WIRELESS SENSOR NETWORKS PROJECT

Domotic Wi-Fi Mesh

Juan Carlos Rodríguez Tallón, Gabriel González Rial

ÍNDICE

1. INTRODUCCIÓN Y OBJETIVO

El presente proyecto tiene como objetivo la creación de una red Wi-Fi mesh utilizando como elemento principal el SOC ESP8266 para la asignatura de Wireless Sensor Networks, del Máster en Electrónica Industrial.

Se ha decidido diseñar un sistema para control de domótica. Se ha pensado en el diseño de cubo electrónico, que, en función de su posición relativa y movimientos, permitirá el control de diversos elementos domóticos de manera remota. Este proyecto ha sido influenciado por la presentación de la multinacional Xiaomi de su producto Xiaomi AQara.

Ilustración 1: Xiaomi AQara

Para conseguir este objetivo se plantean los siguientes requerimientos mínimos para el diseño:

- Detección de posición relativa del cubo y de la rotación sobre cada una de sus caras.
- Elementos controlando LEDs en función del estado del cubo.
- Funcionamiento inalámbrico vía mesh Wi-Fi que se genere de manera autónoma.
- Puente hacia punto de acceso Wi-Fi independiente que permita el control de manera remota a la red mesh.
- Bajo consumo energético y capacidad de funcionar con baterías.

Todo el código de este proyecto está incluido en el repositorio de GitHub: https://github.com/JCCopyrights/ecube_mesh .

2. HARDWARE.

El principal componente de los nodos de la red es:

- **ESP8266-12-E:** Este componente es un SOC integrado, diseñado para funcionar como un sistema integral para gestionar comunicaciones Wi-Fi. Compuesto por un procesador de 32 bits ESP8266X, que cuenta con un stack IP completo, una memoria flash y una antena Wi-Fi. Puede trabajar como host de aplicaciones o descargar el material de Wi-Fi networking de otro procesador. Trabaja a 3.3V y posee una interfaz de programación serie a través de la UART. Este dispositivo

esta soportado por una gran comunidad de hobbistas, y posee integración completa con la IDE de Arduino.

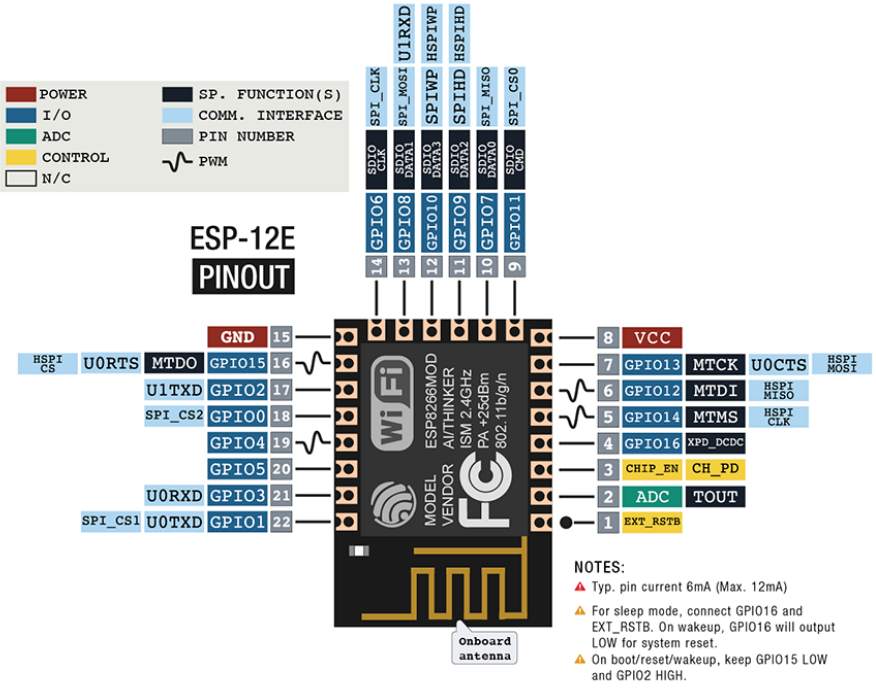

Ilustración 2: Pinout ESP8266-12E

Se utilizarán dos plataformas para generar la red y funcionar como redes de esta:

- ***NodeMCU:*** Plataformas open-source basadas en el SOC ESP8266-12E, interactivas, programables y de bajo coste. Se utilizarán estas tarjetas para controlar diversos LEDs para simular los elementos domóticos de la red.
- ***Ecube:*** Plataforma desarrollada en proyectos anteriores basada en ESP8266-12E y con un sensor acelerómetro y giroscopio de 6 ejes MPU-6050. Este dispositivo funcionará como el principal elemento de la red, utilizando la información de sus sensores mandará distintos comandos para controlar el resto de los nodos de la red. Para más información sobre el desarrollo de esta tarjeta puede encontrarse en ***https://github.com/JCCopyrights/ecube***

Adicionalmente se utilizarán LEDs RGB para ser controlados por la red, y una resistencia LDR para generar datos adicionales para retransmitir en la red desde una de las plataformas NodeMCU.

3. PAINLESSMESH. DISEÑO DE LA RED.

El diseño de la red está basado en la biblioteca *painlessMesh* que permite la programación y utilización de los módulos ESP8266-12E en la IDE de Arduino, así como proporcionar soporte para la creación y mantenimiento de la red Mesh. Adicionalmente la biblioteca proporciona utilidades adicionales como la sincronización de los nodos y automatiza el rutado de los mensajes.

- Mensajes basados en JSON (JavaScript Object Notation). Un formato de texto ligero para intercambio de datos.
- Implementado como un protocolo de capa 3. Sin conexión y sin confirmación.
- Totalmente asíncrono.
- Sincronización de tiempo entre todos los nodos con una máxima desviación de 10ms.
- Reconfiguración dinámica de los nodos.
- Bucles de red activamente evitados.
- La identificación de los nodos se realiza mediante los últimos 4 bytes de la dirección MAC de las radios Wi-Fi.

Todos los nodos pueden actuar como punto de acceso (AP) para que el resto de los nodos puedan conectarse, y a la vez se pueda conectar a otros nodos. Existe una limitación hardware debido al chipset del ESP8266-12E que restringe a un máximo de 4 estaciones por cada puto de acceso.

Una lista de todas las conexiones del nodo, así como una lista de todas las sub-conexiones para cada conexión del nodo están almacenadas en el mismo. De esta manera todos los nodos conocen la estructura completa de la red en cada momento. La lista de sub-conexiones se realiza mediante peticiones a los nodos circundantes, que devuelven sus conexiones directas y sus sub-conexiones. De manera periódica se escanea el estado de la red, si se detectan modificaciones en la misma:

1) En el caso de la pérdida de un punto de acceso la red se reestructura para completar las conexiones.
2) Se realizan peticiones para reconstruir las listas de sub-conexiones.
3) Se re-sincronizan los nodos.

Cada nodo que no esta en un momento dado conectado a un punto de acceso se conectará al AP con mayor señal que no esté presente en la lista de conexiones o sub-conexiones del nodo. Conectarse únicamente a punto de acceso desconocidos evita la creación de bucles de la red, de manera que se consigue una sola ruta entre cualquier par de nodos en la red.

Con una topología tipo estrella sin ningún tipo de bucles, y el hecho de que cada nodo es conocedor de todas las conexiones de la red, el rutado de los mensajes se convierte en un problema trivial.

- Para un único receptor (*unicast),* tanto el emisor como el receptor son conocidos, el nodo emisor manda el mensaje al nodo receptor si está directamente conectado a él, o a uno de sub-nodos que esté directa o indirectamente conectado al destinatario. Cuando este mensaje llega a un nodo que no es destinatario reenvía el mismo mensaje hacia la siguiente conexión hacia el destinatario, este proceso se repite hasta que el mensaje alcanza al nodo destinatario.

- Para múltiples receptores (*broadcast)* todas las conexiones son iteradas y el mensaje se manda a cada una de las conexiones. Cuando un nodo recibe un broadcast este mensaje se vuelve a mandar a todas las conexiones de este nodo exceptuando la dirección desde la que recibió el mensaje.

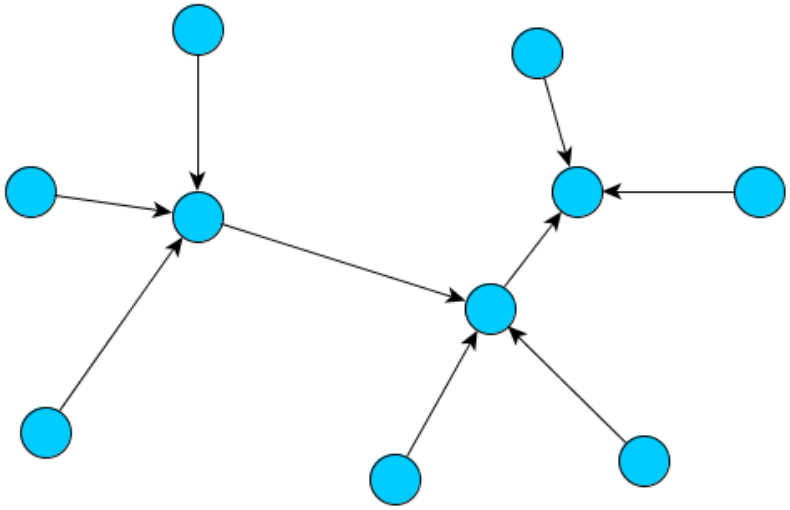

Ilustración 3: Topología de la red

3.1. PROTOCOLO DE RED.

El protocolo utilizado en esta biblioteca está basado en distintos esquemas JSON. Los mensajes entre nodos pueden dividirse entre mensajes de control y mensajes de usuario. Los mensajes de control entre nodos intercambian información sobre el rutado y la sincronización de los mimos, mientras que os mensajes de usuario transportan objetos JSON a un o varios nodos objetivos. El esquema básico de los mensajes es el siguiente:

```
{
  "dest":887034362,
  "from":37418,
  "type":9,
   "msg": "The message I want to
send",
    “subs”: [
       "nodeId": ...,
               "subs": [ {
                       "nodeId":
                       "subs": []}
             ]
}
```

- **“dest”:** ID de destino
- **“from”:** ID de origen
- **“type”:** Tipo de mensaje
  - 0,1,2: Sincronización de tiempos
  - 5,6: Petición/Respuesta de estructura de red para rutado
  - 8: Broadcast de usuario
  - 9: Unicast de usuario
- **“msg”:** Mensaje en paquetes de usuario
- “**subs”:** En paquetes de rutado contiene la información de las sub-conexiones de un nodo.

3.2. PUENTE ENTRE MESH Y RED WI-FI.

En este diseño de red se incorporará un nodo que funcionará como root y gestionara comunicaciones entre la red mesh y una red Wi-Fi externa a través de un servidor HTTP, sirviendo a peticiones.

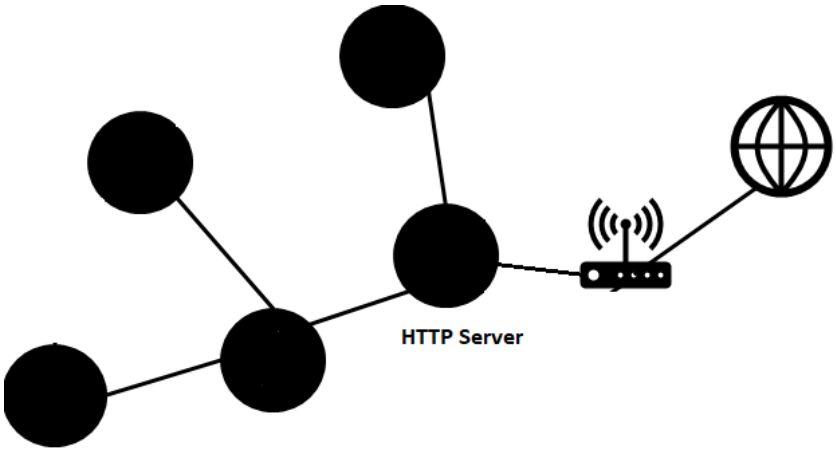

Ilustración 4: Puente Mesh-Wi-Fi

El servidor web será capaz de crear broadcast con información que se de desde la red exterior, y de mostrar la actual disposición de la red en formato JSON. El nodo puente alternará entre AP y modo estación para dar servicio tanto al mesh como a la red externa.

Ilustración 5: Servidor HTTP

4. SOFTWARE

4.1. BIBLIOTECAS Y DEPENDENCIAS

Para el software se ha utilizado la IDE de Arduino gracias a la compatibilidad de las bibliotecas painlessMesh con la iniciativa “arduino-esp8266” que permite incluir placas basadas en ESP8266 como tarjetas objetivo en la IDE de Arduino. Para desarrollar la red este proyecto se han utilizado las siguientes bibliotecas:

- **painlessMesh:** Se encarga de crear y gestionar la red https://github.com/gmag11/painlessMesh
- **ArduinoJson:** Biblioteca C++ para codificar y decodificar JSON para IOT https://github.com/bblanchon/ArduinoJson
- **TaskScheduler:** Gestor de tareas ligero para varios microcontroladores. https://github.com/arkhipenko/TaskScheduler

- **ESPAsyncTCP:** Biblioteca para comunicación TCP asíncrona para ESP8266 https://github.com/me-no-dev/ESPAsyncTCP
- **ESPAsyncWebServer:** Servidores HTTP asíncronos para ESP8266-Arduino. https://github.com/me-no-dev/ESPAsyncWebServer

Adicionalmente para el control de los sensores y dispositivos del nodo cubo controlador se requieren las siguientes bibliotecas:

- **I2Cdev:** Gestión de comunicaciones para interfaces I2C https://github.com/jrowberg/i2cdevlib
- **Adafruit_GFX /Adafruit_SSD1306:** Bibliotecas de gestión de pantallas OLED de Adafruit.

4.2. NODOS Y UTILIDADES

El proyecto consta de 4 tipos diferentes de nodo con distintos programas y funciones. Todas las comunicaciones en esta red se realizarán mediante broadcasts de manera que la red sea escalable, y se puedan incorporar más nodos a la red sin que esto afecte a la programación.

Mensaje	Interpretación
FAC: XXXX	XXXX numero entero de la cara apoyada
ANG: XXXX	XXXX numero entero del ángulo girado
LDR: XXXX	XXXX numero entero de iluminación

Tabla 1: Mensajes en la red.

Los nodos pueden clasificarse en:

- **RootNode:** Funciona como puente entre la mesh y otra red Wi-Fi. Tiene un servidor HTTP que permite mandar broadcast a la red mesh y mostrar la topología actual de la red. Imprime por el puerto serie todos los mensajes de la red para depurado de la misma.
- **CubeNode:** Este nodo calcula de manera periódica cada 20 ms la cara sobre la que se apoya el cubo y el ángulo girado sobre su eje y comunica esta información mediante broadcast al resto de la red.
- **RGBNode:** Este nodo interpreta la información que llega desde la red y actúa modificando el estado de su led RGB.

Cara Apoyada	Función	Angulo
1	Led Rojo	Intensidad
2	Led Verde	Intensidad
3	Led Azul	Intensidad
4	Selección de Color	Color
5	Parpadeo sincronizado	Frecuencia
6	Cede el control a LDR	Nada

Tabla 2: Funcionamiento de RGBNodes

- **LDRNode:** Este nodo actúa de manera similar al RGBNode, pero adicionalmente posee un sensor LDR con el calcula la iluminación cada segundo y lo comunica mediante broadcast al resto de la red. Si el cubo está apoyado sobre la cara 6 los RGBNodes brillaran proporcionalmente al valor de iluminación comunicado.

5. DESPLIEGUE Y CONCLUSIONES.

Se realiza una prueba con el siguiente despliegue de dos RGBNode, LDRNode, RootNode, CubeNode:

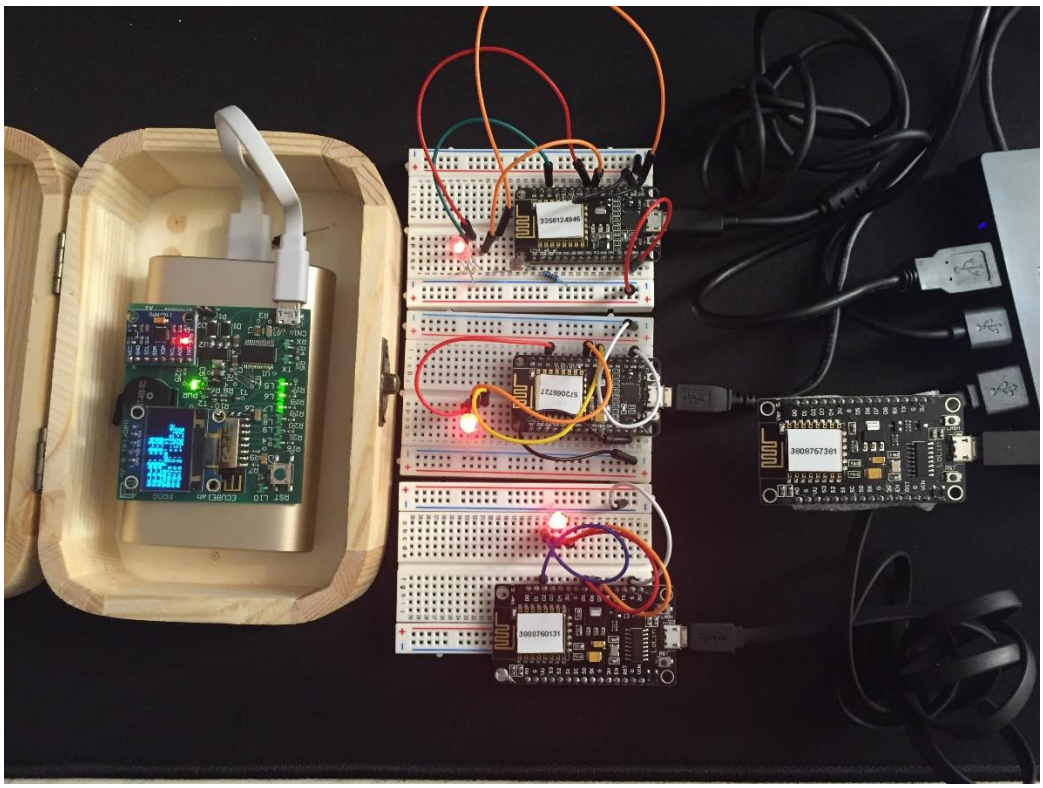

Ilustración 6: Despliegue real

Tras unos segundos la red se constituye con la siguiente estructura:

[{"nodeId":3808760131,"subs": [{"nodeId":3256124945,"subs": []},
{"nodeId":572065727,"subs": [{"nodeId":2139211003,"subs": []}]}]}]

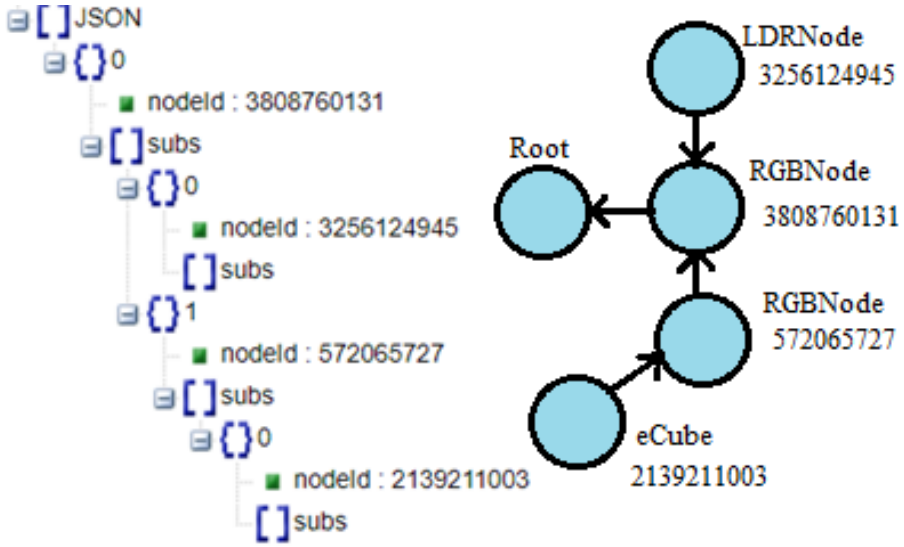

Ilustración 7: Topología de la red

Cabe detallar que la estructura de la red en este caso en el que todos los nodos están en el alcance de los demás, es totalmente aleatoria, y solo depende del momento en el que los nodos intentan conectarse a la red y la intensidad de las radios.

Se comprueba que el sistema cumple con la funcionalidad deseada. A pesar de que la red no tiene ningún mecanismo de control de recepción de mensajes no se detecta ningún a pérdida de mensajes, y todos los mensajes son recibidos por todos los nodos.

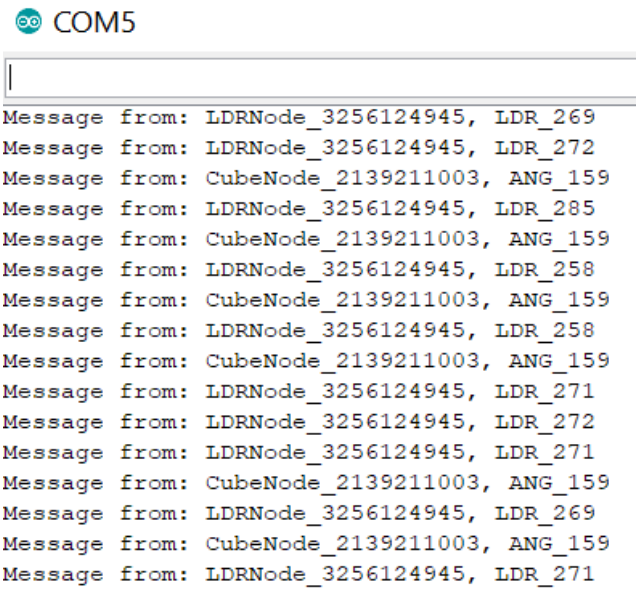

Ilustración 8: Lista de mensajes de la Red

Por otro lado, la frecuencia de comunicación sí que puede crear cierta latencia en la red, y para futuros despliegues se debería limitar el numero de comunicaciones por segundo, o generar mensajes únicamente sobre cambios de las variables con cierta histéresis.