ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

ANÁLISIS DE UNA RED INALÁMBRICA MALLADA AUTOCONFIGURABLE, UTILIZANDO EL MÓDULO NODEMCU ESP32 CON EL ESTÁNDAR 802.11

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

ROBINSON EFRAÍN LOACHAMÍN SANGOQUIZA

robinson.loachamin@epn.edu.ec

DIRECTOR: ING. LUIS FELIPE URQUIZA, PhD

luis.urquiza@epn.edu.ec

Quito, enero 2021

AVAL

Certifico que el presente trabajo fue desarrollado por Robinson Efraín Loachamín Sangoquiza, bajo mi supervisión.

DR. LUIS FELIPE URQUIZA AGUIAR
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Robinson Efraín Loachamín Sangoquiza, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

Robinson Efraín Loachamín Sangoquiza

DEDICATORIA

A mis padres, por estar siempre pendientes, brindándome un apoyo incondicional cuando más lo necesito. A mi compañera de vida Mayra gracias por empujarme a conseguir mis objetivos, cuando más lo necesitaba me ayudaste a no dejar la cosas a medias y darle con todo hasta cumplir mis metas planteadas.

AGRADECIMIENTO

Primeramente, a Dios, por darme la vida y la sabiduría para poder concluir este trabajo de titulación.

A mis queridos padres, Rocío y Efraín, gracias por darme incondicionalmente su apoyo en todas las etapas de mi vida. Son los mejores padres que me pudo tocar.

A Mayra, gracias mi amor por nunca dejarme solo y estar ahí empujándome a que terminé lo que empecé. Te amo.

Para Alexander, a pesar que no te tengo entre mis brazos todavía hijo mío, pero eres uno de los motivos por quien termino está meta planteada.

A Kathy por saber entender y ayudar con toda predisposición.

Al Dr. Luis Felipe Urquiza, gracias por el apoyo, la paciencia y su disposición en todo momento para guiarme en las inquietudes y dudas que se presentaron al realizar este trabajo de titulación.

Finalmente, agradeciéndole a la Escuela Politécnica Nacional y su personal docente por impartirme sus enseñanzas y experiencias adquiridas a largo de su vida profesional.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGO

RESUMEN

En la actualidad, el uso de las tecnologías inalámbricas se encuentra en auge, pero el uso de redes inalámbricas malladas usando el estándar IEEE 802.11 con dispositivos de bajo coste no ha sido muy estudiado. Por lo que, el presente trabajo tiene como finalidad presentar un estudio experimental de las redes malladas con nodos autoconfigurables y autónomos que permitan la comunicación entre ellos enviando tráfico unicast y broadcast para su evaluación.

El presente trabajo inicia con una breve descripción de las redes malladas inalámbricas y las características del dispositivo NodeMCU ESP32, seguido del estudio de entornos de desarrollo que se adapte de mejor manera al dispositivo antes mencionado.

En el segundo capítulo se describe, los requerimientos que debe tener la red mallada inalámbrica, así como el desarrollo de scripts para trasmisiones unicast y broadcast.

En el tercer capítulo se muestra, los resultados y el análisis de las métricas evaluadas en las diferentes pruebas aplicadas a la red.

En el cuarto capítulo se presentan las conclusiones, a las que se ha llegado después de realizar el análisis de las métricas obtenidas, de la misma manera se presenta en este capítulo las recomendaciones que van relacionadas con el tema.

Por último, en la sección de anexos se encuentra el manual de instalación de los entornos y las librerías de acceso libre, así como los scripts desarrollados para la trasmisión unicast y broadcast.

PALABRAS CLAVE: Malla, NodeMCU ESP32, script, métricas, unicast, broadcast.

ABSTRACT

Nowadays, wireless technologies are on the rise; nevertheless, the use of wireless mesh networks using the IEEE 802.11 standard with low-cost devices has not been much studied. Therefore, the present work aims to present an experimental study of meshed networks with self-configuring and autonomous nodes that allow communication between them by sending unicast and broadcast traffic for evaluation.

This work begins with a brief description of wireless mesh networks and the characteristics of the NodeMCU ESP32 device, followed by the study of development environments that best adapt to the device.

The second chapter describes the requirements that the wireless mesh network must have, as well as the development of scripts for unicast and broadcast transmissions.

The third chapter shows, the results and the analysis of the metrics evaluated in the different tests applied to the network.

The fourth chapter presents the conclusions, which have been reached after carrying out the analysis of the metrics obtained, in the same way the recommendations related to the subject are presented in this chapter.

Finally, in the annexes section you will find the installation manual for the free access libraries and environments, as well as the scripts developed for unicast and broadcast transmission.

KEYWORDS: Mesh, NodeMCU ESP32, script, metrics, unicast, broadcast.

1. INTRODUCCIÓN

En la actualidad, el estudio de redes malladas en dispositivo de bajo coste, no ha sido muy analizadas a profundidad por lo que este proyecto técnico que se presenta se da como una alternativa para estudiar este tipo de redes.

Una red tipo malla inalámbrica, emplea arreglos de conexiones, es decir una topología total o una parcial [1]. Todos los dispositivos que se encuentran dentro de esta red, tiene la capacidad de conectarse entre ellos, dando como resultado una única red con un solo identificador y contraseña. Las redes malladas permiten una amplia y mejor cobertura entre los nodos que se tenga dentro de la red, encontrando las rutas más rápidas y fiables para enviar datos, cabe mencionar que este tipo de redes se estudian generalmente en simuladores.

A partir de este antecedente, el presente trabajo tiene como finalidad presentar un estudio experimental de las redes malladas con nodos autoconfigurables. Se usará el módulo NodeMCU ESP32 que integra tecnología wifi y tiene la capacidad de soportar scripts, para una comunicación entre nodos autónomos con diferentes métodos de difusión (unicast y broadcast).

Este estudio permitirá conocer de forma experimental el rendimiento de una red mallada de bajo coste, lo que ayudará a conocer de forma práctica sus características como, alcances, throughput, pérdidas que se pueden dar dentro de las redes tipo malla, por lo tanto, dará una idea del tipo de aplicaciones que podrían soportar sobre las mismas.

1.1. OBJETIVOS

El objetivo general de este Proyecto Técnico es:

- Analizar una red inalámbrica mallada autoconfigurable, utilizando el módulo NodeMCU ESP32 con el estándar 802.11

Los objetivos específicos del Proyecto Técnico son:

- Analizar las especificaciones técnicas que ofrecen el módulo NodeMCU ESP32 para crear una red mallada inalámbrica.

- Analizar la factibilidad de usar librerías de acceso libre que permita construir redes tipo malla.

- Desarrollar scripts con el uso de una librería compatible con el módulo NodeMCU ESP32, para probar el comportamiento de la red.

- Analizar los resultados obtenidos a partir de las pruebas de funcionamiento de los tres escenarios planteados.

1.2. ALCANCE

En el presente proyecto se analiza una red mallada de cuatro nodos que contienen placas NodeMCU ESP32 [2], la cual integra un módulo wifi y un microcontrolador, estas placas se integrarán con librerías de acceso abierto para que soporte configurar las características de una red mallada inalámbrica. En este sentido el dispositivo por sí solo no tiene precargado ningún tipo de software que permita tener característica de una red tipo malla, por lo que un primer paso será el análisis de las librerías de acceso abierto que permitan crear redes tipo malla en ambientes inalámbricos, después se escogerán las librerías adecuadas para desarrollar este tipo de redes.

A continuación, se compilarán las librerías de acceso abierto elegida sobre las placas NodeMCU ESP32 para poder obtener nodos que sean autónomos y autoconfigurables. Cada uno de los nodos tendrán la capacidad de ser configurados, para mantener la comunicación con el resto de nodos presentes mediante un escaneo y reconexión.

Se probarán distancias máximas que pueden alcanzar los nodos de la red sometiéndoles a diferentes cambios de escenarios, como afecta el número de saltos en el envío de datos entre los nodos (throughput), la pérdida de paquetes que se puede presentar al momento de intercambiar mensajes. Adicional a esto los nodos de la red mallada podrán comunicarse, uno a uno (unicast), uno para todos (broadcast).

Las pruebas se realizarán con cuatro nodos (NodeMCU ESP32) los cuales pertenecerán a la red mallada cargados con librerías de acceso libre. Se programarán diferentes scripts para probar los métodos de difusión (unicast y broadcast) que se pueden dar dentro de la red con una visualización en consola de los datos trasmitidos.

Se evaluarán en tres escenarios diferentes:

Escenario A: Todos los nodos están en el alcance de los otros nodos, todos se ven entre sí. La figura 1.1 muestra el primer escenario de comunicación entre nodos donde se van a realizar las pruebas.

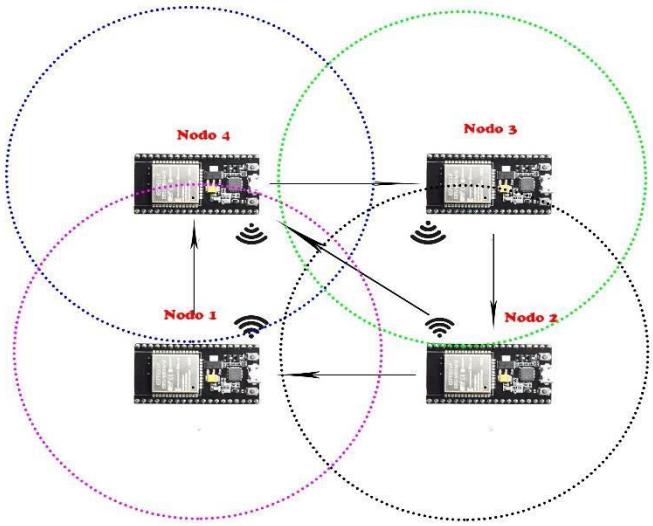


Figura 1.1. Escenario A.

Escenario B: La comunicación de los nodos se dan en pares.
La figura 1.2 muestra el segundo escenario de comunicación donde los nodos se ven en pares para realizar las pruebas.

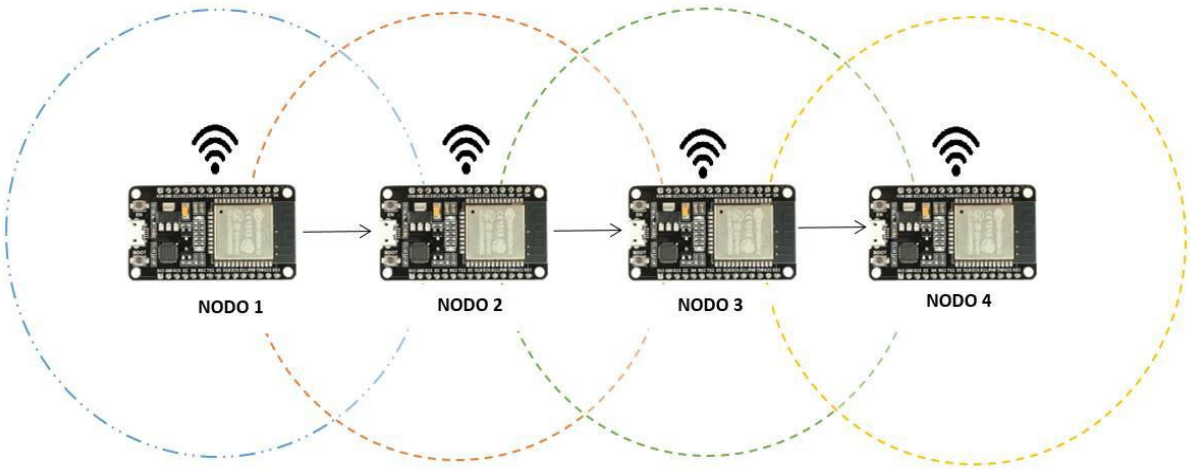


Figura 1.2. Escenario B.

Escenario C: Se conectan en una topología tipo estrella en donde el nodo dos es el punto de convergencia de los otros nodos.

La figura 1.3 muestra el tercer escenario de comunicación de los nodos donde existe un nodo de convergencia con los demás de la red.

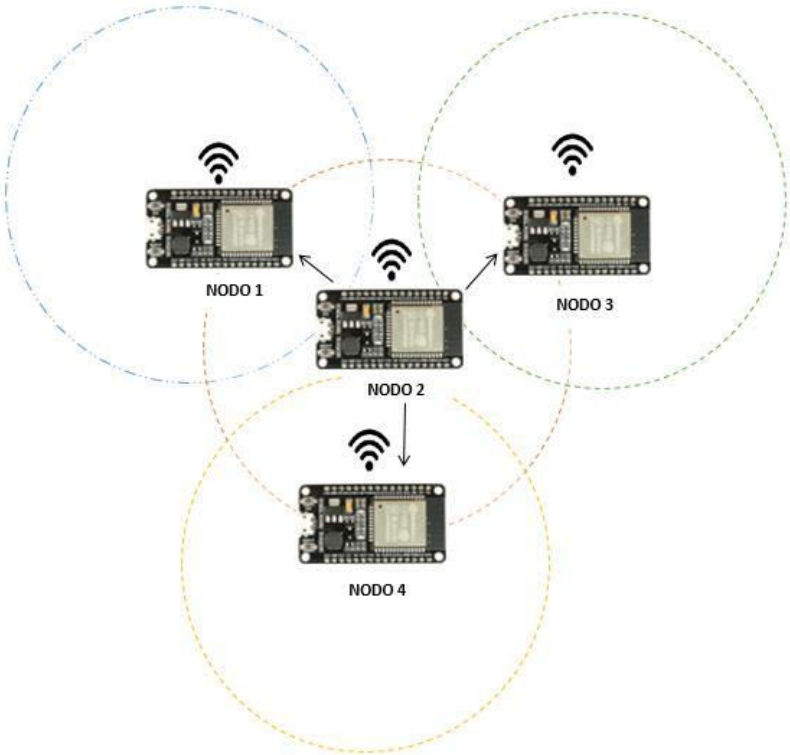


Figura 1.3. Escenario C.

Por último, se analizarán los datos obtenidos a partir de las pruebas de funcionamiento de los tres escenarios planteados.

1.3. MARCO TEÓRICO

En la actualidad, los estudios de las redes se enfocan en redes de infraestructura y ad-hoc. Las redes de infraestructura se caracterizan por ser sistemas centralizados donde los clientes dependen del punto central llamado Access Point (AP), todo el tráfico de la red circula a través del AP por lo cual pueden presentar congestión de la red cuando una gran cantidad de clientes envían paquetes simultáneos al mismo sistema centralizado.

En las redes ad-hoc cada uno de los nodos que pertenecen a la red se pueden comunicarse directamente con los demás nodos siempre y cuando pertenezcan a la red. Cada una de las redes mencionadas anteriormente presentan una desventaja siendo la congestión de la red y las distancias que pueden alcanzar. Por tal motivo se presenta una alternativa de red para solucionar los problemas antes mencionados para extender su rango de transmisión o de cobertura y el control centralizados es el permitir que cada uno de los nodos que se encuentren en la red actúen como un repetidor, por lo que cada uno de los nodos tendrán la facultad de reenviar los datos hacia el nodo destino, por tal razón aparecen las redes malladas.

1.3.1. RED

Es un conjunto de dispositivos conectados entre sí ya sea de forma alámbrica o inalámbrica con el único propósito de enviar y/o recibir paquetes de datos. Actualmente las redes de información se encuentran inmersas en nuestro diario vivir y entorno, las cuales son usadas en la mayoría de campos como los procesos industriales, servicios financieros, teleconferencias, mensajería electrónica, entre otras.

Existen dos tipos de topología dentro de una red se detalla a continuación:

- **Topología Física:** indica cómo se encuentran conectados y distribuidos los dispositivos en una red.
- **Topología Lógica:** indica de qué manera se realiza la comunicación dentro de la red.

1.3.2. RED MALLADA

Red mallada o también conocida como mesh network, es aquella que se encuentra distribuida en una topología física tipo malla, en donde cada nodo que pertenece a la red se encuentra conectado en un enlace punto a punto y dedicado a cualquier nodo de la red.

Cada uno de estos nodos pueden encontrarse conectados de forma parcial o total entre sí, obteniendo varias rutas por donde se puede enviar y/o recibir datos. Gracias a esto tenemos como resultado redes robustas, tolerantes a fallas, escalables y adaptables, comparadas con otras redes. Hay que estar conscientes que el usar este tipo de redes en un ambiente alámbrico implica tener una red muy costosa por la gran cantidad de cables de red que se usa para tener este tipo de comunicación, cabe mencionar que un ambiente atractivo para estas redes es un ambiente inalámbrico.

El modo de trabajo de esta red hace que los datos se retrasmitan por cada uno de los nodos hasta llegar al nodo destino formando una jerarquía plana. Este tipo de redes pueden sufrir cambios en la topología, afectando la distribución de la carga de la red y su rendimiento. [3]

1.3.3. RED INALÁMBRICA

Una red inalámbrica o también conocido como Wireless Network es un tipo de red que no requiere un medio guiado para realizar la transmisión. Existen varios tipos de redes que interactúan sus nodos entre sí utilizando tecnología inalámbrica como redes celulares, redes LAN inalámbricas entre otras.

A continuación, se podrá ver en la figura 1.4 un ejemplo de cómo es una red inalámbrica, con sus conexiones.

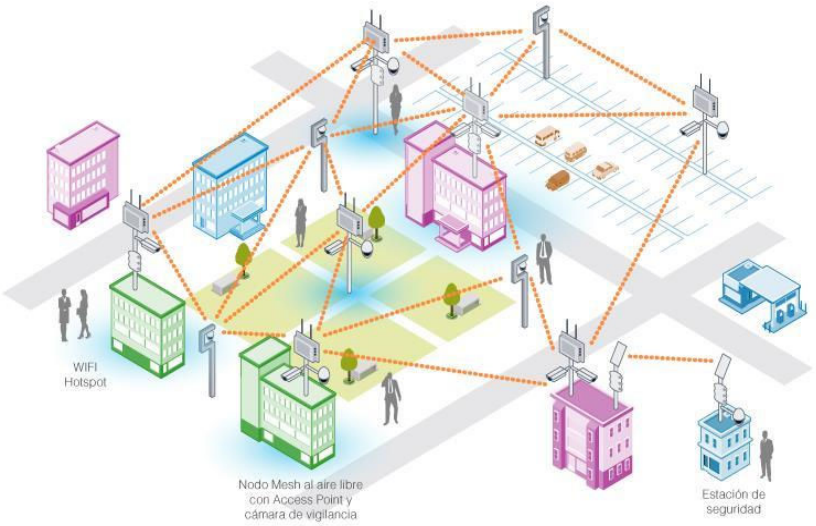


Figura 1.4. Red inalámbrica [3].

1.3.4. RED MALLADA INALÁMBRICA

Una red mallada inalámbrica o también conocida como Wireless Mesh Network (WMN) es aquella que posee nodos interconectados entre sí con un enlace punto a punto entre los diferentes nodos que pertenecen a la red, por lo que los nodos pueden estar conectados de manera parcial o total a la red, creando diferentes rutas por donde los datos se trasmiten desde el nodo transmisor al receptor.

Las redes malladas inalámbricas usan retransmisión inalámbrica multisalto que se encuentra implementada sobre una topología de malla. En la parte práctica para que los

nodos puedan comunicarse directamente entre sí, deben estar configurados en modo Ad-hoc, por lo que es importante que todos los nodos coincidan en el mismo canal de comunicación y el nombre que tenga la red.

En resumen, las redes malladas tienen las siguientes características principales:

- Permite implementar enlaces inalámbricos con un bajo coste.
- Permite aumentar con facilidad el área de cobertura añadiendo nuevos nodos dentro de la malla.
- Presenta una optimización del espectro de frecuencia ya que los nodos poseen una mayor proximidad.
- No se requiere una línea de vista entre los nodos de la red, ya que todos los nodos se comunican y evaden obstáculos.
- Las redes malladas son muy robustas.

1.3.5. DISPOSITIVOS DE COMUNICACIÓN

Hoy en día existen muchos microcontroladores de bajo coste que permiten configurar redes tipo malla, los cuales son dispositivos semiconductores en forma de circuitos integrados (CI) que poseen memorias no volátiles como volátiles, con sus respetivos puertos de entradas como de salidas digitales y analógicos.

Este tipo de microcontroladores en general son utilizados en tareas específicas, como la automatización de sistemas, configuración de redes en donde se requiere grandes cantidades de datos en la transmisión. A continuación, se describirá el microcontrolador que permite realizar todo lo antes descrito.

1.3.5.1. NODEMCU ESP32

ESP32 es un microcontrolador desarrollado por Espressif Systems, este sistema es de bajo coste y bajo consumo en un chip SoC[1] incorporando un módulo wifi y modo dual con Bluetooth.

El módulo ESP32 está constituido por una antena interna, balun para RF, amplificador de potencia, filtros y módulos de administración de energía, totalmente integrados dentro del mismo chip [4].

---

[1] **SoC (System On Chip):** circuito integrado que incorpora gran parte de los componentes de un ordenador o cualquier otro sistema informático o electrónico

Este tipo de dispositivos se encuentran diseñados para aplicaciones de electrónica, dispositivos móviles, para las IoT[2], lo cual implica un consumo de energía de bajo consumo a través de ahorro de energía. En la figura 1.5 se muestra el microcontrolador en estudio.

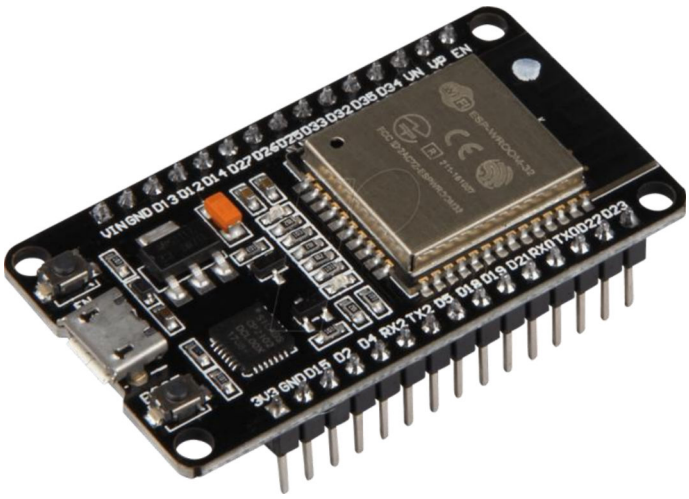


Figura 1.5. NodeMCU ESP32. [5]

Es preciso mencionar que este tipo de chip se integra dentro de las distintas placas de bajo coste que se encuentra dentro del mercado.

Entre los dispositivos más utilizados tenemos el NodeMCU ESP32 que tiene características similares a las de un arduino, contiene un módulo USB y un firmware que es capaz de proporcionar un entorno de operación para las funciones más complejas del componente.

La plataforma NodeMCU ESP32 es una tarjeta que se encuentra basada en la familia ESP32 y se encuentra orientada a la creación de prototipos IoT[2], es un proyecto de código abierto en donde el programa se distribuye libremente para ser usado y modificado por los usuarios.

---

[2] **IoT (Internet of Things):** es un sistema de dispositivos de computación interrelacionados, maquinas mecánicas y digitales, que tienen identificadores únicos y la capacidad de transferir datos a través de una red.

Las características técnicas principales del NodeMCU ESP32 son:

- **Procesador principal:** Tensilica Xtensa LX6 de 32 bits.
- **Wi-Fi:** 802.11 b / g / n (802.11n @ 2.4 GHz hasta 150 Mbit /s)
- **Bluetooth:** v4.2 BR / EDR y Bluetooth Low Energy (BLE)
- **Frecuencia de Reloj:** Programable, hasta 240MHz.
- **Rendimiento:** hasta 600DMIPS.
- **ROM:** 448KB, para arranque y funciones básicas.
- **SRAM:** 520KB, para datos e instrucciones.

El microcontrolador posee entradas y salidas analógicas y digitales.

- 36 GPIO (entradas o salidas).
- 18 entradas A/D.
- 2 salidas D/A.

En la Figura 1.6 se muestra la distribución de los puertos que posee el módulo NodeMCU ESP32.

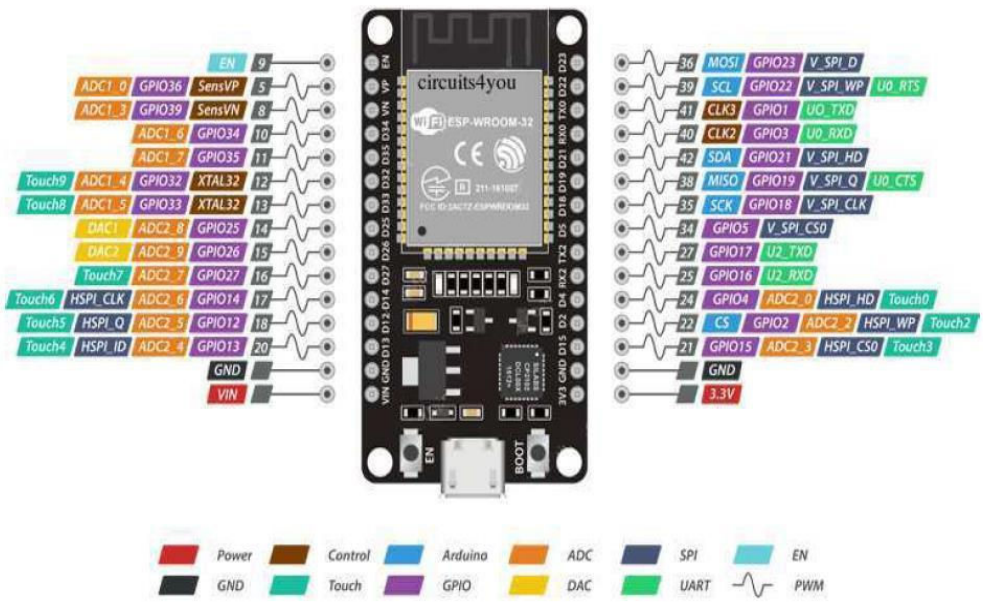


Figura 1.6. Distribución de pines del NodeMCU ESP32. [4]

Las ventajas de usar el NodeMCU ESP32 son las siguientes:

- Bajo coste.
- Bajo consumo de energía y alto desempeño.
- Soporta configurar redes inalámbricas.
- Tamaño reducido de la placa.
- Se basa en un módulo SoC.
- Amplificador de bajo ruido, robustez, versatilidad y fiabilidad.

El dispositivo NodeMCU ESP32 está diseñado para varios tipos de aplicaciones que se pueden implementar en diferentes áreas. Estos dispositivos cuentan con la más alta tecnología como un chip de baja potencia, activación de reloj, múltiples modos de potencia dinámica entre otros. A continuación, se detallarán los distintos campos donde se puede utilizar el dispositivo los cuales son:

- Internet de las cosas.
- Cámaras para video streaming.
- Reconocimiento de voz.
- Reconocimiento de imagen.
- Red de malla.
- Automatización del hogar.
- Edificios inteligentes.
- Automatización industrial.
- Agricultura inteligente.
- Aplicaciones de audio.
- Aplicaciones de atención médica.

1.3.6. TECNOLOGÍAS DE COMUNICACIÓN

En esta sección se tratan las principales características de la tecnología de comunicación que se usa junto al módulo NodeMCU ESP32.

1.3.6.1. Wi-Fi

El estándar IEEE 802.11 (Wi-Fi) define un conjunto de normas de transmisión y codificación. Desde su lanzamiento en 1997, se han ido proponiendo nuevas versiones del

estándar, en la actualidad se encuentran en la versión IEEE 802.11ay el cual brinda velocidades de transmisión de hasta 100Gbits/s.

En un principio este tipo de estándar era propiedad de cada fabricante por lo cual era difícil su implementación en proyectos. Hoy en día, el estándar IEEE 802.11 (Wi-Fi) tiene uso libre, lo que permite el desarrollo de gran cantidad de tecnología inalámbrica que se conecta entre sí. A medida que avanza el tiempo este estándar ha mejorado, siendo así que, hoy en día contamos con diferentes versiones y mejora en sus características.

A continuación, se presenta en la tabla 1.1 las diferentes versiones del estándar con las características más sobresalientes de cada uno. Se menciona los estándar b, g y n exclusivamente ya que son con los que trabajan el dispositivo NodeMCU ESP32.

Tabla 1.1. Características del estándar 802.11.

Protocolo 802.11	Frecuencia (GHz)	Velocidad (Mbit/s)	Rango (m)	Ancho de banda
802.11b	2,4	11	35	22
802.11g	2,4	54	70	20
802.11n	2,4 / 5	72,2 (20) 150 (40)	70	20 / 40

1.3.7. ENTORNOS DE DESARROLLO Y LENGUAJES DE PROGRAMACIÓN

A continuación, se detallan los principales entornos de desarrollo y lenguajes de programación que se adaptan al NodeMCU ESP32.

1.3.7.1. ARDUINO IDE (ARDUINO INTEGRATED DEVELOPMENT ENVIROMENT)

Es un entorno de desarrollo integrado (IDE)[3] y multiplataforma creado por Arduino, este entorno fue desarrollado con el lenguaje de programación java, que brinda una interfaz amigable y confiable para realizar la compilación de diferentes programas.

[3] **IDE (Entorno de desarrollo integrado):** es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

Es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. [6]

El entorno se adapta a diferentes tipos de placas aparte de arduino, lo cual ayuda a tener en un solo software todo tipo de herramientas para desarrollar diferentes tipos de proyectos.

Una gran ventaja que presenta el entorno, es que posee una amplia comunidad de desarrolladores que están en continua actualización sobre los temas relacionados a tecnología, lo que permite tener más opciones al momento de compilar proyectos.

El IDE de Arduino se adapta de buena forma con placas externas que no son netamente de la familia Arduino, y no es la excepción con el módulo NodeMCU ESP32 por lo que es una gran ventaja el disponer de todas las funcionalidades que posee el dispositivo para desarrollar distintos proyectos sobre la placa.

1.3.7.2. ESP-IDF (ESPRESSIF IoT DEVELOPMENT FRAMEWORK)

ESP-IDF es un entorno de desarrollo que fue realizado por Espressif Systems. Es el entorno oficial para la serie ESP32, su estructura se encuentra basado en el lenguaje C/C++, el cual brinda una gran cantidad de funciones establecidas lo que permite con mayor facilidad la programación del dispositivo por lo que es posible el acceso a una gran cantidad de documentación en su página web. ESP-IDF se basa en el código nativo de FreeRTOS[4] para el desarrollo de distintas funciones. Los proyectos que se encuentran basados en este framework pueden desarrollarse desde un terminal con la instalación de toolchain[5].

---

[4] **FreeRTOS:** es una variante robusta para la construcción de sistemas embebidos por las excelentes prestaciones y funcionalidades que brinda como planificador.

[5] **Toolchain:** es un conjunto de herramientas de desarrollo de software distintas que están unidas (o encadenadas) juntas por etapas específicas.

Espressif Systems además del entorno de desarrollo ESP-IDF, presentan otras plataformas que van orientadas a manejar aplicaciones de audio como ESP-ADF (Audio Development Framework), de igual manera para manejar protocolos de redes malladas como ESP-MDF (Mesh Development Framework) [7].

1.3.7.3. WHITECAT

Se encuentra basado en LUA-RTOS, es un sistema operativo que trabaja en tiempo real basado precisamente en el lenguaje LUA, el cual ha sido diseñado para sistemas embebidos que requieren funciones mínimas como flash y RAM.

Para realizar la programación con LUA-NodeMCU se puede realizar de dos formas que son:

- Usando el lenguaje LUA de forma directa.
- Usando la programación por bloques, que luego será traducidos por LUA.

Esta herramienta puede ser programada desde entornos como WhiteCat[6] IDE, este IDE permite acceder a través de un navegador web lo que permite la programación por bloques o en código. Al igual que los anteriores entornos de desarrollo acepta diferentes tipos de placas que posean microcontroladores.

1.3.7.4. MONGOOSE IDE (MONGOOSE INTEGRATED DEVELPMENT ENVIROMENT)

Es una IDE que permite trabajar con el sistema operativo Mongoose OS. Es un sistema operativo de código abierto que se utiliza para trabajar con el internet de las cosas, usa librerías especializadas que permite trabajar con soluciones en la nube como AWS IoT y Google Cloud IoT.

Mongoose soporta lenguajes de programación como JavaScript y una utilidad para portar código en C en JavaScript, reduce de forma amplia el tiempo que ocupa en descarga y modificación del código lo que resulta muy eficiente en tiempo. Para trabajar con este entorno de desarrollo se lo puede realizar a través del navegador.

---

[6] **WhiteCat:** creado para implementar casos de uso reales de IOT de una manera fácil, cubre todos los aspectos para este tipo de soluciones: hardware y software.

Además de ser un sistema operativo también trabaja con microcontroladores de baja potencia tales como ESP32 , ESP8266 , TI CC3200, STM32 las cuales tienen licencia disponible bajo GPLv2 dual / esquema de licencias comerciales [8].

1.3.7.5. COMPARACIÓN DE LOS ENTORNOS DE DESARROLLO Y LENGUAJES DE PROGRAMACIÓN

A continuación, en la tabla 1.2 se muestran las características más relevantes sobre los entornos de desarrollo y los lenguajes de programación que se adaptan con el dispositivo, y a partir de esto se elige el mejor software para usarse en la placa NodeMCU ESP32.

Tabla 1.2. Comparación de los entornos de desarrollo y lenguajes de programación.

Entorno Desarrollado	**Arduino IDE**	**Eclipse**	**WhiteCat**	**MonGoose**
Lenguaje de programación	C++	C++ Java	Lua	C++, C y JavaScript
Sistema Operativos	Windows Linux	Windows Linux	Windows	Mangoose
Hardware Soportado	Arduino, ESP8266, ESP32, etc.	Arduino, ESP32 entre otros.	ESP8266, ESP32.	STMicroelectronics Sistemas Espressif (ESP8266 Y ESP32)
Acceso a librerías	Si	Si	Si, básico	Si, básico
Tipo de acceso	Escritorio	Escritorio	Navegador web	Escritorio
Tipo de programación	Directa	Directa	Directa Por bloques	Directa
Software	Libre	Libre	Libre	Libre
Complejidad	Media	Medio	Alta	Alta

A partir del cuadro comparativo de los entornos de desarrollo y sus respectivos lenguajes de programación, se elige el entorno de desarrollo Arduino IDE, ya que brinda una complejidad media al momento de aprender a usar dicha plataforma, de la misma forma se adaptan a sistemas operativos conocidos, y lo más importante que puede trabajar con la

placa NodeMCU ESP32 y es de acceso libre lo que ayuda a obtener un desarrollo y análisis para el presente proyecto.

1.3.8. BIBLIOTECAS DE ACCESO LIBRE PARA REDES MALLADAS

A continuación, se detalla cada una de las librerías que serán utilizadas para la creación de la red tipo malla en un ambiente inalámbrico. Cabe mencionar que cada una de estas librerías son de acceso libre, lo que se puede utilizar en diferentes proyectos.

1.3.8.1. BIBLIOTECA PAINLESSMESH

Es una biblioteca que se ocupa de la creación de una red malla simple utilizando hardware como ESP8266 y ESP32, su principal objetivo es permitir al usuario que trabaje con una red mallada sin tener que estar preocupado por su estructura o la administración de la red. Painlessmesh es una verdadera red ad-hoc, lo que significa que no requiere una planificación, controlador central o enrutador. [8] Todos sus mensajes están basados en objetos JSON, lo cual permite que el código y los mensajes sean legibles y fácil de entender por el usuario; de la misma manera hace que sea sencillo el integrar la biblioteca con aplicaciones front-end[7] de JavaScript.

Uno de los puntos esenciales de esta biblioteca es que no crea una red de nodos tradicional TCP/IP, lo que la diferencia de las típicas redes malladas. PainlessMesh identifica a cada uno de sus nodos con un id único (chipId) que son de 32 bits el cual lo toma del NodeMCU ESP32 utilizando una función system_get_id () llamada en SDK [8].

Cabe mencionar que PainlessMesh está diseñado para usarse con bibliotecas nativas de ESP32 que se adaptan a placas arduino, cabe mencionar que no se hace uso de la biblioteca Arduino wifi para no tener problemas de latencia.

---

[7] **Front-end:** es la parte visible, la que muestra el diseño, los contenidos y la que permite a los visitantes navegar por las diferentes páginas.

[8]**SDK (Software Development Kit):** es un paquete de herramientas y datos que facilita e incluso permite a los programadores desarrollar programas en un lenguaje concreto o una plataforma o aplicación específica.

1.3.8.2. LIMITACIONES DE LA BIBLIOTECA PAINLESSMESH

Como toda biblioteca de acceso libre posee sus limitaciones al momento de trabajar en un ambiente real, por lo que a continuación se describen las limitaciones que presenta esta biblioteca a continuación:

- PainlessMesh evita usar delay(), ya que hace que la red tipo malla pierda estabilidad lo que puede ocasionar una caída de los nodos que pertenecen a la red, en su lugar usa programadores de tareas (TaskSheduler) lo que permite mantener una malla íntegra con todos sus nodos funcionales.

- Esta biblioteca trabaja suscribiéndose a eventos wifi.

- PainlessMesh puede ser incompatible con los programas de usuario es decir otras bibliotecas que intentan incorporarse.

- Los mensajes transmitidos pueden perderse o descartarse debido a la circulación de un alto tráfico en la red.

- Al trabaja con dispositivos de bajo coste implica tener una memoria de procesamiento limitado, por lo que hay que ser conservador con la cantidad de mensajes enviados en especial con los mensajes de difusión [9].

1.3.8.3. LIBRERIA ARDUINOJSON

ArduinoJson es una librería de JSON la cual se utiliza de forma eficiente en sistemas integrados. Esta biblioteca a primera instancia fue pensando en Arduino, pero no está vinculado a las bibliotecas Arduino, por lo que se puede usar esta biblioteca en cualquier otro proyecto de C ++ [10].

ArduinoJson incorpora entre sus funciones el serializar y deserializar los objetos en una forma sencilla, lo que permite intercambiar la información de forma de texto plano facilitando al usuario final interpretar los datos de la mejor manera. Esta librería es compatible con múltiples placas de desarrollo como arduino uno, nano, mega, micro, Leonardo, due, ESP8266, ESP32 entre otros.

Una de las ventajas primordiales de esta librería es que son de acceso libre (Open Source) y se pueden añadir a diferentes proyectos con solo añadir en el gestor de librerías del entorno del desarrollo como Arduino IDE.

1.3.8.3.1 FICHERO JSON

JavaScript Object Notation es un formato de texto plano el cual permite almacenar datos estructurados que son estándar para el intercambio de información entre los sistemas de comunicación.

Estos ficheros se pueden observar de forma cotidiana en la mayoría de las páginas web, presentando ventajas con respecto a otros formatos como XML[10] por su sencillez y su forma ligera de implementación, lo que permite un menor consumo de ancho banda debido al tipo de formato al momento de intercambiar datos y brinda una menor carga entre los dispositivos que actúan formando la red. Su formato es sencillo de leer por el usuario, debido a que tiene cuatro tipos de objetos básicos que son texto, números, booleanos y null los que se pueden agrupar en objetos o arrays.

1.3.8.4. BIBLIOTECA TASKSCHEDULER

TaskScheduler es una librería liviana y de multitarea cooperativa que son usados en microcontroladores como ESP8266 y ESP32. Esta librería permite realizar muchas tareas las cuales se describen a continuación:

- Permite la ejecución periódica de tareas, con un periodo de ejecución dinámico el cual puede ser milisegundos (estado predeterminado) o microsegundos.

- Permite establecer el número de iteraciones de forma limitada o ilimitada.

- Permite la ejecución de tareas en una secuencia predefinida.

- Permite cambiar dinámicamente los parámetros de ejecución de las tareas como frecuencia, número de iteraciones, método de devolución de llamada.

---

[10]**XML (Extensible Markup Language):** lenguaje marcado que define reglas para la codificación de documentos.

- Soporta la priorización de capas.

- Soporta el ID de tareas.

- Soporta un punto de control para el manejo de errores.

- Permite invocar las tareas controladas por eventos a través de una solicitud de estado.

1.3.8.5. TAREAS

Es la parte lógica del programa, que requiere una ejecución programada. Las tareas realizan ciertas funciones que requieren la ejecución constante, pueden estar controlando hardware en específico.

Para fines de ejecución, las tareas se encuentran vinculadas a cadenas de ejecución, que son procesadas por el programador en el orden que fueron agregadas. Cada tarea realiza su función mediante un método de devolución de llamada. El programador hace la llamada de tareas de forma periódica, hasta que la tarea se deshabilita o se queda sin interacciones.

Las tareas son responsables de respaldar la multitarea cooperativa al ser “buenos vecinos”, es decir, ejecutar sus métodos de devolución de llamada rápidamente de una manera no bloqueante y liberar el control nuevamente al planificador lo antes posible [11].

1.3.8.5.1 Programadores

El programador (Scheduler) está ejecutando los métodos de devolución de llamada de las tareas en el orden que se agregaron a la cadena, en forma ascendente. El programador se detiene y existe después de procesar la cadena una vez para permitir que se ejecute otras instrucciones que se encuentran en el código principal del método de programación. [12]

1.3.8.6. ASYNCTCP (LIBRERÍA TCP ASÍNCRONA PARA ESP32)

AsyncTCP es una librería TCP completamente asíncrona, la que permite un entorno de red de conexión múltiple y sin problemas para el uso en el ESP32. A la vez es un ecosistema de código abierto que permite desarrollar proyectos en los más populares sistemas operativos como Mac OS, Windows, Linux entre otros.

Esta librería nos permite realizar una transferencia de datos de forma confiable a través de TCP[11]. Esta librería se debe de utilizar con la biblioteca ASync, permite la entrada y salida de la red de alto rendimiento en un solo subproceso. AsyncTCP crea una nueva instancia para administrar conexiones TCP establecidas local y remotamente.

[11]**TCP (Transmission Control Protocol):** es uno de los protocolos fundamentales en Internet, nos permite que las aplicaciones puedan comunicarse con garantías independientes de las capas inferiores del modelo TCP/IP.

2. METODOLOGÍA

En el capítulo anterior, se han estudiado aspectos importantes de las redes tipo malla en un ambiente inalámbrico, analizando de forma breve el estándar IEEE 802.11 que se adapta a los módulos de bajo coste como el NodeMCU ESP32. A la vez se analizan los diferentes tipos de entorno de desarrollo como los lenguajes de programación.

También cabe mencionar que se realiza el estudio de las diferentes bibliotecas de acceso libre que permita la creación de una red mallada con sus distintas características que brinda cada una de las bibliotecas para la implementación dentro este proyecto de titulación.

En el presente capítulo se presenta la implementación de la red mallada con el uso del estándar 802.11 en dispositivos de bajo costo como es el módulo NodeMCU ESP32, con la finalidad que se realice el análisis de las métricas que afectan al rendimiento de la red. Se cargarán en los módulos cada una de las librerías que permitan tener una red tipo malla. Adicional se desarrollan scripts para ponerle a la red a trabajar y obtener resultados en los diferentes escenarios planteados.

2.1. REQUERIMIENTOS PARA LA IMPLEMENTACIÓN DE LA RED TIPO MALLA

En esta sección se describe tanto el hardware como el software que se utiliza para la implementación de la red antes mencionada. A continuación, en la tabla 2.1 se procede a detallar cada requerimiento que se necesita en el hardware para comenzar con la implementación del proyecto.

Tabla 2.1. Requisitos en hardware.

DISPOSITIVOS	CARACTERÍSTICAS
NodeMCU ESP32	Procesador Xtensa LX6 de 32 bits. SRAM 520KB ROM 448KB para arranque y funciones básica. Conectividad Wi-FI
Laptop	Procesador Intel Core I3 mínimo RAM de 2GB mínima. Espacio mínimo de 512MB Conectividad Wi-Fi.

Una vez detallado los requerimientos en hardware, a continuación, en la tabla 2.2 se describen los requisitos en software que se necesita en cada uno de los dispositivos antes mencionados.

Tabla 2.2. Requisitos en software.

DISPOSITIVOS	SISTEMA OPERATIVO	ENTORNO DE DESARROLLO	LENGUAJE DE PROGRAMACIÓN
NodeMCU ESP32	-	IDE Arduino	C++
Laptop	Windows 10	IDE Arduino	C++

Para el desarrollo de la presente red tipo malla inalámbrica se dispone de cuatro nodos NodeMCU ESP32 con su respectivo módulo wifi cada uno ya incorporado en el dispositivo. La red cambia dependiendo de los escenarios en donde se realizan las pruebas. La implementación de la red mallada requiere la instalación de diferentes herramientas de software como:

- Sistema operativo Windows 10
- Entorno de desarrollo Arduino.
- Librerías de acceso libre como se detalló en el primer capítulo.

2.2. IMPLEMENTACIÓN Y DESARROLLO DE LA RED MALLADA INALÁMBRICA.

La implementación de una red tipo malla se realiza en varias etapas, como se detalla en la figura 2.1.

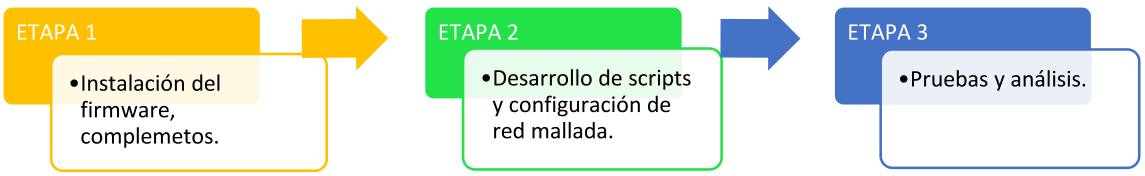


Figura 2.1. Etapas de la implementación de la red.

La implementación consta de tres etapas en donde como primer paso se realiza la instalación de entornos de desarrollo y las librerías de acceso abierto. En la segunda etapa se desarrolla scripts con la ayuda de las librerías antes mencionadas y se carga en los módulos NodeMCU ESP32. Al final se realiza diferentes pruebas cambiando la ubicación de los nodos hasta formar las topologías de prueba.

2.2.1. PLUGIN NODEMCU ESP32 EN EL ENTORNO DE DESARROLLO

Al instalar el entorno de desarrollo de arduino de forma predeterminada no viene pre cargado con los modelos de dispositivos de la familia ESP32 para su uso, debido a esto se requiere la instalación del plugin[12] con los modelos de los dispositivos ESP32 para su respectivo manejo dentro del IDE de arduino. Uno de los pasos es colocar en el gestor de URL adicionales de tarjeta de Arduino, es la dirección donde se encuentra el archivo a descargarse para así proceder con la instalación dentro del IDE.

A continuación, en la figura 2.2 se observa al gestor de las tarjetas de la familia de microcontroladores ESP32 ya en el entorno de Arduino para ser instalada.

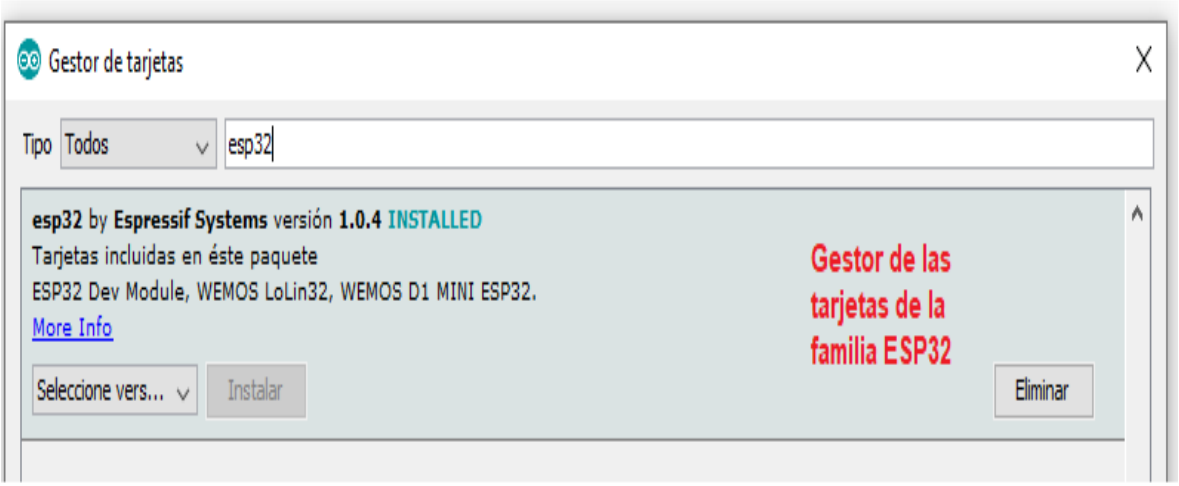


Figura 2.2. Gestor de tarjetas de la familia ESP32.

Finalmente, instalado el firmware para trabajar con la familia ESP32, ya se puede elegir entre las diferentes placas basadas en la plataforma ESP32 como se observa en la Figura 2.3.

[12] **Plugin:** es una aplicación que en un programa informático, es el que añade una funcionalidad adicional o una nueva característica al software.

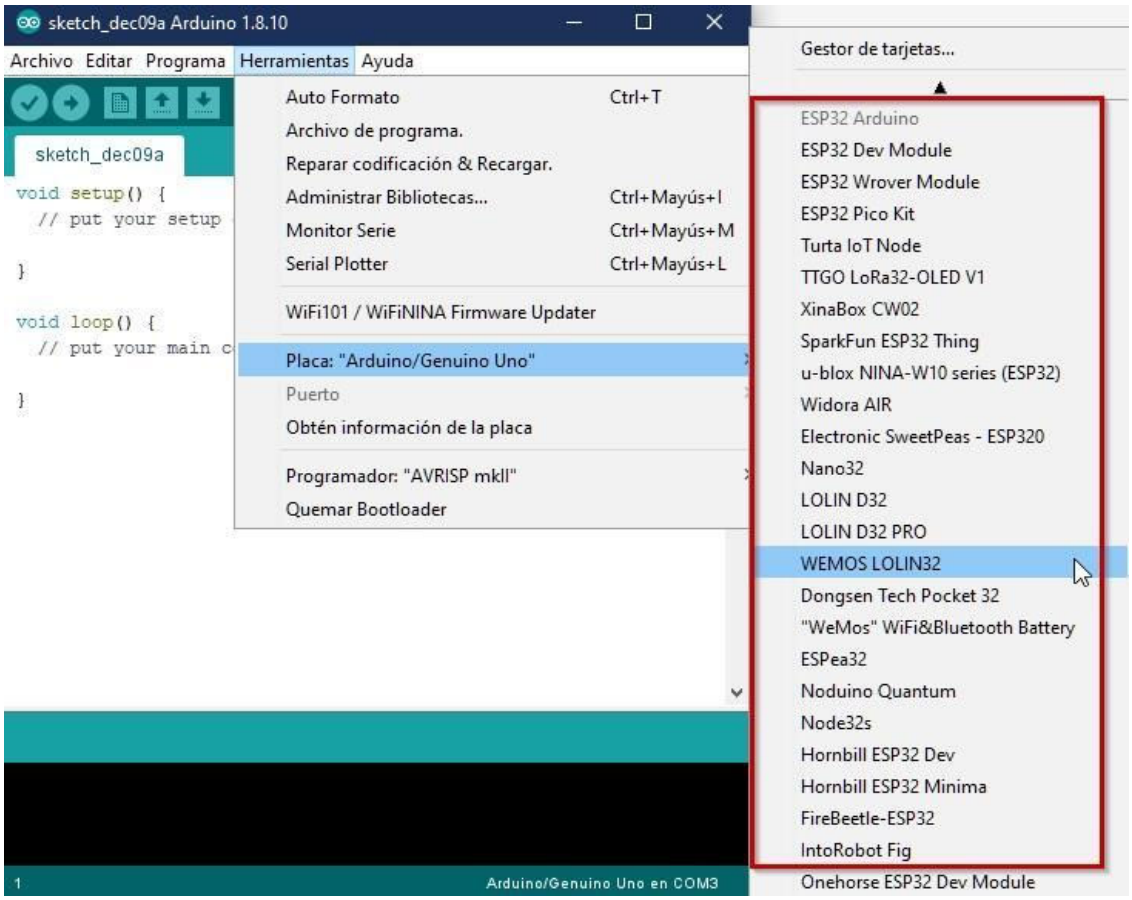


Figura 2.3. Plataforma ESP32 basado en el IDE Arduino.

Una vez realizado este proceso, se procede a desarrollar los scripts con la ayuda de las librerías de acceso libre.

2.2.2. DESARROLLO DE LA RED MALLADA INALÁMBRICA

En esta sección se detalla la creación de la red tipo malla haciendo uso de las librerías de acceso libre antes mencionadas. Se describe cada uno de los procesos que se sigue para la ejecución de las tareas dentro de la red, el envío y recepción de datos, así como las etapas de detección y descubrimiento de cada uno de los nodos (NodeMCU ESP32).

2.2.2.1. CREACIÓN DE LA RED MALLADA

El usar la biblioteca painlessMesh nos proporciona la capacidad de separar la creación, detección, el envío y recepción de la red tipo malla, y permite añadir seguridad a la red.

En el siguiente código 2.1 se establece las variables para inicializar la red mallada como son el ssid[13], contraseña y puerto. Todos los nodos que pertenezcan a la red tienen que establecer esta información.

Código 2.1. Creación de la red tipo malla.

```
#include "painlessMesh.h"
painlessMesh  malla;
#define   MESH_PREFIX     "redMallaEPN"
#define   MESH_PASSWORD   "mallaTesisEPN"
#define   MESH_PORT       5555
```

A continuación, se detalla porqué se utiliza estos parámetros:

- Se incluye la biblioteca painlessMesh y se crea el objeto malla para iniciar la creación de la red.

- MESH_PREFIX: es el SSID o nombre de la malla, todos los nodos compartirán el mismo AP SSID.

- MESH_PASSWORD: es la seguridad o contraseña wifi que se le da a la malla.

- MESH_PORT: es el puerto TCP en el que se ejecuta la red tipo malla. El valor 5555 es un valor determinado sino se especifica otro.

2.2.2.2. GESTIÓN DE LOS PROGRAMADORES DE TAREAS

Se ejecuta el programador de tareas dentro del código con el objetivo de que exista una ejecución de tareas periódicas dentro de la red creada anteriormente, este tiempo designado para la ejecución de tareas se determina en segundos.

Esta gestión se la realiza de forma continua al momento de estar detectando la red para descubrir nuevos nodos que ingresen o salgan de la red, y a la vez permitir la autoconfiguración de la red.

---

[13] **SSID (Service Set IDentifier):** es el nombre que identifica a una red inalámbrica (Wi-Fi), y a la vez viaja junto con cada paquete de información.

En el código 2.2 se visualiza el proceso de inicialización del programador de tareas, con un específico evento, en este caso el envío de mensajes de modo broadcast dentro de la red tipo malla.

Código 2.2. Programador de tareas de envió de mensajes.

```
Scheduler usuarioPlanificador;
painlessMesh  malla;

Task tareaEnviarMensaje( TASK_SECOND * 1 , TASK_FOREVER, &enviarMensaje );
```

El instanciar Scheduler como un usuario planificador dentro de la clase, nos permite realizar el control de distintas tareas, en este caso como se puede observar en el código 2.2 nos permite hacer uso del método Task, el cual permite crear tareas en específico, en nuestro caso el de enviar los mensajes dentro de la red creada, con distintas propiedades que se detallan a continuación:

- **TASK_SECOND:** se establece el tiempo que se demora en enviar la tarea en este caso en segundos, de forma predeterminada viene en milisegundos.

- **TASK_FOREVER:** se establece el tiempo que se realice la tarea de forma indefinida, hasta que el programa termine de ejecutarse.

- **&enviarMensaje:** es el método de devolución de llamada, en este caso el método de envío de mensajes, este mensaje tiene que estar en formato JSON [14]

2.2.2.3. ENVÍO DE MENSAJES

El método de envío de mensajes consta de distintos parámetros para que la información se trasmita entre los nodos.

A continuación, se describe los parámetros para el envío de los mensajes dentro de la red inalámbrica:

---

[14] **JSON (JavaScript Object Notation):** formato de texto el que es utilizado para un intercambio de datos.

- **Mensaje (msg):** es el mensaje que va a ser trasmitido por la red.
- **getNodeId:** es el identificador del chip NodeMCU ESP32, este identificador se encuentra asignado por el fabricante del chip, cabe mencionar que es un número único dentro de la red.
- **Nombre del nodo:** es el alias del nodo para ser identificado de manera fácil dentro de la red.
- **Envío de mensajes:** es el tipo de tráfico que se desea enviar a través de la red.

Existe dos tipos de tráfico que se puede enviar y que se describe a continuación.

- **sendSingle (dest, msg):** este tipo de tráfico nos permite enviar el mensaje a un solo nodo en particular. Dentro de las propiedades que se utiliza, está el dest que es el identificador del nodo que recibe, la otra propiedad msg es el mensaje que se desea enviar por la red.

- **sendBroadcast (msg):** este tipo de difusión nos permite enviar el mensaje a todos los nodos que se encuentran activos dentro de la red, esto quiere decir, que cada nodo que se encuentre presente en la red recibirá el mensaje que se envía desde el nodo transmisor.

  Todo lo antes mencionado se podrá observar en el código 2.3, en este caso se muestra el método de difusión broadcast.

Código 2.3. Propiedades del método de envío de mensajes.

```
void enviarMensaje() {
  String msg = "Hola desde el nodo: ";
  msg += malla.getNodeId();
  msg += ", alias";
  msg += nombreNodo;
  malla.sendBroadcast( msg );
  tareaEnviarMensaje.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
```

Como se mencionó anteriormente los mensajes que se transmiten por la red deben llevar una estructura y eso se explica a continuación:

2.2.2.3.1 Estructura del mensaje usando JSON

La red tipo malla para estructurar los mensajes, usan el formato JSON (JavaScript Object Notation), donde los mensajes que son enviados entre los nodos se subdividen en mensajes de control y mensajes de usuario cada uno de estos se detallan a profundidad a continuación.

- **Mensajes de control:** este tipo de mensajes son enviados entre los nodos de la red, para intercambiar información correspondiente al enrutamiento y la sincronización del tiempo de los relojes de cada uno de los dispositivos en la red, las sincronizaciones de los relojes se dan de forma automática.

- **Mensajes de usuario:** son mensajes que estrictamente los genera el usuario y son enviados mediante un objeto JSON ya sea a un solo nodo (unicast), o a varios nodos (broadcast) los cuales serán trasmitidos a través de la red.

En la figura 2.4 se presenta el esquema que usa JSON para estructurar los mensajes que van a ser trasmitidos por la red tipo malla.

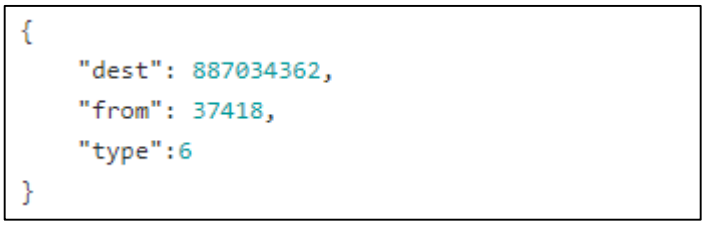


Figura 2.4. Estructura JSON.

Donde “dest” es el nodo destino donde se va a enviar el mensaje, “from” pertenece al nodo anterior por donde pasó el mensaje, mientras que “type” es el tipo de mensaje que se envía, adicionalmente se puede enviar el timestamp que es la hora a la que se envía el mensaje.

2.2.2.3.2 Enrutamiento de los mensajes de control

La información de enrutamiento que toma el mensaje al pasar de un nodo a otro se comparte en forma de mensajes de sincronización entre los nodos.

Cada uno de los nodos informan a los nodos vecinos sobre otros nodos que se encuentran conectados de forma directa y todas las respectivas subconexiones que existen entre ellos.

Gracias a la información compartida entre los nodos de la red, se puede tener una imagen en tiempo real de todos los nodos que se encuentran conectados en la red. Con esta información proporcionada se puede tener conocimiento sobre los nodos que se encuentran activos dentro de la red.

Los datos recolectados por los nodos se van actualizando cada 3 segundos. Dentro de la información recolectada se encuentra la de sincronización, que está constituida por un par de mensajes que siguen el siguiente proceso:

- En primer lugar, el nodo que va a trasmitir envía un NODE_SYNC_REQUEST a los nodos vecinos. El mensaje enviado es de tipo 5 (“type”:5).

- A continuación, los nodos vecinos responden con un NODE_SYNC_REPLY, el mensaje de respuesta es de tipo 6 (“type”:6).

Tanto los mensajes de tipo 5 y tipo 6 poseen el mismo esquema, cada subconexión de los nodos tiene un campo subs el que contiene sus subconexiones. Las subconexiones contienen el "nodeId" del nodo conectado y (opcionalmente) si ese nodo es un nodo raíz ("root": true).

2.2.2.3.3 Enrutamiento de los mensajes de usuarios

Los mensajes de usuario se pueden enviar a través de la red, pueden ser de dos tipos y se detallan a continuación:

- **Mensaje de dirección única**

  Este tipo de mensajes se encuentran etiquetados con la dirección del nodo origen y la dirección del nodo destino, adicional el tipo de mensaje es de número 9, este número se agrega a la cadena que contiene el mensaje.

A continuación, se puede observar en la figura 2.5 como se encuentra constituido el mensaje en esquema JSON.

Figura 2.5. Enrutamiento mensajes dirección única.

- **Mensajes de difusión**

  El tipo de transmisión es casi idéntica a los de un mensaje de dirección única, la diferencia que se presenta con este tipo de mensaje es que es de tipo 8 y el destino es idéntico al ID del nodo receptor. Al volver a reenviar un mensaje de este tipo, el campo de destino se cambia al siguiente nodo al que se envía el mensaje.

2.2.2.4. RECEPCIÓN DE MENSAJES

Para la recepción de los mensajes que llegan a los nodos de la red se hace uso del método receivedCallback, este método es una rutina que se activa con una devolución de llamada cada vez que se recepta un mensaje.

Dentro del método receivedCallback tenemos las propiedades de from y msg que se explicaran a continuación.

- **from:** es el identificador (chip ID) del nodo remitente es decir es del nodo transmisor.
- **msg:** es la cadena que contiene el mensaje, el mensaje receptado puede ser bien una cadena de texto o datos binarios en un formato JSON.

En el código 2.4 se presenta como se usa el método receivedCallback para la recepción de los mensajes.

Código 2.4. Propiedades del método de la recepción de mensajes.

```
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("startHere: Recibiendo mensajes desde el nodo: %u msg=%s\n", from, msg.c_str());
}
```

2.2.2.5. CONEXIÓN DE NUEVOS NODOS A LA RED

La conexión de nuevos nodos a la red se lo realiza mediante la programación de un método que se activa cada vez que el nodo local descubre una nueva conexión. Esta devolución de llamada se ejecuta cuando se conecta un nuevo nodo a la red.

La devolución de llamada posee la siguiente estructura que se observa a continuación en el código 2.5.

Código 2.5. Conexión de nuevos nodos a la red tipo malla

```
void newConnectionCallback(uint32_t nodeId) {
    Serial.printf("--> startHere: Nueva Conexion, Id del nodo: = %u\n", nodeId);
}
```

A continuación, se explica el funcionamiento del código 2.5 donde la conexión del nuevo nodo se basa en la obtención del nuevo chip ID del módulo NodeMCU ESP32 que desea pertenecer a la red.

Esta solicitud se lo realiza mediante el método newConnectionCallback () que tiene dentro de sus parámetros el nodeId que es el nuevo ID que se obtendrá al momento que se conecta un nuevo nodo a la red; este nodo pasa a ser un nuevo miembro de la red.

2.2.2.6. AUTOCONFIGURACIÓN DE LA RED

La autoconfiguración entra en funcionamiento cada vez que existe un cambio dentro de la topología de la red, el método changedConnectionsCallback () se activa con una devolución de llamada.

Este cambio se da cuando ingresa o sale un nuevo nodo a la red, ocasionando que la red se modifique. Esta modificación se la conoce como autoconfiguración de la red dándose de forma independiente sin la necesidad de tener administradores de red que realicen esta funcionalidad.

A continuación, se muestra en el código 2.6 la estructura con la que se programa la funcionalidad de autoconfiguración de los nodos.

Código 2.6. Método de autoconfiguración de la red.

```
void changedConnectionCallback() {
  Serial.printf("Cambios en la conexion\n");
}
```

Cabe mencionar que dentro de este método no se pasa ningún parámetro, ya que esto solo emite una señal.

2.2.2.7. SINCRONIZACIÓN DE LOS RELOJES DE LA RED

El estar trabajando en una red tipo malla, se necesita tener una sincronización de tiempos que compara y corrige la hora actual del dispositivo con la hora de la malla, para esta sincronización se hace uso del método nodeTimeAdjustedCallback ().

A continuación, en el código 2.7 se puede observar cómo se emplea este método en la programación de la red.

Código 2.7. Sincronización hora de la red tipo malla

```
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Ajuste de tiempo %u. Offset = %d\n", malla.getNodeTime(),offset);
}
```

Al emplear el método para la sincronización de la hora de un dispositivo con la hora de la malla, se pasa el parámetro offset que es el delta o intervalo de ajuste que se ha calculado y se aplica al reloj local del dispositivo NodeMCU ESP32.

2.2.2.8. ACTUALIZACIÓN DE LA RED

Un aspecto de gran importancia es el tener actualizado el estado de los nodos dentro de la red, y para esta funcionalidad se usa el método update().

Update() nos permite tener la red actualizada, obteniendo información de como se encuentra la red en cada momento, adicional nos envía información de mantenimiento de la red, esta devolución de llamado se realiza de forma repetitiva ya que se encuentra dentro de un loop.

Update () mantiene a la red actualizada de forma permanente, informándonos como se encuentra el estado de cada dispositivo NodeMCU ESP32 que pertenece a la red.

A continuación, en el código 2.8 se puede observar cómo se usa el método antes mencionado dentro de la programación de la red.

Código 2.8. Método de la actualización de la red.

```
void loop() {
  malla.update();
}
```

2.2.3. CÓDIGO DE LA APLICACIÓN

Una vez visto cada uno de los métodos que intervienen para la creación de una red mallada inalámbrica con el uso de las librerías de acceso libre, se procede a describir como están desarrollados los scripts en cada uno de los tipos de tráfico que se van a enviar por la red.

2.2.3.1. SCRIPT TRANSMISIÓN TIPO BROADCAST

Para la comunicación tipo broadcast se crea el script, que permite las siguientes funciones que son el crear una red tipo malla inalámbrica con seguridad, enviar datos tipo de comunicación broadcast, autoconfiguración de los nodos, actualización de la red, ajuste de sincronización, cada una de estas funciones caracteriza a la red tipo malla.

Para el envío del mensaje se hace uso de las librerías arduinoJson, taskscheduler, definiendo un planificador que tiene como tarea el envío de mensajes en modo broadcast a los nodos vecinos de la red antes creada. Cada uno de estos métodos trabajan con las librerías de acceso libre las cuales permiten la administración autónoma de la red.

Una vez definido el proceso de trabajo de la red, el script queda listo para ser cargado en el dispositivo que trabajara como transmisor, cabe mencionar que al script antes de ser cargado se compila para que no exista errores de programación.

2.2.3.2. SCRIPT RECEPCIÓN TIPO BROADCAST

A diferencia del nodo que transmite, los nodos de recepción de los mensajes sufren ligeros cambios dentro del código. Este cambio se presenta en el planificador de tareas específicamente en la tarea de envío de mensajes, esta tarea se elimina del código para que no envíe mensajes a los otros nodos de la red.

Se mantiene las funciones que son la creación de la red malla, métodos de recepción de mensajes, nuevas conexiones, cambios en la topología de la red, sincronización de la hora

de la malla y la ejecución de la red malla para que trabaje de forma transparente con el nodo transmisor.

2.2.3.3. SCRIPT TRANSMISIÓN TIPO UNICAST

Para la comunicación tipo unicast se desarrolla el script, que permite las siguientes funciones que son el crear la red tipo malla inalámbrica con seguridad, enviar datos con tipo de comunicación unicast, autoconfiguración de los nodos, actualización de la red, ajuste de sincronización, cada una de estas funciones detalladas anteriormente caracterizan a este tipo de redes.

El código unicast comparte muchas similitudes con el código desarrollado para la transmisión broadcast con ligeros cambios que se detallaran a continuación:

- Para el envío de mensajes unicast entre los nodos se utiliza el tipo de difusión sendSingle () que permite enviar la información a un nodo en específico.

2.2.3.4. SCRIPT RECEPCIÓN TIPO UNICAST

El script para la recepción de mensajes tipo unicast sufre pequeños cambios dentro del código, se elimina la tarea de transmisión de mensajes debido que solo se encargara de receptar los mensajes. Se mantiene las funciones de creación de la red malla, métodos de recepción de mensajes, nuevas conexiones, cambios en la topología de la red, sincronización de la hora de la malla y la ejecución de la red malla.

2.2.4. CARGA DE SCRIPTS EN LOS DISPOSITIVOS NODEMCU ESP32

Finalizado el proceso de programación de los diferentes parámetros que tendrá la red tipo malla, se procede a cargar los scripts en cada uno de los dispositivos de bajo coste NodeMCU ESP32 para someter a la red a las pruebas.

2.2.4.1. CONEXIÓN DISPOSITIVOS NODEMCU ESP32

Como se mencionó, el prototipo se implementa usando dispositivos de bajo coste como las placas NodeMCU ESP32, los que actúan como nodos dentro de la red mallada inalámbrica, para su funcionamiento se procede a la conexión de los dispositivos NodeMCU ESP32 con el ordenador.

Esta conexión se realiza mediante el puerto USB por lado del ordenador y micro USB tipo B por el lado del NodeMCU ESP32, todo lo antes mencionado se puede observar en la figura 2.6.

Figura 2.6. Conexión física entre el ordenador y NodeMCU ESP32.

Una vez conectado se procede a observar el puerto de comunicación que fue asignado por parte del ordenador al conectarse. La elección del puerto de comunicación se la realiza dentro del entorno de desarrollo Arduino.

A continuación, se podrá observar en la figura 2.7 el proceso de elección del puerto de comunicación serial.

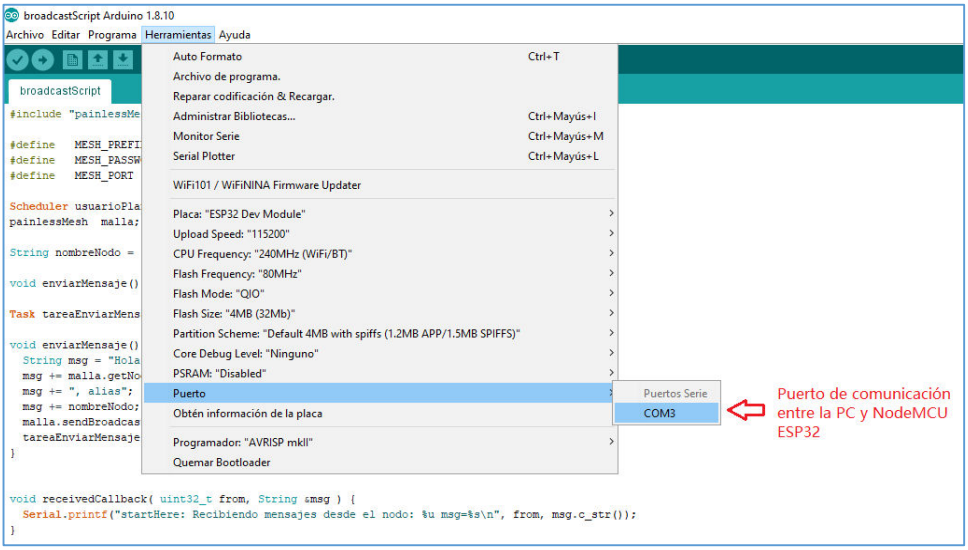


Figura 2.7. Elección puerto de comunicación en el IDE Arduino.

2.2.4.2. CONFIGURACIÓN DEL NODEMCU ESP32 CON EL IDE DE ARDUINO

Para cargar los scripts en los dispositivos NodeMCU ESP32 se configuran varios parámetros que son de importancia para poder tener una conexión exitosa entre el ordenador y el dispositivo. A continuación, se muestra las configuraciones de los diferentes parámetros que se necesita configurar dentro del entorno de desarrollo Arduino.

- **Gestor de tarjetas**

El primero paso que se debe realizar es elegir el tipo de tarjeta con la que se va a trabajar, en este caso se usa la NodeMCU ESP32.

El entorno de desarrollo de Arduino posee una gran variedad de placas que se puede elegir, cabe mencionar que de forma predeterminada no se encuentra instalada las tarjetas de la familia ESP 32, anteriormente se indicó como es el proceso de instalación de esta familia de tarjetas. Se enfoca en esta sección en la elección de la tarjeta adecuada que se acopla directamente con las características que nos brinda la placa NodeMCU ESP32 a nivel de hardware. En la figura 2.8 se podrá observar la elección del dispositivo que se encuentra dentro del gestor de tarjetas.

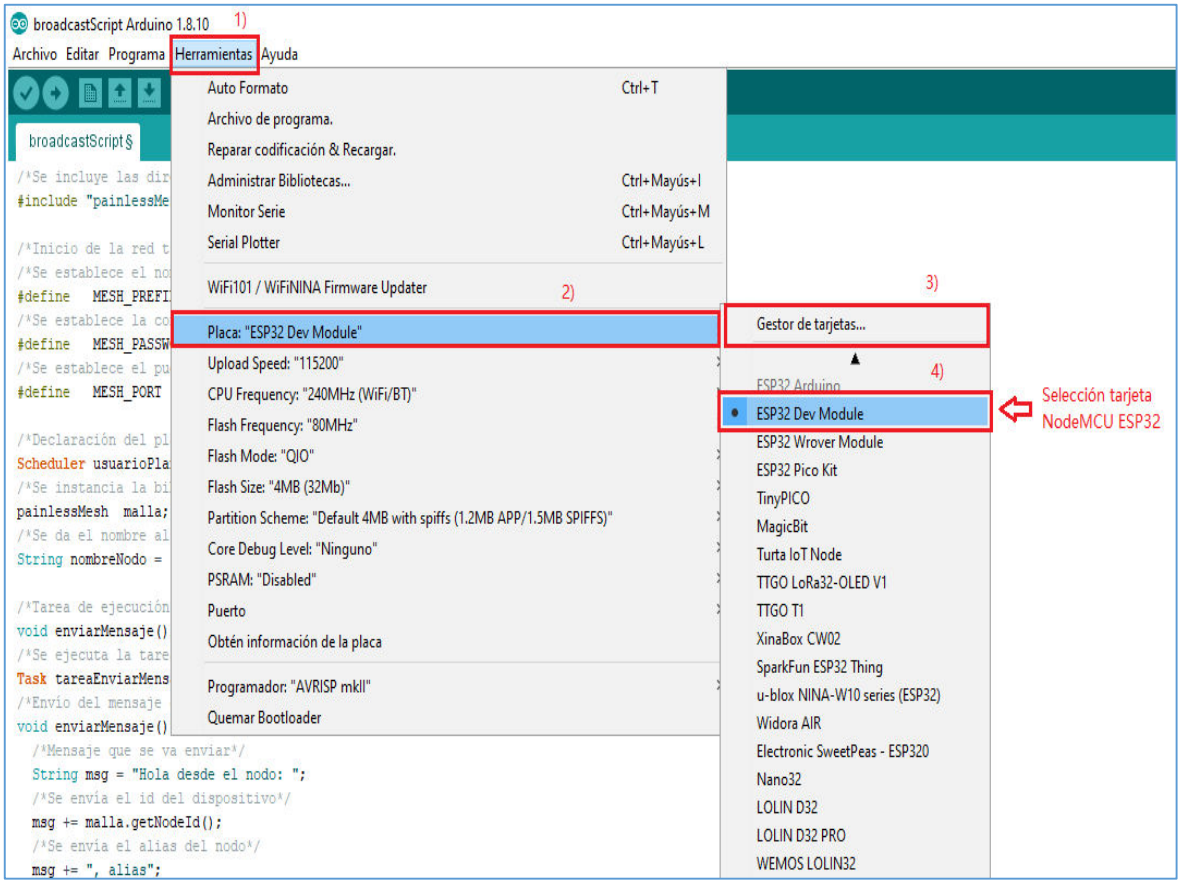


Figura 2.8. Selección de la tarjeta NodeMCU ESP32.

Una vez elegido el puerto de comunicación, como la tarjeta NodeMCU ESP32 se procede a cargar los scripts realizados en los cuatro dispositivos que son parte de la red tipo malla inalámbrica. En una de las tarjetas se carga el script desarrollado exclusivamente para transmitir, mientras que en las otras tres tarjetas se carga el script de recepción.

A continuación, en la figura 2.9 se puede observar el proceso de carga de los scripts en los dispositivos que pertenece a la red tipo malla inalámbrica. Cada uno de los pasos mencionados anteriormente se aplica para todos los dispositivos que se encuentran en la red.

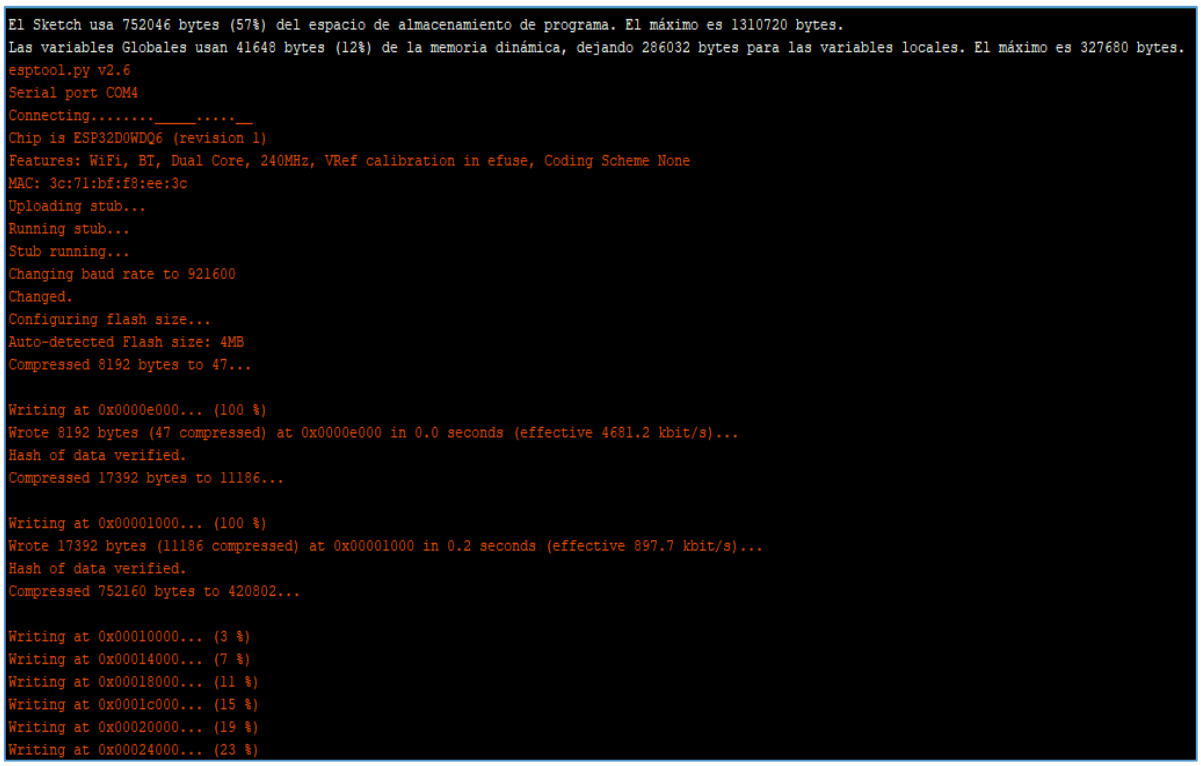


Figura 2.9. Carga del script en el dispositivo NodeMCU ESP32.

Al finalizar el proceso de carga de los scripts, los dispositivos quedan listos para realizar las pruebas de funcionalidad, autoconfiguración, conectividad y rendimiento de la red tipo malla inalámbrica.

3. RESULTADOS Y DISCUSIÓN

En esta sección se presentan las pruebas realizadas en los diferentes escenarios planteados para medir pruebas de funcionalidad, autoconfiguración, conectividad y rendimiento de la red. Los resultados obtenidos de las pruebas se presentarán en base a tablas y gráficos estadísticos, lo que permite evaluar el desempeño de la red tipo malla inalámbrica en dispositivos de bajo coste.

Para las pruebas de rendimiento de la red se usa a nivel de hardware los siguientes elementos:

- 2 ordenadores
- 4 placas NodeMCU ESP32.
- Fuentes de alimentación (4 baterías recargables de litio a 3,7v – 4800mAh y porta batería).

Mientras que los elementos de software necesarios para realizar las pruebas de rendimiento son los siguientes:

- Ordenador con Windows 10 y el IDE de Arduino
- Ordenador con Linux y con el analizador de tráfico Wireshark.
- Analizador de redes Aircrack-ng

A continuación, se presenta en forma gráfica como se encuentran distribuidos los diferentes escenarios y los elementos de hardware usados para obtener los datos para la evaluación del rendimiento de la red.

Escenario A

Todos los nodos están al alcance de los otros nodos, todos se ven entre sí, se coloca un ordenador para poder observar los datos que llegan a los nodos que pertenecen a la red tipo malla; al mismo instante se coloca otro ordenador con los programas para analizar la red el cual se encuentra configurado en modo monitor escuchando todo tipo de paquetes que estén transcurriendo por la red en este caso redMallaEPN y ser capturado, a continuación, se muestra en la figura 3.1 el escenario planteado anteriormente.

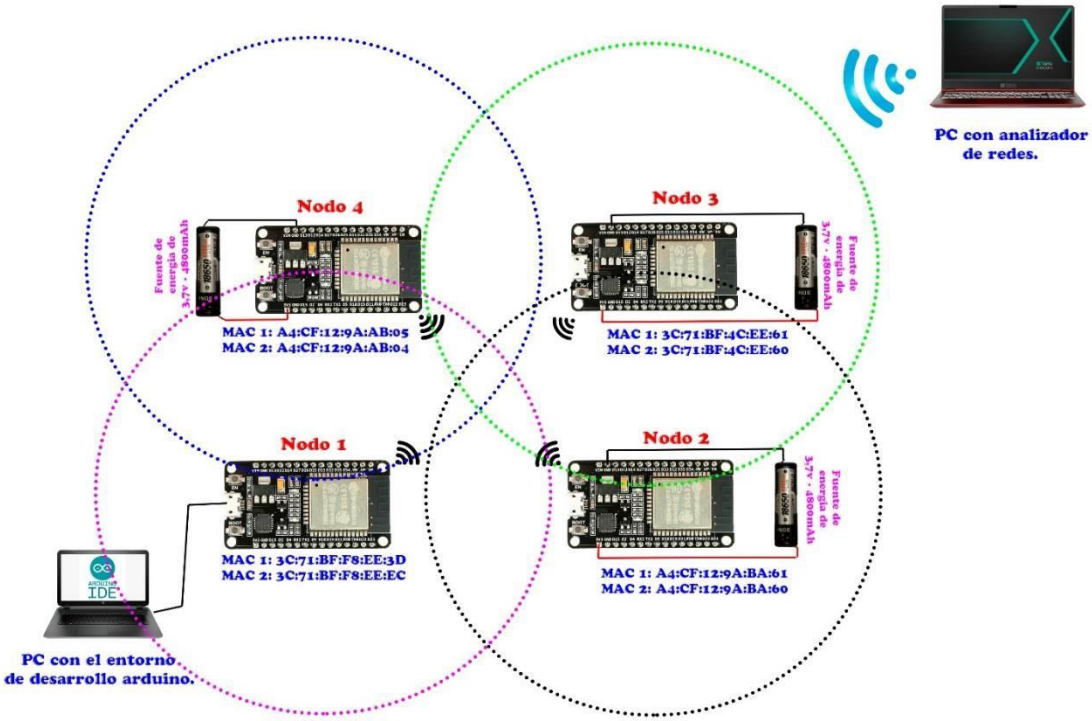


Figura 3.1. Escenario A red tipo malla con dispositivos NodeMCU ESP32.

Escenario B

En este escenario los nodos se ven en pares, de la misma manera que el escenario A tiene un ordenador que se conecta a los nodos para observar vía consola mediante el entorno de Arduino la información que le llega de los otros nodos, y se tiene otro ordenador el cual se encarga de monitorear la red y capturar los paquetes, en la figura 3.2 se puede observar como está diagramado la red antes descrita.

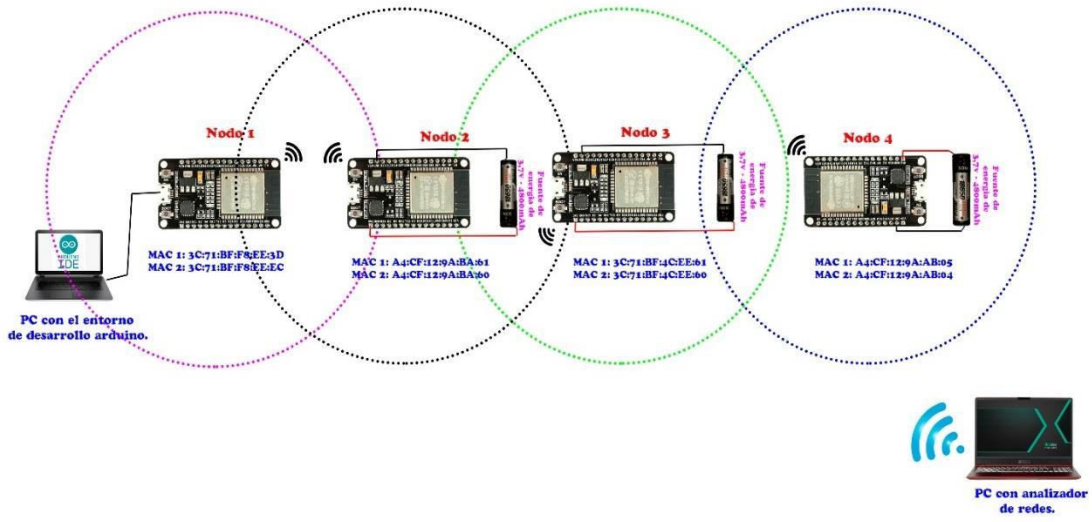


Figura 3.2. Escenario B red en línea recta con dispositivos NodeMCU ESP32.

Escenario C

En este escenario se conectan en una topología tipo estrella en donde el nodo 2 es el punto de convergencia de los otros nodos, al igual que los otros escenarios se tiene un ordenador el que permite observar vía consola los datos que llegan a los distintos nodos que pertenece al redMallaEPN, de la misma forma otro ordenador se encuentra en modo monitor para capturar los paquetes que transcurren por la red. Posteriormente, en la figura 3.3 se puede observar lo antes mencionado.

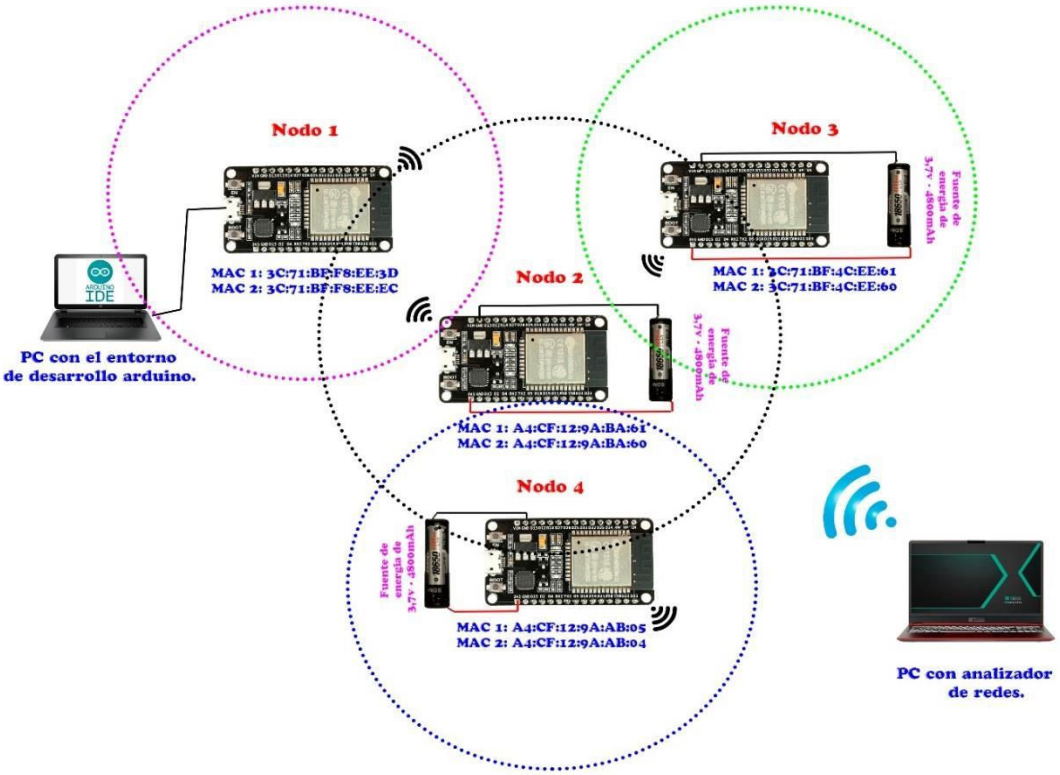


Figura 3.3. Escenario C red tipo estrella con dispositivos NodeMCU ESP32.

3.1. PRUEBAS DE FUNCIONAMIENTO

Se presenta el funcionamiento de la red con la transmisión y recepción de mensajes entre los nodos de la red; así como las diferentes funcionalidades que tiene la red con nodos autónomos y autoconfigurables.

En primera instancia se observa como trabajan los nodos NodeMCU ESP32 en la red, cuya ejecución se da de forma exitosa, esto se puede evidenciar mediante la consola de uno de

los nodos que pertenece a la red y con una trasmisión tipo broadcast, en la siguiente figura 3.4 se muestran los resultados.

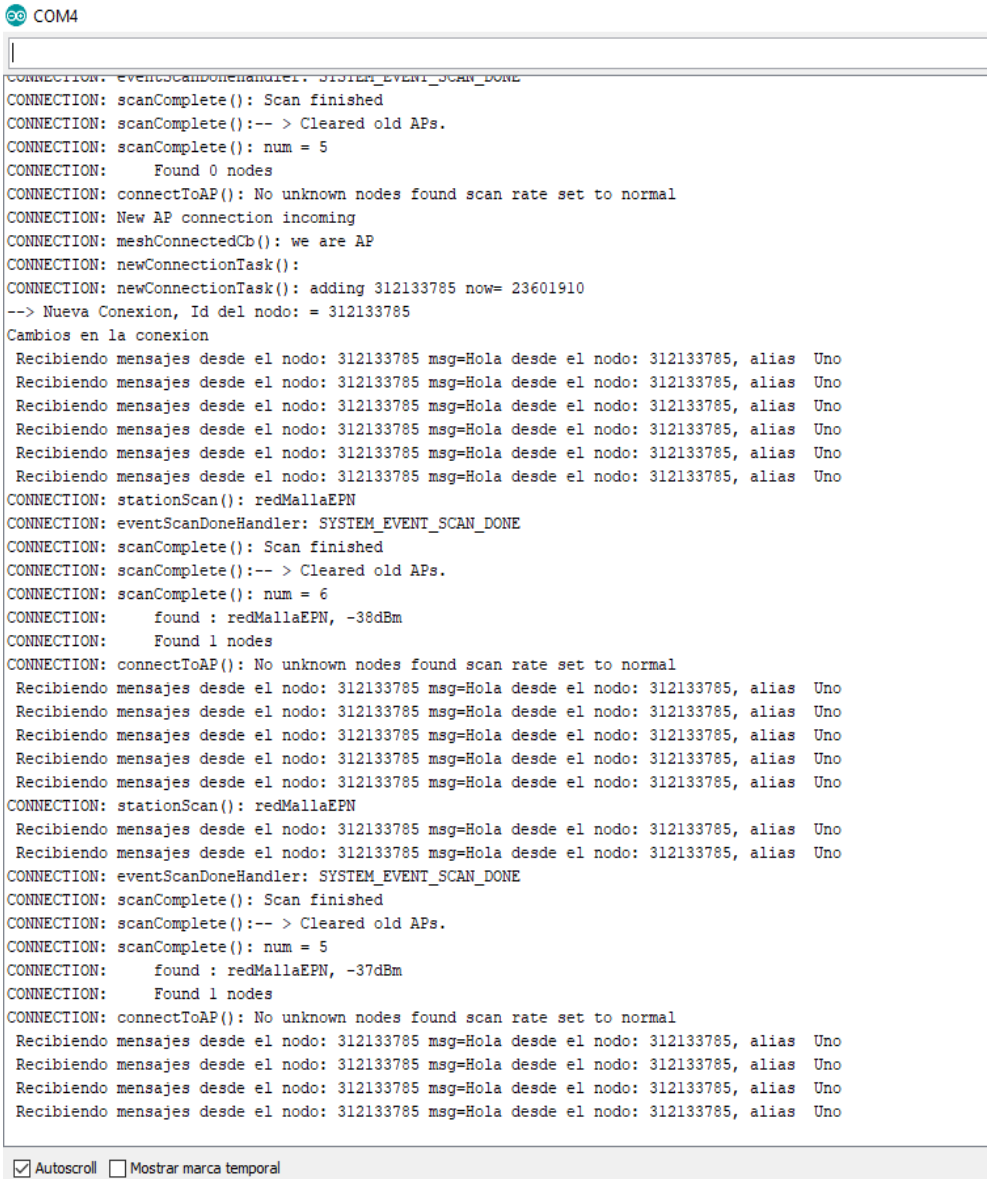


Figura 3.4. Funcionamiento red tipo malla inalámbrica usando la placa NodeMCU ESP32.

Una vez realizado este paso con éxito, se puede observar que la red tipo malla trabaja de forma correcta por lo que se procede a implementar los diferentes escenarios planteados para realizar las pruebas y la recolección de los datos del desempeño de la red.

Antes de iniciar con el proceso de pruebas y recolección de datos, se muestran en la tabla 3.1 las direcciones MAC que poseen cada uno de los nodos que están interactuando en la trasmisión de datos en la red malla.

Es importante mencionar que cada uno de los nodos trabajan bajo el modo softAP, este modo permite trabajar al dispositivo NodeMCU ESP32 (Nodo) de forma de Access Point o cliente al mismo tiempo dependiendo de la distribución de los nodos en la red.

Estas direcciones MAC son de mucha importancia al momento de obtener los datos en los analizadores de software para poder visualizar como se conectan los nodos.

Tabla 3.1. Dirección MAC de los nodos.

Nodos	Access Point	Cliente
Nodo 1	A4:CF:12:9A:C8:99	A4:CF:12:9A:C8:98
Nodo 2	3C:71:BF:4C:EE:61	3C:71:BF:4C:EE:60
Nodo 3	A4:CF:12:9A:BA:61	A4:CF:12:9A:BA:60
Nodo 4	A4:CF:12:9A:AB:05	A4:CF:12:9A:AB:04

Estas direcciones serán de suma importancia al momento de ver como se encuentra distribuido físicamente la topología, mientras que para poder identificar al nodo dentro de la red se usa el chip ID. En la siguiente tabla 3.2. se pueden observar los chips ID de cada uno de los nodos.

Tabla 3.2. Dirección chip ID de los nodos.

Nodos	Chip ID
Nodo 1	312133785
Nodo 2	3209490017
Nodo 3	312130145
Nodo 4	312126213

3.1.1. DETECCIÓN Y ESTABLECIMIENTO DE CONEXIÓN DE LA RED

Uno de los primeros pasos que realizan los scripts cargados en las placas, es la detección de la red buscando los nodos más cercanos para poder conectarse; cada uno de los nodos tienen la capacidad de comportarse como Access Point o cliente ya que se encuentran programados en modo softAP. La detección finaliza al encontrar nodos con los cuales pueden establecer conexión, caso contrario, inicia de nuevo el proceso de detección de la red en busca de dispositivos nuevos para establecer la conexión. Al tener una detección de forma constante permite a los dispositivos (nodos) ser autónomos y autoconfigurables, en la figura 3.5 se puede observar lo antes mencionado, mostrando vía consola de Arduino

el proceso de detección y el número de nodos encontrados para formar parte de la red tipo malla y sus respectivas potencias que poseen cada uno de los nodos.

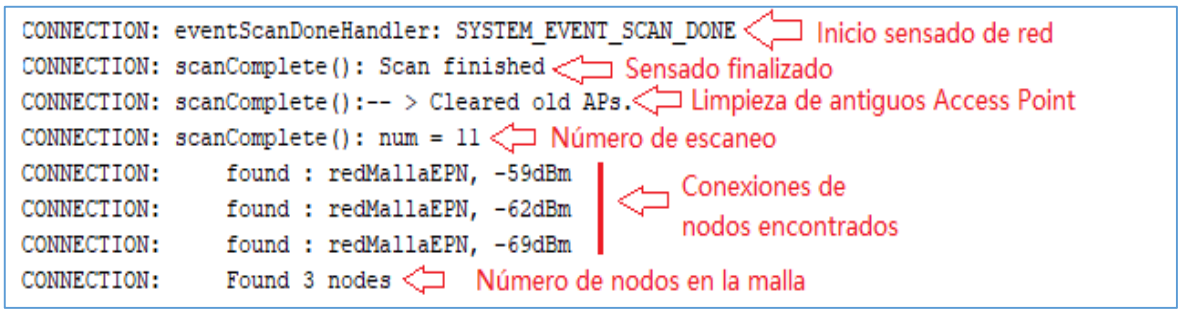


Figura 3.5. Detección y conexión de los nodos en la red.

3.1.1 ENVÍO Y RECEPCIÓN DE DATOS

Una vez establecida la conexión con los nodos de la red, el nodo principal en este caso el Nodo 1 de forma automática empezará a enviar la información a los nodos que están en la red, cabe mencionar que el nodo 1 es el nodo principal el cual está configurado para enviar datos en una transmisión con tráfico broadcast, mientras que los restantes nodos se encuentran configurados como receptores de los mensajes.

En la figura 3.6 se puede observar al nodo principal encargado de enviar datos, mientras que en la figura 3.7 se puede observar la recepción de los mensajes enviados desde el nodo principal; para una mejor identificación dentro de la red se le coloca un nombre llamado uno al nodo principal en la transmisión tipo broadcast.

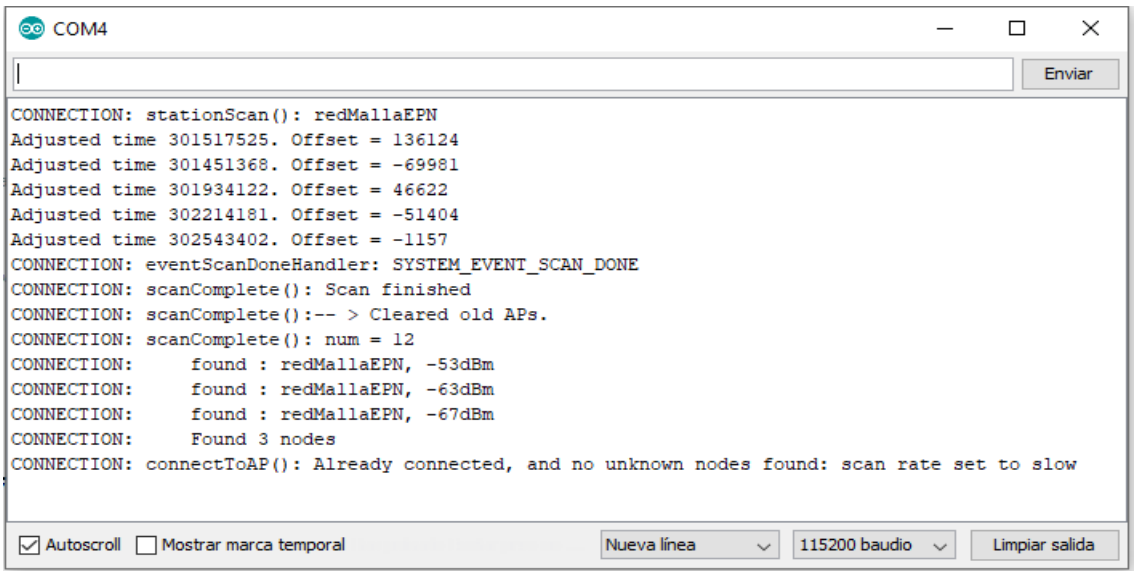


Figura 3.6. Envío de datos tipo broadcast desde el nodo principal (Nodo 1).

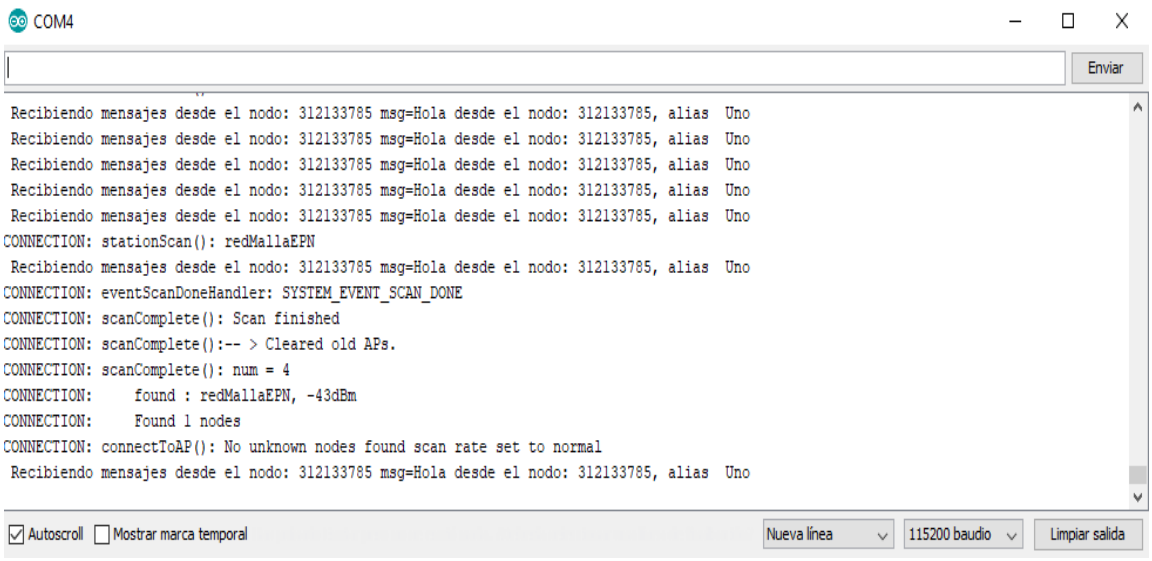


Figura 3.7. Recepción de datos tipo broadcast en los nodos de la red.

3.2. PRUEBAS DE AUTOCONFIGURACIÓN

Se procede con el análisis de la autoconfiguración de los nodos en cada uno de los escenarios, donde se muestra mediante diagrama de secuencias el intercambio de tramas de gestión al establecer la conexión en un ambiente inalámbrico.

3.2.1. ESCENARIO DE PRUEBA A

Para esta prueba, se procede a colocar los 4 nodos NodeMCU ESP32 interconectados en su totalidad es decir en una topología tipo malla, mediante una transmisión tipo broadcast se envían paquetes de información a cada uno de los nodos de la red, siendo el nodo 1 el que comienza con la trasmisión y el nodo 2, 3 y 4 los que reciben la información de forma inalámbrica, cada uno de los nodos se encuentran distanciados uno del otro.

Cada uno de los nodos al trabajar en un medio inalámbrico sigue un proceso de conexión entre cada uno de ellos. Por lo que, para esta prueba nos basamos en las tramas de gestión del estándar inalámbrico 802.11 que se transmiten al momento de darse la conexión entre los dispositivos.

En la figura 3.8 se indica el proceso de intercambio de tramas que realizan los dispositivos al momento de la autenticación de los nodos. Los nodos que actúan como Access Point envían de forma continua una trama de baliza que son captados por los nodos clientes que se encuentran a su alrededor anunciando sus SSID y las velocidades de datos, los nodos cercanos trasmiten tramas de solicitud de onda anunciando las velocidades de datos

admitidas por el nodo, los nodos que actúen como puntos de acceso envían respuesta a la sonda preparándose así los nodos clientes a enviar una solicitud de autenticación hacia el nodo actúa como Access Point.

Los nodos que están como punto de acceso envían una respuesta de autenticación invitando a los nodos clientes a autenticarse en el SSID, tras una autenticación exitosa, los nodos clientes envían tramas de solicitud de asociación hacia los nodos que son puntos de acceso, este proceso termina con la respuesta de asociación de forma exitosa.

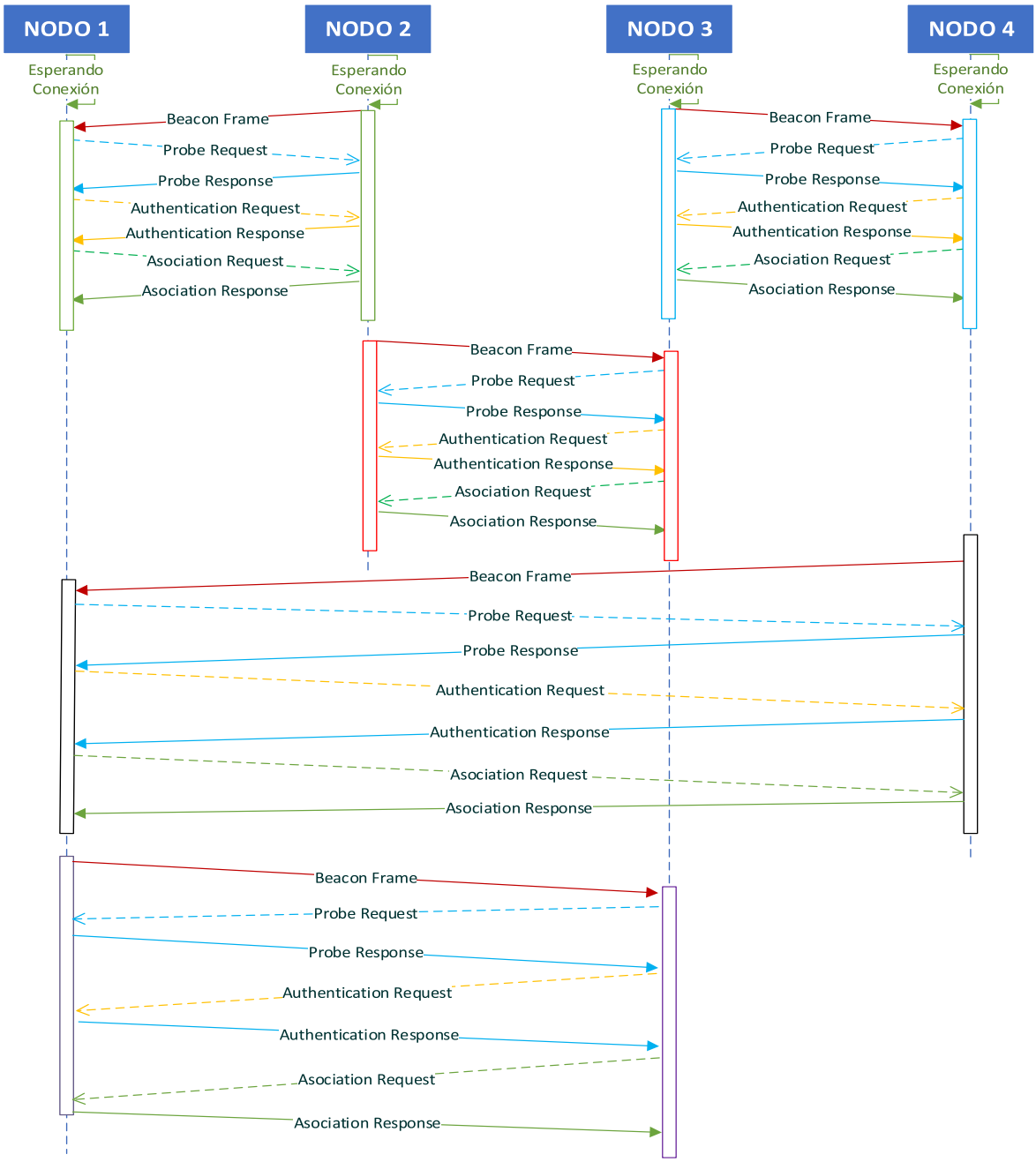


Figura 3.8. Diagrama de secuencia conexión entre nodos escenario A

Para demostrar la autoconfiguración y la autonomía de los nodos en la red, se procede a desconectar uno de los nodos; en este caso se desconecta el nodo 3 durante un intervalo de tiempo y se observa mediante el analizador de redes Wireshark los diferentes paquetes que se emiten entre los nodos que pertenecen a la red y la autonomía que poseen para poder autoconfigurarse por sí solos sin la ayuda de ninguna entidad exterior.

Cabe mencionar que al momento de desactivar el nodo este envía mensajes de disociación, en el analizador de Wireshark se observa como un paquete llamado Disassociate el cual es enviado a los nodos vecinos que se encuentran conectados en ese momento. En la figura 3.9 se puede observar mediante diagramas de secuencias como se da este proceso de disociación del nodo 3 con el envío de paquetes a los nodos que trabajan como puntos de acceso.

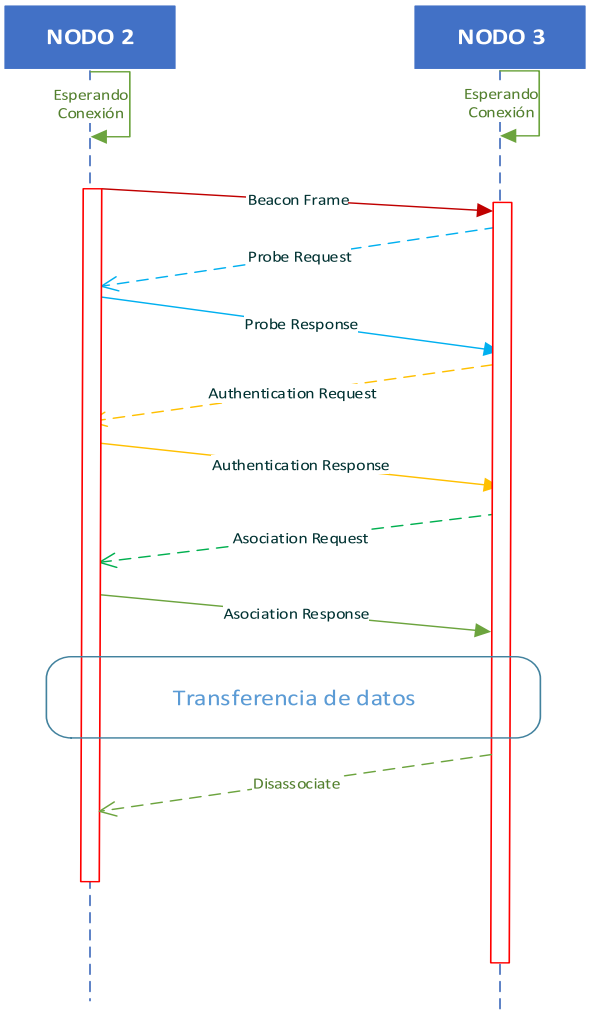


Figura 3.9. Diagrama de secuencia disociación del nodo 3 a la red.

Una vez disociado el nodo 3 de la red, los nodos vecinos comienzan a enviar Probe Request hacia los puntos de acceso (nodos) más cercanos para comenzar de nuevo con el proceso de conexión y formar de nuevo la red tipo malla con los nodos que se encuentran activos en la red, cambiando la topología de la red y autoconfigurándose de manera autónoma cada uno de los nodos.

A continuación, se puede observar en la figura 3.10 mediante un diagrama de secuencia la nueva conexión que se estableció ante la ausencia de unos de los nodos de la red, obteniendo así nodos que son independientes y que se autoconfiguran sin la necesidad de tener un administrador de red que realice una nueva reestructuración de la malla.

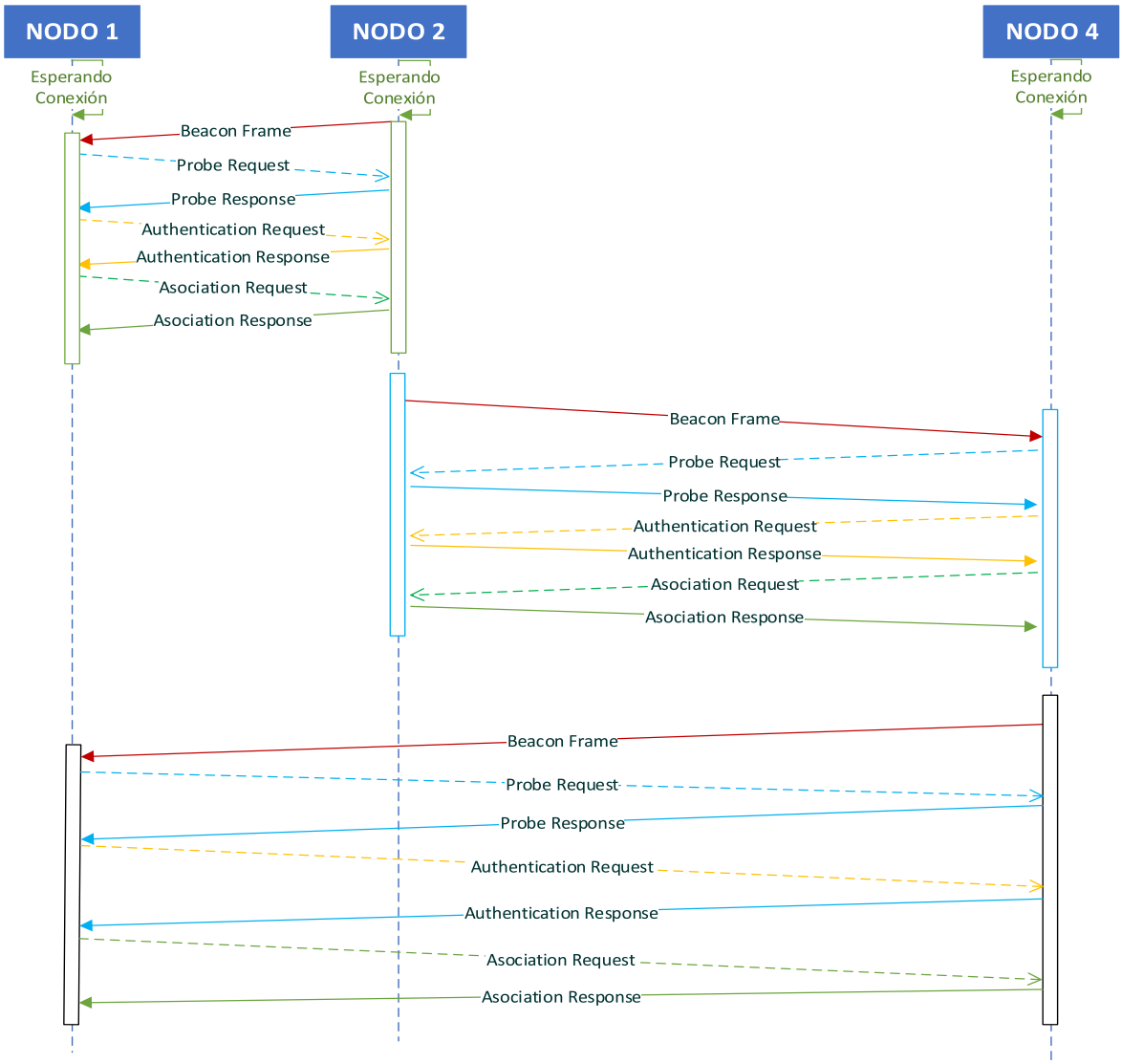


Figura 3.10. Diagrama de secuencia de la autoconfiguración de la red malla inalámbrica.

3.2.2. ESCENARIO DE PRUEBA B

Para la realizar esta prueba y obtener este tipo de topología se tiene que considerar distancias mínimas de 15 metros entre cada uno de los dispositivos NodeMCU ESP32 para que puedan conectarse en pares y en línea recta, siendo así, el nodo 2 tiene a su alcance al nodo 1 y al nodo 3, mientras que el nodo 3 tiene a su alcance al nodo 2 y al nodo 4, para llegar al nodo 4 desde el nodo 1 se debe hacer mediante el uso de los nodos intermedios 2 y 3. En la siguiente figura 3.11 se observa la conexión del escenario antes mencionado.

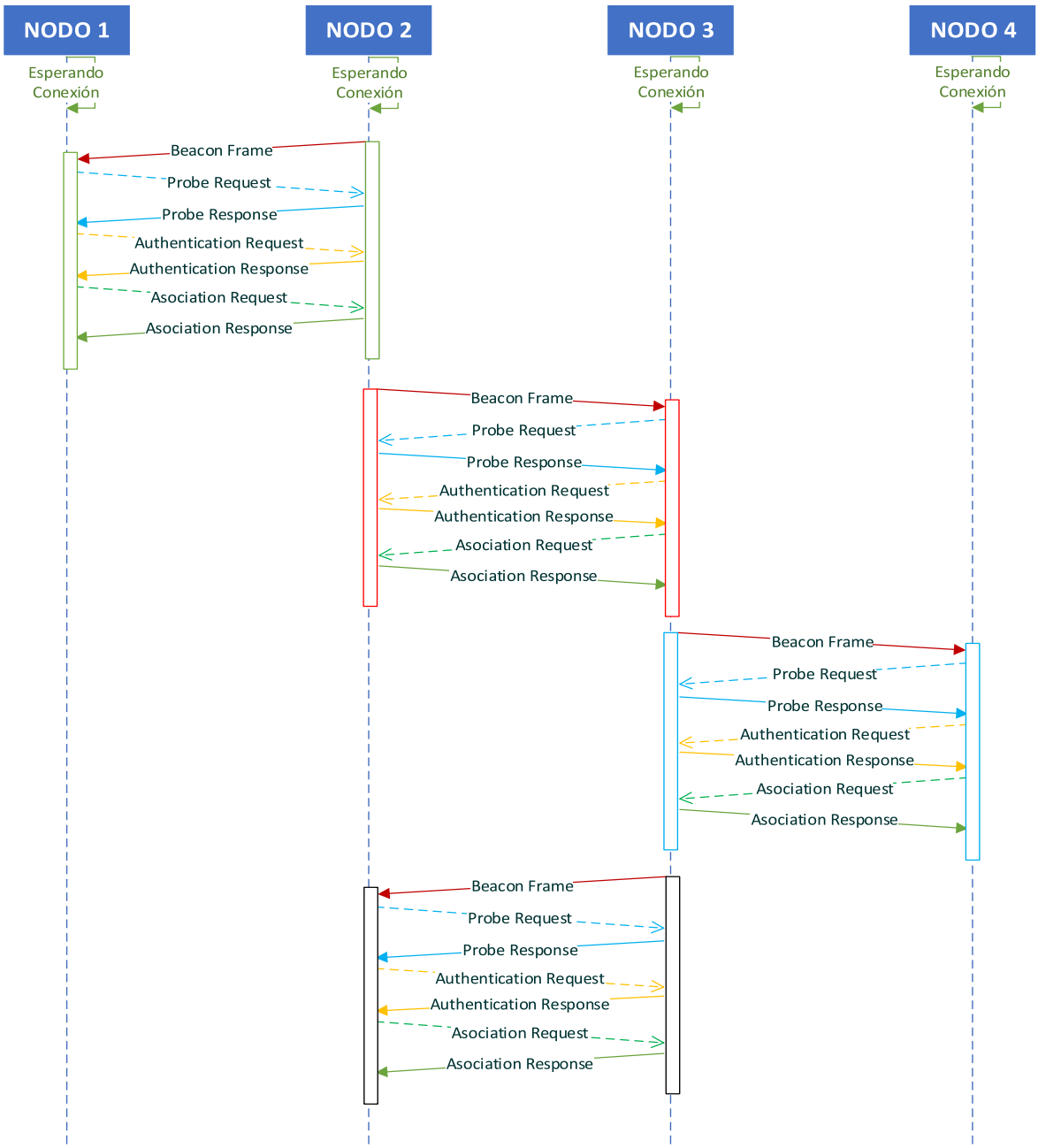


Figura 3.11. Diagrama de secuencia conexión entre nodos escenario B.

Dentro de la operatividad de la red se desconecta el nodo 3 que es uno de los nodos intermedios quedando la red sin funcionamiento ya que no se puede darse la autoconfiguración debido a que no se llegan a ver el uno con el otro, porque los nodos activos no logran intercambiar paquetes de conexión entre ellos, negando así el establecer la conexión entre los nodos activos.

3.2.3. ESCENARIO DE PRUEBA C

En este escenario al tener un nodo que converge con el resto de los nodos en una topología tipo estrella y en un estado de operatividad de la red se procede a realizar la prueba. En la figura 3.12 se muestra mediante un diagrama de secuencia la conexión de los nodos.

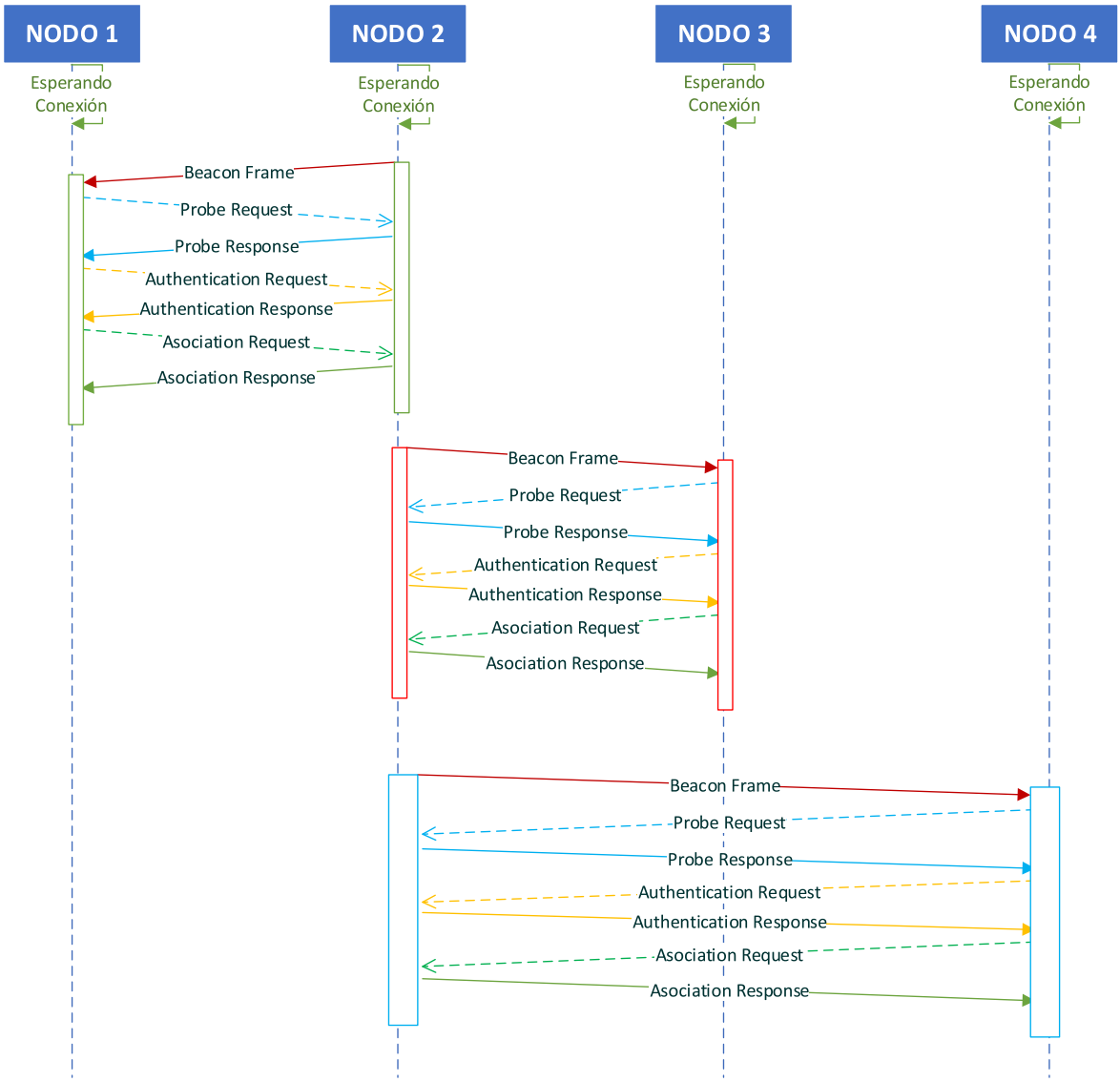


Figura 3.12. Diagrama de secuencia conexión entre nodos escenario C.

Se procede a desconectar el nodo 3 como en los anteriores casos, se observa que no sufre ningún cambio debido a que es un nodo que se encuentra actuando como nodo cliente y está conectado al nodo 2 que actúa como nodo principal, por lo que la autoconfiguración no se efectúa, porque al desactivar el nodo 3 de la red no existe un cambio significativo dentro de la topología por lo que no afecta a la red.

Uno de los aspectos importantes que hay que considerar en este tipo de topología es que al depender de un nodo principal como en este caso del nodo 2 donde convergen todos los nodos y dejar fuera de funcionamiento a este nodo, la red sufre una caída donde no se podrá autoconfigurar, ya que los nodos clientes se encuentran fuera del alcance uno del otro, lo que imposibilita el intercambio de los paquetes de conexión lo que implica que no se puede establecer la conexión.

3.3. PRUEBAS DE CONECTIVIDAD

En esta sección se realizan las pruebas de conectividad que se dan entre los nodos que pertenecen a la red. Primero se realiza las pruebas con tráfico broadcast para luego proceder con las pruebas unicast cada uno de estas pruebas se realizan en los tres escenarios planteados, donde se muestra la recepción del mensaje en uno de los nodos, probando así el funcionamiento de la conectividad de la red.

3.3.1. TRÁFICO BROADCAST

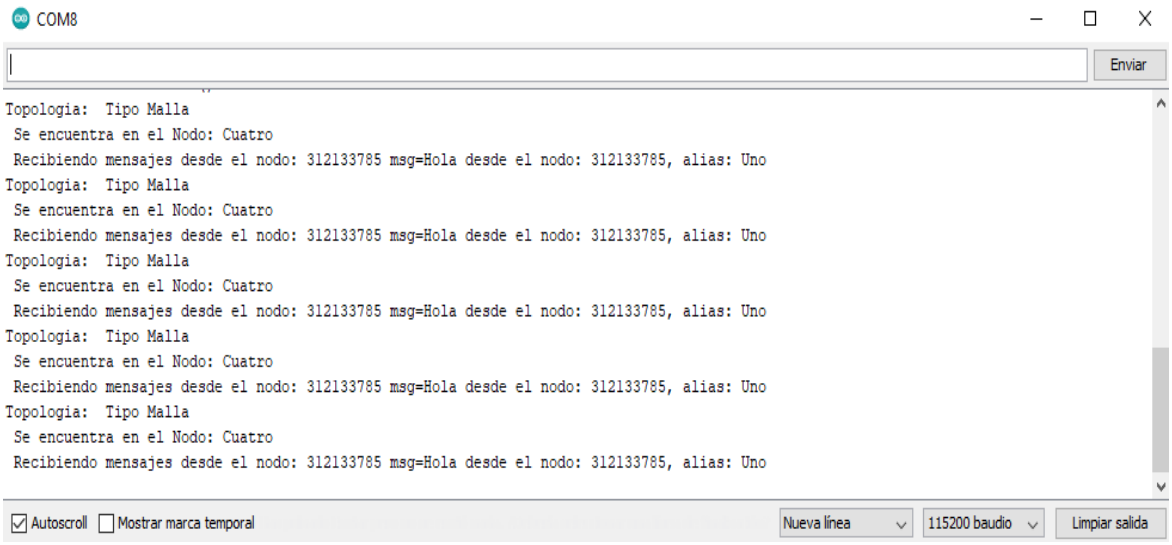
A continuación, se realizan las pruebas de conectividad para una transmisión tipo broadcast para poder demostrar la conectividad que existe entre los nodos de la red.

3.3.1.1. ESCENARIO DE PRUEBA A

Para esta prueba se procede a formar una topología tipo malla, donde cada uno de los nodos se encuentran conectados entre sí, y se procede a enviar desde el nodo 1 un mensaje tipo broadcast a cada uno de los nodos que están dentro de la red. El mensaje que se envía desde el nodo 1, es de un saludo de "*Hola desde el nodo: se ubica el identificador del nodo y el nombre del nodo transmisor asignado.*"

Se toma un nodo de forma aleatoria de la red para observar el mensaje enviado, en este caso el receptor elegido es el nodo 4, donde se puede visualizar el mensaje transmitido, todo lo antes mencionado se puede ver en la figura 3.13.

En la figura se puede observar como primera instancia el tipo de topología donde se encuentra realizando la prueba, el nombre del nodo que actúa como receptor, así como el mensaje que llega desde el nodo trasmisor con su identificador de red.



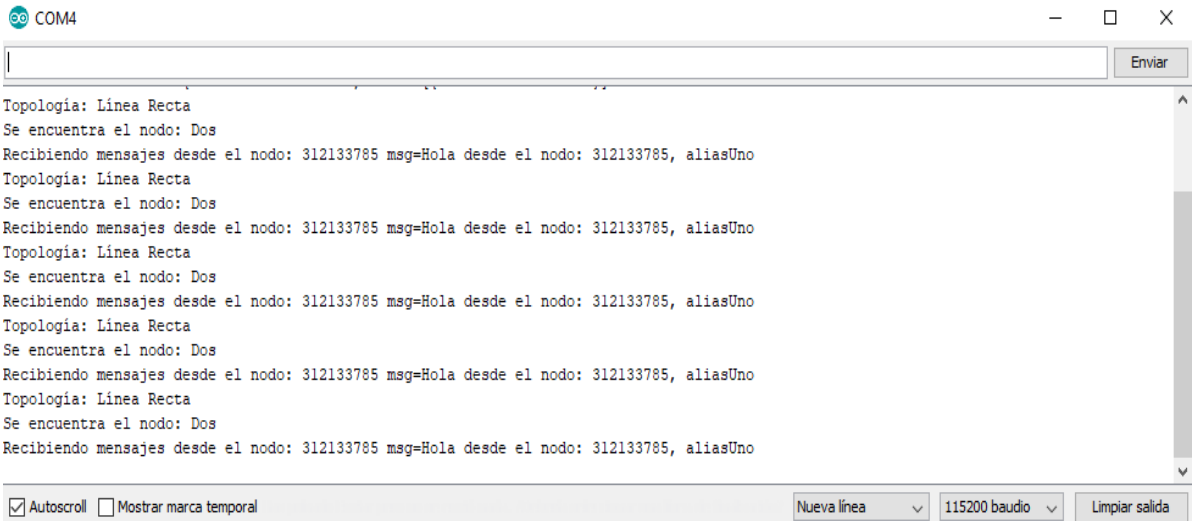
```
Topologia:  Tipo Malla
 Se encuentra en el Nodo: Cuatro
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Malla
 Se encuentra en el Nodo: Cuatro
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Malla
 Se encuentra en el Nodo: Cuatro
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Malla
 Se encuentra en el Nodo: Cuatro
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Malla
 Se encuentra en el Nodo: Cuatro
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
```

Figura 3.13. Recepción del mensaje de tráfico broadcast en el nodo 4.

3.3.1.2. ESCENARIO DE PRUEBA B

Se procede a colocar los nodos en línea recta y se procede a enviar un mensaje de tráfico tipo broadcast sobre la red, la cual lleva la misma estructura que en el escenario A con el identificador del nodo transmisor, donde el nodo transmisor es el nodo 1 y el nodo receptor donde se conecta el ordenador para visualizar los datos es el nodo 2.

A continuación, se puede observar en la figura 3.14 la recepción del mensaje indicando el tipo de topología en donde se está haciendo la prueba, el nombre del nodo donde llega el mensaje y el mensaje transmitido desde el nodo 1, todo eso se puede visualizar vía consola en el entorno de desarrollo de Arduino.



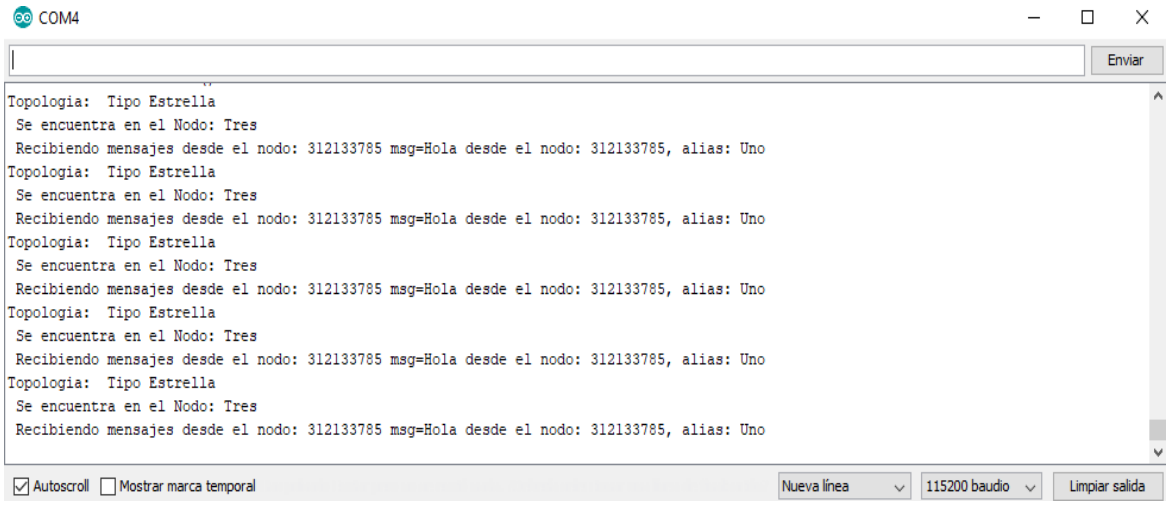
```
Topología: Línea Recta
Se encuentra el nodo: Dos
Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, aliasUno
Topología: Línea Recta
Se encuentra el nodo: Dos
Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, aliasUno
Topología: Línea Recta
Se encuentra el nodo: Dos
Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, aliasUno
Topología: Línea Recta
Se encuentra el nodo: Dos
Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, aliasUno
Topología: Línea Recta
Se encuentra el nodo: Dos
Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, aliasUno
```

Figura 3.14. Recepción del mensaje de tráfico broadcast en el nodo 2.

3.3.1.3. ESCENARIO DE PRUEBA C

Para esta prueba se procede a colocar los nodos en topología tipo estrella, en donde el nodo 1 actúa como el nodo intermedio y el resto de los nodos se encuentran localizados en las puntas, de la misma forma que los anteriores escenarios se envía el mensaje de tráfico tipo broadcast con su identificador y su nombre.

En este caso el nodo que transmite el mensaje tipo broadcast es el nodo 1 que se encuentra localizado en medio de la red. Para probar la conectividad de los nodos se escoge de forma aleatoria al nodo 3 para observar el mensaje que recepta el nodo. En la figura 3.15 se puede observar el mensaje que llega y el nombre del nodo donde se recibe el mensaje, así como en el tipo de topología con la cual se está realizando la prueba de conectividad.



```
Topologia:  Tipo Estrella
 Se encuentra en el Nodo: Tres
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Estrella
 Se encuentra en el Nodo: Tres
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Estrella
 Se encuentra en el Nodo: Tres
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Estrella
 Se encuentra en el Nodo: Tres
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
Topologia:  Tipo Estrella
 Se encuentra en el Nodo: Tres
 Recibiendo mensajes desde el nodo: 312133785 msg=Hola desde el nodo: 312133785, alias: Uno
```

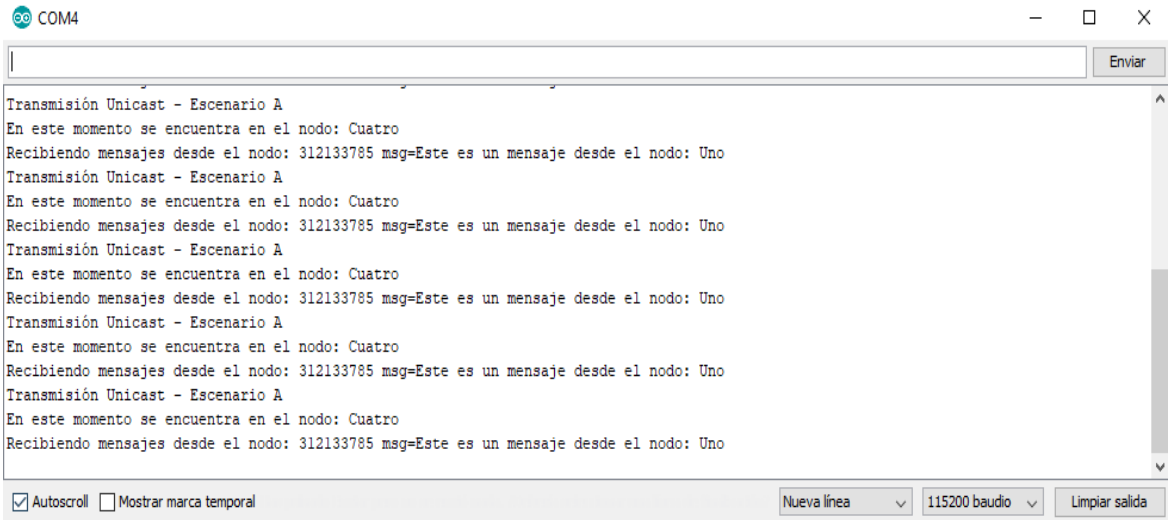
Figura 3.15. Recepción del mensaje de tráfico broadcast en el nodo 3.

3.3.2. TRÁFICO UNICAST

A continuación, se procede a presentar las pruebas de conectividad de los nodos enviando tráfico unicast sobre la red.

3.3.2.1. ESCENARIO DE PRUEBA A

A partir de que los nodos se encuentran distribuidos en una topología tipo malla y conectados, se procede a enviar tráfico unicast desde el nodo 1 hasta el nodo 4, para probar la conectividad que existe en la red. A continuación, se observa en la figura 3.16 el proceso exitoso de enviar el mensaje a través del nodo 1 al nodo 4, mostrando a través del monitor serie del IDE Arduino los mensajes receptados en el nodo 4.

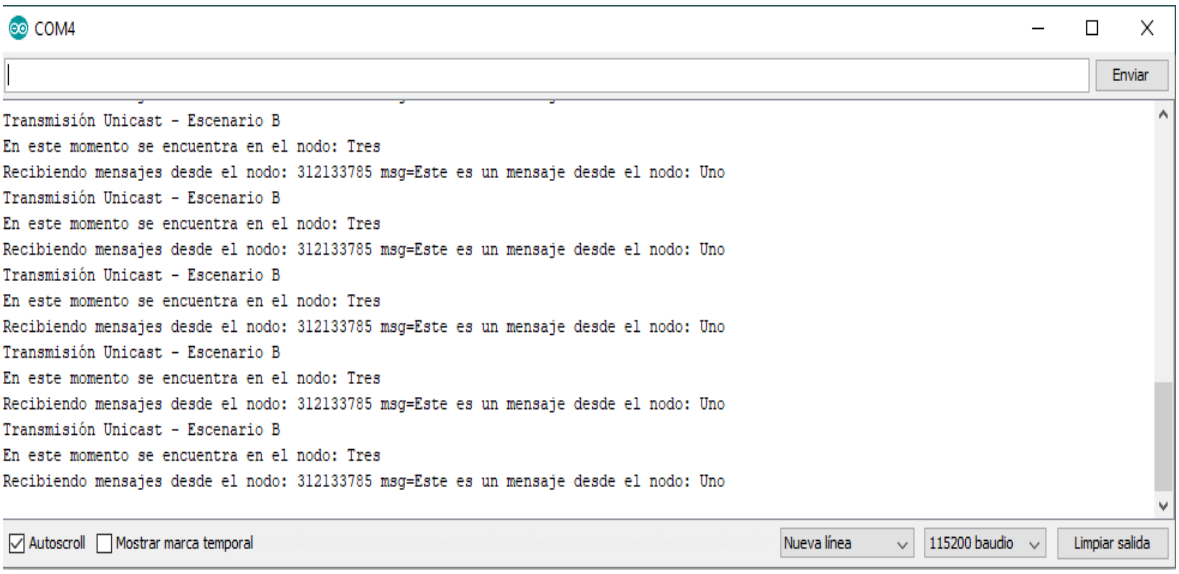


```
Transmisión Unicast - Escenario A
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario A
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario A
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario A
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario A
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
```

Figura 3.16. Recepción del mensaje de tráfico unicast en el nodo 4.

3.3.2.2. ESCENARIO DE PRUEBA B

En este escenario, el nodo 1 es el transmisor mientras el nodo 3 es el receptor, para comenzar con la prueba se verifica que se encuentren conectados según lo mencionado para este escenario, se procede a enviar el mensaje desde el nodo 1, el cual va pasar obligatoriamente por el nodo 2 ya que es el nodo intermedio entre los dos nodos para luego retransmitirlo hacia el nodo 3 que es el nodo receptor, en la figura 3.17 se observa el mensaje en el nodo receptor y quien es el nodo que lo envía, todo esto se refleja en el monitor serie de Arduino.

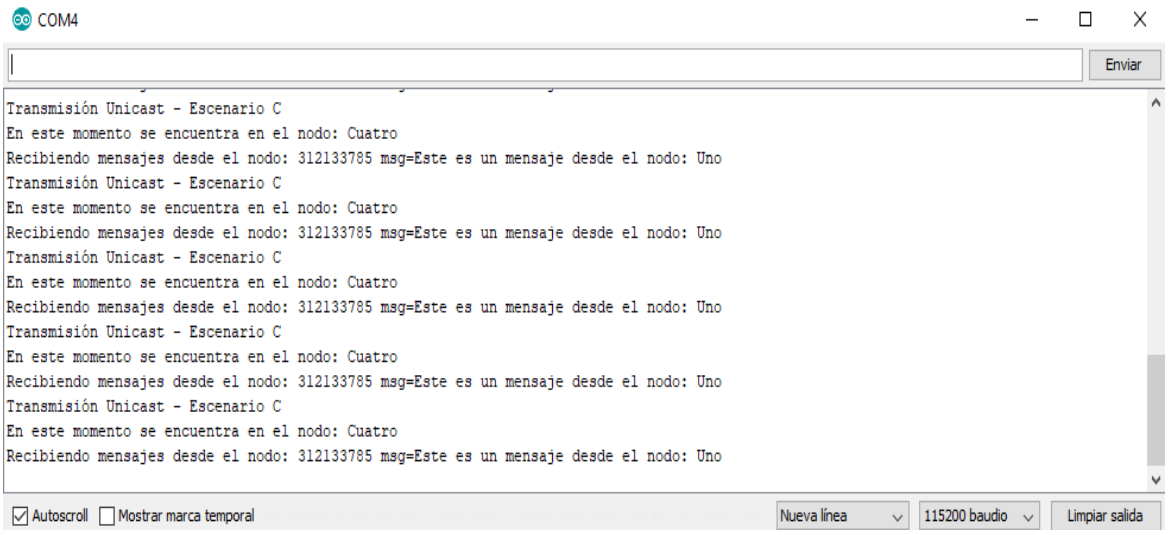


```
Transmisión Unicast - Escenario B
En este momento se encuentra en el nodo: Tres
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario B
En este momento se encuentra en el nodo: Tres
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario B
En este momento se encuentra en el nodo: Tres
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario B
En este momento se encuentra en el nodo: Tres
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario B
En este momento se encuentra en el nodo: Tres
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
```

Figura 3.17. Recepción del mensaje de tráfico unicast en el nodo 3

3.3.2.3. ESCENARIO DE PRUEBA C

Para este escenario, la prueba se realiza enviando un mensaje desde el nodo 1 que es el transmisor hacia el nodo 4 que es el receptor, cabe mencionar que estos dos nodos se encuentran ubicados en las puntas de la topología; el mensaje tiene que pasar de forma obligada por el nodo 2 para luego ser retransmitido hacia el nodo receptor, a continuación, se puede observar en la figura 3.18 el mensaje que llega de forma exitosa al nodo 4 lo que asegura la conectividad entre los nodos de la red en este escenario.



```
Transmisión Unicast - Escenario C
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario C
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario C
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario C
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
Transmisión Unicast - Escenario C
En este momento se encuentra en el nodo: Cuatro
Recibiendo mensajes desde el nodo: 312133785 msg=Este es un mensaje desde el nodo: Uno
```

Figura 3.18. Recepción del mensaje de tráfico unicast en el nodo 4.

3.4. PRUEBAS DE RENDIMIENTO

Esta sección se enfoca en mostrar las métricas de rendimiento que se dan entre los nodos y la recolección de los datos que se obtienen en los distintos escenarios. Estas pruebas se enfocan en el envío de tráfico unicast, lo cual permite tener valores más exactos para el análisis.

Las métricas que se evalúan dentro de estas pruebas sobre la red son las siguientes:

- Retardos
- Pérdida de paquetes
- Throughput
- Alcances máximos

Con el objetivo de analizar cada una de las métricas de la red antes mencionadas, se realiza la prueba con 4 nodos NodeMCU ESP32 conectados e intercambiando mensajes, las pruebas se realizan en un ambiente al aire libre la que consta con un área 19.819,46m^2 y se encuentra ubicada en el cantón de Mejía sector la Moya.

En la tabla 3.3 se indica la estructura del mensaje que se trasmite entre los nodos para poder detectar cada uno de estos parámetros de mejor manera.

El nodo 1 que pertenece a la red envía un mensaje con el nombre que lo identifica, así como un número que indica la cantidad de mensajes que se envía desde el nodo transmisor hacia el receptor, de la misma manera se envía el tiempo que se demora el mensaje en salir desde el nodo transmisor hacía el receptor.

Al recibir el mensaje un nodo de la red, le añade su nombre que lo identifica en la red y el tiempo que recepta el mensaje. El nodo que recibe el mensaje se encuentra conectado al ordenador lo que permite acceder a los mensajes que circulan dentro de la red para después analizarlo.

Para obtener cada uno de los datos antes mencionados en la estructura del mensaje, se usan funciones que nos brinda la librería *painlessMesh,* esta función es *getNodeTime* que permite determinar la hora de recepción como la de trasmisión, la función antes mencionada es de mucha utilidad para determinar los retardos que se presentan en la

circulación de mensajes entre nodos, algo de destacar de esta función es que la propia red sincroniza los relojes de cada uno de los dispositivos.

Con respecto al número de mensajes que se envían por la red se hace uso de un contador que aumenta a medida que se envía el mensaje.

Tabla 3.3. Formato del mensaje final que se intercambian entre los nodos.

Nombre nodo receptor	Tiempo de recepción	
Nombre nodo transmisor	Número de mensaje	Tiempo de transmisión

Una vez explicado el proceso del mensaje que se transmite por la red, se procede a implementar cada uno de los escenarios con distintos cambios dentro de los parámetros que permite medir el rendimiento que tiene la red.

Con los resultados obtenidos en la red se procede a utilizar las siguientes fórmulas que se muestran a continuación para calcular las métricas mencionados anteriormente.

$$Retardo\ [s] = tiempo_{Rx} - \ tiempo_{Tx}$$

Ecuación 3.1. Retardo del paquete

$$Pérdida\ de\ paquetes\ [paquetes] = DatosTotalesTransmitidos - DatosTotalesRecibidos$$

Ecuación 3.2. Pérdida del paquete

$$Throughput = \frac{Longitud\ del\ paquete}{Retardo\ del\ paquete}$$

Ecuación 3.3. Throughput de la red

3.4.1. ESCENARIO DE PRUEBA A

En esta prueba se colocan los cuatro nodos en un ambiente outdoor separados distancias de 10, 15 y 30 metros en cada una de las variaciones realizadas se envían por la red 1, 10 y 50 paquetes por cada segundo con un total de cien paquetes analizados.

A continuación, se puede observar la distribución de los nodos en la figura 3.19.

Figura 3.19. Distribución física tipo malla donde los nodos se encuentran separados entre sí una distancia de 30 metros, en el en el sector de la Moya cantón Mejía, escenario A.

Colocados los nodos en topología tipo malla y en funcionamiento, se obtienen posteriormente los siguientes datos que son analizados en tablas de hojas de cálculo en Excel. En la tabla 3.4 se muestra un ejemplo de los datos obtenidos al realizar las pruebas que se encuentran en las tablas de Excel.

Tabla 3.4. Formato del mensaje final que se intercambian entre los nodos.

Nombre Nodo Receptor	Tiempo de Recepción (uS)	Nombre Nodo Transmisor	N° mensaje del transmisor	Tiempo de Transmisión (uS)
TRES	484642764	UNO	5	484635527
TRES	485639604	UNO	6	485635488
TRES	486636498	UNO	7	486629807
TRES	487636850	UNO	8	487629846
TRES	488636486	UNO	9	488629812
TRES	489639602	UNO	10	489629847
TRES	490644204	UNO	11	4906298
TRES	491636943	UNO	12	491629819

Una vez recolectados los datos correspondientes, se procede a analizar los parámetros de rendimiento, cabe mencionar que la totalidad de las tablas se encuentran en el Anexo C.

3.4.1.1. ANÁLISIS DE LA MÉTRICA DE RETARDOS

En la siguiente figura 3.20 se muestran los tiempos de retardos promedios que se obtienen al colocar en funcionamiento a la red.

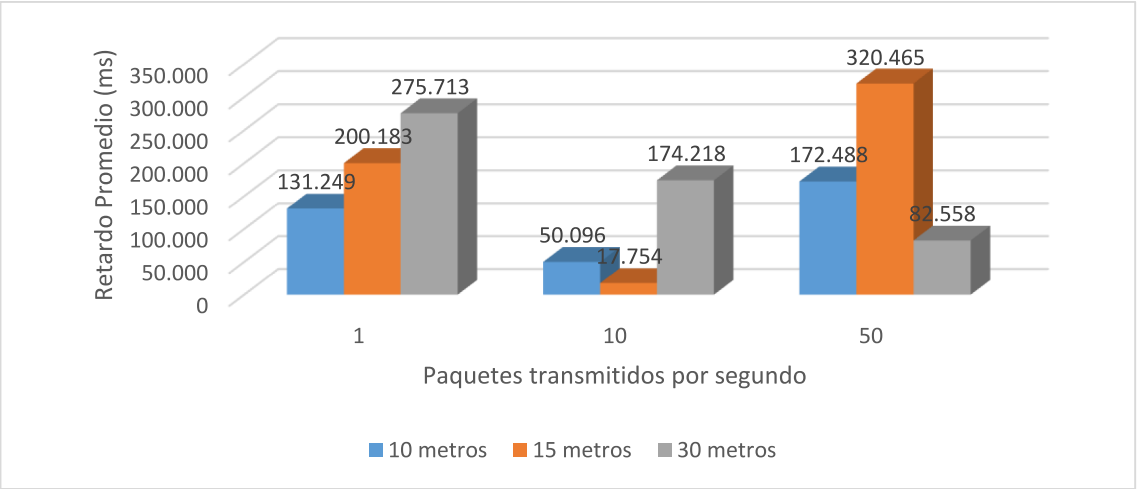


Figura 3.20. Retardo promedio vs paquetes transmitidos por segundo a diferentes distancias escenario A.

Los resultados que se obtienen de esta métrica, son muy favorables en lo que concierne a los retardos, como se puede observar en la figura 3.20 en donde se muestran los retardos promedios que se obtienen con respecto a la variación de paquetes enviados por la red, así como la distancia entre ellos; a continuación, se explica cada uno de los datos más importantes obtenidos en la gráfica.

- Para medir los retardos promedios en la transmisión de paquetes entre los nodos, se usan los relojes internos de cada uno de los dispositivos, lo que implica que pueden existir errores en la medición de estos tiempos, cabe mencionar que, para evaluación de esta métrica el uso de los relojes fue la única forma disponible en ese momento.

- Se puede observar que el comportamiento cuando se envía 10 paquetes por segundo es el de menor retardo con respecto a las diferentes pruebas, esto se debe a que en el desarrollo de la prueba los nodos se autoconfiguraron lo que implicó que se conecten de forma directa el nodo 1 y el nodo 3.

3.4.1.2. ANÁLISIS DE LA MÉTRICA DE PÉRDIDA DE PAQUETES

En el siguiente apartado se presentan los datos que se obtienen a partir de las pruebas realizadas con sus diferentes cambios, en la figura 3.21 se puede observar cada uno de los datos obtenidos referente a la pérdida de paquetes, como el porcentaje de los mismos.

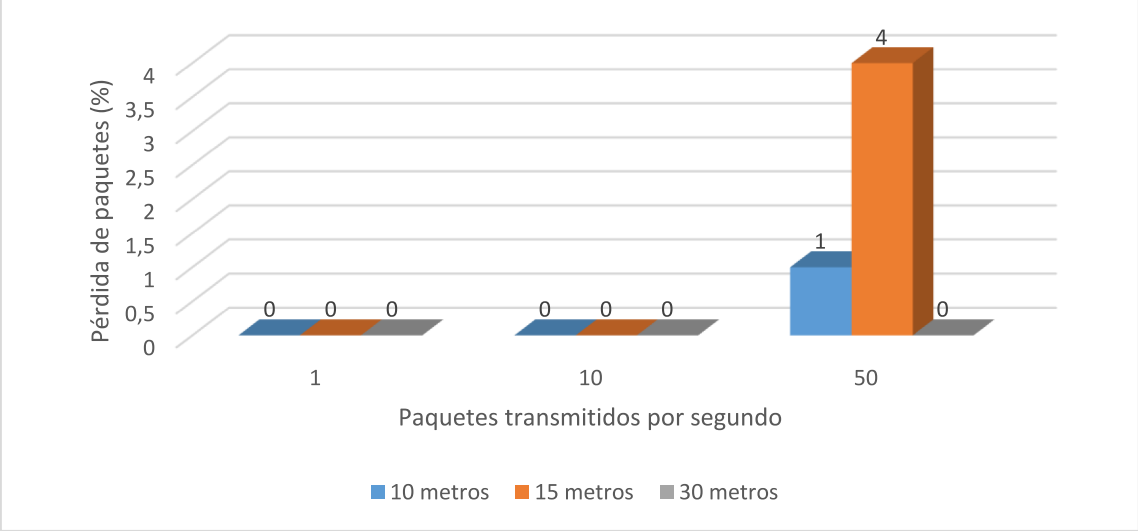


Figura 3.21. Tasa de entrega de paquetes vs los paquetes transmitidos por segundo a distintas distancias enviados por segundo en el escenario A.

Los resultados obtenidos se representan en la figura 3.21 de forma estadística, en donde se detallan los aspectos más destacados que se dan en este tipo de prueba.

- Se puede observar en la gráfica que en cortas distancias no se presenta la pérdida de paquetes.

- En la figura se puede observar, que se hacen presentes las pérdidas de paquetes cuando se tiene distancias largas y sobrecargándole al sistema en este caso cincuenta paquetes por segundo.

- En la figura se puede visualizar que la máxima pérdida de paquetes se da en la distancia de 15 metros con un porcentaje del 4%, debido a que las retrasmisiones funcionaron de la mejor manera llevando a todos los paquetes a su nodo receptor.

3.4.1.3. ANÁLISIS DE LA MÉTRICA DE THROUGHPUT

En la figura 3.22 se muestran los datos obtenidos en las pruebas realizadas con respecto a la métrica de throughput, donde se presentan las distancias donde se realizaron las pruebas, como los paquetes transmitidos por la red.

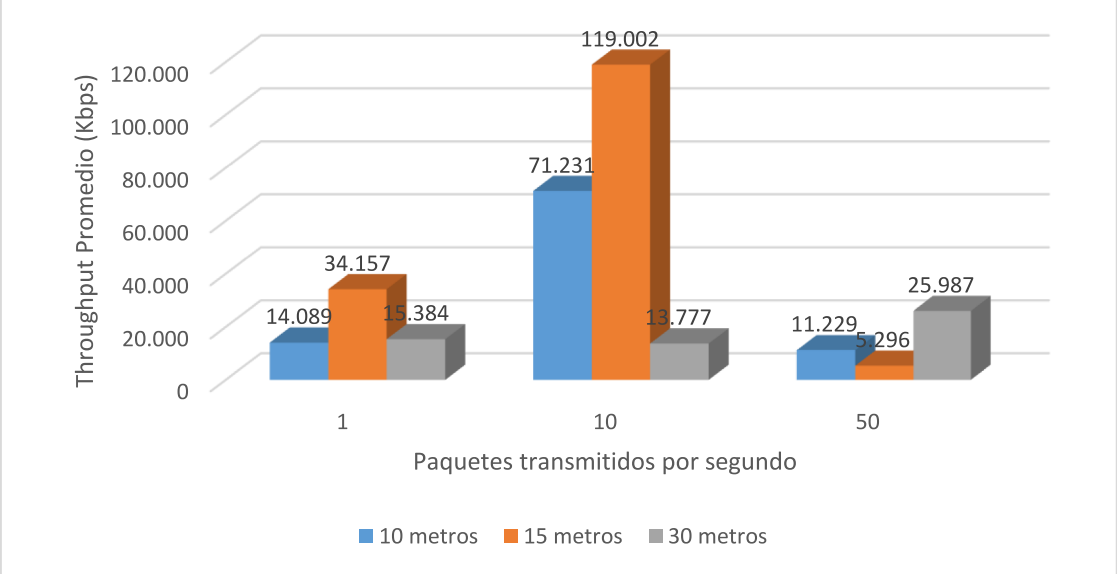


Figura 3.22. Throughput vs paquetes transmitidos por segundo a distintas distancias entre nodos escenario A.

Cabe mencionar que para la evaluación de esta métrica, los datos obtenidos están directamente relacionados con la cantidad de paquetes perdidos y el retardo, por lo tanto, los resultados que se muestran son consecuencia de las anteriores métricas analizadas.

A partir de esta aclaración se puede observar en la figura 3.22 lo siguiente:

- A la distancia de 15 metros y enviando diez paquetes por segundo por la red se obtiene el rendimiento máximo.

- Al sobrecargar la red con un tráfico extremadamente grande, en este caso 50 paquetes por segundo se tiene rendimientos de la red bajos.

- Se puede visualizar que los mejores rendimientos de la red se obtienen a los 15 metros enviando 1 y 10 paquetes por segundo, al momento de enviar 50 paquetes se tiene un descenso del rendimiento a la misma distancia.

3.4.1.4. ANÁLISIS DE LA MÉTRICA DE DISTANCIAS MÁXIMAS

Con los cuatro nodos en funcionamiento, se comienza a realizar la prueba a una distancia de 30 metros ya probadas anteriormente que existe intercambio de paquetes, para posterior cambiar las distancias con respecto a la anterior 10 metros entre nodo y nodo, hasta llegar a una distancia aproximada de 80 metros cubriendo un área de 6400 m2, donde se puede confirmar que es la distancia máxima para una comunicación eficaz.

Al superar la distancia máxima evaluada de 80 metros un nodo pierde conectividad en la malla haciendo que el resto de nodos se autoconfiguren y continúen con la transmisión, cabe mencionar que los datos descritos pueden tener un margen de error del ±5%, por limitaciones de terreno no se avanzó más distancias.

3.4.2. ESCENARIO DE PRUEBA B

Con los cuatro nodos colocados en línea recta, y con la red en funcionamiento se procede a enviar datos entre los nodos 1 y 3, se van realizando respecto a las distancias iniciando con una de 15 metros, para luego proceder a cambiar a 20 metros y terminando con 30 metros, todo esto se lo realiza en un ambiente abierto (sin obstáculos).

Para la visualización de los paquetes se conecta un ordenador al nodo que recibe el paquete. A continuación, en la figura 3.23 se puede observar la distribución física en la que se colocaron los nodos.

Cabe mencionar que se comienza las pruebas desde 15 metros, porque es la distancia mínima para que los nodos se conecten según el escenario planteado.

Figura 3.23. Distribución física en línea recta donde los nodos se encuentran separados entre sí una distancia de 30 metros, en el en el sector de la Moya cantón Mejía, escenario B.

Una vez obtenidos los datos con la red en funcionamiento, a continuación, se procede a analizar cada uno de sus parámetros de rendimiento.

3.4.2.1. ANÁLISIS DE LA MÉTRICA DE RETARDOS

En la siguiente figura 3.24 se pueden observar los distintos retardos promedios que se obtiene de cada una de las variaciones de distancia y cantidad de paquetes enviados por la red.

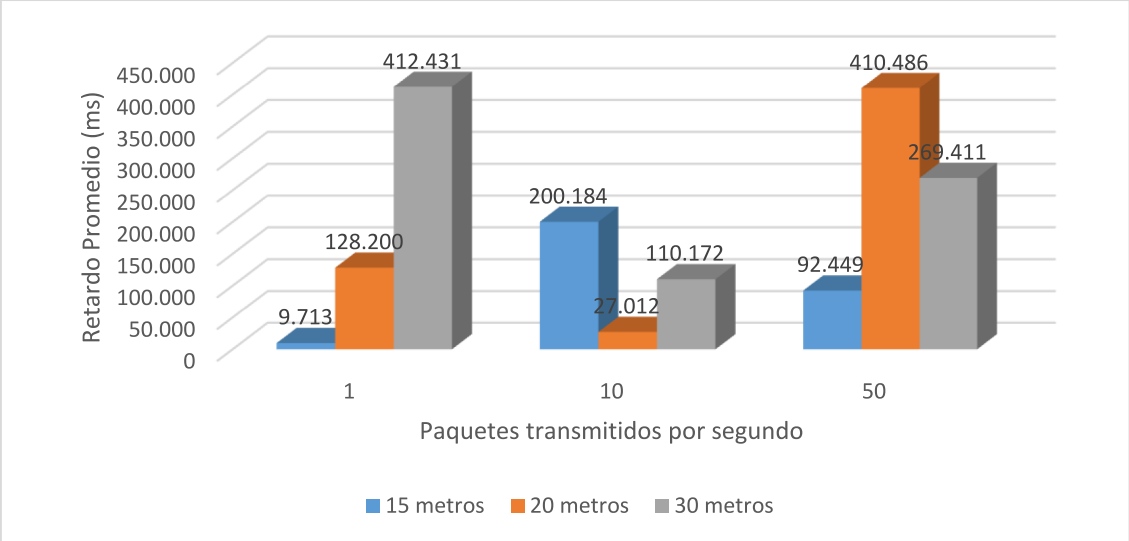


Figura 3.24. Retardo promedio vs paquetes transmitidos por segundo a diferentes distancias escenario B.

A partir de los datos obtenidos, se procede a realizar gráficos que permiten de mejor manera apreciar los resultados obtenidos en cada uno de las variaciones realizadas dentro de la red. De la figura 3.24 se obtiene como resultados lo siguiente:

- De todos los cambios efectuados en la red, se puede observar que el mínimo retardo se presenta cuando se envía un paquete por la red a una distancia de 15 metros.

- Se puede observar en la figura que existen un tiempo significativo de retardo en cambiar de 15 metros a 30 metros y enviando 1 paquete por segundo, por lo que a medida que se alejan más los nodos a los paquetes les cuesta llegar a su destino final.

- Se puede observar que la red, tiene su máximo retardo al enviar 50 paquetes sobre la red y a una distancia de 30 metros, por lo que se puede deducir que al existir un nodo intermedio el paquete tiene que pasar de forma obligatoria por este nodo aumentando los retardos.

3.4.2.2. ANÁLISIS DE LA MÉTRICA DE PÉRDIDA DE PAQUETES

Se presenta a continuación en la figura 3.25 los datos recolectados en esta prueba, hay que tener en cuenta que para obtener estos datos se cambian diferentes parámetros en la red, con lo que permite evaluar la pérdida de paquetes que se presentan durante el envío de la información.

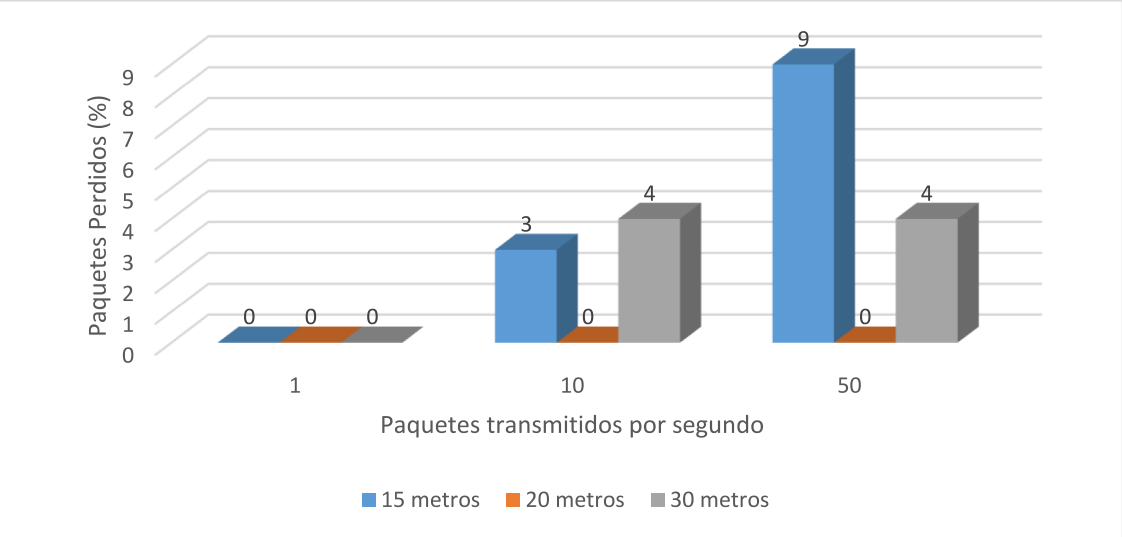


Figura 3.25. Tasa de entrega de paquetes vs la distancia para tres distintas cantidades de paquetes enviados por segundo en el escenario B.

En la figura 3.25 se representa un gráfico estadístico que muestra el análisis detallado de como afecta esta métrica en la red; a continuación, se describen los aspectos más importantes de esta métrica:

- En la figura se puede observar, que no se tiene pérdidas de paquetes cuando se envía por la red un paquete por segundo en las tres distancias evaluadas.

- Se visualiza en la figura, que a medida que se aumenta el envío de la cantidad de paquetes sobre la red inalámbrica empieza a tener pérdida de paquetes, pero no son significativos.

- Tomando en cuenta que las pruebas se realizaron durante varios días, se observa en la figura 3.25 que la máxima pérdida de paquetes se da al sobrecargar la red con 50 paquetes por segundo a una distancia de 15 metros, este dato anómalo se presentó posiblemente por la presencia desfavorable del clima en ese momento.

3.4.2.3. ANÁLISIS DE LA MÉTRICA DE THROUGHPUT

A continuación, en la figura 3.26 se observa los datos obtenidos en las pruebas realizadas con respecto a la métrica de throughput, donde se presentan las distancias que se varían, así como los paquetes transmitidos por la red y su rendimiento promedio obtenido.

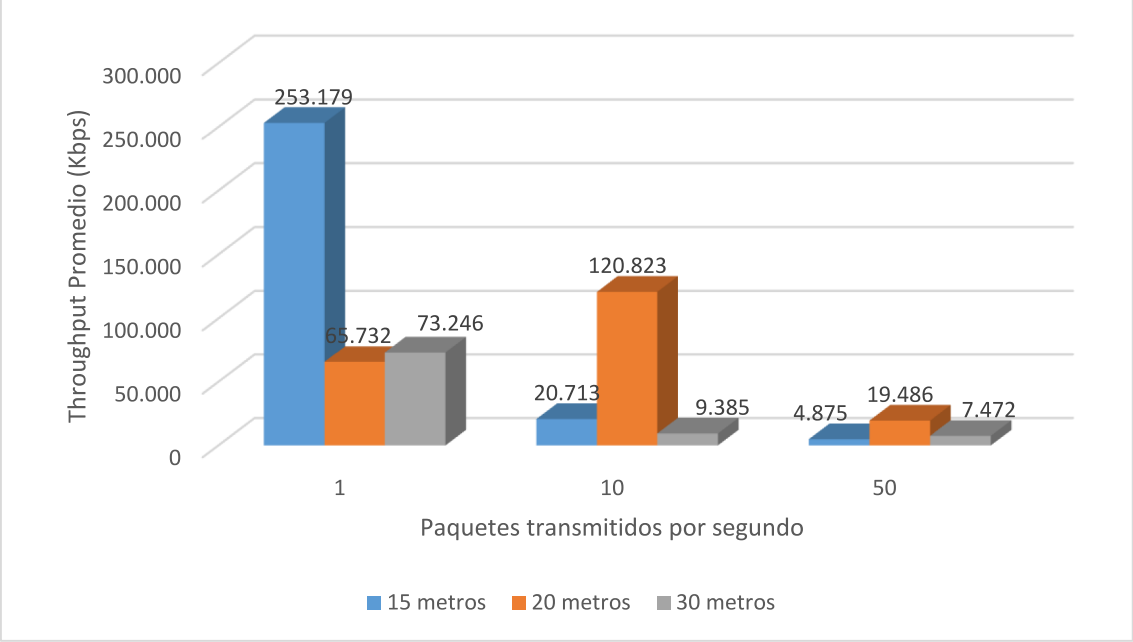


Figura 3.26. Throughput vs distancia entre nodos escenario B.

Cabe mencionar que para la evaluación de está métrica, los datos obtenidos están directamente relacionados con la cantidad de paquetes perdidos y el retardo, por lo tanto, los resultados que se muestran son consecuencia de las anteriores métricas analizadas. A continuación, se detallan los aspectos más importantes que se visualizan en la figura 3.26.

- Se obtiene el máximo rendimiento a distancias de 15 metros y enviando 1 paquete por segundo, se obtiene un rendimiento alto ya que los retardos medios son bajos, así como la pérdida de paquetes es nula.

- Se puede visualizar que en la variación donde se transmite 10 paquetes por segundo por la red, afecta su rendimiento en dos de las tres distancias, teniendo un dato anómalo a la distancia de 20 metros esto se debe a que en la mitad de la prueba, se dio una autoconfiguración de los nodos lo que ocasionó una conexión de forma directa entre los dos nodos que intercambian la información.

- Se obtiene un bajo rendimiento de la red cuando se transmiten 50 paquetes por segundo, bajando sustancialmente su desempeño con esta cantidad de tráfico.

3.4.2.4. ANÁLISIS DE LA MÉTRICA DE DISTANCIAS MÁXIMAS

Con los cuatro nodos en funcionamiento y colocados en línea recta, se comienza a realizar la prueba a una distancia de 30 metros enviando paquetes de datos del nodo 1 al nodo 4 que actúan como extremos de la topología y los nodos 2 y 3 los nodos intermedios, se va variando las distancias con respecto a la anterior 10 metros entre nodo y nodo, se llega a tener una distancia aproximada de extremo a extremo con los dos nodos intermedios de 240 metros, distancia en la cual se mantiene la comunicación, cabe mencionar que los datos descritos pueden tener un margen de error de ± 5%, por limitaciones del terreno no se avanzó más distancias.

Una vez superada esta distancia, los nodos que se encontraban a los extremos comenzaron a tener fallos en la conexión.

3.4.3. ESCENARIO DE PRUEBA C

En esta prueba se colocan los cuatro nodos de acuerdo con la topología, distancias y paquetes a ser enviados de acuerdo a como se detalla en el análisis; a continuación, en la figura 3.27 se puede observar la distribución de los nodos.

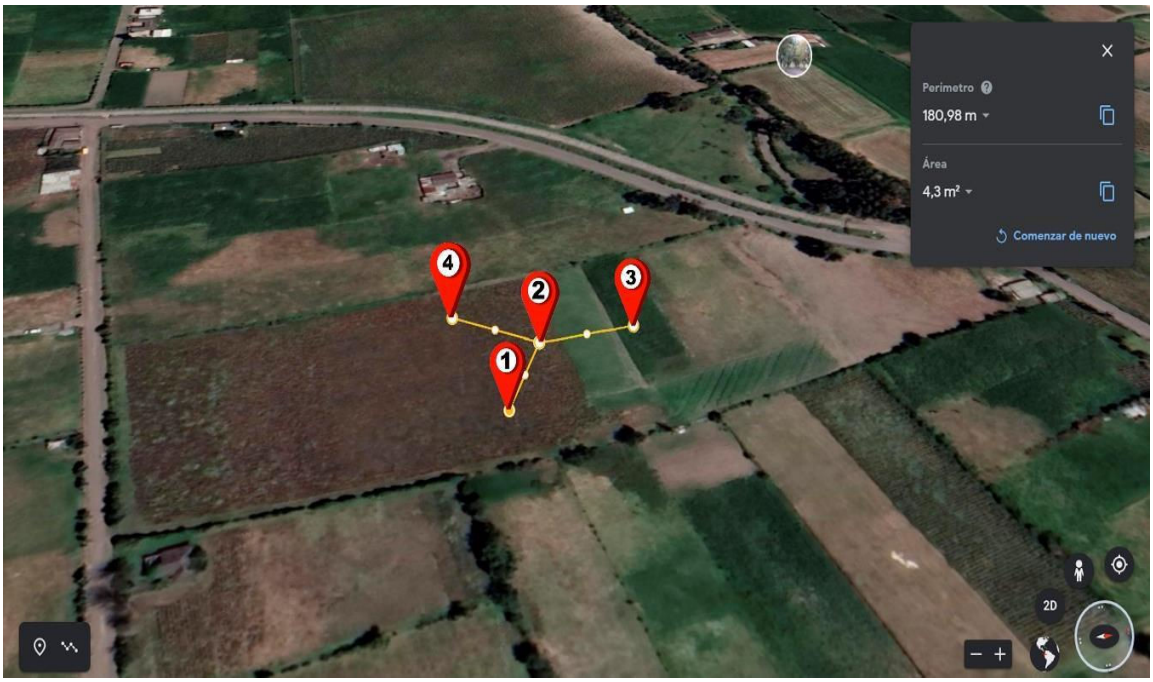


Figura 3.27. Distribución física en estrella donde los nodos se encuentran separados entre sí una distancia de 30 metros, en el sector de la Moya cantón Mejía, escenario C.

3.4.3.1. ANÁLISIS DE LA MÉTRICA DE RETARDOS

Los datos que se recolectan en esta prueba se pueden observar en la figura 3.28.

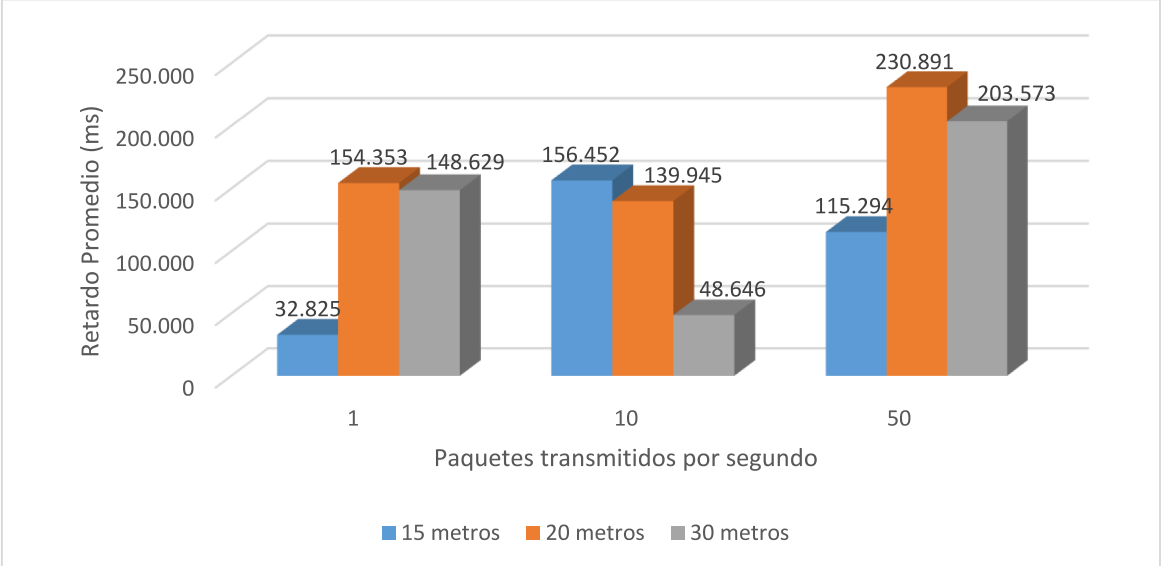


Figura 3.28. Retardo promedio vs paquetes transmitidos por segundo a diferentes distancias escenario C.

A continuación, se da un análisis más detallado de los datos obtenidos con la ayuda de la figura 3.28, en donde se presentan gráficos estadísticos de cómo se da la evolución de la red en todas las variaciones efectuadas.

- En la figura se puede visualizar, que existe el menor retardo cuando se envía un solo paquete por la red y a una distancia de 15 metros, esto se puede dar ya que a distancias pequeñas puede existir menos interferencia.

- Se puede observar en la figura, que aumenta el retardo a medida que se aumenta la cantidad de paquetes transmitidos por la red y la distancia entre los nodos.

- Existe un dato anómalo al transmitir 10 paquetes por segundo a una distancia de 30 metros, ya que se tiene un bajo retardo esto a que se usan los relojes internos de cada uno de los dispositivos para medir estos retardos lo que implica que pueden existir errores en la medición de estos tiempos.

3.4.3.2. ANÁLISIS DE LA MÉTRICA DE PERDIDA DE PAQUETES

A continuación, en la figura 3.29 se puede observar los datos obtenidos en la prueba realizada y su porcentaje de la misma.

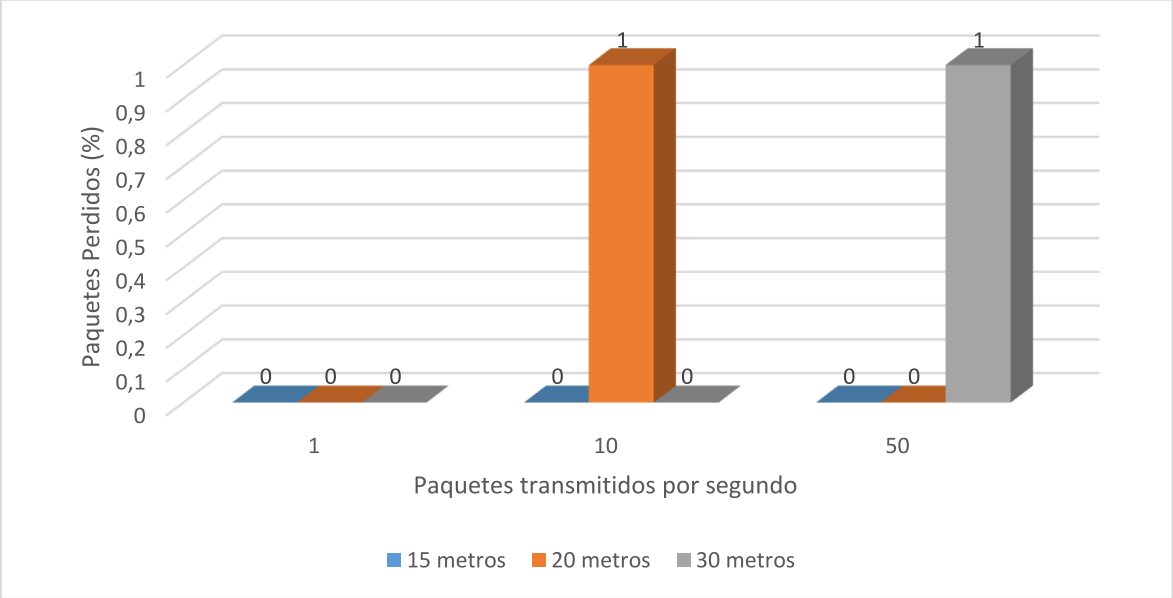


Figura 3.29. Tasa de entrega de paquetes vs la distancia para tres distintas cantidades de paquetes enviados por segundo en el escenario C.

En la figura 3.29 se representa un gráfico estadístico que muestra el porcentaje de paquetes perdidos vs la distancia, en esta gráfica se destacan los siguientes datos importantes:

- Se puede apreciar en la figura, que el mínimo porcentaje de pérdida de paquetes se encuentra a la distancia más corta evaluada cuando se envía un paquete por segundo.

- Se puede visualizar, que la red puede variar en todo momento, como es el caso donde se obtienen valores casi similares de pérdida de paquetes ya sea transmitiendo un paquete o cincuenta paquetes por la red, todo esto hablando en porcentajes.

- Se puede observar en este gráfico que la red obtuvo sus mayores porcentajes de pérdidas de paquetes al momento de enviar diez paquetes por la red, por lo que este tipo de redes y con esta topología varía sustancialmente sin tener un patrón a seguir en la pérdida de paquetes.

3.4.3.3 ANÁLISIS DE LA MÉTRICA DE THROUGHPUT

A continuación, en la figura 3.30 se observan los datos obtenidos en las pruebas realizadas con respecto a la métrica de throughput, donde se presentan las distancias que se van variando, como los paquetes transmitidos por la red y su rendimiento promedio obtenido.

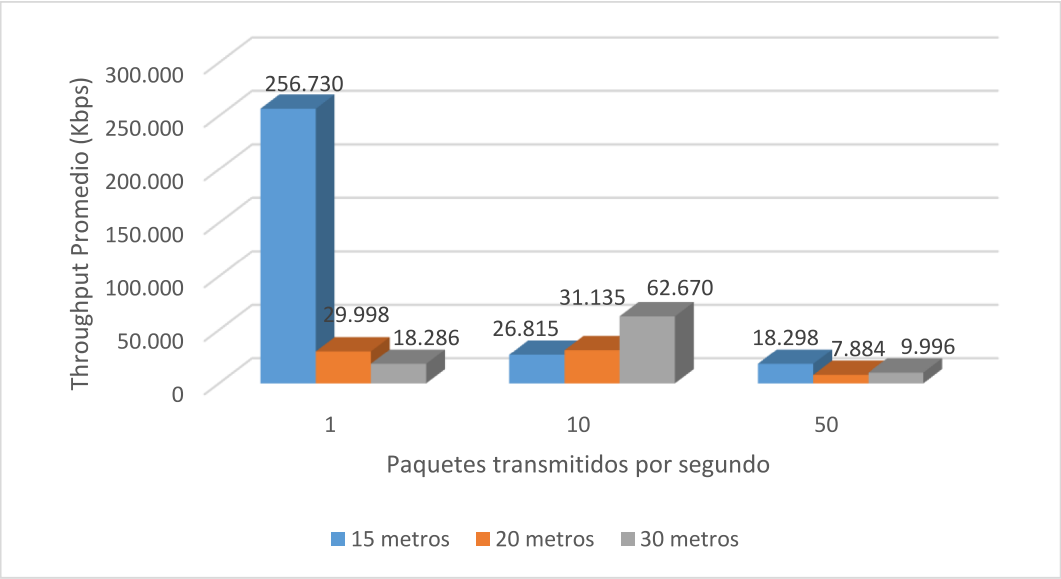


Figura 3.30. Throughput vs distancia entre nodos escenario C.

Cabe mencionar que para la evaluación de está métrica, los datos obtenidos están directamente relacionados con la cantidad de paquetes perdidos y el retardo, por lo tanto, los resultados que se muestran son consecuencia de las anteriores métricas analizadas.

Posteriormente, se detalla el punto más importante que se puede observar en la figura 3.30:

- Se puede observar que se obtiene el resultado esperado, cuando se envía un paquete por segundo, lo que implica que a menores distancias existe un mayor rendimiento de la red.

- Al sobrecargar la red con una cantidad de 50 paquetes por segundo, se obtiene rendimientos bajos con respecto a las demás pruebas, esto se da debido a la cantidad de información enviada y las retrasmisiones generadas entre los nodos.

3.4.3.4 Análisis de distancias máximas

Colocados los cuatro nodos en topología tipo estrella y en funcionamiento, se procede a probar la distancia máxima que alcanza entre un nodo extremo hasta otro nodo extremo pasando de forma obligatorio por el nodo intermedio el paquete, se inicia la prueba a una distancia de 30 metros desde el nodo central a los nodos extremos, para posteriormente cambiar las distancias con respecto a la anterior 10 metros entre el nodo central y sus extremos. Se llega a tener una distancia de extremo a extremo aproximada de 140 metros, y del nodo intermedio al nodo extremo de aproximadamente de 70 metros con un margen de error del ±5%, distancia en la cual se mantiene la comunicación, por limitaciones del terreno no se avanzó más distancias.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- Los dispositivos NodeMCU ESP32 van ocupando un lugar muy importante en lo referente a placas de desarrollo, ya que se adaptan a entornos y lenguajes de programación consolidados como son Arduino y C++. Gracias a esto, estos dispositivos son usados con más frecuencia en distintas aplicaciones del internet de las cosas lo que es el auge en la actualidad.

- El uso de librerías de código abierto, como es el caso de painlessMesh, permite la creación de redes tipo malla de forma simplificada, lo que permite al programador trabajar sin estar pendiente de cómo está estructurada o de la administración de la red.

- El tener dispositivos que se sincronizan de forma automática sus relojes internos cada vez que ingresan a la red, permite un funcionamiento adecuado de la red inalámbrica, sin embargo, no es lo recomendable para medir retardos a nivel de capa MAC, dado que este tipo de medidas requiere de mayor precisión. A falta de una mejor alternativa, se usó los mismo en el desarrollo de este proyecto.

- Mediante el análisis de los resultados obtenidos se concluye que, al trabajar con dispositivos de bajo coste su memoria es muy limitada, lo que implica que se debe ser conservador con la cantidad de información que se transmite por la red, en especial cuando se trabaja con un tráfico de difusión tipo broadcast, ya que podemos saturar la red en su totalidad.

- La eficiencia de la red varía por factores como: las distancias entre los nodos, el cual es un factor de suma importancia que afecta en la pérdida de paquetes. Así como la sobrecarga de la red con información afectando sustancialmente en el retardo de la lectura de los datos por parte del receptor.

- Con respecto a la autoconfiguración de la red, se pudo observar en las pruebas que los nodos se auto configuran en forma autónoma en un tiempo promedio de unos

7 segundos, permitiendo a la red estar operativa de nuevo y enviando mensajes por las mejores rutas.

- Se obtuvo un máximo throughput de 256.730 Kbps, con un porcentaje de pérdidas del 0% y un retardo de 32.825 ms usando los dispositivos NodeMCU ESP32 y en el escenario C. Estos valores se obtienen a distancias cortas que influye en el rendimiento de las métricas evaluadas. Además, se comprobó que, a una tasa de envío de un paquete por segundo, se obtienen menores retardos y menores pérdidas de paquetes

4.2. RECOMENDACIONES

- A pesar de que los dispositivos NodeMCU ESP32 son de bajo coste, tienen un amplio número de características y funciones, por lo que se recomienda antes de su uso una lectura minuciosa de sus hojas de especificaciones o valerse de la comunidad de desarrollo que existe en internet sobre estas placas de desarrollo, lo que ayudará a entender de mejor manera el funcionamiento del mismo.

- Pueden existir mejoras en los análisis realizados, mejorando la precisión en la toma de los tiempos de retardo entre el envío y recepción de los mensajes de la red, Una forma de hacer esto es incorporando nuevos parámetros que permitan determinar más características que ayuden a precisar los resultados de mejor manera.

- Se recomienda que para darle movilidad a los dispositivos sin que dependa del puerto USB, se debe de alimentar al NodeMCU ESP32 con una batería de litio, donde su tensión debe estar entre el intervalo de 3,3 a 4,2 V y su corriente de 4800 mAh para que funcione de forma correcta, sin tener desconexiones a la hora de estar en funcionamiento el dispositivo.

- Se recomienda realizar las pruebas en un ambiente con obstáculos para su respectivo análisis.

- Se recomienda a partir de este análisis de la red, implementar un prototipo de sistema real de aplicaciones IoT, usando distintos módulos sensores que permitan capturar los datos de los elementos que se están evaluando.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Sebastián Buettrich. REDES MESH. [En línea]. Available: http://www.itrainonline.org/itrainonline/mmtk/wireless_es/files/13_es_redes_mesh_presentacion_v02.pdf. [Último acceso: 10 de marzo del 2020].

[2] Akyildiz, I., Wang, X., Wang, W.: Wireless mesh networks: a survey, In Computer Networks. Vol. 47. No.4 pp. 455--487 (2005)

[3] DMS, «CloudService, » [En línea]. Available: https://dms.com.pe/redes-mesh/. [Último acceso: 25 de marzo del 2020]

[4] Andrés Raúl Bruno Saravia. «ESP32 NODE MCU.» [En línea]. Available: http://www.microelectronicash.com/ofertas.php. [Último acceso:25 de marzo del 2020].

[5] Xukyo. «Crear una interfaz web para controlar su ESP32 NodeMCU» [En línea]. Available: https://www.aranacorp.com/es/crear-una-interfaz-web-para-controlar-su-esp32-nodemcu/. [Último acceso: 29 de marzo del 2020].

[6] Enrique Crespo. «APRENDIENDO ARDUINO» [En línea]. Available: https://aprendiendoarduino.wordpress.com/2016/03/29/entorno-de-programacion-de-arduino-ide/. [Último acceso: 29 de marzo del 2020]

[7] «Software Espressif Systems,» [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/. [Último acceso: 30 de marzo del 2020].

[8] Peter Scargill. «Mongoose OS» [En línea]. Available: https://tech.scargill.net/mongoose-os/. [Último acceso: 31 de marzo del 2020].

[9] painlessMesh. «Librería painlessmesh para arduino.» [En línea]. Available: https://gitlab.com/painlessMesh/painlessMesh, [Último acceso: 02 de abril del 2020].

[10] DOMOTIC KNOWLEDGE CENTER. «Biblioteca Arduino – Arduino Json.» [En línea]. Available: http://domoticx.com/arduino-library-arduinojson/. [Último Acceso: 02 de abril del 2020].

[11] TaskScheduler «Concept of Task and Cooperative Task Scheduling» [En línea]. Available: https://github.com/arkhipenko/TaskScheduler/wiki/Concept-of-Task-and-Cooperative-Task-Scheduling#task. [Último Acceso: 04 de abril del 2020].

[12] Joel L. Wolf, Philip S. Yu, John JE Turek «Task scheduler for a multiprocessor system » [En línea]. Available: https://patents.google.com/patent/US5437032A/en. [último Acceso: 16 de Abril del 2020].

6. ANEXOS

ANEXO A. Manual de instalación del entorno de desarrollo y librerías de acceso libre

ANEXO B. Script para pruebas de funcionamiento y conectividad.

ANEXO C. Tablas de datos obtenidos en las pruebas.

ANEXO A

INSTALACIÓN DEL ENTORNO DE DESARROLLO IDE ARDUINO

Para la instalación del entorno de desarrollo Arduino, será necesario descargarse el software desde el sitio oficial de Arduino para luego proceder con su respectiva instalación de la misma.

Para la descarga del entorno de desarrollo lo podemos realizar en la siguiente dirección web: https://www.arduino.cc/en/main/software, cabe mencionar que este entorno de desarrollo es open source por lo que puede tener acceso de forma gratuita.

Una vez descargado el archivo ejecutable se procede con la instalación, a continuación, en la figura 1 se podrá observar la instalación del entorno de desarrollo.

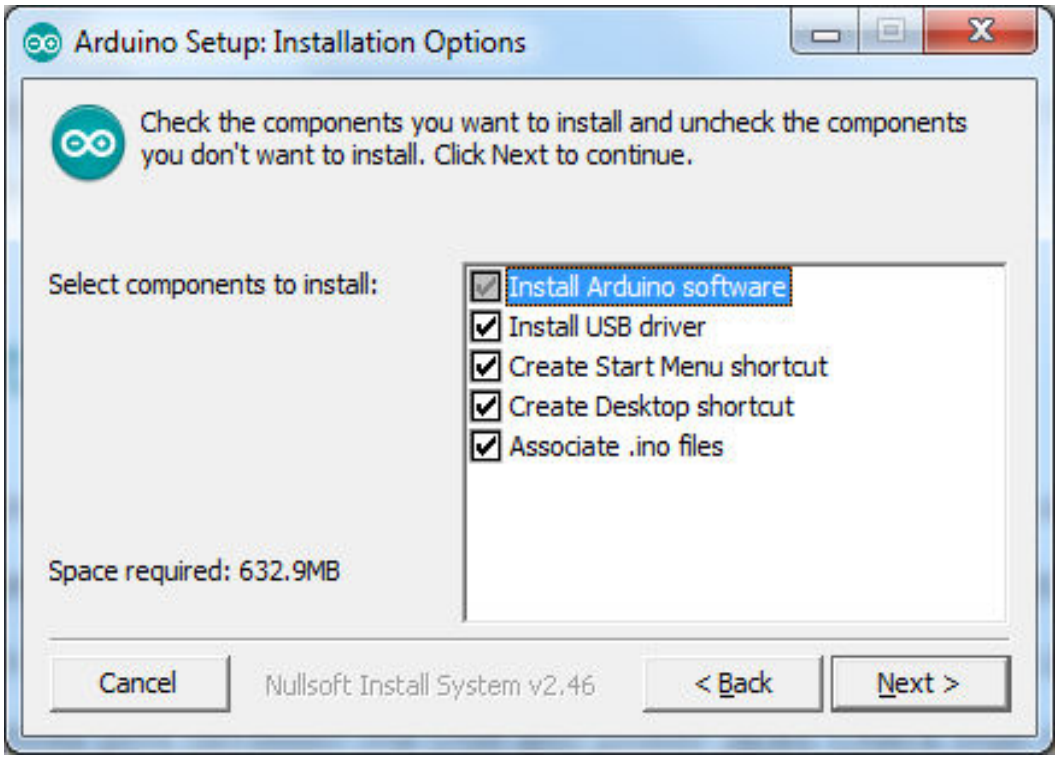


Figura A1. Entorno de instalación IDE Arduino

Una vez realizado todas las directrices que el momento de instalación indica el software nos presenta una interfaz gráfica, donde ya se encuentra instalado el entorno de desarrollo arduino.

INSTALACIÓN DE BIBLIOTECAS DE ACCESO LIBRE

A continuación, se muestra como se carga cada una de las bibliotecas antes mencionadas, dentro del entorno de desarrollo arduino.

INSTALACIÓN BIBLIOTECA PAINLESSMESH

Se realiza la instalación de la biblioteca painlessMesh en el entorno de arduino 1.8.10, la instalación de esta librería se puede realizar de diferentes formas, la manera de instalación que se utiliza en este proyecto es a través del gestor de librerías que posee el IDE de arduino, es importante mencionar que las bibliotecas que se instala a continuación se obtienen de GitHub.

Para la instalación de esta biblioteca se sigue los diferentes pasos que se detalla a continuación:

- Después de iniciar el IDE Arduino, procedemos a dirigirnos a Archivo\Preferencias en donde se escribirá de forma manual o copiar y pegar la dirección https://github.com/gmag11/painlessMesh.git.

- Una vez gestionada la URL, lo que me permite saber en qué directorio se va a colocar las librerías para futuras actualizaciones procedemos a dirigirnos al menú Programa\Incluir Librería\Administrar Bibliotecas, una vez desplegada la ventana del gestor de librerías colocamos en el cuadro de búsqueda painlessMesh lo que se nos abrirá la opción de la librería para ser instalada, a continuación, en la figura 2 se puede observar el procedimiento antes mencionado.

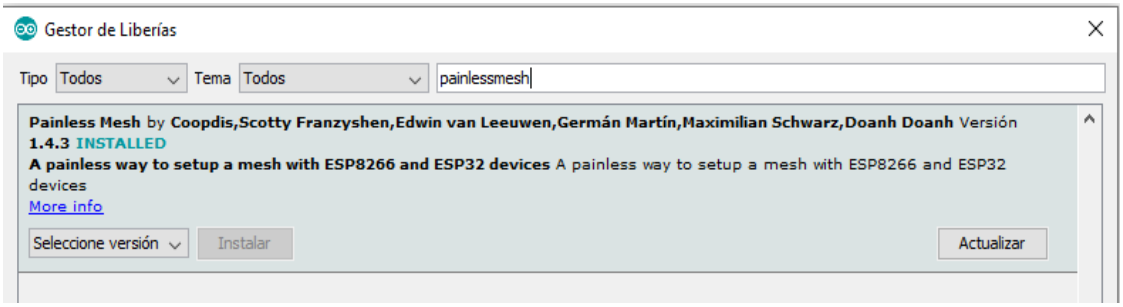


Figura A2. Instalación librería painlessMesh

• Finalmente, realizado todo el proceso antes mencionado se puede usar la biblioteca painlessMesh dentro del IDE Arduino. En la figura 3 se muestra la inclusión de la biblioteca dentro del entorno de Arduino.

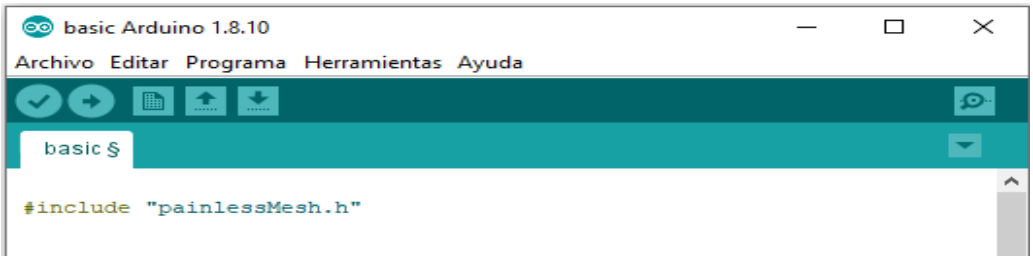


Figura A3. Uso de la biblioteca painlessMesh dentro del IDE Arduino

INSTALACIÓN DE LA BIBLIOTECA ARDUINOJSON

La instalación de esta biblioteca nos ayuda a trabajar con objetos JSON, esta biblioteca trabaja junto a painlessMesh para el envió y/o recepción de los mensajes dentro de la red tipo malla. En la instalación de esta biblioteca se siguen los siguientes pasos que se muestra a continuación:

• Luego de arrancar el IDE nos dirigimos al menú a Archivo\Preferencias en donde se escribirá de forma manual o copiar y pegar la dirección https://github.com/bblanchon/ArduinoJson.git, tal como se mostró en la figura 2.4.

• Gestionada la ubicación de la biblioteca procedemos a instalar la librería, nos dirigirnos al menú Programa\Incluir Librería\Administrar Bibliotecas, una vez desplegada la ventana del gestor de librerías colocamos en el cuadro de búsqueda arduinoJson lo que se nos abrirá la opción de la librería para ser instalada, a continuación, en la figura 4 observaremos el procedimiento antes mencionado.

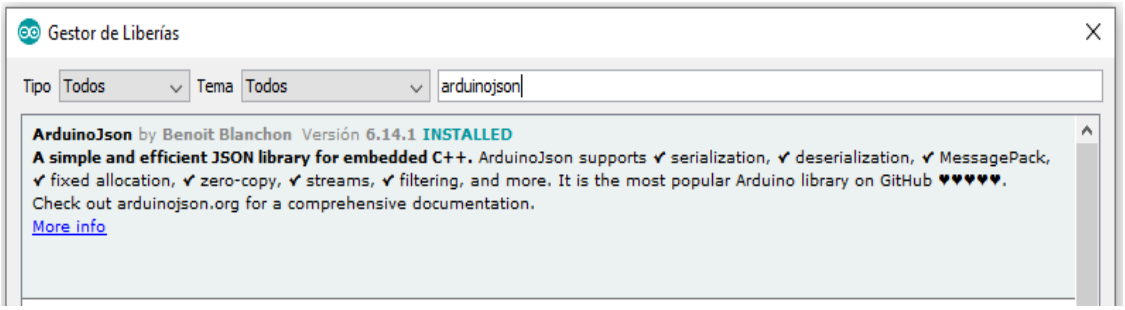


Figura A4. Instalación librería arduinoJson

- Finalizada la instalación de la biblioteca arduinoJson, la biblioteca se puede usar dentro del entorno de desarrollo como se muestra en la figura 5.

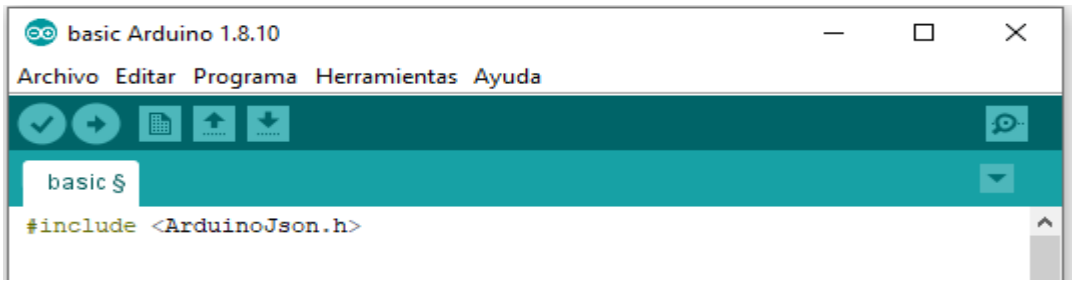


Figura A5. Uso de la biblioteca arduinoJson dentro del IDE Arduino

INSTALACIÓN DE LA BIBLIOTECA TASKSCHEDULER

TaskScheduler trabaja conjuntamente con la biblioteca painlessMesh para realizar la ejecución periódicas de tareas dentro de la red tipo malla. Esta biblioteca trabaja muy bien con el módulo EPS32.

Para la instalación de esta biblioteca se siguen los siguientes pasos que se muestra a continuación:

- Iniciado el entorno de Arduino, procedemos a dirigirnos a Archivo\Preferencias en donde se escribirá de forma manual o copiar y pegar la dirección https://github.com/arkhipenko/TaskScheduler.git, tal como se mostró en la anterior figura 6.

- Procedemos dirigirnos al menú Programa\Incluir Librería\Administrar Bibliotecas, una vez desplegada la ventana del gestor de librerías se coloca en el cuadro de búsqueda el nombre de la librería en este caso TaskScheduler lo que se abrirá la opción de la librería para ser instalada, a continuación, en la figura 6, observaremos el cuadro de dialogo donde se encuentra la biblioteca a instalarse.

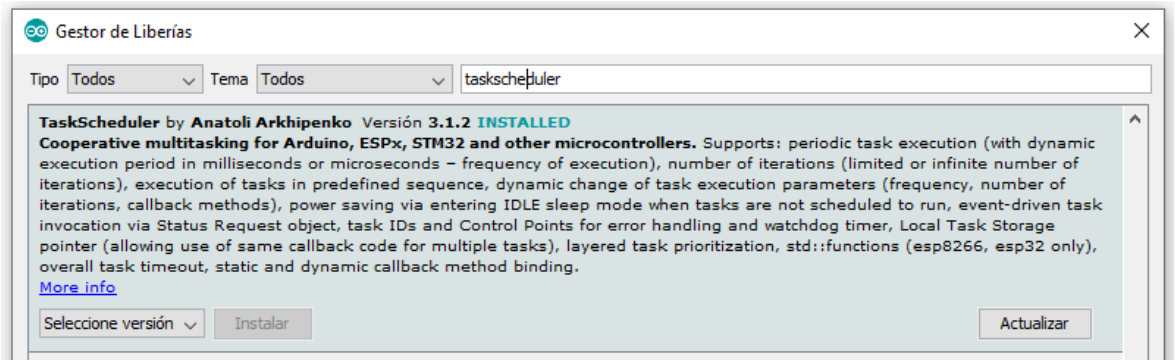


Figura A6 Instalación librería TaskScheduler

- Una vez finalizado el proceso antes mencionado se puede usar la biblioteca TaskScheduler dentro del IDE Arduino. En la figura 7 se mostrará la inclusión de la biblioteca dentro del entorno de Arduino.

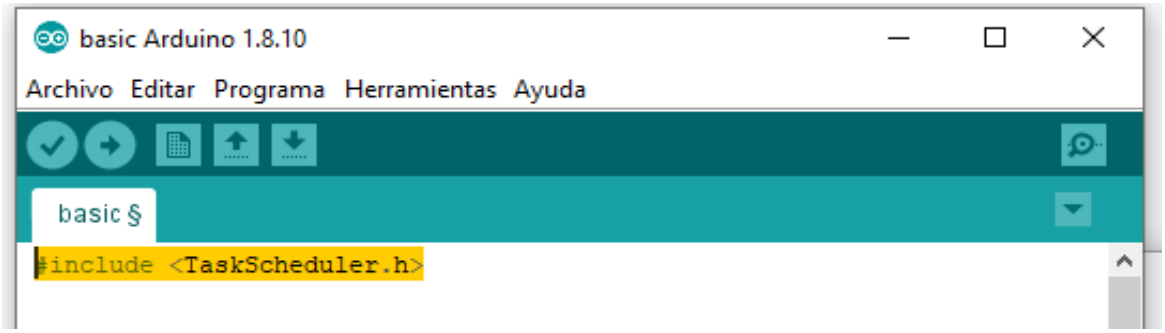


Figura A7. Uso de la biblioteca TaskSchedule dentro del IDE Arduino

INSTALACIÓN DE LA BIBLIOTECA ASYNCTCP

AsyncTCP es una biblioteca TCP completamente asíncrona que trabaja junto con las otras bibliotecas ya que permite la conexión múltiple y sin problemas, de la misma forma que las otras bibliotecas juegan un papel importante al momento de trabajar con la red tipo malla. Para la instalación de esta biblioteca se sigue diferentes pasos que se describirán a continuación:

- Ya iniciado el IDE de Arduino, procedemos a dirigirnos a Archivo\Preferencias en donde se escribirá de forma manual o copiar y pegar la dirección https://github.com/me-no-dev/AsyncTCP.git, tal como se mostró en la figura 8.

- Nos dirigimos al menú Programa\Incluir Librería\Administrar Bibliotecas, una vez desplegada la ventana del gestor de librerías se coloca en el cuadro de búsqueda el nombre de la librería en este caso AsyncTCP lo que se desplegará la librería para

ser instalada, a continuación, en la figura 8. observaremos el cuadro de dialogo donde se encuentra la biblioteca.

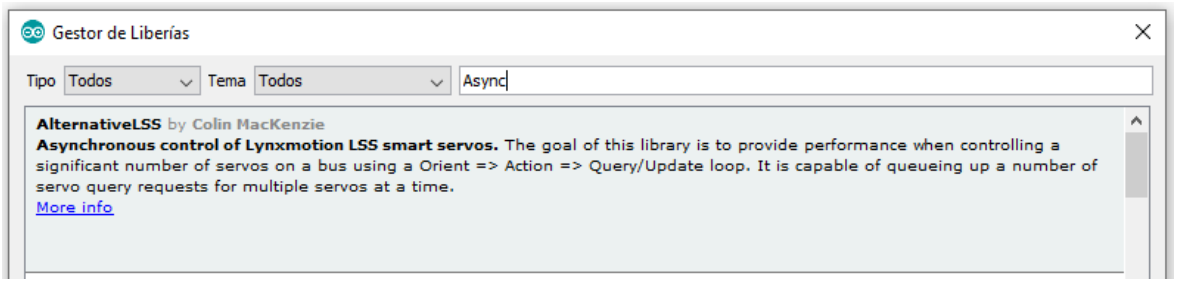


Figura A8. Instalación librería AsyncTCP

- Una vez finalizado el proceso antes mencionado se puede usar la biblioteca AsyncTCP dentro del IDE Arduino. En la figura 9 se muestra la inclusión de la biblioteca dentro del entorno de Arduino.

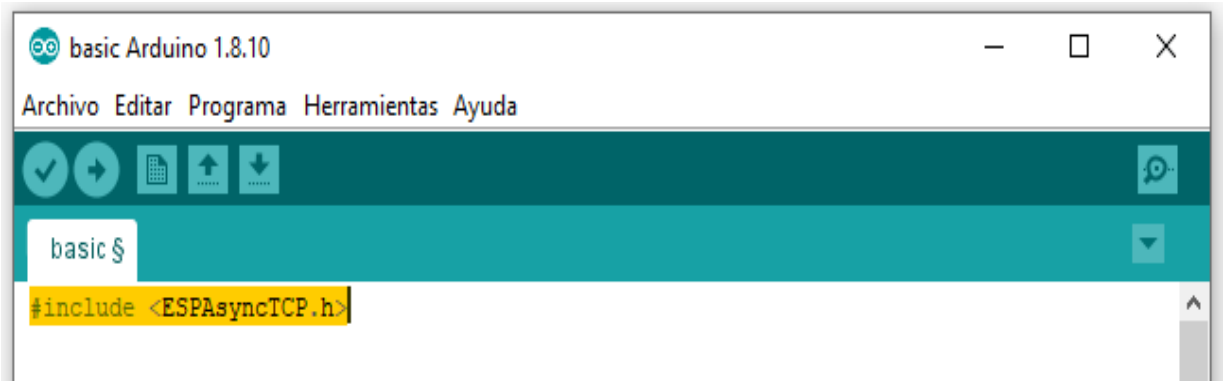


Figura A9. Uso de la biblioteca AsyncTCP dentro del IDE Arduino.

ANEXO B

SCRIPT PARA PRUEBAS DE FUNCIONAMIENTO Y CONECTIVIDAD

Trasmisión broadcast

```
/*Se incluye las directiva de painlessmesh*/
#include "painlessMesh.h"

/*Inicio de la red tipo malla*/
/*Se establece el nombre de la malla*/
#define   MESH_PREFIX     "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD   "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT       5555

/*Declaración del planificador, para gestionar las acciones fuera del
bucle*/
Scheduler usuarioPlanificador;
/*Se instancia la biblioteca painlessmesh*/
painlessMesh  malla;
/*Se da el nombre al dispositivo NodeMCU ESP32*/
String nombreNodo = "Uno";

/*Tarea de ejecución por parte del planificador*/
void enviarMensaje() ;
/*Se ejecuta la tarea en lapso de un segundo y para siempre*/
Task tareaEnviarMensaje( TASK_SECOND * 1 , TASK_FOREVER, &enviarMensaje
);
/*Envío del mensaje con el nombre del nodo y el id del dispositivo*/
void enviarMensaje() {
  /*Mensaje que se va enviar*/
  String msg = "Hola desde el nodo: ";
  /*Se envía el id del dispositivo*/
  msg += malla.getNodeId();
  /*Se envía el alias del nodo*/
  msg += ", alias  ";
  /*Se envía el nombre del dispositivo que esta trasmitiendo*/
  msg += nombreNodo;
```

```
  /*Enviar el mensaje a cada nodo de toda la red malla  */
  malla.sendBroadcast( msg );
  /*Cambio de intervalo en forma aleatoria entre 1 a 5 segundos*/
  tareaEnviarMensaje.setInterval( random( TASK_SECOND * 1, TASK_SECOND *
5 ));
}

/*Es necesario para que trabaje la red tipo malla inalámbrica*/

/*Se maneja los mensajes de entrada*/
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("Recibiendo mensajes desde el nodo: %u msg=%s\n", from,
msg.c_str());
}

/*Se maneja las nuevas conexciones que ingresan a la malla*/
void newConnectionCallback(uint32_t nodeId) {
    Serial.printf("-->  Nueva Conexion, Id del nodo: = %u\n", nodeId);
}

/*Se maneja un cambio en la topología de la malla, si alguien entra o
sale*/
void changedConnectionCallback() {
  Serial.printf("Cambios en la conexion\n");
}

/*Se sincroniza la hora de la red malla*/
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Tiempo Ajustado  %u. Offset = %d\n",
malla.getNodeTime(),offset);
}

/*Función de ejecución de las libreria al momento de iniciar*/
void setup() {
  /*Velocidad de comunicación serial*/
  Serial.begin(115200);
  /*Estado de depuraión*/
  malla.setDebugMsgTypes( ERROR | STARTUP | CONNECTION  );
  /* Inicializa la red malla.
    Inicia una red wifi
```

```
    Comienza a buscar otras redes wifi que son parte de la malla
    Inicia sesión en el mejor nodo de red de malla que encuentra. Si no
encuentra nada, comienza una nueva búsqueda en 5 segundos.
  */
  malla.init( MESH_PREFIX, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT
);

  /*Id de callback declarados anteriormente*/
  malla.onReceive(&receivedCallback);
  malla.onNewConnection(&newConnectionCallback);
  malla.onChangedConnections(&changedConnectionCallback);
  malla.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);

  /*Se añade la tarea en el planificador*/
  usuarioPlanificador.addTask( tareaEnviarMensaje );
  /*Se habilita la tarea*/
  tareaEnviarMensaje.enable();
}

void loop() {
  /*Se ejecuta varias tareas de mantenimiento y actualización de la red*/
  malla.update();
}
```

Recepción broadcast

```
/*Se incluye las directiva de painlessmesh*/
#include "painlessMesh.h"
/*Inicio de la red tipo malla*/
/*Se establece el nombre de la malla*/
#define   MESH_PREFIX     "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD   "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT       5555

Scheduler usuarioPlanificador;
/*Se instancia la biblioteca painlessmesh*/
painlessMesh  malla;
/*Se da el nombre al dispositivo NodeMCU ESP32*/
```

```
String nombreNodo = "Dos";

/*Es necesario para que trabaje la red tipo malla inalámbrica*/
/*Se maneja los mensajes de entrada*/
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf(" Recibiendo mensajes desde el nodo: %u msg=%s\n", from,
msg.c_str());
}

/*Se maneja las nuevas conexciones que ingresan a la malla*/
void newConnectionCallback(uint32_t nodeId) {
    Serial.printf("--> Nueva Conexion, Id del nodo: = %u\n", nodeId);
}
/*Se maneja un cambio en la topología de la malla, si alguien entra o
sale*/
void changedConnectionCallback() {
  Serial.printf("Cambios en la conexion\n");
}
/*Se sincroniza la hora de la red malla*/
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Tiempo Ajstado: %u. Offset = %d\n",
malla.getNodeTime(),offset);
}
/*Función de ejecución de las libreria al momento de iniciar*/
void setup() {
/*Velocidad de comunicación serial*/
  Serial.begin(115200);
/*Estado de depuraión*/
  malla.setDebugMsgTypes( ERROR | STARTUP | CONNECTION  );
    /* Inicializa la red malla.
    Inicia una red wifi
    Comienza a buscar otras redes wifi que son parte de la malla
    Inicia sesión en el mejor nodo de red de malla que encuentra. Si no
encuentra nada, comienza una nueva búsqueda en 5 segundos.
  */
  malla.init( MESH_PREFIX, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT
);
  /*Id de callback declarados anteriormente*/
  malla.onReceive(&receivedCallback);
  malla.onNewConnection(&newConnectionCallback);
```

```
  malla.onChangedConnections(&changedConnectionCallback);
  malla.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
}
void loop() {
  /*Se ejecuta varias tareas de mantenimiento y actualización de la red*/
  malla.update();
}
```

Transmisión unicast

```
/*Se incluye las directiva namedMesh heradada de painlessmesh*/
#include "namedMesh.h"

/*Inicio de la red tipo malla*/
/*Se establece el nombre de la malla*/
#define   MESH_SSID       "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD   "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT       5555

/*Declaración del planificador, para gestionar las tareas*/
Scheduler usuarioPlanificador;
/*Se instancia la biblioteca namedMesh*/
namedMesh  malla;
/*Se da un nombre único al dispositivo NodeMCU ESP32*/
String nombreNodo = "Uno";
/*Se ejecuta la tarea en lapso de treinta segundos y para siempre*/
Task tareaEnvioMensajeUnicast( TASK_SECOND*30, TASK_FOREVER, []() {
  /*Mensaje que se envia a traves de la red con el nombre del nodo y el
id*/
    String msg = String("Este es un mensaje desde: ") + nombreNodo +
String("Dos");
    /*Declaración del nombre del nodo hacia donde esta dirigido*/
    String to = "Dos";
    /*Envío del mensaje de forma unicast */
    malla.sendSingle(to, msg);
});

/*Función de ejecución de las libreria al momento de iniciar*/
```

```
void setup() {
  /*Velocidad de comunicación serial*/
  Serial.begin(115200);
  /*Estado de depuraión de la malla*/
  malla.setDebugMsgTypes(ERROR | DEBUG | CONNECTION);

 /* Inicializa la red malla.
    Inicia una red wifi
    Comienza a buscar otras redes wifi que son parte de la malla
    Inicia sesión en el mejor nodo de red de malla que encuentra. Si no
encuentra nada, comienza una nueva búsqueda en 5 segundos.
  */
  malla.init(MESH_SSID, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT);
  /*Se especifica un nombre unico al nodo dentro de la red malla*/
  malla.setName(nombreNodo);
  /*Se maneja los mensajes de entrada al dispositivo por id*/
  malla.onReceive([](uint32_t from, String &msg) {
    Serial.printf("Mensaje Recibido por el id desde: %u, %s\n", from,
msg.c_str());
  });
  /*Se maneja los mensajes de entrada al dispositivo por nombre*/
  malla.onReceive([](String &from, String &msg) {
    Serial.printf("Mensaje recibido por el nombre desde: %s, %s\n",
from.c_str(), msg.c_str());
  });
  /*Se maneja un cambio en la topología de la malla, si alguien entra o
sale*/
  malla.onChangedConnections([]() {
    Serial.printf("Cambio de conexión en la malla\n");
  });
  /*Se añade la tarea en el planificador*/
  usuarioPlanificador.addTask(tareaEnvioMensajeUnicast);
  /*Se habilita la tarea*/
  tareaEnvioMensajeUnicast.enable();
}
void loop() {
   /*Se ejecuta varias tareas de mantenimiento y actualización de la
red*/
  malla.update();
}
```

Recepción unicast

```
/*Se incluye las directiva namedMesh heradada de painlessmesh*/
#include "namedMesh.h"

/*Inicio de la red tipo malla*/
/*Se establece el nombre de la malla*/
#define   MESH_SSID       "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD   "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT       5555

/*Declaración del planificador, para gestionar las tareas*/
Scheduler usuarioPlanificador;
/*Se instancia la biblioteca namedMesh*/
namedMesh  malla;
/*Se da un nombre único al dispositivo NodeMCU ESP32*/
String nombreNodo = "Uno";

/*Función de ejecución de las libreria al momento de iniciar*/
void setup() {
  /*Velocidad de comunicación serial*/
  Serial.begin(115200);
  /*Estado de depuraión de la malla*/
  malla.setDebugMsgTypes(ERROR | DEBUG | CONNECTION);
  /* Inicializa la red malla.
    Inicia una red wifi
    Comienza a buscar otras redes wifi que son parte de la malla
    Inicia sesión en el mejor nodo de red de malla que encuentra. Si no
encuentra nada, comienza una nueva búsqueda en 5 segundos.
  */
  malla.init(MESH_SSID, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT);
  /*Se especifica un nombre unico al nodo dentro de la red malla*/
  malla.setName(nombreNodo);
  /*Se maneja los mensajes de entrada al dispositivo por id*/
  malla.onReceive([](uint32_t from, String &msg) {
```

```
    Serial.printf("Mensaje Recibido por el id desde: %u, %s\n", from,
msg.c_str());
  });
  /*Se maneja los mensajes de entrada al dispositivo por nombre*/
  malla.onReceive([](String &from, String &msg) {
    Serial.printf("Mensaje recibido por el nombre desde: %s, %s\n",
from.c_str(), msg.c_str());
  });
  /*Se maneja un cambio en la topología de la malla, si alguien entra o
sale*/
  malla.onChangedConnections([]() {
    Serial.printf("Cambio de conexión en la malla\n");
  });
}
void loop() {
   /*Se ejecuta varias tareas de mantenimiento y actualización de la
red*/
  malla.update();
}
```

SCRIPT PARA PRUEBAS DE RENDIMIENTO

Transmisión Unicast

```
#include "painlessMesh.h"
/*Se establece el ssid de la red malla*/
#define   MESH_PREFIX    "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD  "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT      5555

Scheduler usuarioPlanificador;
painlessMesh malla;

void enviarMensaje();
uint32_t destino = 312130145;

Task tareaEnviarMensaje( TASK_SECOND * 1 , TASK_FOREVER, &enviarMensaje
);
/*Inicialización del contador de mensajes*/
```

```
int nmes=0;
String connectionMap;
/*Nombre del nodo*/
const char* idNodo ="UNO";
char msg[50];

void enviarMensaje() {
  /*EStructura del mensajes*/
  /*Nombre del nodo, número de mensaje, tiempo de transmisión*/
  /* Incremento del contador*/
  nmes++;
  /*Obtención del tiempo de transmisión*/
  int timeTX = malla.getNodeTime();
  /*Mensaje que se envía*/
  snprintf (msg, 50, "%s, %d, %d, ", idNodo, nmes, timeTX);
  /*Envío tráfico unicast*/
  malla.sendSingle( destino, msg);
  /*Intervalo de envío es de un segundo*/
  tareaEnviarMensaje.setInterval( TASK_SECOND * 1);
}

/*Rcepción del mensaje*/
void receivedCallback( uint32_t from, String &msg ) {
  // RX node, power rssi, RX time, message
  /*int rssi= WiFi.RSSI();*/
  /*Estructura de la recepción*/
  /*Tiempo de la recepción*/
  int timeRX = malla.getNodeTime();
  /*Nombre del nodo receptor*/
  Serial.print(idNodo);
  Serial.print(", ");
  /*Tiempo de recepción*/
  Serial.print(timeRX);
  Serial.print(", ");
  /*Mensaje recibido*/
  Serial.println(msg);
}

/*Método conexión del nodo*/
void newConnectionCallback(uint32_t idNodo) {
```

```
    Serial.print(idNodo);
    Serial.print(", ");
    Serial.println("Nueva Conexion");

}

/* Cambio de conexión*/
void changedConnectionCallback() {
  Serial.printf("Cambio de Conexion\n");
  connectionMap=malla.subConnectionJson();
  char msg[connectionMap.length()];
  connectionMap.toCharArray(msg, connectionMap.length());
  Serial.printf("Conexion en la red = %s\n", msg);;
}

/*Sinconización del reloj de la red*/
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Tiempo Ajustado %u. Offset = %d\n",
malla.getNodeTime(),offset);
}

void setup() {
  /*Velocidad de comunicación serial*/
  Serial.begin(115200);
  malla.setDebugMsgTypes( ERROR | STARTUP | MESH_STATUS );
  malla.init( MESH_PREFIX, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT
);

  /*Id de callback declarados anteriormente*/
  malla.onReceive(&receivedCallback);
  malla.onNewConnection(&newConnectionCallback);
  malla.onChangedConnections(&changedConnectionCallback);

  /*Se añade la tarea en el planificador*/
  usuarioPlanificador.addTask( tareaEnviarMensaje );
  /*Se habilita la tarea*/
  tareaEnviarMensaje.enable();
}

void loop() {
```

```
  /*Se ejecuta varias tareas de mantenimiento y actualización de la red*/
  usuarioPlanificador.execute();
  malla.update();
}
```

Recepción Unicast

```
#include "painlessMesh.h"
/*Se establece el ssid de la red malla*/
#define   MESH_PREFIX       "redMallaEPN"
/*Se establece la constraseña de la red malla*/
#define   MESH_PASSWORD   "mallaTesisEPN"
/*Se establece el puerto TCP en el que se ejecuta la malla*/
#define   MESH_PORT       5555

Scheduler usuarioPlanificador;
painlessMesh malla;
/*Inicialización del contador de mensajes*/
int nmes=0;
String connectionMap;
/*Nombre del nodo*/
const char* idNodo ="CUATRO";
char msg[50];

/*Recepción del mensaje*/
void receivedCallback( uint32_t from, String &msg ) {
 /*int rssi= WiFi.RSSI();*/
  /*Estructura de la recepción*/
  /*Tiempo de la recepción*/
  int timeRX = malla.getNodeTime();
  /*Nombre del nodo receptor*/
  Serial.print(idNodo);
  Serial.print(", ");
  /*Tiempo de recepción*/
  Serial.print(timeRX);
  Serial.print(", ");
  /*Mensaje recibido*/
  Serial.println(msg);
}
```

```
/*Método conexión del nodo*/
void newConnectionCallback(uint32_t idNodo) {
    Serial.print(idNodo);
    Serial.print(", ");
    Serial.println("Nueva Conexion");

}
/* Cambio de conexión*/
void changedConnectionCallback() {
  Serial.printf("Cambio de Conexion\n");
  connectionMap=malla.subConnectionJson();
  char msg[connectionMap.length()];
  connectionMap.toCharArray(msg, connectionMap.length());
  Serial.printf("Conexion en la red = %s\n", msg);;
}

/*Sinconización del reloj de la red*/
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Tiempo Ajustado %u. Offset = %d\n",
malla.getNodeTime(),offset);
}
void setup() {
  /*Velocidad de comunicación serial*/
  Serial.begin(115200);
  malla.setDebugMsgTypes( ERROR | STARTUP | MESH_STATUS );
  malla.init( MESH_PREFIX, MESH_PASSWORD, &usuarioPlanificador, MESH_PORT
);

  /*Id de callback declarados anteriormente*/
  malla.onReceive(&receivedCallback);
  malla.onNewConnection(&newConnectionCallback);
  malla.onChangedConnections(&changedConnectionCallback);

  /*Se habilita la tarea*/
}
void loop() {
  /*Se ejecuta varias tareas de mantenimiento y actualización de la red*/
  usuarioPlanificador.execute();
  malla.update();
}
```

ANEXO C

Se incluyen todos los datos obtenidos de las pruebas realizadas según los diferentes escenarios con los cálculos.

Este anexo se encuentra en el cd adjunto.

ORDEN DE EMPASTADO