ENTORNO DE PRUEBAS EN UNA RED MESH
TEST ENVIRONMENT IN A MESH NETWORK

TRABAJO FIN DE MÁSTER
CURSO 2023-2024

AUTOR
DANIEL CALATAYUD CRISTÓBAL

DIRECTORES
JOSÉ I. GÓMEZ PÉREZ Y CHRISTIAN T. TENLLADO VAN DER REIJDEN

MASTER EN IOT
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

ENTORNO DE PRUEBAS EN UNA RED MESH
TEST ENVIRONMENT IN A MESH NETWORK

TRABAJO DE FIN DE MASTER EN IOT
DEPARTAMENTO DE ACYA

AUTOR
DANIEL CALATAYUD CRISTÓBAL

DIRECTORES
JOSÉ I. GÓMEZ PÉREZ Y CHRISTIAN T. TENLLADO VAN DER REIJDEN

CONVOCATORIA: SEPTIEMBRE 2024
CALIFICACIÓN: 6

MÁSTER EN IOT
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

19 DE SEPTIEMBRE DE 2024

DEDICATORIA

A todos los que me habéis apoyado en esta loca aventura, elegida en una época difícil.

AGRADECIMIENTOS

A mis directores de proyecto, por su paciencia ante mi ajetreada vida y desapariciones temporales, que ha hecho complicado el avance del trabajo.

A mis hijos y esposa, por la comprensión ante la maraña de cables y SOCs que se han llegado a hacer un sitio en nuestra vida, llegando a tener un espacio propio en la maleta de las vacaciones.

RESUMEN

Entorno de pruebas en una red mesh

Este trabajo analiza las opciones para establecer funcionalidades que permitan evaluar el rendimiento y estabilidad de una red mesh generando situaciones hipotéticas de carga o escalabilidad que puedan llegar a producirse.

Como herramienta para conseguir el objetivo será necesario implementar un mecanismo que permita interactuar con los nodos de la red, siendo deseable que esto se realice desde un dispositivo externo a la propia red y sin que dicha interacción sea intrusiva con el código de aplicación que se desee testear o monitorizar.

Palabras clave

Red, mesh, testing, administrar, resiliencia, eventos, espressif, rendimiento

ABSTRACT

Test environment in a mesh network

This work analyzes the options to establish functionalities that allow evaluating the performance and stability of a mesh network in the face of hypothetical workload or scalability situations that may occur.

As a tool to achieve the objective, it will be necessary to implement a mechanism that allows interaction with the nodes of the network, it being desirable that this be done from a device external to the network itself and without said interaction being intrusive with the desired application test or monitoring code.

Keywords

Network, mesh, testing, manage, resilience, events, expressif, performance

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1 - Introducción

Con la miniaturización de la electrónica y la proliferación de dispositivos inalámbricos de diferente índole, en las últimas décadas ha existido un boom en la utilización de las redes inalámbricas. Es importante conocer que existen diferentes redes inalámbricas en función a su alcance, en ese sentido podemos identificar entre:

- Redes inalámbricas de área personal (WPAN), son redes de corto alcance utilizadas para eliminar cables entre dispositivos próximos, normalmente en un uso de cliente/servidor. Estas redes suelen utilizar tecnologías como el bluetooth, infrarrojos o Zigbee.
- Redes inalámbricas de área local (WLAN), son redes normalmente privadas con unos cientos de metros de alcance. Actualmente son compatibles entre sí al imponerse la marca Wireless Fidelity (WIFI) del consorcio WECA como estándar de facto. Hay que reseñar que esta implementada sobre el estándar IEEE 802.11, que a su vez fue diseñado para sustituir la norma 802.3 definido para la Ethernet. Como resultado una red WIFI además de permitir la conexión de dispositivos de cualquier fabricante, es compatible o interoperable con una red cableada.
- Redes inalámbricas de área extendida (WWAN), son redes de un alcance muy extenso, inclusive a nivel internacional o global. Por ejemplo, las comunicaciones telefónicas han ido ampliando los servicios ofrecidos conforme se sucedían las diferentes generaciones tecnológicas, desde el 2G con GPRS y SMS para servicios de voz y mensajes cortos al actual 5G con velocidades ya probadas 3.6Gbits gracias a la banda milimétrica (mmWave). Sus especificaciones incluyen velocidades teóricas de 20 Gbps de descarga y 10 Gbps de subida, con una latencia de 4 ms. Esta tecnología se espera implantar para 2025, lo cual supondrá una revolución en el ámbito de las comunicaciones en tiempo real entre dispositivos.

1.1 Redes inalámbricas dinámicas

El número creciente de dispositivos moviéndose en espacios relativamente pequeños con necesidad de cobertura constante y el auge de la interoperabilidad de dispositivos, han creado una necesidad de soportar topologías cambiantes y dinámicas, lo cual tensiona las topologías de red centralizadas y estáticas. Lo que deja camino a la evolución de las redes ad-hoc o dinámicas, caracterizadas por no depender de infraestructura preexistente o puntos de acceso a una red administrada. En este tipo de redes los nodos disponen de conexiones directas entre ellos, enrutando la información de la red, de forma que cada nodo hace las veces de maestro y esclavo en simultáneo.

Este tipo de redes habitualmente actúan dentro del ámbito de las WLAN mediante tecnología WIFI, aunque existen posibilidades de ser utilizadas con tecnología Bluetooth ya que permiten ampliar considerablemente el rango de cobertura de la red.

1.2 Redes Mesh

Como una tipología de red ad-hoc, aparecen las redes mesh como un tipo de red híbrida, que dispone de puntos de acceso análogos a los de una red tradicional, pero su estructura o fishbone es distribuida. Nos facilita la conexión entre dispositivos o acceso a una red externa en entornos inestables o cambiantes, proporcionándonos siempre una conexión o cobertura óptima con un coste de conexión relativamente pequeño.

Pero mantener la estabilidad en entornos cambiantes normalmente viene en detrimento de la velocidad de transmisión end to end de los paquetes.

El pilar básico que permite cumplir el objetivo de mantener la estabilidad en la red descansa sobre la característica principal de las redes ad-hoc: la conexión directa entre los nodos que conforman la red.

Esto hace que los nodos busquen los caminos de comunicación entre ellos y exista redundancia de enlaces, lo que ayuda en la resiliencia, facilita la adaptación a los cambios de topología y obtener caminos de comunicación alternativos. Es muy difícil que la red completa se caiga, y por tanto permite tener una alta disponibilidad de forma nativa.

Sin embargo, estas ventajas tienen a su vez ciertas desventajas asociadas. Por ejemplo: los protocolos de enrutamiento son más complejos, sobre todo los dinámicos, y el envío de paquetes hay momentos en los que no son tan óptimos con respecto a los recursos de red consumidos.

Además, los nodos tienen cierta sobrecarga, al deber de tener el rol de cliente y de servidor.

También suelen ser frecuentes los retardos o las colisiones en el envío de paquetes, sobre todo si coincide con una re-configuración de la topología de red debido a la aparición o desaparición de un nodo en la red o al realizar el envío de un mensaje a un grupo de nodos o a la red completa.

1.3 Protocolos de enrutamiento en redes dinámicas

El enrutamiento es una de las partes más importantes dentro de una red dinámica o hibrida. Sobre todo, teniendo en cuenta que al final en las redes dinámicas existe competencia entre los nodos para transmitir la información.

La implementación de un protocolo de enrutamiento de una red mesh podría agruparse en una de las siguientes categorías:

- Enrutamiento proactivo, los nodos realizan envíos de información sobre sí mismos y la red de forma periódica. Esto permite a cada nodo conocer toda la red o al menos los destinos alcanzables por cada uno de sus vecinos. El enrutamiento, se decide a partir de esta información. Los cambios en la topología deben ser informados al resto de la red, por lo que normalmente se utilizan para entornos poco cambiantes.
- Enrutamiento reactivo, las rutas en estos protocolos se construyen bajo demanda, por lo que cuesta más establecer la conexión. Sin embargo, se adaptan bien a los cambios de topología, favoreciendo una mejor escalabilidad de la red.
- Enrutamiento híbrido, este como su nombre indica es una mezcla de ambos, y suele utilizarse cuando las debilidades de los dos anteriores son muy acusadas.

Los protocolos de enrutamiento utilizan diferentes métricas para establecer la calidad de la conexión, entre las diferentes métricas que se pueden utilizar algunas de ellas serían: el número de saltos, el retardo, la estabilidad del enlace, el ancho de banda, la perdida de paquetes, el ETX[1] o el RTT[2].

En definitiva, los diferentes protocolos cumplirán mejor o peor en base a las necesidades que vayamos teniendo. Y es que por regla general al potenciar una característica se suele mermar otra. Por ejemplo, existen implementaciones que optimizan la fiabilidad, es decir son protocolos que facilitan una rápida reconfiguración de la red o dan seguridad en el envío de mensajes. Pero para conseguirlo se suele consumir mayor ancho de banda con comunicación de control de la red y por tanto sobrecargar a los propios nodos, de forma que probablemente penalicemos la escalabilidad.

Otro punto importante sería la calidad del servicio, es decir que los mensajes sean correctamente enrutados de la forma óptima, para ello es importante elegir el parámetro que mejor defina las métricas de conexión en el contexto de nuestra red.

Por último, podríamos desear que la conexión entre los diferentes nodos de la red por parte de los clientes sea lo más transparente posible, intentando que la transición de la conexión de un nodo a otro sea muy ligera.

1.4 Estándar IEEE 802.11s

Esta norma nace como Grupo de Estudio (Study Group) del IEEE 802.11 en 2003, se convierte en Grupo de Trabajo (Task Group) en Julio de 2004, para terminar siendo el estándar para redes WIFI mesh.

Amplia el estándar IEEE 802.11 que define las redes inalámbricas. En concreto esta norma define una arquitectura y un protocolo que soporta broadcast/multicast y el

---

[1] Expected Transmission Count, métrica para enrutamiento en redes MANET que predice el número de transmisiones necesarias para el envío de un paquete a través de un enlace.

[2] Round Trip Time, métrica sobre el tiempo medio que tarda un paquete en transmitirse dentro de un enlace.

reenvío de tramas y selección de camino en el nivel 2 del modelo OSI, es decir a nivel de enlace de datos.

La arquitectura define una serie de tipos de nodos que conforman la red y que según su tipología tendrán una serie de responsabilidades dentro de esta.

- **Station (STA):** Son nodos pertenecientes a equipos externos a la red que no implementan el protocolo.
- **Mesh Point (MP)**. Nodos de comunicación, vertebran la red sirviendo como pasarela a los mensajes y extendiendo la cobertura.
- **Mesh Access Point (MAP)**. Son nodos de tipo Mesh Point que además permiten la conexión de clientes o de nodos Station.
- **Mesh Portal Point (MPP)**. Pasarela entre la red mesh y otra red exterior

El protocolo implementa por defecto un enrutamiento HWMP, y aunque puede utilizarse otras implementaciones, esta debe estar presente para que funcione.

Existen múltiples implementaciones de la norma, por ejemplo:

- Open 802,11s [1], desarrollada por un consorcio de empresas entre las que destacan Nortel, Google, OLPC y Cozybit.
- Driver para los ordenadores XO de la fundación One Laptop Per Child (OLPC) [2], esta fundación desarrolló un driver que permite a los portátiles hacer de MP inclusive si el portátil esta en stand by.
- Wifi Mesh de FreeBSD, [3] la comunidad de FreeBSD desarrolló la posibilidad de establecer redes mesh a partir para la distribución FreeBSD[3] a partir de su versión 8.

1.4.1 Protocolo de enrutamiento HWMP

El Hybrid Wireless Mesh Protocol es el protocolo de enrutamiento hibrido que el estándar IEEE 802.11s provee por defecto. Tal y como define IEEE 802.11s, está

[3] Sistema Operativo de la familia Unix.

implementado en el nivel 2 de ISO, en la capa de enlace. Esto significa que genera sus cálculos del enrutamiento a partir de la dirección MAC y hace que la red se comporte hacia las capas superiores como una red local estándar (LAN 802.x). Sin embargo, imposibilita el uso de servicios ofrecidos por las capas superiores, como los ofrecidos por la capa de direccionamiento: la interconexión de subredes o el uso de la jerarquía de direccionamiento, por lo que para redes grandes la gestión se vuelve compleja y suele necesitar combinarse con otros protocolos de enrutamiento.

Los nodos disponen de un árbol de enrutamiento proactivo, aunque el descubrimiento de rutas se realiza de forma reactiva basándose en el algoritmo del protocolo AODV. Ambos modos pueden usarse de forma simultánea y su uso depende de la configuración existente en los MP.

Utiliza la métrica de coste del enlace para las decisiones de enrutamiento. Este coste se calcula al establecer el enlace, y para calcularlo utiliza: la tasa de pérdida de tramas, el ancho de banda del enlace y la sobrecarga del canal y el protocolo. Los costes son almacenados en el árbol de enrutamiento como vectores de coste desde el nodo raíz. Para propagar esta información sobre la métrica de cada enlace, se usa un campo en los mensajes propios que intercambia el protocolo, en concreto: PREQ, PREP y RANN

Permite disponer de múltiples nodos raíz, por lo que puede utilizarse esta configuración para marcar los nodos de salida a otras redes como root y construir arboles de rutas hacia estos.

1.5 Motivación

El auge de las redes P2P y del intercambio de información en bloques, ha atraído muchas miradas hacia el funcionamiento de las redes ad hoc.

En el ámbito de los entornos de intercambio de información entre dispositivos IoT, las redes mesh son muy populares debido a su versatilidad y a la autogestión que estas proporcionan. Sin embargo, como puede verse en la Figura 1: Capas que recorre una trama a su paso por una red mesh con ESP_WIFI_MESH, los *frameworks* de desarrollo

suelen optar por ocultar a la capa de aplicación lo que sucede en el camino que recorre un paquete y en definitiva lo que sucede dentro de la red mesh, que termina funcionando como una especie de router entre dos puntos externos que quieren comunicarse. Con ello la comunicación en el interior de la red mesh se convierte en una caja negra que se autogestiona y toma decisiones sin dar demasiada información de su lo que sucede.

Un ejemplo real de esta situación sería el de una compañía que dispone de una red wifi mesh con nodos ESP32 que miden la curva I-V de los paneles solares que tiene instalados para determinar si estos necesitan mantenimiento o no. Han comprobado que cuando la red supera cierto tamaño, la red se desestabiliza y la información de los nodos se pierde. Sin embargo, son incapaces de encontrar el origen del problema al no poder monitorizar el camino real que siguen los mensajes o donde se pierden.

Esta opacidad de los frameworks en mostrar las métricas que manejan los protocolos de enrutamiento o del estado de los enlaces, dificulta a los desarrolladores o a los administradores de red conocer aspectos tan importantes como: el grado de estabilidad de la red, la elasticidad[4], la resiliencia o predecir los umbrales de tolerancia de carga de transmisión. La realidad es, que incluso la mera monitorización de lo que sucede dentro de la red *mesh* se puede tornar harto complicada.

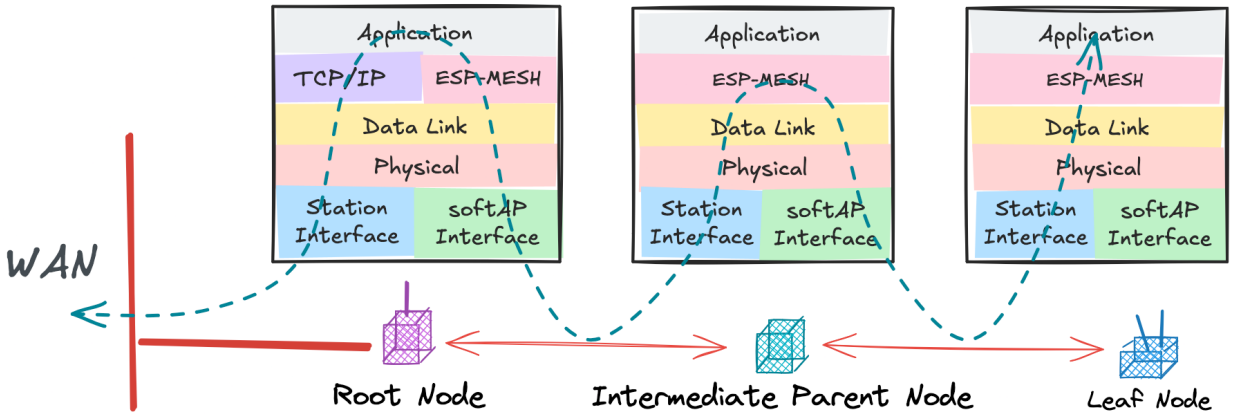


Figura 1: Capas que recorre una trama a su paso por una red mesh con ESP_WIFI_MESH

[4] Capacidad de la red para cambiar de tamaño en cortos periodos de tiempo para adaptarse a las necesidades de carga de un momento concreto.

Si Netfllix tuviese que implantar una red mesh con ESP_WIFI_MESH, no podría seguir su filosofía de ingeniería del caos[5] para garantizar la resiliencia del sistema. Dicho de otra forma, le hubiese sido complicado desarrollar su famoso Chaos Monkey[6].

La implantación de una red mesh está llena de incertidumbres sobre su desempeño final, pudiendo llegar a casuísticas en las que la red no responda correctamente y no se sea capaz de visualizar los motivos. Tampoco es posible predecir los comportamientos de un sistema hasta no implantarlo completamente o suceda la casuística concreta.

Este trabajo pretende ser una muestra sobre la necesidad de disponer de una base sobre la que poder desarrollar módulos que permitan operar la red sin tener que interferir en el desarrollo de los aplicativos.

1.6 Objetivos

Este trabajo hace una investigación en el ecosistema de productos de *Espressif* sobre las opciones existentes para incluir una capa separada de la lógica de aplicación que permita: el testeo, la administración y la monitorización de una red mesh.

Como puede verse en la Figura 2, el objetivo principal de la investigación es brindar a los desarrolladores y operadores de red una interfaz sobre la que desarrollar módulos de administración en los aplicativos, que permitan caracterizar y monitorizar la red, permitiendo cuantificar y verificar los problemas existentes antes de que estos sucedan.

---

[5] Chaos Engineering Técnica que implanta modelos de perturbación en producción para verificar la resiliencia de un sistema de software.

[6] Librerias desarrolladas por Netflix que fuerzan fallos en los nodos de la red con una aleatoriedad controlada para verificar la resiliencia de los aplicativos distribuidos que desarrolla:

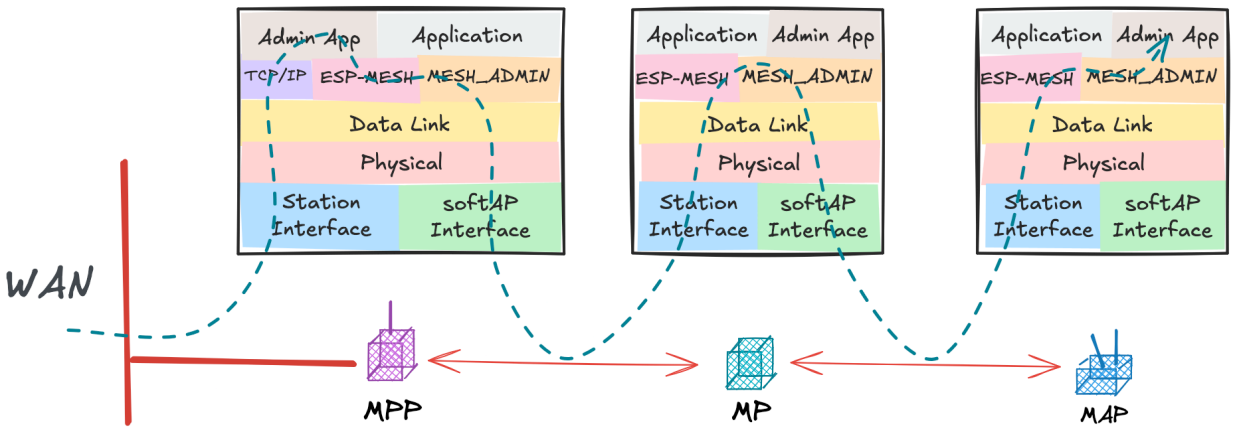


Figura 2: Envío de un hipotético mensaje de administración.

Esta capa de servicio de administración brindaría la gestión de la recepción y la notificación de la llegada de mensajes de administración y testeo. Y los mecanismos para que estos quedasen fuera del ámbito y del alcance de la aplicación.

Esta abstracción de administración alojada en la capa de servicio debe permitir o delegar la definición de los mensajes y sus consecuencias a los desarrolladores. Permitiéndoles desde definir e implementar comandos que modifiquen el comportamiento de uno o de un grupo de nodos para poder seguir una filosofía de ingeniería del caos a mensajes que soliciten o faciliten la recolección de información específica de la comunicación, como la latencia de los mensajes o el estado del nodo.

Como objetivo secundario, se busca que la solución, aunque pueda ser usada desde la capa de aplicación, este integrado en una capa de servicio. De esta forma sería lo menos intrusiva posible respecto al software de aplicación desarrollado, intentando que su coste de ejecución y memoria impacte lo menos posible en el rendimiento de la aplicación cuando esta se encuentre en un entorno productivo.

1.7 Plan de trabajo

La realización del trabajo ha ido intercalando los aspectos teóricos con las pruebas de concepto prácticas. Sin embargo, podrían agruparse conceptualmente en varias fases, con hitos de finalización y objetivos diferenciados.

1.7.1 Investigación de redes mesh y ecosistema tecnológico

La investigación parte de conceptos genéricos, centrándose en el ecosistema tecnológico provisto por *Espressif* para interactuar con redes *mesh*. Con ello, por un lado se ha obtenido un mayor contexto sobre la problemática y por otro se ha acotado la posible solución para conseguir los objetivos perseguidos.

Inicialmente se tienen en cuenta dos posibilidades ofrecidas por *Espressif*:

- El módulo ESP_WIFI_MESH del SDK ESP_IDF, que es el utilizado durante las prácticas del máster.
- El framework de desarrollo para entornos mesh ESP_MDF

Posteriormente durante el análisis del código fuente se destapa que el framework ESP_MDF acaba de ser descontinuado en favor de una nueva solución llamada ESP_MESH_LITE, sobre el que todavía no hay mayor documentación que la existente en el repositorio de código. Este hallazgo supone la inclusión de dicha solución en el estudio.

1.7.2 Análisis en profundidad de los frameworks o módulos de Espressif

La segunda etapa ha tenido que ver con un estudio en profundidad y el diseño teórico de las posibles soluciones. Para realizar esta tarea se ha analizado el código fuente de las diferentes soluciones o frameworks disponibles en *Github*.

Una de las características más sorprendentes encontrada, es que, aunque el ecosistema en su conjunto se vende como Open Source bajo licencias Apache 2.0, como puede verse en Figura 3: Tipos de licencias en ESP-IDF, si es mayoritariamente *Open Source,* no lo es en su totalidad.

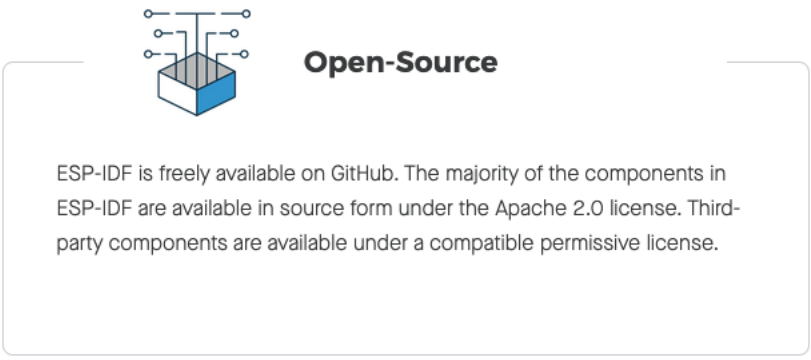


Figura 3: Tipos de licencias en ESP-IDF

De hecho, gran parte de la implementación del core no está disponible. En lo referente al módulo de wifi mesh, sucede lo mismo con la implementación del protocolo de intercambio de mensajes y aprovisionamiento.

1.7.3 Implementación incremental de una red mesh con ESP_WIFI_MESH

Se realiza un primer aplicativo sobre el módulo de WIFI_MESH del SDK proporcionado por Espressif ESP_IDF por tener como apoyo los distintos trabajos y ejercicios entregados para el máster en la asignatura de Redes Protocolos e Interfaces. Se toma como base los ejemplos proporcionados por Espressif llamados "*internal_communication*" y "*ip_internal_network*".

En este aplicativo el nodo root envía un paquete en modalidad broadcast al resto de nodos de una red mesh, que están leyendo periódicamente los mensajes y mostrándolos por consola.

La estructura del mensaje enviado es la siguiente:

```
typedef struct {
    uint8_t id_pkg;
    char * msg_txt;
} msg_t;
```

- id_pkg: Identificador al mensaje, permitirá trazar los mensajes enviados son recibidos correctamente y verificar que la información recibida es correcta

- msg_txt contiene un texto con una base constante “Hola mundo:” y un número aleatorio para que el texto cambie para cada mensaje enviado.

Se prevé que este programa pueda usarse de base para probar el envío de mensajes del sistema.

1.7.4 Selección de una opción e implementación de una PoC

Se decide realizar la implementación de la PoC para el módulo de ESP_WIFI_MESH. Esta solución tiene como hándicap ser intrusiva de cara a los aplicativos, ya que hay que construir una capa sobre la interfaz de lectura de mensajes proporcionada por ESP_WIFI_MESH.

Sin embargo, el resto de las opciones están descontinuadas o demasiado verdes para ser usadas. Además, están construidas sobre ESP_WIFI_MESH o son compatibles, por lo que es la opción más versátil y que aporta mayor posibilidad de reutilización.

Capítulo 2 - Requisitos generales de la arquitectura

Para llegar a cumplir los objetivos sea cual fuese el framework o librería seleccionada, es necesario disponer de una serie de funcionalidades en los nodos que compongan o que estén conectados a la red.

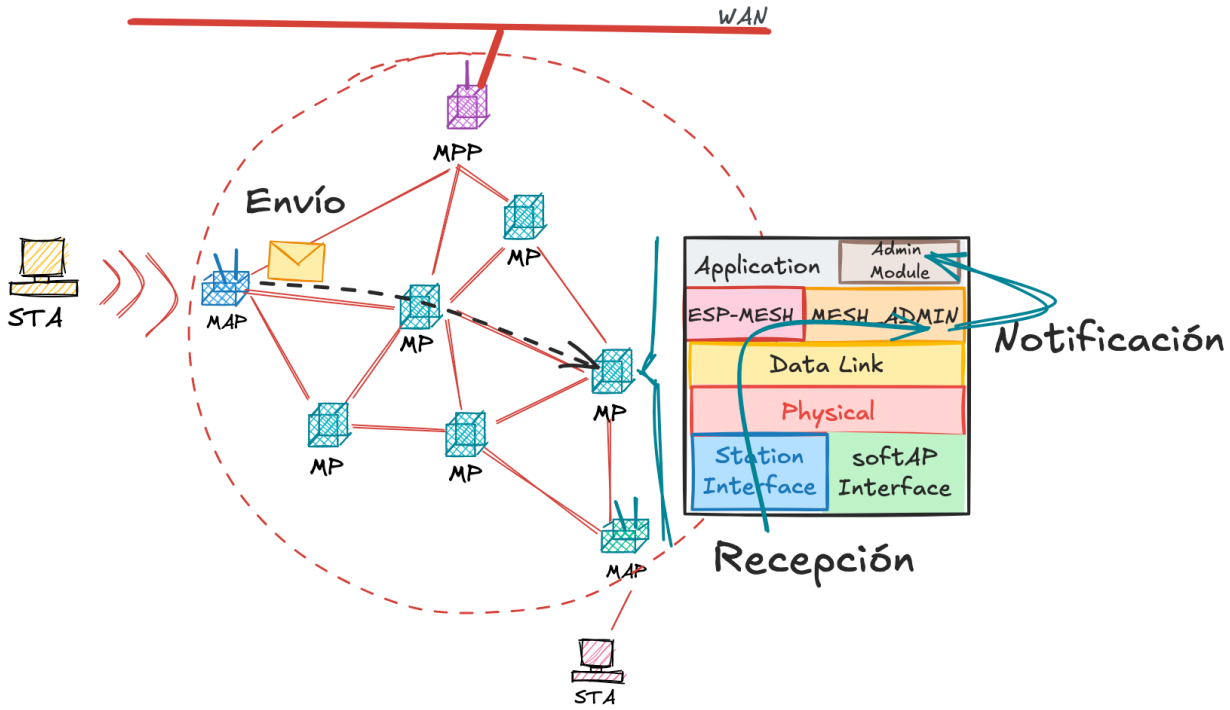


Figura 4: Funcionalidades básicas del sistema

Tal y como puede verse en la Figura 4: Funcionalidades básicas del sistema, para cumplir con los objetivos y poder interaccionar con los nodos dentro de la red es necesario poder realizar envíos de paquetes con información relativa a la acción que se desee.

Además, el framework tiene que ser capaz de identificarlos y recibirlos sin que exista interferencia en el buffer de mensajes utilizado por la aplicación. Es decir que el framework sea capaz de leerlos y notificar su llegada sin que el aplicativo sea consciente de que ha existido dicha comunicación.

2.1 Envío de comandos

Para realizar cualquier interacción de administración o monitorización con la red mesh, es necesario poder interactuar con ella o con los nodos que la componen desde el exterior.

En ese sentido se debe poder enviar comandos a nodos concretos o a un grupo de ellos para que estos reaccionen conforme nos interese. Por ejemplo, en casos de uso de testeo de la red: estableciendo tiempos muertos de inactividad para simular una saturación, aumentando el envío de mensajes a un nodo concreto o a un grupo de estos para aumentar el tráfico de red en una parte concreta de esta o reiniciando un nodo para verificar la resiliencia.

2.2 Recepción de mensajes

La red debe reaccionar a los comandos, por lo que cualquier nodo o cualquier grupo de estos debe leer los comandos enviados.

La lectura debe realizarse con la premisa de ser transparente de cara al desarrollo de aplicaciones. Por ello este apartado debe cumplir las siguientes premisas:

1. Que las comunicaciones necesarias para la interacción con la red no compitan con las de los aplicativos.
2. Poder disponer de las funcionalidades de control y monitorización sobre la red sin tener que modificar los aplicativos ya desarrollados.
3. Poder mantener las funcionalidades en stand by sin que suponga una sobrecarga para el aplicativo.

Existen dos opciones para abordar estas premisas:

- Contar con canales específicos de comunicación
- Identificar y filtrar los mensajes asociados con estos servicios del resto de mensajes en una capa previa.

En caso de tener que filtrar los mensajes por no poder contar con canales específicos, lo ideal es poder identificar y manipular el mensaje en una capa previa y de la forma óptima posible, en este caso sin tener que analizar el payload.

Introducir una cabecera brinda múltiples opciones para abordar la problemática del filtrado: hacer split de los mensajes en buffer diferentes sin leer el contenido, un prefiltro que extraiga los mensajes del buffer, …

Por otro lado, la lectura debe estar alineada con el envío de mensajes en cuanto a la formula escogida, ya sea un canal de comunicación alternativo o un prefiltrado de mensajes.

2.3 Notificar la llegada de comandos

Una vez se han extraído los comandos, hay que notificar su llegada a la capa de aplicación, de forma que el nodo en concreto pueda realizar las acciones que se esperan.

Este módulo deberá apoyarse en las funcionalidades que IDF nos ofrece y será análogo a cualquiera de las implantaciones o frameworks disponibles en Espressif

Capítulo 3 - Integración en ESP_WIFI_MESH

ESP-WIFI-MESH es un módulo del Espressif IoT Development Framework o ESP-IDF que puede entenderse como un protocolo de red implementado sobre el protocolo de WIFI.

3.1 Arquitectura actual

Como ilustra la Figura 5: Arquitectura de componentes de ESP-WIFI-MESH está construido encima del driver WiFI/RTOS, por lo que en casos específicos tiene a disposición la pila LwIP y la potestad de utilizar la pila de mensajes definidos en WIFI, además de la propia de WIFI-MESH. Sin embargo, para entornos de redes mesh autoorganizados no es recomendable su uso porque puede interferir en los procesos internos, por lo que el interfaz WIFI (y por tanto el acceso LwIP) debe permanecer deshabilitado en los nodos.

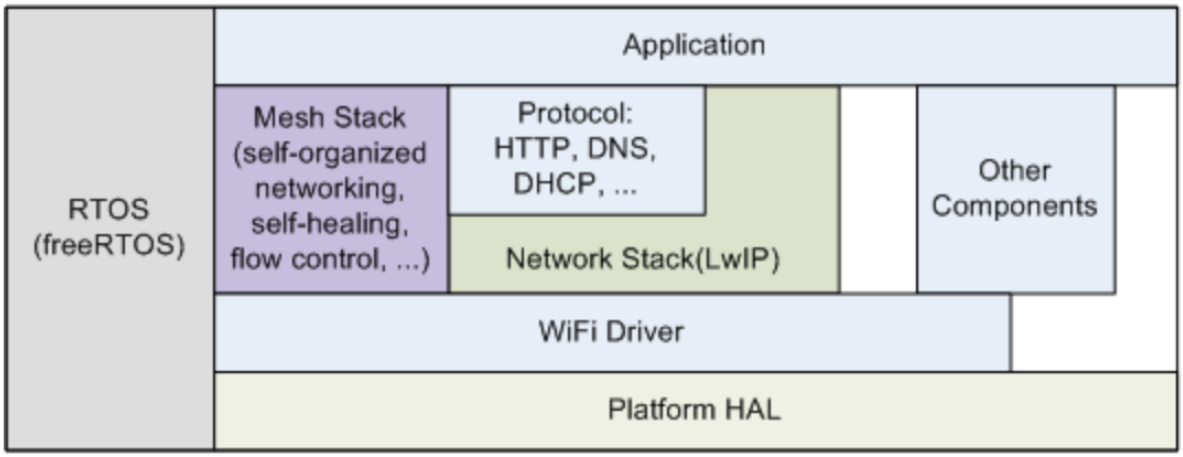


Figura 5: Arquitectura de componentes de ESP-WIFI-MESH

3.1.1 Tipología de nodos

El protocolo ESP_WIFI_MESH [4] esta construido sobre el protocolo WI-FI por la empresa Espressif como un componente de ESP-IDF [5], que es el framework oficial de desarrollo IoT que dicha empresa proporciona para la implementación en C o C++ sobre SoCs de la serie ESP32.

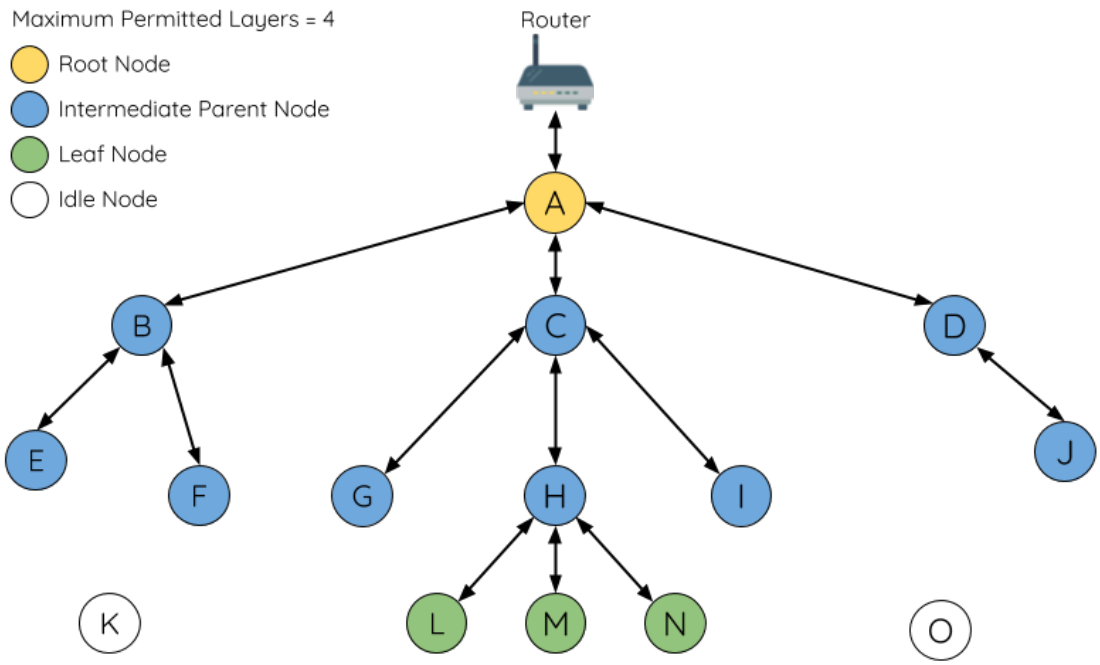


Figura 6: Tipos de nodo y tipología de ESP-WIFI-MESH

Para la composición de la red mesh, y como podemos ver en la Figura 6: Tipos de nodo y tipología de ESP-WIFI-MESH se definen los siguientes tipos de nodos:

- **Nodo raíz**: Es el nodo superior de la red y sirve como la única interfaz entre la red ESP-WIFI-MESH y una red IP externa. El nodo raíz está conectado a un enrutador Wi-Fi convencional y transmite paquetes hacia/desde la red IP externa a los nodos dentro de la red ESP-WIFI-MESH. Solo puede haber un nodo raíz dentro de una red ESP-WIFI-MESH y la conexión ascendente del nodo raíz solo puede ser con el enrutador.
- **Nodos de hoja**: Serían los equivalentes a los nodos MAP, es un nodo al que no se le permite tener nodos secundarios, es decir conexiones descendentes. Por lo tanto, solo puede transmitir o recibir sus propios paquetes, pero no puede reenviar los paquetes de otros nodos. Los nodos situados en la capa máxima permitida de la red o sin una interfaz softAP (solo estación) también se asignarán como nodos hoja.

- **Nodos primarios intermedios**: Son los nodos conectados que no son ni el nodo raíz ni un nodo hoja. Deben tener una sola conexión ascendente (un solo nodo principal), pero puede tener de cero a varias conexiones descendentes (de cero a varios nodos secundarios). Por lo tanto, debe transmitir y recibir paquetes, pero también reenviar paquetes enviados desde sus conexiones ascendentes y descendentes.
- **Nodos inactivos**: Son los nodos que aún no se han unido a la red, intentarán formar una conexión ascendente con un nodo principal intermedio o intentarán convertirse en el nodo raíz.

3.1.2 Topología de red

El protocolo utiliza el concepto WI-FI para crear la red con una estructura de árbol. En Wi-Fi, las estaciones están limitadas a una sola conexión con un AP (conexión ascendente) en cualquier momento, mientras que un AP se puede conectar simultáneamente a varias estaciones (conexiones descendentes).

Por tanto, nos debemos imaginar cada nodo como una red WIFI, y por tanto la red completa como un conjunto de estas. Digamos que puede considerarse como un protocolo de red que combina muchas redes Wi-Fi individuales en una sola WLAN.

Lógicamente como se espera en una red mesh, ESP-WIFI-MESH permite que los nodos actúen simultáneamente como una estación y un AP. Por lo tanto, un nodo en ESP-WIFI-MESH puede tener múltiples conexiones descendentes usando su interfaz softAP, mientras que simultáneamente tiene una única conexión ascendente usando su interfaz de station, generándose la estructura de árbol antes citada.

3.1.3 Enrutamiento

Cada nodo dentro de una red ESP-WIFI-MESH mantendrá su tabla de enrutamiento individual. Esta constará de las direcciones MAC de todos los nodos dentro de la subred del nodo en particular, incluida la dirección MAC del propio nodo.

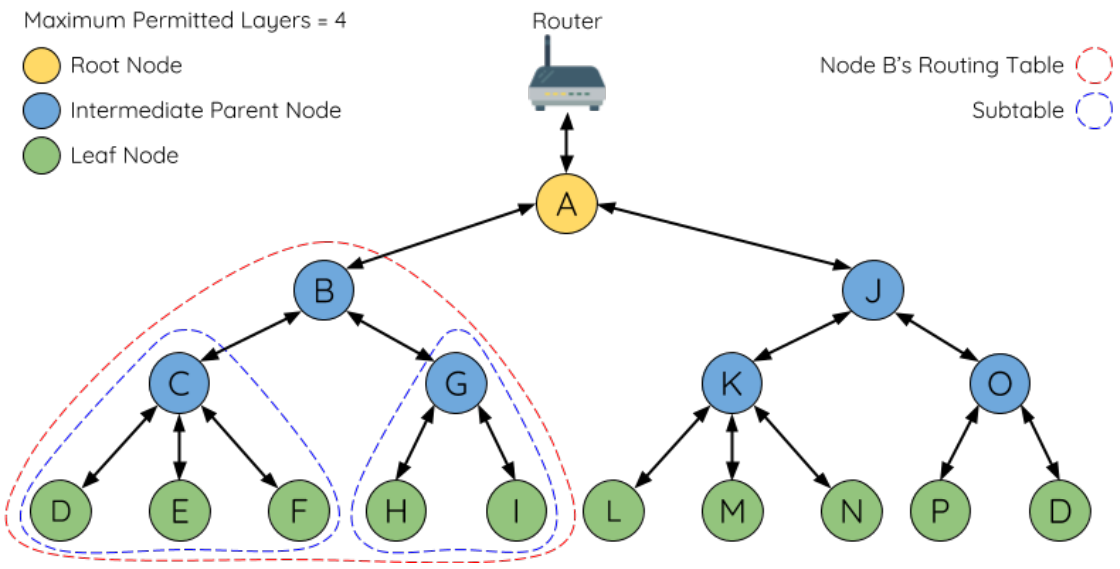


Figura 7: Tablas y subtablas de enrutamiento

Como puede verse en la Figura 7: Tablas y subtablas de enrutamiento, cada tabla de enrutamiento se divide internamente en varias subtablas y cada subtabla corresponde a la subred de cada nodo secundario.

El protocolo de enrutamiento usa las tablas para determinar si un paquete debe reenviarse en sentido ascendente o descendente según las siguientes reglas:

1. Si la dirección MAC de destino del paquete está dentro de la tabla de enrutamiento del nodo actual y no es el nodo actual, se selecciona la subtabla que contiene la dirección MAC de destino y reenvía el paquete de datos en sentido descendente al nodo secundario correspondiente a la subtabla.

2. Si la dirección MAC de destino no está dentro de la tabla de enrutamiento del nodo actual, reenvía el paquete de datos en sentido ascendente al nodo principal del nodo actual. Si el paquete llega al nodo raíz, este direccionará al nodo, al disponer de la tabla de enrutamiento con todos los nodos de la red.

3.2 Arquitectura de integración planteada

La arquitectura de integración planteada se adapta fácilmente al envío de mensajes disponible, haciendo uso de la infraestructura proporcionada por ESP-WIFI-MESH mediante el método esp_mesh_send, de la interfaz de esp_wifi.

Para facilitar el intercambio de mensajes, se propone una estructura base sobre la que enviar los mensajes.

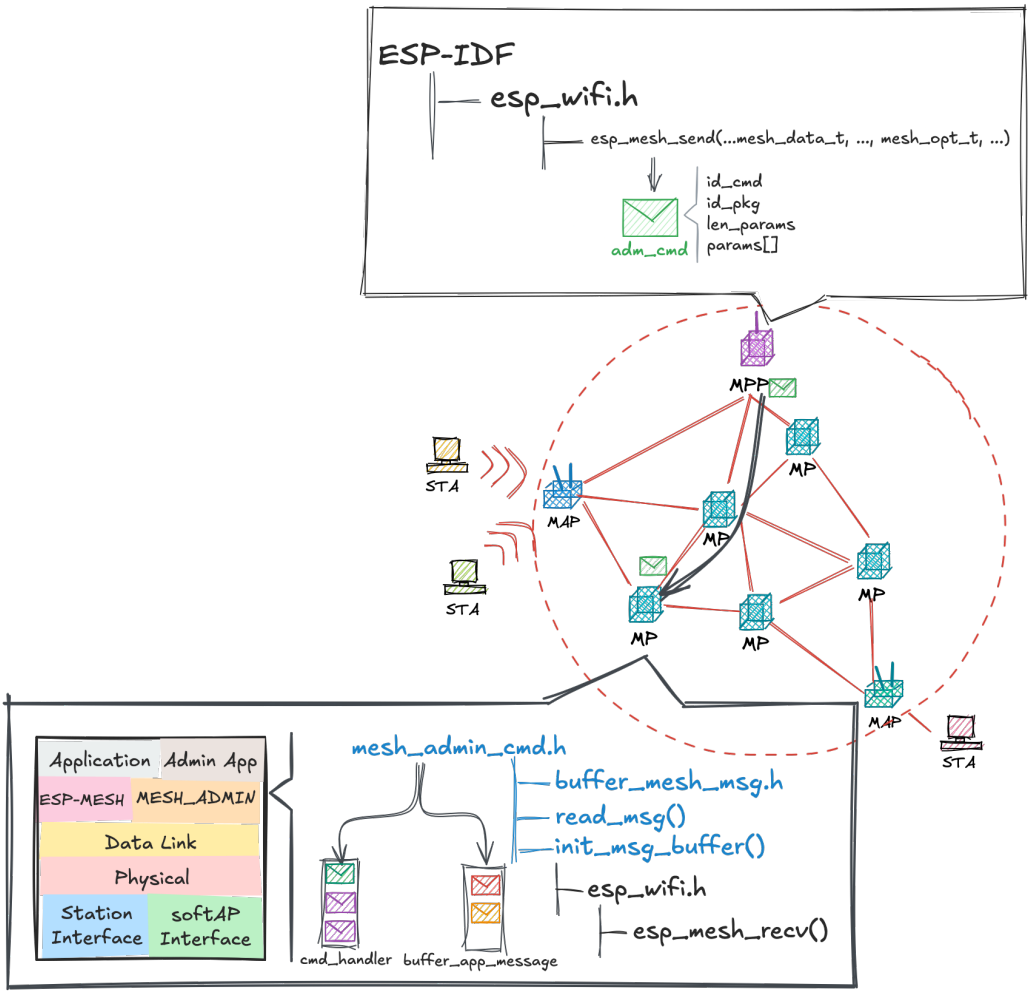


Figura 8: Mapa de interfaces de integración con ESP-WIFI-MESH

Como podemos ver en la Figura 8, durante la recepción de mensajes, hay que incluir un wrapper de los métodos de lectura proporcionados por ESP-WIFI-MESH. De esta forma el módulo desarrollado mesh_admin_cmd tomará el control de los mensajes entrantes, de forma que pueda filtrar los mensajes de administración y testing.

3.2.1 Envío de comandos

Por las características de los nodos intermedios de una red mesh autoorganizada desarrollada con ESP-WIFI-MESH, en la que se recomienda evitar tener los nodos intermedios con el interfaz WIFI habilitada (pila de mensajes, LwIP, …), se descarta la posibilidad de utilizar canales alternativos para el envío de información entre los nodos de la red y se utilizará la comunicación WIFI-MESH provista.

Como se puede verse en la Figura 9, los paquetes de transmisión son propietarios y viajan íntegramente dentro del payload de un frame de datos WI-FI.

Sin embargo, dichos paquetes sólo llegarán a la capa de aplicación en el origen y en el destino. La transmisión de un paquete que tenga que pasar por varios nodos dentro de la red para llegar a destino será encapsulado en diferentes tramas WI-FI, aunque el paquete será el mismo en todos los saltos.

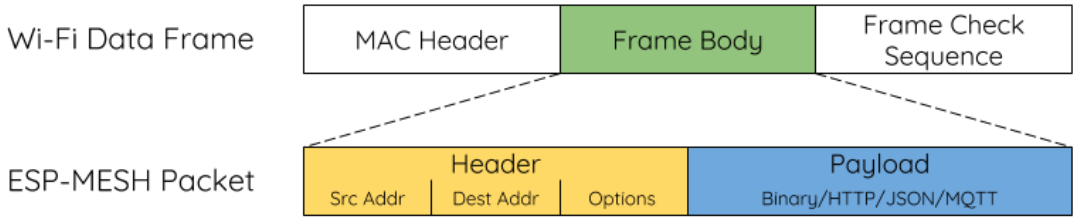


Figura 9: Estructura de un paquete ESP-WIFI-MESH

El encabezamiento, parte que nos interesa, se compone de: la dirección origen del envío, la dirección destino (que en caso de ser una dirección externa se compondrá de la IP y el puerto en vez de la dirección MAC) y del campo *options*.

Este último campo, está reservado para indicar condiciones especiales en los mensajes, por ejemplo, que es una transmisión grupal o que el mensaje llega desde fuera de la red.

La implementación de la estructura *options* correspondiente con esta parte del protocolo, es la siguiente:

```
typedef struct {
    uint8_t type;     /**< option type */
    uint16_t len;     /**< option length */
    uint8_t *val;     /**< option value */
} __attribute__((packed)) mesh_opt_t;
```

Lo óptimo sería incluir un tipo nuevo para comandos, en cuyo valor se añada el tipo de comando enviado. Actualmente tan sólo hay definidos dos tipos de opciones: MESH_OPT_SEND_GROUP para el envío a grupos y MESH_OPT_RECV_DS_ADDR para mensajes que llegan desde fuera de la red.

Para modificar el comportamiento anterior sería necesario modificar el funcionamiento del componente ESP_WIFI. Sin embargo, al igual que otros componentes, esta parte del código no es código abierto. Por lo que para ello es necesario abrir una petición a la comunidad de desarrollo de Espressive.

Otra opción, que permitiría mayor autonomía, es usar el campo *val* de la estructura para marcar el mensaje como un evento, ya que excepto en casos concretos ese atributo no está en uso.

A partir de ahí y tal como puede apreciarse en la Figura 10, se puede utilizar el método expuesto por la interfaz esp_mesh_send para el envío de cualquier tipo de mensaje.

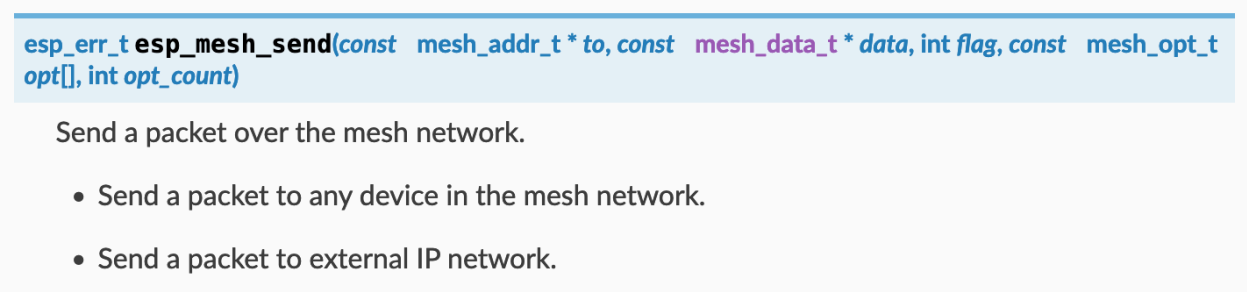


Figura 10: API de envío de mensajes en ESP-WIFI-MESH

Para la información en sí del comando: identificador e información anexa, se usará el propio payload del mensaje (parámetro data del método *esp_mesh_send*), ya que la cabecera *options* tiene un tamaño bastante reducido.

Para dar una base sólida a los mensajes, se define una estructura que contenga la información asociada al evento (*cmd_t*). Esta contendrá información básica como el identificador del propio evento o los parámetros extra que se necesiten para su ejecución. A continuación, se detalla la información mínima de un comando:

- id_cmd: Identificador del comando a transmitir.

- id_pkg: Identificador del paquete asociado al comando, permitiendo labores de trazabilidad.
- len_params: Tamaño de memoria que ocupan los parámetros asociados al evento.
- params[]: Buffer con la información de los parámetros.

3.2.2 Recepción y filtrado de mensajes

Para no interferir en los mensajes del propio aplicativo y debido a que la implementación de la interfaz de lectura no es código libre, es necesario hacer un *wrapper*. Aunque es una solución intrusiva, que no cumple con el objetivo secundario buscado, después del análisis de la implementación del componente es la única viable.

Tal y como se aprecia en la Figura 11, el objetivo de la capa a incluir sería filtrar los mensajes que lleguen, identificando y extrayendo los mensajes de administración y test, dejando el resto de mensajes en un buffer separado donde puedan ser extraídos por la aplicación bajo demanda.

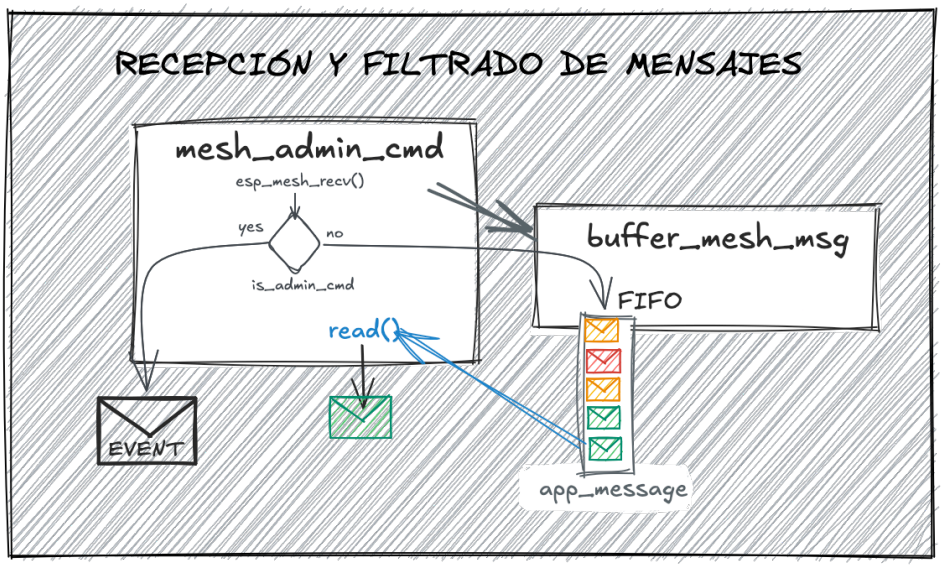


Figura 11: Esquema de recepción y filtrado de mensajes

El diseño de dicha capa consta de dos módulos, uno para el filtrado de los mensajes, y el otro con la gestión del buffer de mensajes disponibles para el aplicativo.

3.2.2.1 Filtrado de los mensajes

Este componente gestiona y filtra los mensajes dejando a disposición del aplicativo un interfaz análogo al proporcionado por ESP_WIFI_MESH, de forma que una posible migración sea lo más limpia posible.

- *esp_err_t* ***read_msg****(mesh_addr_t *from, mesh_data_t *data, int *flag, mesh_opt_t *opts):* Este método copia en las estructuras pasadas como parámetro uno de los mensajes del buffer junto con el contexto asociado a este, y libera la memoria que ocupaban estas en el buffer. Los parámetros asociados coinciden en tipo de dato y objetivo con los de ESP_WIFI_MESH.
- *void* ***init_msgs_buffer***: Antes de poder comenzar a leer mensajes es necesario iniciar el buffer y el proceso de que encarga del filtrado de información y la carga del buffer.
- *int* ***get_msgs_pending****:* Para completar la interfaz proporcionada originalmente por ESP_WIFI_MESH se proporciona un método que indique el número de mensajes que están disponibles para ser leídos.

Por otro lado, una de las labores más importantes del componente es copiar los mensajes que llegan a una nueva región de memoria controlada por el propio componente, de forma que la información pueda ser almacenada sin peligro de que la memoria sea liberada. Para ello se hace una copia en profundidad de los mensajes de las estructuras asociadas al método *esp_mesh_recv*.

3.2.2.2 Gestión del buffer

Este componente recibe y almacena los mensajes de aplicación a la espera que sean leídos. La implementación sigue un funcionamiento análogo al proporcionado por ESP_WIFI_MESH, es decir una cola FIFO.

El componente proporciona una interfaz en la que el componente de filtrado pueda apoyarse para poder añadir y recuperar mensajes del buffer, así como un método para consultar cuantos mensajes hay en el buffer en un momento concreto.

3.2.3 Notificación de la llegada de comandos

La implementación se lleva a cabo mediante eventos, por lo que una de las labores del componente es declarar y definir la base de estos eventos, con nombre: MESH_TEST_EVENTS.

Los eventos recibidos son enviados por el *event loop* por defecto, a través de la llamada al método esp_event_post. Los parámetros del evento que pueda llegar a tener son recuperados del payload del mensaje y transladados al parámetro correspondiente de la función esp_event_post.

Aunque el objetivo real del proyecto es proporcionar la base para poder implementar un juego de eventos que permitan testear la resiliencia real de una red mesh, puede ser interesante que el componente proporcione eventos predefinidos que ayuden a los desarrolladores.

Capítulo 4 - Integración en ESP-MDF

ESP-MDF o Espressif Mesh Development Framework es un proyecto de Espressif que nace para disponer de un framework de desarrollo con redes mesh. Utiliza el componente Espressif Wifi Mesh y sus capacidades como base sobre la que incorporar algunas capacidades que facilitan el desarrollo.

4.1 Arquitectura actual

El componente encargado de incluir funcionalidades sobre el envío de paquetes y principal componente del framework es Mwifi (Wi-Fi Mesh) [6], dicho componente es la encapsulación de las API de ESP-WIFI-MESH, añadiendo nuevas funciones de:

- **Filtro de retransmisión**: Añade un control del flujo de datos al transmitir datos en sentido descendente, evitando fragmentos redundantes en caso de red inestable o interferencia inalámbrica. Para ello, Mwifi añade un ID de 16 bits a cada fragmento y los fragmentos redundantes con el mismo ID se descartarán.
- **Transmisión fragmentada**: cuando el paquete de datos excede el límite del tamaño máximo de paquete, Mwifi lo divide en fragmentos antes de transmitirlos al dispositivo de destino para su reensamblaje.
- **Compresión de datos**: Añade la posibilidad de compresión o descompresión indicado a través de un parámetro. Esta funcionalidad puede aumentar la velocidad de transmisión sobretodo para tipos de dato en formato texto.
- **Multidifusión P2P**: como la multidifusión en ESP-WIFI-MESH puede provocar la pérdida de paquetes, Mwifi utiliza un método de multidifusión P2P (peer-to-peer) para garantizar una entrega mucho más confiable de paquetes de datos.

4.2 Arquitectura de integración planteada

Como se verá a continuación, la solución planteada es análoga en estructura a la solución anterior. Esto se debe a que el framework esta implementado sobre ESP-WIFI-MESH y comparten una arquitectura común y por tanto también las problemáticas, y por suerte las oportunidades de integración.

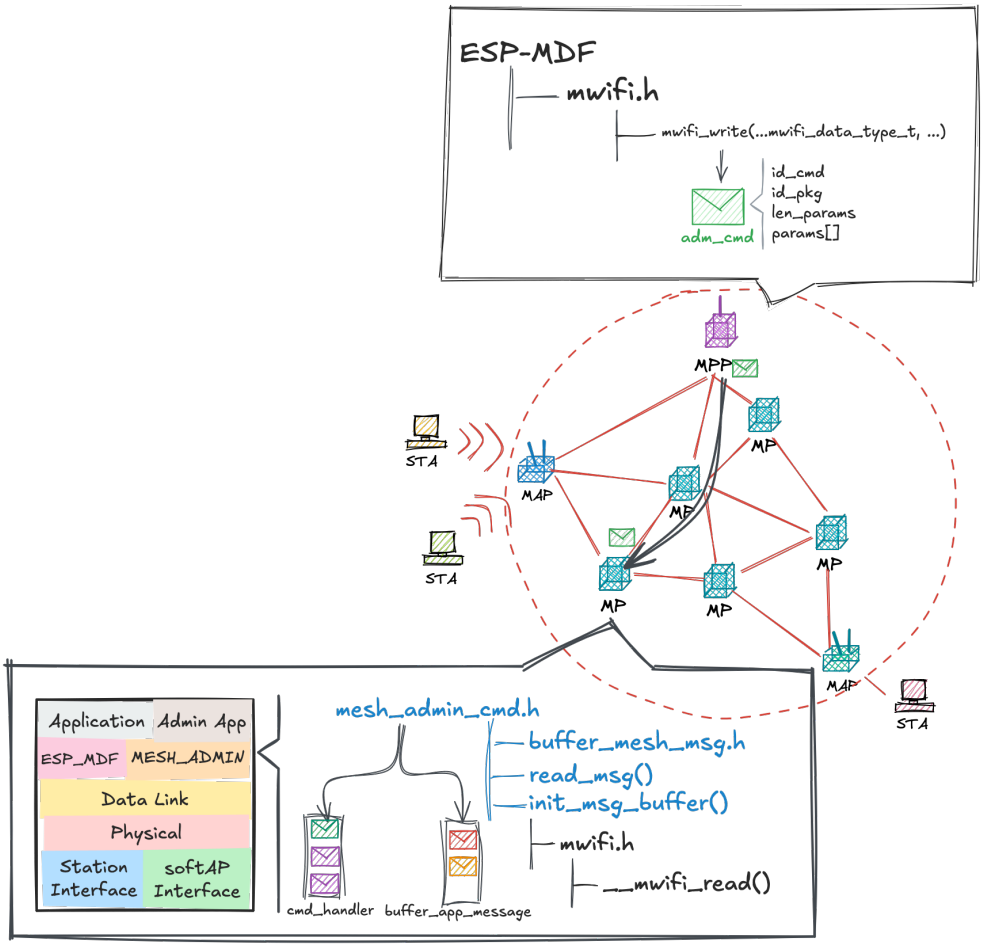


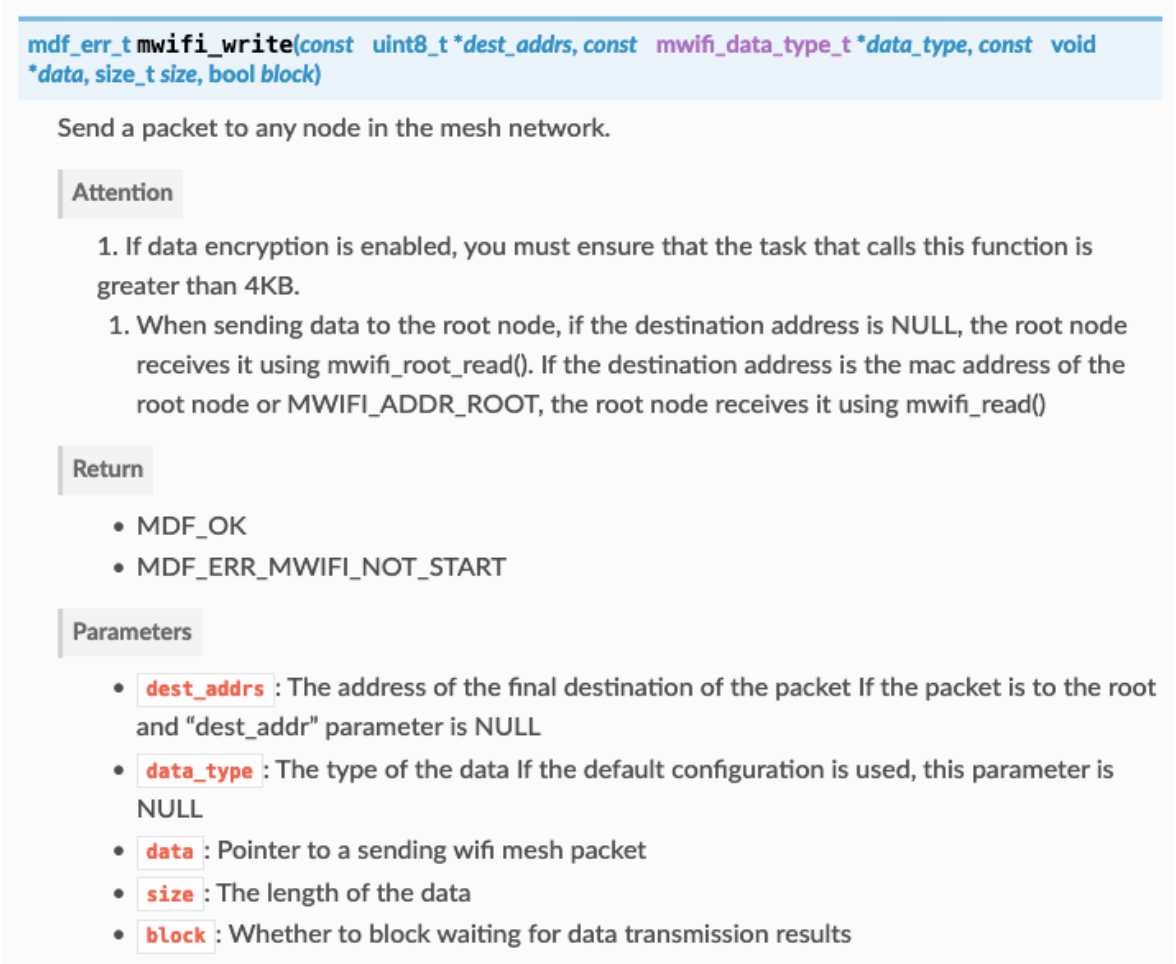
Figura 12: Mapa de interfaces de integración con ESP-MDF

Como puede verse en la Figura 12, puede adaptarse la misma arquitectura de solución anterior para conseguir los objetivos buscados. Y simplemente habría que adaptar los puntos de integración con las interfaces expuestas por el framework.

4.2.1 Envío de comandos

Al entrar en mayor profundidad sobre el desarrollo de las funcionalidades que proporciona la interfaz mwifi.h de ESP-MDF, se puede observar que durante el encapsulamiento de las funciones originales de ESP-WIFI-MESH, han simplificado el API expuesto tanto para la lectura, como para la escritura de mensajes.

Para la escritura de comandos Mwifi expone un API como el que puede verse en la Figura 13. Desaparecen varios parámetros con respecto al API de WIFI-MESH.



mdf_err_t **mwifi_write**(*const* uint8_t **dest_addrs*, *const* mwifi_data_type_t **data_type*, *const* void **data*, size_t *size*, bool *block*)

Send a packet to any node in the mesh network.

Attention

1. If data encryption is enabled, you must ensure that the task that calls this function is greater than 4KB.
1. When sending data to the root node, if the destination address is NULL, the root node receives it using mwifi_root_read(). If the destination address is the mac address of the root node or MWIFI_ADDR_ROOT, the root node receives it using mwifi_read()

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START

Parameters

- `dest_addrs` : The address of the final destination of the packet If the packet is to the root and "dest_addr" parameter is NULL
- `data_type` : The type of the data If the default configuration is used, this parameter is NULL
- `data` : Pointer to a sending wifi mesh packet
- `size` : The length of the data
- `block` : Whether to block waiting for data transmission results

Figura 13: API escritura de ESP-MDF

Entre los parámetros desaparecidos podemos encontrar el parámetro opt, que en WIFI-MESH nos permite incluir cabeceras en los mensajes. Esto lo han hecho porque el envío de la información necesaria para el filtrado de paquetes redundantes, la información de la compresión del paquete o establecer si es un mensaje completo o

una parte, termina haciéndose a través de un juego de cabeceras que están contenidas en un tipo privado llamado *mwifi_data_head_t* que termina insertándose en el options. La estructura es la siguiente:

```
typedef struct {
    uint32_t magic;                   /**< Filtro de paquetes duplicados */
    struct {
        bool transmit_self      : 1;  /**< Si el paquete reenviado es para mi */
        bool transmit_all       : 1;  /**< Enviar los paquetes a todos */
        size_t transmit_num     : 10; /**< Número de dispositivos a los que reenviar */
        size_t total_size_low   : 12; /**< Longitud total del paquete */
        uint8_t packet_seq      : 3;  /**< Número de serie del paquete */
        size_t total_size_hight : 1;  /**< Longitud total del paquete */
        uint8_t compress_rate   : 4;  /**< Ratio de compresión */
    };
    mwifi_data_type_t type;           /**< Tipo de los datos */
} __attribute__((packed)) mwifi_data_head_t;
```

Por suerte, el tipo de datos contenido en el atributo *type de tipo mwifi_data_type_t*, corresponde con el parámetro *data_type* del método *mwifi_write:*

```
typedef struct {
    bool compression    : 1; /**< Activa la compresión */
    bool upgrade        : 1; /**< Actualización del paquete */
    uint8_t communicate : 2; /**< Metodo de comunicación: Unicast, Multicast o Broadcast */
    bool group          : 1; /**< Envio del paquete a un grupo */
    uint8_t reserved    : 1; /**< reservado */
    uint8_t protocol    : 2; /**< Protocolo de transmisión */
    uint32_t custom;         /**< Tipo de datos del payload */
} __attribute__((packed)) mwifi_data_type_t;
```

Para introducir la información que necesitamos sin ser intrusivos con el propio framework podríamos utilizar el campo *custom* para marcar el mensaje como un comando de administración.

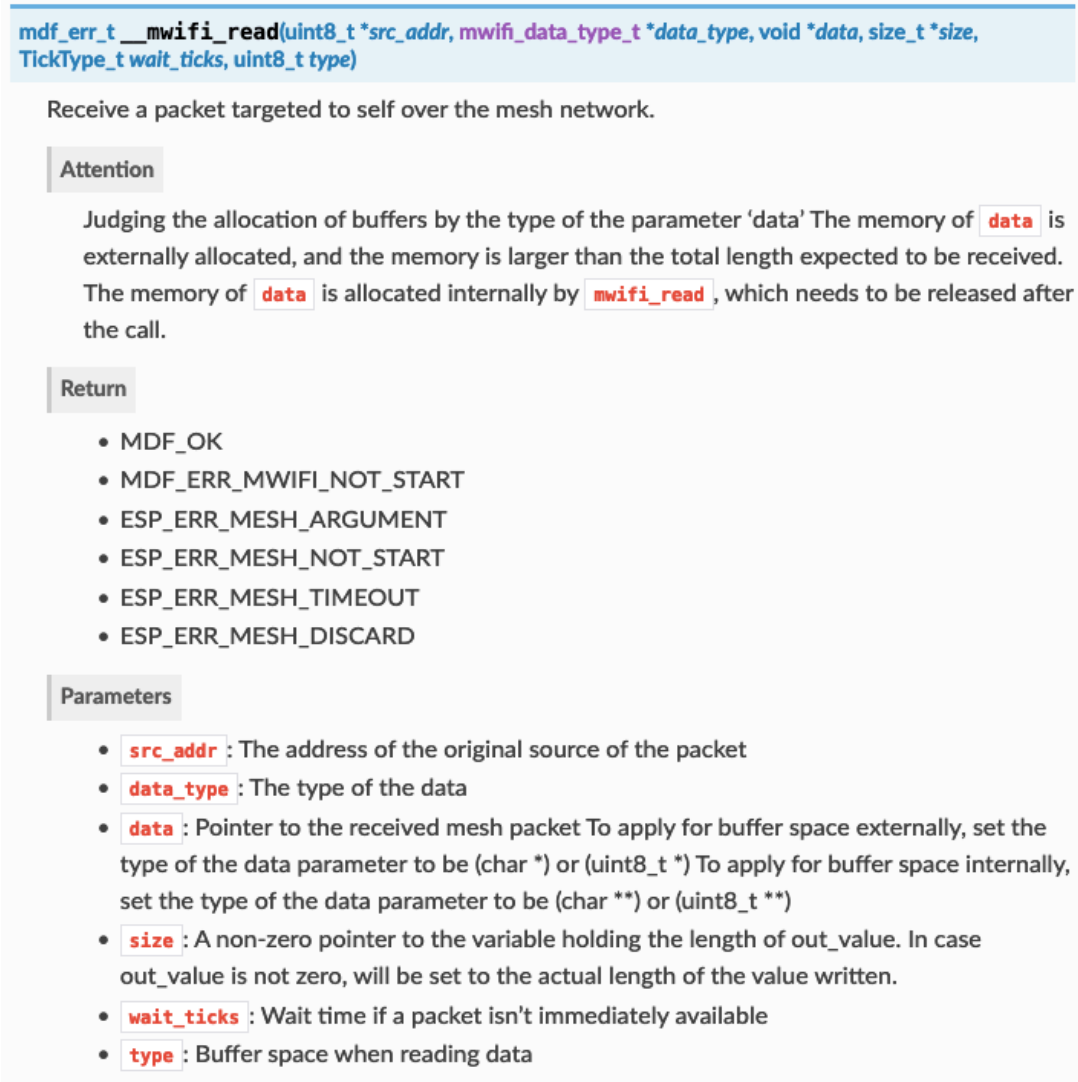
Como en el caso anterior, para la información relativa al propio comando podemos utilizar el payload del mensaje. Usando igualmente el parámetro *data* de la función.

4.2.2 Recepción, filtrado de mensajes y notificación de eventos

En el caso de la recepción, filtrado y notificación de los eventos asociados a los mensajes de administración y test entrantes, la solución sigue siendo análoga a la descrita en el apartado 3.2.2 y 3.2.3.

Los cambios más significativos son los relativos a la integración pueden verse en la Figura 14, donde se describe la interfaz de lectura de la librería mwifi.h.

Habría que adaptar al menos los mapeos entre la estructura de datos de intercambio con el aplicativo y el interfaz expuesto por __mwifi_read, fundamentalmente en lo referente al parametro mwifi_data_type_t.



mdf_err_t **__mwifi_read**(uint8_t *src_addr, mwifi_data_type_t *data_type, void *data, size_t *size, TickType_t wait_ticks, uint8_t type)

Receive a packet targeted to self over the mesh network.

Attention

Judging the allocation of buffers by the type of the parameter 'data' The memory of `data` is externally allocated, and the memory is larger than the total length expected to be received. The memory of `data` is allocated internally by `mwifi_read`, which needs to be released after the call.

Return

- MDF_OK
- MDF_ERR_MWIFI_NOT_START
- ESP_ERR_MESH_ARGUMENT
- ESP_ERR_MESH_NOT_START
- ESP_ERR_MESH_TIMEOUT
- ESP_ERR_MESH_DISCARD

Parameters

- `src_addr` : The address of the original source of the packet
- `data_type` : The type of the data
- `data` : Pointer to the received mesh packet To apply for buffer space externally, set the type of the data parameter to be (char *) or (uint8_t *) To apply for buffer space internally, set the type of the data parameter to be (char **) or (uint8_t **)
- `size` : A non-zero pointer to the variable holding the length of out_value. In case out_value is not zero, will be set to the actual length of the value written.
- `wait_ticks` : Wait time if a packet isn't immediately available
- `type` : Buffer space when reading data

Figura 14: Interfaz de lectura de ESP-MDF

Por otro lado, aunque la implementación realmente sería un wrapper que hace de proxy y las funcionalidades ofrecidas por el framework se ejecutan internamente y por tanto no afectarían. En caso de llegar a implementar esta integración, sería conveniente hacer pruebas en profundidad sobretodo de las funcionalidades relativas a la segmentación de mensajes.

Capítulo 5 - Integración en ESP_LITE.

Espressif Mesh Lite [6] es el nuevo SDK pensado para gestionar mallas WIFI en detrimento de ESP_MDF que actualmente se encuentra descontinuado.

Pero la realidad es que se encuentra en un periodo embrionario. Por hacernos una idea, todavía no existe una versión cerrada de código, una guía de uso o un API de referencia, más allá de los ficheros readme alojados en el repositorio de código de Github.

Ha sido implementada por Espressif para permitir interconectar nodos en una misma WLAN con funcionamiento mesh.

5.1 Arquitectura actual

Como puede verse en los módulos disponibles en el esquema de comunicación de la Figura 15. La principal diferencia con respecto a ESP-WIFI-MESH, es que habilita la pila LWIP en los nodos intermedios. Permitiendo acceder a redes externas, y utilizarlos para realizar envíos de paquetes sin que el nodo raíz sea afectado.

De esta forma, los nodos intermedios pueden ser tratados como un dispositivo conectado directamente al router, que puede invocar de forma independiente interfaces de red como Socket, MQTT, HTTP, etc., en la capa de aplicación.

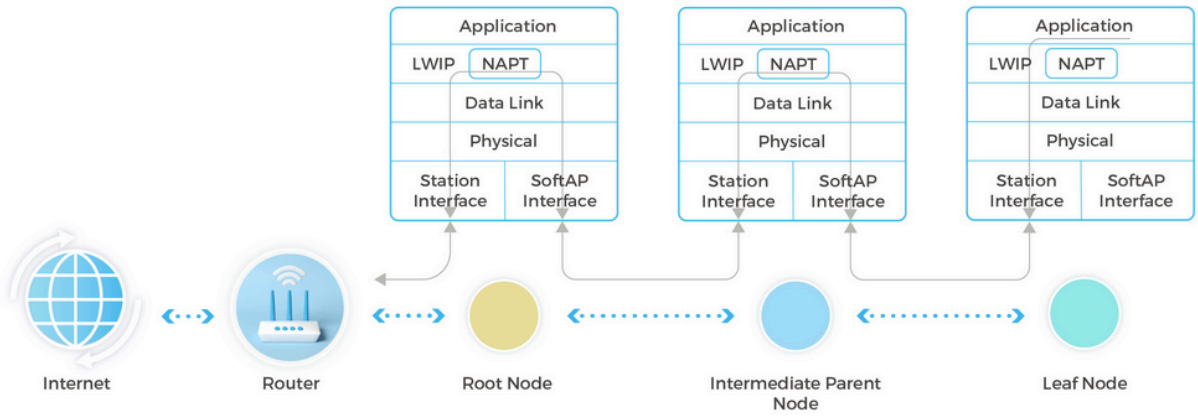


Figura 15: Esquema de comunicación en ESP-LITE

Para ello ESP-LITE optimiza el uso de memoria a costa de la potencia en las labores de aprovisionamiento y reparación de la red.

La forma de selección automática del nodo principal también es diferente entre ESP-MESH-LITE y ESP-MESH.

- ESP-MESH selecciona el nodo raíz de acuerdo con el RSSI cuando todos los dispositivos están inactivos y luego este se conecta al enrutador.
- ESP-MESH-LITE el dispositivo que se enciende primero convierte en nodo raíz. Si se enciende más de un dispositivo al mismo tiempo, después de que todos los dispositivos estén conectados al root seleccionado, todos los dispositivos comienzan a transmitir el RSSI de su root y el que tiene el RSSI más potente se selecciona como nodo raíz. El resto de los nodos se desconectan del root que habían seleccionado y vuelven a buscar un nuevo nodo principal.

5.2 Arquitectura de integración

Es posible, que, por la compatibilidad descrita por el fabricante, lo más interesante en esta opción de integración fuese reutilizar exactamente la misma implementación descrita para ESP-WIFI-MESH. Sin embargo, a nivel técnico es muy interesante explorar la posibilidad de conectar directamente con cada uno de los nodos e interactuar directamente.

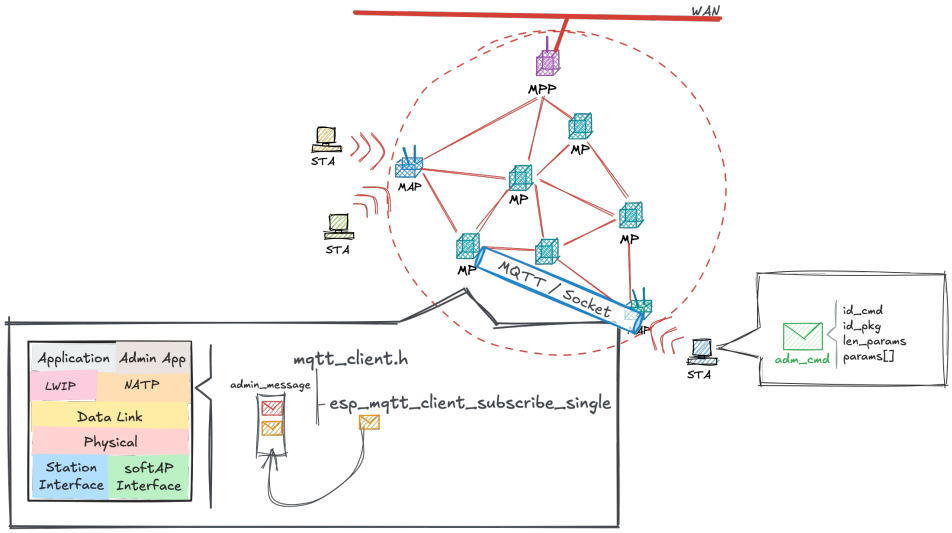


Figura 16: Arquitectura de integración con ESP-MESH-LITE

5.2.1 Envío de comandos

En ESP_MESH_LITE se activa LWIP en todos los nodos de la red, y por tanto estos tienen acceso al protocolo y servicios de IP, pudiendo optar por abrir un socket o una conexión MQTT con el exterior por el que comunicar los comandos.

Aparentemente todo hace suponer que la salida al exterior se realiza a través de la red mesh y no de forma directa. Sin embargo, esto es algo que en el momento de realización de la investigación no se ha encontrado claramente descrito y se debe aclarar. De no hacer uso de la malla para la salida al exterior restringiría el uso de esta solución a redes en las que todos los nodos sean accesibles desde el exterior por el operador, lo cual no siempre es factible.

5.2.2 Recepción de comandos

Para la lectura de los comandos simplemente sería suficiente con hacer una lectura a través del canal seleccionado, bien sea un socket o una suscripción a través de un bróker MQTT.

Capítulo 6 - Caso de uso práctico

Para poder verificar que se cumple el objetivo principal y se ha realizado una prueba de concepto sobre un posible caso de uso.

Tras analizar la tecnología sobre la que realizar la prueba de concepto, se ha decidido hacer sobre ESP-WIFI-MESH por dos motivos fundamentales:

1. ESP-WIFI-MESH esta desarrollado dentro de IDF y por tanto es el protocolo base sobre el que *Espressif* ha desarrollado el resto de *frameworks* disponibles.
2. En el momento de la implementación de la prueba de concepto, ESP-MESH-LITE, opción más novedosa y disruptiva, se encuentra en un estadio demasiado inicial, sin haber publicado todavía la primera *release*.

El caso de uso seleccionado es el de una empresa que desea implementar en sus desarrollos de IoT una filosofía de ingeniería del caos para garantizar la resiliencia de las redes mesh que genera con sus dispositivos.

6.1 Librería de gestión del buffer

Esta librería implementa lo descrito en el capítulo 3.2.2.2 , en la que se debe gestionar un buffer tipo FIFO que contenga los mensajes de aplicación que la librería de administración haya ido pudiendo leer, pero que el aplicativo todavía no ha consumido. Para ello se definen los métodos:

```
esp_err_t add_msg(mesh_msg_t *msg);
mesh_msg_t * get_msg();
int get_num_msgs();
```

La estructura que contendrá los mensajes de tipo mesh_msg contiene los datos recibidos durante la transmisión de un mensaje a través del API de ESP_WIFI_MESH:

```
typedef struct mesh_msg {
       mesh_addr_t from;
       mesh_data_t payload;
       mesh_opt_t opt;
       int flag;
} mesh_msg_t;
```

6.2 Librería de administración de comandos

Como puede visualizarse en la Figura 17: Esquema de lectura y filtrado de mensajes, esta librería tiene tres objetivos fundamentales:

1. Leer periódicamente el buffer de mensajes de ESP-WIFI-MESH, filtrando los mensajes de administración y testing.
2. Proveer de un interfaz de lectura de mensajes a los aplicativos, y por tanto almacenar los mensajes recibidos por la aplicación.
3. Transformar los mensajes de administración y testing recibidos en eventos del tipo correspondiente.

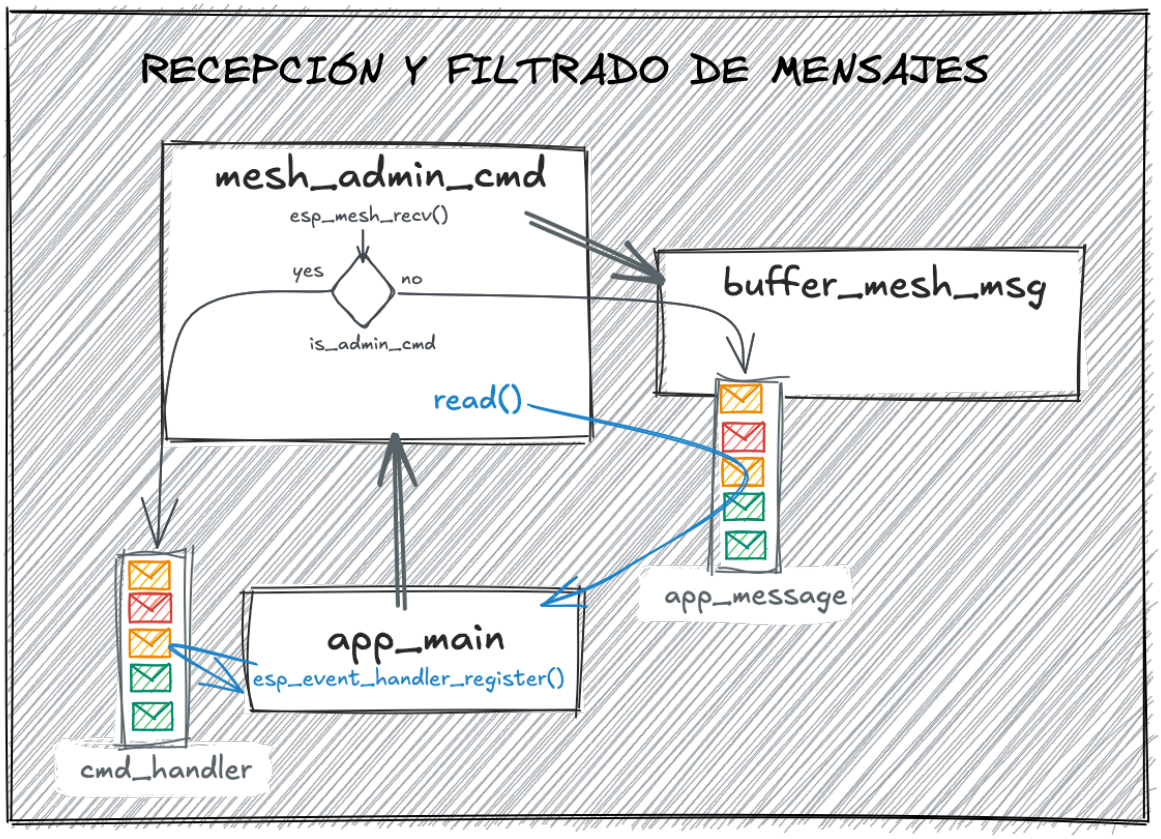


Figura 17: Esquema de lectura y filtrado de mensajes

6.2.1 Lectura y filtrado de mensajes del buffer de ESP_WIFI_MESH

La lectura periódica se realiza tras inicializar una tarea generada a través de la invocación al método init_msgs_buffer.

El filtrado de mensajes se lleva a cabo reconociendo el valor del atributo val de la estructura *mesh_opt_t* de ESP-WIFI-MESH. Este debe tener el valor predefinido en la librería en la constante OPT_MSG_CMD.

Para facilitar el intercambio, se ha definido una estructura mínima para los mensajes de administración y testing. Durante la implementación se ha seguido la estructura definida en el capitulo 3.2.1.

Los mensajes de aplicación que han sido sacados del buffer de ESP-WIFI-MESH, deberán almacenarse en un buffer como una copia en profundidad. Para el almacenamiento se hará uso de la librería de gestión del buffer, que mantendrá los mensajes hasta que sean consumidos por el aplicativo.

6.2.2 Interfaz de lectura de mensajes

Este método se encarga de recuperar el mensaje de aplicación más antiguo a través de la librería de gestión del buffer.

Una vez recuperado, se hace una copia en profundidad y un mapeo a estructuras de ESP-WIFI-MESH: mesh_opt_t, mesh_addr, … y posteriormente se libera la memoria del mensaje.

6.2.3 Notificación de eventos de administración y testing

La librería define el evento base de los MESH_TEST a través de la macro ESP_EVENT_DECLARE_BASE.

Para poder probar el caso de uso se definen tres mensajes / eventos predefinidos dentro de la librería que ayudarán a implementar nuestro módulo de ingeniería del caos para redes mesh:

- MESH_TEST_NODE_RESTART_EVENT: Al recibir este evento el nodo debe reiniciarse.
- MESH_TEST_NODE_SLUGGISH_EVENT: Al recibir este evento el nodo se vuelve vago, y comienza a entrar en inactividad. Este evento requiere de parámetros, que viajaran bajo la estructura sluggish_t:
    - laziness_time_milis: Milisegundos que estará inactivo el nodo.

- total_time_sg: Tiempo total que el nodo estará en modo perezoso.

- MESH_TEST_NODE_NOISE_EVENT: Al recibir el evento el nodo comenzará a "saturar" a otro nodo de la red con mensajes. Este evento también requiere del paso de parámetros bajo la estructura noise_t:
  - msg2second: Mensajes por segundo que enviará.
  - to_addr: Dirección del nodo al que se enviarán los mensajes.
  - total_time_sg: Tiempo total que el nodo estará en modo molesto.

Cuando la librería lee un mensaje de administración y test del buffer de esp_wifi_recv, se envía un evento de la pila de sistema a través del método *esp_event_post*, mapeando del mensaje al evento de la siguiente forma:

- id_cmd → event_id
- params → event_data
- len_params → event_data_size

6.3 Aplicación de prueba

El aplicativo principal parte del ejemplo mesh internal_communication proporcionado por Espressif en ESP-IDF. Este ejemplo implementa una aplicación en la que el nodo root envía mensajes de encendido y apagado de un led de forma alterna a todos los nodos de la red mesh.

Tomando este caso como base se elimina todo excepto la estructura básica de creación y control de eventos mesh, incluyendo varias funcionalidades nuevas que nos ayuden a verificar que podemos recibir los mensajes de test por separados de los de apliación.

En caso de que el nodo sea el root creará una tarea que enviará mensajes en broadcast cada cierto periodo de tiempo. Estos mensajes serán aleatorios se elegirán entre mensajes de aplicación o alguno de los mensajes de ingeniería del caos descritos.

En caso de ser un nodo de la red que no sea el root, se creará una tarea que lea y muestre por consola los mensajes de aplicación que lleguen identificándolos como tal.

Para ello se inicializa el buffer y hará uso del método de lectura de la librería de administración.

Todos los nodos realizan el registro en el handler de eventos de tipo administración y test con un método, que para cada tipo de evento que se pueda escriba por consola la acción que realizaría en caso de ser un caso real.

6.4 Ejecución del caso de prueba

Para la prueba se utilizan 3 SOCs ESP32 que han cargado el mismo firmware, de forma que uno quede como root y los otros dos como nodos intermedios u hojas de la red.

El objetivo principal de la prueba es identificar los mensajes enviados por el root y poder trazar el modo en el que son recibidos por el destino: evento o mensaje de aplicación. Esto permitirá hacer una doble comprobación:

- Que los mensajes de administración y test son filtrados de forma efectiva por la librería, dejando disponible a la aplicación los mensajes de aplicación.
- Que el contenido tanto de los mensajes como de los eventos se recibe correctamente, sin que haya perdidas o cruces de información entre mensajes.

Como objetivo secundario se aprovecha para probar que no existen fugas de memoria o problemas con el buffer que almacena los mensajes de aplicación. Para ello se realiza una segunda ejecución configurando la latencia de lectura de mensajes en 10 segundos (contra la latencia de 1 segundo que tiene la escritura), esto hará que el buffer de mensajes de aplicación se comience a llenar rápidamente.

6.4.1 Resultados

Como se puede visualizar en el recuadro azul de la figura la Figura 18, en un bloque de envíos de comandos el receptor a invocado al método para leer mensajes de aplicación y no ha podido recuperar porque no existían.

Figura 18: Trazas de ejecución de envío y lectura de mensajes

Con ello se ha podido comprobar que la aplicación no puede acceder directamente a los mensajes de tipo administración y test, y estos son notificados mediante la pila de sistema.

Además, como muestran las flechas naranjas de la propia Figura 18, coinciden tanto el tipo de mensaje, el paquete en el que es enviado y su contenido con lo recibido por el nodo receptor.

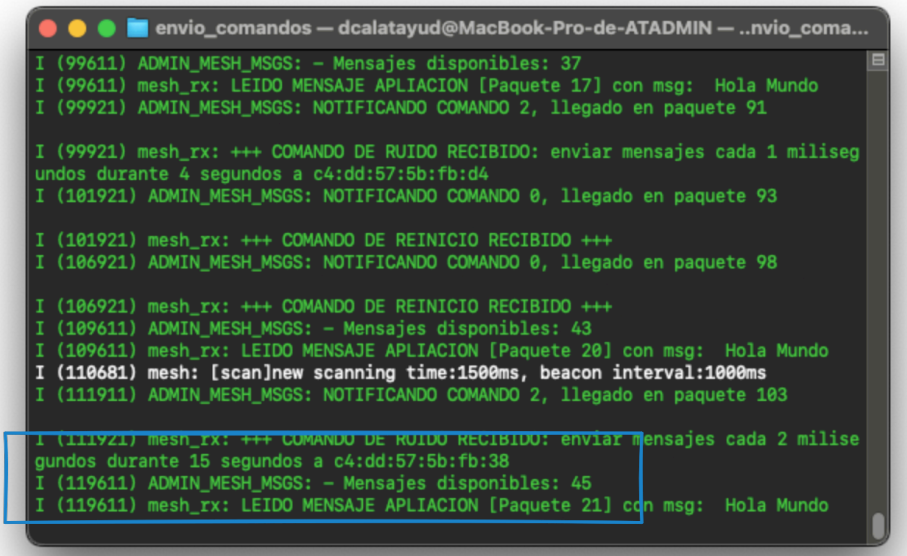


Figura 19: Ejecución con latencia en la lectura de mensajes

Durante la ejecución en la que la latencia de lectura ha sido aumentada, puede verse en el recuadro azul la Figura 19, que en el momento de la captura y tras unos minutos de ejecución, se acumulaban 45 mensajes sin leer. Sin dar muestras de problemática alguna.

Teniendo en cuenta que la reserva de memoria para cada mensaje es el máximo que permite el payload del framework. A falta de pruebas más exhaustivas en las que se verifique que efectivamente la memoria disponible se mantiene estable y va liberándose correctamente, a priori se podría suponer que dicha gestión es correcta.

En cualquier caso, de llevar esta prueba de concepto a un desarrollo de mayor entidad, sería recomendable realizar una verificación exhaustiva tanto de: la gestión de la memoria, el impacto en el aplicativo y la capacidad máxima del buffer de mensajes.

Capítulo 7 - Conclusiones y trabajo futuro

Durante la realización del trabajo he sufrido dos hadicaps importantes: la opacidad del código de Espressif, que no facilita mecanismos para saber que está pasando realmente en la red de forma interna y oculta la implementación de la transmisión y recepción de paquetes dentro de la red.

Sin embargo, se ha logrado cumplir el objetivo principal del proyecto y encontrar una implantación que permita el envío y la recepción y notificación de la llegada de mensajes específicos de administración y testing. Además, se ha logrado probar dicha implementación teórica en un marco práctico mediante la prueba de concepto realizada con éxito.

Aunque de facto, no se haya podido cumplir el objetivo secundario de lograr que la implementación no fuese intrusiva si se ha encontrado un mecanismo que, con leves modificaciones del framework de IDF, permitiría lograr marcar mediante una cabecera los mensajes dedicados a la administración y testing. Sin embargo, y aunque seguramente la opción fuese muy similar a la propuesta, no se ha podido aterrizar una alternativa contrastada para la recepción de los mensajes, ya que esa parte del código fuente no está disponible.

El otro hándicap ha sido encontrarme en un momento en el que la compañía ha realizado un viraje en el enfoque con respecto a la implementación de las redes mesh. Ha descontinuado el framework mesh, orientado a poder establecer mecanismos de control y mejoras en la transmisión de volúmenes grandes de información. Por otro enfoque más liviano y orientado a la posibilidad de establecer una comunicación directa entre los nodos internos y el exterior. Sin embargo, el proyecto todavía se encuentra en una fase muy preliminar, por lo que no la he tenido en cuenta para la PoC realizada.

Aun con la nueva perspectiva de Espressif que puede abrir implementaciones menos intrusivas para el código fuente a través de canales de comunicación alternativos, creo que se ve muy necesario crear un entorno que permita quitar

incertidumbre y permitan la verificación de la resiliencia de la red ante diferentes escenarios.

Por otro lado, he quedado preocupado por la opacidad del desarrollo y la falta de documentación de Espressif en algunos apartados importantes y por mi parte buscaría alternativas de código abierto con los que desarrollar el firmware de mis ESP32.

Como línea de trabajo futuro sería conveniente realizar pruebas de carga que realicen un análisis del impacto que la implementación puede generar sobre los desarrollos.

También sería interesante ampliar las capacidades del componente para facilitar el lanzamiento de tareas asociadas a los propios comandos y ampliaría el pool de comandos predefinidos.

Cuando madure un poco el proyecto de ESP_MESH_LITE sería interesante estudiar la posibilidad de contar con un módulo que establezca la comunicación punto a punto entre el nodo y el exterior de la red como forma de comunicar los comandos.

Chapter - Introduction

During the work I have suffered two important disadvantages: the opacity of the Espressif code, which does not provide mechanisms to know what is really happening in the network internally and hides the implementation of the transmission and reception of packets within the network.

The other handicap has been finding myself at a time when the company has made a shift in approach with respect to the implementation of mesh networks. The mesh framework has been discontinued, aimed at establishing control mechanisms and improvements in the transmission of large volumes of information. For another, lighter approach aimed at the possibility of establishing direct communication between the internal and external nodes. However, the project is still in a very preliminary phase, so I have not taken it into account for the PoC carried out.

Even with the new perspective of Espressif that can open less intrusive implementations for the source code through alternative communication channels, I think it is very necessary to create an environment that allows removing uncertainty and allows verification of the network's resilience to different scenarios.

On the other hand, I was concerned about the opacity of the development and the lack of Espressif documentation in some important sections and for my part I would look for open source alternatives with which to develop the firmware of my ESP32.

As a line of future work, I would expand the capabilities of the component to facilitate the launching of tasks associated with the commands themselves and expand the pool of predefined commands.

On the other hand, when the ESP_MESH_LITE project matures a little, it would be interesting to study the possibility of having a module that establishes point-to-point communication between the node and the outside of the network as a way of communicating commands.

.

Chapter - Conclusions and future work

During the work I have suffered two important handicaps, on the one hand, the opacity of the Espressif code, which on the one hand makes it very difficult to establish mechanisms to know what is really happening on the network internally and on the other hand hides the implementation of the transmission and reception of packets within the network.

The other handicap has been finding myself at a time when the company has made a shift in approach with respect to the implementation of mesh networks. Proof of this is that it has discontinued an open framework project aimed at establishing communication control mechanisms. And it has focused on a project that establishes an approach that enables direct communication between a node and the outside world and that simplifies the interface and further hides the communication between nodes in the network. However, the project is still in a very preliminary phase, so I have not taken it into account for the PoC.

In any case and even with the new perspective of Espressif that can open less intrusive implementations for the source code, I think it is very necessary to create an environment that favors the performance of tests that verify the resilience of the network in different scenarios.

On the other hand, I have been concerned by the opacity of the development and the lack of Espressif documentation in some important sections and for my part I would look for alternative open source languages.

As a line of future work, I would expand the capabilities of the component to facilitate the launching of tasks associated with the commands themselves and expand the pool of predefined commands.

On the other hand, when the ESP_MESH_LITE project matures a little, it would be interesting to have a module that establishes point-to-point communication between the node and the outside of the network as a way of communicating commands.

BIBLIOGRAFÍA

[1] «Open 802.11s,» 2013. [En línea]. Available: http://open80211s.org/. [Último acceso: 2023].

[2] «Wiki OLPC,» [En línea]. Available: https://wiki.laptop.org/go/Mesh_Network_Details. [Último acceso: 2023].

[3] «Wiki de FreeBSD,» [En línea]. Available: https://wiki.freebsd.org/WiFi/Mesh. [Último acceso: 2023].

[4] Espressif, «ESP WIFI MESH,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32/api-guides/esp-wifi-mesh.html. [Último acceso: 2023].

[5] Espressif, «ESP-IDF,» [En línea]. Available: https://www.espressif.com/en/products/sdks/esp-idf. [Último acceso: 2023].

[6] Espressif, «ESP-MDF Mwifi,» [En línea]. Available: https://docs.espressif.com/projects/esp-mdf/en/latest/api-guides/mwifi.html. [Último acceso: 2023].

[7] «Espressif Mesh Lite,» [En línea]. Available: https://github.com/espressif/esp-mesh-lite/blob/master/components/mesh_lite/User_Guide.md. [Último acceso: 2023].

[8] L. A. Bucki, Word 2013 Bible, John Wiley & Sons, 2013.

[9] CFI, «Cursos de Formación en Informática,» [En línea]. Available: http://cursosinformatica.ucm.es. [Último acceso: 01 06 2019].