**Course: THCNTT3**

**Project: OmniCare**

**Author: Hieu Pham**

**ArduinoFire**

Timeout: 10s

**ArduinoAir**

Timeout: 10s

**ArduinoPIR**

Timeout: 10s

**@ Kiểm tra kết nối thiết bị**

Trong đồ án 2 tôi đã lạm dụng Serial.print() để truyền tín hiệu liên tục giữa các arduino, điều này đã gây tràn buffer của cổng TX.

Sang đồ án 3 này, tôi đã định sử dụng một dây nối riêng (ngoài dây Serial) chỉ để đảm nhận việc cho biết arduino có kết nối hay không. Tuy nhiên, việc này không đảm bảo được trạng thái của tất cả thiết bị sẽ được cập nhập khi mà esp8266 bị ngắt điện.

Tôi quyết định để timeout cho các signal của từng arduino, nếu sau khoảng timeout đó mà không có signal mới thì xem như arduino đó đã mất kết nối.

**@ Https connection với ESP8266**

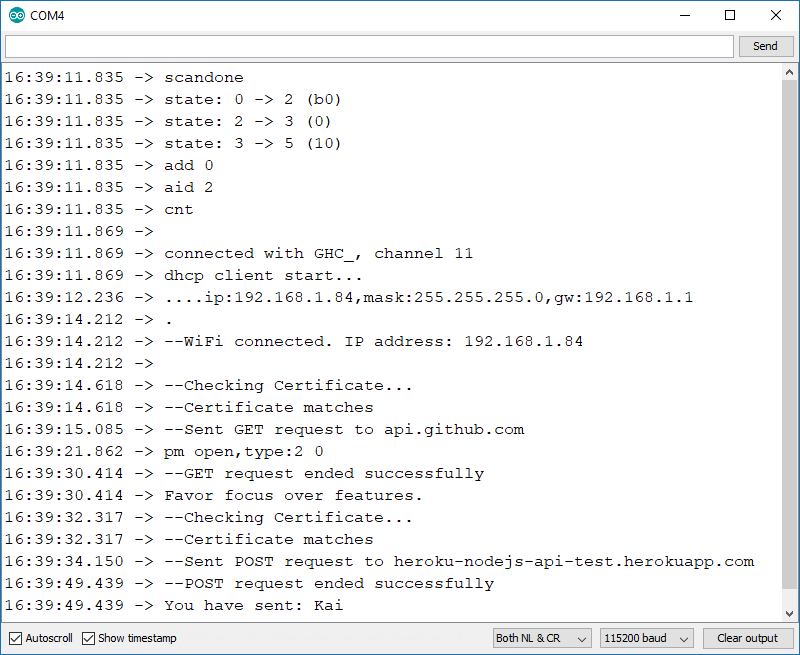
Certificate của https website sẽ được reset sau 90 ngày.

Tôi đã có thể kết nối tới các api dùng giao thức https. Tuy nhiên, tôi gặp vấn đề với việc xác định fingerprint của các https website. Ví dụ, trang api.github.com có thumbprint (lấy từ certificate của chrome) khác hoàn toàn với fingerprint cần để kết nối (lý do có thể liên quan đến việc subdomain có certificate khác domain chính). Tôi quyết định không sử dụng fingerprint để xác thực server cho project này.

Ta có thể đặt Serial.setDebugOutput(true); trong setup() để xem 2 fingerprint khi chúng không đồng nhất thông qua màn hình Serial.

Từ lúc tạo request cho tới lúc nhận response của cả get và post https với esp8266 là hơn **16s**.

Tuy nhiên, chỉ mất **0.5s** để gửi request nếu ta không quan tâm đến response.



String **sendHttpsGet**(WiFiClientSecure httpsClient, String host, int port, String url) {

if (httpsClient.connect(host.c\_str(), port)) {

httpsClient.println("GET " + url + " HTTP/1.1");

httpsClient.println("Host: " + host);

httpsClient.println("User-Agent: ESP8266/Hieu");

httpsClient.println("Connection: close\r\n");

String response = httpsClient.readString();

int bodypos = response.indexOf("\r\n\r\n") + 4;

return response.substring(bodypos);

}

else {

return "ERROR";

}

}

String **sendHttpsPost**(WiFiClientSecure httpsClient, String host, int port, String url, String data) {

if (httpsClient.connect(host, port)) {

httpsClient.println("POST " + url + " HTTP/1.1");

httpsClient.println("Host: " + host);

httpsClient.println("User-Agent: ESP8266/Hieu");

httpsClient.println("Connection: close");

httpsClient.println("Content-Type: **text/plain**; charset=UTF-8");

httpsClient.print("Content-Length: ");

httpsClient.println(data.length());

httpsClient.println();

httpsClient.println(data);

delay(10);

String response = httpsClient.readString();

int bodypos = response.indexOf("\r\n\r\n") + 4;

return response.substring(bodypos);

}

else {

return "ERROR";

}

}

**@ Thư viện ArduinoQueue**

Có nhiều thư viện hỗ trợ queue trong môi trường C++, tôi sẽ dùng nó để enqueue signal, và cứ sau 1 giây sẽ dequeue 1 signal và đồng thời gửi signal đó sang cho nodeMCU. Mục đích là để tránh việc nodeMCU không kịp gửi hết post request chứa signal đến server (1 request tốn 0.5s, trong 0.5s đó esp8266 không loop tiếp).

**@ NRF24L01 Transceiver**

Có 2.4 GHz band, 1.9 - 3.6V, có thể giao tiếp tối đa 6 NRF2401 khác, khoảng cách là 100m.

Tutorial: https://www.youtube.com/watch?v=7rcVeFFHcFM

**# Arduino Serial**

**@ Gửi một chuỗi string giữa 2 Arduino**

Khi dùng serial để giao tiếp giữa 2 Arduino, ta phải đảm bảo 2 arduino này sử dụng chung 1 nguồn điện (ví dụ như được lấy từ cùng một máy tính…), nếu không thì Serial không hoạt động.

Vì tốc độ loop của arduino nhanh hơn quá trình nhận Serial signal, nên khi ta truyền 1 string dài bằng Serial.write() từ Arduino 1 thì khi Arduino 2 nhận bằng Serial.read(), nó sẽ chỉ đọc được một 1 char đầu rồi thực thi tiếp các dòng lệnh tiếp theo. Vì vậy, tôi sử dụng hàm while để chờ đến khi nào đọc hết Serial signal được gửi rồi mới cho phép Aruidno 2 được loop tiếp.

String readSerial() {

String str = "";

while(Serial.available()) {

str += (char) Serial.read();

delay(1);

}

return str;

}

Lưu ý: Serial.read() chỉ đọc 1 byte tại 1 thời điểm, và sẽ remove 1 byte đó khỏi buffer sau khi đọc.

Tuy nhiên, trong quá trình arudino dừng loop để chạy hàm while, nếu các cổng Serial khác được gửi signal sang thì signal đó sẽ bị mất. Arduino không thể đồng thời nhận 2 Serial signals từ 2 cổng khác nhau (Đây chỉ là phỏng đoán, vì trong đồ án 2 tôi nhận thấy có tình trạng arduino giữ lại serial signal mà không hề xóa hết các signal chưa được read, nhưng giữ lại trong bao lâu thì tôi không tìm được tài liệu).

**@ Serial.flush()**

Tutorial: https://www.baldengineer.com/when-do-you-use-the-arduinos-to-use-serial-flush.html

Serial.flush bắt buộc arduino phải đợi quá trình transmit dữ liệu thực hiện xong rồi mới đi đến dòng lệnh tiếp theo. Nếu không có Serial.flush, quá trình transmit sẽ diễn ra ở background, arduino thực hiện tiếp câu lệnh tiếp theo. Dữ liệu sẽ được đưa vào buffer và transmit ở background.

**@ Buffer**

Từ những tài liệu tôi đọc trên mạng, tôi có thể đưa dến kết luận rằng Serial có một nơi chứa dữ liệu gọi lại **buffer**.

Đối với Transmit arduino (TX pin): Khi hàm Serial.write("value") được gọi, value đó sẽ được chuyển vào buffer, arduino sẽ thực hiện dòng lện tiếp theo; trong lúc đó, Serial sẽ transmit dần từng byte của value trong buffer ở background. Nếu muốn arduino chờ quá trình transmit thì dùng Serial.flush (đọc ở dưới).

Đối vối Receive arduino (RX pin): value mà được transmit từ một arduino khác sẽ được lưu vào buffer. Nếu buffer đó không rỗng thì Serial.available() sẽ trả về số lượng byte có trong buffer. Hàm Serial.read() sẽ dequeue 1 byte ra khỏi buffer.