**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**
**ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH**

**KHOA ĐIỆN – ĐIỆN TỬ**


BÁO CÁO MINI PROJECT
THIẾT KẾ VÀ PHÁT TRIỂN ỨNG DỤNG IOT

# DESIGN AND IMPLEMENTATION OF SMART-HOME APPLICATION USING CONTIKI

***GVHD: Võ Quế Sơn***
***Nhóm 02 – Lớp L01***
***Danh sách thành viên:***

| **Họ và Tên** | **MSSV** | **Đóng góp** |
|---|---|---|
| Lê Công Thịnh | 2112363 | 100% |
| Nguyễn Nhật Minh | 2114061 | 100% |

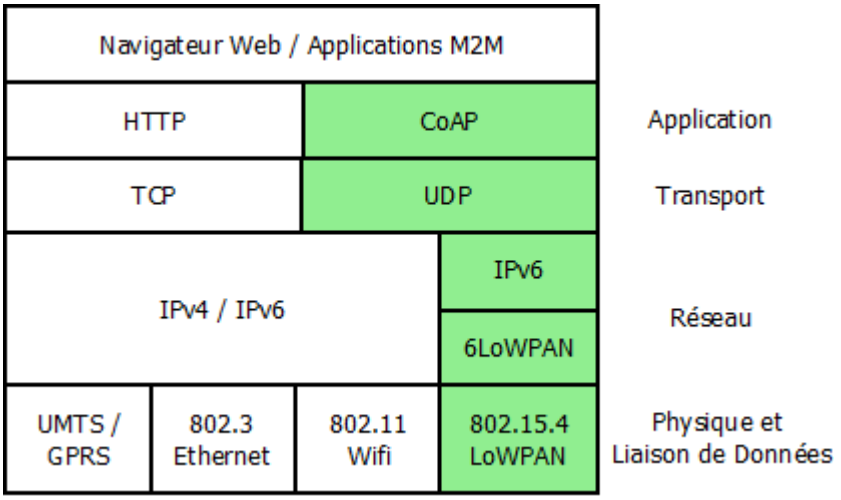
*TP. Hồ Chí Minh, tháng 12, năm 2024*

# I. Cơ sở lý thuyết.

## 1. CoAP (Constrained Application Protocol)

CoAP là một giao thức ứng dụng được thiết kế đặc biệt cho các mạng có băng thông thấp, độ trễ cao và các thiết bị có tài nguyên hạn chế trong môi trường Internet of Things (IoT). Giao thức này nhằm cung cấp một phương thức truyền thông hiệu quả, đáng tin cậy và tiết kiệm năng lượng cho các thiết bị như cảm biến, bộ điều khiển và các thiết bị nhúng.

CoAP chủ yếu hoạt động  ở tầng Application (layer 7) và tầng Transport (layer 4) . CoAP là một giao thức tầng ứng dụng, tương tự như HTTP nhưng được tối ưu hoá cho các thiết bị IoT hạn chế. Sử dụng mô hình RESTful cho phép các thiết bị thực hiện các phương thức như GET, POST, PUT và DELETE để thao tác với các tài nguyên. CoAP sử dụng UDP (User Datagram Protocol) ở tầng Transport thay vì TCP để giảm thiểu độ trễ và chi phí tài nguyên. Các cơ chế xác nhận và tái truyền lại gói tin được triển khai để đảm bảo độ tin cậy.




*Hình 1: Mô hình OSI của CoAP*

Sử dụng CoAP trong dự án Smart Home:

- Tối ưu hoá cho các thiết bị hạn chế: CoAP được thiết kế để hoạt động trên các thiết bị có tài nguyên hạn chế như cảm biến, đèn chiếu sáng, khóa cửa thông minh và các thiết bị khác trong hệ thống nhà thông minh.
- Hiệu quả băng thông: Sử dụng UDP giúp giảm thiểu yêu cầu băng thông và giảm độ trễ, phù hợp với các mạng không dây trong môi trường nhà thông minh.

- Kiến trúc RESTful: CoAP sử dụng mô hình RESTful, dễ dàng tích hợp với các dịch vụ web và các hệ thống quản lý nhà thông minh khác.
- Tính năng quan sát: Tính năng quan sát của CoAP cho phép các thiết bị nhận thông báo khi có thay đổi trong tài nguyên, giúp cải thiện khả năng giám sát và điều khiển trong thời gian thực.
- Hỗ Trợ Multicast: Khả năng sử dụng multicast giúp gửi thông điệp tới nhiều thiết bị cùng một lúc, hữu ích cho việc điều khiển đồng thời nhiều thiết bị trong hệ thống nhà thông minh.
- Bảo mật: CoAP hỗ trợ DTLS để mã hóa và bảo vệ dữ liệu truyền tải, đảm bảo an toàn thông tin trong hệ thống nhà thông minh.

# II. Chi Tiết Về Mini Project:

## 1. Implement Smart Home Application With CoAP Using Contiki OS:

Về mô hình ứng dụng IOT trong nhà thông minh chúng em lựa chọn thì hệ thống sẽ chạy trên giao thức CoAP, một giao thức ứng dụng phổ biến được thiết kế đặc biệt cho các mạng có băng thông thấp, độ trễ cao và các thiết bị có tài nguyên hạn chế trong môi trường Internet of Things (IoT). Mô hình IOT SmartHome cơ bản này được xây dựng bao gồm các node cảm biến thu thập thông tin, một node border router đóng vai trò giao tiếp với mạng internet bên ngoài và sử dụng tiện ích mở rộng (add-on) Copper cho trình duyệt Mozilla Firefox được thiết kế để tương tác với các thiết bị sử dụng giao thức CoAP thông qua border router - đóng vai trò như một node client server mà người dùng có thể tương tác và điều khiển các node trong mạng nhà thông minh thông qua các lệnh trên Copper như: GET, POST, PUT và DELETE.

**Mô hình Smart Home sẽ có 1 node border router và 6 node ứng dụng chính:**

- **CO_sensor (CoAP server node)**: Node này mô phỏng một máy chủ cảm biến khí CO (Carbon Monoxide) sử dụng giao thức CoAP, cung cấp thông tin nồng độ CO giả lập qua tài nguyên /Sensor/co_sensor. Nó sinh ra giá trị CO ngẫu nhiên (1-120 ppm) và trả lời client với thông báo tương ứng: "Safe" (dưới 50 ppm), "Warning" (51-100 ppm), hoặc "Danger" (trên 100 ppm). Máy chủ cập nhật giá trị CO mỗi 10 phút bằng timer định kỳ và có thể được sử dụng để giám sát chất lượng không khí trong các mạng IoT hoặc môi trường nhà thông minh.
- **Door_sensor (CoAP server node)**: Node này cho phép giám sát và điều khiển trạng thái cửa qua tài nguyên /Sensor/door_sensor. Người dùng có thể gửi yêu

cầu GET để lấy trạng thái hiện tại của cửa (mở hoặc đóng) và yêu cầu POST với payload "OPEN" hoặc "CLOSE" để thay đổi trạng thái cửa. Trạng thái cửa cũng có thể được thay đổi bằng cách nhấn nút phần cứng, hoạt động như một cảm biến để chuyển đổi giữa trạng thái "Mở" và "Đóng". Máy chủ này có thể tích hợp vào hệ thống IoT để quản lý an ninh hoặc điều khiển thông minh.

- **Fan_sensor (CoAP server node)**: Chương trình này triển khai một máy chủ cảm biến quạt sử dụng giao thức CoAP, cho phép giám sát và điều khiển trạng thái quạt thông qua tài nguyên /Sensor/fan_sensor. Người dùng có thể gửi yêu cầu GET để lấy trạng thái hiện tại của quạt (Bật hoặc Tắt) và gửi yêu cầu POST với payload "TURNON" hoặc "TURNOFF" để thay đổi trạng thái quạt. Ngoài ra, trạng thái quạt có thể được chuyển đổi bằng cách nhấn nút phần cứng, đóng vai trò như một cảm biến, để bật hoặc tắt quạt. Chương trình này phù hợp cho các hệ thống IoT quản lý thiết bị gia đình thông minh.
- **Light_switch (CoAP server node)**: Node này triển khai một máy chủ CoAP để điều khiển và giám sát trạng thái của công tắc đèn thông qua tài nguyên /Control/Light_switch. Người dùng có thể gửi yêu cầu GET để kiểm tra trạng thái hiện tại của đèn (Bật hoặc Tắt) và yêu cầu POST với payload "TURNON" hoặc "TURNOFF" để bật hoặc tắt đèn. Ngoài ra, trạng thái đèn cũng có thể được chuyển đổi thủ công bằng cách nhấn một nút phần cứng được cấu hình làm cảm biến, cho phép bật hoặc tắt đèn. Chương trình thích hợp cho các ứng dụng IoT quản lý thiết bị thông minh trong nhà.
- **Temp_sensor (CoAP server node)**: Node này triển khai một máy chủ CoAP giả lập cảm biến nhiệt độ, cung cấp tài nguyên /Sensor/temp_solar để truy xuất thông tin nhiệt độ. Khi nhận yêu cầu GET, chương trình trả về nhiệt độ giả lập (20–42°C) và thông báo tình trạng ("Temperature Normal" hoặc "High Temp --> Turn Fan" nếu nhiệt độ vượt ngưỡng 26°C). Nhiệt độ được cập nhật định kỳ mỗi 10 phút bằng bộ định thời (timer), và giá trị được lấy từ hàm sinh số ngẫu nhiên. Chương trình minh họa cách sử dụng CoAP trong ứng dụng IoT để giám sát nhiệt độ.
- **CO_Fan_Client (CoAP client node)**: Node triển khai một client CoAP giám sát mức độ CO (carbon monoxide) và điều khiển quạt dựa trên dữ liệu từ một cảm biến CO. Client định kỳ (12 phút) gửi yêu cầu GET tới cảm biến CO (địa chỉ IPv6 được định nghĩa sẵn) để nhận thông tin mức CO (ppm). Nếu mức CO vượt quá 100 ppm, chương trình gửi yêu cầu POST tới cảm biến quạt (fan sensor) để bật quạt bằng cách truyền thông điệp "TURNON". Toàn bộ quá trình được thực hiện bằng giao thức CoAP với khả năng xử lý phản hồi từ server.

Tất cả đều được lập trình trên hệ điều hành Contiki OS và được đánh giá, mô phỏng trên Cooja - một công cụ mô phỏng mạng được tích hợp trong hệ điều hành

Contiki, được thiết kế để hỗ trợ nghiên cứu và phát triển các ứng dụng mạng cảm biến không dây. Và thông qua Copper để đánh giá, kiểm thử thay thế cho User Interface trước khi thực hiện trên hệ thống thực.

## 2. Chi tiết về các node và hoạt động của các node trong mạng:

### a. CO_sensor:

- Khởi tạo và mô phỏng cảm biến CO: Biến co_level đại diện cho nồng độ khí CO được mô phỏng. Hàm get_fake_co() trả về một giá trị ngẫu nhiên trong khoảng từ 1 đến 120 ppm để giả lập giá trị nồng độ khí CO.
- Xử lý yêu cầu CoAP: Hàm res_get_handler() xử lý các yêu cầu GET từ client qua giao thức CoAP. Khi có yêu cầu từ client: Nồng độ khí CO hiện tại được lấy từ hàm get_fake_co().
- Thông báo trả về cho client được định dạng theo giá trị nồng độ CO: Dưới 50 ppm ~ An toàn. Từ 51 đến 100 ppm ~ Cảnh báo. Trên 100 ppm ~ Nguy hiểm. Phản hồi được gửi về client Cooper qua CoAP.
- Tài nguyên cảm biến CO: Tài nguyên cảm biến được đăng ký với REST engine qua rest_activate_resource() với đường dẫn tài nguyên là /Sensor/co_sensor. Tài nguyên này mô phỏng cảm biến CO và cung cấp dữ liệu qua giao thức CoAP.
- Timer định kỳ: Timer định kỳ (etimer) được sử dụng để mô phỏng việc đo nồng độ khí CO mỗi 10 phút (600 giây). Mỗi lần timer hết hạn, giá trị CO mới được cập nhật và in ra màn hình debug.
- Cơ chế chính của chương trình: Quy trình chính (PROCESS_THREAD) chứa vòng lặp chờ sự kiện: Chờ sự kiện từ timer định kỳ. Khi timer hết hạn, nó: Lấy giá trị CO mới từ cảm biến giả lập. In giá trị này ra màn hình. Đặt lại timer để tiếp tục chu kỳ.

### b. Fan_sensor:

- Khởi tạo trạng thái quạt: Biến fan_state được sử dụng để lưu trữ trạng thái hiện tại của quạt, với giá trị 1 là BẬT (ON) và 0 là TẮT (OFF). Mặc định, trạng thái quạt khởi tạo là ON.
- Xử lý yêu cầu GET qua giao thức CoAP: Hàm res_get_handler() xử lý các yêu cầu GET từ client qua giao thức CoAP. Khi client gửi yêu cầu, trạng thái hiện tại của quạt (ON hoặc OFF) được lấy từ biến fan_state. Một thông báo tương ứng ("Fan State: ON" hoặc "Fan State: OFF") được tạo và gửi lại cho client cùng với header HTTP phù hợp.

- Xử lý yêu cầu POST qua giao thức CoAP: Hàm res_post_handler() xử lý các yêu cầu POST từ client. Dữ liệu payload từ yêu cầu POST được kiểm tra. Nếu payload chứa chuỗi "TURNON", quạt được bật (ON), và nếu chứa "TURNOFF", quạt được tắt (OFF). Nếu không, lệnh không hợp lệ và quạt giữ nguyên trạng thái. Sau đó, trạng thái cập nhật của quạt được trả về cho client.
- Tài nguyên quạt: Tài nguyên quạt được định nghĩa thông qua macro RESOURCE() với tên là fan_sensor. Tài nguyên này được kích hoạt qua rest_activate_resource() với đường dẫn /Sensor/fan_sensor, cho phép client tương tác với trạng thái quạt thông qua các phương thức GET và POST.
- Kích hoạt và xử lý nút nhấn: Cảm biến nút nhấn được kích hoạt bằng lệnh SENSORS_ACTIVATE(button_sensor). Trong vòng lặp sự kiện chính của quy trình (PROCESS_THREAD), khi nút nhấn được kích hoạt, trạng thái quạt sẽ tự động chuyển đổi giữa ON và OFF. Trạng thái mới được in ra màn hình để debug.
- Cơ chế chính của chương trình: Quy trình chính rest_server_example khởi tạo REST engine qua rest_init_engine() và kích hoạt tài nguyên quạt. Vòng lặp sự kiện chờ xử lý các sự kiện như nút nhấn, cho phép người dùng điều khiển quạt thông qua cảm biến hoặc giao thức CoAP.

**c. Door_sensor & Light_switch:** Hoạt động giống cơ chế của Fan_sensor, thầy có thể xem thêm trong code ở link GitHub bên dưới (mục 4).

**d. Temp_sensor:** Hoạt động giống cơ chế của CO_sensor, thầy có thể xem thêm trong code ở link GitHub bên dưới (mục 4).

**e. CO_Fan_Client:**

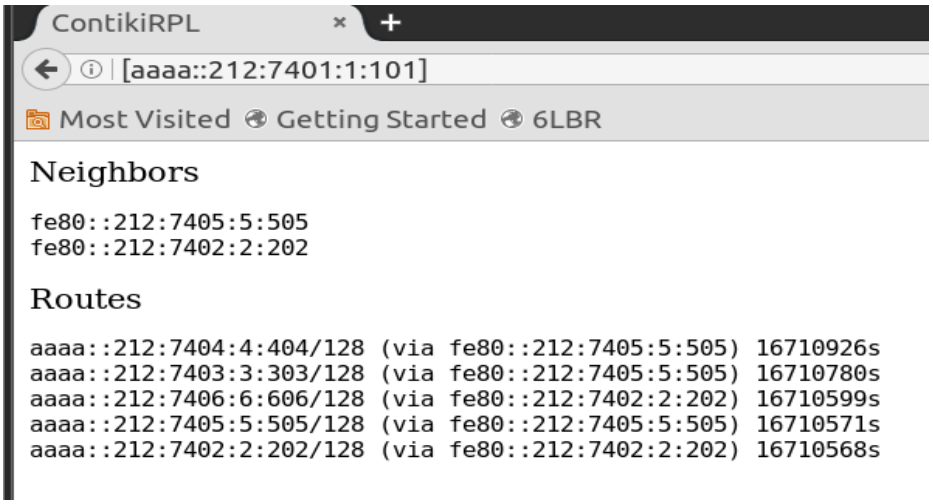
- Cấu hình địa chỉ và khởi tạo CoAP engine: Hai địa chỉ IPv6 được định nghĩa trước cho node cảm biến CO và node cảm biến quạt. Hàm CO_SENSOR_NODE() và FAN_SENSOR_NODE() thiết lập các địa chỉ này. CoAP engine được khởi tạo thông qua hàm coap_init_engine() để hỗ trợ giao tiếp qua giao thức CoAP.
- Gửi yêu cầu GET đến cảm biến CO: Bộ đếm thời gian định kỳ etimer được đặt giá trị là QUERY_INTERVAL (720 giây) để gửi yêu cầu GET đến cảm biến CO. Mỗi khi hết hạn timer, một gói tin CoAP GET được tạo và gửi đến tài nguyên "Sensor/co_sensor" trên node cảm biến CO.
- Xử lý phản hồi từ cảm biến CO: Hàm co_sensor_response_handler() được gọi khi nhận được phản hồi từ node cảm biến CO.

- Nội dung phản hồi (payload) được phân tích: Nếu giá trị CO vượt quá 100 ppm, chương trình xác định môi trường nguy hiểm. Một gói tin CoAP POST được gửi đến tài nguyên "Sensor/fan_sensor" trên node quạt để bật quạt (TURNON). Nếu giá trị CO an toàn (≤ 100 ppm), không có hành động nào được thực hiện. Nếu phản hồi không hợp lệ hoặc không thể phân tích được giá trị CO, chương trình ghi nhận lỗi.
- Gửi yêu cầu POST để bật quạt: Gói tin POST được tạo trong hàm co_sensor_response_handler(). Nội dung payload "TURNON" được gửi đến tài nguyên của quạt thông qua hàm COAP_BLOCKING_REQUEST() với địa chỉ node quạt đã cấu hình. Cơ chế chính của chương trình: Quy trình chính (PROCESS_THREAD) sử dụng vòng lặp chờ sự kiện: Chờ timer hết hạn: Sau mỗi chu kỳ QUERY_INTERVAL, một yêu cầu GET được gửi đến cảm biến CO.
- Xử lý phản hồi: Dựa trên giá trị CO trong phản hồi, chương trình quyết định bật quạt nếu cần.
- Đặt lại timer: Đảm bảo chu kỳ kiểm tra lặp lại.

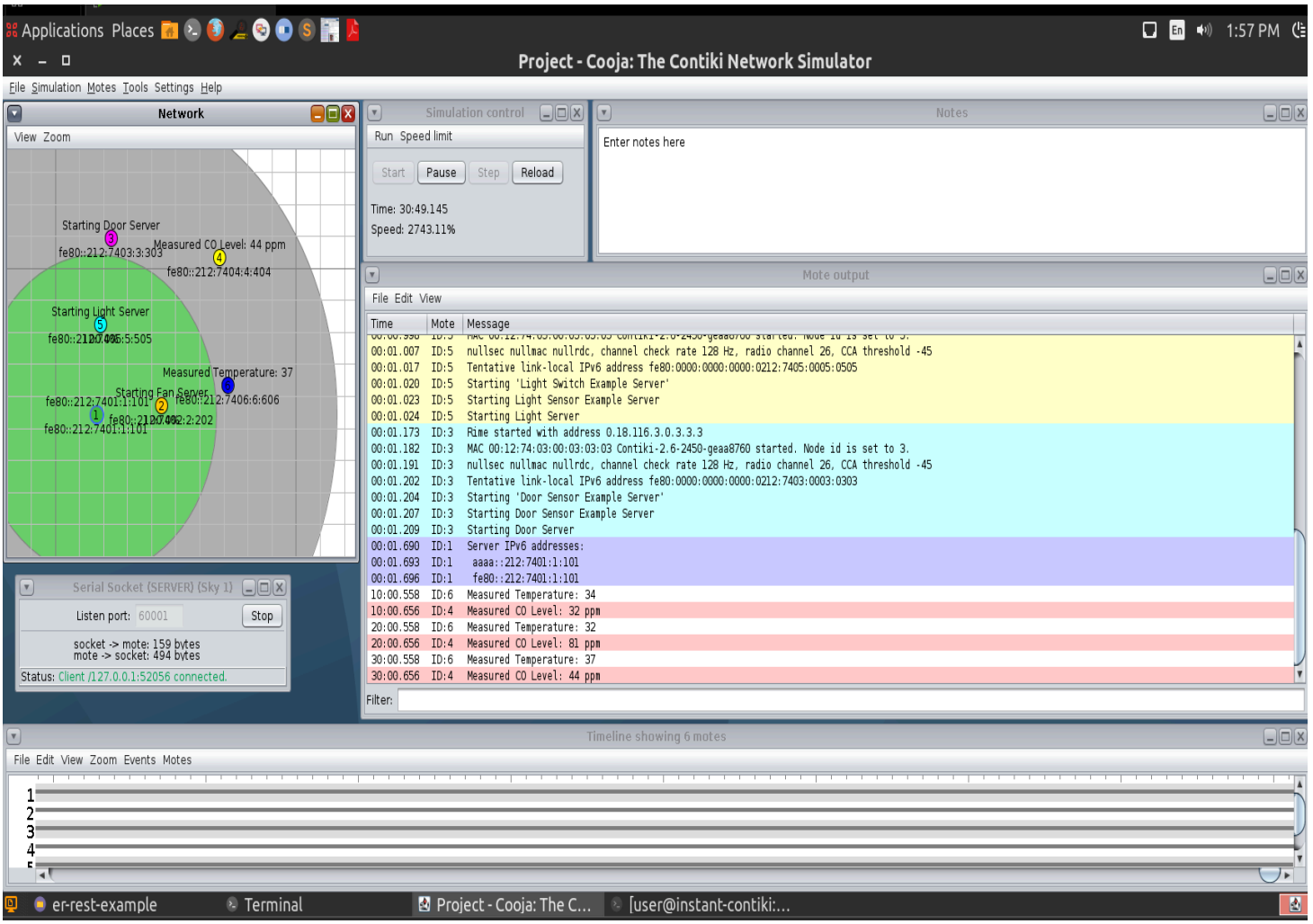
## 3. Mô phỏng trên Cooja:

**LƯU Ý:** *CÁC MÔ PHỎNG DƯỚI NÀY KHÔNG BAO GỒM NODE CO_Fan_Client. Vì khi phát triển ứng dụng này, chúng em chỉ chạy được nó trên Contiki NG và khi tích hợp với các node hiện tại thì chưa đạt được đầy đủ tính năng mong muốn.*

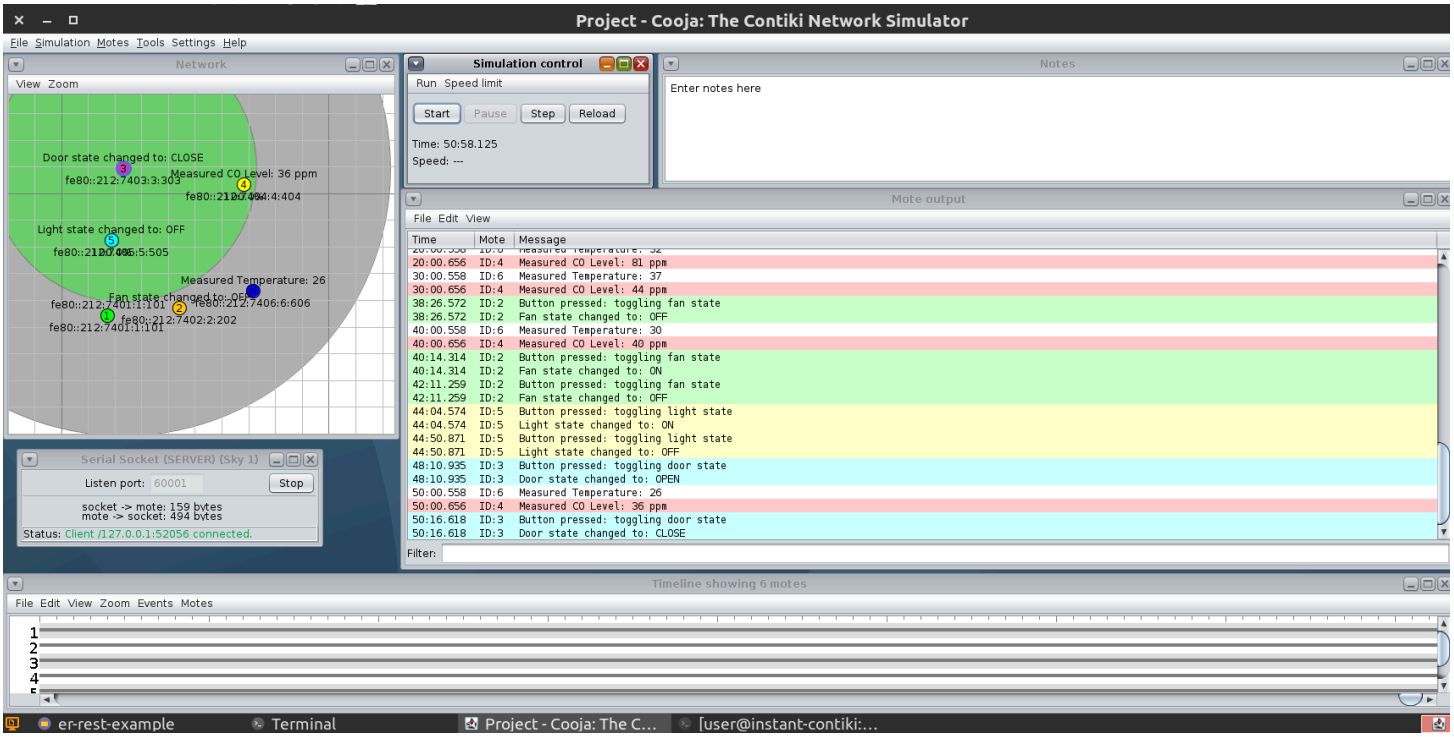
**HÌNH 2:** Bảng neighbors và routing của các node đến node border router → đều đúng so với lý thuyết.



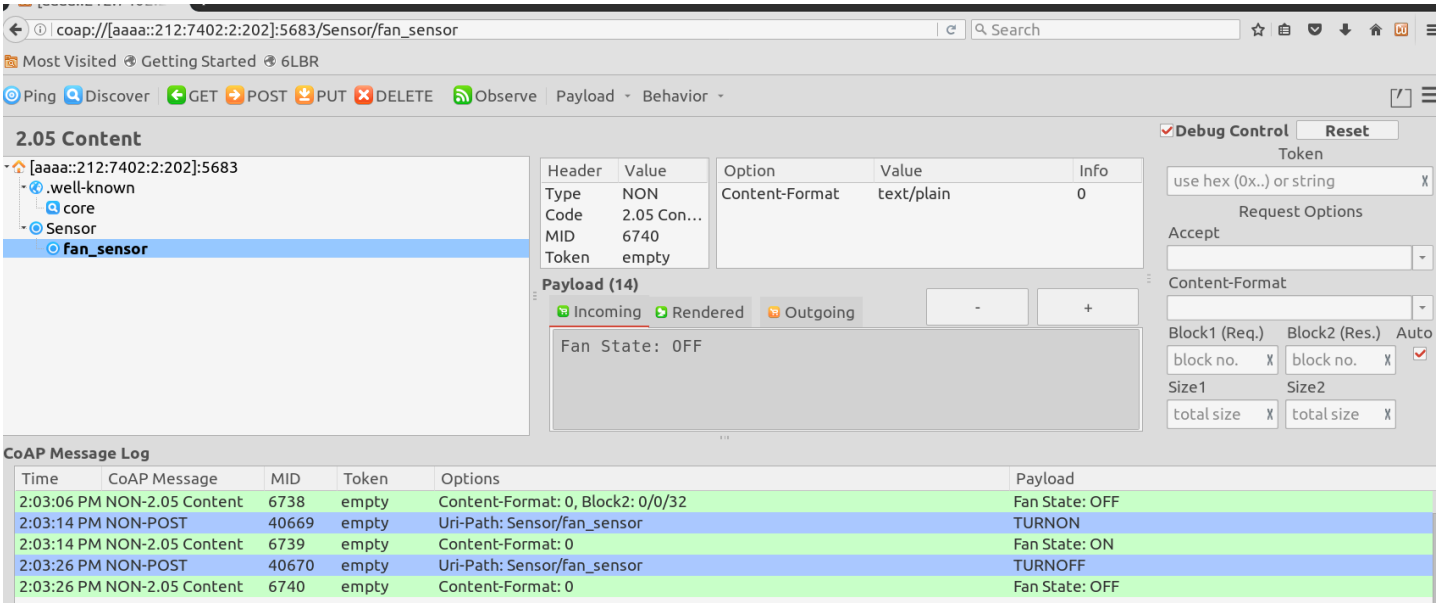
**HÌNH 3:** Khởi động mạng → Các giá trị từ cảm biến nhiệt độ và cảm biến khí CO lần lượt được đọc và xử lý sau mỗi chu kỳ 10 phút. Nếu vượt ngưỡng thì quạt trong nhà sẽ bật (node Fan).



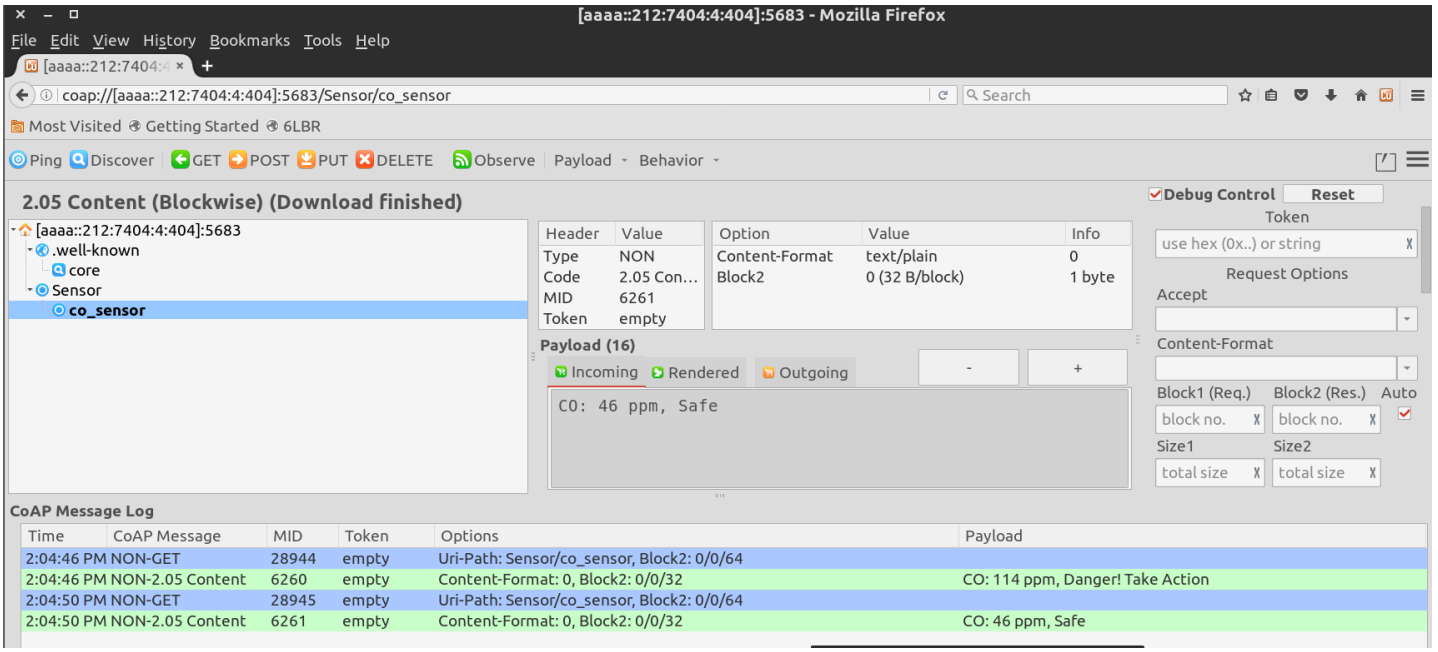
**HÌNH 4:** Nếu nhấn nút nhấn ở các node Light, Door và Fan thì trạng thái chúng sẽ thay đổi tương ứng (bật tắt, đóng mở), đồng thời in thông tin lên message.



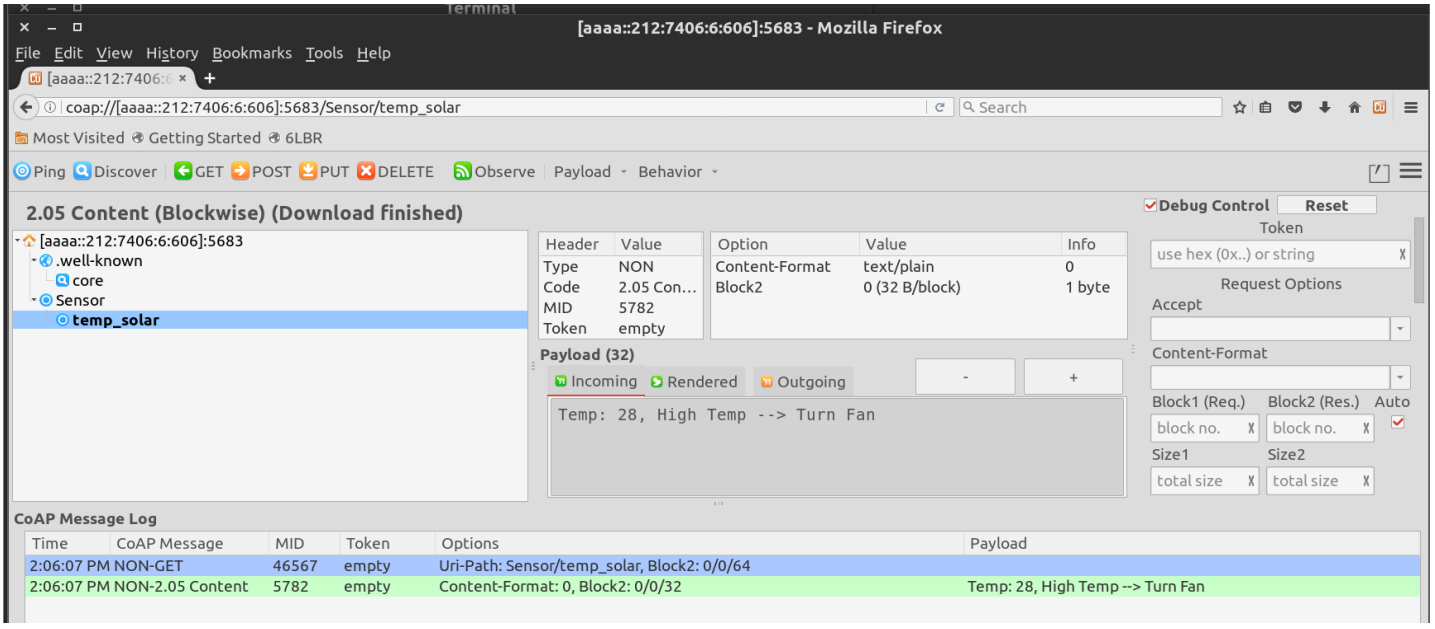

**HÌNH 5:** Chạy Cooper lấy Get thông tin trạng thái quạt và có thể điều khiển quạt với câu lệnh POST: TURNON, TURNOFF



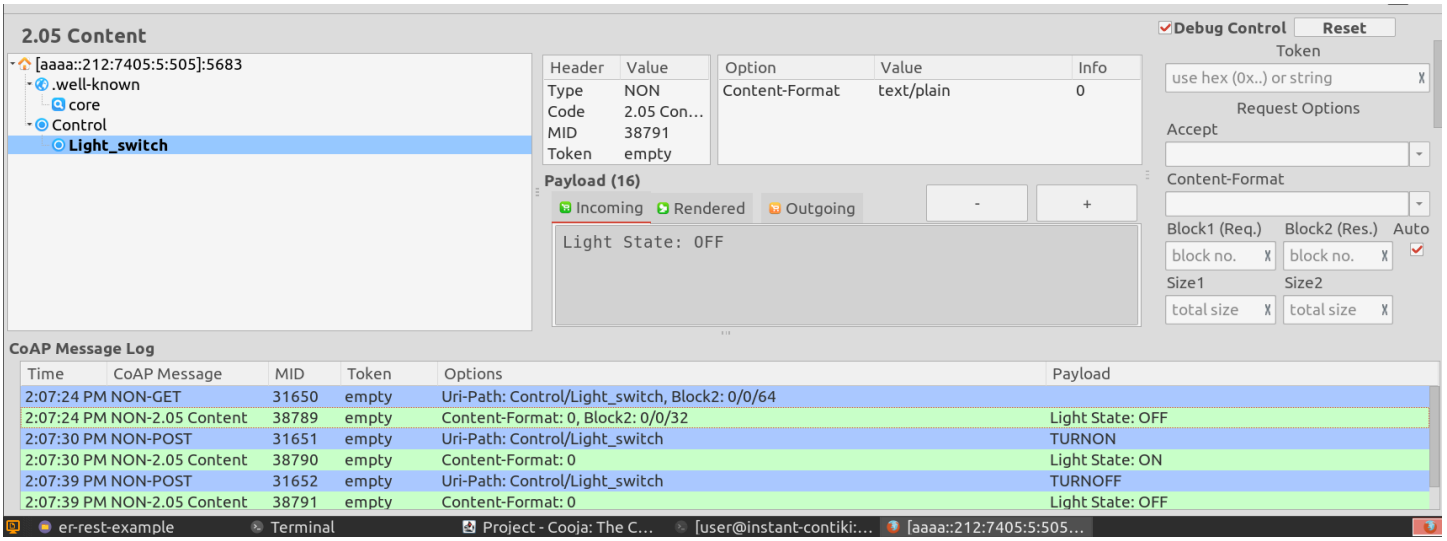
**HÌNH 6:** Chạy Cooper lấy Get thông tin trạng khí CO, và nhận thông báo hành động cần làm.



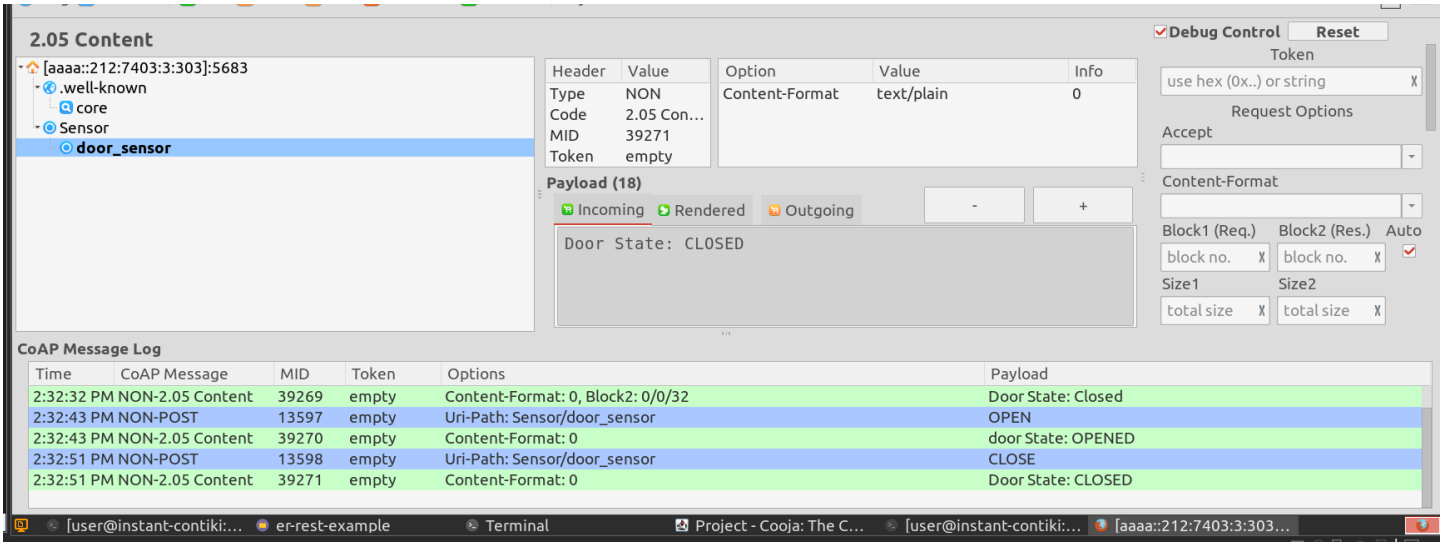

**HÌNH 7:** Chạy Cooper lấy Get thông tin trạng thái nhiệt độ và nhận thông báo hành động.



**HÌNH 8:** Chạy Cooper lấy Get thông tin trạng thái đèn và có thể điều khiển đèn với câu lệnh POST: TURNON, TURNOFF




**HÌNH 9:** Chạy Cooper lấy Get thông tin trạng thái door và có thể điều khiển cửa với câu lệnh POST: OPEN, CLOSE



**HÌNH 10:** Các thông tin về hoạt động của các thiết bị đều sẽ được in lên message của các node, kể cả sự thay đổi sau các câu lệnh POST từ Copper.

```
1:47:04.990  ID:2   GET Request received, responding with: Fan State: OFF
1:50:00.558  ID:6   Measured Temperature: 41
1:50:00.656  ID:4   Measured CO Level: 57 ppm
1:50:25.599  ID:2   POST Request received: TURNON, Fan state set to ON
1:55:25.507  ID:2   POST Request received: TURNOFF, Fan state set to OFF
2:00:00.558  ID:6   Measured Temperature: 25
2:00:00.656  ID:4   Measured CO Level: 30 ppm
2:10:00.558  ID:6   Measured Temperature: 24
2:10:00.656  ID:4   Measured CO Level: 80 ppm
2:20:00.558  ID:6   Measured Temperature: 33
2:20:00.656  ID:4   Measured CO Level: 100 ppm
2:28:29.089  ID:4   GET Request received, responding with: CO: 114 ppm, Danger! Take Action
2:30:00.558  ID:6   Measured Temperature: 32
2:30:00.656  ID:4   Measured CO Level: 45 ppm
2:30:03.400  ID:4   GET Request received, responding with: CO: 46 ppm, Safe
```

***Và toàn bộ những chức năng khác được mô tả ở các mục trên đều chạy đúng với mô phỏng (lưu ý là câu lệnh POST của door là CLOSE và OPEN). Chi tiết về code và cách chạy mô phỏng sẽ được đề cập ở file README trong link GitHub bên dưới.***

## 4. Link GitHub chứa Source Code của nhóm và những điểm cần cải thiện:

**File README sẽ chứa cách chạy mô phỏng và nhận xét về Mini Project**
**LINK:**
**https://github.com/ThinhCT03/IOTSmartHome_CoAP_ContikiOS.git**