University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 5 (Intake 2021)

EN3251 - Internet of Things

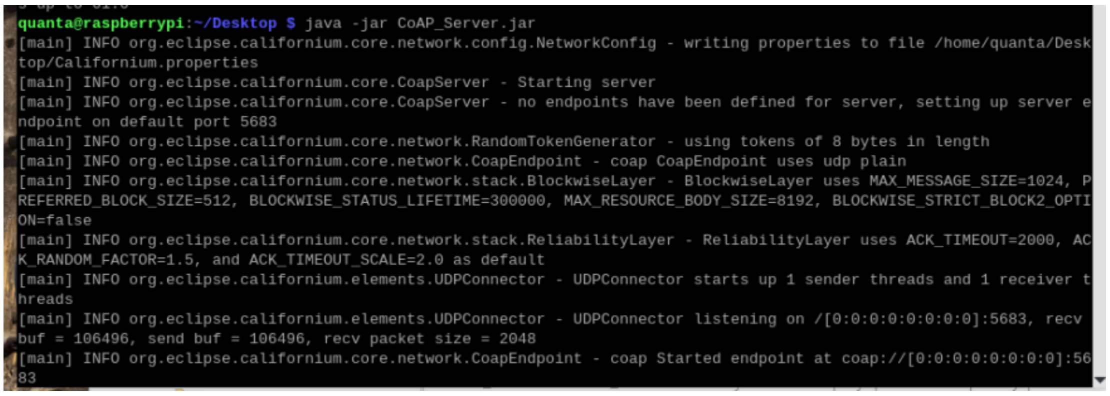
Laboratory Exercise 4: Hardware Implementation of IoT System Components

Gunawardana W.N.M.	210199D
Dilshan N.L.	210129P
Sehara G.M.M.	210583B

This report is submitted as a partial fulfillment for the moduleEN3251 - Internet of Things, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

1 Implementing a CoAP server on Raspberry Pi

Starting CoAP Server



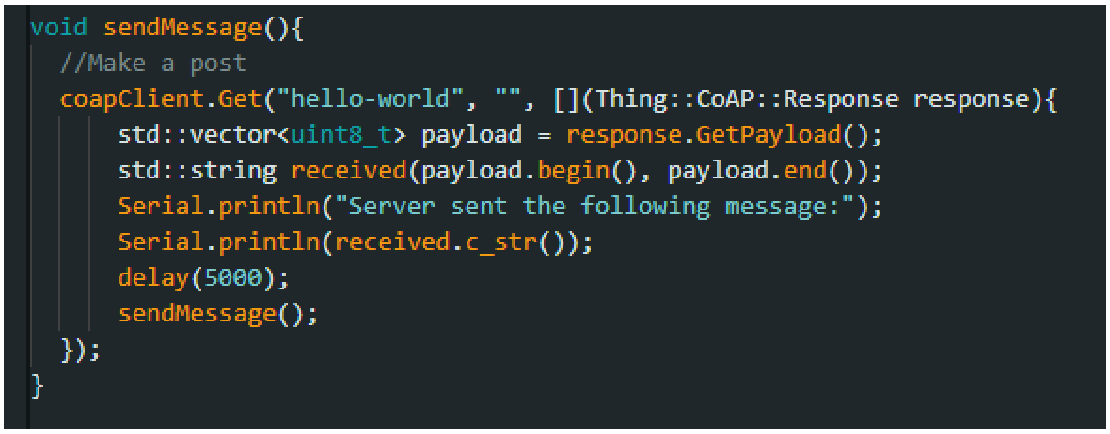
```
quanta@raspberrypi:~/Desktop $ java -jar CoAP_Server.jar
[main] INFO org.eclipse.californium.core.network.config.NetworkConfig - writing properties to file /home/quanta/Desk
top/Californium.properties
[main] INFO org.eclipse.californium.core.CoapServer - Starting server
[main] INFO org.eclipse.californium.core.CoapServer - no endpoints have been defined for server, setting up server e
ndpoint on default port 5683
[main] INFO org.eclipse.californium.core.network.RandomTokenGenerator - using tokens of 8 bytes in length
[main] INFO org.eclipse.californium.core.network.CoapEndpoint - coap CoapEndpoint uses udp plain
[main] INFO org.eclipse.californium.core.network.stack.BlockwiseLayer - BlockwiseLayer uses MAX_MESSAGE_SIZE=1024, P
REFERRED_BLOCK_SIZE=512, BLOCKWISE_STATUS_LIFETIME=300000, MAX_RESOURCE_BODY_SIZE=8192, BLOCKWISE_STRICT_BLOCK2_OPTI
ON=false
[main] INFO org.eclipse.californium.core.network.stack.ReliabilityLayer - ReliabilityLayer uses ACK_TIMEOUT=2000, AC
K_RANDOM_FACTOR=1.5, and ACK_TIMEOUT_SCALE=2.0 as default
[main] INFO org.eclipse.californium.elements.UDPConnector - UDPConnector starts up 1 sender threads and 1 receiver t
hreads
[main] INFO org.eclipse.californium.elements.UDPConnector - UDPConnector listening on /[0:0:0:0:0:0:0:0]:5683, recv
buf = 106496, send buf = 106496, recv packet size = 2048
[main] INFO org.eclipse.californium.core.network.CoapEndpoint - coap Started endpoint at coap://[0:0:0:0:0:0:0:0]:56
83
```

Figure 1: Running the CoAP server in Raspberry Pi

The terminal on the Raspberry Pi shows that the CoAP server has started successfully using the Californium framework. The server is set up with important network and configuration settings, such as message size limits and timeout settings. It is listening for CoAP requests on port 5683, the standard port for CoAP communication. The logs also show that the server is ready to handle data transfers, including options for efficient message handling. This confirms that the CoAP server is working correctly and can now receive and respond to client requests.

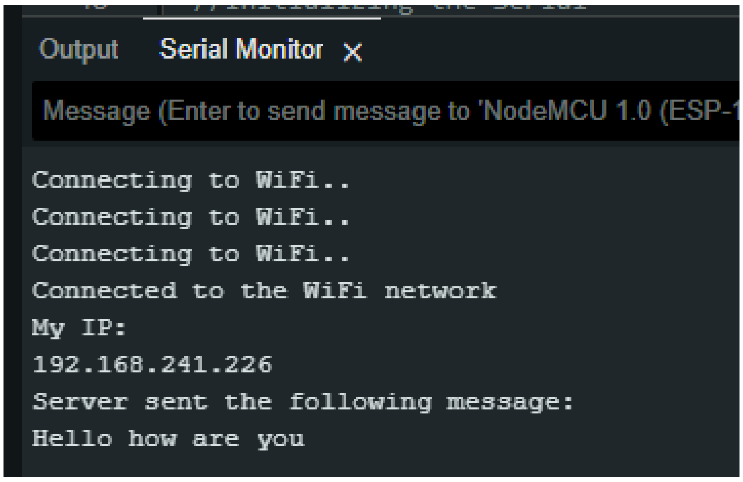
2 Implement a CoAP client on the NODE MCU

Uploading the code



```
void sendMessage(){
  //Make a post
  coapClient.Get("hello-world", "", [](Thing::CoAP::Response response){
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server sent the following message:");
      Serial.println(received.c_str());
      delay(5000);
      sendMessage();
  });
}
```

Figure 2: Send message function for the node MCU.

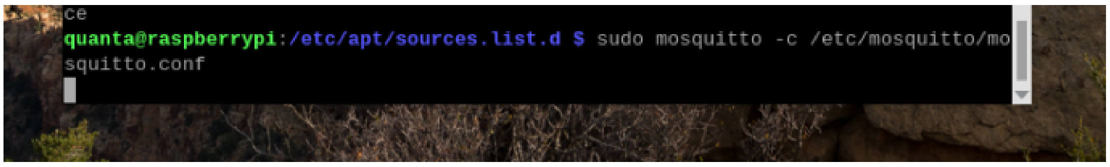


```
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
Connected to the WiFi network
My IP:
192.168.241.226
Server sent the following message:
Hello how are you
```

Figure 3: Node MCU startup

3 Implementing an MQTT Broker on Raspberry Pi

Running the mosquitto.config



```
ce
quanta@raspberrypi:/etc/apt/sources.list.d $ sudo mosquitto -c /etc/mosquitto/mo
squitto.conf
```

Figure 4: Running mosquitto broker

This shows the steps to start the MQTT broker on a Raspberry Pi. The `mosquitto` command is used to start the broker, and the configuration file `mosquitto.conf` specifies the settings for the broker.

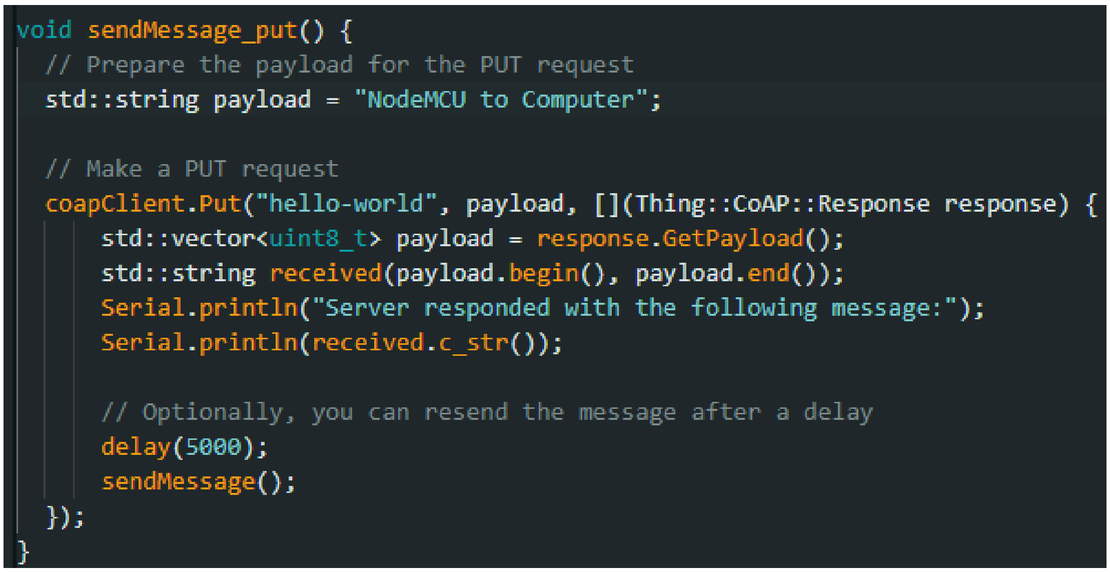
4 Implementation of GET and PUT Methods with CoAP in NODE MCU

4.1 GET Method for Receiving Data

The NODE MCU is configured to send a GET request to the CoAP server to retrieve data from the resource `"hello-world"`. Upon receiving the response, the data payload from the server is printed to the Serial Monitor, displaying the message sent by the server. This GET request is set to loop with a delay to continually check for updates from the server.

4.2 PUT Method for Sending Data

The NODE MCU is also programmed to send a PUT request with a payload message, `"NodeMCU to Computer"`, to the `"hello-world"` resource on the CoAP server. The server's response to this PUT request is printed to the Serial Monitor, allowing verification that the message has been successfully received and acknowledged by the server.

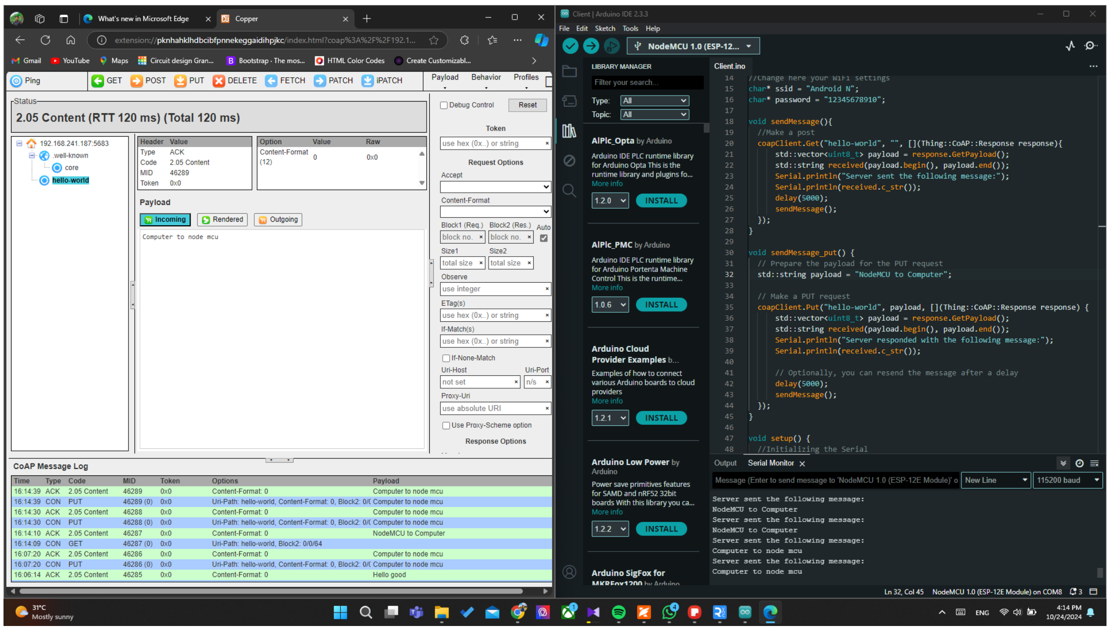


```
void sendMessage_put() {
  // Prepare the payload for the PUT request
  std::string payload = "NodeMCU to Computer";

  // Make a PUT request
  coapClient.Put("hello-world", payload, [](Thing::CoAP::Response response) {
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server responded with the following message:");
      Serial.println(received.c_str());

      // Optionally, you can resend the message after a delay
      delay(5000);
      sendMessage();
  });
}
```

Figure 5: Code for the put method

4.3 Raspberry Pi to node MCU communication (GET)

The GET request confirms the reverse communication, where the CoAP server responds to the NODE MCU with data. The retrieved messages demonstrate successful message reception and data retrieval by the NODE MCU.



2.05 Content (RTT 120 ms) (Total 120 ms)

Header	Value
Type	ACK
Code	2.05 Content
MID	46289
Token	0x0

Payload

```
Computer to node mcu
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
16:14:39	ACK	2.05 Content	46289	0x0	Content-Format: 0	Computer to node mcu
16:14:39	CON	PUT	46289 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Computer to node mcu
16:14:30	ACK	2.05 Content	46288	0x0	Content-Format: 0	Computer to node mcu
16:14:30	CON	PUT	46288 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Computer to node mcu
16:14:10	ACK	2.05 Content	46287	0x0	Content-Format: 0	NodeMCU to Computer
16:14:09	CON	GET	46287 (0)	0x0	Uri-Path: hello-world, Block2: 0/0/64	
16:07:20	ACK	2.05 Content	46286	0x0	Content-Format: 0	Computer to node mcu
16:07:20	CON	PUT	46286 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Computer to node mcu
16:06:14	ACK	2.05 Content	46285	0x0	Content-Format: 0	Hello good

Client.ino

```
//Change here your WiFi settings
char* ssid = "Android N";
char* password = "12345678910";

void sendMessage(){
  //Make a post
  coapClient.Get("hello-world", "", [](Thing::CoAP::Response response){
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server sent the following message:");
      Serial.println(received.c_str());
      delay(5000);
      sendMessage();
  });
}

void sendMessage_put() {
  // Prepare the payload for the PUT request
  std::string payload = "NodeMCU to Computer";

  // Make a PUT request
  coapClient.Put("hello-world", payload, [](Thing::CoAP::Response response) {
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server responded with the following message:");
      Serial.println(received.c_str());

      // Optionally, you can resend the message after a delay
      delay(5000);
      sendMessage();
  });
}

void setup() {
  //Initializing the Serial
```

Serial Monitor — New Line — 115200 baud

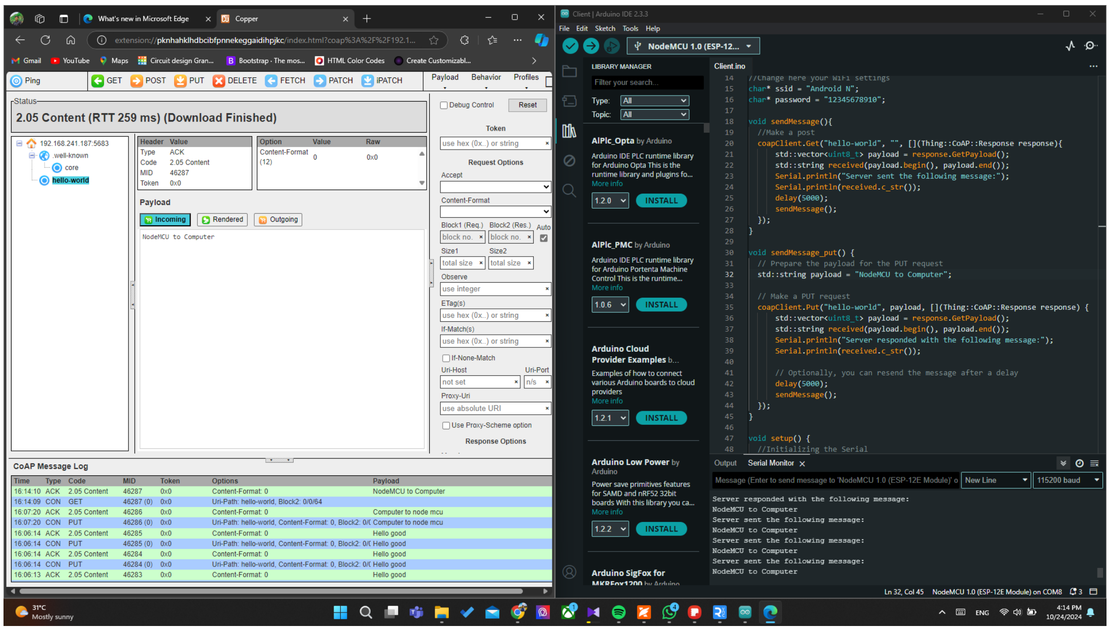
```
Server sent the following message:
NodeMCU to Computer
Server sent the following message:
NodeMCU to Computer
Server sent the following message:
Computer to node mcu
Server sent the following message:
Computer to node mcu
```

Ln 32, Col 45 NodeMCU 1.0 (ESP-12E Module) on COM8

Figure 6: Data transmission between Node MCU to Computer

4.4 Node MCU to Raspberry Pi communication (PUT)

Through the PUT request, data packets are successfully sent from the NODE MCU to the server (Raspberry Pi). The payload "`NodeMCU to Computer`" verifies that the device can send data effectively.



2.05 Content (RTT 259 ms) (Download Finished)

Header	Value
Type	ACK
Code	2.05 Content
MID	46287
Token	0x0

Payload

```
NodeMCU to Computer
```

CoAP Message Log

Time	Type	Code	MID	Token	Options	Payload
16:14:10	ACK	2.05 Content	46287	0x0	Content-Format: 0	NodeMCU to Computer
16:14:09	CON	GET	46287 (0)	0x0	Uri-Path: hello-world, Block2: 0/0/64	
16:07:20	ACK	2.05 Content	46286	0x0	Content-Format: 0	Computer to node mcu
16:07:20	CON	PUT	46286 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Computer to node mcu
16:06:14	ACK	2.05 Content	46285	0x0	Content-Format: 0	Hello good
16:06:14	CON	PUT	46285 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Hello good
16:06:14	ACK	2.05 Content	46284	0x0	Content-Format: 0	Hello good
16:06:14	CON	PUT	46284 (0)	0x0	Uri-Path: hello-world, Content-Format: 0, Block2: 0/(	Hello good
16:06:13	ACK	2.05 Content	46283	0x0	Content-Format: 0	Hello good

Client.ino

```
//Change here your WiFi settings
char* ssid = "Android N";
char* password = "12345678910";

void sendMessage(){
  //Make a post
  coapClient.Get("hello-world", "", [](Thing::CoAP::Response response){
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server sent the following message:");
      Serial.println(received.c_str());
      delay(5000);
      sendMessage();
  });
}

void sendMessage_put() {
  // Prepare the payload for the PUT request
  std::string payload = "NodeMCU to Computer";

  // Make a PUT request
  coapClient.Put("hello-world", payload, [](Thing::CoAP::Response response) {
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server responded with the following message:");
      Serial.println(received.c_str());

      // Optionally, you can resend the message after a delay
      delay(5000);
      sendMessage();
  });
}

void setup() {
  //Initializing the Serial
```

Serial Monitor — New Line — 115200 baud

```
Server responded with the following message:
NodeMCU to Computer
Server sent the following message:
NodeMCU to Computer
Server sent the following message:
NodeMCU to Computer
Server sent the following message:
NodeMCU to Computer
```

Ln 32, Col 45 NodeMCU 1.0 (ESP-12E Module) on COM8

Figure 7: Data transmission between Node MCU to Computer

5 Observations

In this lab, we used different methods to exchange data between the NODE MCU client and the Raspberry Pi CoAP server. Here, we summarize the observations for each method:

GET Method

- The GET method was used by the NODE MCU client to retrieve data from the CoAP server.
- The server responded with the requested data, confirming reliable data retrieval.

PUT Method

- The PUT method enabled the NODE MCU client to update specific resources on the server.
- The CoAP server accepted the updates, confirming successful data modification.

Network Stability and Performance

- Data transmission was stable with minimal latency.
- The Wi-Fi connection was configured correctly, ensuring reliable communication.

6 Arduino CoAP Client

```
#include <ESP8266WiFi.h>

//Include Thing.CoAP
#include "Thing.CoAP.h"

//[RECOMMENDED] Alternatively, if you are NOT using Arduino IDE you can
   include each file you need as bellow:
//#include "Thing.CoAP/Client.h"
//#include "Thing.CoAP/ESP/UDPPacketProvider.h"

//Declare our CoAP client and the packet handler
Thing::CoAP::Client coapClient;
Thing::CoAP::ESP::UDPPacketProvider udpProvider;

//Change here your WiFi settings
char* ssid = "Android␣N";
char* password = "12345678910";

void sendMessage(){
  //Make a post
  coapClient.Get("hello-world", "", [](Thing::CoAP::Response response){
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server␣sent␣the␣following␣message:");
      Serial.println(received.c_str());
      delay(5000);
      sendMessage();
  });
}

void sendMessage_put() {
  // Prepare the payload for the PUT request
  std::string payload = "NodeMCU␣to␣Computer";

  // Make a PUT request
  coapClient.Put("hello-world", payload, [](Thing::CoAP::Response response
     ) {
      std::vector<uint8_t> payload = response.GetPayload();
      std::string received(payload.begin(), payload.end());
      Serial.println("Server␣responded␣with␣the␣following␣message:");
      Serial.println(received.c_str());

      // Optionally, you can resend the message after a delay
      delay(5000);
      sendMessage();
  });
}

void setup() {
  //Initializing the Serial
  Serial.begin(115200);
  Serial.println("Initializing");

  //Try and wait until a connection to WiFi was made
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting␣to␣WiFi..");
  }
  Serial.println("Connected␣to␣the␣WiFi␣network");
  Serial.println("My␣IP:␣");
  Serial.println(WiFi.localIP());

  //Configure our server to use our packet handler (It will use UDP)
  coapClient.SetPacketProvider(udpProvider);
  IPAddress ip(192, 168, 241, 187);

  //Connect CoAP client to a server
  coapClient.Start(ip, 5683);

  //Send A Message
  sendMessage();
  sendMessage_put();
}

void loop() {
  coapClient.Process();
}
```
Listing 1: Arduino Code for CoAP Communication