Department of Electronic and Telecommunication Engineering
University of Moratuwa
Sri Lanka

EN3251 - Internet of Things

Laboratory Exercise 4
Hardware Implementation of IoT System Components

Group 9

Name	Index Number
M. P. Wickramarathne	210703V
A.A.W.L.R.Amarasinghe	210031H
A.D.T. Dabare	210089P

This report is submitted as a partial fulfillment of module EN3251
2024.11.10

Contents

1 Implementing a CoAP server on Raspberry Pi

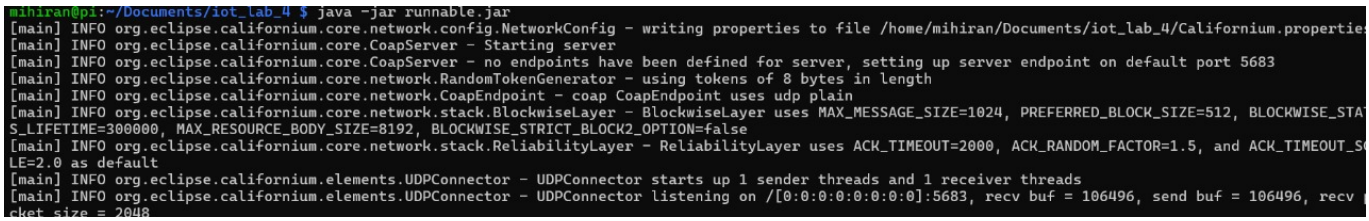

Figure 1: Running CoAP on Raspberry Pi

This is the terminal output on a Raspberry Pi during the startup of a CoAP server using the Californium library. The server is initialized with specific configurations, including network properties and token settings, and begins listening on the default CoAP port (5683) via a UDP connector. The log entries detail various settings such as message size limits, blockwise layer configurations, reliability settings (e.g., ACK timeout), and buffer sizes for sending and receiving data. This output confirms the successful initialization and readiness of the CoAP server to handle requests.

2 Implementing a CoAP client on NODE MCU

2.1 GET Request

```
15:50:12.845 -> Connecting to WiFi..
15:50:13.838 -> Connecting to WiFi..
15:50:15.577 -> Connecting to WiFi..
15:50:16.578 -> Connecting to WiFi..
15:50:17.561 -> Connecting to WiFi..
15:50:18.575 -> Connecting to WiFi..
15:50:19.588 -> Connecting to WiFi..
15:50:19.588 -> Connected to the WiFi network
15:50:19.588 -> My IP:
15:50:19.588 -> 192.168.31.72
15:50:19.666 -> Server sent the following message:
15:50:19.666 -> Mihiran Rules the World!
15:50:24.680 -> Server sent the following message:
15:50:24.680 -> Mihiran Rules the World!
```

Figure 2: GET serial monitor

Following the successful connection, the device sends a GET request to the CoAP server at the specified URI, receiving the message "Mihiran Rules the World!" in response to the message stored on the server.

```
void sendMessage(){
  //Get a Message
  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 3: Get Code

The function `sendMessage()` which was used to send GET requests is given above.

2.2 PUT Request

```
15:57:26.222 -> Connecting to WiFi..
15:57:27.218 -> Connecting to WiFi..
15:57:28.216 -> Connecting to WiFi..
15:57:29.210 -> Connecting to WiFi..
15:57:30.198 -> Connecting to WiFi..
15:57:31.204 -> Connecting to WiFi..
15:57:32.185 -> Connecting to WiFi..
15:57:32.185 -> Connected to the WiFi network
15:57:32.185 -> My IP:
15:57:32.185 -> 192.168.31.72
15:57:32.262 -> Server response:
15:57:32.262 -> Hello, this is a PUT request
```

Figure 4: PUT serial Monitor

After establishing the connection, the device sends a PUT request to the CoAP server with the message "Hello, this is a PUT request." The server's response, confirming receipt of the message, is displayed on the serial monitor, which validates the successful execution of the PUT request and communication with the CoAP server.

```
void sendPutRequest() {
    std::string payload = "Hello, this is a PUT request";

    // Send PUT request to the CoAP server
    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 response:");
        Serial.println(received.c_str());
    });
}
```

Figure 5: PUT Code

The `sendPutRequest()` function defines a payload containing the message "Hello, this is a PUT request," which is then sent to the CoAP resource at the URI "hello-world." The code specifies a callback function to process the server's response, extracting the payload, converting it to a string, and printing it to the serial monitor.

2.3 CoAP Message Log and Payload Details

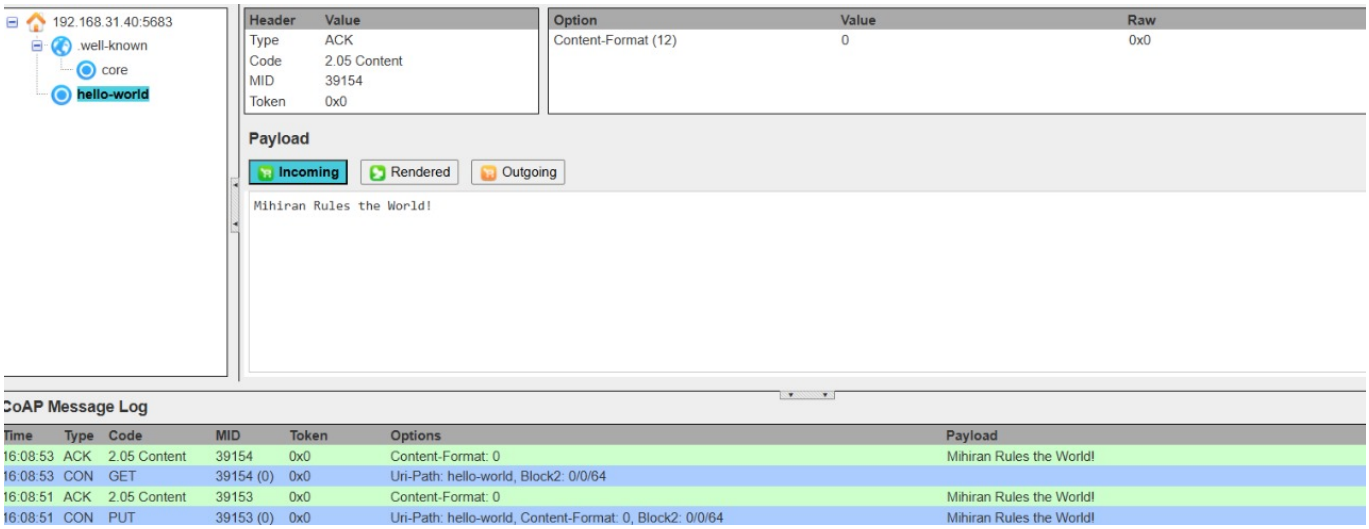

Figure 6: Put Get from Edge

This displays the CoAP message log and payload details captured from a CoAP client(in Edge Browser) interacting with a CoAP server hosted on a Raspberry Pi. We can see the Server responding to the "Mihiran Rules the world" for Get Requests.

The CoAP message log at the bottom shows a sequence of messages:

- A GET request is sent to the "hello-world" resource, receiving a `2.05 Content` response with the payload "Mihiran Rules the World!" This confirms that the server successfully processed the GET request and returned the expected content.
- A PUT request follows, also directed at the "hello-world" resource, confirming two-way interaction with the CoAP server.

The detailed payload view displays the server's response to the GET request with the message "Mihiran Rules the World!" This interface provides a clear view of incoming and outgoing messages and the content format and options associated with each CoAP message, confirming successful CoAP communication and resource manipulation.

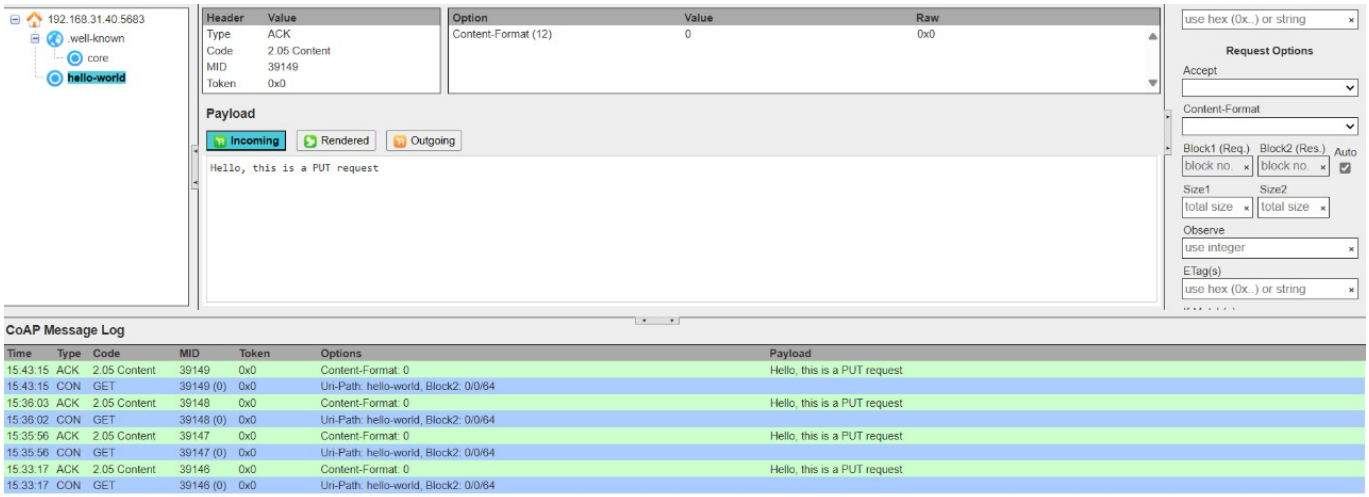

Figure 7: CoAP Message Log and Server Response

Here we can see the Response for the Get Requests after the PUT Request of the Node-MCU. We can see "Hello, This is a PUT request" as the response.

3 Implementing an MQTT Broker on Raspberry Pi

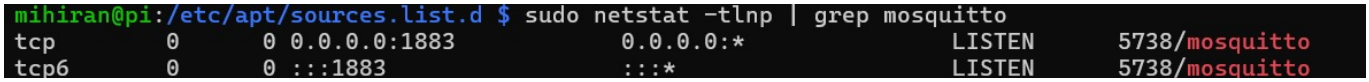

Figure 8: Mosquitto Broker

This shows the terminal output from the Raspberry Pi, verifying that the Mosquitto MQTT broker is actively listening on port 1883.

4 Mosquitto

Operating within ISP networks that employ NAT on routers followed by CGNAT, we initially attempted to use DDNS for external access but found it was not feasible due to multiple layers of NAT. As a solution, we turned to ngrok, which offers a secure tunneling service that bypasses these limitations. With ngrok, we can dynamically create a public endpoint that tunnels directly to our Mosquitto server on the Raspberry Pi, allowing controlled internet exposure

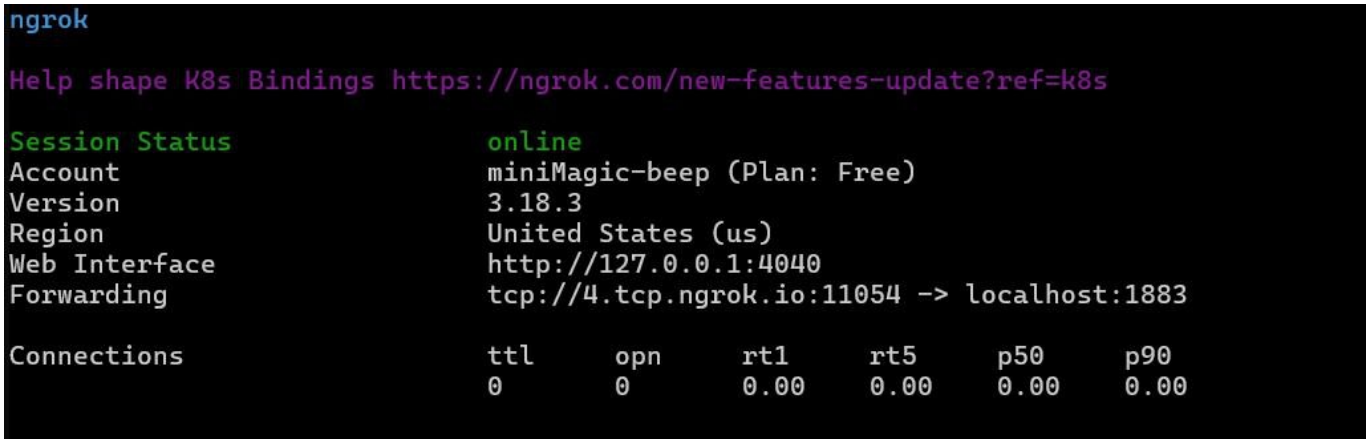

Figure 9: ngrok Session

We can see that mosquito broker is exposed to internet with the URL tcp://4.tcp.ngrok.io:11054.

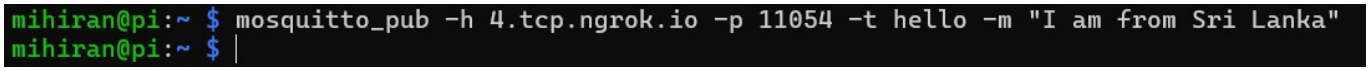

Figure 10: mosquitto pub

I have published message "I am from Sri Lanka" under hello topic.

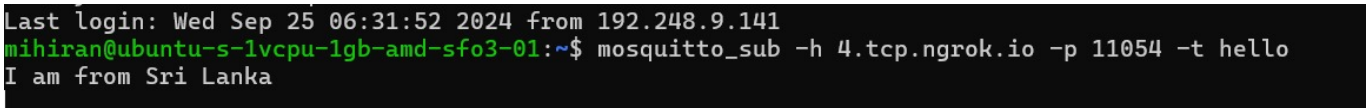

Figure 11: mosquitto sub

I have subscribed to the same topic with a VM from Digital Ocean. We can see we received the message.

Since Ngrok does not support UDP, which is required for a CoAP server, exposing the CoAP server to the internet via Ngrok is not feasible. Therefore, I am running the executable JAR on a virtual machine (VM) hosted on DigitalOcean, which provides a public IP address, allowing the CoAP server to be accessible over the internet.

```
mihiran@ubuntu-s-1vcpu-1gb-amd-sfo3-01:~/iot$ java -jar runnable.jar
[main] INFO org.eclipse.californium.core.network.config.NetworkConfig - writing properties to file /home/mihiran/iot/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 endpoint 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, PREFERRED_BLOCK_SIZE=512, BLOCKWISE_STATUS_LIFETIME=300000, MAX_RESOURCE_BODY_SIZE=8192, BLOCKWISE_STRICT_BLOCK2_OPTION=false
[main] INFO org.eclipse.californium.core.network.stack.ReliabilityLayer - ReliabilityLayer uses ACK_TIMEOUT=2000, ACK_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 threads
[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]:5683
```

Figure 12: Starting CoAP Server

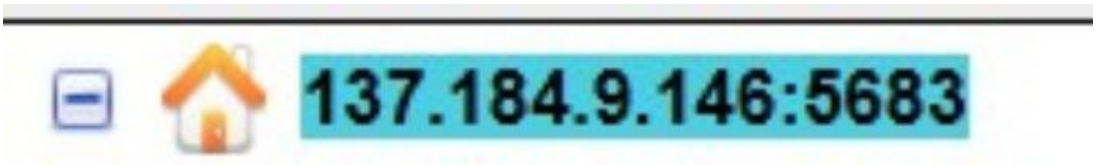

Figure 13: Address

This is the IP of CoAP Server.

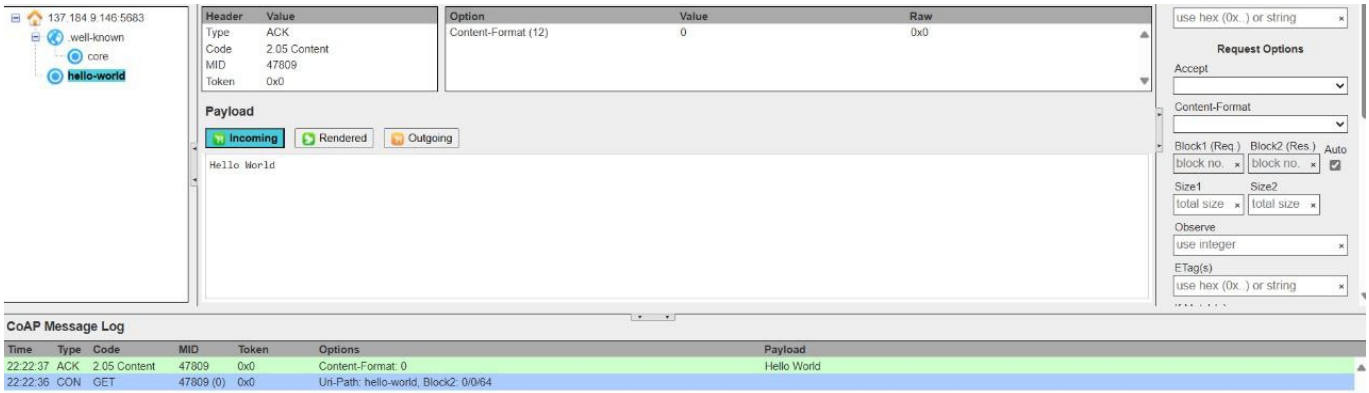

Figure 14: Client-Server Interaction

We can see that CoAP Server is accessible over the internet. Here I am sending a GET request to the server with edge browser on desktop.

5 Arduino Code for Setup, Get and PUT

```
#include <ESP8266WiFi.h>

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

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

//Change here your WiFi settings
const char* ssid = "Wi likes Fi";
const char* password = "mihiran1";

void sendMessage(){
  //Get a Message
  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 sendPutRequest() {
    std::string payload = "Hello, this is a PUT request";

    // Send PUT request to the CoAP server
    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 response:");
        Serial.println(received.c_str());
    });
}

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, 31, 40);

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


```

```
61    //Get A Message
62    sendMessage();
63
64    //Put a Message
65    sendPutRequest();
66  }
67
68  void loop() {
69    coapClient.Process();
70  }
```

Listing 1: Arduino Code for setup, Get and PUT