BÁO CÁO THỰC HÀNH

**Bài thực hành số 05: Wireless Mesh Network**

**Môn học:** Hệ thống nhúng mạng không dây

**Lớp:** NT131.O12.MMCL.1

**THÀNH VIÊN THỰC HIỆN (Nhóm 11):**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Lê Hoàng Khánh | 21522205 |
| 2 | Ngô Thế Anh | 21521826 |

|  |
| --- |
| **Điểm tự đánh giá** |
| **10** |

**ĐÁNH GIÁ KHÁC:**

|  |  |
| --- | --- |
| Tổng thời gian thực hiện | 1 tuần. |
| Phân chia công việc | Thế Anh: Task 1, 2  Hoàng Khánh: Task 2, 3 |
| Ý kiến *(nếu có)*  + Khó khăn  + Đề xuất, kiến nghị |  |

Phần bên dưới của báo cáo này là báo cáo chi tiết của nhóm thực hiện

MỤC LỤC

[**A.** BÁO CÁO CHI TIẾT 3](#_Toc154599599)

[1. Mô tả chi tiết về cách thức vận hành của một Wireless Mesh Network. Hoàn thành tiếp tục câu 2 ở trên để các node gửi broadcast dữ liệu sensor với nhau. 3](#_Toc154599600)

[a. Mô tả chi tiết về cách thức vận hành của một Wireless Mesh Network. 3](#_Toc154599601)

[b. Các node gửi broadcast dữ liệu sensor với nhau. 3](#_Toc154599602)

[2. Sử dụng node 3 làm Root Node để kết nối thêm một mạng khác (vd: UiTiOt E3.1), viết một webserver trên trên Root Node này để cung cấp giao diện hiển thị dữ liệu cảm biến cùng với đó là khả năng điều khiển đèn gắng trên root node. 5](#_Toc154599603)

[3. Tương tự kịch bản 2, thêm đèn led ở tất các node (có thể 1 hoặc nhiều led/node). Giao diện web thêm tính năng cho phép bật/tắt các đèn này. 9](#_Toc154599604)

[**B.** TÀI LIỆU THAM KHẢO 14](#_Toc154599605)

# BÁO CÁO CHI TIẾT

## Mô tả chi tiết về cách thức vận hành của một Wireless Mesh Network. Hoàn thành tiếp tục câu 2 ở trên để các node gửi broadcast dữ liệu sensor với nhau.

### Mô tả chi tiết về cách thức vận hành của một Wireless Mesh Network.

Một Wireless Mesh Network là một mạng không dây trong đó các nút mạng (node) được kết nối với nhau theo cấu trúc lưới (mesh). Trong cấu trúc này, mỗi nút đều có thể vừa là điểm truy cập (access point) vừa là máy khách (client). Điều này cho phép các nút có thể truyền dữ liệu trực tiếp với nhau, ngay cả khi không có điểm truy cập trung tâm.

Cấu trúc của một Wireless Mesh Network trong ESP8266 có thể rất linh hoạt. Các nút mạng có thể được kết nối với nhau theo bất kỳ cách nào. Tuy nhiên, thường thì các nút mạng sẽ được kết nối với nhau theo cấu trúc cây (tree). Trong cấu trúc này, một nút mạng sẽ được chọn làm điểm truy cập gốc (root access point). Các nút mạng khác sẽ kết nối với điểm truy cập gốc này.

Wireless mesh network được ứng dụng trong các môi trường khó khăn như tình huống khẩn cấp, đường hầm, giàn khoan dầu, giám sát chiến trường, truy cập mạng công cộng, nhà thông minh, … Hầu hết các ứng dụng trong wireless mesh network tương tự như các ứng dụng trong wireless ad hoc network.

### Các node gửi broadcast dữ liệu sensor với nhau.

Để gửi thông tin của Sensor dưới dạng Json giữa các Node với nhau ta cần chỉnh sửa hàm **sendMessage().**

Cấu trúc của file Json sẽ như sau:

A black and yellow pencils

Description automatically generated

Đối với Node DHT22, node sẽ đọc thông tin nhiệt độ và độ ẩm sau đó thêm thông số này vào Json sau đó gửi đi. Thông tin sẽ được gửi BoardCast trong vòng mỗi 2 giây.

void sendMessage() {

float temp = dht.readTemperature();

float hum = dht.readHumidity();

StaticJsonDocument<2048> msgJson;

JsonObject dataJson = msgJson.createNestedObject("data");

msgJson["node"] = mesh.getNodeId();

dataJson["temp"] = temp;

dataJson["hum"] = hum;

String msg;

serializeJson(msgJson, msg);

Serial.printf("%s\n", msg.c\_str());

mesh.sendBroadcast(msg);

taskSendMessage.setInterval(TASK\_SECOND \* 2);

}

Đối với Node BH1750, node sẽ đọc thông tin về ánh sáng sau đó thêm thông số này vào Json sau đó gửi đi. Thông tin sẽ được gửi BoardCast trong vòng mỗi 2 giây.

void sendMessage() {

lightMeter.begin(BH1750::ONE\_TIME\_HIGH\_RES\_MODE, 0x23, &Wire);

float lux = lightMeter.readLightLevel();

​

StaticJsonDocument<2048> msgJson;

​

JsonObject dataJson = msgJson.createNestedObject("data");

msgJson["node"] = mesh.getNodeId();

dataJson["lux"] = lux;

​​

String msg;

serializeJson(msgJson, msg);

​

// Send message broadcast

Serial.printf("sendMessage: Sent msg=%s\n", msg.c\_str());

mesh.sendBroadcast( msg );

taskSendMessage.setInterval( TASK\_SECOND \* 2 );

}

Các node khác khi nhận được thông tin về Json sẽ in thông điệp nhận được vào Serial.

## Sử dụng node 3 làm Root Node để kết nối thêm một mạng khác (vd: UiTiOt E3.1), viết một webserver trên trên Root Node này để cung cấp giao diện hiển thị dữ liệu cảm biến cùng với đó là khả năng điều khiển đèn gắng trên root node.

Để có thể truy cập vào WebServer ta cần cho RootNode kết nối vào Station ta mong muốn đồng thời vẫn có thể kết nối vào mạng Mesh để nhận thông tin từ các Node khác.

Đâu tiên cần điều chỉnh hàm **init** ở RootNode sao cho channel hoạt động của Mesh trùng với Channel của Station. Đồng thời các Node khác cũng sẽ phải điều chỉnh Channel hoạt động về cùng Channel của Mesh ở RootNode cũng thông qua lệnh **init** tương tự như ở Root Node, VD channel hoạt động của Station là 6 ta sẽ có lệnh sau:

mesh.init( MESH\_SSID, MESH\_PASSWORD, MESH\_PORT, WIFI\_AP\_STA, 6 );

Sau đó ta cho RootNode kết nối đến Station và đặt RootNode là Root của Mesh thông qua các lệnh sau:

mesh.stationManual(STATION\_SSID, STATION\_PASSWORD);

mesh.setRoot(true);

mesh.setContainsRoot(true);

Tiếp đên WebServer, thư viện **ESPAsyncWebSrv** sẽ được sử dụng giúp hỗ trợ tạo WebSocket để giúp đẩy thông tin Sensor đến Client.

Khi Node khác gửi thông tin Sensor đến RootNode. RootNode sẽ đưa dữ liệu Json này đến Client thông qua WebSocket.

void receivedCallback( const uint32\_t &from, const String &msg ) {

Serial.printf("receive: from %u msg=%s\n", from, msg.c\_str());

ws.textAll(String("MSG:" + msg));

}

Tại Client sử dung JavaScript để sử lý dữ liệu Json từ RootNode sau đó hiện thị lên giao diện.

document.body.addEventListener('htmx:wsBeforeMessage', function(evt) {

message = evt.detail.message;

socket = evt.detail.socketWrapper;

elt = evt.detail.elt;

console.log(`receive: ${message}`);

if (message.startsWith("MSG")) {

content = JSON.parse(message.replace("MSG:", ""));

if (content.hasOwnProperty("data")) {

let hasLuxData = content["data"].hasOwnProperty("lux");

let hasTempData = content["data"].hasOwnProperty("temp");

let hasHumData = content["data"].hasOwnProperty("hum");

if (hasLuxData) {

sensorLux.innerText = content["data"]["lux"];

node1Id.innerText = content["node"];

diffLedControls(node1LedControls, content);

}

if (hasTempData) {

sensorTemp.innerText = content["data"]["temp"];

}

if (hasHumData) {

sensorHum.innerText = content["data"]["hum"];

}

}

}

}

Đối với điều khiển Led, tạo một endpoint **/toggle** khi gọi tới enpoint này RootNode gọi sẽ gọi đến hàm chuyển trạng thái của đèn và gửi trạng thái của đèn lên thông qua WebSocket đến Client để Update.

void ledControl(AsyncWebServerRequest \*request) {

if (nextLedState.compareTo(UNKNOWN) == 0) {

nextLedState = ON;

}

if (nextLedState.compareTo(ON) == 0) {

digitalWrite(ledPin, HIGH);

ledState = nextLedState;

nextLedState = OFF;

Serial.println("Light is up");

} else {

digitalWrite(ledPin, LOW);

ledState = nextLedState;

nextLedState = ON;

Serial.println("Light is down");

}

ws.textAll(String("LED:" + ledState));

request->send(200, "text/plain", ledState);

}

Code HTMX, gọi đến endpoint và update trạng thái.

<p>Root LED: <span id="root-led-state"></span></p>

<button id="led-toggle-btn"

hx-get="/toggle"

hx-trigger="click"

hx-target="#root-led-state"

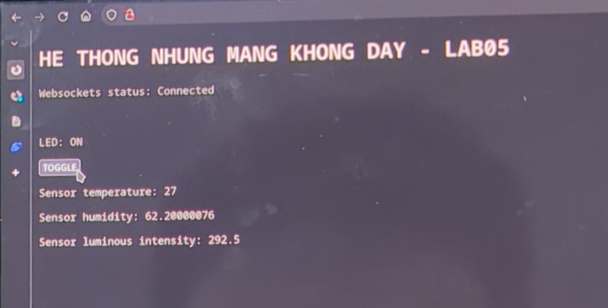
hx-swap="innerHTML"

>

TOGGLE

</button>

Qua các bước trên, ta sẽ xem được thông tin mới nhất từ Sensor, kèm vào đó à trạng thái của Led hiện tại và chuyển trạng thái thông qua một nút.



## Tương tự kịch bản 2, thêm đèn led ở tất các node (có thể 1 hoặc nhiều led/node). Giao diện web thêm tính năng cho phép bật/tắt các đèn này.

Để bất tắt đền ở các Node ta cần thiết kế thông điệp RootNode gửi để điều khiển led ở các Node còn lại.

{

"action": ledToggle,

"node": nodeid,

"id": id

}

Trong đó **action** sẽ hành đồng cần thực hiện, nếu đúng là **ledToggle** sẽ thực hiện chuyển trạng thái đèn. **Node** là nodeid của node cần bật tắt đèn. Id là id của đèn cần bật tắt.

Khi ấn nút bật tắt đèn. Client sẽ thực hiện GET đến endpoint **/ledToggle/nodeid/id** trong đó nodeid là id của node đích, id là id của đèn.

targetLedToggleBtn.onclick = () => {

fetch(`/ledToggle/${nodeId}/${ledId}`, {method: 'GET'});

};

Sau khi gọi đến Endpoint này RootNode sẽ BoardCast thông điệp Json có cấu trúc như trên đến các Node còn lại.

server.on("/ledToggle/3038481794/0", HTTP\_GET, [] (AsyncWebServerRequest \*request) {

String nodeId("3038481794");

String ledIdx(0);

StaticJsonDocument<2048> msgJson;

msgJson["action"] = "ledToggle";

msgJson["node"] = nodeId;

msgJson["id"] = ledIdx;

String msg;

serializeJson(msgJson, msg);

if (!mesh.sendBroadcast(msg)) {

request->send(400, "text/plain", "Error");

}

request->send(200, "text/plain", "Ok");

});

server.on("/ledToggle/3038481794/1", HTTP\_GET, [] (AsyncWebServerRequest \*request) {

String nodeId("3038481794");

String ledIdx(1);

StaticJsonDocument<2048> msgJson;

msgJson["action"] = "ledToggle";

msgJson["node"] = nodeId;

msgJson["id"] = ledIdx.toInt();

String msg;

serializeJson(msgJson, msg);

if (!mesh.sendBroadcast(msg)) {

request->send(400, "text/plain", "Error");

}

request->send(200, "text/plain", "Ok");

});

server.on("/ledToggle/3038482123/0", HTTP\_GET, [] (AsyncWebServerRequest \*request) {

String nodeId("3038482123");

String ledIdx(0);

StaticJsonDocument<2048> msgJson;

msgJson["action"] = "ledToggle";

msgJson["node"] = nodeId;

msgJson["id"] = ledIdx;

String msg;

serializeJson(msgJson, msg);

if (!mesh.sendBroadcast(msg)) {

request->send(400, "text/plain", "Error");

}

request->send(200, "text/plain", "Ok");

});

Các Node còn lại sẽ kiểm tra nếu đúng là NodeID của mình và có ID đèn mình hiện có sẽ thực hiện chuyển trạng thái của đèn là Update trạng thái của mình.

void ledControl(int ledIdx) {

int \_ledPin = ledPin[ledIdx];

String \_ledState = ledState[ledIdx];

String \_nextLedState = nextLedState[ledIdx];

if (\_nextLedState.compareTo(UNKNOWN) == 0) {

\_nextLedState = ON;

}

if (\_nextLedState.compareTo(ON) == 0) {

digitalWrite(\_ledPin, HIGH);

\_ledState = \_nextLedState;

\_nextLedState = OFF;

} else {

digitalWrite(\_ledPin, LOW);

\_ledState = \_nextLedState;

\_nextLedState = ON;

}

ledState[ledIdx] = \_ledState;

nextLedState[ledIdx] = \_nextLedState;

}

// Needed for painless library

void receivedCallback( uint32\_t from, String &msg ) {

Serial.printf("receive: node=%u, msg=%s\n", from, msg.c\_str());

StaticJsonDocument<2048> msgJson;

deserializeJson(msgJson, msg);

const String action(msgJson["action"]);

const String nodeId(msgJson["node"]);

if (action.compareTo(String("ledToggle")) != 0) {

return;

}

if (nodeId.compareTo(String(mesh.getNodeId())) != 0) {

Serial.printf("Not current node=%s\n", nodeId.c\_str());

return;

}

const int ledIdx = String(msgJson["id"]).toInt();

if (ledIdx > ledCount || ledIdx < 0) {

Serial.printf("Led out of bound ledCount=%d, ledIdx=%d\n", ledCount, ledIdx);

return;

}

ledControl(ledIdx);

Serial.printf("Toggled ledIdx %d\n", ledIdx);

}

Để Client có thể Update trạng thái đèn, cần thêm thông tin về trạng thái vào Json gửi đi. Khi đó Json sẽ có cấu trúc sau.

A computer code with text

Description automatically generated

Hàm **sendMessage** lúc này sẽ như sau.

void sendMessage() {

float temp = dht.readTemperature();

float hum = dht.readHumidity();

StaticJsonDocument<2048> msgJson;

JsonObject dataJson = msgJson.createNestedObject("data");

msgJson["node"] = mesh.getNodeId();

dataJson["temp"] = temp;

dataJson["hum"] = hum;

JsonArray ledsArr = dataJson.createNestedArray("leds");

for (int ledIdx = 0; ledIdx < ledCount; ledIdx++) {

JsonObject ledJson = ledsArr.createNestedObject();

ledJson["id"] = ledIdx;

ledJson["state"] = ledState[ledIdx].c\_str();

}

String msg;

serializeJson(msgJson, msg);

Serial.printf("%s\n", msg.c\_str());

mesh.sendBroadcast(msg);

taskSendMessage.setInterval(TASK\_SECOND \* 3);

}

Giao diện sau khi chỉnh sửa.

A computer screen with text and numbers

Description automatically generated

# TÀI LIỆU THAM KHẢO