VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Assignment

Application based Internet of things

Mentor: Le Trong Nhan
Vu Trong Thien
Student: Van Chan Duong - 1811824

Ho Chi Minh City, June 2021

Contents

List of Figures

List of Tables

1 Introduction

In this assignment, I implement a simple system where it gathers temperature and humidity information provided by a sensor. The user also can monitor the information given by the system by using a simple mobile application or the provided UI from the IoT server.

The system consist of 4 main parts: The sensor node, the gateway, the server and the monitoring application. Each of the component and their connection is presented as follows:

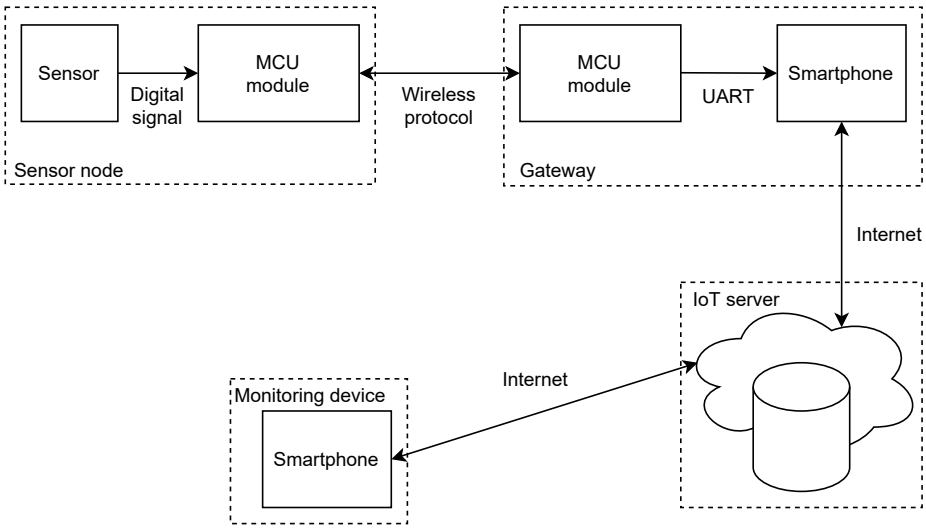

Figure 1: Overall architecture

The hardware, protocol used in the system and their implementation will be described in the upcoming sections.

2 Sensor implementation

For the sensor node, the sensor will gather information and MCU module through a dedicated GPIO pin receives the info and send it to the gateway's MCU module through *ESP-NOW* protocol (the protocol designed specifically for the ESP products).

The sensor used is the DHT11, the MCU module used to gather the information provided by the sensor is the LUA ESP8266 CP2102 Nodemcu WIFI Module.

The MCU module use the provided DHT11 interface from Adafruit. To prevent failure on the firmware, when reading the sensor's value, value availability checking is included:

```
DHT dht(DHTPIN, DHTTYPE);

dht.begin();

float humidity = dht.readHumidity();
float temp = dht.readTemperature();

if (isnan(humidity) || isnan(temp)) {
  Serial.println("Failed to read from DHT sensor!");
  return;
}
```

With the help of the ESP-NOW protocol's API, I manage to setup a wireless communication between the sensor node and the gateway. The data is encapsulated in a vendor-specific action frame and then transmitted from one Wi-Fi device to another without connection.

The data send through the MCU modules is organized to a `C-struct` as follows:

```
typedef struct struct_message {
  int id;
  float temp;
  float humidity;
} struct_message;
```

Thus the receiver must also have this struct declared to be able to extract the data sent.

To send the data, simply declare the `struct_messages` variable and issue the send function:

```
struct_message myData;

// the receiver's MAC address
uint8_t broadcastAddress[] = {0x24, 0x0A, 0xC4, 0xEE, 0xAB, 0xD4};

// Set values to send
myData.temp = temp;
myData.humidity = humidity;
myData.id = id;

// Send message via ESP-NOW
esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));
```

The protocol also supports delivery status, making it easier for me to keep track the packet sending process.

By using a simple callback function as below:

```
// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
  if (sendStatus == 0){
    Serial.println("Delivery success");
  }
  else{
    Serial.println("Delivery fail");
  }
}
```

and setup the callback on delievering message:

```
esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
esp_now_register_send_cb(OnDataSent);
```

The packet sending status can be tracked from the serial monitor as follows:

Figure 2: The status of the sensor node on the Serial monitor

The status includes the failure on getting the data from the sensor.

The detail on the setting up the sensor and ESP-NOW protocol for the sensor node is included in the script at the appendix, namely Sensor node setup.

3 Gateway implementation

The gateway consists of two part, the MCU module and a smartphone with dedicated application. The MCU module will receive the sensor's infomation from the sensor node's MCU module. Through UART communication, the mobile application will able to get the sensor's info and send it to the IoT server through *MQTT* protocol.

The MCU module chosed is the ESP32 NodeMCU LuaNode32 Wifi module

3.1 Data received process

The gateway must also have the same C-struct declared to be able to extract the data sent from the sensor node.

The setup for the gateway's MCU module is simpler than the sensor node's MCU module. Only assigning a callback function on receiving is needed after ESP-NOW initialization:

```
struct_message myData;

void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
↪  {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.printf("<");
  Serial.print(myData.id);
  Serial.print(":");
  Serial.print(myData.temp);
  Serial.print(":");
  Serial.print(myData.humidity);
  Serial.print(">");
}
```

Setup the callback on receiving message:

```
esp_now_register_recv_cb(OnDataRecv);
```

The detailed ESP-NOW setup for the gateway's MCU module is included in the script at the appendix, namely Gateway MCU setup.

3.2 UART received data

The data from the MCU module to the mobile application was formatted as below:

```
-------------------------------------------------------
| '<' | ID | ':' | Temperature | ':' | Humidity | '>' |
-------------------------------------------------------
Example data frame:
    <2951412:27.50:78.00>
```

The data frame includes the character '<' and '>' to indicates the starting and ending point of the data frame. The ':' character is added between the information for parsing purpose.

To setup the UART communication for the mobile application, the usb-serial-for-android library is used.

The implementation on the receiving function is as follows:

```
String buffer = "";
String dataStr = "";
Float temp = new Float(0);
Float humidity = new Float(0);
String id = "";
Boolean begin = false;
Boolean finish = false;

private void receive(byte[] data) {
    if (!begin) {
        buffer = new String(data);
        int beginIdx = buffer.indexOf("<");
        if (beginIdx != -1) {
            if (beginIdx + 1 < data.length) {
                dataStr = dataStr + buffer.substring(beginIdx + 2);
            }
            begin = true;
        }
    }
    if (!finish) {
        buffer = new String(data);
        int endIdx = buffer.indexOf(">");
        if (endIdx == -1) {
            dataStr = dataStr + buffer;
        }
        else {
            dataStr = dataStr + buffer.substring(0, endIdx);
```

```
            finish = true;
        }
    }
    if (begin && finish) {
        String[] split = dataStr.split(":");
        String display = "";
        Date date = Calendar.getInstance().getTime();
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd
        ↪  hh:mm:ss");
        String strDate = dateFormat.format(date);
        id = split[0].substring(1);
        temp = Float.parseFloat(split[1]);
        humidity = Float.parseFloat(split[2]);
        display = id + ": ID" + "\n"
                + temp + " :TEMP" + "\n"
                + humidity + " :HUMID" + "\n"
                + strDate + "\n\n";
        // Send data to MQTT
        //===
        txtData.setText(display);
        dataStr = "";
        begin = false;
        finish = false;
    }
}
```

3.3 Data uploading process

After the sensory data is received by the gateway, the data will immediately be sent to the MQTT server through a provided function. The calling of the function is located at the commented part in the `void receive(byte[] data)` function mentioned above.

The detail of the send to MQTT function is as follows:

```
private void sendDataToMQTT(String topic, String data) {
    MqttMessage msg = new MqttMessage();
    msg.setId(1234);            // each message has its own id
    msg.setQos(0);              // Quality of service -> 0 is fastest
    msg.setRetained(true);

    byte[] b = data.getBytes(Charset.forName("UTF-8"));
```

```
    msg.setPayload(b);

    try {
        mqttHelper.mqttAndroidClient.publish(topic, msg);
    } catch (MqttException e) {
    }
}
```

The continuation of the `void receive(byte[] data)`:

```
...
// Send data to MQTT
if (enabled) {
    sendDataToMQTT("shinyo_dc/feeds/iot-project.iot-project-humidity",
    ↪  humidity.toString());
    sendDataToMQTT("shinyo_dc/feeds/iot-project.iot-project-temp",
    ↪  temp.toString());
}
//===
...
```

The `enabled` variable is to indicate if the user wants to send the data onto the MQTT server or not.

The setup for the Gateway application on receiving the data through UART communication and on sending data to MQTT server is provided as an Android Studio project, namely *UART_MQTT*, together with this report.

4 Server configuration

The IoT server stored the information and allows the user to track the information using the dedicated UI.

The Adafruit IO IoT server was used for the purpose. The Dashboard function allow me to create functional UI based on the information on the data feeds.

The link to the dashboard can be found here:

https://io.adafruit.com/shinyo_dc/dashboards/humid-temp

It is recommended to have an Adafruit IO account to see the streaming data on the dashboard when the whole system is working.

A brief overview of what the dashboard looks like:

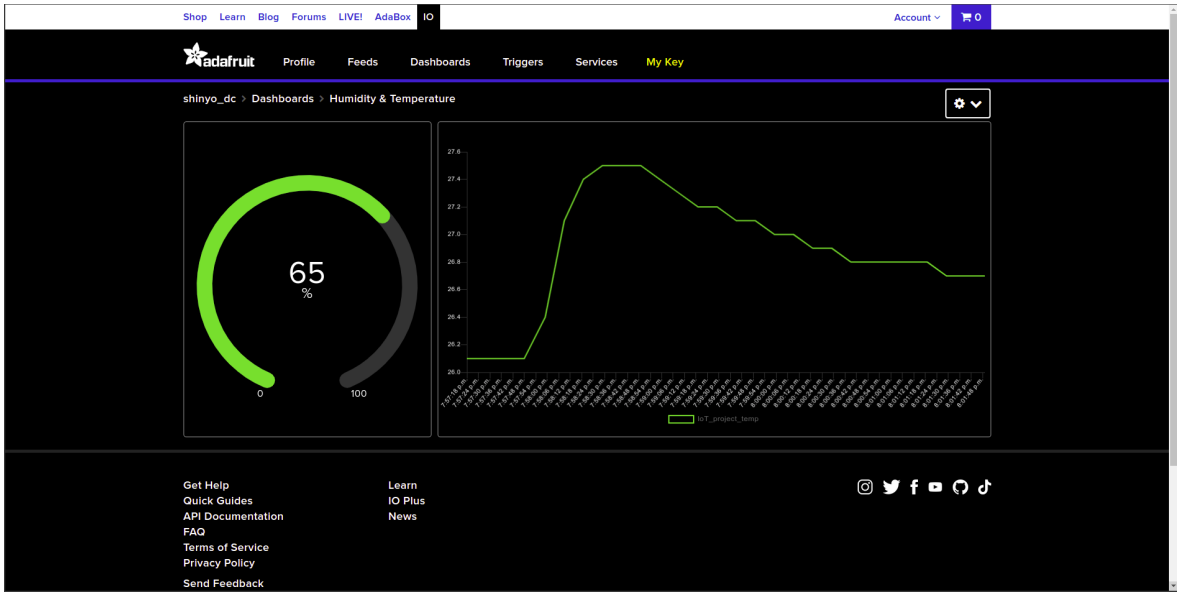

Figure 3: The server's dashboard

The gauge is for the humidity value and the Chart is for the temperature value.

5 Monitoring application

The monitoring device will also be able to keep track of the information by fetching the data from the IoT server by the internet through the provided server's HTTP API. The HTTP request is made using the provided library `volley`.

The application capable of viewing the latest 20 points of data for the temperature value, and getting the lastest point of data for the humidity value.

The temperature values is presented as a line chart and the humidity values is presented as a pie chart, indicating how many % moisture there is.

Both of the data types are fetched using the HTTP API provided by the Adafruit IO server. The fetching cycle is 2 seconds.

The implementation for functions (fetching data together with plotting the data) can be found at the appendix, namely Fetch Temperature and Fetch Humidity.

The detailed setup for the monitoring application to fetch the data through HTTP request and present the data graphically through graphs is provided as an Android Studio project, namely *DataFetcher*, together with this report.

Here is a brief overview of what the monitoring application looks like:

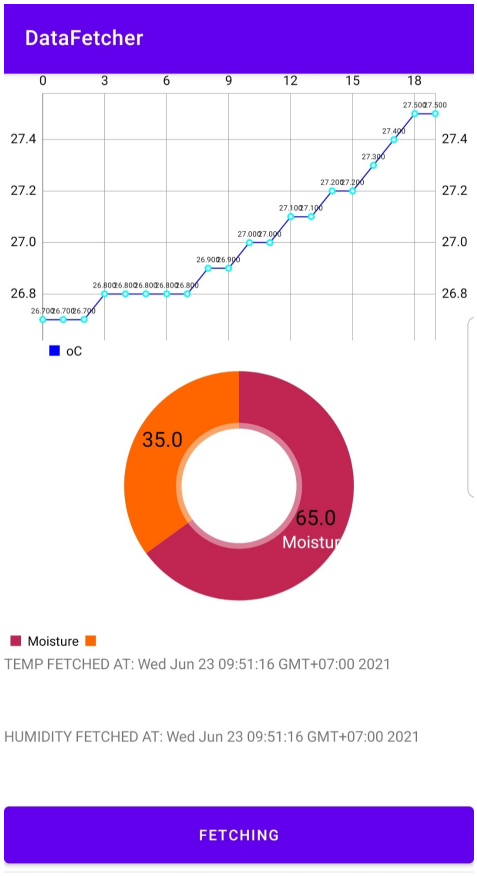

Figure 4: Monitoring application interface

6 Conclusion

In this assignment, a simple 4 part IoT system is implemented.

I got a chance to work with ESP-NOW protocol, implement it in a simple way to send and receive data through wireless communication.

I learnt how to use the provided library to program simple application for monitoring purpose and sending, receiving data. I also learnt how the UART communication work and found a solution (yet not really refined) to extract the serial data sent from the MCU module.

Also I have the opportunity to learn how to use the provided API from the server to fetch the data, working on the library to make HTTP request and graphically present the data for demonstrating purpose.

The experience on working on this assignment is new and exiciting for me. Although the system is somehow simple, I am proud of the result and the knowledge I learnt are meaningful to me in my study journey.

A demo video is made for demonstration purpose. The demo video can be viewed here: https://youtu.be/MITGLIiJNmw

Bibliography

Adafruit (2021a). *Adafruit IO HTTP API - Get Feed Data*. URL: https://io.adafruit.com/api/docs/#get-feed-data (visited on June 22, 2021).

– (2021b). *Using a DHTxx Sensor*. URL: https://learn.adafruit.com/dht/using-a-dhtxx-sensor (visited on June 22, 2021).

Espressif (2021). *ESP-NOW*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html (visited on June 22, 2021).

RandomNerdTutorials (2021a). *Getting Started with ESP-NOW (ESP32 with Arduino IDE)*. URL: https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/ (visited on June 22, 2021).

– (2021b). *Getting Started with ESP-NOW (ESP8266 NodeMCU with Arduino IDE)*. URL: https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/ (visited on June 22, 2021).

Trong Nhan, Le (2021a). *Android Studio Manual*.

– (2021b). *Mở kết nối UART*.

– (2021c). *MQTT Tutorial*.

Appendix

A Sensor node setup

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <espnow.h>
#include <DHT.h>

#define SLEEP_SECS 30
#define DHTPIN 4 // what digital pin we're connected to
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);
// REPLACE WITH RECEIVER MAC Address
uint8_t broadcastAddress[] = {0x24, 0x0A, 0xC4, 0xEE, 0xAB, 0xD4};

// Structure example to send data
// Must match the receiver structure
typedef struct struct_message {
  int id;
  float temp;
  float humidity;
} struct_message;

// Create a struct_message called myData
struct_message myData;

unsigned long lastTime = 0;
unsigned long timerDelay = 10000;  // send readings timer

void goToSleep() {
  int sleepSecs = SLEEP_SECS;
  Serial.printf("Up for %li ms, going to sleep for %i secs...\n", millis(),
  ↪  sleepSecs);
  ESP.deepSleep(sleepSecs * 1000000, RF_NO_CAL); // sleep for 30 seconds
}

// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
```

```
  if (sendStatus == 0){
    Serial.println("Delivery success");
  }
  else{
    Serial.println("Delivery fail");
  }
}

void setup() {
  // Init Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

  // init Sensor
  dht.begin();

  // Init ESP-NOW
  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    goToSleep();
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);

}

void loop() {
  if ((millis() - lastTime) > timerDelay) {

    // Read value from sensor:
    float humidity = dht.readHumidity();
    float temp = dht.readTemperature();
```

```
    if (isnan(humidity) || isnan(temp)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

    int id = ESP.getChipId();

    // Set values to send
    myData.temp = temp;
    myData.humidity = humidity;
    myData.id = id;

    Serial.printf("Getting temp = %f\n", myData.temp);
    Serial.printf("Getting humidity = %f\n", myData.humidity);
    Serial.printf("Getting id = %lu\n", myData.id);

    // Send message via ESP-NOW
    esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));

    lastTime = millis();
  }
}
```

B Gateway MCU setup

```
#include <Arduino.h>
#include <esp_now.h>
#include <WiFi.h>

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
  int id;
  float temp;
  float humidity;
} struct_message;

// Create a struct_message called myData
struct_message myData;
```

```
// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
 ↪  {
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.printf("<");
  Serial.print(myData.id);
  Serial.print(":");
  Serial.print(myData.temp);
  Serial.print(":");
  Serial.print(myData.humidity);
  Serial.print(">");
}

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    ESP.restart();
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(OnDataRecv);
}

void loop() {

}
```

C Fetch Temperature

```
private void getTemperature(String userName, String feedKey, TextView view,
↪  LineChart chart){
    // Get current 20 data points
    // id, value, feed_id, feed_key, created_at, created_epoch, expiration
    String apiURL = "https://io.adafruit.com/api/v2/" + userName +
    ↪  "/feeds/" + feedKey + "/data?limit=20";

    JsonArrayRequest request = new JsonArrayRequest(apiURL, new
    ↪  Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            ArrayList<Entry> dataSet = new ArrayList<Entry>();
            for (int i = 0; i < response.length(); i++) {
                try {
                    JSONObject entry = response.getJSONObject(i);
                    String value = entry.getString("value");
                    dataSet.add(new Entry(i, Float.parseFloat(value)));
                } catch (JSONException e) {
                    e.printStackTrace();
                    view.setText("PARSING ERROR");
                }
            }
            java.util.Date date=new java.util.Date();
            view.setText("TEMP FETCHED AT: " + date.toString());
            LineDataSet lineDataSet = new LineDataSet(dataSet, "oC");

            lineDataSet.setColor(Color.BLUE);
            lineDataSet.setCircleColor(Color.CYAN);
            lineDataSet.setDrawCircles(true);
            lineDataSet.setDrawCircleHole(true);

            ArrayList<ILineDataSet> iLineDataSets = new ArrayList<>();
            iLineDataSets.add(lineDataSet);
            LineData lineData = new LineData(iLineDataSets);

            chart.setData(lineData);
            chart.getDescription().setEnabled(false);
            chart.invalidate();
        }
```

```
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            error.printStackTrace();
            view.setText("PARSING ERROR");
        }
    });

    mQueue.add(request);
}
```

D Fetch Humidity

```
private void getHumidity(String userName, String feedKey, TextView view,
↪  PieChart chart){
    // Get lasted data points
    // id, value, feed_id, feed_key, created_at, created_epoch, expiration
    String apiURL = "https://io.adafruit.com/api/v2/" + userName +
    ↪  "/feeds/" + feedKey + "/data?limit=1";

    JsonArrayRequest request = new JsonArrayRequest(apiURL, new
    ↪  Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            ArrayList<PieEntry> dataSet = new ArrayList<PieEntry>();
            for (int i = 0; i < response.length(); i++) {
                try {
                    JSONObject entry = response.getJSONObject(i);
                    String value = entry.getString("value");
                    dataSet.add(new PieEntry(Float.parseFloat(value),
                    ↪  "Moisture"));
                    dataSet.add(new PieEntry(100-Float.parseFloat(value),
                    ↪  ""));
                } catch (JSONException e) {
                    e.printStackTrace();
                    view.setText("PARSING ERROR");
                }
            }
            java.util.Date date=new java.util.Date();
            view.setText("HUMIDITY FETCHED AT: " + date.toString());
```

```
            PieDataSet pieDataSet = new PieDataSet(dataSet, "");
            pieDataSet.setColors(ColorTemplate.COLORFUL_COLORS);
            pieDataSet.setValueTextColor(Color.BLACK);
            pieDataSet.setValueTextSize(16f);

            PieData pieData = new PieData(pieDataSet);

            chart.setData(pieData);
            chart.getDescription().setEnabled(false);
            chart.animate();
            chart.invalidate();
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            error.printStackTrace();
            view.setText("PARSING ERROR");
        }
    });

    mQueue.add(request);
}
```