**ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH**
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
**BỘ MÔN ĐIỆN TỬ**
----------o0o----------


**ĐỒ ÁN MÔN HỌC**

# Xây dựng và kiểm thử hệ thống thu thập dữ liệu cảm biến đa điểm bằng ESP-MESH và MQTT

**GVHD: Nguyễn Trọng Luật**
**SVTH: Nguyễn Trường Sơn**
**MSSV: 2212930**

**TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2025**

# *LỜI CẢM ƠN*

***Em xin trân trọng gửi lời cảm ơn đến thầy Nguyễn Trọng Luật, giảng viên hướng dẫn đồ án 2 của em, vì đã luôn tận tình chỉ bảo, định hướng và góp ý chi tiết trong suốt quá trình thực hiện. Những góp ý chuyên môn cùng sự động viên của thầy đã giúp em xác định rõ mục tiêu, lựa chọn phương pháp phù hợp và hoàn thiện sản phẩm theo đúng tiến độ.***

***Em cũng xin gửi lời cảm ơn đến quý thầy cô trong Khoa đã truyền đạt cho em nền tảng kiến thức vững chắc trong suốt thời gian học tập tại trường. Chính những kiến thức cơ sở và kỹ năng tư duy được tích lũy từ các học phần đã giúp em triển khai ý tưởng, xử lý các vấn đề kỹ thuật phát sinh và trình bày kết quả một cách mạch lạc, có định hướng.***

***Bên cạnh đó, em chân thành cảm ơn gia đình đã luôn ủng hộ, tạo điều kiện về thời gian và tinh thần; cảm ơn thầy cô, anh chị kỹ thuật và các bạn trong lớp/nhóm đã hỗ trợ, chia sẻ tài liệu, góp ý và cùng em thảo luận trong quá trình làm đồ án. Những hỗ trợ quý báu ấy là động lực để em nỗ lực hoàn thành đề tài.***

***Một lần nữa, em xin chân thành cảm ơn!***

*Tp. Hồ Chí Minh, ngày 18 tháng 12 năm 2025.*

**Sinh viên**

**Nguyễn Trường Sơn**

# TÓM TẮT ĐỒ ÁN

Đồ án này trình bày về xây dựng một hệ thống IoT dựa trên ESP-MESH gồm các Leaf node(thu thập cảm biến), hai Relay node(trung gian nhiều chặng) và một Root node làm gateway kết nối MQTT Broker trong LAN. Hệ thống được thiết kế để bảo đảm truyền dữ liệu cảm biến ổn định, mở rộng vùng phủ mà không tăng công suất phát. Phần mềm hiện thực bằng ESP-IDF 5.4: Leaf node đóng gói JSON và gửi qua mesh; Relay node hoạt động MESH-NODE để tự động chuyển tiếp; Root node(cố định vai trò MESH-ROOT) nhận dữ liệu và publish lên MQTT. Cùng với đó là sự kiểm chứng khả năng Child node chọn Parent node và cơ chế tự phục hồi của ESP-MESH.

# MỤC LỤC

# DANH SÁCH HÌNH MINH HỌA

# DANH SÁCH BẢNG SỐ LIỆU

# 1. GIỚI THIỆU

## 1.1 Tổng quan

Internet of Things (IoT) đang tạo ra nhu cầu thu thập và truyền dữ liệu cảm biến ở quy mô rộng, trong khi các mô hình mạng kiểu sao (star) thường bị giới hạn về vùng phủ, độ tin cậy khi vật cản và khả năng mở rộng. ESP-MESH – kiến trúc mạng lưới dựa trên ESP32 – cho phép các nút trung gian (relay) tự động chuyển tiếp gói tin, mở rộng phạm vi phủ sóng mà không cần tăng công suất phát, đồng thời duy trì chi phí thấp. Ở lớp ứng dụng, giao thức MQTT cung cấp cơ chế publish/subscribe nhẹ, phù hợp thiết bị tài nguyên hạn chế. Đề tài này triển khai một kiến trúc ESP-MESH trong mạng nội bộ (LAN) gồm: các nút Leaf (cảm biến), hai nút Relay (trung gian) và một nút Root đóng vai trò cổng kết nối (gateway) tới MQTT Broker. Kiến trúc cho phép dữ liệu cảm biến được chuyển tiếp nhiều chặng (multi-hop) tới Root và được công bố trên các topic MQTT để hệ thống giám sát/ứng dụng phía trên tiêu thụ. Trọng tâm là thiết kế, hiện thực và đánh giá khả năng vận hành ổn định của mạng đa trung gian.

## 1.2 Nhiệm vụ đề tài

Các nhiệm vụ mà đề tài cần phải thực hiện được:

**Nhiệm vụ 1**: Thiết kế kiến trúc hệ thống ESP-MESH đa trung gian (Leaf - Relay A/B - Root) tích hợp MQTT trong LAN.

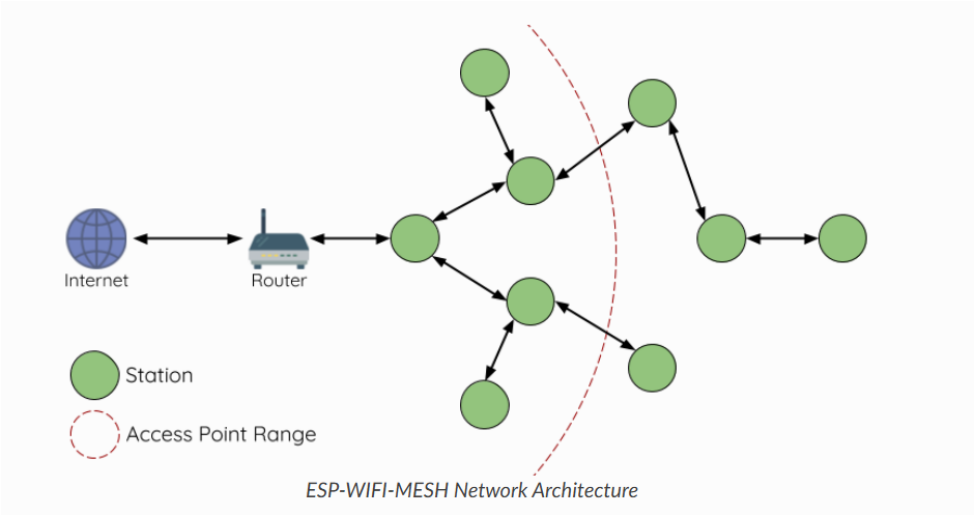
**Nhiệm vụ 2**: Xây dựng phần mềm cho ba vai trò: Leaf node (đọc cảm biến, đóng gói JSON), Relay node(MESH-NODE chuyển tiếp), Root node(MESH-ROOT nhận dữ liệu và publish lên MQTT) trên ESP-IDF.

**Nhiệm vụ 3**: Thử nghiệm cách mà Child node lựa chọn kết nối với Parent node và cơ chế tự phục hồi khi mất 1 node trong mạng.

**Nhiệm vụ 4**: Thu các dữ liệu từ chính xác theo thời gian thực bằng các cảm biến có sẵn: cam biến PIR, cảm biến LDR và cảm biến nhiệt độ và độ ẩm.

## 2. LÝ THUYẾT

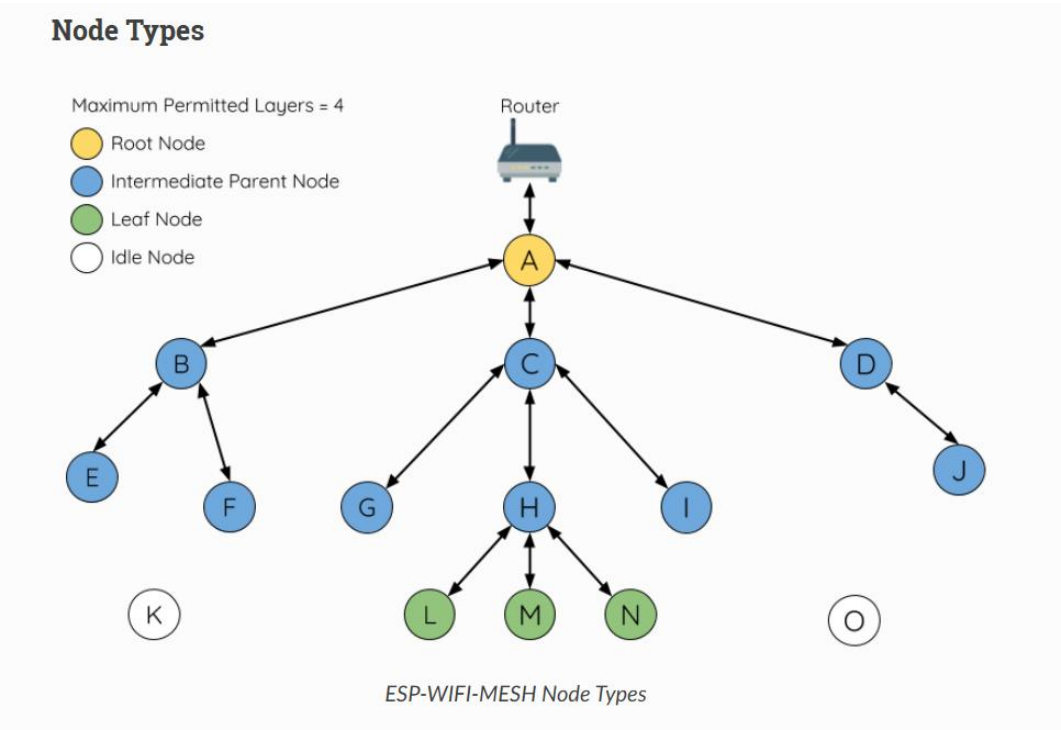
ESP-MESH là một giao thức mạng của Espressif dựa trên Wi-fi. Giao thức này cho phép nhiều thiết bị ESP32 được kết nối với nhau trong một không gian vật lý rộng lớn [1] "được kết nối với nhau thông qua một WLAN (mạng cục bộ không dây) duy nhất". Chúng làm tăng tầm kết nối của một Router.




Hình 2.1: Mô hình ESP-MESH

ESP-MESH được tổ chức theo mô hình cấu trúc cây (tree topology). Trong cấu trúc này, toàn bộ các thiết bị tham gia mạng mesh được xem như những node (nút mạng) và được phân bổ thành các tầng (layer) khác nhau dựa trên vị trí của chúng trong cây.

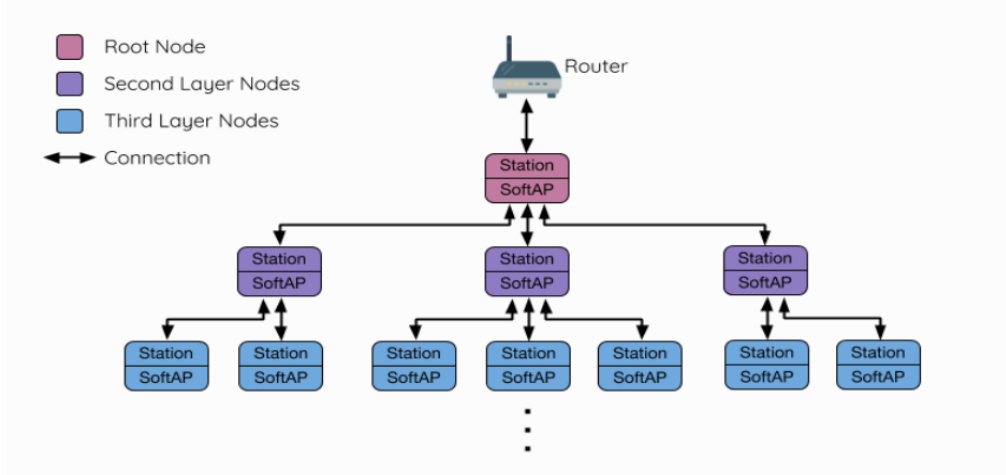
- Root node: Nút duy nhất trong toàn mạng có khả năng kết nối trực tiếp với bộ định tuyến Wi-fi. [2] "chỉ có thể có một nút gốc trong mạng ESP-WIFI_MESH và kết nối ngược dòng của nút gốc chỉ có thể là với bộ định tuyến".
- Child node: [3] "Bất kỳ nút nào được kết nối với Parent node. Dựa vào Parent node của nó để giao tiếp với phần còn lại của lưới và Root node"
- Parent node: Nút mà cho phép các Child node kết nối vào.
- Leaf node( End node): Nút biên nằm ở tầng dưới cùng của cây. Nút này không nhận thêm Child node và nhiệm vụ chính của nút này là thu thập dữ liệu cảm biến hoặc điều khiển thiết bị sau đó sẽ gửi dữ liệu ngược lên cây thông qua các Parent node.




Hình 2.2: Cấu trúc cây trong ESP-MESH

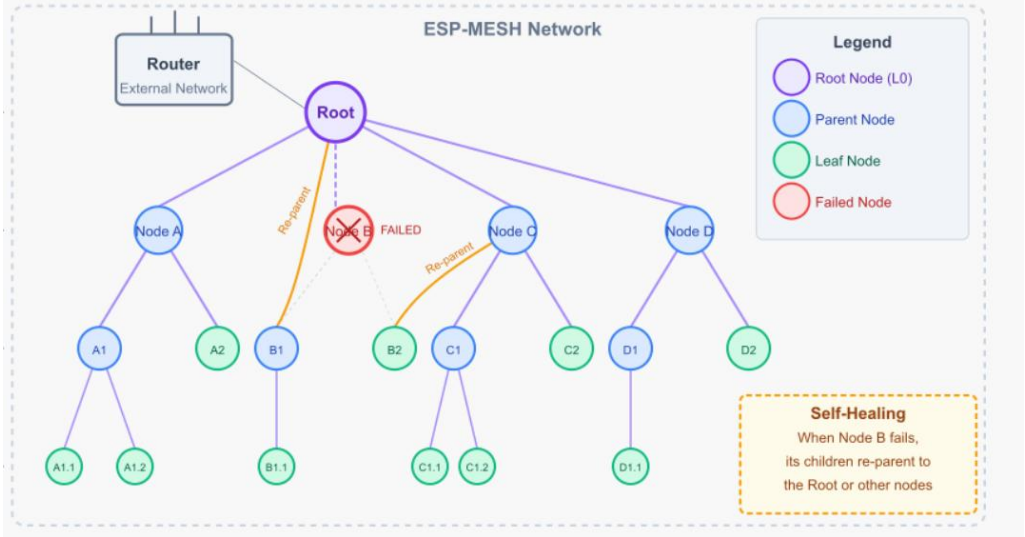
ESP-WIFI dựa trên cơ chế đặc biệt của ESP32 là có thể hoạt động đồng thời ở chế độ Station(STA) và một AP( Acess point). Nhờ vậy, mỗi nút trong mạng MESH đều có hai vai trò cùng lúc:

- Kết nối ngược dòng (upstream): Mỗi nút có duy nhất một liên kết STA để gắn vào Parent node. Đây là đường truyền ngược về phía gốc của mạng.
- Kết nối xuôi dòng (downstream): Mỗi nút mở giao diện softAP của riêng mình để cho phép các Child node kết nối vào. Đây là đường truyền cho các nút cấp dưới.




Hình 2.3: ESP-MESH với cơ chế Station–SoftAP đồng thời

ESP-MESH có cơ chế tự tổ chức và cơ chế tự phục hồi. Tự tổ chức là khả năng mạng MESH hình thành cấu trúc cây, gán vai trò cho các nút và duy trì kết nối mà không cần cấu hình thủ công ở tầng ứng dụng. [4] " ESP-WIFI-MESH là một mạng tự phục hồi, nghĩa là nó có thể phát hiện và sửa chữa lỗi định tuyến. Lỗi xảy ra khi một Parent node có một hoặc nhiều Child node bị hỏng, hoặc khi kết nối giữa Parent node và Child node trở nên không ổn định. Các Child node trong ESP-WIFI-MESH sẽ tự động chọn một Parent node mới và thiết lập kết nối ngược dòng với Parent node đó để duy trì kết nối mạng".




Hình 2.4: Cơ chế tự phục hồi của ESP-MESH

Cơ chế mà Child node chọn Parent node dựa vào các tiêu chí sau:

- RSSI: ưu tiên có cường độ mạnh, liên kết ổn định.

  [5] "RSSI (Rêcived Signal Strength Indicator) bản chất của nó là chỉ số cường độ tín hiệu thu hay là chỉ số để độ mạnh của tín hiệu thiết bị thu".

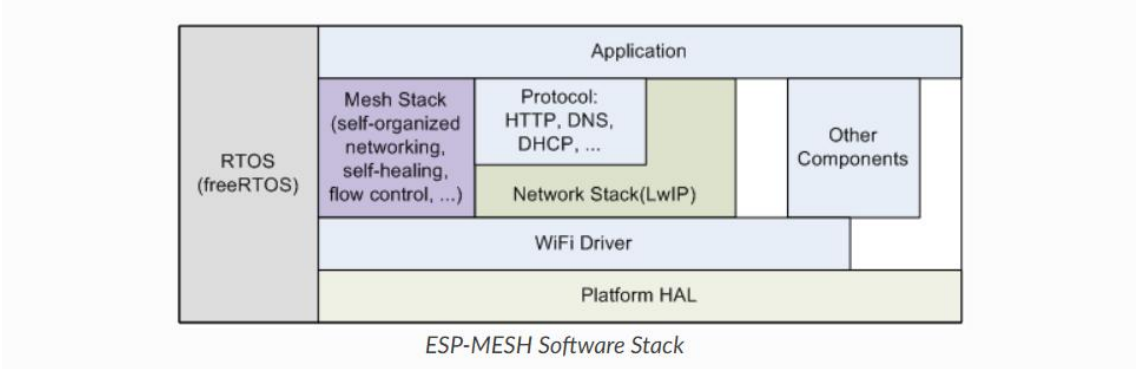
Bảng 2.1: Đánh giá chất lượng kết nối theo RSSI

| Cường độ tín hiệu (RSSI) | Chất lượng | Tốc độ DOWNLOAD | Dịch vụ có thể sử dụng |
|---|---|---|---|
| Lớn hơn -60 dBm | Tốt | 80% | Tất cả các dịch vụ: Voice, video, web, mail,.. |
| -61dBm -> -70dBm | Bình thường | 60% | Tất cả các dịch vụ: Voice, video, web, mail,... |
| -71dBm -> -80dBm | Yếu | 40% | Web, mail |

| | | | |
|---|---|---|---|
| Nhỏ hơn -80dBm | Rất yếu | N/A | Chạy không ổn định hoặc không hoạt động |

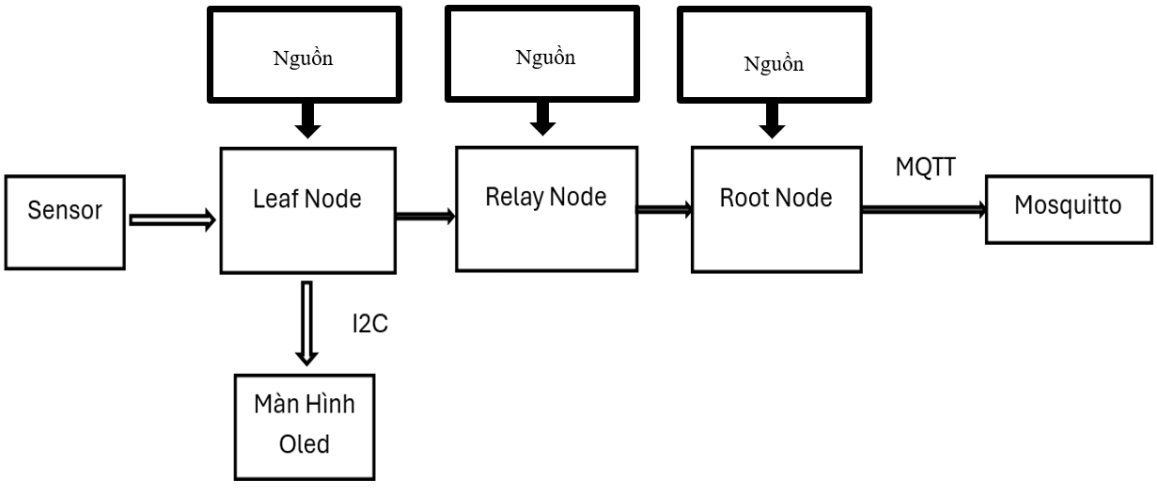
- Layer: ưu tiên Parent node có layer nhỏ tức đường truyền đến Root node ngắn hơn, độ trễ thấp hơn.
- Dung lượng: ưu tiên các Parent node có ít Child node đang kết nối sẵn.

[6]"ESP-MESH software stack được xây dựa trên trình điều khiển Wifi/FreeRTOS và có thể sử dụng LwIP stack trong một số trường hợp ( ví dụ: Root node)".



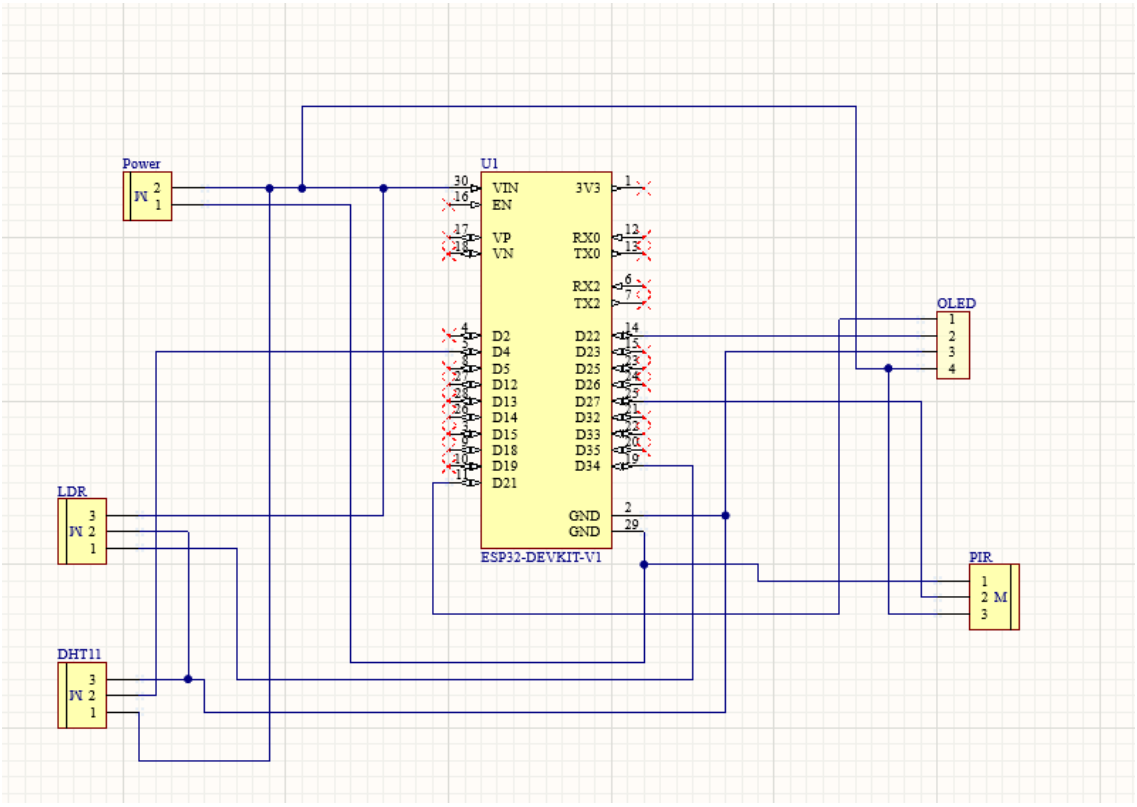

Hình 2.5: ESP-MESH software stack

## 3. THIẾT KẾ VÀ THỰC HIỆN PHẦN CỨNG

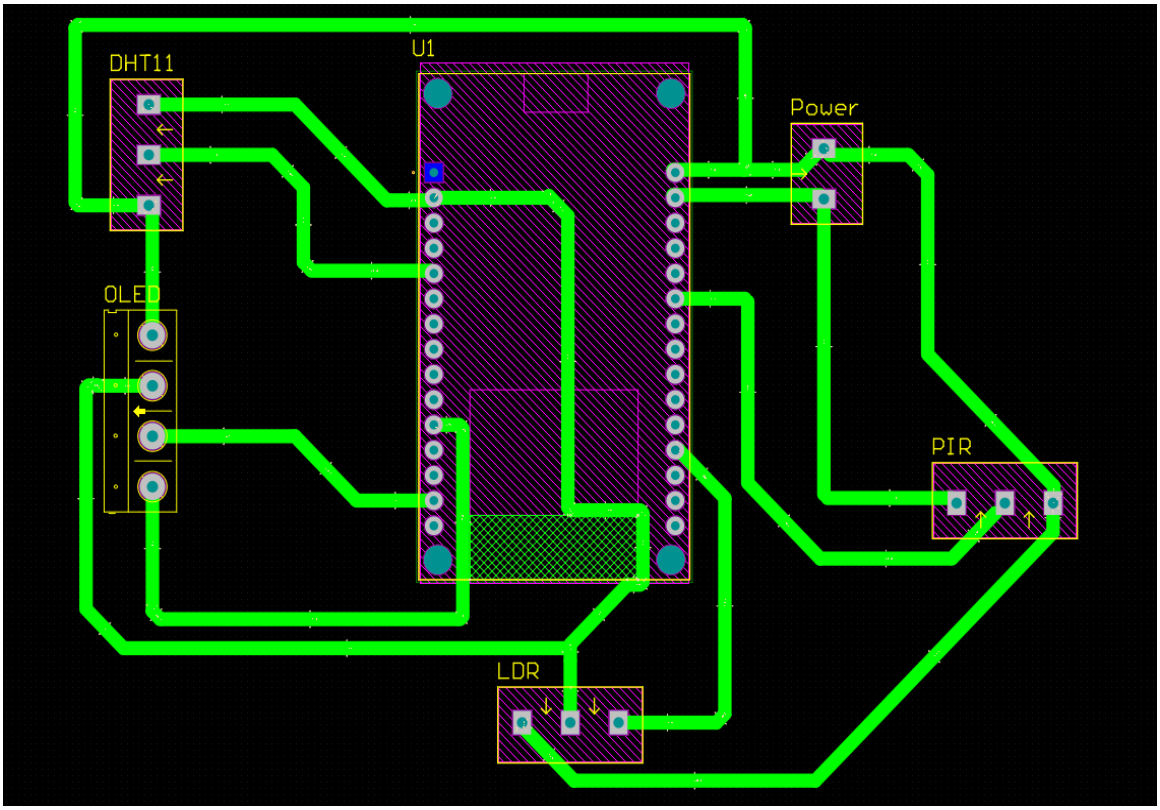



Hình 3.1: Block diagram ESP-MESH

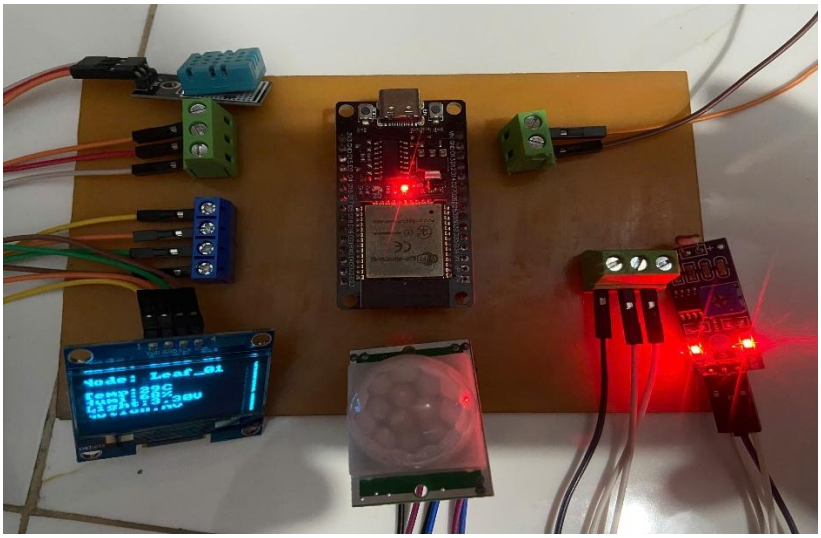
- Leaf Node




Hình 3.2: Schematic của Leaf Node

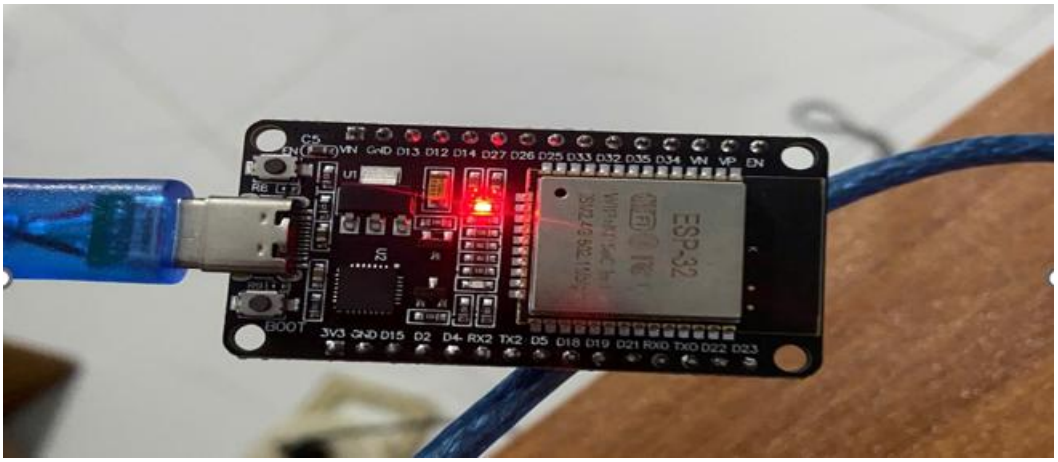



Hình 3.3: PCB của Leaf Node



Hình 3.4: Leaf Node thực tế

- Relay Node



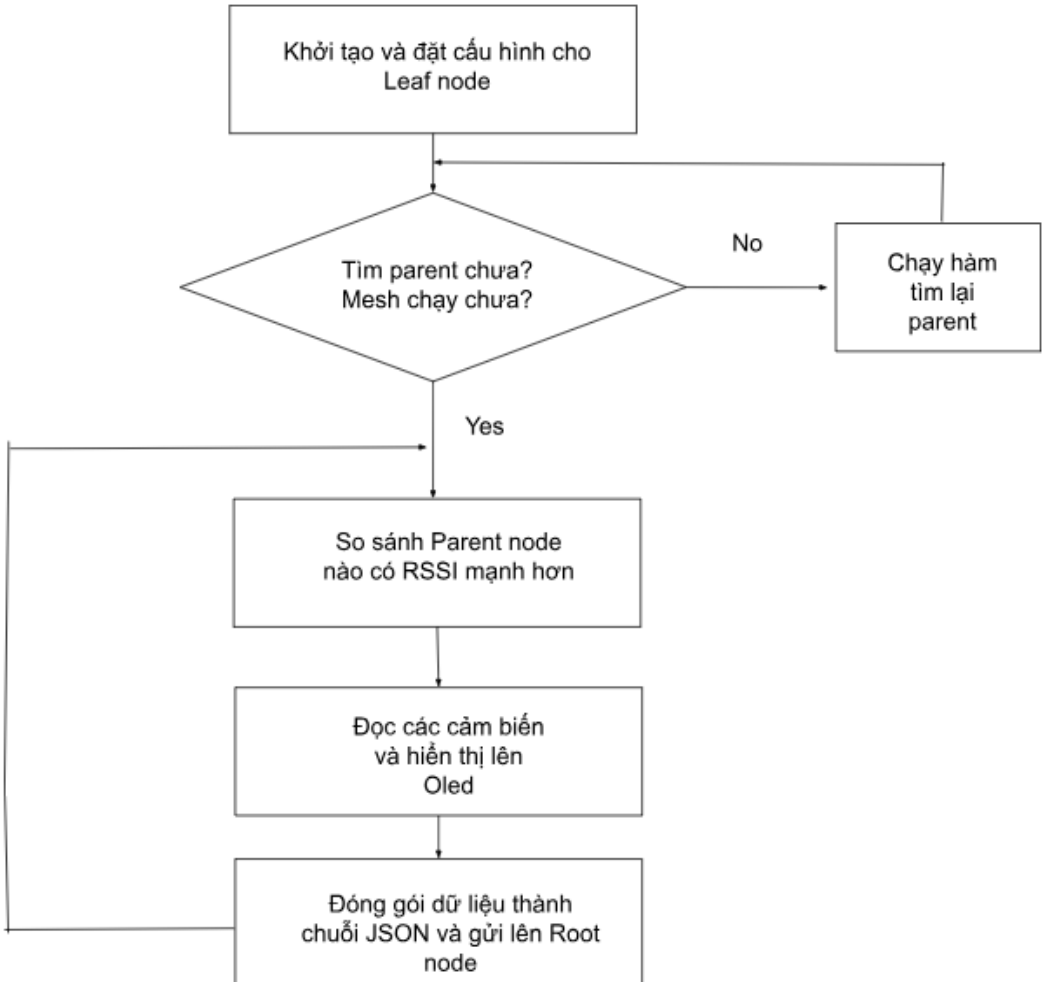

Hình 3.5: Relay Node thực tế

- Root Node

Hình 3.6: Root Node thực tế

# 4. THIẾT KẾ VÀ THỰC HIỆN PHẦN MỀM

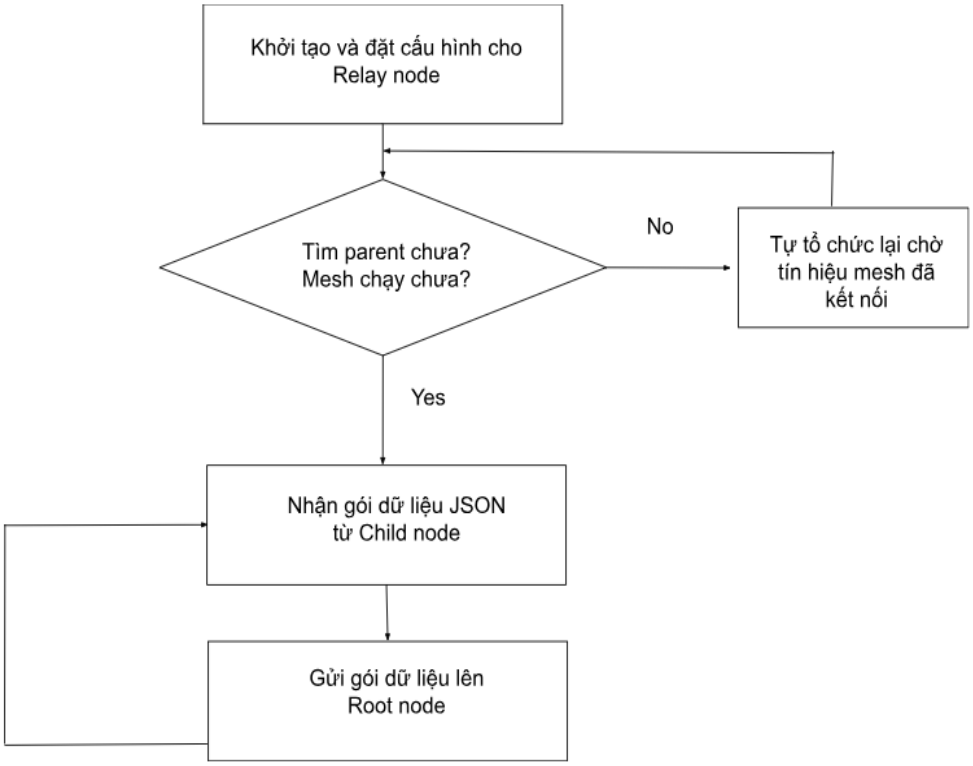
- Leaf Node




Hình 4.1: Lưu đồ giải thuật Leaf node

Leaf node bắt đầu bằng việc khởi tạo Wi-Fi/MESH, cấu hình MESH_ID/role và bật các cảm biến và OLED. Sau đó nó kiểm tra trạng thái: đã có parent và mesh đã chạy chưa. Nếu chưa, node gọi hàm tìm lại parent (quét và thử nối lại) rồi quay về bước kiểm tra. Nếu đã có, node so sánh các parent khả dụng theo RSSI và chọn đường lên tốt hơn để đảm bảo kết nối ổn định. Tiếp theo, node đọc dữ liệu cảm biến, hiển thị ngắn gọn lên OLED, đóng gói thành JSON và gửi P2P lên Root. Toàn bộ chu trình này lặp liên tục, nên khi mất kết nối node sẽ tự quay lại bước tìm parent và khôi phục truyền dữ liệu.

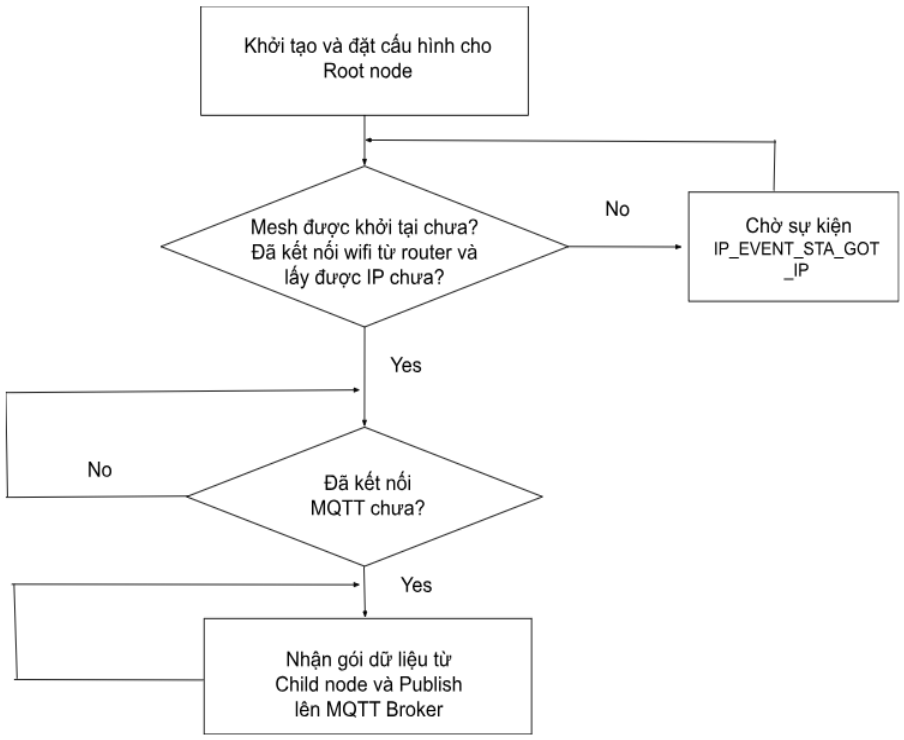
- Relay node




Hình 4.2: Lưu đồ giải thuật Relay node

Lưu đồ mô tả vòng đời của Relay node trong mạng ESP-Mesh: sau khi khởi tạo NVS–Wi-Fi–Mesh và phát SoftAP, nút kiểm tra đã tìm được parent và mesh đã chạy chưa; nếu chưa, nó để cơ chế tự tổ chức tiếp tục tìm/kết nối lại rồi quay về kiểm tra. Khi đã kết nối, relay lắng nghe và nhận gói JSON từ các Child node, sau đó gửi upstream: nếu đã biết địa chỉ root thì đẩy thẳng lên root, còn chưa thì gửi cho parent để mesh tự định tuyến. Quá trình này lặp liên tục; nếu rớt parent, relay quay lại nhánh tự tổ chức để khôi phục kết nối rồi tiếp tục nhận và gửi dữ liệu.

- Root node




Hình 4.3: Lưu đồ giải thuật Root node

Lưu đồ mô tả vòng đời Root node làm cổng ra mạng: sau khi khởi tạo Wi-Fi và Mesh, root kiểm tra đã khởi tạo mesh và STA đã kết nối router để lấy IP hay chưa; nếu chưa có IP, nó chờ sự kiện IP_EVENT_STA_GOT_IP rồi quay lại kiểm tra. Khi có IP, root tiến hành kết nối MQTT tới broker; nếu chưa kết nối được, tiếp tục lặp thử lại cho đến khi thành công. Khi MQTT đã kết nối, root nhận gói dữ liệu từ các child/relay qua mesh và publish lên broker theo topic quy ước. Chu trình này lặp liên tục; nếu mất IP hoặc rớt MQTT, nút quay lại các bước kiểm tra/kết nối tương ứng để khôi phục.

## 5. KẾT QUẢ THỰC HIỆN

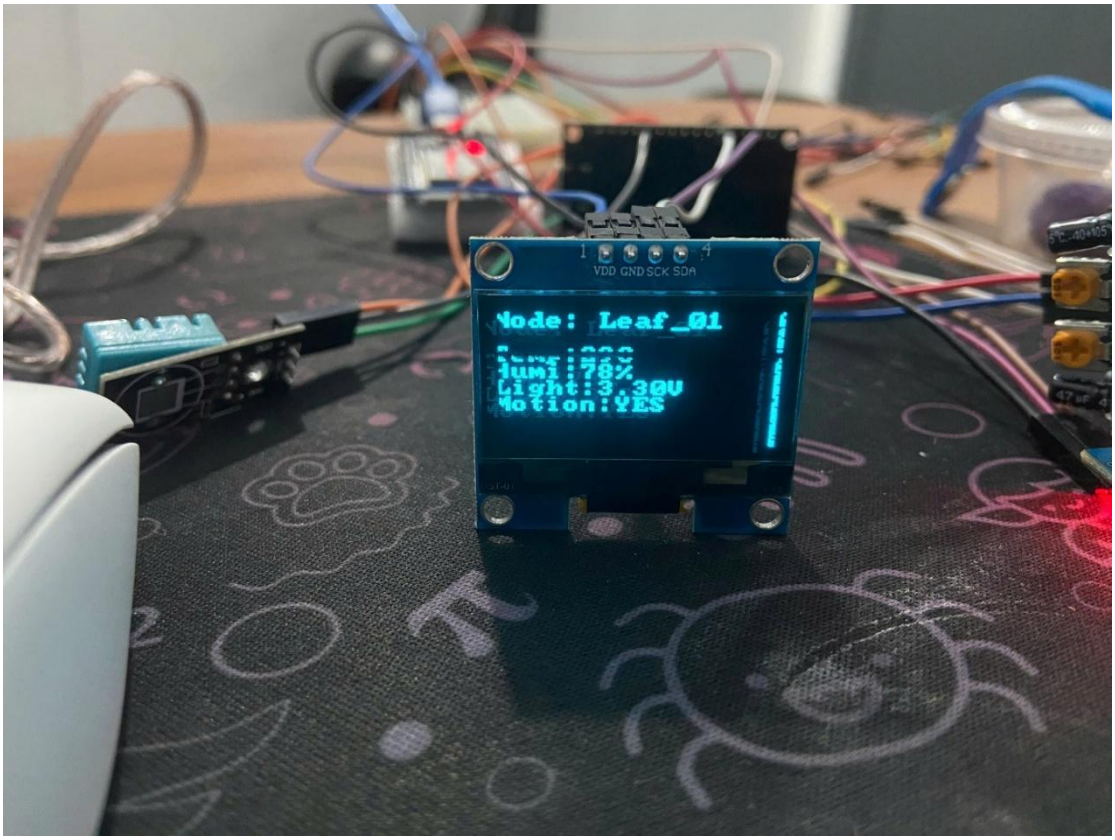
- Leaf node

```
I (26162) LEAF_NODE: Scan xong: KHONG thấy Relay A/B
I (28762) LEAF_NODE: Chọn parent: SSID="", BSSID=88:57:21:b3:56:f5, ch=8, RSSI=-49
```

Hình 5.1: Leaf Node chọn Parent Node

```
I (43892) LEAF_NODE: Light raw=4095, Vout=3.30 V
I (44052) LEAF_NODE: Sent to ROOT 30:ae:a4:77:10:2d: {"node_id":"Leaf_01","role":"leaf","temp":29,"humi":79,"light_v":3.
2999999523162842,"light_raw":4095,"motion":1}
```

Hình 5.2: Leaf Node gửi dữ liệu




Hình 5.3: Dữ liệu hiển thị trên OLED

- Relay node

```
I (1114) RELAY_NODE_A: Mesh started
I (1114) RELAY_NODE_A: RELAY STA MAC : 00:70:07:7e:6f:bc
I (1114) RELAY_NODE_A: RELAY BSSID   : 00:70:07:7e:6f:bd (Mesh SoftAP)
```

Hình 5.4: Địa chỉ STA và SoftAP của Relay Node A

```
I (5694) RELAY_NODE_A: Parent 30:ae:a4:77:10:2d, layer=2, is_root=NO
I (5694) mesh: <MESH_NWK_ROOT_ADDR>node, layer:2, root_addr:30:ae:a4:77:10:2d, conflict_roots.num:0<>
I (5704) RELAY_NODE_A: Parent RSSI: -39 dBm
I (5714) RELAY_NODE_A: Root MAC: 30:ae:a4:77:10:2d
```

Hình 5.5: Relay Node A đã kết nối được với Root Node

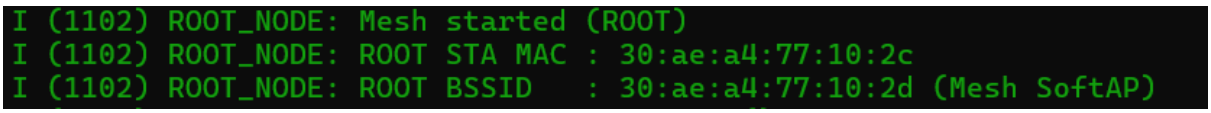
```
I (1104) RELAY_NODE_B: Mesh started
I (1104) RELAY_NODE_B: RELAY STA MAC : 88:57:21:b3:56:f4
I (1114) RELAY_NODE_B: RELAY BSSID   : 88:57:21:b3:56:f5 (Mesh SoftAP)
```

Hình 5.6: Địa chỉ STA và SoftAP của Relay Node B

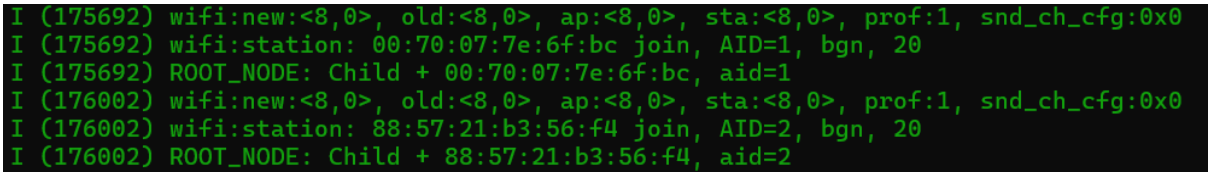
```
I (5704) RELAY_NODE_B: Parent 30:ae:a4:77:10:2d, layer=2, is_root=NO
I (5714) RELAY_NODE_B: Parent RSSI: -47 dBm
I (5724) RELAY_NODE_B: Root MAC: 30:ae:a4:77:10:2d
```

Hình 5.7: Relay Node B đã kết nối được với Root Node

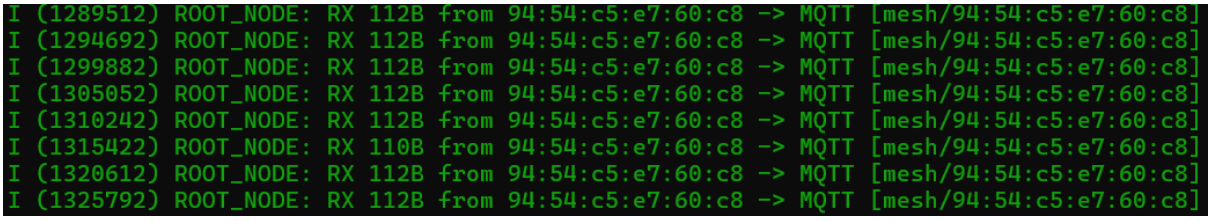
- Root node



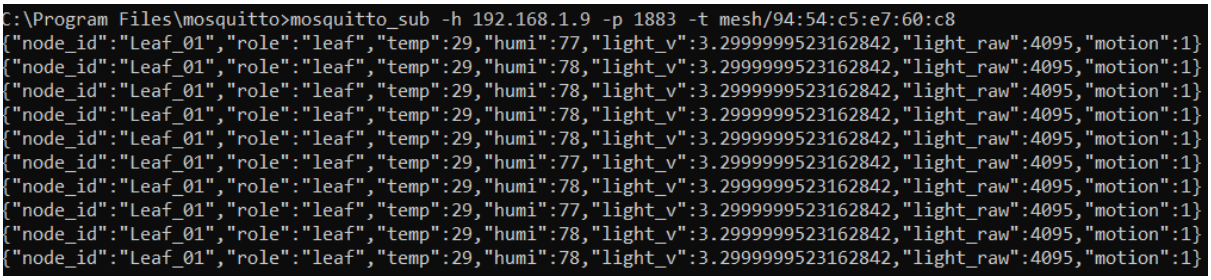

Hình 5.8: Địa chỉ STA và SoftAP của Root Node




Hình 5.9: Các Child Node đã kết nối được đến Root Node




Hình 5.10: Root Node đã nhận được dữ liệu từ Leaf Node




Hình 5.11: Root Node đã gửi được dữ liệu lên Server

# 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 6.1 Kết luận

Đề tài đã xây dựng thành công mô hình ESP-Mesh 3 tầng gồm Leaf - Relay - Root: leaf đọc cảm biến, relay nhận gói qua esp_mesh_recv() và chuyển upstream, root đóng vai gateway publish dữ liệu lên MQTT Broker. Hệ thống khởi tạo đúng quy trình (NVS, Wi-Fi, Mesh), tự tổ chức kết nối parent, ghi nhận RSSI/layer, và duy trì kênh HT20 để ổn định. Trên root, cơ chế sự kiện IP_EVENT_STA_GOT_IP – mqtt_start_if_needed() đảm bảo chỉ mở MQTT sau khi có IP, giúp pipeline Mesh -> MQTT hoạt động tin cậy.

Ưu điểm: kiến trúc gọn, dễ mở rộng node; phân tách rõ đường điều khiển (event handlers) và đường dữ liệu (tasks nhận/gửi); log minh bạch, dễ giám sát.

Nhược điểm: Mesh SoftAP vẫn ở chế độ mở nên chưa an toàn; chưa có hàng đợi và cơ chế gửi lại khi MQTT gián đoạn nên có nguy cơ mất dữ liệu; chưa áp dụng QoS và xác nhận ở tầng ứng dụng; chưa mã hóa đầu cuối và quản lý khóa; chưa tích hợp cập nhật firmware qua mạng; chưa có dự phòng cho nút gốc; và chưa đo lường bài bản các chỉ số hiệu năng như độ trễ, tỉ lệ mất gói, thông lượng theo số tầng và số lượng nút.

### 6.2 Hướng phát triển

- Bổ sung hàng đợi đệm và cơ chế gửi lại có kiểm soát thời gian để bảo đảm dữ liệu không thất lạc khi mạng gián đoạn.
- Áp dụng QoS mức một cho MQTT kèm xác nhận ở tầng ứng dụng nhằm nâng độ tin cậy truyền tải.
- Tăng cường bảo mật bằng WPA2 cho Mesh SoftAP, mã hóa đầu cuối và quy trình cấp phát khóa an toàn.
- Tích hợp cập nhật firmware qua mạng, kèm theo cơ chế kiểm tra tính toàn vẹn và khôi phục an toàn.
- Xây dựng bộ đo hiệu năng chuẩn hóa gồm độ trễ, tỉ lệ mất gói và thông lượng theo số tầng và số lượng nút, kèm dashboard giám sát.

## 7. TÀI LIỆU THAM KHẢO

[1] Espressif Systems (Shanghai) Co., Ltd, “ESP-WIFI-MESH”, docs.espressif.com

[2] Espressif Systems (Shanghai) Co., Ltd, “ESP-WIFI-MESH”, docs.espressif.com

[3] CircuitLabs, “Chapter 46: ESP32 WIFI Mesh Networking”, circuitlabs.net

[4] Espressif Systems (Shanghai) Co., Ltd, “ESP-WIFI-MESH”, docs.espressif.com

[5] CTS GROUP, “RSSI: Chỉ số cường độ tín hiệu thu – Cáchc tính suy hao và công suất thu Wifi”, ctsgroup.com.vn

[6] Espressif Systems (Shanghai) CO., LTD, “ESP-MESH Programming Guide”, docs.espressif.com

## 8. PHỤ LỤC

- Leaf node

```
static bool find_best_parent(mesh_parent_t *out)
{
    // Scan blocking. task đang gọi sẽ bị “chặn” (block) đến khi firmware quét xong tất cả kênh.
    wifi_scan_config_t sc = { .ssid=0, .bssid=0, .channel=0, .show_hidden=true };
    ESP_ERROR_CHECK(esp_wifi_scan_start(&sc, true));

    // Lấy số lượng AP rồi cấp phát đúng size
    uint16_t ap_num = 0;
    ESP_ERROR_CHECK(esp_wifi_scan_get_ap_num(&ap_num));
    if (ap_num == 0) {
        ESP_LOGI(TAG, "Scan xong: KHÔNG thấy AP nào");
        return false;
    }

    wifi_ap_record_t *recs = (wifi_ap_record_t*)calloc(ap_num, sizeof(wifi_ap_record_t));
    if (!recs) {
        ESP_LOGE(TAG, "calloc ap records fail");
        return false;
    }

    uint16_t n = ap_num;
    esp_err_t e = esp_wifi_scan_get_ap_records(&n, recs);
    if (e != ESP_OK) {
        ESP_LOGE(TAG, "esp_wifi_scan_get_ap_records err: %s", esp_err_to_name(e));
        free(recs);
        return false;
    }

    bool needA = bssid_is_nonzero(RELAY_A_BSSID);
    bool needB = (!ONLY_USE_RELAY_A) && bssid_is_nonzero(RELAY_B_BSSID);

    bool seenA = false, seenB = false;
    wifi_ap_record_t recA = {0}, recB = {0};

    for (int i = 0; i < n; ++i) {
        if (needA && !memcmp(recs[i].bssid, RELAY_A_BSSID, 6)) { recA = recs[i]; seenA = true; }
        if (needB && !memcmp(recs[i].bssid, RELAY_B_BSSID, 6)) { recB = recs[i]; seenB = true; }
    }

```

```
    if (!seenA && !seenB) {
        ESP_LOGI(TAG, "Scan xong: KHÔNG thấy Relay A/B");
        free(recs);
        return false;
    }

    const wifi_ap_record_t *best = NULL;
    if (seenA && seenB) best = (recA.rssi >= recB.rssi) ? &recA : &recB;
    else if (seenA)     best = &recA;
    else                best = &recB;

    memset(out, 0, sizeof(*out));
    memcpy(out->bssid, best->bssid, 6);
    strlcpy(out->ssid, (const char*)best->ssid, sizeof(out->ssid));
    out->channel = best->primary;
    out->rssi    = best->rssi;

    ESP_LOGI(TAG, "Chọn parent: SSID=\"%s\", BSSID=" MACSTR ", ch=%u,
RSSI=%d",
             out->ssid, MAC2STR(out->bssid), out->channel, out->rssi);

    free(recs);
    return true;
}
```

Hàm find_best_parent: Hàm quét Wi-Fi theo chế độ blocking để thu danh sách AP, sau đó chỉ lọc hai BSSID đã định danh trước (Relay A/B). Nếu tìm thấy, hàm so sánh cường độ tín hiệu (RSSI) và chọn parent có chất lượng tốt hơn, đồng thời điền thông tin ứng viên (BSSID, SSID, kênh, RSSI) vào cấu trúc đầu ra và trả về true; nếu không thấy A/B hoặc xảy ra lỗi khi quét/lấy bản ghi, hàm giải phóng bộ nhớ tạm và trả về false. Cách tiếp cận này bảo đảm leaf chỉ bám đúng các relay mong muốn, tránh join nhầm AP cùng SSID nhưng khác BSSID.

```
static void mesh_event_handler(void *arg, esp_event_base_t base, int32_t id,
void *event_data)
{
    switch (id) {
    case MESH_EVENT_PARENT_CONNECTED: {
        g_mesh_connected = true;
        mesh_event_connected_t *connected = (mesh_event_connected_t
*)event_data;
        esp_mesh_get_parent_bssid(&g_parent_bssid);

        ESP_LOGI(TAG, "PARENT_CONNECTED: " MACSTR ", layer:%d",
                 MAC2STR(g_parent_bssid.addr), connected->self_layer);
```

```
            wifi_ap_record_t ap_info;
            if (esp_wifi_sta_get_ap_info(&ap_info) == ESP_OK) {
                ESP_LOGI(TAG, "Parent RSSI: %d dBm", ap_info.rssi);
            }
            try_set_bandwidth();
            log_path();
            break;
        }
        case MESH_EVENT_PARENT_DISCONNECTED:
            g_mesh_connected = false;
            ESP_LOGW(TAG, "PARENT_DISCONNECTED -> reselect");
            schedule_reselect_parent();
            break;

        case MESH_EVENT_NO_PARENT_FOUND: {
            mesh_event_no_parent_found_t *e =
    (mesh_event_no_parent_found_t*)event_data;
            ESP_LOGW(TAG, "NO_PARENT_FOUND scan=%d -> reselect", e->scan_times);
            schedule_reselect_parent();
            break;
        }
        case MESH_EVENT_LAYER_CHANGE: {
            mesh_event_layer_change_t *e =
    (mesh_event_layer_change_t*)event_data;
            ESP_LOGI(TAG, "Layer -> %d", e->new_layer);
            break;
        }
        case MESH_EVENT_ROOT_ADDRESS: {
            const mesh_event_root_address_t *e = (const mesh_event_root_address_t
    *)event_data;
            memcpy(g_root_addr.addr, e->addr, 6);
            g_root_addr_ok = true;
            ESP_LOGI(TAG, "Root MAC: " MACSTR, MAC2STR(g_root_addr.addr));
            break;
        }
        default:
            break;
        }
    }
```

Hàm mesh_event_handler: Bộ xử lý sự kiện trung tâm của Mesh, cập nhật trạng thái kết nối và tuyến đường. Khi PARENT_CONNECTED, hàm đặt cờ g_mesh_connected, ghi nhận BSSID của parent, log RSSI và đường đi; khi PARENT_DISCONNECTED hoặc NO_PARENT_FOUND, hàm xóa cờ kết nối và kích hoạt quy trình chọn lại parent (schedule_reselect_parent()). Với MESH_EVENT_ROOT_ADDRESS, hàm lưu MAC của root vào g_root_addr và đặt

cờ g_root_addr_ok để các tác vụ gửi dữ liệu biết điểm đích. Ngoài ra, hàm log thay đổi layer để phục vụ giám sát.

```
static void send_sensor_task(void *arg)
{
    while (!g_mesh_connected || !g_root_addr_ok)
vTaskDelay(pdMS_TO_TICKS(300));
    ESP_LOGI(TAG, "TX ready: layer=%u, root=" MACSTR, esp_mesh_get_layer(),
MAC2STR(g_root_addr.addr));

    for (;;) {
        // DHT11
        struct dht11_reading dht = DHT11_read();
        int temp = 0, hum = 0;
        if (dht.status == DHT11_OK) { temp = dht.temperature; hum =
dht.humidity; }
        else                        { ESP_LOGW(TAG, "DHT11 read error"); }

        // PIR
        int motion = gpio_get_level(PIR_PIN);

        // LDR
        int   raw  = adc1_get_raw(LDR_ADC_CHANNEL);
        float vout = ((float)raw / 4095.0f) * Vref;
        ESP_LOGI(TAG, "Light raw=%d, Vout=%.2f V", raw, vout);

        // OLED
        char line[24];
        ssd1306_clear(&oled);
        ssd1306_display_text(&oled, 0, "Node: Leaf_01", false);
        snprintf(line, sizeof(line), "Temp:%dC",
temp);   ssd1306_display_text(&oled, 2, line, false);
        snprintf(line, sizeof(line), "Humi:%d%%",
hum);   ssd1306_display_text(&oled, 3, line, false);
        snprintf(line, sizeof(line), "Light:%.2fV",
vout);ssd1306_display_text(&oled, 4, line, false);
        snprintf(line, sizeof(line), "Motion:%s", motion ? "YES" : "NO");
        ssd1306_display_text(&oled, 5, line, false);

        // JSON
        cJSON *root = cJSON_CreateObject();
        cJSON_AddStringToObject(root, "node_id", "Leaf_01");
        cJSON_AddStringToObject(root, "role", "leaf");
        cJSON_AddNumberToObject(root, "temp", temp);
        cJSON_AddNumberToObject(root, "humi", hum);
        cJSON_AddNumberToObject(root, "light_v", vout);
        cJSON_AddNumberToObject(root, "light_raw", raw);
```

```
            cJSON_AddNumberToObject(root, "motion", motion);

            char *json_str = cJSON_PrintUnformatted(root);
            size_t len = json_str ? strnlen(json_str, sizeof(tx_buf) - 1) : 0;
            if (json_str && len > 0) {
                memcpy(tx_buf, json_str, len);
                tx_buf[len] = '\0';
            } else {
                const char *fallback = "{\"err\":\"json\"}";
                len = strlen(fallback);
                memcpy(tx_buf, fallback, len + 1);
            }

            data.data  = tx_buf;
            data.size  = len;
            data.proto = MESH_PROTO_BIN;
            data.tos   = MESH_TOS_P2P;


            mesh_addr_t dest = {0};
            memcpy(dest.addr, g_root_addr.addr, 6);
            esp_err_t err = esp_mesh_send(&dest, &data, MESH_DATA_P2P, NULL, 0);
            if (err == ESP_OK) ESP_LOGI(TAG, "Sent to ROOT " MACSTR ": %s",
MAC2STR(dest.addr), (char*)data.data);
            else               ESP_LOGE(TAG, "Mesh send failed: %s (0x%x)",
esp_err_to_name(err), err);

            if (json_str) free(json_str);
            cJSON_Delete(root);
            vTaskDelay(pdMS_TO_TICKS(5000));
        }
    }
```

Hàm send_sensor_task : chịu trách nhiệm thu thập và truyền dữ liệu. Nó chỉ bắt đầu sau khi đã có kết nối mesh và biết địa chỉ root (g_mesh_connected && g_root_addr_ok), sau đó lặp: đọc cảm biến DHT11/PIR/ADC LDR, hiển thị nhanh lên OLED, đóng gói payload JSON (node_id, role, temp, humi, light, motion), cấu hình mesh_data_t và gửi P2P tới root bằng esp_mesh_send(), ghi log thành công/thất bại rồi nghỉ 5 giây. Thiết kế này tách biệt rõ phần truyền thông (mesh) và phần cảm biến/hiển thị, giúp dễ bảo trì và theo dõi lỗi.

- Relay node

```
static void mesh_event_handler(void *arg, esp_event_base_t base, int32_t id,
void *event_data) {
    switch (id) {
        case MESH_EVENT_STARTED: {
```

```
            ESP_LOGI(TAG, "Mesh started");
            try_set_bw20();
            break;
        }
        case MESH_EVENT_PARENT_CONNECTED: {
            g_mesh_connected = true;
            mesh_event_connected_t *e = (mesh_event_connected_t*)event_data;
            esp_mesh_get_parent_bssid(&g_parent_bssid);
            ESP_LOGI(TAG, "Parent " MACSTR ", layer=%d, is_root=%s",
                     MAC2STR(g_parent_bssid.addr), e->self_layer,
                     esp_mesh_is_root() ? "YES" : "NO");
            wifi_ap_record_t ap_info;
            if (esp_wifi_sta_get_ap_info(&ap_info) == ESP_OK) {
                ESP_LOGI(TAG, "Parent RSSI: %d dBm", ap_info.rssi);
            }
            try_set_bw20();
            break;
        }
        case MESH_EVENT_PARENT_DISCONNECTED: {
            g_mesh_connected = false;
            ESP_LOGW(TAG, "Parent disconnected");
            break;
        }
        case MESH_EVENT_ROUTING_TABLE_ADD: {
            mesh_event_routing_table_change_t *e =
(mesh_event_routing_table_change_t*)event_data;
            ESP_LOGI(TAG, "+RT: %d, total=%d", e->rt_size_change, e-
>rt_size_new);
            break;
        }
        case MESH_EVENT_ROUTING_TABLE_REMOVE: {
            mesh_event_routing_table_change_t *e =
(mesh_event_routing_table_change_t*)event_data;
            ESP_LOGI(TAG, "-RT: %d, total=%d", e->rt_size_change, e-
>rt_size_new);
            break;
        }
        case MESH_EVENT_ROOT_ADDRESS: {
            const mesh_event_root_address_t *ra = (const
mesh_event_root_address_t*)event_data;
            memcpy(g_root_addr.addr, ra->addr, 6);
            g_have_root = true;
            ESP_LOGI(TAG, "Root MAC: " MACSTR, MAC2STR(g_root_addr.addr));
            break;
        }
        default:
            break;
    }
}
```

Hàm mesh_event_handler: lắng nghe toàn bộ sự kiện ESP-MESH để cập nhật trạng thái topo: khi mesh start thì ép HT20; khi nối được parent thì bật g_mesh_connected, lưu BSSID của parent, in layer và RSSI; khi mất parent thì tắt cờ và chờ tự tổ chức lại; khi nhận ROOT_ADDRESS thì lưu MAC root vào g_root_addr và bật g_have_root.

```
static void mesh_sniff_task(void *arg) {
    mesh_addr_t from;
    uint8_t rx_buf[256];
    mesh_data_t rx = {
        .data  = rx_buf,
        .size  = sizeof(rx_buf),
        .proto = MESH_PROTO_BIN,
        .tos   = MESH_TOS_P2P
    };
    int flag = 0;

    for (;;) {
        rx.size = sizeof(rx_buf);
        esp_err_t err = esp_mesh_recv(&from, &rx, 1000 / portTICK_PERIOD_MS,
&flag, NULL, 0);
        if (err == ESP_OK) {
            size_t n = (rx.size < sizeof(rx_buf)) ? rx.size : sizeof(rx_buf)-
1;
            rx_buf[n] = '\0';
            ESP_LOGI(TAG, "RX (child=" MACSTR "): %s", MAC2STR(from.addr),
(char*)rx.data);
            flag = 0;
        }
    }
}
```

Hàm mesh_sniff_task: là hàm thu các gói dữ liệu trọng ESP-MESH nó chạy vòng lặp esp_mesh_recv() để nhận gói P2P nhị phân từ các Child node, chuẩn hóa chuỗi rồi in log kèm MAC nguồn và đẩy lên Root node.

- Root node

```
static void mqtt_start_if_needed(void) {
    if (g_mqtt) return;
    esp_mqtt_client_config_t cfg = {
        .broker.address.uri = MQTT_URI,
        .credentials.username = MQTT_USERNAME,
        .credentials.authentication.password = MQTT_PASSWORD,
    };
    g_mqtt = esp_mqtt_client_init(&cfg);
```

```
    ESP_ERROR_CHECK(esp_mqtt_client_register_event(g_mqtt, ESP_EVENT_ANY_ID,
mqtt_evt_handler, NULL));
    ESP_ERROR_CHECK(esp_mqtt_client_start(g_mqtt));
    ESP_LOGI(TAG, "MQTT: connecting to %s", MQTT_URI);
}
```

Hàm mqtt_start_if_needed: Hàm khởi động MQTT nếu chưa có. Nếu g_mqtt đã tồn tại thì thoát ngay (tránh khởi tạo trùng). Nếu chưa, hàm cấu hình esp_mqtt_client_config_t với MQTT_URI, đăng ký mqtt_evt_handler để cập nhật cờ g_mqtt_connected theo MQTT_EVENT_CONNECTED/DISCONNECTED, rồi esp_mqtt_client_start() để client tự kết nối nền.

```
static void ip_evt_handler(void *arg, esp_event_base_t base, int32_t id, void
*event_data) {
    if (base == IP_EVENT && id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t *ev = (ip_event_got_ip_t *)event_data;
        ESP_LOGI(TAG, "GOT IP: " IPSTR ", GW: " IPSTR ", MASK: " IPSTR,
                 IP2STR(&ev->ip_info.ip), IP2STR(&ev->ip_info.gw),
IP2STR(&ev->ip_info.netmask));
        mqtt_start_if_needed();
    }
}

```

Hàm ip_evt_handler: Đây là bộ xử lý sự kiện IP cho giao diện STA của Root. Khi nhận IP_EVENT_STA_GOT_IP tức Root đã bắt tay được với router và được cấp địa chỉ IP hợp lệ hàm ghi log đầy đủ IP/Gateway/Netmask để tiện chẩn đoán, rồi kích hoạt đường ra ngoài bằng cách gọi mqtt_start_if_needed() nhằm khởi động client MQTT. Cách khởi động chỉ sau khi có IP giúp tránh tình trạng client kết nối sớm, lặp retry vô ích và làm cho pipeline Mesh -> MQTT ổn định hơn.

```
static void mesh_event_handler(void *arg, esp_event_base_t base, int32_t id,
void *event_data) {
    switch (id) {
        case MESH_EVENT_STARTED: {
            ESP_LOGI(TAG, "Mesh started (ROOT)");
            break;
        }
        case MESH_EVENT_PARENT_CONNECTED: {
            mesh_event_connected_t *e = (mesh_event_connected_t*)event_data;
            ESP_LOGI(TAG, "ROOT parent connected? layer=%d is_root=%s", e-
>self_layer, esp_mesh_is_root()?"YES":"NO");
            if (esp_mesh_is_root()) {
                // Đảm bảo DHCP Client chạy trên mesh STA để lấy IP
                esp_netif_dhcpc_stop(g_mesh_netif_sta);    // dừng trước,
tránh EALREADY
                esp_netif_dhcpc_start(g_mesh_netif_sta);   // khởi động lại
```

```
                }
                try_set_bw20();
                break;
            }
            case MESH_EVENT_CHILD_CONNECTED: {
                mesh_event_child_connected_t *e =
(mesh_event_child_connected_t*)event_data;
                ESP_LOGI(TAG, "Child + " MACSTR ", aid=%d", MAC2STR(e->mac), e-
>aid);
                break;
            }
            case MESH_EVENT_CHILD_DISCONNECTED: {
                mesh_event_child_disconnected_t *e =
(mesh_event_child_disconnected_t*)event_data;
                ESP_LOGW(TAG, "Child - " MACSTR ", aid=%d", MAC2STR(e->mac), e-
>aid);
                break;
            }
            default:
                break;
        }
    }
```

Hàm mesh_event_handler: Hàm mesh_event_handler xử lý các sự kiện của ESP-Mesh. Khi MESH_EVENT_STARTED cho biết mesh đã khởi động, hàm giữ cấu hình kênh ở HT20 bằng try_set_bw20() để ổn định RF. Khi MESH_EVENT_PARENT_CONNECTED, handler in log layer và, nếu node đang là ROOT, khởi động lại DHCP client trên interface STA (esp_netif_dhcpc_stop/start) để chắc chắn xin lại IP từ router sau các thay đổi liên kết. Ngoài ra, MESH_EVENT_CHILD_CONNECTED/DISCONNECTED ghi nhận child vào/ra, hữu ích để giám sát số node treo vào Root.

```
static void mesh_recv_task(void *arg) {
    mesh_addr_t from;
    uint8_t rx_buf[512];
    mesh_data_t rx = {
        .data  = rx_buf,
        .size  = sizeof(rx_buf),
        .proto = MESH_PROTO_BIN,
        .tos   = MESH_TOS_DEF
    };
    int flag = 0;
    char topic[128];

    for(;;){
        rx.size = sizeof(rx_buf);
```

```
        esp_err_t err = esp_mesh_recv(&from, &rx, portMAX_DELAY, &flag, NULL,
0);
        if (err == ESP_OK) {
            size_t n = (rx.size < sizeof(rx_buf)) ? rx.size : sizeof(rx_buf)-
1;
            rx_buf[n] = '\0';
            snprintf(topic, sizeof(topic),
"%s/%02x:%02x:%02x:%02x:%02x:%02x",
                     MQTT_BASE_TOPIC,
                     from.addr[0], from.addr[1], from.addr[2],
                     from.addr[3], from.addr[4], from.addr[5]);

            ESP_LOGI(TAG, "RX %uB from " MACSTR " -> MQTT [%s]",
(unsigned)rx.size, MAC2STR(from.addr), topic);

            if (g_mqtt_connected && g_mqtt) {
                int msg_id = esp_mqtt_client_publish(g_mqtt, topic, (const
char*)rx.data, (int)rx.size, 0, 0);
                if (msg_id < 0) {
                    ESP_LOGW(TAG, "MQTT publish failed");
                }
            } else {
                ESP_LOGW(TAG, "MQTT not connected — skip publish");
            }
            flag = 0;
        }
    }
}
```

Hàm mesh_recv_task: Đây là task nhận dữ liệu chạy vĩnh viễn ở Root. Mỗi vòng lặp nó gọi esp_mesh_recv() để lấy gói từ bất kỳ child/relay nào, kèm địa chỉ MAC nguồn và cờ flag. Sau khi nhận, code cắt chuỗi cho an toàn, tạo topic động theo mẫu mesh/<MAC_nguon> rồi kiểm tra trạng thái g_mqtt_connected. Nếu MQTT đang online, nó publish payload lên broker bằng esp_mqtt_client_publish(); nếu chưa online, nó chỉ log cảnh báo và bỏ qua gói.