mediante la retransmisión automática.

En lo que respecta a la publicación de mensajes MQTT, se observa una secuencia periódica de paquetes PUBLISH dirigidos a los topics *esp32/temp* y *esp32/hum*, que contienen los datos recogidos por el sensor integrado en el ESP32. Cada publicación va seguida por un paquete de tipo ACK, que confirma la recepción por parte del bróker.

En la parte final de la secuencia capturada, se observa un mensaje de tipo PUBLISH dirigido al topic *esp32/hum*, inmediatamente seguido por un paquete PINGREQ, lo que indica que el ESP32, además de publicar los datos del sensor, ejecuta una solicitud para mantener activa la conexión con el bróker. Esta práctica es habitual en sesiones MQTT con un intervalo de actividad definido mediante el parámetro *keep-alive*. El bróker responde con un mensaje ACK, lo que confirma la recepción tanto de los datos como de la solicitud de mantenimiento de la sesión. Aunque en el gráfico no se visualizan explícitamente mensajes PUBACK, puede inferirse que todos los mensajes han sido enviados con calidad de servicio QoS 0, es decir, sin confirmación.

Cabe destacar que, si bien se detectó una retransmisión TCP durante las primeras fases de la comunicación, esta no afectó al comportamiento global del sistema. La conexión se estabilizó rápidamente, lo que demuestra la robustez del protocolo TCP frente a pequeñas interrupciones o pérdidas puntuales.

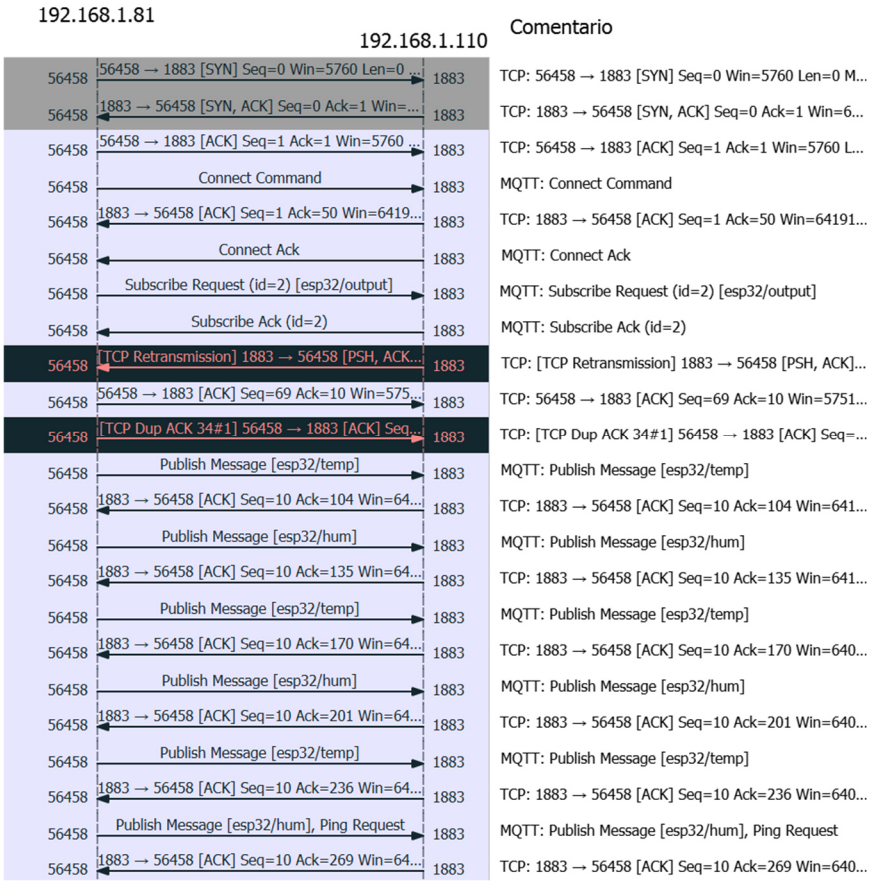

Figura 5.15: Intercambio de mensajes entre el cliente MQTT y el bróker

5.3 Matter

Una vez establecido el entorno de integración con dispositivos compatibles con el protocolo Matter, resulta esencial realizar un análisis detallado del tráfico generado durante su funcionamiento. Al igual que se realizó en el caso de MQTT, este apartado se centra en examinar las tramas intercambiadas entre los dispositivos involucrados en la red Matter, permitiendo comprender su arquitectura de comunicación, los mecanismos de emparejamiento y control, así como los protocolos subyacentes que sustentan dicha interacción.

El análisis de las capturas de red ofrece una visión profunda de cómo se realiza la transmisión de comandos entre el controlador, en este caso, Home Assistant, y los dispositivos Matter, destacando aspectos como el uso de IPv6, la autenticación entre nodos, la distribución del tráfico entre los distintos protocolos involucrados (TCP, UDP, mDNS, etc.), y las medidas de seguridad que garantizan la confidencialidad e integridad de los datos.

Al igual que en el caso de MQTT, el análisis se ha realizado mediante la combinación de las herramientas *tcpdump* y Wireshark, utilizando archivos en formato *.pcap* generados directamente desde la máquina virtual donde reside HA.

Durante la fase de análisis se han empleado filtros como *mdns* para el descubrimiento de dispositivos Matter mediante multicast, *udp.port == 5540* para capturar el tráfico propio del protocolo Matter, o *ipv6* para estudiar las direcciones de red implicadas, dado que este estándar se basa fundamentalmente en el protocolo IP versión 6. A través de esta captura se han podido identificar tramas clave del protocolo y observar el comportamiento de las capas de transporte utilizadas.

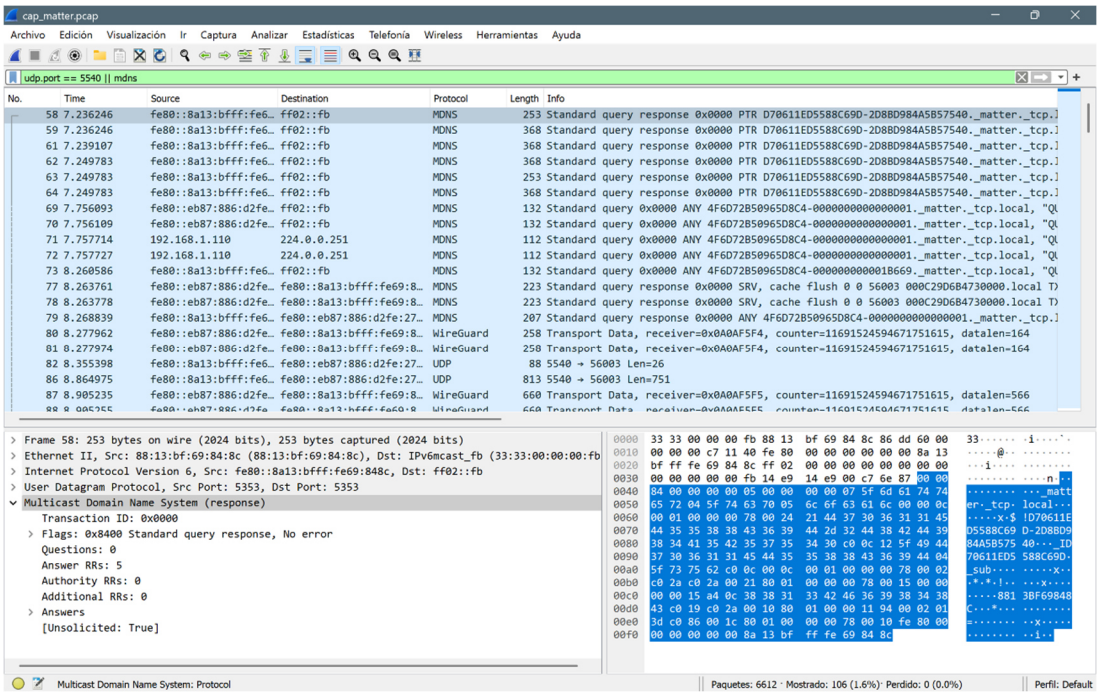

Figura 5.16: Captura realizada con Wireshark en local, es decir analizando la interfaz Wi-Fi

En el entorno de pruebas se ha detectado el uso de diversas direcciones IP privadas (IPv4), aunque en la práctica Matter no depende de este tipo de direcciones para su funcionamiento principal. Algunas de las direcciones identificadas son las siguientes:

- **192.168.1.110:** dirección asignada al nodo controlador, correspondiente al servidor de HA.
- **192.168.1.82:** dirección asignada al dispositivo ESP32, aunque no utilizada activamente en las comunicaciones Matter debido a la naturaleza basada en IPv6 del protocolo.

Tal como establece la especificación de Matter, la comunicación entre los nodos se lleva a cabo utilizando direcciones IPv6. En la captura de tráfico se han identificado direcciones *link-local*, es decir, con prefijo fe80::/10, asociadas a los dispositivos participantes:

- **fe80:eb87:886:d2fe:** dirección IPv6 correspondiente a HA.
- **fe80:8a13:bfff:fe69:848c:** dirección del dispositivo Matter basado en ESP32.

Estas direcciones permiten la comunicación directa entre nodos dentro de la misma red local, sin necesidad de configuración adicional de enrutamiento. Esto respalda uno de los principios fundamentales de Matter: la operatividad local sin depender de servicios en la nube.

En cuanto a los puertos utilizados, se han identificado los siguientes:

- **Puerto 5540:** se trata del puerto por defecto definido por el estándar Matter para el transporte de mensajes de capa de aplicación sobre UDP. Todos los dispositivos compatibles con Matter escuchan en ese puerto.
- **Puerto 5353:** utilizado por el protocolo *Multicast DNS (mDNS)* para descubrir dispositivos disponibles en la red local, lo que permite que los controladores identifiquen automáticamente los nodos Matter activos.
- **Puerto 56826:** corresponde a un puerto efímero asignado dinámicamente por el sistema operativo del dispositivo cliente, por ejemplo, HA o una aplicación móvil. En contextos como Matter, donde las conexiones pueden iniciarse desde cualquier nodo, es común que los dispositivos utilicen puertos de origen temporales para iniciar la comunicación hacia el puerto 5540 del servidor.

El uso combinado de IPv6, puertos estándar como el 5540 y mecanismos de descubrimiento automático como mDNS ejemplifica el enfoque descentralizado, seguro y autónomo adoptado por Matter. Esta infraestructura de red permite establecer conexiones punto a punto entre dispositivos sin intermediarios, optimizando la latencia, reduciendo la dependencia de servicios en la nube y facilitando una mayor privacidad en el tratamiento de datos.

A pesar de que el contenido de los paquetes UDP dirigidos al puerto 5540 se encuentra cifrado, como es habitual en las implementaciones de Matter una vez completado el proceso de emparejamiento, durante la captura de tráfico fue posible observar la emisión y recepción de múltiples paquetes entre el nodo controlador (HA) y el dispositivo ESP32. Estos paquetes se generan tras ejecutar acciones como el encendido o apagado de una bombilla inteligente desde el panel de control. La coincidencia temporal entre las acciones realizadas por el usuario y la aparición de estos paquetes cifrados sugiere que, bajo el encapsulamiento de Matter, se están transmitiendo comandos del tipo CoAP sobre UDP, utilizados para controlar los dispositivos.

Además, se identificó el servicio *_matter._tcp.local* como uno de los elementos clave del descubrimiento automático de dispositivos Matter en la red local. Este servicio es publicado mediante mDNS sobre el puerto 5353, permitiendo a los dispositivos anunciar su presencia y ser descubiertos sin necesidad de servidores DNS centralizados. Aunque el nombre del servicio incluye el sufijo *_tcp*, este hace referencia al tipo de servicio según la convención de DNS-SD

(Service Discovery) y no al protocolo de transporte utilizado para la comunicación, que sigue siendo UDP durante el descubrimiento.

Durante el análisis también se detectaron tramas PTR, SRV y TXT que forman parte del proceso de resolución mDNS. Estas tramas contienen metainformación sobre el dispositivo, como su nombre, dirección IP, puerto y capacidades funcionales. Gracias a esta infraestructura, el proceso de emparejamiento resulta sencillo, autónomo y sin necesidad de configuraciones manuales, reforzando así el modelo plug-and-play que Matter promueve.

De forma paralela, en la captura se identificó tráfico relacionado con otros protocolos como QUIC y WireGuard, los cuales no forman parte del stack oficial de Matter, pero podrían estar asociados a servicios adicionales. QUIC, utilizado por HTTP/3, es un protocolo de transporte moderno basado en UDP y se emplea comúnmente para mejorar la velocidad y seguridad en comunicaciones con servidores web. Por otro lado, la aparición de tráfico WireGuard sugiere el uso de túneles VPN activos, posiblemente empleados para el acceso remoto seguro al entorno domótico.

De forma paralela se ha realizado otra captura con tcpdump dentro de la máquina virtual para observar si hay alguna diferencia. Un elemento relevante observado en la captura fue la existencia de tráfico correspondiente al puerto 8123, que es utilizado por HA tanto para su interfaz web como para su API WebSocket. En concreto, se identificaron tramas WebSocket con llamadas del tipo *call_service*, lo que indica que se estaban ejecutando comandos desde la interfaz gráfica de usuario (dashboard) del sistema. Estas tramas permiten invocar servicios en tiempo real, como puede ser el encendido de luces, la activación de escenas o la lectura de sensores.

El flujo observado muestra cómo un cliente, por ejemplo, el navegador del PC con IP 192.168.1.90, abre una conexión WebSocket al puerto 8123 y envía peticiones que son interpretadas por el backend de Home Assistant. La conexión se mantiene abierta durante toda la sesión, utilizando un puerto efímero del cliente, por ejemplo, 1807, y habilita el intercambio bidireccional de datos sin necesidad de recargar la interfaz o realizar nuevas solicitudes HTTP.

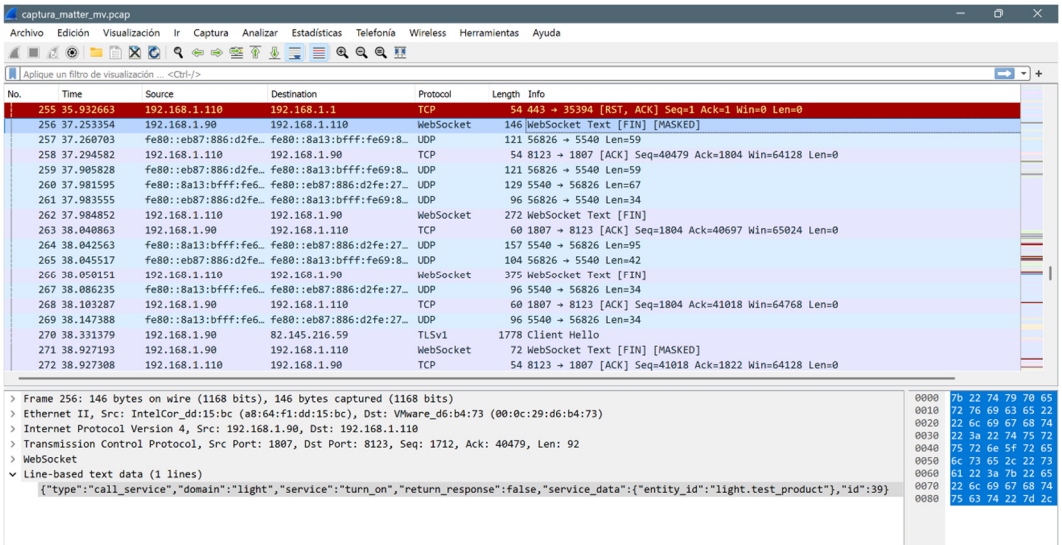

Figura 5.17: Captura realizada con tcpdump en la máquina virtual

Durante la ejecución de una acción desde la interfaz de Home Assistant, como puede ser el encendido de una bombilla Matter mediante el panel de usuario, se produce una secuencia bien definida de eventos que refleja el diseño modular y seguro de la arquitectura Matter. Este flujo de comunicación se ha podido analizar a través de la captura de tráfico de red, revelando la implicación de distintos protocolos y capas de comunicación.

a. **Trama WebSocket:** la primera acción registrada es una trama correspondiente al protocolo WebSocket, en la que el navegador del usuario envía una orden a HA a través del puerto 8123, utilizando un puerto efímero de origen (1807). Esta trama contiene información de alto nivel como el tipo de servicio (*light, switch, climate*, etc), la acción a realizar (*turn_on, turn_off*) y la entidad afectada (*entity_id: light.test_product*). Esta comunicación ocurre sobre TCP y utiliza HTTP encapsulado, garantizando la transmisión fiable del comando desde la interfaz hasta el backend de HA.

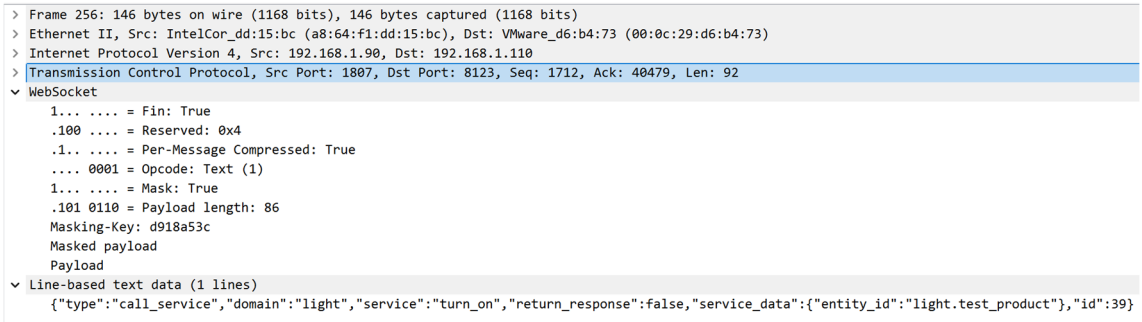

Figura 5.18: Trama WebSocket

b. **Trama UDP:** de forma inmediata, tras recibir la orden, HA genera un paquete UDP dirigido a la dirección IPv6 del dispositivo Matter correspondiente. En esta trama se encapsula el comando de control, ahora transformado en una instrucción cifrada bajo el protocolo Matter. Si bien el contenido del mensaje no puede ser interpretado directamente, debido al uso de cifrado extremo a extremo, su tamaño, momento de envío y puerto de destino coinciden con las características esperadas de un paquete Matter. El dispositivo Matter había anunciado previamente su punto de control mediante mDNS, lo que permite a HA dirigir correctamente la instrucción a través de la red local. El mensaje, que presenta una estructura binaria consistente con TLV viaja a través del puerto 5540, definido como el puerto estándar de comunicación Matter sobre UDP.

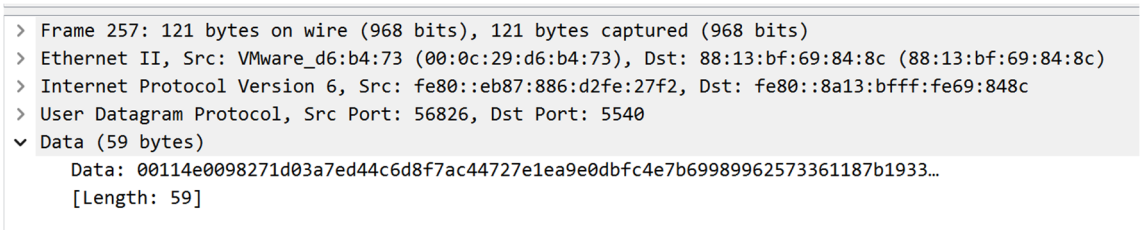

Figura 5.19: Trama UDP

c. **Trama TCP ACK:** la tercera trama observada es una confirmación TCP ACK enviada desde HA al cliente. Este paquete no contiene datos útiles a nivel de aplicación, pero cumple una función crucial en la gestión de la sesión WebSocket, confirmando que el backend ha recibido y procesado correctamente el comando del usuario. En este intercambio, se mantiene la conexión viva y estable, permitiendo nuevas interacciones sin necesidad de reiniciar el canal de comunicación.

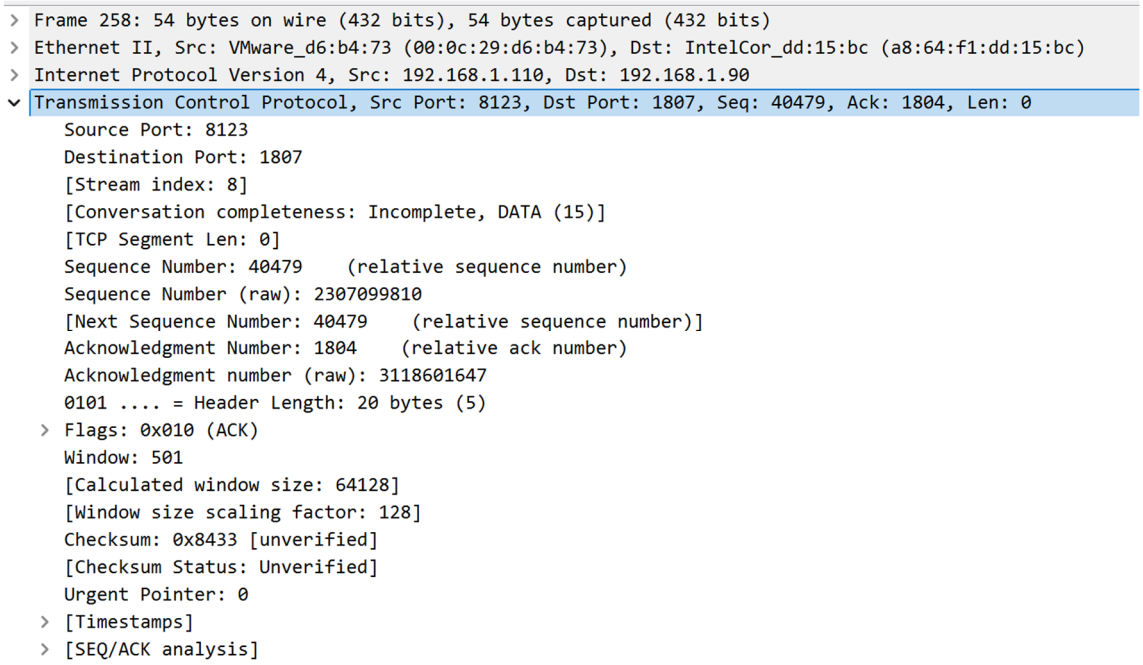

Figura 5.20: Trama TCP ACK

Este patrón de tres fases confirma el enfoque en capas de la arquitectura Matter, donde:

- La interfaz gráfica utiliza WebSocket para comunicarse con HA.
- El core del sistema transforma la acción del usuario en una instrucción Matter cifrada.
- La red local entrega dicha instrucción mediante UDP e IPv6 al dispositivo.
- La comunicación TCP garantiza la retroalimentación inmediata al usuario.

Cabe destacar que, debido al diseño de seguridad de Matter, los paquetes UDP capturados no contienen información legible sin acceso a las claves de sesión. El protocolo implementa cifrado a nivel de aplicación utilizando TLS 1.3 y el mecanismo CASE (Certificate Authenticated Session Establishment), lo que impide que terceros puedan interceptar y leer los comandos transmitidos, incluso en redes locales. Esta característica convierte a Matter en una solución robusta en términos de seguridad, privacidad y resistencia frente a ataques de interceptación o manipulación.

Durante la creación de una sesión segura de Matter, como es el caso del emparejamiento inicial entre un controlador y un dispositivo, se lleva a cabo un intercambio de claves mediante el algoritmo ECDH (Elliptic Curve Diffie-Hellman). Este mecanismo permite generar una clave de sesión de forma dinámica y efímera, lo que implica que dicha clave nunca se transmite en texto plano ni se reutiliza entre sesiones, garantizando así *Forward Secrecy* y elevando el nivel de seguridad de las comunicaciones.

Aunque Matter fue concebido como un protocolo nativamente basado en IPv6, la captura de tráfico realizada evidencia la coexistencia de paquetes IPv4 e IPv6 en el entorno de red analizado.

Esta situación es habitual en entornos mixtos, como laboratorios o redes domésticas, donde cohabitan dispositivos legacy (limitados a IPv4) y otros compatibles con IPv6. En estos casos, las pilas de red de los dispositivos suelen operar en modo *dual stack*, permitiendo el uso simultáneo de ambos protocolos.

Matter utiliza exclusivamente IPv6 para la comunicación entre dispositivos del ecosistema, ya que muchos de sus mecanismos internos, como mDNS para el descubrimiento de servicios o la autoconfiguración de direcciones mediante SLAAC (Stateless Address Autoconfiguration), están específicamente optimizados para este protocolo. No obstante, otros servicios auxiliares, como la conexión del usuario con HA a través del navegador, pueden seguir operando sobre IPv4, especialmente si el cliente no está preparado para IPv6.

En este sentido, por ejemplo, los mensajes de control enviados desde la interfaz de usuario de HA a través de WebSocket, como *turn_on* o *turn_off,* pueden circular sobre IPv4, mientras que la acción final que se entrega al dispositivo Matter se transmite mediante IPv6, aprovechando las capacidades del dispositivo y el controlador para este protocolo.

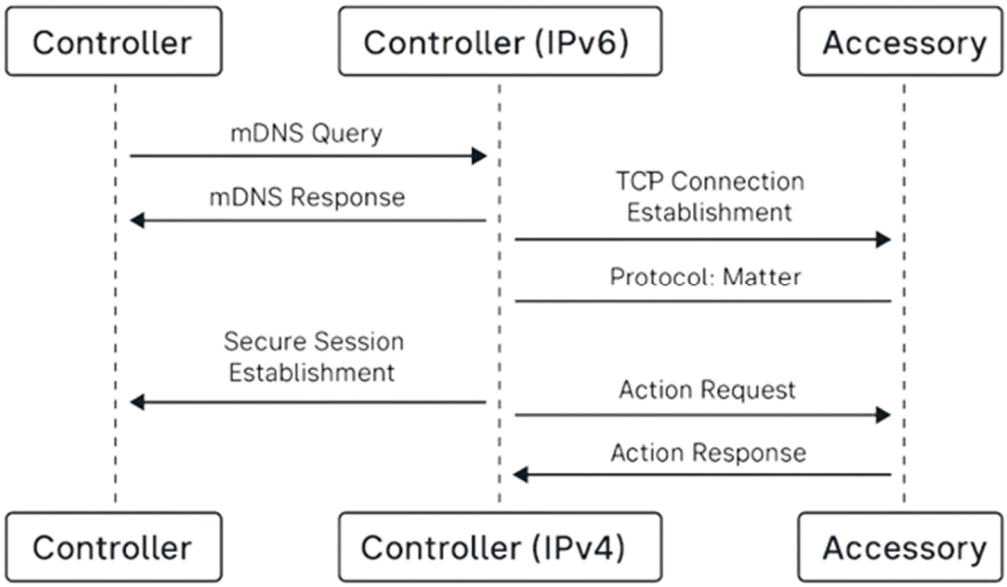

Figura 5.21: Secuencia de comunicación del protocolo Matter

En el diagrama de flujo generado en la Figura 5.21 aparecen representados el *Commissioner* y el *Controller* dentro del mismo bloque. Esta representación, aunque útil a nivel didáctico, no refleja con exactitud la separación de roles en el protocolo Matter. En la práctica, estos componentes pueden estar integrados en el mismo dispositivo, como un smartphone que actúa primero como *Commissionerk* durante el provisionamiento y, posteriormente como *Controller* para el control del dispositivo, o pueden estar distribuidos entre diferentes sistemas.

- El *Commissioner* es el encargado del proceso de *commissioning*, es decir, de añadir el dispositivo a la red Matter.
- El *Controller* es quien mantiene el control funcional del dispositivo tras su integración, ya sea a través de una app móvil, un hub o una plataforma como HA.

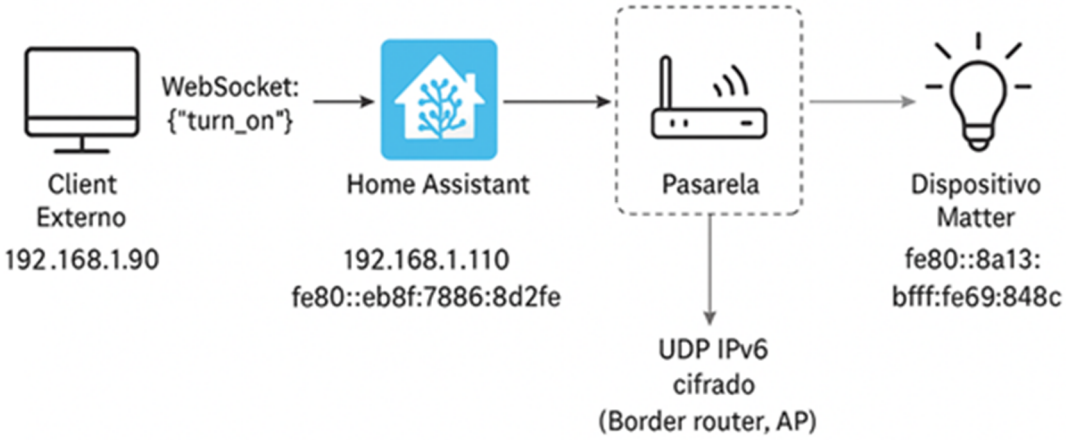

Figura 5.22: Coexistencia de IPv4 e IPv6 en una red Matter

La Figura 5.21 representa un esquema detallado de la coexistencia de ambos protocolo en un entorno doméstico donde se ha desplegado Matter:

- **Cliente externo (IPv4 192.168.1.90):** se trata de un PC que se comunica con HA mediante WebSocket sobre el puerto 8123. Esta vía se emplea para emitir órdenes como encender o apagar dispositivos.
- **Servidor HA:** está alojado en una máquina virtual configurada en modo *dual stack.* Dispone de conectividad IPv4 (192.168.1.110) e IPv6 (fe80::eb87:886:d2fe). Actúa como el *Controller* en el ecosistema Matter.
- **Dispositivos Matter (ESP32):** posee dirección IPv6 (fe80::8a13:bfff:fe69:848c) y se comunica únicamente bajo este protocolo, utilizando UDP cifrado para recibir los comandos de control.

Este flujo confirma cómo Matter aprovecha plenamente IPv6 para la comunicación segura entre dispositivos, mientras que IPv4 sigue siendo utilizado por las capas superiores, como la interfaz de usuario o los servicios web asociados a HA. Esta coexistencia refleja el enfoque gradual con el que se está adoptando IPv6 en el ámbito IoT, sin abandonar por completo los protocolos heredados.

La Figura 5.23 representa una captura de tráfico que evidencia la secuencia de mensajes intercambiados entre un cliente (navegador web del usuario) y el servidor de HA (instalado en una máquina virtual con IP 192.168.1.110) en el contexto del control de dispositivos Matter.

- **Establecimiento de conexión TCP (Three-way-handshake):** se inicia el intercambio de paquetes SYN, SYN-ACK y ACK entre el puerto 1805 del cliente y el puerto 8123 del servidor. Esto establece una conexión fiable entre ambas máquinas, que servirá como canal para transmitir los mensajes de la sesión.
- **Autenticación y carga inicial de la interfaz:** tras la conexión, se realiza una petición GET que solicita la interfaz principal de HA. El servidor responde con un *HTTP/1.1 200 OK*, indicando una entrega exitosa. Posteriormente, el cliente envía una petición POST a */auth/token*, que es el endpoint encargado de obtener el token de autenticación. Esto es un paso necesario para establecer una sesión segura. Se obtiene la misma respuesta que antes, confirmando que la autenticación fue válida y que el cliente está autorizado.

- **Inicio de sesión WebSocket:** el cliente abre una nueva conexión TCP y solicita una actualización al protocolo WebSocket mediante una petición GET */api/websocket HTTP/1.1.* HA responde con un *101 Switching Protocols*, que indica que la conexión HTTP ha sido convertida correctamente a una sesión WebSocket persistente. A partir de aquí, la comunicación es bidireccional y en tiempo real, permitiendo al cliente enviar comandos, recibir actualizaciones y mantener una sesión activa sin necesidad de abrir nuevas conexiones.
- **Intercambio de mensajes WebSocket:** se observa una serie de paquetes *Websocket Text [FIN]* y *Websocket Text [MASKED],* lo cual corresponde a mensajes enviados por el cliente cifrados o truncados por motivos de seguridad o configuración del visor. El uso de tramas con [MASKED] es típico en WebSocket cuando el contenido está encriptado o comprimido para optimizar el rendimiento de la comunicación. Finalmente, se aprecian múltiples mensajes consecutivos que corresponden a comandos o eventos transmitidos por el cliente o por el servidor, relacionados con acciones sobre dispositivos Matter o actualizaciones de estado.

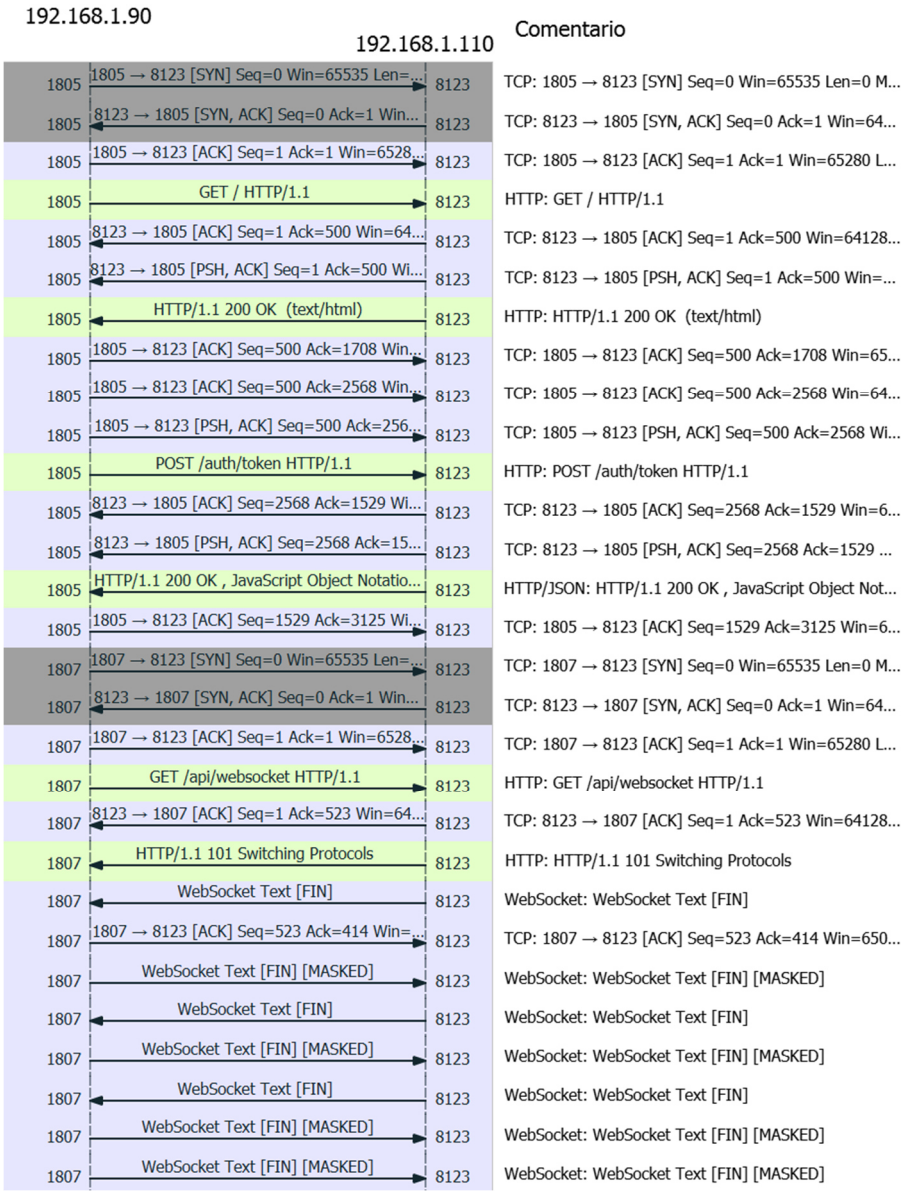

Figura 5.23: Captura del establecimiento y uso de una sesión WebSocket entre cliente y HA

5.4 Envío de imágenes desde ESP32-CAM a Telegram

Tras haber logrado la integración funcional del módulo ESP32-CAM en la plataforma HA, con la posibilidad de visualizar la transmisión en directo desde el panel principal, se consideró pertinente ampliar sus capacidades mediante la implementación de un mecanismo de captura y envío automático de imágenes a través de la plataforma de mensajería Telegram.

Esta funcionalidad resulta especialmente útil en entornos orientados a la vigilancia o supervisión remota, dado que permite recibir notificaciones visuales de eventos relevantes como la detección de movimiento, sin necesidad de acceder directamente a la interfaz de HA. De esta manera, se refuerza el carácter reactivo del sistema y se mejora significativamente la capacidad de interacción remota por parte del usuario.

El objetivo de esta sección es aprovechar la ya establecida integración de HA con Telegram para transmitir imágenes generadas por el ESP32-CAM. Para ello, se explora el procedimiento que permite capturar una imagen en tiempo real, almacenarla en el sistema de archivos y, seguidamente, enviarla como archivo adjunto al usuario, sin necesidad de exponer servicios a través de HTTPS o dominios externos. Esta estrategia, además de mantener la seguridad de la red local, mejora la fiabilidad del sistema al reducir su dependencia de servicios en la nube o configuraciones complejas.

En primer lugar, se requiere una carpeta accesible dentro del sistema donde almacenen temporalmente las imágenes capturadas. Para ello, mediante el complemento *File Editor* se procede a la creación de una carpeta denominada *www* dentro del directorio de configuración, */config.* Dentro de esta carpeta, se crea una subcarpeta adicional denominada *capturas*, destinada específicamente al almacenamiento de las imágenes. El nombre de esta última puede variar según el usuario desee.

Con la estructura de directorios establecida, se procedió a garantizar que HA tuviera los permisos necesarios para escribir en dicha ubicación. Para ello, se añade la siguiente configuración dentro del fichero *configuration.yaml*, utilizando la directica actualizada *allowlist_external_dirs:*

```
homeassistant:
  allowlist_external_dirs:
    - /config/www/capturas
```

Una vez añadida esta directiva, es necesario verificar que la ruta especificada existe físicamente en el sistema, ya que HA valida cada entrada durante el arranque. Tras la comprobación, se procede a reiniciar el sistema asegurando así la correcta aplicación de los cambios.

Una vez habilitada la ruta de almacenamiento, se llevó a cabo una prueba inicial del proceso de captura de imágenes. Para ello, mediante el menú *Herramientas para desarrolladores > Acciones*, se puede ejecutar el servicio *camera.snapshot.*

La prueba puede realizarse tanto desde la interfaz gráfica (IU) como en modo YAML. En ambos casos, se seleccionó el dispositivo correspondiente (en este caso, el ESP32-CAM), y se especificó la ruta completa donde se deseaba guardar la imagen, tal como se muestra en el siguiente ejemplo. Si la operación se ha ejecutado correctamente, el archivo de imagen aparecerá en la carpeta especificada, pudiendo ser visualizado desde el gestor de archivos.

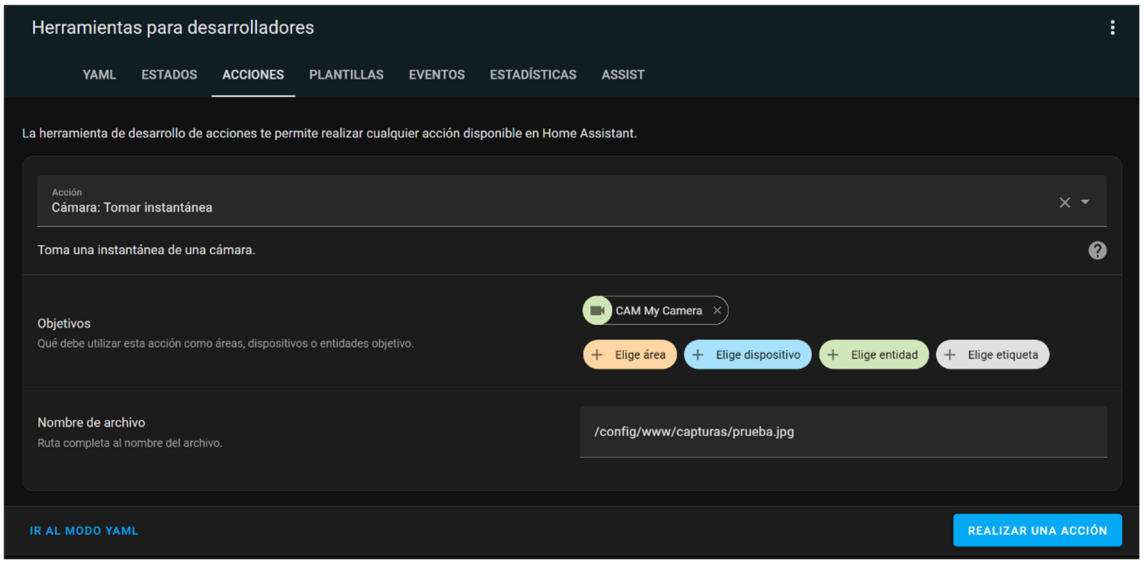

Figura 5.24: Prueba del correcto funcionamiento de toma de instantánea

Para completar el flujo de captura y envío automatizado de imágenes desde el módulo ESP32-CAM, se procede a la creación de una secuencia de acciones dentro del apartado *Configuración > Automatizaciones y escenas > Scripts,* disponible en la interfaz de HA. Esta funcionalidad permite definir flujos de trabajo personalizados que pueden ser invocados de manera directa, desde otras automatizaciones o por eventos externos.

En este caso, se definió un script que encapsula todo el proceso, desde la captura de la imagen hasta su envío a través de Telegram. La secuencia implementada contempla tres pasos esenciales, que garantizan la robustez y el correcto funcionamiento del sistema:

1. **Captura de la imagen desde la cámara:** Se utilizó el servicio *camera.snapshot*, con el identificador de entidad correspondiente al ESP32-CAM. La imagen fue almacenada localmente en la ruta definida previamente, con un nombre de archivo generado de forma dinámica utilizando la fecha y hora del sistema para evitar sobreescrituras accidentales.
2. **Retardo para asegurar la creación del archivo:** A fin de garantizar que la imagen haya sido completamente generada y almacenada antes de intentar acceder a ella, se incluyó un breve retardo en la secuencia. Esta pausa, de 10 milisegundos, se consideró suficiente para evitar condiciones de carrera, especialmente en dispositivos con menor rendimiento o baja carga.
3. **Envío del archivo por Telegram:** Finalmente, se utilizó el servicio *telegram_bot-send_photo* para adjuntar la imagen almacenada en la ruta local y enviarla al chat previamente configurado mediante el bot de Telegram. Dado que se trataba de una imagen generada dinámicamente, se aprovechó la misma plantilla utilizada en la captura para construir la ruta completa del archivo. El identificador del chat fue obtenido previamente durante el proceso de integración del bot.

Mediante esta secuencia, se logró establecer un proceso autónomo, confiable y completamente local para el envío de imágenes a Telegram.

Cabe destacar que el sistema implementado presenta diversas posibilidades de mejora que podrían reforzar su funcionalidad y adaptabilidad a distintos escenarios. Una de las ampliaciones más evidentes consistiría en vincular la ejecución del script a un sensor de movimiento, de manera que la captura de imágenes y su posterior envío se desencadene únicamente ante la detección de presencia, evitando así generar archivos innecesarios y optimizando el uso de recursos.

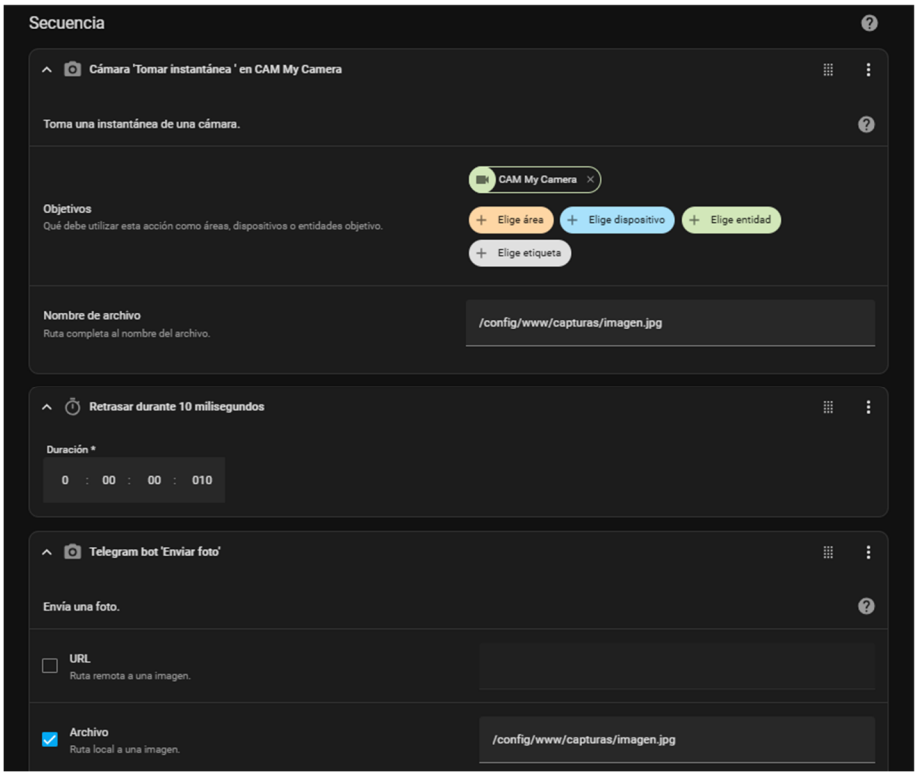

Figura 5.25: Secuencia generada para el envío de imágenes vía Telegram

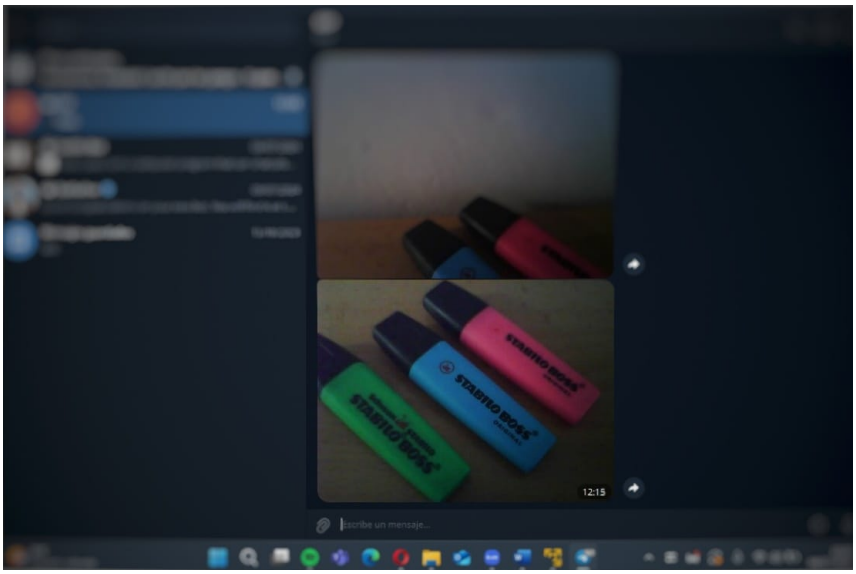

Figura 5.26: Imagen recibida en Telegram

6

Capítulo 6 Conclusiones y líneas futuras

Una vez completado el desarrollo e integración del sistema domótico propuesto, es fundamental reflexionar sobre los resultados obtenidos y extraer las conclusiones más relevantes del trabajo realizado. Este capítulo ofrece una síntesis de los logros alcanzados, los retos encontrados durante la implementación y las decisiones adoptadas para garantizar el correcto funcionamiento del sistema. Además, se presentan posibles líneas de trabajo futuras que permitirían ampliar, optimizar o adaptar la solución desarrollada a nuevos escenarios, tecnologías o necesidades específicas, fomentando así la evolución continua del proyecto.

6.1 Conclusiones

El desarrollo de este sistema ha permitido constatar el enorme potencial de las plataformas domóticas abiertas como Home Assistant, especialmente cuando se combinan con hardware de bajo coste como el ESP32, y protocolos eficientes como MQTT o Matter. A lo largo del proyecto, se ha logrado diseñar, implementar y validar una arquitectura flexible y escalable que permite la monitorización y control de dispositivos en entornos domésticos, haciendo uso tanto de redes locales como de accesos remotos.

Una de las principales fortalezas de esta propuesta radica en su carácter modular. Se ha comprobado que es posible incorporar distintos sensores, placas y métodos de comunicación sin necesidad de rehacer toda la infraestructura, lo cual otorga una enorme ventaja en términos de adaptabilidad y mantenimiento. Además, el uso de tecnologías estandarizadas ha permitido una integración sin fisuras entre componentes heterogéneos, favoreciendo la interoperabilidad entre dispositivos.

En cuanto al uso de MQTT, se ha confirmado su idoneidad para aplicaciones IoT que requieren comunicación ligera y eficiente. El análisis del tráfico generado ha permitido validar que, incluso en escenarios con conectividad limitada o dispositivos con recursos reducidos, el protocolo funciona de forma estable y fiable. La organización jerárquica mediante *topics*, la capacidad de mantener sesiones persistentes, y la gestión de calidad de servicio permiten adaptar su uso a diferentes necesidades y niveles de criticidad.

Por otro lado, el protocolo Matter ha demostrado ser una alternativa prometedora para el futuro de los entornos domóticos, gracias a su enfoque interoperable, seguro y descentralizado. Aunque aún en fases tempranas de adopción, su arquitectura basada en IP, su compatibilidad con IPv6 y su enfoque de seguridad por diseño lo posicionan como una pieza clave en la evolución del Internet de las Cosas. La implementación experimental llevada a cabo en este trabajo ha permitido verificar la viabilidad de flashear dispositivos ESP32 con firmware Matter y emparejarlos exitosamente con HA, sentando las bases para futuras integraciones más complejas.

También ha resultado clave el análisis de las distintas formas de acceso remoto a Home Assistant. Se han evaluado varias alternativas, desde el uso de servicios externos como Nabu Casa hasta configuraciones manuales de router y DNS dinámico. Esta comparación ha evidenciado que, si bien existen soluciones más accesibles en términos de configuración, como Home Assistant Cloud, también es posible lograr un acceso completamente funcional y seguro sin incurrir en costes adicionales, mediante el uso de herramientas como DuckDNS o ESPHome API, siempre que se disponga de los conocimientos técnicos necesarios.

Durante el proceso de integración y pruebas se ha puesto de manifiesto la importancia de una buena gestión de red y de los aspectos de seguridad, tanto a nivel de configuración como de protección de datos. La segmentación de redes, el cifrado de las comunicaciones, el control de acceso y la monitorización de logs se consolidan como aspectos críticos a tener en cuenta en cualquier implementación de domótica conectada.

6.2 Líneas futuras

A partir de la experiencia adquirida en este proyecto, de identifican diversas líneas de trabajo futuras que permitirían continuar ampliando y perfeccionando el sistema implementado:

- **Integración de nuevos dispositivos y sensores:** incorporar sensores de movimiento, cámaras IP, actuadores o módulos de control ambiental, lo que permitiría desarrollar automatizaciones más avanzadas y adaptadas a distintas necesidades domésticas.
- **Implementación de lógica inteligente:** emplear motores de reglas más sofisticados o incluso técnicas de aprendizaje automático para anticipar comportamientos, mejorar la eficiencia energética o adaptar el sistema a las rutinas del usuario.
- **Integración con asistentes virtuales**: aunque se ha mencionado la posibilidad de utilizar plataformas como Google Assistant o Amazon Alexa mediante servicios como Nabu Casa, no se ha desarrollado en profundidad. Explorar esta línea podría facilitar el control por voz del sistema sin interacción directa con la interfaz.
- **Mejora en la gestión de seguridad**: implementar mecanismos más avanzados de autenticación, control de accesos o detección de anomalías contribuiría a fortalecer la protección del sistema frente a posibles amenazas externas.
- **Rediseño de la infraestructura de red**: estudiar el uso de redes Thread en lugar de Wi-Fi para dispositivos con batería permitiría mejorar el consumo energético y aumentar la fiabilidad del sistema en redes malladas.
- **Análisis de rendimiento en condiciones adversas**: evaluar el sistema bajo condiciones de red inestables, alta concurrencia de dispositivos o entornos con interferencias permitiría validar su robustez y realizar ajustes orientados a la resiliencia.
- **Interacción bidireccional mediante Telegram:** una posible ampliación del sistema consiste en habilitar la capacidad de interacción bidireccional a través del bot de Telegram, permitiendo que el usuario no solo reciba notificaciones o imágenes, sino que también pueda enviar comandos específicos al sistema desde la propia aplicación de mensajería.

Estas posibles ampliaciones suponen un punto de partida para futuros trabajos, permitiendo tanto la mejora técnica del sistema como su adaptación a nuevas necesidades del usuario o avances tecnológicos emergentes.

Referencias

[1] Home Assistant, “Home Assistant: Open source home automation that puts local control and privacy first.” [Online].
Available: https://www.home-assistant.io [Accessed: Jun. 4, 2025]

[2] Amazon Web Services, “What is MQTT?” [Online].
Available: https://aws.amazon.com/es/what-is/mqtt/ [Accessed: Jun. 4, 2025].

[3] Paessler, “MQTT explicado.” [Online].
Available: https://www.paessler.com/es/it-explained/mqtt [Accessed: Jun. 4, 2025].

[4] Connectivity Standards Alliance, “Matter overview.” [Online].
Available: https://csa-iot.org/all-solutions/matter/ [Accessed: Jun. 4, 2025].

[5] Espressif Systems, “ESP-Matter Solution.” [Online].
Available: https://www.espressif.com/en/solutions/device-connectivity/esp-matter-solution [Accessed: Jun. 4, 2025].

[6] GitHub, “Matter (Project CHIP) Repository.” [Online].
Available: https://github.com/project-chip/connectedhomeip [Accessed: Jun. 4, 2025].

[7] S. H. Alsamhi, F. Sahdoudi, S. H. Almalki, and Q.-V. Pham, “Matter: A smart home communication standard,” *Smart Cities*, vol. 6, no. 1, pp. 205–218, Mar. 2023. [Online].
Available: https://www.sciencedirect.com/science/article/pii/S2542660523003281
[Accessed: Jun. 4, 2025].

[8] Espressif Systems, “ESP32-WROOM-32 Datasheet.” [Online].
Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf [Accessed: Jun. 4, 2025].

[9] Espressif Systems, “ESP32-WROOM-32D & ESP32-WROOM-32U Datasheet.” [Online].
Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf
[Accessed: Jun. 4, 2025].

[10] Home Assistant, “Home Assistant Cloud.” [Online].
Available: https://www.home-assistant.io/cloud/ [Accessed: Jun. 4, 2025].

[11] Nabu Casa, “Nabu Casa: Seamless Home Assistant integration.” [Online].
Available: https://www.nabucasa.com [Accessed: Jun. 4, 2025].

[12] ESPHome, “ESPHome API Component.” [Online].
Available: https://esphome.io/components/api.html [Accessed: Jun. 4, 2025]