

**UNIVERSIDAD SIMÓN BOLÍVAR  
DECANATO DE ESTUDIOS PROFESIONALES  
COORDINACIÓN DE INGENIERÍA Y TECNOLOGÍA ELECTRÓNICA**

**ESTUDIO E IMPLEMENTACIÓN DE TOPOLOGÍAS MESH CON BASE EN EL  
MÓDULO ESP8266 COMO SOLUCIÓN DE RED EN INTERNET DE LAS COSAS**

Por:  
Luis Alejandro Pérez Bustos  
carnet 10-10545

**PROYECTO DE GRADO**  
Presentado ante la Ilustre Universidad Simón Bolívar  
como requisito parcial para optar al título de  
Ingeniero Electrónico

**Sartenejas, Abril de 2016**



**UNIVERSIDAD SIMÓN BOLÍVAR  
DECANATO DE ESTUDIOS PROFESIONALES  
COORDINACIÓN DE INGENIERÍA Y TECNOLOGÍA ELECTRÓNICA**

**ESTUDIO E IMPLEMENTACIÓN DE TOPOLOGÍAS MESH CON BASE EN EL  
MÓDULO ESP8266 COMO SOLUCIÓN DE RED EN INTERNET DE LAS COSAS**

Por:  
Luis Alejandro Pérez Bustos  
carnet 10-10545

Realizado con la asesoría de:  
Prof. Miguel Díaz

**PROYECTO DE GRADO**  
Presentado ante la Ilustre Universidad Simón Bolívar  
como requisito parcial para optar al título de  
Ingeniero Electrónico

**Sartenejas, Abril de 2016**

# Acta de Evaluación

**UNIVERSIDAD SIMÓN BOLÍVAR**  
**DECANATO DE ESTUDIOS PROFESIONALES**  
**COORDINACIÓN DE INGENIERÍA Y TECNOLOGÍA ELECTRÓNICA**

**ESTUDIO E IMPLEMENTACIÓN DE TOPOLOGÍAS MESH CON BASE EN EL  
MÓDULO ESP8266 COMO SOLUCIÓN DE RED EN INTERNET DE LAS COSAS**

**PROYECTO DE GRADO**

Por: Luis Alejandro Pérez Bustos

Realizado con la asesoría de: Prof. Miguel Díaz

**RESUMEN**

La ubiquidad de las conexiones inalámbricas y la penetración que han tenido los dispositivos inteligentes en la sociedad ha generado el desarrollo de nuevas tendencias. *Internet de las Cosas* define un ecosistema sobre el cual se añade conectividad a dispositivos que anteriormente no disponían de estas capacidades, facilitando el control y automatización de procesos cotidianos. Sensores, procesamiento, comunicación, almacenamiento y visualización son parámetros esenciales en IoT (del inglés *Internet of Things*). El presente trabajo de grado explora la posibilidad de implementación de topologías malladas de fácil despliegue, que sirva como red de transporte en entornos como *Hogares Inteligentes* y *Energy Grids*. Se selecciona el protocolo IEEE 802.11 como estándar de radiocomunicación, su amplia popularidad facilita su uso en entornos urbanos e industriales. Se hace uso del ESP8266, sistema embebido de bajo costo manufacturado por *Espressif Systems*, compatible con el estándar Wi-Fi y la pila de protocolos TCP/IP, con lo cual se disponen de un conjunto de API's suministradas por el fabricante para acceder a las funcionalidades de hardware. Se establece una estructura de varios niveles donde un dispositivo adquiere el rol de nodo raíz, enlace entre la red mallada y el router AP que actúa como puerta de enlace hacia redes externas. Se consideran aplicaciones de poca movilidad y bajas tasas bits como foco de este proyecto. Se utilizan 4 módulos para implementar una red de sensores donde cada nodo emula valores de temperatura generados en intervalos de tiempo preestablecidos. Se utiliza la plataforma cloud *ThingSpeak* para almacenar y visualizar estos datos.

**Palabras clave:** IoT, 802.11, WiFi, sensores, redes, mesh, ESP8266, NodeMCU, ThingSpeak.

Gracias por mantenerme a flote en los momentos más difíciles,  
Elizabeth, Edgar Julián, Edgar Augusto y Genesis,  
esto es para ustedes...

## **AGRADECIMIENTOS**

And I would like to acknowledge ...

## ÍNDICE GENERAL

ÍNDICE DE TABLAS	x
ÍNDICE DE FIGURAS	xi
LISTA DE ABREVIATURAS	xiii
INTRODUCCIÓN	1
<b>1 MARCO TEÓRICO</b>	<b>4</b>
1.1 Topologías de Red . . . . .	5
1.1.1 Topología tipo Bus . . . . .	5
1.1.2 Topología tipo Anillo . . . . .	5
1.1.3 Topología tipo Estrella . . . . .	5
1.1.4 Topología tipo Malla . . . . .	6
1.1.4.1 Protocolos de enrutamiento para redes <i>Mesh</i> . . . . .	7
1.1.4.2 ¿Por qué escoger <i>Mesh</i> para <i>IoT</i> ? . . . . .	8
1.2 IEEE 802.11 . . . . .	8
1.2.1 Pila de Protocolos . . . . .	8
1.2.2 Elementos de la red . . . . .	9
1.2.2.1 Estaciones . . . . .	9
1.2.2.2 Puntos de Acceso . . . . .	9
1.2.3 Arquitecturas de red . . . . .	9
1.2.3.1 Conjunto de Servicios Básicos . . . . .	9
1.2.3.2 Conjunto de Servicios Básicos Independientes . . . . .	10
1.2.3.3 Conjunto de Servicio Extendidos . . . . .	10
1.2.3.4 Conjunto de Servicios Básicos Personales . . . . .	10
1.2.4 Versiones de 802.11 . . . . .	10
1.2.5 Especificaciones a nivel de capa MAC . . . . .	11
1.2.5.1 Espaciado entre Tramas . . . . .	12
1.2.5.2 Acceso Múltiple por Detección de Portadora . . . . .	12
1.2.5.3 Formato de las Tramas . . . . .	13
1.2.6 Otros protocolos de RF para <i>IoT</i> . . . . .	14
1.2.6.1 <i>Bluetooth Smart</i> . . . . .	14
1.2.6.2 <i>ZigBee</i> . . . . .	14
1.2.6.3 Redes de Área Extendidas de Baja Potencia . . . . .	15
1.2.7 ¿Por qué elegir <i>Wi-Fi</i> para <i>IoT</i> ? . . . . .	15

1.3	<i>Wi-Fi</i> y compatibilidad nativa con redes <i>Mesh</i> : IEEE 802.11s . . . . .	16
1.3.1	Enrutamiento . . . . .	17
1.3.2	Formato de tramas . . . . .	18
1.4	Proyecto de red <i>Mesh</i> en el campus de la Universidad Nacional de Singapur . . . . .	20
<b>2</b>	<b>MÓDULO ESP8266 DE ESPRESSIF SYSTEMS</b>	<b>22</b>
2.1	Especificaciones generales . . . . .	22
2.1.1	Especificaciones de radio . . . . .	25
2.1.2	Especificaciones de potencia . . . . .	25
2.1.2.1	Modos de ahorro de energía . . . . .	26
2.2	Procesamiento . . . . .	27
2.2.1	CPU . . . . .	27
2.2.2	Memoria . . . . .	27
2.2.3	Flash . . . . .	27
2.3	Modelos disponibles . . . . .	28
2.3.1	esp8266 de <i>NodeMCU</i> . . . . .	28
2.4	Programación . . . . .	29
2.4.1	Comandos AT . . . . .	29
2.4.2	Arduino IDE . . . . .	30
2.4.3	Lua . . . . .	30
2.4.4	Espressif SDK RTOS . . . . .	31
2.4.5	Espressif SDK NON-OS . . . . .	32
2.4.5.1	crosstool-NG . . . . .	33
2.4.5.2	esptool.py . . . . .	33
<b>3</b>	<b>ESPRESSIF-MESH</b>	<b>35</b>
3.1	Mesh API's . . . . .	35
3.1.1	Limitaciones . . . . .	36
3.2	Especificaciones generales . . . . .	36
3.3	Configuración inicial . . . . .	37
3.4	Topología . . . . .	38
3.4.1	Nodo raíz . . . . .	39
3.4.2	Otros nodos de la red . . . . .	41
3.4.3	Limitaciones de memoria . . . . .	42
3.5	Enrutamiento . . . . .	43
3.6	Modos de funcionamiento . . . . .	45
3.6.1	MESH_ONLINE . . . . .	45
3.6.2	MESH_LOCAL . . . . .	46
3.7	Estructura del encabezado <i>Mesh</i> . . . . .	46
3.7.1	Opciones del paquete <i>Mesh</i> . . . . .	49

3.8	Protocolos compatibles a nivel de capa de aplicación . . . . .	52
3.9	Tipos de Mensajes . . . . .	53
3.9.1	Ejemplo de paquete de datos <i>unicast</i> . . . . .	53
3.9.2	Ejemplo de paquete de datos <i>p2p</i> . . . . .	54
3.9.3	Ejemplo de paquete de datos <i>broadcast</i> . . . . .	54
3.9.4	Ejemplo de paquete de datos <i>multicast</i> . . . . .	55
3.10	Resultados . . . . .	56
3.10.1	Mensajes de control a través de UART . . . . .	57
3.10.2	Ánálisis del la banda ISM de 2,4GHz con <i>Acrylic Wi-Fi</i> . . . . .	60
<b>4</b>	<b>IMPLEMENTACIÓN DE RED DE SENsoRES BASADA EN ESP-MESH</b>	<b>63</b>
4.1	Criterios de diseño . . . . .	63
4.2	Configuración inicial . . . . .	63
4.3	Interfaz con Sensores . . . . .	64
4.4	Modo <i>Mesh Online</i> con servidor local . . . . .	64
4.5	Modo <i>Mesh Online</i> con plataforma <i>Cloud</i> . . . . .	64
4.5.1	Acerca de <i>ThingSpeak</i> . . . . .	64
4.5.1.1	MQTT API's . . . . .	65
4.5.2	Recolección y análisis de datos a través de <i>ThingSpeak</i> . . . . .	66
<b>5</b>	<b>FUTUROS DESARROLLOS</b>	<b>67</b>
5.1	Redes <i>Mesh</i> basadas en el módulo nrf24l01 . . . . .	67
5.2	Redes <i>Mesh</i> basadas en <i>BLE</i> . . . . .	67
5.3	Redes <i>Mesh</i> basadas en dispositivos multi-protocolos . . . . .	68
5.4	Protocolos de enrutamiento para redes ad hoc con eficientes consideraciones de potencia . . . . .	68
5.5	Protocolos de enrutamiento para redes ad hoc a través de algoritmos de <i>Machine Learning</i> . . . . .	68
<b>CONCLUSIONES Y RECOMENDACIONES</b>		<b>69</b>
<b>REFERENCIAS</b>		<b>70</b>
<b>APÉNDICE A</b>		<b>73</b>
<b>APÉNDICE B</b>		<b>83</b>
<b>APÉNDICE C</b>		<b>85</b>

## **ÍNDICE DE TABLAS**

1.1	Versiones de IEEE 802.11 . . . . .	11
2.1	Potencia de Transmisión y Sensibilidad . . . . .	25
2.2	Ubicación de archivos binarios para grabar en la memoria del ESP8266 . . . . .	34
4.1	Posibles configuración en clientes MQTT . . . . .	65

## ÍNDICE DE FIGURAS

1	Evolución en la cantidad de dispositivos conectados hacia Internet . . . . .	2
1.1	Aplicaciones típicas bajo el ecosistema <i>IoT</i> [3] . . . . .	4
1.2	Topologías de red . . . . .	5
1.3	Implementación de redes <i>mesh</i> en ambientes rurales . . . . .	6
1.4	Implementación de redes <i>mesh</i> en situaciones militares . . . . .	7
1.5	Pila de protocolos en IEEE 802.11 . . . . .	9
1.6	Arquitecturas de redes sobre 802.11 . . . . .	10
1.7	<i>Smart Home</i> basado en 802.11 [12] . . . . .	11
1.8	Capa MAC común en diferentes estándares de 802.11 . . . . .	12
1.9	Funcionamiento de CSMA-CA en 802.11 . . . . .	13
1.10	Encabezado MAC en 802.11 . . . . .	14
1.11	Arquitectura de red en 802.11s . . . . .	16
1.12	Dispositivos presentes en una red compatible con IEEE 802.11s . . . . .	17
1.13	Selección de rutas en 802.11s . . . . .	18
1.14	Composición de tramas en 802.11s [18] . . . . .	19
1.15	Direccionamiento en 802.11s . . . . .	19
1.16	Red de distribución heterogénea con compatibilidad <i>Mesh</i> [9] . . . . .	20
1.17	Proyecto de red <i>Mesh</i> en el campus de la Universidad Nacional de Singapur . . . . .	21
2.1	Diagrama de bloques funcionales del ESP8266 [21] . . . . .	23
2.2	Diagrama de flujo entre estados de ahorro de energía . . . . .	27
2.3	Versiones comerciales del <i>ESP8266</i> . . . . .	28
2.4	Versión comercial <i>ESP-12</i> . . . . .	28
2.5	Diagrama de pines del ESP8266 de <i>NodeMCU</i> [30] . . . . .	29
3.1	Topología de la red <i>Mesh</i> . . . . .	37
3.2	Red <i>Mesh</i> multi-canal . . . . .	39
3.3	Selección del nodo raíz . . . . .	40
3.4	Adquisición de topología en el nodo raíz . . . . .	41
3.5	Adquisición de topología en otros nodos de la red . . . . .	42
3.6	Enrutamiento a través de la red . . . . .	43
3.7	Topología <i>Mesh</i> demostrativa . . . . .	45
3.8	Pila de protocolos en el ESP8266 con conectividad <i>Mesh</i> . . . . .	46
3.9	Estructura del encabezado <i>Mesh</i> [38] . . . . .	47
3.10	Estructura de tramas dependiendo del tipo de <i>opción</i> en el encabezado <i>Mesh</i> . . . . .	50
3.11	Estructura del campo de <i>opciones</i> en el encabezado <i>Mesh</i> . . . . .	51

3.12 Creación de mensaje <i>Unicast</i> . . . . .	53
3.13 Creación de mensaje <i>P2P</i> . . . . .	54
3.14 Creación de mensaje <i>Broadcast</i> . . . . .	55
3.15 Creación de mensaje <i>Multicast</i> . . . . .	55
3.16 Módulos ESP8266 de NodeMCU utilizados para la implementación de la red . . . . .	56
3.17 Redes Wi-Fi disponibles en la banda ISM de 2,4 GHz . . . . .	62
A.1 Diagrama de pines del encapsulado ESP8266 [21] . . . . .	83
A.2 Especificaciones de tamaño del encapsulado del ESP8266 [21] . . . . .	83
A.3 Esquemático del módulo de desarrollo fabricado por NodeMCU [30] . . . . .	84

## LISTA DE ABREVIATURAS

<i>ADC</i>	Analog-to-Digital Converter Conversor Digital-Analógico
<i>AES</i>	Advanced Encryption Standard Estándar de Encriptación Avanzado
<i>AHB</i>	Advanced High-Performance Bus Canal de Alto-Desempeño Avanzado
<i>AODV</i>	Ad Hoc On-Demand Distance Vector Vector Distancia por-demanda para redes Ad Hoc
<i>AP</i>	Access Point Punto de Acceso
<i>API</i>	Application Program Interface Interfaz de Programación de Aplicaciones
<i>ASCII</i>	American Standard Code for Information Interchange Código Estándar Estadounidense para el Intercambio de Información
<i>BLE</i>	Bluetooth Low Energy Bluetooth de Baja Energía
<i>BSS</i>	Basic Service Set Conjunto de Servicios Básicos
<i>CoAP</i>	Constrained Application Protocol Protocolo de Control de Transmisiones
<i>CSMA – CA</i>	Carrier Sense Multiple Access with Collision Avoidance Acceso Múltiple por Detección de Portadora con Evasión de Colisión
<i>DCF</i>	Distributed Coordination Function Función de Coordinación Distribuida
<i>DHCP</i>	Dynamic Host Configuration Protocol Protocolo de Configuración Dinámica de Host
<i>ESS</i>	Extended Service Set Conjunto de Servicios Extendidos
<i>GPIO</i>	General-Purpose Input/Output Entrada/Salida de Propósitos Generales

<i>HTTP</i>	Hypertext Transfer Protocol Protocolo de Transferencia de Hipertexto
<i>HWMP</i>	Hybrid Wireless Mesh Protocol Protocolo Híbrido Inalámbrico para Redes Malladas
<i>I2C</i>	Inter-Integrated Circuit Circuito Inter-Integrado
<i>I2S</i>	Integrated-IC Sound Sonido-IC integrado
<i>iBSS</i>	Independent Basic Service Set Conjunto de Servicios Básicos Independiente
<i>IDE</i>	Integrated Development Environment Ambiente de Desarrollo Integrado
<i>IFS</i>	Inter-Frame Space Espacio entre Tramas
<i>IoT</i>	Internet of Things Internet de las Cosas
<i>IP</i>	Internet Protocol Protocolo de Internet
<i>ISP</i>	Internet Service Provider Proveedor de Servicios de Internet
<i>JSON</i>	JavaScript Object Notation Notación de Objeto en JavaScript
<i>LAN</i>	Local Area Networks Redes de Área Local
<i>LPWAN</i>	Low Power Wide Area Networks Redes Extendidas de Baja Potencia
<i>M2M</i>	Machine-To-Machine Máquina-a-Máquina
<i>MAC</i>	Medium Access Control Control de Acceso al Medio

<i>MQTT</i>	Message Queuing Telemetry Transport Transporte de Telemetría de Cola de Mensaje
<i>NIC</i>	Network Interface Card Tarjeta de Interfaz de Red
<i>NS3</i>	Network Simulator 3 Simulador de Red 3
<i>OFDM</i>	Orthogonal-Frequency Division Multiplexing Multiplexado por División Ortogonal de Frecuencia
<i>OLSR</i>	Optimized Link State Routing Protocol Protocolo de Enrutamiento Optimizado por Estado de Enlace
<i>p2p</i>	Peer-to-Peer Par-a-Par
<i>PBSS</i>	Personal Basic Service Set Conjunto de Servicios Básicos Personales
<i>PCF</i>	Point Coordination Function Función de Coordinación Puntual
<i>PLCP</i>	Protocolo de Convergencia de la Capa Física Physical Layer Convergence Protocol
<i>PMD</i>	Physical Medium Dependent Dependiente del Medio Físico
<i>PWM</i>	Pulse Width Modulation Modulación por Ancho de Pulso
<i>QoS</i>	Quality of Service Calidad de Servicio
<i>RF</i>	Radiofrecuencia
<i>RGB</i>	Red-Green-Blue Rojo-Verde-Azul
<i>ROM</i>	Read-Only Memory Memoria Solo-Lectura
<i>RSSI</i>	Received Signal Strength Indication Indicador de Fuerza de la Señal Recibida

<i>RTC</i>	Real-Time-Counter Reloj en Tiempo Real
<i>RTOS</i>	Real-Time Operating System Sistema Operativo en Tiempo Real
<i>SDIO</i>	Secure Digital Input/Output Entrada/Salida de Seguridad Digital
<i>SDK</i>	Software Development Kit Conjunto de Desarrollo de Software
<i>SoC</i>	System on Chip Sistema en Chip
<i>SoftAP</i>	Software Access Point Punto de Acceso por Software
<i>SPI</i>	Serial Peripheral Interface Interfaz de Periféricos Serial
<i>SRAM</i>	Static Random-Access Memory Memoria de Acceso Aleatorio Estático
<i>SSID</i>	Service Set Identifier Identificador de Conjunto de Servicios
<i>STA</i>	Station Estación
<i>TCP</i>	Transmission Control Protocol Protocolo de Control de Transmisiones
<i>TKIP</i>	Temporal Key Integrity Protocol Protocolo de Integridad de Clave Temporal
<i>UART</i>	Universal Asynchronous Receiver/Transmitter Receptor/Transmisor Asíncrono Universal
<i>UDP</i>	User Datagram Protocol Protocolo de Datagramas de Usuario
<i>WEP</i>	Wired Equivalent Privacy Privacidad Equivalente al Cableado

<i>WLAN</i>	Wireless Local Area Networks Redes Inalámbricas de Área Local
<i>WMN</i>	Wireless Mesh Networks Redes Malladas Inalámbricas
<i>WPA</i>	Wi-Fi Protected Access Acceso Inalámbrico Protegido

## INTRODUCCIÓN

Las últimas cuatro décadas han generado desarrollados tecnológicos sin precedentes, creando importantes tendencias que moldean el comportamiento e impulsan la economía global. En años recientes, debido a la ubiquidad de las conexiones inalámbricas, menores costos por servicio, aumento significativo de plataformas web y dispositivos *inteligentes*, se desarrolla el movimiento del Internet de las Cosas (se utilizará la abreviación *IoT* proveniente del inglés *Internet of Things* a lo largo del documento). En *IoT*, los desarrollos se concentran en diversas áreas, destacándose actualmente las que involucran *Hogares Inteligentes*, *Ciudades Inteligentes*, *Vehículos Conectados* y *Redes de Energía* o *Energy Grids*. A pesar de ser campos aparentemente heterogéneos, el ecosistema *IoT* involucra aspectos comunes entre sí con los cuales se generan plataformas que soportan un gran número de aplicaciones. Estos aspectos generales se resumen en la siguiente lista.

- Sensores y/o actuadores: dependen de las funcionalidades necesarias.
- Sistemas embebidos: dependen de las funcionalidades y capacidades necesarias.
- Protocolos de comunicación: típicamente a través de RF (del castellano *radiofrecuencia*) abarcando estándares como Wi-Fi, Bluetooth, ZigBee, LoRa, SigFox y sistemas celulares 3G/4G.
- Topologías de red: depende del protocolo de radiocomunicación utilizado.
- Protocolos de transporte: típicamente UDP (del inglés *User Datagram Protocol*) o TCP (del inglés *Transmission Control Protocol*).
- Protocolos a nivel de capa de aplicación: típicamente HTTP (del inglés *Hypertext Transfer Protocol*), JSON (del inglés *JavaScript Object Notation*), MQTT (del inglés *Message Queuing Telemetry Transport*) y CoAp (del inglés *Constrained Application Protocol*).
- Almacenamiento, organización y aprovechamiento de los datos: plataformas de computación en la nube y algoritmos de *Big Data*.
- Interfaces de usuario: plataformas web, aplicaciones móviles y/o asistentes virtuales.

Para ofrecer conectividad a los servicios antes mencionados, el número de dispositivos hacia Internet ha aumentado de manera exponencial. Según el portal estadístico [www.statista.com](http://www.statista.com) [1] para el año 2020 se alcanzará la cifra de 50.1 mil millones de dispositivos, unas siete veces el estimado de la población mundial para el momento, muchos de ellos asociados a *IoT*. En la figura 1 se visualiza el crecimiento sostenido.

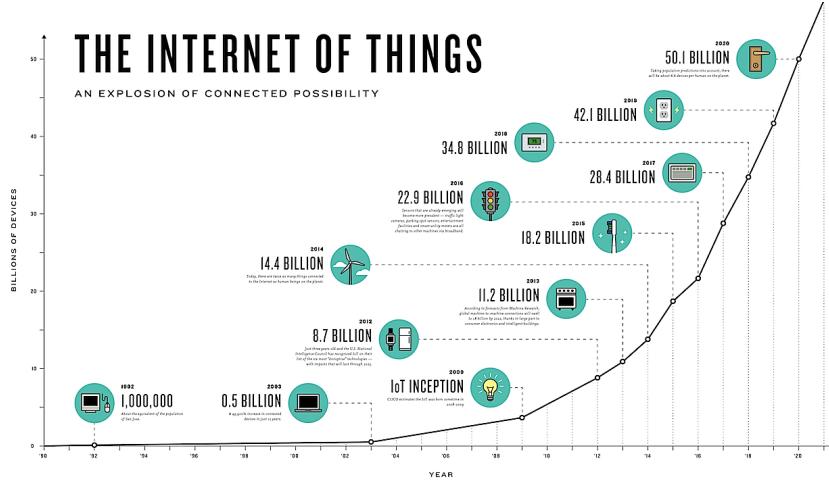


Figura 1: Evolución en la cantidad de dispositivos conectados hacia Internet

Para soportar esta demanda sin aumentar significativamente los costos, es necesario desarrollar nuevas metodologías y mejorar los estándares utilizados a lo largo del ecosistema. La oferta de sistemas embebidos y protocolos de radiocomunicación disponibles [2] es cada vez mayor. Por ello se hace imperativo escoger adecuadamente los componentes necesarios en el momento de diseño.

Gran número de compañías están en proceso de descubrir el potencial de mercado de *IoT* más allá de las ventajas aparentes en reducción de costos y aumento de la productividad que la automatización ofrece. La habilidad para recolectar y analizar datos, con la posibilidad de generar respuestas en tiempo real sobre un entorno, podría cambiar la forma como se hacen negocios en la actualidad. Además, la información digital ha adquirido valor importante para desarrollar innovadoras estrategias de mercado en los últimos años e importantes empresas lo saben. Recolectar datos de los usuarios se ha convertido en un modelo de negocios bastante rentable y de acelerado crecimiento.

El presente trabajo de grado explora la posibilidad de implementar nuevas topologías en redes Wi-Fi utilizando el módulo de bajo costo ESP8266 manufacturado por la compañía *Espressif Systems*, con la finalidad de desarrollar aplicaciones para *entornos inteligentes* compatibles con el ecosistema *IoT*. Se hace especial énfasis en topologías *malladas* (se utilizará el término en inglés *Mesh* de ahora en adelante), lo que hace fundamental establecer adecuados protocolos de enruteamiento que garanticen conectividad con cada punto de la red. Es necesario además, adaptarse a los recursos limitados de procesamiento, memoria y energía que presenta un nodo típico para *IoT*.

### **Objetivo General:**

- Implementar una red mallada basada en los estándares IEEE 802.11 a través del módulo ESP8266.

**Objetivos Específicos:**

- Estudiar los fundamentos del estándar Wi-Fi bajo la serie de protocolos 802.11 de la IEEE.
- Estudiar estructura y principios de redes malladas.
- Analizar las características del módulo ESP8266.
- Verificar conectividad Wi-Fi entre dos módulos ESP8266 siguiendo topologías tradicionales.
- Establecer criterios de enrutamiento a utilizar.
- Verificar algoritmos de enrutamiento.
- Realizar pruebas de comunicación entre nodos de la red.
- Elaboración de informe final de proyecto de grado.

# CAPÍTULO 1

## MARCO TEÓRICO

El auténtico pensar, como el vivir, es tarea que se aprende. Quien aprende, debe dialogar con aquellos que han pensado previamente: recorrer sus itinerarios, seguir sus rutas y estelas, consultar sus bitácoras

---

*Ernesto Mayz Vallenilla*  
Rector fundador de la USB

Para diseñar aplicaciones basadas en el ecosistema *IoT* (del inglés *Internet of Things*) es necesario establecer adecuados criterios de diseño que cumplan con los requerimientos necesarios sin exceder los recursos disponibles. De esta manera, el problema de conectividad ha sido abordado de distintas maneras dependiendo del fabricante. El contenido del actual capítulo se centra en explicar dos aspectos fundamentales de la conectividad de cualquier dispositivo: el protocolo de comunicación utilizado y la topología de red en la cual se encuentra inmerso compartiendo un mismo medio con múltiples nodos. Ambas características se encuentran fuertemente entrelazadas, ciertos estándares de radiocomunicación utilizan tradicionalmente topologías específicas y por esta razón los hace más conveniente para uno u otro tipo de aplicación. La figura 1.1 muestra un amplio rango de estas.

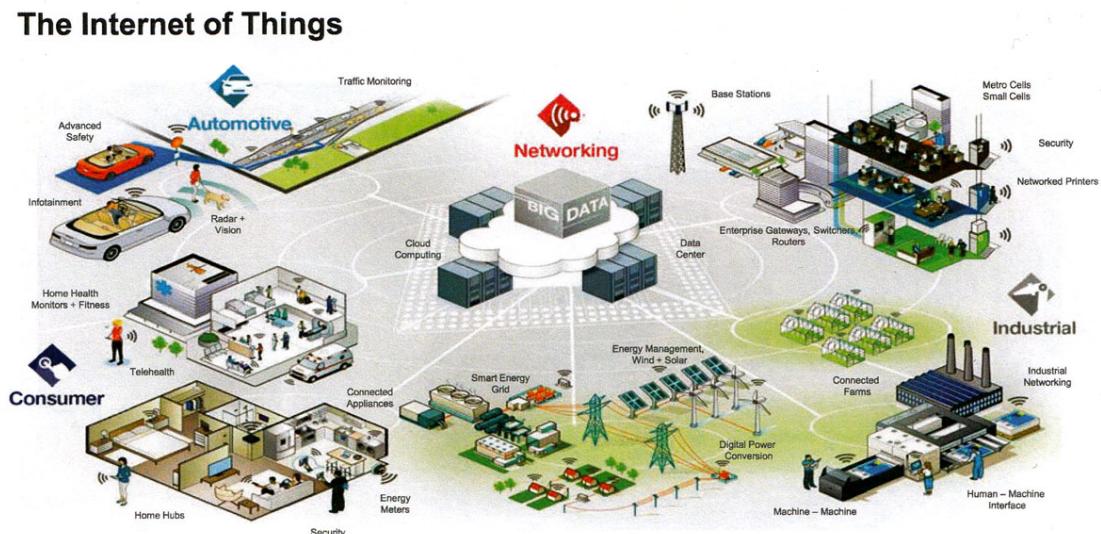


Figura 1.1: Aplicaciones típicas bajo el ecosistema *IoT* [3]

## 1.1. Topologías de Red

Una topología de red es el patrón que describe un arreglo de nodos dentro de una red de computadoras. El medio en el cual se lleva a cabo la comunicación, que define el tipo de enlace entre cada punto contiguo de la red, es independiente de la topología escogida. Existen esencialmente 4 tipos de estructuras [4]: bus, anillo, estrella y malla. Se describen cada uno a continuación y se argumenta la utilización de esta última para el tipo de aplicación a desarrollar en el presente trabajo de grado.

### 1.1.1. Topología tipo Bus

La esquina inferior derecha de la figura 1.2 muestra el patrón que sigue una red tipo bus. Cada nodo está conectado directamente al mismo medio de transporte sin poseer caminos alternativos. Cualquier falla en el sistema va a evitar que se tenga acceso al mismo. Sin embargo, su fácil instalación y pocos recursos necesarios representan ventajas con respecto a otras topologías.

### 1.1.2. Topología tipo Anillo

Se representa en la esquina superior izquierda de la figura 1.2. Los paquetes de datos viajan en forma circular por cada nodo de la red hasta llegar a su destino. Al igual que las topologías tipo bus, son de fácil instalación. Sin embargo, al ser unidireccional y fallar algún segmento, se detiene su funcionamiento.

### 1.1.3. Topología tipo Estrella

Es la topología más utilizada y se representa en la esquina superior derecha de la figura 1.2. Un dispositivo central controla el flujo de paquetes a través de la red. Es la topología tradicional en redes inalámbricas basadas en la serie de protocolos IEEE 802.11, donde el aire representa el medio común para todos los nodos. Además, es de fácil escalabilidad en comparación a las estructuras explicadas anteriormente.

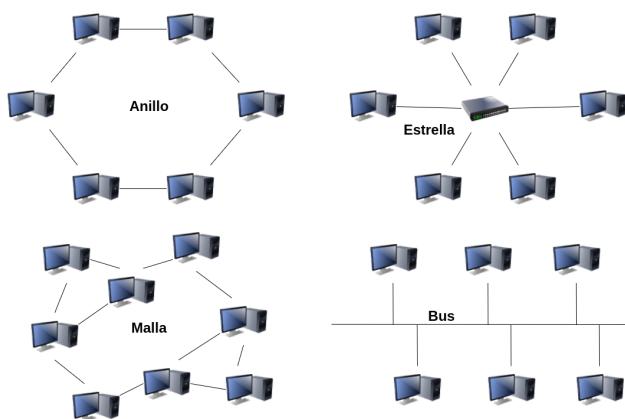


Figura 1.2: Topologías de red

#### 1.1.4. Topología tipo Malla

Las topologías malladas (se utilizará el término en inglés *Mesh* a lo largo del documento) representan un paradigma de comunicación manejado desde hace bastante tiempo en el área de redes, sin embargo no se disponían del tipo de aplicaciones que requieran este tipo de conexiones. El patrón que describen se observa en la esquina inferior izquierda de la figura 1.2. Los enlaces son generados tomando en cuenta métricas establecidas con anterioridad lo que provee a cada nodo de la capacidad de configurarse y re-configurarse dinámicamente para asegurar conectividad. De esta manera, los paquetes realizan múltiples saltos hasta llegar a su destino. La gran cantidad de uniones hace que sean inviables para entornos cableados pero convenientes en aplicaciones inalámbricas.

Las redes *Mesh* han surgido como un concepto prometedor para hacer frente a los retos de la nueva generación de redes al proveer de arquitecturas flexibles y adaptables con un bajo costo de implementación, pero con alto nivel de complejidad. Este tipo de topologías funcionan de manera *standalone* al suministrar rutas de datos entre nodos pares o además ofreciendo conectividad hacia infraestructuras fijas como la de los ISP's (del inglés Internet Service Providers) a través de puntos específicos que actúan como puertas de acceso.

Según lo descrito en [5], algunas aplicaciones típicas de las redes *Mesh* se citan a continuación:

- Dispositivos del hogar: ofreciendo conectividad a un creciente número de dispositivos.
- Municipalidades: incluyendo zonas comerciales, residenciales o parques donde el costo de implementación de redes convencionales es alto para suplir la baja demanda. Situaciones como las descritas en [6] sirven de ejemplo para este tipo de aplicaciones (figura 1.3).
- Operaciones militares: desastres naturales, operaciones de rescate o cualquier tipo de situación donde no se dispone de infraestructuras de comunicación estables (figura 1.4).
- Comunicación vehículo-a-vehículo: ofreciendo al conductor información adicional acerca de su entorno y dando paso a la tecnología de comunicación para vehículos autónomos.

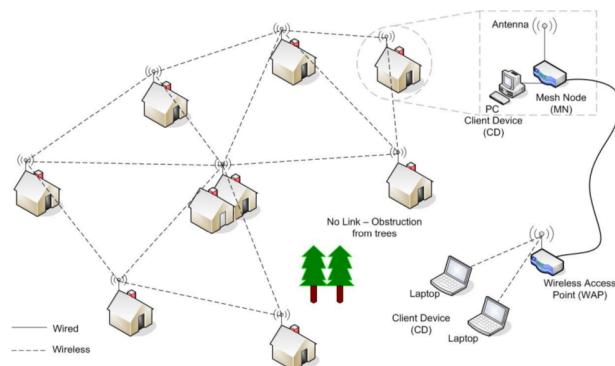


Figura 1.3: Implementación de redes *mesh* en ambientes rurales

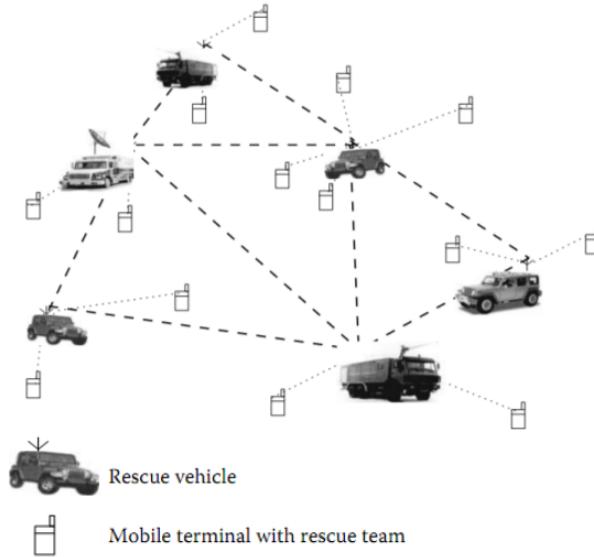


Figura 1.4: Implementación de redes *mesh* en situaciones militares

#### 1.1.4.1. Protocolos de enrutamiento para redes *Mesh*

Se deben cumplir con ciertos criterios para lograr enrutamiento a través de redes *Mesh*. Se han desarrollado protocolos que toman en consideración métricas prestablecidas para definir mejores rutas posibles. Se detallan a continuación dos de los más utilizados.

- **AODV:** del inglés *Ad Hoc On Demand Distance Vector* [7]. Este protocolo implementa mecanismos *por-demanda* para el descubrimiento o mantenimiento de caminos más enrutamiento *salto-a-salto* y mensajes de verificación periódicos. Cuando un nodo *S* necesita una ruta hacia otro nodo *D*, este envía a sus vecinos un *broadcast* denominado *ROUTE REQUEST*. Este mensaje se transmite a través de la red hasta que encuentra otro nodo *P* con una ruta determinada hasta *D*. En ese momento *P* genera un *ROUTE REPLY* que contiene el número de saltos necesarios para alcanzar *D* y el número de secuencia correspondiente. Cada nodo que ha participado en este proceso crea una ruta inversa desde su posición hasta el punto inicial *S* y en la ruta directa hasta *D*, recuerda sólo el siguiente salto necesario. Para mantener estas trayectorias, AODV requiere que cada nodo transmita un mensaje *HELLO* periódico. Al no recibir consecutivamente tres de estos paquetes desde un vecino, el nodo da de baja este enlace y, si es necesario, se comienza un descubrimiento de ruta nuevamente.
- **OLSR:** del inglés *Optimized Link State Routing Protocol* [8]. El protocolo está basado en algoritmos de estado de enlace con naturaleza *proactiva*. Los nodos intercambian constantemente información de la topología de la red, lo que conlleva a disponer de rutas inmediatas a cualquier destino al momento que se necesiten. Debido al comportamiento periódico de estas transmisiones, no es imperativo verificar la recepción de los mensajes. OLSR optimiza el tamaño de los paquetes de control al enviarlos solamente a través de nodos previamente

seleccionados y denominados *Relés Multipunto*. El protocolo es factible mayormente para redes *Mesh* con alta densidad de nodos. Cada uno de estos utiliza la última información disponible para calcular una ruta específica. Esta característica requiere de una mayor capacidad de cómputo para funcionar.

#### **1.1.4.2. ¿Por qué escoger *Mesh* para IoT?**

Debido al crecimiento exponencial de dispositivos conectados se necesita una topología de red que disponga de las capacidades necesarias para proveer conectividad a cada nodo, sin aumentar considerablemente los costos por infraestructura. Topologías convencionales basadas en niveles jerárquicos donde dispositivos centrales controlan el flujo de datos y el acceso al medio no son viables en entornos donde redes *mesh* encuentran los usos principales descritos en anteriores secciones.

Es necesario desarrollar sistemas que utilicen estándares de radiocomunicación disponibles y de amplio uso, que sean compatibles con las funcionalidades de enrutamiento necesarias en redes *mesh*. Para ello, se selecciona el stack de protocolos que ofrece 802.11 de la IEEE.

## **1.2. IEEE 802.11**

Creado en 1997 y bajo la tutela de la *WiFi Alliance* desde 1999, el estándar ha evolucionado para cumplir con una alta demanda de aplicaciones. De esta manera, ha sido imperativo cumplir con la necesidades de tasas de bits más elevadas, mayor eficiencia energética, mejores protocolos de acceso al medio y mejor uso espectral. IEEE 802.11 constituye un conjunto especificaciones a nivel de capa física y subcapa MAC para garantizar conectividad a Redes Inalámbricas de Área Local (WLAN) en las bandas ISM [9], específicamente en las frecuencias de 2,4 GHz, 5 GHz y más recientemente en torno a los 60 GHz. Este tipo de redes tienen características especiales que las hacen diferentes que sus contrapartes cableadas.

### **1.2.1. Pila de Protocolos**

La figura 1.5 muestra el esquema de la pila de protocolos de 802.11 [10]. Además de las funcionalidades típicas de la capa MAC (del inglés *Medium Access Control*), la subcapa PLCP (del inglés *Physical Layer Convergence Protocol*) se encarga del mapeo de los paquetes MAC a paquetes PHY apropiados. Por su parte, la subcapa PMD (del inglés *Physical Medium Dependent*) tiene funciones típicas de modulación, codificación de canal y separación OFDM (del inglés *Orthogonal-Frequency Division Multiplexing*).

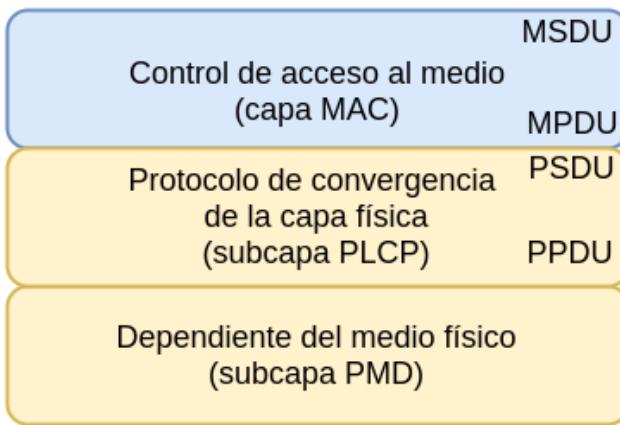


Figura 1.5: Pila de protocolos en IEEE 802.11

### 1.2.2. Elementos de la red

#### 1.2.2.1. Estaciones

Cualquier unidad direccionable en 802.11 es una *estación* (se utilizará el acrónimo *STA* de ahora en adelante y a lo largo del presente trabajo de grado). Puede asumir diferentes roles dependiendo del tipo de dispositivo.

#### 1.2.2.2. Puntos de Acceso

Las *estaciones* que dan acceso y controlan el flujo de datos a un sistema de distribución local se denominan *Puntos de Acceso* (se utilizará el acrónimo *AP* de ahora en adelante y a lo largo del presente trabajo de grado). Un AP tiene funcionalidad de *bridge* (en castellano *Puente*) a nivel de capa 2. Cuando los terminales inalámbricos se conectan a la red a través de un AP, estos se encuentran en la misma LAN (de inglés *Local Area Network*) que los terminales cableados.

Por otro lado, un AP no tiene una dirección ip pública asignada. Un dispositivo central típico en 802.11 tiene funcionalidad de AP, router y switch orientado a mover el tráfico desde la WLAN hacia la WAN a la cual está conectado, comúnmente la red del ISP (del inglés *Internet Service Provider*).

### 1.2.3. Arquitecturas de red

#### 1.2.3.1. Conjunto de Servicios Básicos

Una BSS (del inglés *Basic Service Set*) corresponde a una picocelda en sistemas inalámbricos. Es el bloque fundamental de cualquier WLAN de infraestructura. Cuando un AP define una nueva BSS, asigna un identificador denominado SSID (del inglés *Service Set Identifier*) a través del cual nuevos dispositivos podrán reconocer la red inalámbrica en un cierto rango de cobertura.

### 1.2.3.2. Conjunto de Servicios Básicos Independientes

Una iBSS (del inglés *Independent Basic Service Set*) permite establecer conexión entre dos o más STA de manera independiente, sin necesidad de un AP que regule el tráfico de la red.

### 1.2.3.3. Conjunto de Servicio Extendidos

Una ESS (del inglés *Extended Service Set*) consiste de dos o más BSS interconectadas mediante un sistema de distribución hacia cualquier red de datos. La figura 1.6 define las tres arquitecturas de red mencionadas.

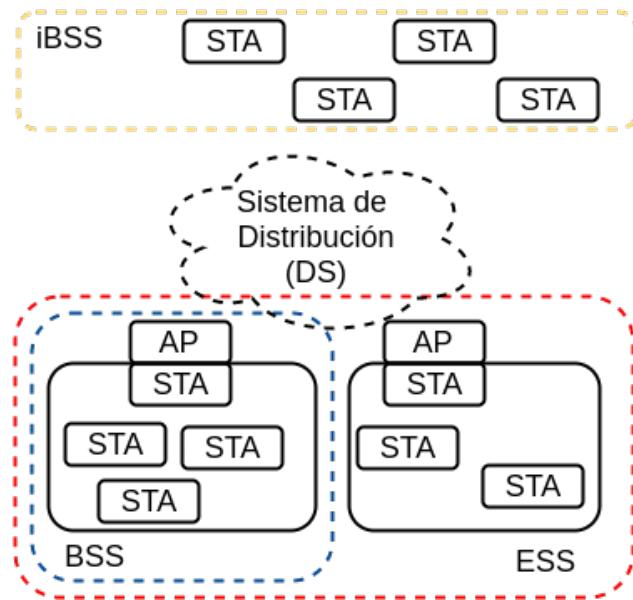


Figura 1.6: Arquitecturas de redes sobre 802.11

### 1.2.3.4. Conjunto de Servicios Básicos Personales

Recientemente se ha introducido la arquitectura PBSS (del inglés *Personal Basic Service Set*) en conjunto con el nuevo protocolo 802.11ad [11] para aplicaciones como almacenamiento y periféricos inalámbricos. Los nodos se comunican como en una iBSS con uno de los nodos asumiendo el rol de *Punto de Control de la PBSS*, anunciando la red y organizando el control de acceso al medio.

## 1.2.4. Versiones de 802.11

El estándar Wi-Fi ha evolucionado continuamente para cumplir con nuevas demandas. En la tabla 1.1 se tienen las especificaciones a nivel de capa física de las versiones comerciales de 802.11. El objetivo de la *Wi-Fi Alliance* es proveer de una plataforma de conectividad completamente basada en su estándar (ver figura 1.7) a través de AP's que se adapten al medio y suministren una u otra versión a cada STA. De esta manera, se garantiza la compatibilidad hacia atrás.

Tabla 1.1: Versiones de IEEE 802.11

Estándar	Banda Frecuencial	Ancho de Banda	Esquema Modulación	Esquema Multiplexación	Max. Tasa de bits	MIMO
<b>b</b>	2,4 GHz	11 MHz	CCK con QPSK	DSSS	11 Mbps	no
<b>a</b>	5 GHz	20 MHz	hasta 64-QAM	OFDM	54 Mbps	no
<b>g</b>	2,4 GHz	20 MHz	hasta 64-QAM	DSSS OFDM	54 Mbps	no
<b>n</b>	2,4 GHz 5 GHz	20 MHz 40 MHz	hasta 64-QAM	OFDM	600 Mbps	SU-MIMO
<b>ac</b>	5 GHz	20 MHz 40 MHz 80 MHz 160 MHz (80+80)	hasta 256-QAM	OFDM	3,47 Gbps	SU-MIMO
<b>ad</b>	60 GHz	2,16 GHz	hasta 64-QAM (OFDM) hasta 16-QAM (SC)	OFDM SC	6,76 Gbps	MU-MIMO

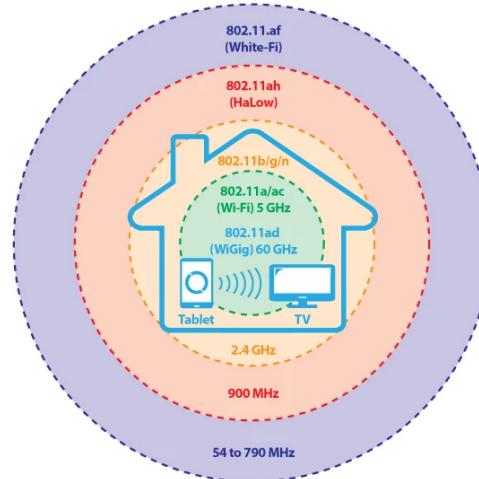


Figura 1.7: Smart Home basado en 802.11 [12]

### 1.2.5. Especificaciones a nivel de capa MAC

La evolución de 802.11 ha consistido en proveer de una capa MAC única, o con ligeros cambios, con numerosas alternativas a nivel de capa física dependiendo de la versión utilizada (ver figura 1.8). La comunicación en los enlaces de subida y bajada se basa en el principio *Duplex por División de Tiempo*. De esta manera, se disponen de dos funciones de coordinación, una opcional y otra obligatoria: PCF (del inglés *Point Coordination Function*) y DCF (del inglés *Distributed Coordination Function*).

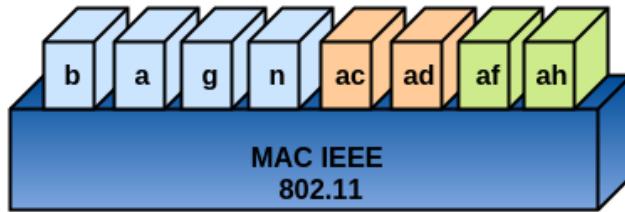


Figura 1.8: Capa MAC común en diferentes estándares de 802.11

### 1.2.5.1. Espaciado entre Tramas

Los IFS (del inglés *Inter-Frame Space*) son tiempos de espera predeterminados por el protocolo de acceso al medio. Son configurables dependiendo del dispositivo y determinan el espaciado entre diferentes tramas. Existen cuatro valores diferentes, en orden de duración:

- Reduced-IFS o RIFS
- Short-IFS o SIFS
- Point Coordination Function-IFS o PIFS
- Distributed Coordination Function-IFS o DIFS
- Arbitration-IFS o AIFS
- Extended-IFS o EIFS

### 1.2.5.2. Acceso Múltiple por Detección de Portadora

Para el caso de DCF, mecanismo de acceso al medio obligatorio en 802.11, se utiliza el protocolo CSMA-CA (del inglés *Carrier Sense Multiple Access - Collision Avoidance*). Cada estación compite con las demás para ocupar el canal durante la transmisión de un paquete de datos. Los terminales se encuentran monitoreando el medio hasta que este se encuentre libre por un tiempo igual a un DIFS o un AIFS en el caso de implementación de parámetros de QoS (del inglés **Quality of Service**). Posteriormente, se genera un número aleatorio que definirá la cantidad de ranuras de tiempo que se deberán esperar para realizar la transmisión (se utilizará el término en inglés *backoff* para hacer referencia a este intervalo). El tiempo por ranura es configurable dependiendo del dispositivo utilizado. Para acceder al medio, la estación debe esperar que este número llegue a cero. La figura 1.9 ilustra este proceso.

Para afrontar colisiones, en [10] se establece:

En cada transmisión, el tiempo de backoff se escoge aleatoriamente con una distribución uniforme en el rango de  $(0, w-1)$ . En el primer intento de transmisión,  $w = W_{\min}$  que es la ventana mínima

de transmisión. Después de cada intento fallido de transmisión,  $w$  se duplica hasta un valor máximo  $W_{max}$ . Para poder determinar si el paquete fue recibido correctamente, la estación de destino envía un paquete de confirmación (ACK) luego de un SIFS después de la recepción del paquete de datos. Si la estación que envió el paquete de datos no detecta un ACK luego de un cierto tiempo de espera o si detecta que el canal está siendo utilizado para otro tipo de transmisión, reintentará enviar el paquete de acuerdo a las reglas de backoff expuestas anteriormente. Este mecanismo básico del modo DCF se denomina MAC convencional.

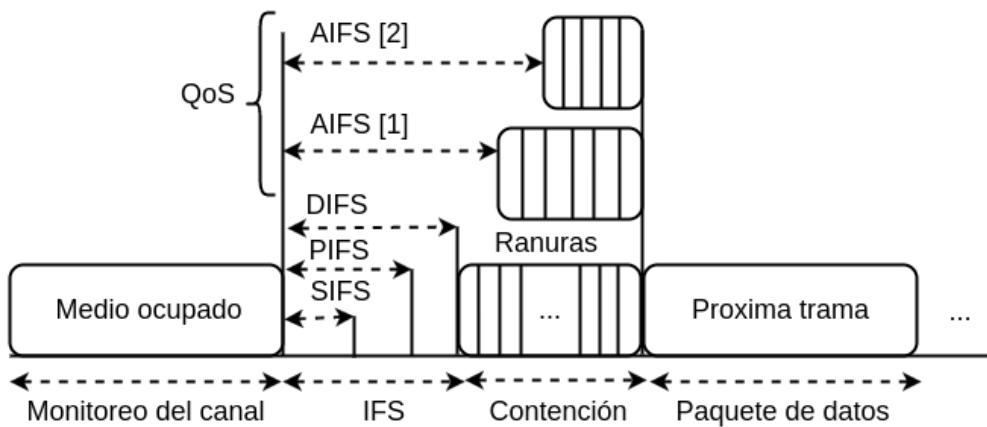


Figura 1.9: Funcionamiento de CSMA-CA en 802.11

### 1.2.5.3. Formato de las Tramas

Existen tres tipos de tramas WLAN:

- **Control:** asisten a las funciones de control de acceso al medio.
- **Gestión:** contienen tráfico del usuario.
- **Datos:** organizan procesos como asociaciones y disociaciones

El encabezado MAC de 802.11 se observa en la figura 1.10. El tamaño máximo del MSDU es de 2312 bytes, mayor que el *payload* tradicional de 1500 Bytes en redes *Ethernet*. Los campos restantes se listan a continuación:

- **Control de Trama:** información relacionada a la trama de datos que será enviada: fragmentación, origen local, origen externo, entre otros.
- **Duración:** En tramas RTS/CTS (del inglés *Request-To-Send/Clear-To-Send*) [9], indica el tiempo en que estará ocupado el canal durante la próxima transmisión.
- **Campos de Dirección:** se agregan cuatro direcciones MAC según el formato: origen-destino-transmisor-receptor. Para casos de múltiples saltos, los campos origen-destino y transmisor-receptor no coinciden, sin embargo todas corresponden a direcciones válidas dentro de la misma WLAN.

- **Secuencia de Control:** dividido en dos partes con una longitud de 16 bits entre ambas. Todos los paquetes tienen un cierto número de secuencia de 12 bits, este es utilizado para generar retransmisiones en caso de ser necesario. Si el paquete de datos, además, ha sido fragmentado, cada uno se puede identificar al combinar un número de secuencia y un número de fragmento de 4 bits.
- **CRC:** cálculo de secuencia de chequeo de la trama con objetivo de detección de errores. Se utiliza un polinomio de grado 32.

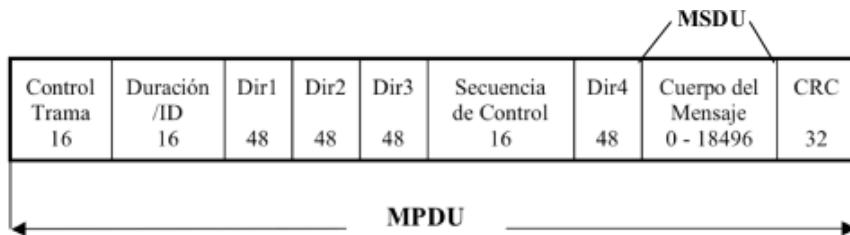


Figura 1.10: Encabezado MAC en 802.11

### 1.2.6. Otros protocolos de RF para IoT

#### 1.2.6.1. Bluetooth Smart

Mejor conocido como BLE (del inglés *Bluetooth-Low-Energy*), es la versión de bajo consumo de energía del popular estándar *Bluetooth*. Este nuevo protocolo fue diseñado especialmente para IoT con foco en periféricos y sensores de baja potencia. Según el fabricante [13]:

La eficiencia en potencia de Bluetooth con funcionalidad de baja energía lo hace perfecto para dispositivos que trabajen durante largos períodos con fuentes de energía como baterías o equipos de auto-generación. Soporte nativo de la tecnología Bluetooth sobre todos los sistemas operativos más importantes habilitan el desarrollo de un amplio rango de dispositivos conectados, desde electrodomésticos y sistemas de seguridad hasta monitores de actividad física y sensores de proximidad.

#### 1.2.6.2. ZigBee

ZigBee es el estándar defacto actualmente para aplicaciones de automatización industrial y del hogar. Soporta nativamente compatibilidad *Mesh* y es diseñado para un bajo consumo energético. Según la *ZigBee Alliance* [14]:

ZigBee Smart Energy es el estándar líder mundial para productos interoperables que monitorean, controlan, informan, y automatizan la entrega y el uso de energía y agua. Es usado para suministrar soluciones innovadoras para sensores inteligentes y redes del hogar que permiten a los consumidores saber y controlar su consumo energético conectándolos a los grids de energía y ayudando a crear casas ecológicas dando a los usuarios la información y automatización necesaria para fácilmente reducir sus consumos y ahorrar dinero.

### 1.2.6.3. Redes de Área Extendidas de Baja Potencia

Las aplicaciones de *IoT* son muy variadas y difieren en sus requerimientos. Normalmente se categorizan dependiendo de su rango, movilidad, consumo de potencia y tasa de bits necesaria. Para aquellas en donde el rango de cobertura es fundamental, redes dedicadas para dar servicios a dispositivos específicos podrían ser una solución de bajo costo y mejor eficiencia energética.

Para ello se disponen de los estándares que siguen la filosofía LPWAN (del inglés *Low Power Wide Area Networks*) que proveen de baja tasa de bits y amplia cobertura en las bandas ISM sub-GHz. Entre estas tecnologías destacan actualmente *LoRa*, *SigFox* e *Ingenu* [15].

### 1.2.7. ¿Por qué elegir *Wi-Fi* para *IoT*?

En una ardua competencia por convertirse en el estándar para desarrollos en *IoT*, gran cantidad de protocolos de radiocomunicaciones intentan dominar un mercado de elevado crecimiento. *Cisco Systems* estima que el mercado global del *Internet de las Cosas* será de \$14,4 trillones en 2022 [16]. La empresa de consultoría *McKinsey & Company* coloca estas estimaciones entre \$3,9 y \$11,1 trillones para 2025 [17]. *Wi-Fi* surge como uno de los estándares de mayor relevancia en el sector debido a su amplia infraestructura ya instalada alrededor del mundo y su completa compatibilidad con la pila de protocolos TCP/IP.

Algunas cifras claves sobre el protocolo se presentan a continuación.

- Habrán más de 7 billones nuevos dispositivos *Wi-Fi* en los próximos 3 años - *SysCon*
- 2/3 de los usuarios de Estados Unidos prefieren utilizar redes *Wi-Fi* que el servicio de datos de su correspondiente operador - *Deloitte*
- En 2015, aproximadamente la mitad de los dispositivos de redes fueron móviles - *Infonetics*
- 71 % de todas las comunicaciones móviles fluyen a través de *Wi-Fi* - *Wi-Fi Alliance*
- 93 % de los dueños de tablets utilizan solamente conectividad a través de *Wi-Fi* - *Deloitte*
- Usuarios de dispositivos móviles revisan sus teléfonos aproximadamente 150 veces al día - *Kleiner Perkins Caufield & Byers's*
- 90 % de las personas tienen un dispositivo móvil al alcance el 100 % del tiempo - *Pew Research Center*
- Hasta 70 % de las personas que entran a una tienda tienen un dispositivo *Wi-Fi* en sus bolsillos - *Retail Touch Points*
- Sólo 2,7 mil millones de personas tenían acceso a internet en 2013. Cerca del 36 % de la población mundial - *StateTech*

Por otro lado, numerosos SoC (del inglés *System on Chip*) integran la pila de protocolos IEEE 802.11, evadiendo la necesidad de módulos externos para ofrecer conectividad a las aplicaciones que albergan. Menores costos por cada uno de estos, hacen que Wi-Fi domine el sector más allá de los teléfonos inteligentes y computadoras personales. En el próximo capítulo se explican a detalle las funcionalidades que ofrece el ESP8266 de *Espressif Systems*.

### 1.3. Wi-Fi y compatibilidad nativa con redes *Mesh*: IEEE 802.11s

Muchos estándares de radiocomunicaciones están desarrollando protocolos que involucren funcionalidad *Mesh*. 802.11s es un esfuerzo de la IEEE que intenta añadir este tipo de conectividad a las Redes Inalámbricas de Área Local. La figura 1.11 muestra la arquitectura de una red que implementa el protocolo mencionado [18].

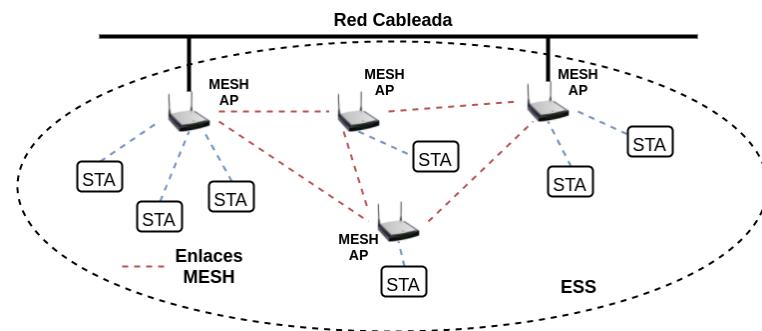


Figura 1.11: Arquitectura de red en 802.11s

El protocolo se implementa a nivel de capa MAC y garantiza compatibilidad con protocolos en capas superiores. Los enlaces son establecidos de forma dinámica y las rutas a través de los algoritmos ya mencionados, permitiendo el transporte del paquete de datos a través de la red en salto único o múltiples saltos. La figura 1.12 muestra los 4 tipos de dispositivos que se pueden encontrar. Se introducen los *Mesh Points* para ofrecer conectividad entre nodos. De esta manera, un *Mesh Access Point* es un elemento que posee enlaces *Mesh* al mismo tiempo que cumple funciones de controlar el tráfico de datos en su propia BSS y enlazar con redes externas.

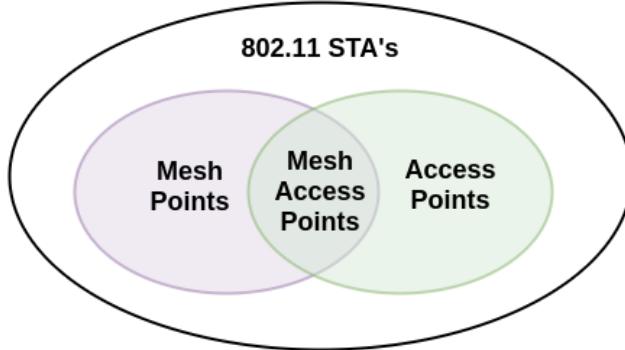


Figura 1.12: Dispositivos presentes en una red compatible con IEEE 802.11s

La caracterización se realiza utilizando un *Mesh ID*, similar al SSID comúnmente utilizado [19]. Esta etiqueta, junto al protocolo de selección de caminos y las métricas correspondientes, identifican la red y definen un perfil. Cuando un *Mesh Point* se introduce en la red, escanea posibles *Mesh ID*'s que posean un perfil compatible. Al conseguirlo, este establece un enlace seguro y se autentica con sus vecinos. Al ocurrir esto, ya es capaz de participar en el proceso selección de rutas y flujo de paquetes de datos.

### 1.3.1. Enrutamiento

El enrutamiento se hace a través del protocolo HWMP (del inglés *Hybrid Wireless Mesh Protocol* o *Protocolo Inalámbrico Híbrido para Redes Malladas* en castellano), que combina la flexibilidad del descubrimiento de ruta *por-demanda* con un eficiente enrutamiento proactivo hacia el gateway (*Portal de acceso* en castellano). Por su parte, la métrica por defecto es el *tiempo en el aire*, pudiendo extenderse para tomar en consideración aspectos como QoS (del inglés *Quality of Service*), balanceo de carga y consumo energético.

- **Enrutamiento reactivo:** buen desempeño en ambientes cambiantes. Se utiliza AODV para descubrir las mejores rutas *on-demand*, sin tener la necesidad de utilizar espacio de la memoria para almacenar caminos que no se encuentren en uso. Cada dispositivo mantiene una secuencia de nodos de destino en las rutas activas para evitar ciclos repetitivos.
- **Enrutamiento proactivo:** eficiente para topologías sin mucha movilidad. Evita el continuo descubrimiento de caminos a través de *flooding* (en castellano *inundación*). Se debe establecer un *Mesh Point* raíz que establezca los caminos hacia los diferentes puntos de la red. Este dispositivo arma su topología a través de mensaje de *broadcast* que responden los correspondiente *Mesh Point's* que se encuentren por debajo. Una vez construidas las *mejores* rutas, el nodo raíz las anuncia a todos los miembros.

En la figura 1.13 se observan las dos aproximaciones. En ambos casos, se hacen uso de 4 primitivas para búsqueda y selección de los *mejores caminos disponibles*:

- **Root Announcement (broadcast)**: anuncia a los *Mesh Point's* sobre la presencia de un *Mesh Point* raíz.
- **Route Request (broadcast/unicast)**: pregunta por ruta hacia el *Mesh Point* de destino.
- **Route Reply (unicast)**: respuesta ante un *Route Request* luego de haber encontrado el *Mesh Point* de interés.
- **Route Error (boradcast)**: indica a los *Mesh Points* que el origen no soporta nuevamente ciertas rutas ya establecidas.

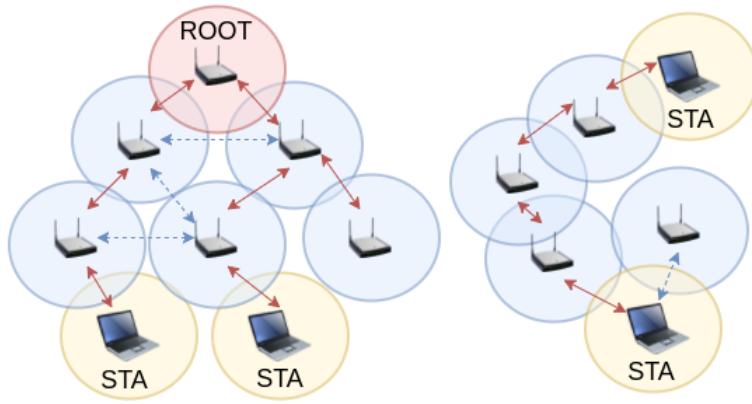


Figura 1.13: Selección de rutas en 802.11s

### 1.3.2. Formato de tramas

Las tramas de 802.11s son muy similares a las tradicionales, se añade un *encabezado mesh* de tamaño variable de 4 a 6 bytes que contiene los elementos presentes en la figura 1.14. Se incluye el valor TTL (del inglés *Time To Live*) para limitar el número de saltos y el número de secuencia ya mencionado. Se observan además dos direcciones adicionales a las presentes en los encabezados MAC de 802.11. Estas sirven para proveer comunicación punto-a-punto entre dos nodos de diferentes redes. La figura 1.15 ilustra este proceso.

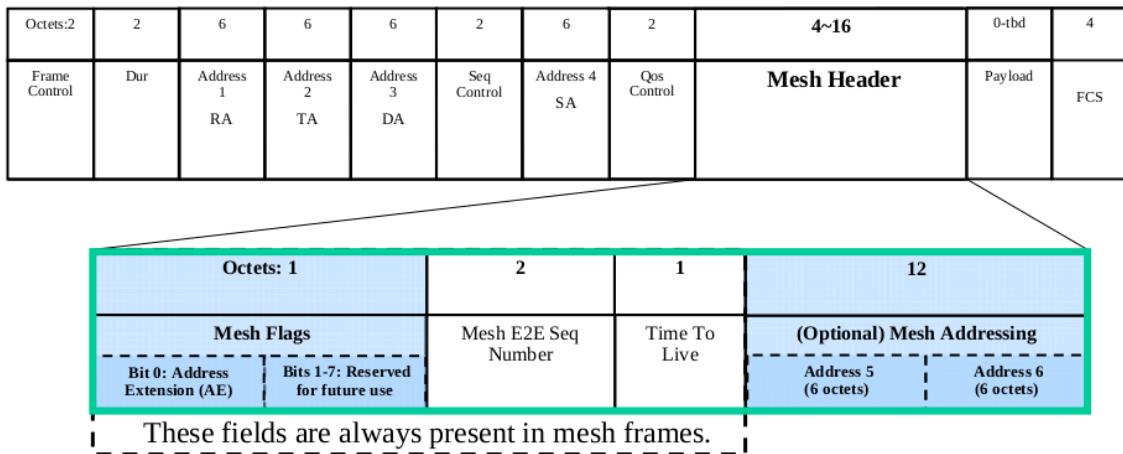


Figura 1.14: Composición de tramas en 802.11s [18]

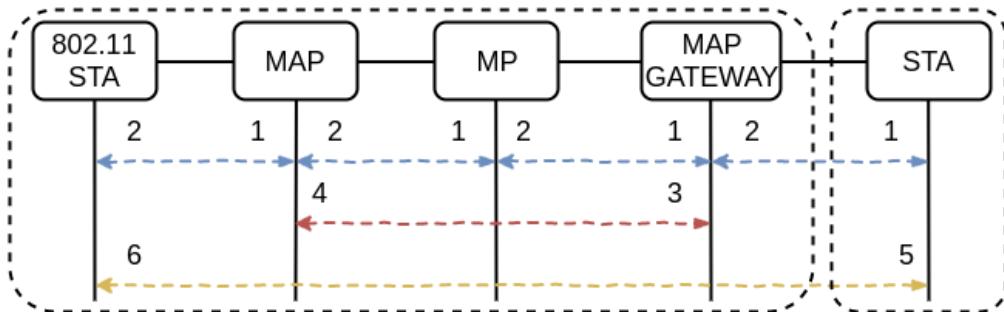


Figura 1.15: Direccionamiento en 802.11s

En conclusión, 802.11s provee de una característica poco utilizada actualmente: el enrutamiento a nivel de capa 2. Toda la operación es realizada por la NIC (del inglés *Network Interface Card*) sin necesidad de implementar completamente la pila de protocolos *TCP/IP*. El objetivo puede ser generar una red de distribución mayormente inalámbrica, heterogénea y de buen rendimiento como la mostrada en la figura 1.16.

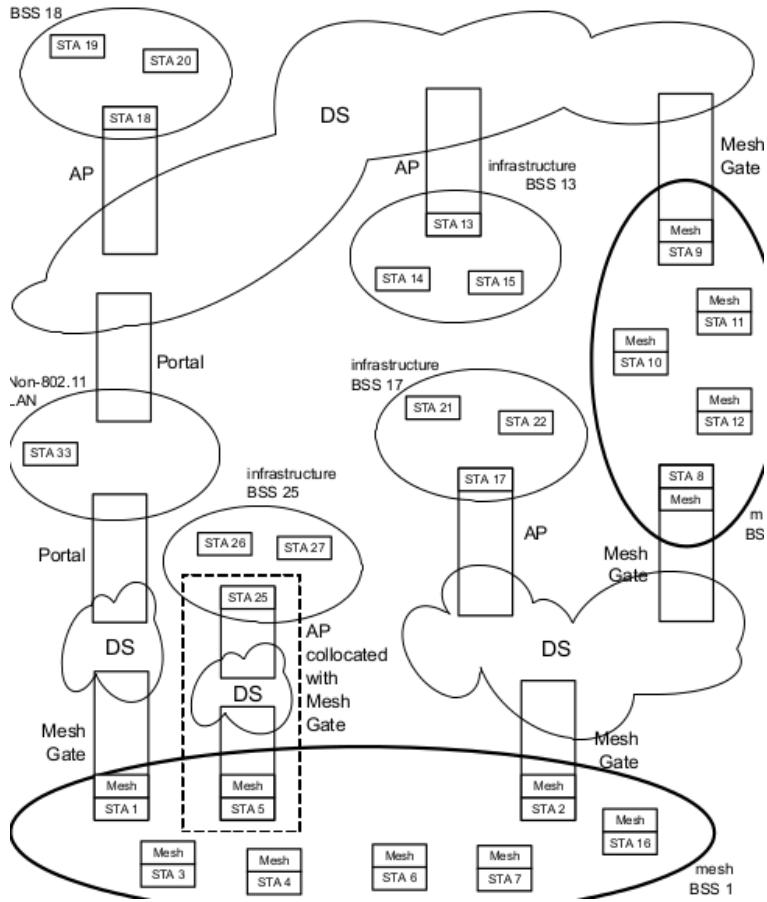


Figura 1.16: Red de distribución heterogénea con compatibilidad *Mesh* [9]

A pesar de los intentos de la IEEE por añadir capacidad *Mesh* a los protocolos 802.11, no existen dispositivos *System on Chip* que integren esta funcionalidad. Por tal motivo, se propone el actual trabajo de grado como una primera implementación de estas topologías utilizando los estándares tradicionales IEEE 802.11 b/g/n, con la finalidad de desarrollar una solución de red para nuevos desarrollos en *Internet de las Cosas* en la Universidad Simón Bolívar.

#### 1.4. Proyecto de red *Mesh* en el campus de la Universidad Nacional de Singapur

El 23 de septiembre de 2016 se dio inicio al programa piloto de conectividad a través de una red *Mesh Wi-Fi* en la sede de la *Universidad Nacional de Singapur* [20] a partir de una alianza con la empresa de telecomunicaciones *StarHub*. La plataforma se basa en Puntos de Acceso Móviles colocados en los buses que circulan a través de las vías y dispositivos fijos colocados en edificios estratégicos. Los enlaces son generados de manera dinámica siguiendo los parámetros ya mencionados en este capítulo. Se genera ubicuidad en la red *Wi-Fi* a medida que los usuarios viajan a lo largo del campus y se plantea recolectar datos para analizar patrones de movimiento que sirvan para proveer y ofrecer mejores productos. También se dispone de la posibilidad de localizar las unidades de transporte en tiempo real y la cantidad de pasajeros que se encuentran dentro de estas.

La iniciativa corresponde al interés de la Universidad de ser parte de programas piloto por parte de compañías que deseen probar sus nuevos desarrollos.

Se ha logrado recolectar más de 540GB de tráfico de internet en 3 meses del programa. Por otro lado, la plataforma pretende proveer de conectividad para nuevos desarrollos en *Internet de las Cosas* que se realicen dentro del campus. El profesor Lawrence Wong, director del *Instituto de Medios Digitales e Interactivos* de la universidad comenta que “La tecnología es bastante nueva y somos entusiastas de entender algunos de los aspectos importantes sobre su rendimiento”. La figura 1.17 ilustra el campus y la posible conectividad entre sus miembros.

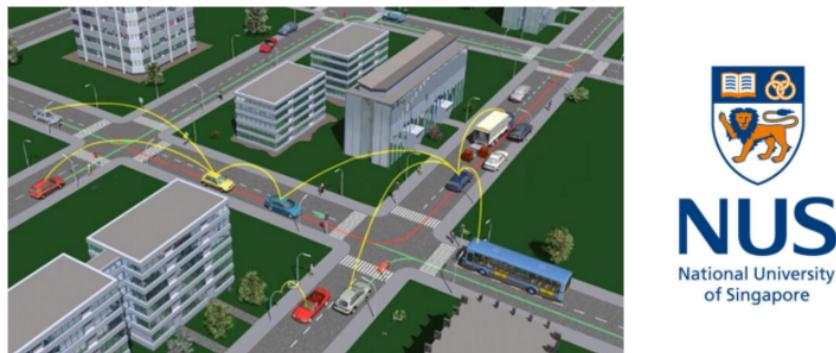


Figura 1.17: Proyecto de red *Mesh* en el campus de la Universidad Nacional de Singapur

## CAPÍTULO 2

### MÓDULO ESP8266 DE ESPRESSIF SYSTEMS

Simplicidad es un prerequisito de  
confiabilidad

---

*Edsger W. Dijkstra*

El ESP8266 es un SoC (del inglés *System on Chip*) que provee soluciones de conectividad *Wi-Fi* y procesamiento para la industria del *Internet de las Cosas*. Su alto nivel de integración, eficiente consumo de energía y bajo costo lo han convertido en una opción popular al momento de desarrollar aplicaciones que alberga de manera *standalone* o suministrando conectividad a otro MCU (del inglés *Micro-Controller Unit*).

Se disponen de varias plataformas para programar el chip, dependiendo del nivel de confiabilidad y dificultad necesario. Su amplio uso en el ecosistema *IoT* (del inglés *Internet of Things*) ha hecho que se genere una creciente comunidad de desarrolladores.

Segmentos típicos para desarrollar productos en torno al ESP8266 involucran: dispositivos del hogar, automatización, control de luces, redes malladas, control inalámbrico industrial, *wearables*, cámaras ip, redes de sensores, etiquetas de identificación, entre otros.

#### 2.1. Especificaciones generales

En la figura 2.1 se observa el diagrama general del ESP8266. Se observan los bloques de radio (en azul) junto a los bloques de la unidad central procesamiento y manejo de interfaces (en amarillo). Algunas características generales se listan a continuación:

- Voltaje de alimentación: 3.0V a 3.6V
- Corriente consumo promedio: 80mA
- Corriente máxima de salida en pines I/O: 12mA
- Tamaño del encapsulado: QFN32-pin (5mm x 5mm)
- Versiones de *Wi-Fi*: IEEE 802.11b/g/n
- Modos *Wi-Fi*: STA/SoftAP/STA+SoftAP (del inglés *Software-AP*)
- Protocolo de red: IP version 4
- Seguridad: WPA/WPA2 del inglés *Wi-Fi Protected Access*)

- Encriptación: WEP/TKIP/AES (del inglés *Wired Equivalent Privacy/Temporal Key Integrity Protocol/Advanced Encryption Standard*)
- Actualización de *firmware*: UART / OTA (del inglés *Over-The-Air*)

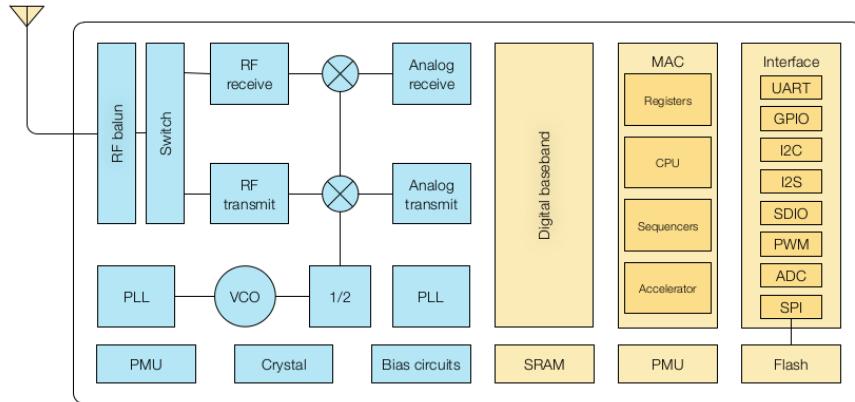


Figura 2.1: Diagrama de bloques funcionales del ESP8266 [21]

Por otro lado, en la imagen se observan los 8 módulos de los cuales se dispone para interactuar con el entorno y desarrollar aplicaciones:

- **UART:** del inglés *Universal Asynchronous Receiver Transmitter*. Dispositivo que realiza conversión paralelo-serial y serial-paralelo entre un periférico y la unidad de procesamiento [22]. El ESP8266 integra dos interfaces de comunicación: *UART0* y *UART1*:
  - U0RXD: pin 25.
  - U0TXD: pin 26.
  - U0RTS: pin 13.
  - UOCTS: pin 12.
  - U1TXD: pin 14.
  - U1RXD: pin 23.
- **I2C:** del inglés *Inter-Integrated Circuit*. Popular protocolo de comunicación serial entre un *Maestro* y un *Esclavo* a través de sólo dos cables entre ambos [23]. El chip incorpora una interfaz de comunicación *I2C*:
  - I2C\_SCL: pin 9.
  - I2C\_SDA: pin 14.
- **I2S:** del inglés *Integrated-IC Sound*. Utilizado principalmente en aplicaciones de recolección, procesamiento y transmisión de datos de audio, el protocolo hace uso de 3 cables en cada sentido [24]. ESP8266 incorpora una interfaz de entrada y otra de salida.

- I2SI\_DATA: pin 10.
  - I2SI\_DBCK: pin 12.
  - I2SI\_WS: pin 9.
  - I2SO\_DATA: pin 25.
  - I2SO\_BCK: pin 13.
  - I2SO\_WS: pin 14.
- **SPI**: del inglés *Serial Peripheral Interface*. Protocolo de comunicación serial *full-duplex* que utiliza 3 cables para la comunicación *Maestro-Esclavo* [25].
- SPICLK: pin 21.
  - SPIQ/MISO: pin 22.
  - SPID/MOSI: pin 23.
  - SPIHD: pin 18.
  - SPIWP: pin 19.
  - SPICS1: pin 26.
  - SPICS3: pin 15
- **SDIO**: del inglés *Secure Digital Input/Output Interface*. Protocolo diseñado como extensión del popular estándar de tarjetas *SD* que permite conectar periféricos a un controlador [26]. Se incorpora un esclavo *SDIO*:
- SDIO\_CLK: pin 21.
  - SDIO\_DATA0: pin 22.
  - SDIO\_DATA1: pin 23.
  - SDIO\_DATA2: pin 18.
  - SDIO\_DATA3: pin 19.
  - SDIO\_CMD: pin 20.
- **GPIO**: del inglés *General-Purpose Input/Output*. El ESP8266 dispone de 17 pines que pueden ser asignados a diversas funciones durante la programación. Cada uno puede ser asignado con resistencias de *pull-up* o *pull-down* internas.
- **PWM**: del inglés *Pulse Width Modulation*. Se disponen de cuatro interfaces para aplicaciones que requieran esta funcionalidad:
- PWM0: pin 10.
  - PWM1: pin 13.

- PWM2: pin 9.
  - PWM3: pin 16.
- **ADC:** del inglés *Analog-to-Digital Converter*. Se dispone de una interfaz para este propósito a través del pin 6.

En el apéndice B se incluye el diagrama de pines completo e información del encapsulado del ESP8266

### 2.1.1. Especificaciones de radio

El ESP8266 de *Espressif Systems* integra bloques que proveen adecuadas funcionalidades de radiocomunicaciones. Estos incluyen amplificadores de potencia, balun RF (del inglés *radio-frequency*), filtros y bloques de ahorro de energía [21].

El transceiver soporta las versiones de *Wi-Fi* 802.11 b/g/n que se detallaron en el capítulo anterior. Adicionalmente, dependiendo de la aplicación, se puede ser parte de una BSS (del inglés *Basic Service Set*) a través del modo *STA*, establecer su propia red de infraestructura con el modo *SoftAP* o cumplir ambas funciones a través del modo *SoftAP+STA*. Es este último el que será de gran utilidad al momento de realizar la implementación de la red *Mesh*.

Se hace uso de la banda ISM (del inglés *Industrial, Scientific and Medical*) de 2,4GHz. Finalmente, algunos valores tabulados de sensibilidad y potencia de transmisión se colocan en la tabla 2.1.

Tabla 2.1: Potencia de Transmisión y Sensibilidad

Versión de 802.11	Potencia de Transmisión	Sensibilidad
b	+20dBm	-91dBm (11Mbps)
g	+17dBm	-75dBm (54Mbps)
n	+14dBm	-72dBm (MCS7)

### 2.1.2. Especificaciones de potencia

El estándar *Wi-Fi* se caracteriza por altas potencias de transmisión con lo cual se logra cubrir un cierto rango en una WLAN (del inglés *Wireless Local Area Network*). Valores tabulados del consumo de corriente para cada versión de *Wi-Fi* que utiliza el ESP8266 se listan a continuación:

- Tx\_802.11b, CCK 11Mbps, Pout=+17dBm: 170mA

- Rx\_802.11b, longitud del paquete=1KB, -80dBm: 50mA
- Tx\_802.11g, OFDM 54Mbps, Pout=+15dBm: 140mA
- Rx\_802.11g, longitud del paquete=1KB, -70dBm: 56mA
- Tx\_802.11n, Modo MCS7, Pout=+13dBm: 120mA
- Rx\_802.11n, longitud del paquete=1KB, -80dBm: 56mA

### 2.1.2.1. Modos de ahorro de energía

El consumo de potencia es un parámetro fundamental en módulos para *IoT*, por ello el fabricante ha implementado cinco estados en el ESP8266.

- **Apagado:** RTC (del inglés *Real-Time-Counter*) se apaga y todos los registros son reiniciados.
- **Modo Deep Sleep:** sólo el módulo RTC se encuentra encendido. Se mantiene cierta información básica de la conexión *Wi-Fi*.
- **Modo Sleep:** sólo el RTC se encuentra operativo, cualquier evento preestablecido coloca al chip en modo *Wake-Up*.
- **Modo Wake-Up:** el ESP8266 pasa de estados de ahorro de energía a modo *Activo*.
- **Modo Activo:** el módulo se encuentra completamente operacional.

En la figura 2.2 se observa el diagrama de flujo que define el comportamiento entre diferentes estados.

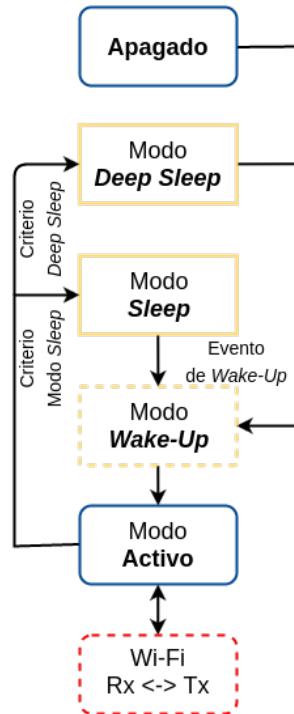


Figura 2.2: Diagrama de flujo entre estados de ahorro de energía

## 2.2. Procesamiento

### 2.2.1. CPU

El ESP8266 incorpora un micro-controlador *Tensilica L106* [27] [28] de 32-bits como unidad de procesamiento con una velocidad de reloj de 80MHz. Sólo 20% de la capacidad se encuentra ocupada por la pila de protocolos *Wi-Fi*, el resto queda libre para el desarrollo de aplicaciones.

### 2.2.2. Memoria

Incluye memorias SRAM (del inglés *Static Random-Access Memory*) y ROM (del inglés *Read-Only Memory*) que pueden ser accedidas a través del *bus de instrucciones*, el bus de datos y la interfaz AHB (del inglés *Advanced High-Performance Bus*).

### 2.2.3. Flash

Se utiliza una memoria *flash* externa para almacenar el programa del usuario. El dispositivo soporta hasta 16MB teóricamente y un mínimo de 512KB. Versiones comerciales utilizan entre 1MB y 4 MB.

### 2.3. Modelos disponibles

*Espressif Systems* ha sacado al mercado varias versiones del ESP8266. La figura ?? muestra las más importantes. Para facilitar su uso y el diseño de aplicaciones para *IoT*, otras compañías han creado módulos de desarrollo en torno al chip. Una de las versiones más utilizadas para este propósito se presenta en la figura 2.4. Este es el modelo *esp-12* que dispone de una mayor cantidad de puertos de entrada/salida disponibles y memoria *flash* externa de 4MB. Su diagrama de pines se ilustra también en la figura correspondiente.

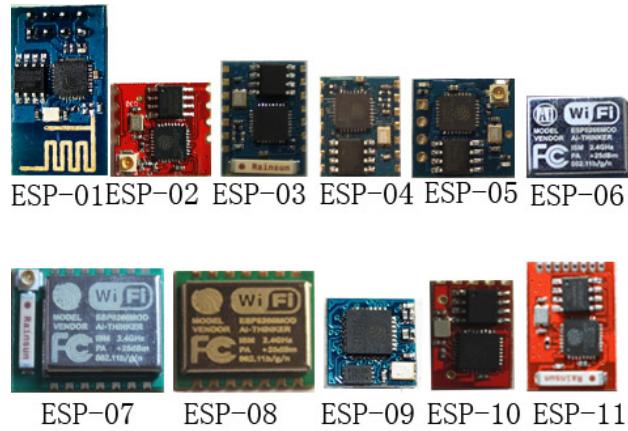


Figura 2.3: Versiones comerciales del *ESP8266*



Figura 2.4: Versión comercial *ESP-12*

#### 2.3.1. esp8266 de *NodeMCU*

La empresa *NodeMCU* [29] ha comercializado un módulo de desarrollo que incluye un conversor *USB-Serial* y un regulador a 3,3V para facilitar la creación de prototipos basados en el

ESP8266. Se exponen más *GPIO* que con otras alternativas. La compañía también ofrece su propio *firmware* para programar el módulo utilizando el lenguaje *Lua*. Se ha decidido utilizar esta versión para la realización del presente trabajo de grado. La figura 2.5 muestra el diagrama de pines correspondiente.

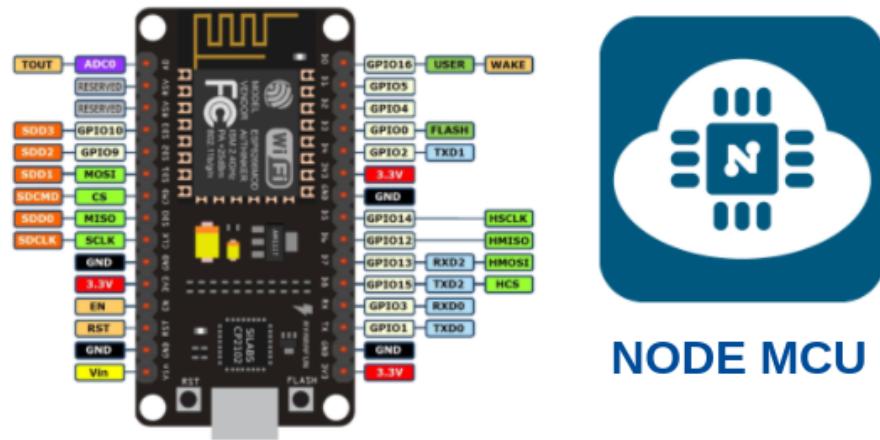


Figura 2.5: Diagrama de pines del ESP8266 de *NodeMCU* [30]

## 2.4. Programación

Debido a la popularidad que ha adquirido, existen varias alternativas para desarrollar soluciones basadas en el ESP8266. Para escoger la más apropiada, se debe hacer un balance entre facilidad y funcionalidades necesarias. A continuación se detallan algunas.

### 2.4.1. Comandos AT

Los comandos AT o conjunto de comandos *Hayes* por el nombre de la compañía en donde fueron desarrollados son instrucciones que se envían al dispositivo por la interfaz UART y son interpretados uno a uno. En el ESP8266 [31], los comandos ofrecen acceso a una gran cantidad de opciones divididas en tres bloques: comandos básicos, relacionados con *Wi-Fi* y comandos para TCP/IP. Esta es la alternativa más común al momento de utilizar el chip junto a otro MCU que alberga la aplicación. A continuación se observa una lista de comandos AT con los cuales se configura el ESP8266 para funcionar como SoftAP y se establece un *Socket UDP* (del inglés *User Datagram Protocol*) con la dirección *192.168.5.2* y el puerto *1024*. La última instrucción permite que los siguientes 8 bytes que se envíen por el terminal serial se transmitan a la dirección antes mencionada.

```

1 AT+CWMODE_DEF=2
2 AT+CWSAP_DEF="SSID" , "PASSWORD" , 5 , 3 , 3
3 AT+CWDHCP_CUR=0 , 1
4 AT+CIPMUX=0

```

```

5 AT+CIPSTART="UDP" , "192.168.5.2" ,1024,1024,0
6 AT+CIPSEND=8 , "192.168.4.2" ,1024

```

## 2.4.2. Arduino IDE

El siguiente código de programación utiliza las librerías del ESP8266 compatibles con el IDE (del inglés *Integrated Development Environment*) de Arduino. Este constituye una de las plataformas más utilizadas por la comunidad en torno a este chip.

```

1 /* CONFIGURAR ESP8266 EN MODO SOFTAP */
2 #include <ESP8266WiFi.h>
3 #include <WiFiClient.h>
4
5 const char *ssid = "SSID";
6 const char *password = "PASSWORD";
7
8 void setup() {
9     delay(1000);
10    Serial.begin(115200); // Tasa de Baudios
11    Serial.println();
12    Serial.print("Configuring access point... ");
13    WiFi.softAP(ssid, password); // iniciar modo SoftAP
14
15    IPAddress myIP = WiFi.softAPIP();
16    Serial.print("AP IP address: "); // imprimir direccion ip
17    Serial.println(myIP);
18 }

```

## 2.4.3. Lua

*NodeMCU* ha desarrollado su propio *firmware* para programar proyectos rápidamente. Este consiste en un interpretador de comandos basados en el lenguaje de programación *Lua* que reduce considerablemente la cantidad de líneas de código necesarias para desarrollar aplicaciones. El *firmware* se escoge de acuerdo a las funcionalidades requeridas para evitar ocupar más memoria de la necesaria, lo que constituye el inconveniente principal al utilizar esta plataforma. En el siguiente segmento de código se detalla un ejemplo en el cual se controla un LED RGB (del inglés *Red-Green-Blue*) a través de PWM.

```

1 --- CONTROLAR LED's A TRAVES DE PWM
2
3 function led(r,g,b) --- PWM en LED

```

```

4     pwm.setduty(1,r)
5     pwm.setduty(2,g)
6     pwm.setduty(3,b)
7 end
8 pwm.setup(1,500,512) — PWM 1
9 pwm.setup(2,500,512) — PWM 2
10 pwm.setup(3,500,512) — PWM 3
11 pwm.start(1) — iniciar PWM
12 pwm.start(2)
13 pwm.start(3)
14 led(512,0,0) — red
15 led(0,512,0) — green
16 led(0,0,512) — blue

```

#### 2.4.4. Espressif SDK RTOS

*Espressif Systems* provee de dos modelos para generar aplicaciones. Están constituidos por un conjunto de API's (del inglés *Application Program Interface*) para acceder a las funcionalidades del hardware. Las actividades de red son realizadas en la librería y no son transparentes a los usuarios, sin embargo se deben inicializar las interfaces en el archivo *user\_main.c*.

El primero de estos modelos es el *Espressif SDK RTOS* (del inglés *Real-Time Operating System*) [32] con el cual se establece un tipo de sistema operativo que regula el despacho de tareas, manejo de recursos y tiempos de ejecución. En el ESP8266 se utiliza *FreeRTOS* [33] para este propósito.

A continuación se coloca un segmento de código que utiliza esta versión del SDK (del inglés *Software Development Kit*) para encender y apagar un LED en períodos de tiempo específicos.

```

1 /* INTERMITENCIA EN LED */
2
3 #include "esp_common.h"
4 #include "gpio.h"
5
6 void LEDBlinkTask (void *pvParameters){
7     while(1){ // ejecutar para siempre
8         vTaskDelay (300/portTICK_RATE_MS); // retardo
9         GPIO_OUTPUT_SET (12, 1); // ON
10        vTaskDelay (300/portTICK_RATE_MS); // retardo
11        GPIO_OUTPUT_SET (12, 0); // OFF
12    }
13 }
14

```

```

15 void user_init(void){ // inicio de la ejecucion
16     printf("SDK version:%s\n", system_get_sdk_version());
17     PIN_FUNC_SELECT (PERIPHS_IO_MUX_MTDI_U, FUNC_GPIO12); // configurar pin 12
18     xTaskCreate(LEDBlinkTask, (signed char *)"Blink", 256, NULL, 2, NULL); // crear tarea
19 }
```

#### 2.4.5. Espressif SDK NON-OS

El segundo modelo de *Espressif Systems* para desarrollar aplicaciones en *IoT* es el *SDK NON-OS* [34]. Utiliza temporizadores y llamadas a funciones para generar secuencias a lo largo del código. Funciones serán ejecutadas si se cumplen ciertas condiciones.

Finalmente, se dispone de un ejemplo en el cual se configura una red *Wi-Fi* de infraestructura con el ESP8266 funcionando en modo *SoftAP*. Se observan claras diferencias con el ejemplo anterior.

```

1 /* CONFIGURAR ESP8266 EN MODO SOFTAP */
2 void user_set_softap_config(void){
3     struct softap_config config;
4     wifi_softap_get_config(&config); // Get config first.
5     char ssid[32] = "SSID";
6     char password[64] = "PASSWORD";
7     os_memcpy(&config.ssid, ssid, 32);
8     os_memcpy(&config.password, password, 64);
9     config.authmode = AUTH_WPA_WPA2_PSK; // seguridad
10    config.ssid_len = 0;
11    config.max_connection = 4; // cantidad de STA
12    wifi_softap_set_config(&config); // configurar e iniciar SoftAP
13 }
14
15 void user_init(void){
16     os_printf("SDK version:%s\n", system_get_sdk_version()); // version del
17     // SDK
18     wifi_set_opmode(0x02); // modo "SoftAP"
19     user_set_softap_config(); // ir a funcion
```

Por ser esta aproximación la que tiene mayor control de las funcionalidades de hardware, incluyendo API's que controlan ciertos parámetros de la comunicación *Wi-Fi*, se utilizará el *SDK NON-OS* para el desarrollo de la red *Mesh* durante el siguiente capítulo.

Se utilizan dos herramientas fundamentales para compilar y grabar el código hacia el ESP8266. Ambas corren bajo el sistema operativo *Linux* y son de libre acceso. Estas se detallan a continuación.

#### 2.4.5.1. crosstool-NG

Es un generador de cadenas de herramientas [35] que se utiliza para compilar, ensamblar y vincular el código realizado en lenguaje C con las API's que ofrece *Espressif Systems* y obtener los archivos binarios que serán descargados en el ESP8266.

#### 2.4.5.2. esptool.py

Es un software libre basado en el lenguaje de programación *python* con la utilidad de acceder al cargador de arranque del ESP8266 y grabar los archivos binarios generados anteriormente en la memoria flash del dispositivo [36]. El chip soporta dos métodos para actualizar el *firmware*: *FOTA* (del inglés *Firmware Over-The-Air*) y *Non-FOTA*. El primero de ellos permite actualizar el módulo de forma remota. Dependiendo de las necesidades de la aplicación se deberá utilizar un método u otro. En la tabla 2.2 aparecen las direcciones de memoria en donde deben grabarse los archivos binarios correspondientes en el modo *Non-FOTA*.

Las siguientes líneas disponen una serie de comandos típicos con los cuales se pueden descargar los archivos binarios generados por *crosstool-NG* en las posiciones de memoria adecuadas según el método *non-FOTA*. Se deben especificar los siguientes parámetros:

- Puerto a utilizar: en Linux */dev/ttyUSB0*.
- Modo SPI: para el caso de buses SPI tipo *Quad* se utiliza el modo *QIO*.
- Tamaño de la memoria flash: en el caso del ESP8266 de *NodeMCU* es 32Mbits (4MB).
- archivos binarios y sus correspondientes direcciones de memoria según lo dispuesto en la tabla 2.2.

```
1 sudo esptool.py --port /dev/ttyUSB0 write_flash --flash_mode qio --flash_size
   32m 0x00000 eagle.flash.bin 0x40000 eagle.irom0text.bin 0x3FC000
   esp_init_data_default.bin 0x3FE000 blank.bin
```

Tabla 2.2: Ubicación de archivos binarios para grabar en la memoria del ESP8266

Binarios	Tamaño de la memoria Flash			
	512	1024	2048	4096
<b>esp_init_data_default.bin</b>	0x7C000	0xFC000	0x1FC000	0x3FC000
<b>blank.bin</b>	0x7E000	0xFE000	0x1FE000	0x3FE000
<b>eagle.flash.bin</b>	0x00000			
<b>eagle.irom0text.bin</b>	0x40000			

## CAPÍTULO 3

### ESPRESSIF-MESH

Nunca alguna gran obra se ha hecho de prisa. Lograr un gran descubrimiento científico, imprimir una excelente fotografía, escribir un poema inmortal...

hacer cualquier gran logro requiere tiempo, paciencia y perseverancia

---

W. J. Wilmont Buxton

Con la necesidad de conectar una mayor cantidad de nodos a las redes de *IoT* (del inglés *Internet of Things*) y otras razones ya detalladas en el capítulo 1 del presente trabajo de grado, *Espressif Systems* ha desarrollado una serie de API's (del inglés *Application Program Interface*) para ayudar a los desarrolladores a implementar sus propias aplicaciones que requieran topologías *Mesh*.

De esta manera, se dispone a explicar la arquitectura que ofrece la división de software del fabricante como solución de red en *IoT*.

#### 3.1. Mesh API's

Se ha utilizado la versión 1.5.3 del *SDK NON-OS* [34] para programar los módulos ESP8266 de *NodeMCU*. A pesar de haber otras más recientes disponibles, esta es la base de la librería para desarrollos *Mesh* [37]. Este conjunto de funcionalidades también incluyen las estructuras básicas que se necesitan para tal propósito .

De forma ilustrativa, a continuación se introducen algunos prototipos de funciones utilizadas en el *firmware* de la red.

```
1 // imprime la version del firmware Mesh que se esta utilizando
2 void espconn_mesh_print_ver(void)
3
4 // muestra la tabla de enrutamiento del nodo actual
5 void espconn_mesh_disp_route_table(void)
6
7 // activa red mesh y establece la funcion de callback
8 void espconn_mesh_enable(espconn_mesh_callback enable_cb, enum mesh_type type)
9
10 // obtiene direccion MAC de destino del paquete Mesh
11 bool espconn_mesh_get_dst_addr(struct mesh_header_format *head, uint8_t **dst_addr)
```

### 3.1.1. Limitaciones

A pesar de las opciones que ofrece, la compañía no ha hecho transparente el acceso a las configuraciones a nivel de la pila de los protocolos de 802.11. Así, todas las funcionalidades de red del estándar *Wi-Fi* no pueden ser accedidas por el usuario. El aspecto más importante que genera esta limitación es el no poder modificar el protocolo de acceso al medio de las versiones de *802.11 b/g/n*, implementadas por el ESP8266. Además, no se tiene soporte del estándar *802.11s*, detallado en el capítulo 1 y concebido para ofrecer conectividad *Mesh*.

Se debe abordar otra aproximación para garantizar enrutamiento *multi-salto* en sistemas embedidos con estas características. Para ello, las API's *Mesh* mencionadas anteriormente sirven de enlace para acceder a ciertas parámetros configurables. Desde el punto de vista del fabricante, esto ofrece una plataforma suficientemente robusta para el desarrollo de aplicaciones.

### 3.2. Especificaciones generales

Como punto de partida, se utiliza el *ESP\_MESH\_DEMO* que ofrece *Espressif Systems* [38]. Esto, como se detallará más adelante, es una primera aproximación, útil para desarrollar rápidamente una topología *Mesh* basada en 802.11 con bajo costo. *802.11s* junto con protocolos que consideren aspectos de potencia y canales de frecuencia separados, constituyen la solución más adecuada.

La implementación hace uso del modo dual *SoftAP+STA* en el cual puede funcionar el ESP8266. Cada nodo hace parte de una BSS (del inglés *Basic Service Set*) y genera al mismo tiempo su propia red de infraestructura para que nuevos nodos puedan adherirse. De esta manera, cada módulo cumpliría funciones de *Mesh Point*, terminología de *802.11s* ya detallada en el capítulo 1, y albergaría su propia aplicación de *IoT* según las configuraciones necesarias. Por otro lado, como limitación de la pila de protocolos del estándar *Wi-Fi*, se tiene que el ESP8266 en modo *SoftAP* sólo puede funcionar en las versiones *802.11 b/g*, lo que limita las mayores tasas de bits a nivel de capa física que se pudiesen alcanzar con la versión *802.11n*.

Se establece una topología tipo árbol donde cada nodo, en diferentes niveles de jerarquía, tiene un *padre* y múltiples *hijos*. El número de *hijos* por nodo se limita a 4, característica impuesta por el máximo número de conexiones que puede soportar el ESP8266 en modo *SoftAP*.

Finalmente, lo que se tiene es una red que sigue el patrón de la figura 3.1. El *nodo raíz* y el *routerAP* se caracterizan más adelante en el presente capítulo. Por consiguiente, se ha implementado una versión de la estructura tipo árbol presente en *802.11s* con enrutamiento proactivo, apta para aplicaciones que necesiten bajas tasas de bits a nivel de capa de aplicación. La topología jerárquica y los lentos procesos de autenticación y asociación del estándar *Wi-Fi* hacen que la movilidad también sea una limitación. Redes de sensores a lo largo de una amplia zona de cobertura con nodos

no-móviles, constituye el foco de este desarrollo.

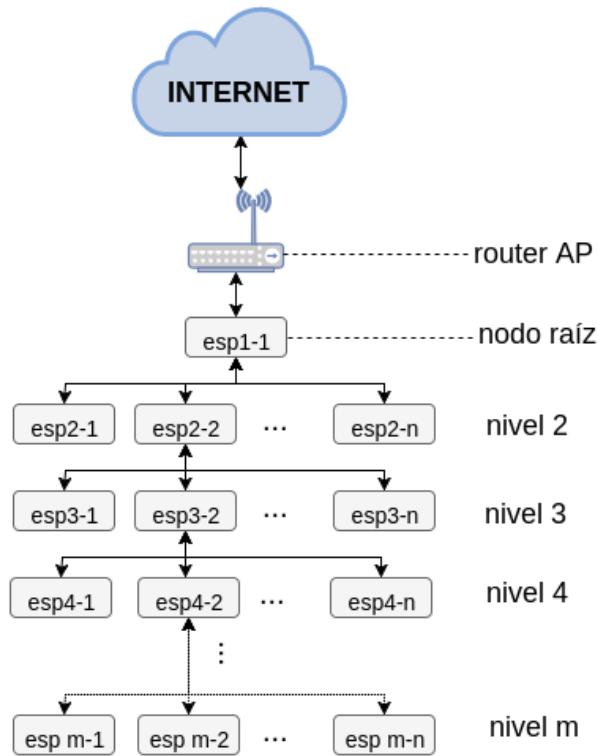


Figura 3.1: Topología de la red *Mesh*

### 3.3. Configuración inicial

El archivo *user\_config.h* de la implementación *Mesh*, dispone de los parámetros iniciales de la red. A continuación se presentan algunos de estos. Destacan aquellos correspondientes al modo *SoftAP* de cada nodo, donde se debe detallar su respectivo SSID, clave y modo de seguridad. Por su parte, *MESH\_MAX\_HOP* limita el número máximo de niveles que se pueden establecer en la estructura jerárquica de la figura 3.1.

El SSID descrito, es una combinación del *MESH\_SSID\_PREFIX* del archivo de configuración, el nivel de la red (X) en el cual se encuentra el nodo y los últimos seis caracteres de su dirección MAC (YYYYYY). El fabricante modifica los tiempos de difusión de *beacons*. Así, no es posible escanear las redes que se establecen tras cada nuevo nodo. Además, esto evita saturar el canal de radio al no reservar el medio durante estos períodos de transmisión.

```

1 // Tasa de Baudios de la interfaz UART
2 static const uint32_t UART_BAUT_RATIO = 115200;
3 // identificador de cada red MESH
4 static const uint8_t MESH_GROUP_ID[6] = {0x18,0xfe,0x34,0x00,0x00,0x50};
5 // direccion MAC del routerAP
  
```

```

6 static const uint8_t MESH_ROUTER_BSSID[6] = {0xE0,0x2A,0x82,0x3C,0xF3,0xAD};
7
8 #define MESH_ROUTER_SSID      "routerAP"      // SSID del routerAP
9 #define MESH_ROUTER_PASSWD   "12345678"      // password del routerAP
10 #define MESH_SSID_PREFIX    "MESH_DEMO"      // SSID :"MESH_SSID_PREFIX_X_YYYYYY"
11 #define MESH_AUTH           AUTH_WPA2_PSK // modo de seguridad Wi-Fi
12 #define MESH_PASSWD         "123123123"     // password en cada SoftAP
13 #define MESH_MAX_HOP        (4)             // maximo numero de saltos

```

### 3.4. Topología

Durante la inicialización, se requiere de un *nodo raíz* que haga parte de la red de infraestructura que establece el *routerAP*. A partir de este nivel primario, se derivan los siguientes nodos. El *routerAP* será la puerta de acceso hacia la WAN (del inglés *Wide Area Network*) cableada, comúnmente del ISP (del inglés *Internet Service Provider*).

El valor *topo\_test\_time* del archivo *user\_config.h* establece el período de tiempo en cual cada nodo verificará la topología de la red y actualizará su tabla de enrutamiento, según las características que se explicarán en las siguientes secciones de este capítulo. Esto constituye un mensaje de control fundamental para agilizar los tiempos de convergencia. El valor por defecto es de 14000ms.

```

1 static const uint32_t topo_test_time = 14000;

```

Al momento en el cual el *nodo raíz* se asocia con el *routerAP*, comienzan a utilizar un canal *Wi-Fi* disponible en la banda de 2,4GHz. A partir de este momento, todo nuevo que se adhiera a la topología utilizará también esta frecuencia. Esto limita considerablemente el rendimiento de la red, cada nodo competirá con los demás para poder transmitir a través del medio. Tramas de control, gestión y datos, de cada nivel de la red, reservan el canal a través del protocolo *CSMA-CA*. Una mejor aproximación, más allá del alcance del presente trabajo de grado, hace uso de múltiples canales de radio para transmitir [802\_11\_radio\_channels]. Se deben generar nuevos y eficientes protocolos de enrutamiento que consideren esta alternativa. Nodos pueden utilizar un canal de radio para mensajes de control y otros para transportar los datos. Se pueden generar algoritmos que distribuyan la carga dependiendo de la cantidad de estaciones que transmiten a la misma frecuencia. La figura 3.2 ilustra lo detallado anteriormente donde cada canal de radio se caracteriza por un color específico y pueden ocurrir transmisiones simultáneas.

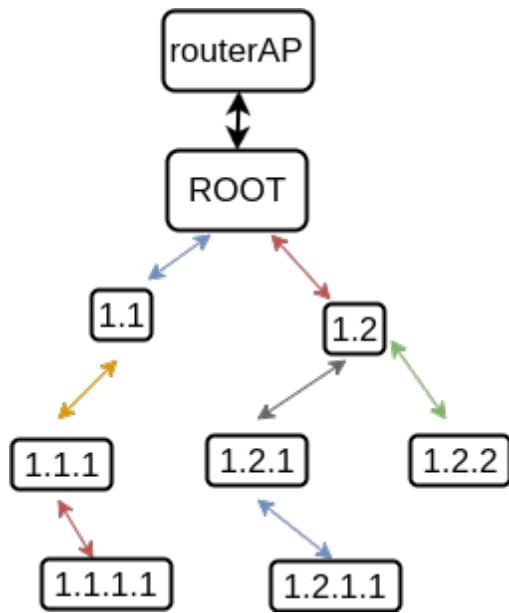


Figura 3.2: Red *Mesh* multi-canal

Las asociaciones entre nodos de la red se hacen tomando en cuenta las métricas de RSSI (del inglés *Received Signal Strength Indication*) y de la capacidad del *SoftAP* respectivo para albergar otra STA dentro de su propia BSS. De esta manera, si un nodo sale del rango de cobertura del nodo *padre* con el cual estaba asociado, inmediatamente escanea por nuevos posibles *SoftAP* con los cuales incorporarse nuevamente a la red. Esta decisión se toma a nivel de la pila de protocolos de 802.11 y su interfaz *Mesh* a las cuales el usuario no tiene acceso.

De esta manera, la distancia por enlace a través de la red, depende de la potencia de transmisión del ESP8266. Este valor, descrito en el capítulo 2, al mismo tiempo depende de la versión de 802.11 en la cual este funcionando. Consecuente a esto, el rango completo de la topología *Mesh* estará definido por la cantidad de niveles que se tengan a lo largo de la estructura. La cantidad de nodos, como se describirá en las siguientes secciones, es un valor relacionado directamente con la cantidad de memoria disponible en el dispositivo.

### 3.4.1. Nodo raíz

En el caso que los datos se deseen transmitir hacia redes externas, el *nodo raíz* se encarga de concentrar todo el tráfico y enrutarlo hacia el *routerAP*. Cuando se inicia un nuevo nodo, este escanea redes que se encuentren en su rango de cobertura. Primero, intenta conectarse con otros módulos ESP8266 que se encuentren funcionando como *SoftAP*. Luego, si no encuentra ninguno, intenta asociarse con el *routerAP* del cual posee información en su archivo de configuración. Si tiene éxito, se convierte en el *nodo raíz* de su propia red *Mesh*.

La selección de este rol ocurre de forma dinámica tomando en cuenta el valor de RSSI. Si en una

red ya existe un *nodo raíz* con un cierto valor de potencia recibida desde el *routerAP* y se adhiere un nuevo nodo con un valor más alto, este último adquiere el nuevo rol. La tabla de direcciones MAC es liberada y los enlaces en los distintos niveles se forman nuevamente de acuerdo a las métricas ya mencionadas. La figura 3.3 ilustra el algoritmo que rige el proceso de selección.



Figura 3.3: Selección del nodo raíz

Todo nodo arma su lista de direcciones MAC con cada miembro de la red. El *nodo raíz* la construye de forma distinta. Este adquiere información a través de la API *espconn\_mesh\_get\_node\_info(enum mesh\_node\_type type, uint8\_t \*\*info, uint16\_t \*count)*. El primer parámetro lo constituye el nodo o conjunto de nodos a los cuales va dirigido el mensaje de solicitud de información. Este parámetro puede tomar uno entre tres valores distintos: MESH\_NODE\_CHILD, MESH\_NODE\_PARENT o MESH\_NODE\_ALL. El último de esta lista es el utilizado en solicitudes de topología. Los valores almacenados en el segundo parámetro corresponden a las direcciones MAC de cada dispositivo y el último incluye la cantidad de nodos que han respondido. La figura 3.4 ilustra este procedimiento.



2\_MainMatter/Capitulo3/Imagenes/root\_node\_get\_topology.png

Figura 3.4: Adquisición de topología en el nodo raíz

### 3.4.2. Otros nodos de la red

Los demás nodos adquieren una posición dentro del árbol y albergan su respectiva aplicación. Estos construyen su lista de direcciones MAC mediante paquetes con opciones de *solicitud de topología* o M\_O\_TOPO\_REQ, los cuales son difundidos a través de mensajes de *broadcast*. Más adelante en el presente capítulo se detallan los tipos de mensajes (unicast, p2p, multicast y broadcast) disponibles. La figura 3.5 ejemplifica el proceso por el cual un nodo en cualquier nivel de la red actualiza su lista de direcciones.



Figura 3.5: Adquisición de topología en otros nodos de la red

El apéndice A incorpora parte del código de programación embebido en el ESP8266 para cada tipo de adquisición de topología. El proyecto completo se podrá encontrar en la plataforma [github.com/LPerezBustos](https://github.com/LPerezBustos).

### 3.4.3. Limitaciones de memoria

El número máximo de nodos que se pueden adherir a la red está limitado por el espacio disponible en la memoria del dispositivo reservada para datos del usuario. El archivo *mesh\_device.c* provee de las funcionalidades necesarias para administrar la tabla de direcciones MAC. La siguiente ecuación determina el tamaño máximo que adquiere este arreglo.

$$mac\_table\_size = \left( \frac{4^{MESH\_MAX\_HOP} - 1}{3} \right) * 6 \quad (3.1)$$

La cantidad entre paréntesis determina cuántos nodos pueden estar en la estructura para un cierto valor de *MESH\_MAX\_HOP*, configurable en el archivo *user\_config.h*. El factor 6 establece la cantidad de bytes necesarios para almacenar cada dirección MAC.

### 3.5. Enrutamiento

A diferencia de los estándares tradicionales, el enrutamiento interno a través de la red *Mesh* se hace utilizando la dirección MAC de cada dispositivo, similar a lo utilizado por el protocolo *IEEE 802.11s*. Con este propósito, se construye la tabla detallada en la sección anterior. A nivel de capa de red, las direcciones ip de cada nodo son asignadas por el servidor DHCP (del inglés *Dynamic Host Configuration Protocol*) de cada *SoftAP* dependiendo del nivel en que se encuentre. De esta manera, un módulo en el segundo nivel de la estructura podrá tener una dirección 2.255.255.x con máscara 255.255.255.0 y gateway 2.255.255.1. El primer dígito de la dirección ip es determinado por la posición en el árbol. En el caso del *nodo raíz*, su dirección es determinada por el *routerAP*.

El enrutamiento por dirección MAC se hace imperativo debido a que en el mismo nivel del árbol, dos nodos pueden compartir la misma ip. La figura 3.6 ilustra esta posibilidad. Al tener la misma configuración DHCP, proveniente del *firmware mesh*, el nodo 1.1.1 comparte la misma dirección a nivel de capa de red que el nodo 1.2.1. Sin embargo, la dirección MAC de cada módulo es asignada a la interfaz de radio de manera única. Esto abre la posibilidad a este tipo de enrutamiento en WMN (del inglés *Wireless Mesh Networks*).

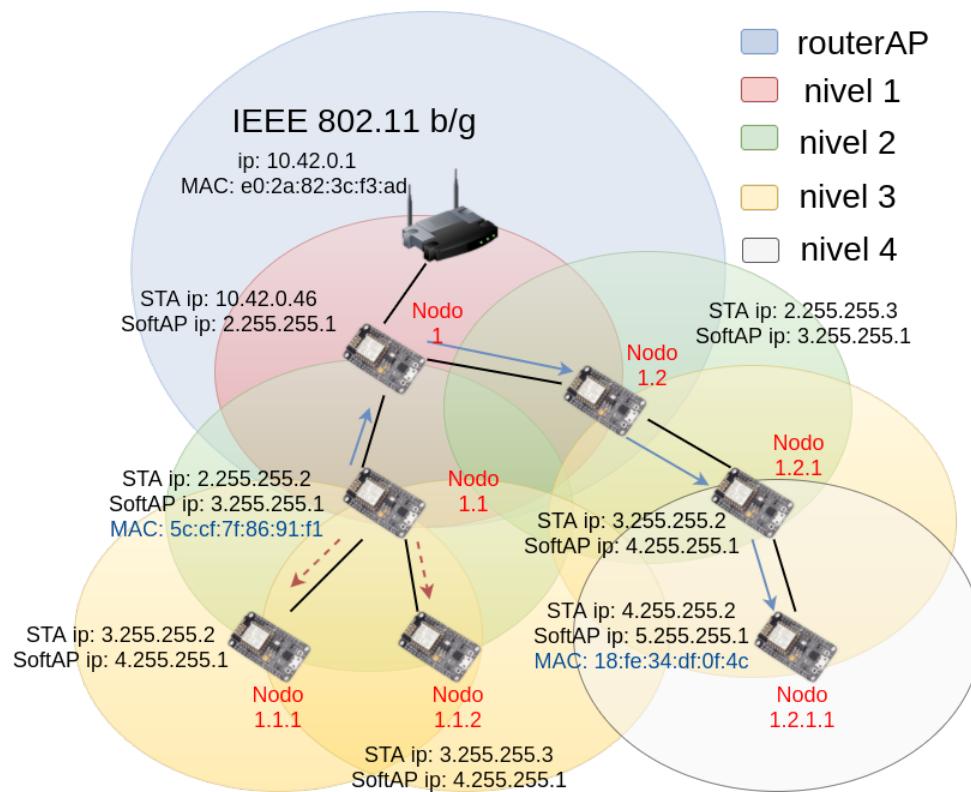


Figura 3.6: Enrutamiento a través de la red

En la estructura jerárquica que se plantea, a pesar de conocer las direcciones MAC de todos los módulos en la red a través de la estructura *g\_node\_list.list*, cada estación sólo tiene la ubicación de

sus respectivos sub-nodos. Si el módulo al cual se desea enviar un paquete se encuentra dentro de la red pero no es un sub-nodo, el mensaje se transmite hacia el *padre*. Este constituye la dirección por defecto de cualquier STA. Por consecuente, el paquete puede realizar múltiples saltos hasta llegar al *nodo raíz*, el cual si dispone de toda la topología de la red para enrutar. De esta manera, a diferencia de ciertas configuraciones *Mesh*, sólo existe un camino entre dos puntos distantes de la estructura.

También en la figura 3.6 se observa el procedimiento descrito. El nodo 1.1 (MAC 5c:cf:7f:86:91:f1), desea enviar un mensaje p2p (del inglés *peer-to-peer*) al 1.2.1.1 (MAC 18:fe:34:df:0f:4c) . Ninguno de sus sub-nodos corresponden con la dirección deseada por lo que se enruta el paquete hacia el nodo *padre*. En este caso, el *nodo raíz*, al conocer la topología completa, sabe la ubicación de 1.2.1.1. Finalmente, el mensaje llega satisfactoriamente a su destino.

Para imprimir la tabla de enrutamiento de cada módulo vía UART (del inglés *Universal Asynchronous Receiver/Transmitter*) se utiliza la API `void espconn_mesh_disp_route_table(void)`. A continuación, para ilustar la sintaxis utilizada, se muestra la tabla de enrutamiento del *nodo raíz* en la topología de cuatro nodos de la figura 3.7.

```

1   cidx:0, count:1 mac_list: // bifurcacion_0 , 1 nodo en bifurcacion_0
2   idx:0, 5c:cf:7f:86:91:f1 // mac_addres_list de la bifurcacion_1
3
4   cidx:1, count:2 mac_list: // bifurcacion_1 , 2 nodos en bifurcacion_1
5   idx:0, 5c:cf:7f:86:93:5f // mac_addres_list de la bifurcacion_1
6   idx:1, 5c:cf:7f:c1:1a:de

```

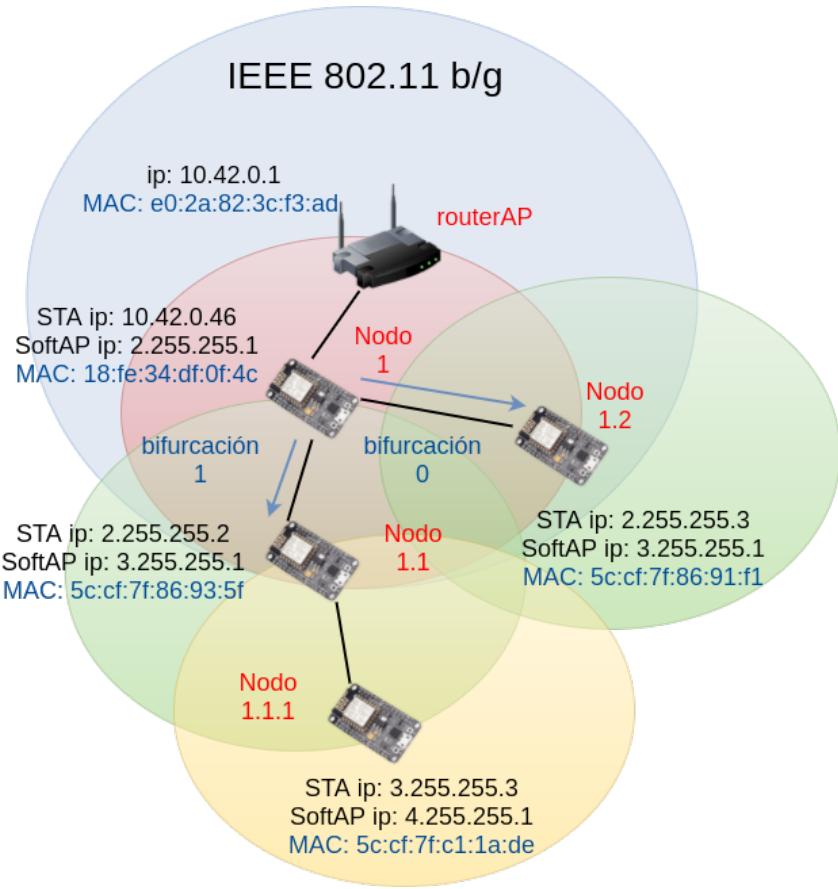


Figura 3.7: Topología *Mesh* demostrativa

### 3.6. Modos de funcionamiento

El modo de funcionamiento se elige al momento de habilitar la red en el archivo *mesh\_demo.c* a través de la API que se presenta a continuación:

```
1 espconn_mesh_enable(mesh_enable_cb, MESH_TYPE)
```

#### 3.6.1. MESH\_ONLINE

En este modo el *nodo raíz* establece una conexión TCP (del inglés *Transmission Control Protocol*) con un servidor a través de un puerto y dirección ip configurables en *user\_config.h*. Cada nodo envía un mensaje unicast al servidor en períodos de 7000ms. Este tiempo también constituye un parámetro configurable.

```
1 static const uint32_t application_time = 7000; // periodo de mensaje unicast
2 static const uint16_t server_port = 7000; // puerto del servidor
3 static const uint8_t server_ip[4] = {10, 42, 0, 1}; // ip del servidor
```

### 3.6.2. MESH\_LOCAL

Este modo sólo puede ser habilitado en el *nodo raíz*. Esto evita que se establezca una conexión TCP entre este nodo y un servidor. El *routerAP* sigue siendo necesario en este topología, sin embargo no hace falta conexión con redes externas.

### 3.7. Estructura del encabezado *Mesh*

La conectividad *Mesh* se implementa entre la capa 2 del estándar *Wi-Fi* y la capa de red. A diferencia de *IEEE 802.11s* que modifica el encabezado MAC, en la implementación en el ESP8266, al no poder modificar el estándar, se incorpora un encabezado adicional al MSDU utilizando parte de los 2312 bytes disponibles (tamaño máximo del MSDU de 802.11). De esta manera, el resultado final es equivalente a agregar una subcapa *Mesh* cuyo PDU se fija en 1300 bytes en el archivo *mesh.h*. En la figura 3.8 se observa la pila de protocolos equivalente.

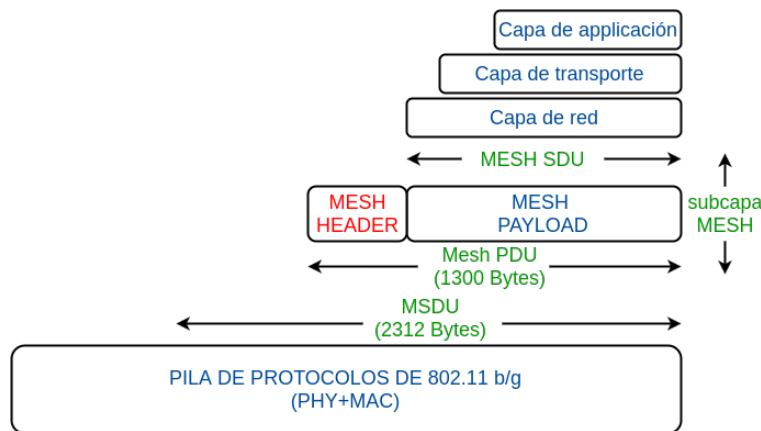


Figura 3.8: Pila de protocolos en el ESP8266 con conectividad *Mesh*

El tamaño del encabezado de la subcapa *Mesh* es variable, depende del número de *opciones* que se agreguen. Estas *opciones* constituyen mensajes de control en la red. De esta manera, la figura 3.9 muestra la estructura respectiva del encabezado y se detalla cada componente a continuación.

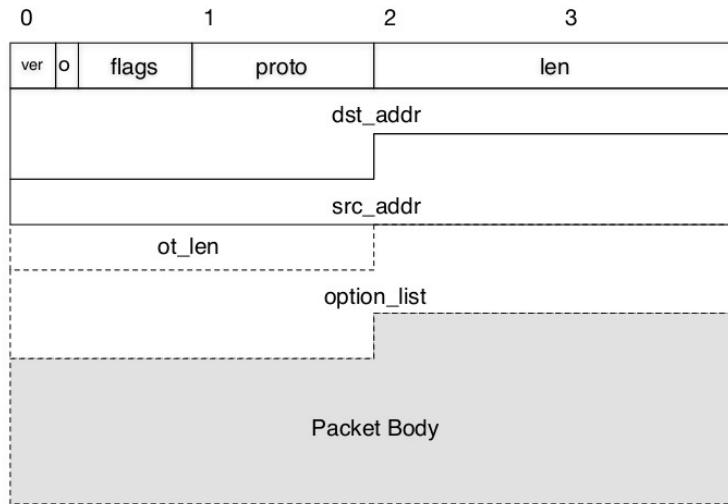


Figura 3.9: Estructura del encabezado *Mesh* [38]

- **ver (2 bits):** versión de la implementación *Mesh*.
- **o (1 bit):** bandera de opción en paquete *Mesh*.
- **flags (5 bits):** 3 banderas:
  - **cp (1 bit):** permite *piggybacking* el paquete *Mesh*.
  - **cr (1 bit):** solicitud de *piggybacking* en paquete *Mesh*.
  - **resv (3 bits):** reservado para futuros desarrollos.
- **proto (8 bits):** 3 campos:
  - **D (1 bit):** bandera que indica dirección que tomará el paquete *Mesh*. Puede ser hacia niveles superiores (1 lógico) o inferiores de la red (0 lógico).
  - **p2p (1 bit):** indica si el paquete es un mensaje p2p o no.
  - **protocol (6 bits):** protocolo de capa de aplicación utilizado. Se dispone de 5 opciones dependiendo de las funcionalidades necesarias. Más adelante en el capítulo se detalla este campo.

```

1 struct {
2     uint8_t d: 1;
3     uint8_t p2p:1;
4     uint8_t protocol:6;
5 } proto;

```

- **len (16 bits):** longitud del cuerpo del mensaje o *Packet Body*.

- **dst\_addr (48 bits):** dirección final de destino. Tres opciones son permitidas:

- **si (proto.D=0) ó (proto.P2P=1):** *dst\_addr* es la dirección MAC del módulo ESP8266 de destino (dentro de la misma red *Mesh*).
- **si es un mensaje broadcast o multicast:** *dst\_addr* es la dirección MAC de *broadcast* (00:00:00:00:00:00) o *multicast* (01:00:5E:00:00:00) respectivamente. Cada dirección se representa como un arreglo de 6 bytes que contiene valores en formato hexadecimal con la MAC correspondiente.
- **si (proto.D=1) y (proto.P2P=0):** *dst\_addr* representa la dirección ip y el puerto en que se puede acceder al servidor (hacia red ip externa). Más adelante se detalla cómo construir cada paquete dependiendo del tipo de mensaje.

- **src\_addr (48 bits):** dirección del dispositivo de origen. Al igual que en *dst\_addr*, tres tipos de direcciones se deben utilizar:

- **si (proto.P2P=1) ó (proto.D=1):** *src\_addr* representa la dirección MAC del ESP8266 de origen.
- **si es un mensaje broadcast o multicast:** *src\_addr* también es la dirección MAC del módulo de origen.
- **si (proto.D=0) y se envía el paquete desde una red ip externa:** *src\_addr* es la combinación de ip y puerto con el cual se establece el *socket*.

- **ot\_len (16 bits):** longitud total del campo de *opción* del encabezado *Mesh*, incluyendo los dos bytes de este campo. Se utiliza la estructura *mesh\_header\_option\_header\_type* para definir el formato.

```

1 struct mesh_header_option_header_type {
2     uint16_t ot_len;
3     struct mesh_header_option_format option_list[0];
4 } __packed;
```

- **option\_list:** tiene tres campos que establecen el formato de la *opción*. Se utiliza la estructura *mesh\_header\_option\_format*.

```

1 struct mesh_header_option_format {
2     uint8_t otype;          // tipo de opcion
3     uint8_t olen;           // longitud de la opcion
4     uint8_t ovalue[0];      // valor de la opcion
5 } __packed;
```

- **Packet Body:** datos del usuario. Su longitud también es variable y depende de la cantidad de bytes que ocupe el encabezado dentro del paquete *Mesh*.

EL *SDK NON-OS* provee de API's para la creación de paquetes y encabezados *Mesh*, a continuación se presentan los prototipos de algunas de estas.

```

1 // crea un nuevo paquete mesh segun los parametros suministrados
2 void *espconn_mesh_create_packet( uint8_t *dst_addr , uint8_t *src_addr , bool
3 p2p , bool piggyback_cr ,
4 enum mesh_usr_proto_type proto , uint16_t data_len , bool option , uint16_t
5 ot_len , bool frag , enum
6 mesh_option_type frag_type , bool mf, uint16_t frag_idx , uint16_t frag_id )
7
8 // obtiene cuerpo del paquete mesh
9 bool espconn_mesh_get_usr_data( struct mesh_header_format *head , uint8_t **
10 usr_data , uint16_t *data_len )
11
12 // agrega data de usuario
13 bool espconn_mesh_set_usr_data( struct mesh_header_format *head , uint8_t *
14 usr_data , uint16_t data_len )
15
16 // obtiene protocolo utilizado en data de usuario
17 bool espconn_mesh_get_usr_data_proto( struct mesh_header_format *head , enum
18 mesh_usr_proto_type
19 *proto )
20
21 // identifica el protocolo usado por data de usuario (NONE, JSON, HTTP, MQTT,
22 BIN)
23 bool espconn_mesh_set_usr_data_proto( struct mesh_header_format *head , enum
24 mesh_usr_proto_type
25 proto )
26
27 // envia paquete a traves de la red
28 int8_t espconn_mesh_sent( struct espconn *usr_esp , uint8 *pdata , uint16 len )

```

### 3.7.1. Opciones del paquete *Mesh*

Las *opciones* son tramas que sirven para control de topología y/o caracterización de un paquete. En el *SDK NON-OS* con compatibilidad *Mesh*, se define la enumeración *mesh\_option\_type* con la cual se establecen los tipos de *opciones* disponibles. La figura 3.10 ilustra algunas de las tramas para cada tipo de opción.

```

1 enum mesh_option_type {

```

```

2   M_O_CONGEST_REQ = 0, // solicitud de ruta posible
3   M_O_CONGEST_RESP, // respuesta a solicitud de ruta posible
4   M_O_ROUTER_SPREAD, // difundir informacion del routerAP
5   M_O_ROUTE_ADD, // nodo entra a la red mesh
6   M_O_ROUTE_DEL, // nodo sale de la red mesh
7   M_O_TOPO_REQ, // solicitud de topologia
8   M_O_TOPO_RESP, // respuesta a solicitud de topologia
9   M_O_MCAST_GRP, // conjunto de nodos en direccion multicast
10  M_O_MESH_FRAG, // manejo de fragmentacion en red mesh
11  M_O_USR_FRAG, // fragmentacion de data del usuario
12  M_O_USR_OPTION, }; // opcion disponible para el usuario

```

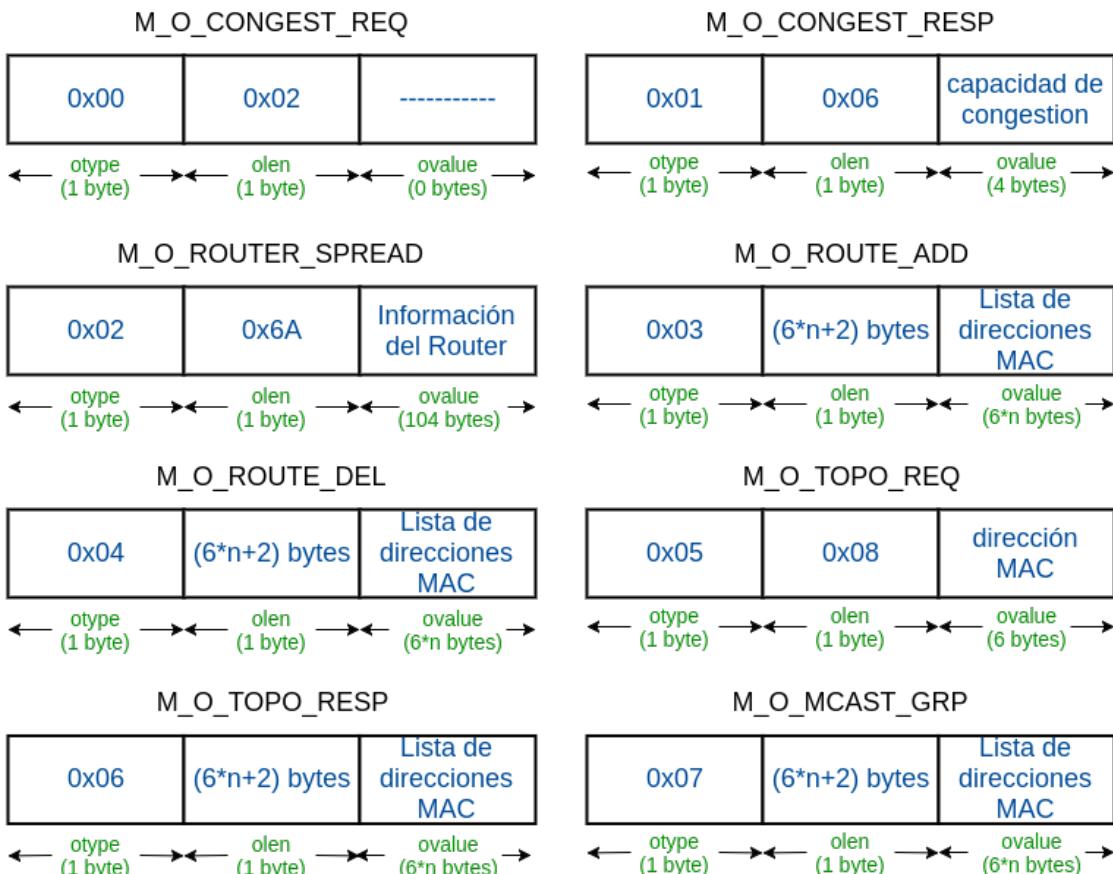


Figura 3.10: Estructura de tramas dependiendo del tipo de *opción* en el encabezado *Mesh*

Según lo detallado en la figura 3.10, comúnmente se desea agregar direcciones de otros nodos en el campo *o\_value* de la estructura *mesh\_header\_option\_format*. Por consecuente, es imperativo saber la cantidad de direcciones MAC que pueden añadirse a un mismo paquete *Mesh*. En el archivo *user\_config.h* se declaran las variables que definen este valor. De esta manera, no se estaría utilizando el correspondiente paquete para transportar datos del usuario, sino opciones de control de topología y configuración del mensaje. La figura 3.11 ilustra la estructura del campo de *opciones*.

dentro del encabezado *Mesh*.

```

1 #define ESP_MESH_ADDR_LEN          (6)
2 #define ESP_MESH_OPTION_MAX_LEN    (255)
3 #define ESP_MESH_PKT_LEN_MAX      (1300)
4 #define ESP_MESH_HLEN             (sizeof( struct mesh_header_format ))
5 #define ESP_MESH_OPTION_HLEN       (sizeof( struct mesh_header_option_format ))
6 #define ESP_MESH_OP_MAX_PER_PKT   ((ESP_MESH_PKT_LEN_MAX - ESP_MESH_HLEN) /
7                                ESP_MESH_OPTION_MAX_LEN)
7 #define ESP_MESH_DEV_MAX_PER_OP    ((ESP_MESH_OPTION_MAX_LEN -
8                                ESP_MESH_OPTION_HLEN) / ESP_MESH_ADDR_LEN)
8 #define ESP_MESH_DEV_MAX_PER_PKT   (ESP_MESH_OP_MAX_PER_PKT *
9                                ESP_MESH_DEV_MAX_PER_OP)

```

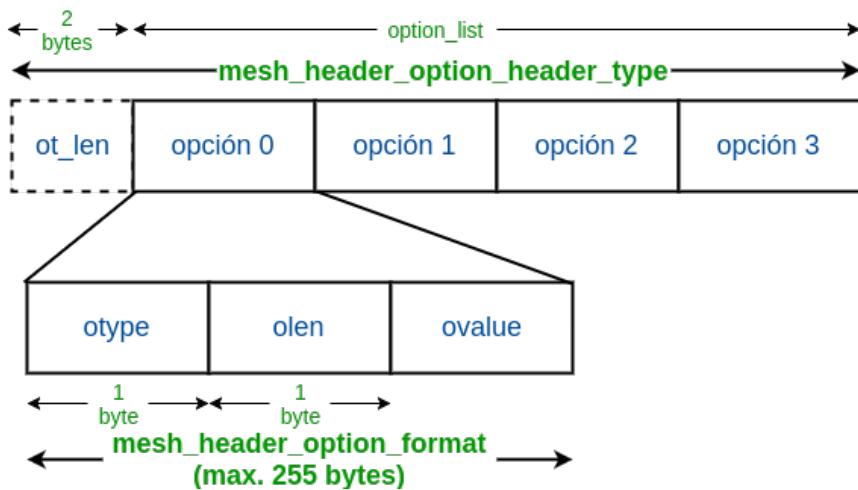


Figura 3.11: Estructura del campo de *opciones* en el encabezado *Mesh*

Las API's para la creación de *opciones* se mencionan a continuación. Estas se complementan con las de manejo de paquetes y encabezados, detalladas anteriormente, para generar la unidad de transporte básica de la red.

```

1 // crea opcion mesh segun el formato predeterminado
2 void *espconn_mesh_create_option(uint8_t otype , uint8_t *ovalue , uint8_t
    val_len )
3
4 // agrega al paquete mesh la opcion generada con espconn_mesh_create_option()
5 bool espconn_mesh_add_option( struct mesh_header_format *head , struct
    mesh_header_option_format *option )
6
7 // obtiene opcion contenida en paquete mesh
8 bool espconn_mesh_get_option( struct mesh_header_format *head , enum

```

```
mesh_option_type otype, uint16_t oidx, struct mesh_header_option_format *
option)
```

### 3.8. Protocolos compatibles a nivel de capa de aplicación

En el encabezado *Mesh* se definen cinco protocolos de capa de aplicación que pueden ser utilizados para establecer el formato de los datos de usuario. Se hace uso de la enumeración *mesh\_usr\_proto\_type* para indicar el protocolo deseado. Cada uno tiene su propia función de *parsing* que lee las tramas de datos según la sintaxis correcta. De los cinco disponibles, *M\_PROTO\_NONE* es usado para difundir mensajes de control de topología a través de la red de acuerdo al intervalo ya mencionado de 14000ms. Su función de *parsing* se encuentra en el archivo *mesh\_none.c* y es utilizada en paquetes que contengan respuesta a solicitud de topología a través de la opción *M\_O\_TOPO\_RESP*.

```
1 enum mesh_usr_proto_type {
2     M_PROTO_NONE = 0,      // para mensajes de control
3     M_PROTO_HTTP,         // Hypertext Transfer Protocol
4     M_PROTO_JSON,         // JavaScript Object Notation
5     M_PROTO_MQTT,         // Message Que Telemetry Transport
6     M_PROTO_BIN, } ;       // datos de usuario son tramas binarias
```

Los otros cuatro protocolos son divididos en dos modelos de acuerdo al tipo de datos en torno a los cuales están estructurados:

- **protocolos basados-en-texto:** se encuentran en formatos fáciles de lectura para un ser humano y son útiles en entornos donde se necesita su interpretación de forma rápida. Comúnmente los datos son encapsulados en tramas de caracteres en código ASCII (del inglés *American Standard Code for Information Interchange*). De este tipo de protocolos, es posible utilizar *JSON* [39] o *HTTP* [40].
- **protocolos binarios:** a diferencia de aquellos basados en textos, estos son enfocados en su lectura por parte de un computador. Son más eficientes en el encapsulamiento de los datos, lo que conlleva a beneficios como mayor velocidad de transmisión e interpretación. Ideales para dispositivos con recursos limitados o que necesiten almacenar y organizar grandes volúmenes de información. En la implementación *Mesh*, es posible utilizar *MQTT* [41] o formato *binario*.

### 3.9. Tipos de Mensajes

El fabricante provee de las funcionalidades necesarias para soportar cuatro tipos de mensajes. De manera demostrativa, *ESP\_MESH\_DEMO* explica como construir cada uno y utilizarlo en la aplicación respectiva. Se utiliza el protocolo de aplicación *JSON* para transportar cadenas de caracteres de un punto a otro de la red. Cada mensaje se transmite en intervalos específicos. En el archivo de configuración es posible cambiar estos períodos de tiempo cuyas unidades se encuentran en milisegundos. Sin embargo, se debe tomar en cuenta que tiempos muy próximos pueden incidir en los mensajes de control de topología y aumentar los tiempos de convergencia de la red.

```

1 static const uint32_t application_time      = 7000; // mensaje unicast
2 static const uint32_t json_p2p_test_time   = 19000; // mensaje p2p
3 static const uint32_t json_bcast_test_time = 18000; // mensaje broadcast
4 static const uint32_t json_mcast_test_time = 17000; // mensaje multicast

```

#### 3.9.1. Ejemplo de paquete de datos *unicast*

Los mensajes *unicast* son utilizados en la red mesh para enviar tramas de datos hacia dispositivos en redes externas. Una aplicación común es transmitir información de interés hacia servidores que almacenan, organizan y procesan estos datos. Existen numerosas plataformas *cloud*, que se detallarán en el capítulo siguiente, disponibles para estos propósitos. Se construye un mensaje *unicast* hacia un servidor con dirección ip y puerto predeterminados en el archivo de configuración. A continuación se ilustra este proceso en la figura 3.12.



Figura 3.12: Creación de mensaje *Unicast*

### 3.9.2. Ejemplo de paquete de datos *p2p*

Un mensaje *p2p* se utiliza para transmitir datos hacia otro nodo de la red. Esto constituye una herramienta fundamental para lograr comunicación M2M (del inglés *Machine-To-Machine*) dentro de la topología *Mesh*. La figura 3.13 ilustra el proceso de construcción del paquete p2p.

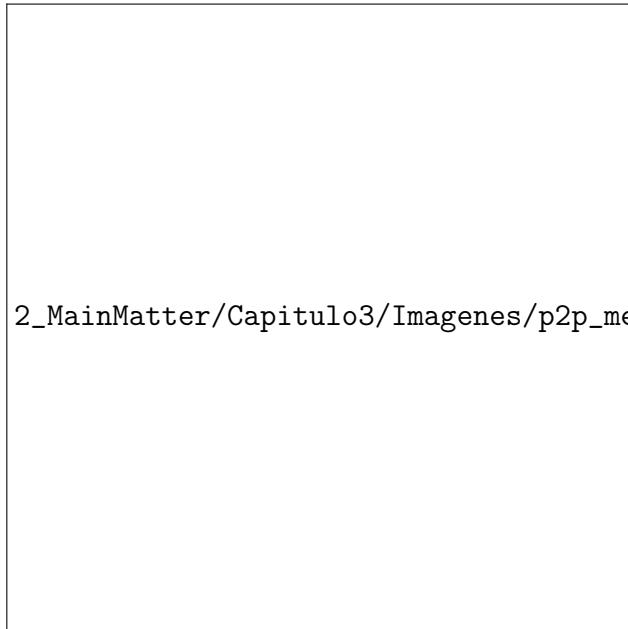
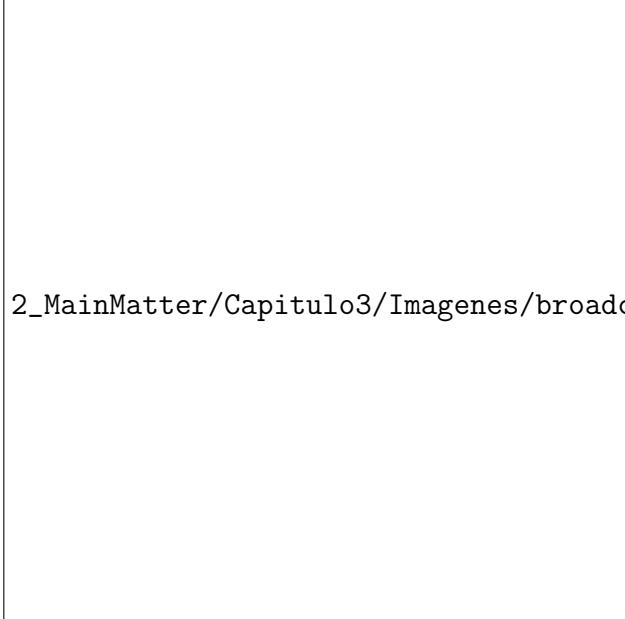


Figura 3.13: Creación de mensaje *P2P*

### 3.9.3. Ejemplo de paquete de datos *broadcast*

Los mensajes *broadcast* son de gran importancia para difundir información de control de topología a lo largo de la estructura. La trama de datos que se desea transmitir llegará a cada miembro de la red realizando múltiples saltos a través de los niveles.

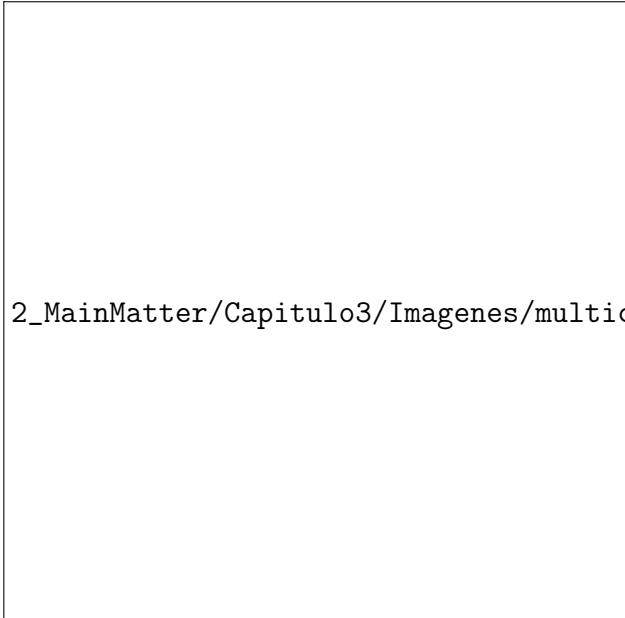


2\_MainMatter/Capitulo3/Imagenes/broadcast\_message.png

Figura 3.14: Creación de mensaje *Broadcast*

### 3.9.4. Ejemplo de paquete de datos *multicast*

Un mensaje *multicast* es útil para difundir datos a un conjunto de nodos dentro de la red, sin que esto signifique transmitir a todos como en el caso del mensaje *broadcast*. Se pueden crear sectores en la topología y asociar cada uno a un grupo *multicast*. La cantidad de direcciones que pueden añadirse está limitada por el tamaño máximo del paquete *Mesh*, fijado en 1300 bytes. Se utiliza la opción *M\_O\_MCAST\_GRP* para crear un nuevo grupo.



2\_MainMatter/Capitulo3/Imagenes/multicast\_message.png

Figura 3.15: Creación de mensaje *Multicast*

Adicionalmente, el apéndice A del presente trabajo de grado, incluye los códigos de programación utilizados para armar cada tipo de mensaje.

### 3.10. Resultados

Luego de detallar cada aspecto de la implementación, se procede a verificar el funcionamiento utilizando dos métodos distintos. En primer lugar, se ha añadido información de control que se transmite a través de la interfaz UART y se visualiza utilizando el terminal serial del computador. La utilización del módulo de desarrollo *NodeMCU* facilita estos propósitos debido al conversor USB-SERIAL que incorpora. Estos datos son de gran importancia para corroborar topología y tramas de datos que sean recibidas. En la figura 3.16 se observan los 4 módulos utilizados. Por último, se utiliza *Acrylic Wi-Fi* [42] para visualizar el espectro ISM de 2,4GHZ y las redes que se establecen en esta banda de frecuencia.

En el siguiente capítulo se desarrolla una aplicación basada en redes de sensores y la estructura *Mesh* del presente trabajo de grado. Esto constituye una primera aproximación para futuros desarrollos en *IoT* que utilicen *Mesh* como topología de red.



Figura 3.16: Módulos ESP8266 de NodeMCU utilizados para la implementación de la red

### 3.10.1. Mensajes de control a través de UART

A partir de la topología de la figura 3.7, se lee la información que es transmitida vía la interfaz UART en cada nodo de la bifurcación 1. Cada uno se identifica con su dirección MAC y da a conocer adicionalmente la información de su nodo *padre*, potencia recibida, el SSID de su propia BSS, cantidad de nodos *hijos*, canal *Wi-Fi* que está utilizando, intervalo de difusión de *beacons*, cantidad de módulos en la red, tabla de direcciones MAC y tabla de enrutamiento.

El nodo raíz, primer nivel de jerarquía, tiene en su tabla de enrutamiento cada punto de la topología. Las dos bifurcaciones se observan a continuación junto a su respectiva información de control.

```

1 // Nodo raiz
2 Hi ! I'm 18:fe:34:df:0f:4c with ip_address 10.42.0.14
3 My parent node is e0:2a:82:3c:f3:ad
4 with RSSI = -52 dBm
5 My SSID is MESH_DEMO_1_DF0F4C
6 I've 2 children nodes
7 I'm using WiFi channel 1
8 with a beacon interval of 5000 ms
9 cantidad de nodos en la red 4
10 ===== network mac list info =====
11 root -> 18:fe:34:df:0f:4c
12 child node 1 -> 5c:cf:7f:c1:1a:de
13 child node 2 -> 5c:cf:7f:86:93:5f
14 child node 3 -> 5c:cf:7f:86:91:f1
15 ===== network mac list end =====
16
17 ***** take the correct path ! *****
18
19 cidx:0, count:1 mac_list:
20 idx:0, 5c:cf:7f:86:91:f1
21
22 cidx:1, count:2 mac_list:
23 idx:0, 5c:cf:7f:86:93:5f
24 idx:1, 5c:cf:7f:c1:1a:de
25
26 ***** Have a nice trip :) *****

```

El nodo 1.1 tiene un sólo *hijo* con dirección MAC *5C:CF:7F:C1:1A:DE*. Si se desea enrutar un paquete hacia otra dirección, dentro o fuera de la red, por defecto se transmite hacia el nodo padre, en este caso *1A:FE:34:DF:0F:4C*. Este proceso ocurre repetidas veces hasta llegar al nodo raíz en el cual se toma la decisión final.

```

1 // Nodo 1.1
2 Hi ! I'm 5c:cf:7f:86:93:5f with ip_address 2.255.255.3
3 My parent node is 1a:fe:34:df:0f:4c
4 with RSSI = -61 dBm
5 My SSID is MESH_DEMO_2_86935F
6 I've 1 children nodes
7 I'm using WiFi channel 1
8 with a beacon interval of 5000 ms
9 cantidad de nodos en la red 4
10 ===== network mac list info =====
11 root -> 18:fe:34:df:0f:4c
12 child node 1 -> 5c:cf:7f:c1:1a:de
13 child node 2 -> 5c:cf:7f:86:93:5f
14 child node 3 -> 5c:cf:7f:86:91:f1
15 ===== network mac list end =====
16
17 ***** take the correct path ! *****
18
19 cidx:0, count:1 mac_list:
20 idx:0, 5c:cf:7f:c1:1a:de
21
22 ***** Have a nice trip :) *****

```

En el tercer nivel de la red, el nodo 1.1.1 constituye una *hoja* dentro de la topología jerárquica. De esta manera, no se tienen direcciones dentro de la tabla de enrutamiento; todo paquete, indiferentemente de su destino, es transmitido hacia el nodo *padre*. El primer dígito de la dirección ip indica el nivel respectivo.

```

1 // Nodo 1.1.1
2 Hi ! I'm 5c:cf:7f:c1:1a:de with ip_address 3.255.255.2
3 My parent node is 5e:cf:7f:86:93:5f
4 with RSSI = -60 dBm
5 My SSID is MESH_DEMO_3_C11ADE
6 I've 0 children nodes
7 I'm using WiFi channel 1
8 with a beacon interval of 5000 ms
9 cantidad de nodos en la red 4
10 ===== network mac list info =====
11 root -> 18:fe:34:df:0f:4c
12 child node 1 -> 5c:cf:7f:86:91:f1
13 child node 2 -> 5c:cf:7f:86:93:5f
14 child node 3 -> 5c:cf:7f:c1:1a:de
15 ===== network mac list end =====
16
17 ***** take the correct path ! *****

```

```

18
19 ***** Have a nice trip :) *****

```

Cuando un nodo de desconecta en algún nivel de la estructura, su nodo padre transmite información de interés vía UART y la dirección sale de su tabla de enrutamiento. En el siguiente mensaje de control de topología, los módulos restantes adquieren esta información. A continuación se ilustra el caso en el cual el nodo 1.1.1, con dirección MAC *5C:CF:7F:C1:1A:DE*, sale de la red.

```

1 disconn , ip:3.255.255.2 port:22127
2 station: 5c:cf:7f:c1:1a:de leave , AID = 1
3 rm 1
4 ap_discon , RIP: 3.255.255.2RP = 22127

```

En modo *MESH\_ONLINE*, detallado anteriormente en el actual capítulo, cada nodo establece una conexión TCP con un servidor con dirección ip y puerto especificados en el archivo de configuración. Cada 7000ms, se envía un mensaje *unicast* que, al llegar a su destino, es transmitido de vuelta sin ninguna modificación hacia el nodo de origen. Este paquete contiene la dirección MAC y dirección ip del módulo. La función de *parsing* del protocolo *JSON* adquiere la información respectiva y la transmite por la interfaz UART.

```

1 // mensaje retransmitido desde el servidor hacia el nodo raiz
2 packet received
3 mesh_json_proto_parser is running
4 mesh package from 0a:2a:00:01:58:1b to 18:fe:34:df:0f:4c
5 data: Hello World ! I'm 18:fe:34:df:0f:4c with ip address 10.42.0.14
6
7 // mensaje retransmitido desde el servidor hacia el nodo 1.1
8 packet received
9 mesh_json_proto_parser is running
10 mesh package from 0a:2a:00:01:58:1b to 5c:cf:7f:86:93:5f
11 data: Hello World ! I'm 5c:cf:7f:86:93:5f with ip address 2.255.255.3
12
13 // mensaje retransmitido desde el servidor hacia el nodo 1.1.1
14 packet received
15 mesh_json_proto_parser is running
16 mesh package from 0a:2a:00:01:58:1b to 5c:cf:7f:c1:1a:de
17 data: Hello World ! I'm 5c:cf:7f:c1:1a:de with ip address 3.255.255.2

```

También se generan mensajes *multicast*, *broadcast* y *p2p* hacia nodos específicos en la red según lo establecido en los ejemplos anteriormente expuestos. A continuación se observa lo recibido por el terminal serial cuando uno de estos tres tipos de mensajes llega a un nodo. La función de *parsing* del protocolo *JSON* indica las direcciones origen y destino de cada paquete. Se observan

las direcciones respectivas de *multicast* y *broadcast*.

```

1 // mensaje P2P con protocolo JSON
2 packet received
3 mesh_json_proto_parser is running
4 mesh package from 5c:cf:7f:86:93:5f to 5c:cf:7f:86:93:5f
5 data: {"using json p2p protocol":"5c:cf:7f:86:93:5f"}
6
7 // mensaje multicast con protocolo JSON
8 packet received
9 mesh_json_proto_parser is running
10 mesh package from 5c:cf:7f:c1:1a:de to 01:00:5e:00:00:00
11 data: {"mcast":"5c:cf:7f:c1:1a:de"}
12
13 // mensaje broadcast con protocolo JSON
14 packet received
15 mesh_json_proto_parser is running
16 mesh package from 5c:cf:7f:86:93:5f to 00:00:00:00:00:00
17 data: {"using json bcast protocol":"5c:cf:7f:86:93:5f"}
```

Adicionalmente, se monitorea en intervalos específicos, el espacio disponible en memoria para datos del usuario. La cantidad de nuevos nodos que pueden adherirse a la topología dependen de este valor. A continuación se observa la memoria (en bytes) libre para cada nodo de la bifurcación 1. El nodo raíz, que almacena la topología completa, tiene el menor espacio disponible.

```

1 // Nodo raiz
2 Espacio libre en el stack de memoria: 38512
3 // Nodo 1.1
4 Espacio libre en el stack de memoria: 40392
5 // Nodo 1.1.1
6 Espacio libre en el stack de memoria: 41864
```

### 3.10.2. Análisis del la banda ISM de 2,4GHz con Acrylic Wi-Fi

La figura 3.17 muestra las redes *Wi-Fi* disponibles. Debido a la característica de canal-único en frecuencia que utiliza esta implementación, las 5 redes de interés que se muestran en la imagen, asociadas a la red *Mesh*, utilizan el mismo canal *Wi-Fi* (canal 1) definido por el enlace entre el routerAP y el nodo raíz. Sólo este último tiene un SSID que no está escondido y definido según la estructura detallada a principios del capítulo: (MESH\_SSID\_PREFIX)+(NIVEL)+(6DIG\_MAC\_ADDRESS). Se observan las direcciones MAC de cada nodo según la topología de la figura 3.7. Finalmente, la imagen detalla las versiones de 802.11 y el modo de seguridad que utiliza la red.

Un análisis previo en la banda de frecuencias de 2,4 GHz puede conllevar a la selección de

un canal *Wi-Fi* con menos interferencia. Por consecuente, mejorará la tasa efectiva de la red al disminuir los tiempos de espera para ocupar el medio según lo establecido por el protocolo MAC de 802.11 (CSMA-CA).

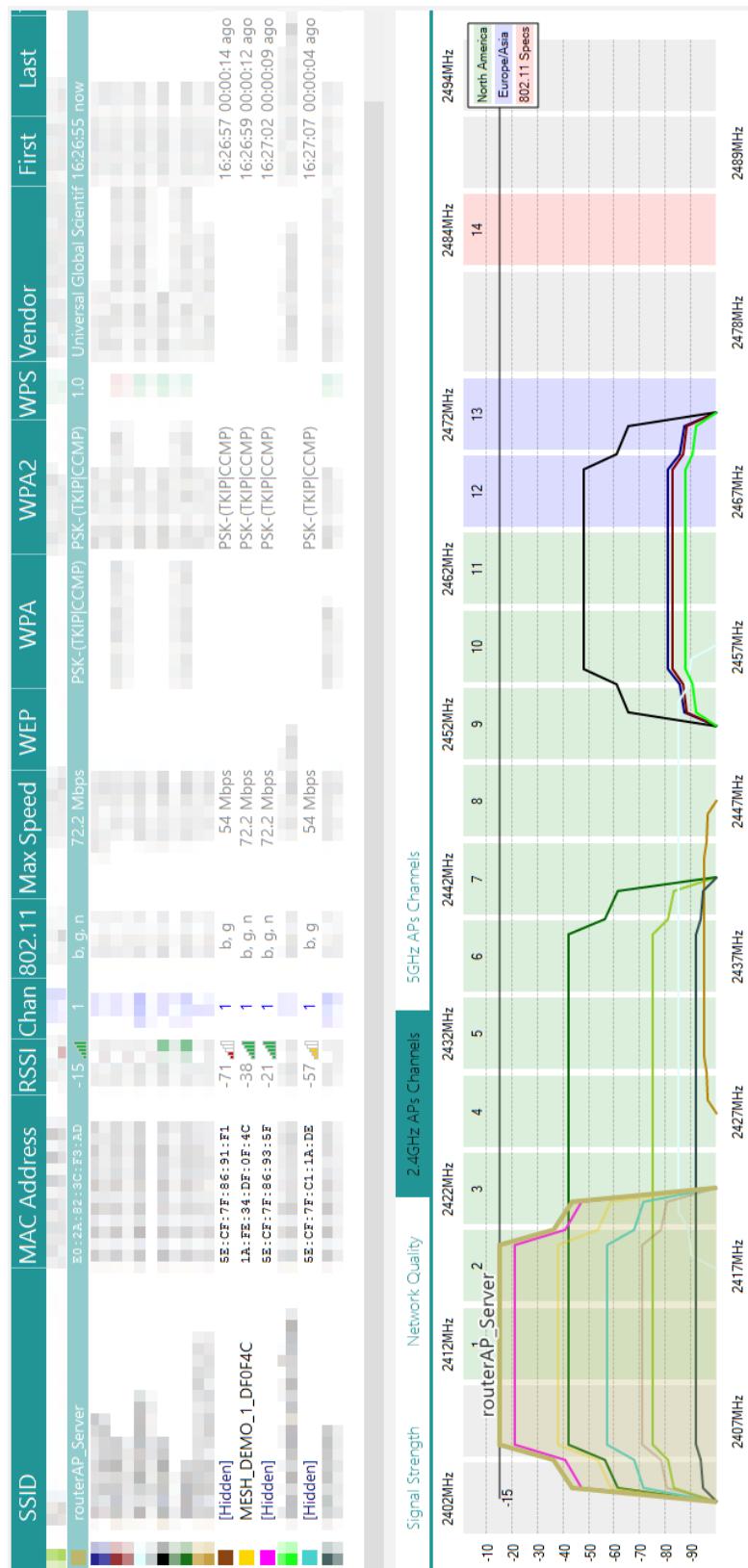


Figura 3.17: Redes Wi-Fi disponibles en la banda ISM de 2,4 GHz

# CAPÍTULO 4

## IMPLEMENTACIÓN DE RED DE SENSORES BASADA EN ESP-MESH

Si buscas resultados distintos no hagas  
siempre lo mismo

---

*Albert Einstein*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### 4.1. Criterios de diseño

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### 4.2. Configuración inicial

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **4.3. Interfaz con Sensores**

Lore ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdier mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lore ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **4.4. Modo *Mesh Online* con servidor local**

Lore ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdier mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lore ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **4.5. Modo *Mesh Online* con plataforma *Cloud***

Lore ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdier mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lore ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

#### **4.5.1. Acerca de *ThingSpeak***

*ThingSpeak* es una plataforma para IoT que permite almacenamiento y análisis de datos provenientes de distintas fuentes. Es compatible con *Matlab*, lo que permite útiles herramientas de procesamiento. Dispone de dos tipos de API's para actualización y adquisición desde y hacia canales específicos: REST y MQTT. En la implementación del presente trabajo de grado se ha utilizado la segunda.

#### 4.5.1.1. MQTT API's

*ThingSpeak* permite actualizar canales bajo el modelo publicador-subscriptor que establece MQTT. Puede establecer conexiones a través de TCP sockets o WebSockets y soporta protocolos de encriptación. La siguiente tabla identifica las configuraciones de cliente y los puertos utilizados.

Tabla 4.1: Posibles configuración en clientes MQTT

Puerto	Tipo de Conexión	Encriptación
1883	TCP	None
8883	TCP	TLS / SSL
80	WebSocket	None
443	WebSocket	TLS / SSL

Según las especificaciones de MQTT [41], se debe construir el tópico según la estructura que se presenta a continuación. *ChannelID* y *apiKey* son los parámetros determinados por la plataforma de *ThingSpeak*.

```
1 channels/<channelID>/publish/<apikey>
```

El payload del mensaje *PUBLISH* debe poseer los argumentos necesarios y sus respectivos valores separados con el símbolo &. De manera ilustrativa, se coloca un ejemplo de la estructura necesaria:

```
1 field1=100&field2=50&field3=75&lat=30.61&long=40.35
```

Los campos disponibles se detallan a continuación:

- **field1 hasta field8:** datos que se requieren enviar al canal correspondiente.
- **lat:** latitud para geolocalización del canal.
- **long:** longitud para geolocalización del canal.
- **elevation:** elevación del canal en metros.
- **status:** mensaje de estatus.
- **twitter:** cuenta de usuario en *twitter* afiliada a *ThingTweet*.
- **tweet:** actualización de estatus en cuenta de *twitter*.

- **created\_at:** fecha y hora del mensaje en formato ISO 8601.
- **timezone:** zona horaria en la cual se localiza el canal.

#### 4.5.2. Recolección y análisis de datos a través de *ThingSpeak*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

# CAPÍTULO 5

## FUTUROS DESARROLLOS

La web, tal como yo la imaginaba, todavía no la hemos visto. El futuro sigue siendo mucho más grande que el pasado

---

*Tim Berners-Lee*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### 5.1. Redes *Mesh* basadas en el módulo nrf24l01

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### 5.2. Redes *Mesh* basadas en BLE

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **5.3. Redes *Mesh* basadas en dispositivos multi-protocolos**

  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **5.4. Protocolos de enrutamiento para redes ad hoc con eficientes consideraciones de potencia**

  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

### **5.5. Protocolos de enrutamiento para redes ad hoc a través de algoritmos de *Machine Learning***

  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

## **CONCLUSIONES Y RECOMENDACIONES**

*Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.*

## REFERENCIAS

- [1] Statista. *Internet of Things (IoT): number of connected devices worldwide from 2012 to 2020 (in billions)*. Consultado el 07 de enero de 2017. Disponible en Internet: <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.
- [2] Inc. Link Labs. *Selecting a Wireless technology for New Industrial Internet of Things Products: A Guide for Engineers and Decision Makers*. 130 Holiday Court, Suite 100, Annapolis, MD 21401, 2016.
- [3] OpenTech Diary. *Part 5 : A walk through Internet of Things*. Consultado el 07 de enero de 2017. Disponible en Internet: <https://opentechdiary.wordpress.com/2015/07/22/part-5-a-walk-through-internet-of-things-iot-basics/>.
- [4] Computer Science Learning Center. *Network Topologies*. Consultado el 05 de enero de 2017. Michigan Technological University. Disponible en Internet: <http://www.csl.mtu.edu/cs4451/www/notes/Network%20Topologies.pdf>.
- [5] P. Pradhan. “Wireless Mesh Network”. Tesis de mtría. National Institute of Science y Technology, 2013.
- [6] D. Johnson, K. Matthee, D. Sokoya, L. Mboweni, A. Makan y H. Kotze. *Building a Rural Wireless Mesh Network*. Versión 0.8. Wireless Africa, Meraka Institute. Oct. de 2007.
- [7] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu y J Jetcheva. “A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols”. En: *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking* (oct. de 1998).
- [8] T. Clausen y P. Jacquet. “Optimized Link State Routing Protocols for Ad Hoc Networks”. En: (oct. de 2003).
- [9] IEEE Computer Society. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Standards Association, 2012.
- [10] R. Badra. *Redes Inalámbricas de Área Local*. EC5451. Redes Inalámbricas de Datos. Valle de Sartenejas, Caracas. Universidad Simón Bolívar., sep. de 2016.
- [11] T. Nitsche, C. Cordeiro, A. Flores, E. W. Knightly, E. Perahia y J. C. Widmer. “IEEE 802.11ad: Directional 60 GHz Communication for Multi-Gbps Wi-Fi”. En: () .
- [12] M. Barrett. *Now That 802.11ac Wave 1 is Rolled Out, What's Next for Wi-Fi ?* Consultado el 08 de enero de 2017. BluWireless Technology. Disponible en Internet: <https://www.wirelessdesignmag.com/article/2016/05/now-80211ac-wave-1-rolled-out-whats-next-wi-fi>.
- [13] Inc. Bluetooth SIG. *Bluetooth Low Energy*. Consultado el 07 de enero de 2017. Disponible en Internet: <https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy>.
- [14] Zigbee Alliance. *What is ZigBee?* Consultado el 07 de enero de 2017. Disponible en Internet: <http://www.zigbee.org/what-is-zigbee/>.
- [15] M. Centenaro, L. Vangelista, A. Zanella y M. Zorzi. “Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios”. En: (sep. de 2015).

- [16] L. Columbus. *Roundup Of Internet of Things Forecasts And Market Estimates, 2015*. Consultado el 07 de enero de 2017. Forbes. Disponible en Internet: <http://www.forbes.com/sites/louiscolumbus/2015/12/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2015/#4d5c6fdc48a0>.
- [17] J. Manyika, M. Chui, P. Bisson, J. Woetze, R. Dobbs, J. Bughin y D. Aharon. *Unlocking the potential of the Internet of Things*. Consultado el 08 de enero de 2017. McKinsey & Company. Disponible en Internet: <http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world>.
- [18] W. S. Conner, J. Kruys, K. Kyeongsoo y J. C. Zuniga. “IEEE 802.11s Tutorial: Overview of the Amendment for Wireless Local Area Mesh Networking”. En: nov. de 2006.
- [19] M. Wielgosz. “IEEE 802.11s Multihop MAC”. En: AGH University of Science y Technology.
- [20] R. C. Amit. “StarHub deploys mesh network at NUS”. En: *Singapore Press Holdings* (sep. de 2016).
- [21] Espressif IOT Team. *ESP8266EX: Datasheet*. Espressif Systems. 2016.
- [22] Texas Instruments. *Universal Asynchronous Receiver/Transmitter (UART)*. Nov. de 2010.
- [23] J. Valdez y J. Becker. *Understanding the I2C Bus*. Texas Instruments. Jun. de 2015.
- [24] Philips Semiconductors. *I2S bus specification*. Jun. de 1996.
- [25] Ingeniería en Microcontroladores. *Protocolo SPI (Serial Peripherical Interface)*. México, Distrito Federal.
- [26] Technical Committee. *SD Specifications, Part E1: SDIO Simplified Specification*. 2.0. SD Card Association. Feb. de 2007.
- [27] Cadence Design Systems. *Tensilica Xtensa 10 Customizable Processor: Small, Ultra-Low-Power 32-bit Embedded Controller*.
- [28] Inc. Tensilica. *Xtensa: Instruction Set Architecture (ISA)*. Santa Clara, CA 95054, Abril de 2010.
- [29] NodeMCU. *NodeMCU Documentation*. Consultado el 10 de agosto de 2016. Disponible en Internet: <https://nodemcu.readthedocs.io/en/master/>.
- [30] NodeMCU. *NodeMCU pin definition*. Consultado el 10 de agosto de 2016. Disponible en Internet: [https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/Documents/NODEMCU\\_DEVKIT\\_V1.0\\_PINMAP.png](https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/Documents/NODEMCU_DEVKIT_V1.0_PINMAP.png).
- [31] Espressif Systems IOT Team. *ESP8266 AT Instruction Set*. 1.5.4. Espressif Systems. 2016.
- [32] Espressif Systems IOT Team. *ESP8266 RTOS SDK Programming Guide*. 1.4. Espressif Systems. 2016.
- [33] FreeRTOS. *The FreeRTOS Reference Manual*. 9.0.0. Real Time Engineers Ltd. 2016.
- [34] Espressif IOT Team. *ESP8266 Non-OS SDK: API Reference*. Espressif Systems. 2016.
- [35] Xtensa. *crosstool-NG*. Consultado el 15 de septiembre de 2016. Disponible en Internet: <https://github.com/jcmvbkb/crosstool-NG/tree/ecfc19a597d76c0eea65148b08d7ccb505cdcac6>.
- [36] Espressif IOT Team. *esptool*. Consultado el 15 de septiembre de 2016. Espressif Systems. Disponible en Internet: <https://github.com/espressif/esptool>.

- [37] Espressif IOT Team. *ESP8266: Mesh API's*. Feb. de 2016.
- [38] Espressif IOT Team. *ESP8266: Mesh User Guide*. Espressif Systems. 2016.
- [39] ECMA international. *The JSON Data Interchange Format*. Consultado el 18 de enero de 2017. Oct. de 2013.
- [40] R. Fielding, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach y T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. The Internet Society, jun. de 1999.
- [41] R. J. Cohn, R. J. Coppen, A. Banks y Gupta R. *MQTT*. OASIS, oct. de 2014.
- [42] Tarlogic Security SL. *Acrylic Wi-Fi: Quick Help*. Abril de 2016.

## APÉNDICE A:

### Código de Programación Embebido en el ESP8266

A continuación se coloca segmentos del código de programación con el cual se implementa la red *mesh*. Como se mencionó durante el capítulo 3, el proyecto completo se podrá encontrar en la plataforma [github.com/LPerezBustos](https://github.com/LPerezBustos).

#### Selección de un nodo como raíz

```
1 void ICACHE_FLASH_ATTR mesh_device_set_root( struct mesh_device_mac_type *root )
2 {
3     if (!g_mesh_device_init) // bandera
4         mesh_device_list_init(); // funcion de inicializacion
5     // si no existe nodo raiz --> g_node_list.scale == 0
6     if (g_node_list.scale == 0){
7         MESH_DEMO_PRINT("new root:" MACSTR "\n", MAC2STR((uint8_t *)root));
8         MESH_DEV_MEMCPY(&g_node_list.root, root, sizeof(*root)); // direccion
9         MAC del nodo raiz en informacion de la red
10        g_node_list.scale = 1; // primer nodo de la red
11        return;
12    }
13    // nodo raiz es el mismo que ya ejerce esta funcion
14    if (!MESH_DEV_MEMCMP(&g_node_list.root, root, sizeof(*root)))
15        return;
16    // se cambia el nodo raiz
17    MESH_DEMO_PRINT("switch root: " MACSTR " to root: " MACSTR "\n",
18                    MAC2STR((uint8_t *)&g_node_list.root), MAC2STR((uint8_t *)root));
19    mesh_device_list_release(); // se libera tabla de direcciones MAC
20    MESH_DEV_MEMCPY(&g_node_list.root, root, sizeof(*root)); // direccion MAC
21    del nodo raiz en informacion de red
22    g_node_list.scale = 1;} // primer nodo de la red
```

#### Adquisición de topología por parte del nodo raíz

```
1 if (espconn_mesh_is_root()) { // verifica si nodo actual es "ROOT"
2     uint8_t *sub_dev_mac = NULL;
3     uint16_t sub_dev_count = 0;
4     if (!espconn_mesh_get_node_info(MESH_NODE_ALL, &sub_dev_mac, &
5         sub_dev_count))
6         return; // MESH_NODE_ALL pregunta a todos los nodos de la red
7     mesh_device_set_root(( struct mesh_device_mac_type *)src); // nodo actual
8     es ROOT_NODE
```

```

7   if (sub_dev_count > 1){ // si hay otros nodos , se completa
8     MAC_ADDRESS_LIST
9       struct mesh_device_mac_type *list = (struct mesh_device_mac_type *)
10    sub_dev_mac;
11    mesh_device_add(list + 1, sub_dev_count - 1);}
12  mesh_device_disp_mac_list(); // mostrar MAC_ADDRESS_LIST del nodo raiz
13 espconn_mesh_get_node_info(MESH_NODE_ALL, NULL, NULL);
14 return ;

```

## Adquisición de topología por parte de otros nodos de la red

```

1 // direccion de broadcast = 00:00:00:00:00:00
2 os_memset(dst, 0, sizeof(dst));
3 // se genera nuevo encabezado MESH
4 header = (struct mesh_header_format *)espconn_mesh_create_packet(
5           dst,          // direccion de destino (bcast)
6           src,          // direccion de origen
7           false,         // not p2p packet
8           true,          // piggyback congest request
9           M_PROTO_NONE, // paquete con formato "NONE"
10          0,            // longitud de data
11          true,         // no option
12          ot_len,        // option total length
13          false,         // fragmentation
14          0,            // fragmentation type
15          false,         // more fragmentation
16          0,            // fragmentation index
17          0);          // fragmentation length
18 if (!header){
19   MESH_PARSER_PRINT("create packet in topology parser have failed\n");
20   return ;
21 // se crea opcion de solicitud de topologia
22 option = (struct mesh_header_option_format *)espconn_mesh_create_option(
23   M_O_TOPO_REQ, dst, sizeof(dst));
24 if (!option) {
25   MESH_PARSER_PRINT("create option in topology parser have failed\n");
26   goto TOPO_FAIL;}
27 // se agrega la opcion al encabezado MESH
28 if (!espconn_mesh_add_option(header, option)){
29   MESH_PARSER_PRINT("set option in topology parser have failed\n");
30   goto TOPO_FAIL;}
31 // se envia paquete con solicitud de topologia
32 if (espconn_mesh_sent(&g_ser_conn, (uint8_t *)header, header->len)){
33   MESH_PARSER_PRINT("topology parser is busy right now\n");
34   espconn_mesh_connect(&g_ser_conn);}

```

```

35 TOPO_FAIL: // se libera el espacio de memoria
36     option ? MESH_DEMO_FREE(option) : 0;
37     header ? MESH_DEMO_FREE(header) : 0;}
```

## Tipos de mensajes disponibles

### Construcción del mensaje *Unicast*

```

1 // funcion se almacena en memoria flash externa
2 void ICACHE_FLASH_ATTR esp_mesh_demo_test(){
3     uint8_t src[6]; // direccion de origen
4     uint8_t dst[6]; // direccion de destino
5     struct mesh_header_format *header = NULL; // encabezado mesh
6
7     // La data de usuario puede ser string-based (HTTP/JSON) o binary-based (MQTT/BIN)
8     char *tst_data = "{\"Hello World !\"}\r\n"; // string-based
9     // es importante ver el espacio disponible en memoria
10    MESH_DEMO_PRINT("\nEspacio libre en el stack de memoria: %u \n",
11                  system_get_free_heap_size());
12
13    if (!wifi_get_macaddr(STATION_IF, src)) {
14        MESH_DEMO_PRINT("get station MAC_ADDRESS have failed\n");
15        return; // direccion MAC de origen en arreglo "src"
16    // proto.D == 1 (upwards) && proto.p2p == 0 (no es p2p)
17    MESH_DEMO_MEMCPY(dst, server_ip, sizeof(server_ip)); // dst == ip+port
18    MESH_DEMO_MEMCPY(dst + sizeof(server_ip), &server_port, sizeof(server_port));
19    header = (struct mesh_header_format *)espconn_mesh_create_packet(
20              dst, // destino: server_ip + server_port
21              src, // origen: STATION_IF
22              false, // no es p2p
23              true, // solicitud de piggyback
24              M_PROTO_JSON, // protocolo de aplicacion JSON
25              MESH_DEMO_STRLEN(tst_data), // longitud de trama de datos
26              false, // no hay opciones mesh
27              0, // longitud de opciones mesh
28              false, // no hay fragmentacion
29              0, // no hay fragmentacion
30              false, // no hay fragmentacion
31              0, // no hay fragmentacion
32              0); // no hay fragmentacion
33    if (!header) { // comprobacion
34        MESH_DEMO_PRINT("create packet have failed\n");
35        return;}
```

```

35 // se agrega data de usuario al mensaje broadcast
36 if (!espconn_mesh_set_usr_data(header, tst_data, MESH_DEMO_STRLEN(tst_data)
37 )) {
38     MESH_DEMO_PRINT("set user data have failed\n");
39     MESH_DEMO_FREE(header);
40     return;
41 }
42 // enviar mensaje a traves de la red mesh
43 if (espconn_mesh_sent(&g_ser_conn, (uint8_t *)header, header->len)) {
44     MESH_DEMO_PRINT("ucast mesh is busy\n");
45     MESH_DEMO_FREE(header);
46     espconn_mesh_connect(&g_ser_conn); // if fail, we re-connect mesh
47     return;
48 }
49 // se libera espacio de memoria
50 MESH_DEMO_FREE(header);

```

## Construcción del mensaje P2P

```

1 // funcion se almacena en memoria flash externa
2 void ICACHE_FLASH_ATTR mesh_json_p2p_test(){
3     uint16_t idx;
4     char buf[64]; // espacio para datos de usuario
5     uint8_t src[6]; // direccion de origen
6     uint8_t dst[6]; // direccion de destino
7     uint16_t dev_count = 0; // cantidad de nodos en la red
8     struct mesh_header_format *header = NULL; // encabezado mesh
9     struct mesh_device_mac_type *list = NULL; // lista de direcciones MAC
10
11    if (!wifi_get_macaddr(STATION_IF, src)) {
12        MESH_PARSER_PRINT("get station_mac_address of current node have failed
13 \n");
14        return; // direccion MAC de origen en arreglo "src"
15
16    if (!mesh_device_get_mac_list((const struct mesh_device_mac_type **)&list,
17 &dev_count)) {
18        MESH_PARSER_PRINT("p2p get mac_address_list have failed\n");
19        return; // informacion de todos los nodos de la red
20
21    // si dev_count > 1 , se selecciona nodo destino de forma aleatoria
22    // si dev_count == 1 , nodo destino es nodo raiz
23    idx = dev_count > 1 ? (os_random() % (dev_count + 1)) : 0;
24    if (!idx) { // idx == 0
25        if (!mesh_device_get_root((const struct mesh_device_mac_type **)&list))
26            return; // mesh_device_get_root ha fallado
27        os_memcpy(dst, list, sizeof(dst)); // se selecciona nodo raiz como

```

```

destino
26 } else { // idx != 0
27     os_memcpy(dst, list + idx - 1, sizeof(dst)); // destino aleatorio
28 }

29
30 // UART debugging
31 MESH_PARSER_PRINT("%s is running\n", __func__);
32 MESH_PARSER_PRINT("sending json p2p package to... " MACSTR "\n", MAC2STR(
33 dst));
34 // data de usuario
35 os_memset(buf, 0, sizeof(buf)); // espacio en memoria
36 os_sprintf(buf, "%s", "{\"using json p2p protocol\":\"\""); // mensaje
37 os_sprintf(buf + os_strlen(buf), MACSTR, MAC2STR(src)); // mensaje + MAC
38 os_sprintf(buf + os_strlen(buf), "%s", "\"}\r\n"); // fin de linea

39 header = (struct mesh_header_format *)espconn_mesh_create_packet(
40         dst, // destino: aleatorio
41         src, // origen: STATION_IF
42         true, // es un mensaje p2p
43         true, // solicitud de piggyback
44         M_PROTO_JSON, // protocolo de aplicacion JSON
45         os_strlen(buf), // longitud de trama de datos
46         false, // no hay opciones mesh
47         0, // longitud de opciones mesh
48         false, // no hay fragmentacion
49         0, // no hay fragmentacion
50         false, // no hay fragmentacion
51         0, // no hay fragmentacion
52         0); // no hay fragmentacion
53 if (!header) { // comprobacion
54     MESH_PARSER_PRINT("p2p create packet have failed\n");
55     return;
56 }
57 // se agrega data de usuario al mensaje broadcast
58 if (!espconn_mesh_set_usr_data(header, buf, os_strlen(buf))) {
59     MESH_DEMO_PRINT("p2p set user data have failed\n");
60     MESH_DEMO_FREE(header); // free header memory
61     return;
62 }
63 // enviar mensaje a traves de la red mesh
64 if (espconn_mesh_sent(&g_ser_conn, (uint8_t *)header, header->len)) {
65     MESH_DEMO_PRINT("p2p mesh is busy right now\n");
66     MESH_DEMO_FREE(header); // free header memory
67     espconn_mesh_connect(&g_ser_conn); // trying to connect again
68     return;
69 }
70 // se libera espacio de memoria
71 MESH_DEMO_FREE(header);

```

## Construcción del mensaje *Broadcast*

```

1 // funcion se almacena en memoria flash externa
2 void ICACHE_FLASH_ATTR mesh_json_bcast_test(){
3     char buf[64]; // espacio para datos de usuario
4     uint8_t src[6]; // direccion de origen
5     uint8_t dst[6]; // direccion de destino
6     struct mesh_header_format *header = NULL; // encabezado mesh
7
8     if (!wifi_get_macaddr(STATION_IF, src)) {
9         MESH_PARSER_PRINT("get station_mac_address of node failed\n");
10        return; // direccion MAC de origen en arreglo "src"
11
12    os_memset(buf, 0, sizeof(buf)); // espacio en memoria
13    os_sprintf(buf, "%s", "{\"using json bcast protocol\":\\\""); // mensaje
14    os_sprintf(buf + os_strlen(buf), MACSTR, MAC2STR(src)); // mensaje + MAC
15    os_sprintf(buf + os_strlen(buf), "%s", "\\}\r\n"); // fin de linea
16    os_memset(dst, 0, sizeof(dst)); // bcast_address = 00:00:00:00:00:00
17
18    header = (struct mesh_header_format *)espconn_mesh_create_packet(
19                dst, // destino: 00:00:00:00:00:00
20                src, // origen: STATION_IF
21                false, // no es p2p
22                true, // solicitud de piggyback
23                M_PROTO_JSON, // protocolo de aplicacion JSON
24                os_strlen(buf), // longitud de trama de datos
25                false, // no hay opciones mesh
26                0, // longitud de opciones mesh
27                false, // no hay fragmentacion
28                0, // no hay fragmentacion
29                false, // no hay fragmentacion
30                0, // no hay fragmentacion
31                0); // no hay fragmentacion
32    if (!header) { // comprobacion
33        MESH_PARSER_PRINT("bcast create packet have failed\n");
34        return; }
35    // se agrega data de usuario al mensaje broadcast
36    if (!espconn_mesh_set_usr_data(header, buf, os_strlen(buf))) {
37        MESH_DEMO_PRINT("bcast set user data have failed\n");
38        MESH_DEMO_FREE(header); // free header memory
39        return; }
40    // enviar mensaje a traves de la red mesh
41    if (espconn_mesh_sent(&g_ser_conn, (uint8_t *)header, header->len)) {
42        MESH_DEMO_PRINT("bcast_mesh is busy right now\n");
43        MESH_DEMO_FREE(header); // free header memory
44        espconn_mesh_connect(&g_ser_conn); // trying to connect again

```

```

45     return;
46 // se libera espacio de memoria
47 MESH_DEMO_FREE(header);
```

## Construcción del mensaje *Multicast*

```

1 // funcion se almacena en memoria flash externa
2 void ICACHE_FLASH_ATTR mesh_json_mcast_test() {
3     char buf[32]; // espacio para datos de usuario
4     uint8_t src[6]; // direccion de origen
5     uint16_t i = 0;
6     uint16_t ot_len = 0, op_count = 0; // parametros de "opciones mesh"
7     uint16_t dev_count = 0, max_count = 0;
8     struct mesh_header_format *header = NULL; // encabezado mesh
9     struct mesh_header_option_format *option = NULL; // formato de "opcion
mesh"
10    uint8_t dst[6] = {0x01, 0x00, 0x5E, 0x00, 0x00, 0x00}; // direccion
multicast
11    struct mesh_device_mac_type *list = NULL, *root = NULL; // lista de
direcciones MAC
12
13    if (!wifi_get_macaddr(STATION_IF, src)) {
14        MESH_PARSER_PRINT("get station_mac_address of current node have failed
\n");
15        return; // direccion MAC de origen en arreglo "src"
16
17    if (!mesh_device_get_mac_list((const struct mesh_device_mac_type **)&list ,
&dev_count)) {
18        MESH_PARSER_PRINT("mcast get mac_address_list have failed\n");
19        return; // informacion de todos los nodos de la red
20
21    if (!mesh_device_get_root((const struct mesh_device_mac_type **)&root)) {
22        MESH_PARSER_PRINT("mcast get root have failed\n");
23        return; // direccion MAC del nodo raiz
24    dev_count++;
25    // el paquete mesh tiene un limite ,
26    // si es posible , se seleccionan todos los nodos de la red en grupo mcast
27    os_memset(buf, 0, sizeof(buf)); // espacio en memoria
28    os_sprintf(buf, "%", "\mcast":"); // mensaje
29    os_sprintf(buf + os_strlen(buf), MACSTR, MAC2STR(src)); // mensaje + MAC
30    os_sprintf(buf + os_strlen(buf), "%", "\}\r\n"); // fin de linea
31
32    // bytes disponibles en paquete Mesh
33    max_count = (ESP_MESH_PKT_LEN_MAX - ESP_MESH_HLEN - os_strlen(buf));
34    // maxima cantidad de opciones por paquete
```

```

35     max_count /= ESP_MESH_OPTION_MAX_LEN;
36     // cantidad de direcciones MAC que pueden agregarse
37     max_count *= ESP_MESH_DEV_MAX_PER_OP;
38     if (dev_count > max_count) // cantidad de nodos en red mesh es mayor que
39         las que se pueden colocar
40         dev_count = max_count;
41
42     op_count = dev_count / ESP_MESH_DEV_MAX_PER_OP; // cantidad de "opciones
43     mesh" necesarias
44     ot_len = ESP_MESH_OT_LEN_LEN // longitud del campo de "opcion" en paquete
45     mesh
46     + op_count * ( sizeof(*option) + sizeof(*root) *
47         ESP_MESH_DEV_MAX_PER_OP)
48     + sizeof(*option) + (dev_count - op_count *
49         ESP_MESH_DEV_MAX_PER_OP) * sizeof(*root);
50
51     header = (struct mesh_header_format *)espconn_mesh_create_packet(
52         dst,          // destino: 01:00:5E:00:00:00
53         src,          // origen: STATION_IF
54         false,        // no es p2p
55         true,         // solicitud de piggyback
56         M_PROTO_JSON, // protocolo de aplicacion JSON
57         os_strlen(buf), // longitud de trama de datos
58         true,         // existe "opcion mesh"
59         ot_len,       // longitud total del campo de opcion
60         false,        // no hay fragmentacion
61         0,            // no hay fragmentacion
62         false,        // no hay fragmentacion
63         0,            // no hay fragmentacion
64         0);           // no hay fragmentacion
65     if (!header) { // comprobacion
66         MESH_PARSER_PRINT("mcast create packet have failed\n");
67         return;
68     }
69
70     while (i < op_count) {
71         // se agregan MAC_ADDRESS en MAC_ADDRESS_LIST al json mcast package
72         option = (struct mesh_header_option_format *)
73         espconn_mesh_create_option(
74             M_O_MCAST_GRP, (uint8_t * )(list + i * ESP_MESH_DEV_MAX_PER_OP)
75             ,
76             (uint8_t)( sizeof(*root) * ESP_MESH_DEV_MAX_PER_OP));
77         if (!option) { // comprobacion
78             MESH_PARSER_PRINT("mcast create option %d have failed\n", i);
79             goto MCAST_FAIL;
80         } else {
81             MESH_PARSER_PRINT("mcast create option %d have been succesfull\n", i
82         );

```

```

74     }
75
76     if (!espconn_mesh_add_option(header, option)) { // agrega opcion al
77         paquete Mesh
78         MESH_PARSER_PRINT("mcast %d set option fail\n", i);
79         goto MCAST_FAIL;
80
81         i++; // proxima opcion
82         MESH_DEMO_FREE(option); // libera espacio en memoria
83         option = NULL;
84
85     { // si queda alguna otra direccion MAC por agregar
86         char *rest_dev = NULL;
87         uint8_t rest = dev_count - op_count * ESP_MESH_DEV_MAX_PER_OP;
88         if (rest > 0) {
89             rest_dev = (char *)os_zalloc(sizeof(*root) * rest);
90             if (!rest_dev) {
91                 MESH_PARSER_PRINT("mcast alloc the last option buf have failed
92 \n");
93                 goto MCAST_FAIL;
94             }
95             os_memcpy(rest_dev, root, sizeof(*root));
96             if (list && rest > 1)
97                 os_memcpy(rest_dev + sizeof(*root),
98                           list + i * ESP_MESH_DEV_MAX_PER_OP,
99                           (rest - 1) * sizeof(*root));
100            option = (struct mesh_header_option_format *)
101 espconn_mesh_create_option(
102             M_O_MCAST_GRP, rest_dev, sizeof(*root) * rest);
103            MESH_DEMO_FREE(rest_dev); // libera espacio en memoria
104            if (!option) { // comprobacion
105                MESH_PARSER_PRINT("mcast create the last option have failed\n");
106            }
107            goto MCAST_FAIL;
108        }
109        // se agrega data de usuario al mensaje broadcast
110        if (!espconn_mesh_set_usr_data(header, buf, os_strlen(buf))) {
111            MESH_DEMO_PRINT("mcast set_user data have failed\n");
112            goto MCAST_FAIL;
113        }
114        // enviar mensaje a traves de la red mesh
115        if (espconn_mesh_sent(&g_ser_conn, (uint8_t *)header, header->len)) {
116            MESH_DEMO_PRINT("mcast mesh is busy right now\n");
117        }
118    }
119 }
```

```
116     espconn_mesh_connect(&g_ser_conn);}
```

```
117
```

```
118 MCAST_FAIL: // en caso de error durante construccion del mensaje mcast
```

```
119     option ? MESH_DEMO_FREE(option) : 0;
```

```
120     header ? MESH_DEMO_FREE(header) : 0;}
```

## APÉNDICE B:

### Información adicional sobre el ESP8266

#### Diagrama de pines del ESP8266

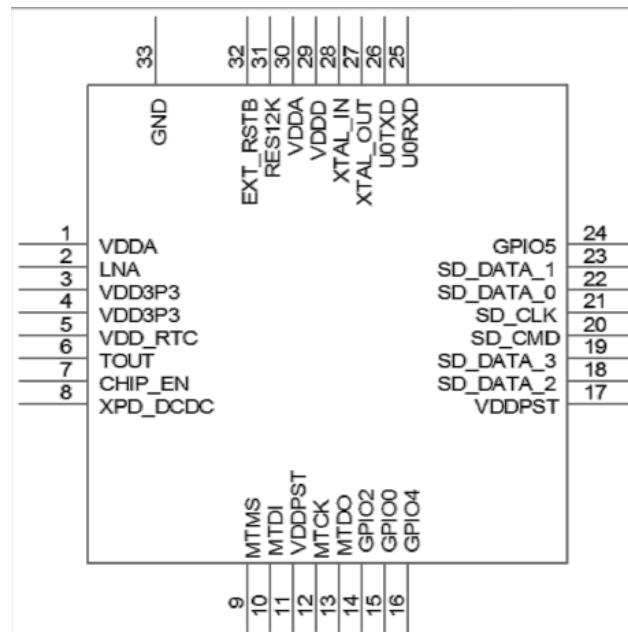


Figura A.1: Diagrama de pines del encapsulado ESP8266 [21]

#### Tamaño del encapsulado en el ESP8266

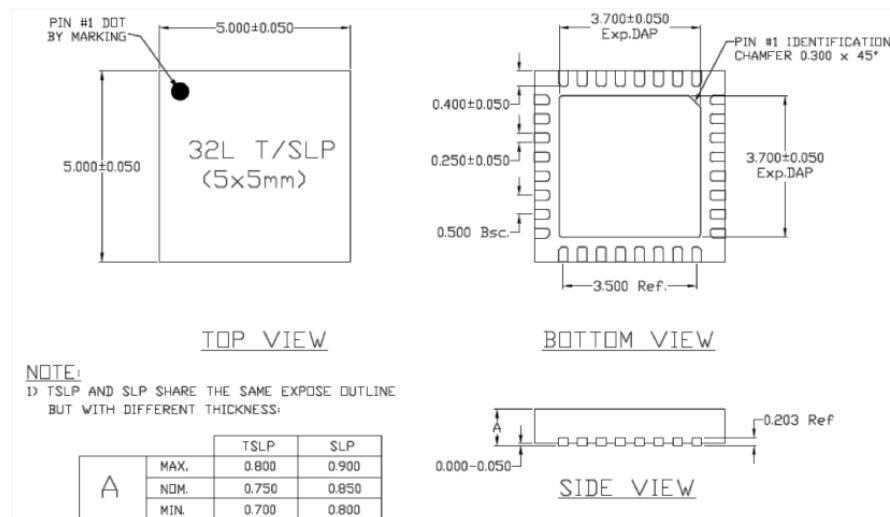


Figura A.2: Especificaciones de tamaño del encapsulado del ESP8266 [21]

## Módulo de desarrollo de NodeMCU con ESP-12

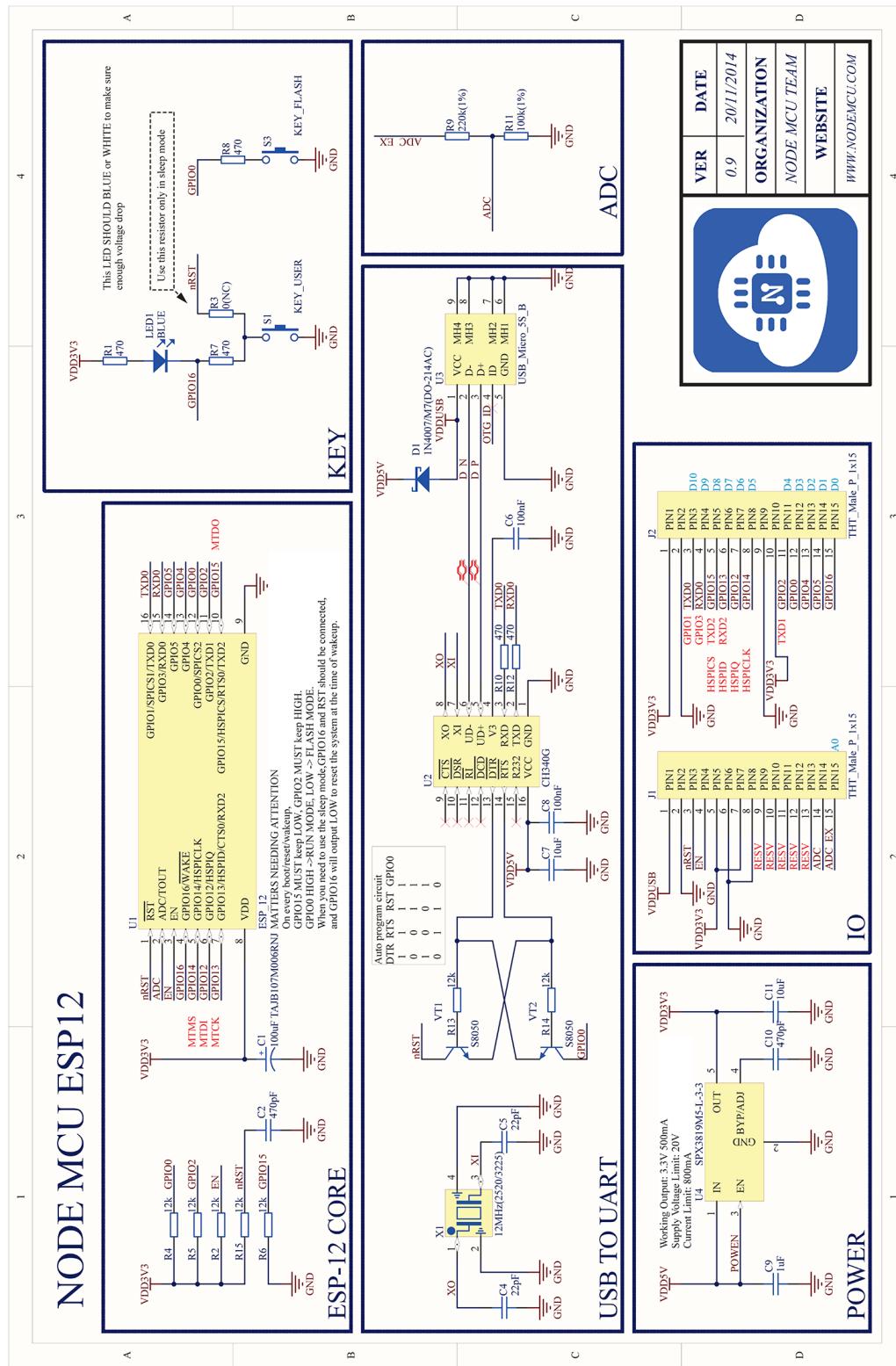


Figura A.3: Esquemático del módulo de desarrollo fabricado por NodeMCU [30]

## APÉNDICE C:

### Código de Programación del Servidor Local

```
1 from __future__ import print_function
2 import sys, datetime, threading
3 import struct, binascii
4 import paho.mqtt.publish as publish
5 if sys.version_info[0] < 3:
6     import SocketServer as socketserver
7 else:
8     import socketserver
9
10 ##### Declaraciones para configuracion de MQTT #####
11 field_list = [] # inicializacion de lista de campos
12 value_list = [] # inicializacion de lista de valores
13 channelID = "226554" # ThingSpeak "Channel ID"
14 apiKey = "8QOW4YE49TXLJBEB" # ThingSpeak "Write API Key"
15 # Coneccion insegura sobre MQTT
16 # utiliza menos cantidad de recursos (puerto 1883)
17 useUnsecuredTCP = False
18 # useUnsecuredWebSockets en "True" configura MQTT
19 # sobre un WebSocket sin seguridad (puerto 80).
20 useUnsecuredWebsockets = False
21 # Este tipo de conexion utiliza m s recursos
22 # pero establece una conexion segura con SSL (puerto 443)
23 useSSLWebsockets = True
24 # Hostname del servicio MQTT de ThingSpeak
25 mqttHost = "mqtt.thingspeak.com"
26 if useUnsecuredTCP:
27     tTransport = "tcp" # transporte utilizando TCP
28     tPort = 1883 # puerto 1883
29     tTLS = None # enlace no encriptado
30 elif useUnsecuredWebsockets:
31     tTransport = "websockets" # transporte utilizando WebSockets
32     tPort = 80 # puerto 80
33     tTLS = None # enlace no encriptado
34 elif useSSLWebsockets:
35     import ssl
36     tTransport = "websockets" # transporte utilizando TCP
37     tTLS = { 'ca_certs':"/etc/ssl/certs/ca-certificates.crt", 'tls_version':ssl.PROTOCOL_TLSv1}
38     tPort = 443 # puerto 443
39 ##### Cadena de caracteres para publicar en ThingSpeak #####
40 topic = "channels/" + channelID + "/publish/" + apiKey
41 ##### Final de declaraciones #####
```

```

42
43 class MeshHandler(socketserver.BaseRequestHandler): # Clase del manejador
44
45     def handle(self):
46         self.buf = bytearray() # Convierte handle.buf en arreglo de bytes
47         print('New node in mesh\n') # un nuevo nodo se ha agregado a la red
48         try:
49             while True:
50                 header = self.read_full(4) # ir a funcion read_full(self, 4)
51                 l, = struct.unpack_from('<H', header[2:4]) # longitud
52                 body = self.read_full(l-4) # ir a funcion read_full(self, l-4)
53                 req = bytearray()
54                 req.extend(header)
55                 req.extend(body)
56                 ##### construccion del mensaje de respuesta #####
57                 resp = bytearray()
58                 msg = "message received by server\r\n" # acuse de recibo
59                 req[2] = 16 + len(msg) # longitud total del paquete mesh
60                 resp.extend(req[0:4]) # parte del encabezado mesh
61                 resp.extend(req[10:16]) # direccion de destino
62                 resp.extend(req[4:10]) # direccion de origen mesh
63                 resp.extend(msg) # se agrega payload
64                 ##### fin del mensaje de respuesta #####
65                 src = binascii.b2a_hex(req[10:16]) # conversion hex --> ascii
66                 value_list.append(req[-7:-5]) # valor de temperatura
67                 ##### lista de nodos afiliados a ThingSpeak #####
68                 if src == "18fe34df0f4c":
69                     field_list.append("field1")
70                     to_thingspeak = True
71                 elif src == "5ccf7f86935f":
72                     field_list.append("field2")
73                     to_thingspeak = True
74                 elif src == "5ccf7fc11ade":
75                     field_list.append("field3")
76                     to_thingspeak = True
77                 elif src == "5ccf7f8691f1":
78                     field_list.append("field4")
79                     to_thingspeak = True
80                 ##### fin de la lista de nodos afiliados a ThingSpeak #####
81                 else:
82                     print("Mesh node doesn't have a channel-field in
83                         ThingSpeak")
84                     to_thingspeak = False
85                     if to_thingspeak:
86                         print("\nsending to channel ... {} in ThingSpeak" .format(
channelID))
86                         print("Waiting for transmission time") # en lista FIFO

```

```

87         list_aux1 = list(src)
88         i = 0 # contador
89         j = 2 # pivot
90         while i<5: # se construye mac address xx:xx:xx:xx:xx:xx
91             list_aux2 = list_aux1[j:]
92             list_aux1[j] = ":"
93             list_aux1[j+1:] = list_aux2
94             j = j+3
95             i += 1
96         src = ''.join(list_aux1) # mac address en formato string
97         ##### Informacion del mensaje recibido #####
98         print("message received from {} at {}".format(src, datetime.
99               datetime.now()))
100        print(req[16:]) # "The temperature is ----"
101        ##### envia mensaje de respuesta #####
102        self.request.sendall(resp)
103    except Exception as e:
104        print(e)
105
106    def read_full(self, n): # lee bytes del paquete mesh
107        while len(self.buf) < n:
108            try:
109                req = self.request.recv(1024) # recibe datos
110                if not req:
111                    raise(Exception('recv error')) # error al recibir
112                    self.buf.extend(req) # | self.buf | <--- req
113                except Exception as e:
114                    raise(e)
115            read = self.buf[0:n] # n bytes leidos en variable "read"
116            self.buf = self.buf[n:] # 1-n bytes en self.buf
117            return bytes(read) # retorna los bytes que han sido leidos
118
119 ##### manejador de lista FIFO #####
120 # value 1(out) <-- [    | value 2 | value 3 | ... | value n ] <-- value n+1(in)
121 def publish_mqtt():
122     if field_list:
123         tPayload = field_list[0] + "=" + str(value_list[0])
124         try:
125             publish.single(topic, payload=tPayload, hostname=mqttHost, port=
126             tPort, tls=tTLS, transport=tTransport)
127             print("the data have been published in channel {} and {}".format(
128               channelID, field_list[0]))
129             field_list[0:] = field_list[1:] # se trasladan listas FIFO
130             value_list[0:] = value_list[1:]
131         except:
132             print ("There was an error while publishing the data\n")

```

```
131     else:
132         print("There's not a message to publish in ThingSpeak")
133     threading.Timer(15, publish_mqtt).start() # temporizador en 15s
134     return
135
136 class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
137     pass
138
139 if __name__ == "__main__":
140     HOST, PORT = "0.0.0.0", 7000 # (ip + puerto) en el servidor local
141     server = ThreadedTCPServer((HOST, PORT), MeshHandler)
142     server.allow_reuse_address = True
143     print('mesh server is working')
144     publish_mqtt() # iniciar temporizador
145     server.serve_forever() # run server.c
```