CO321: Embedded Systems Project – Report

IOT AIR, SOUND AND TEMPERATURE MONITORING SYSTEM

GROUP 16
W.M.K.D.ABEYSINGHE (E/13/004)
K.L.D.DESHAPRIYA (E/13/061)
FERNANDO W.K.M.A. (E/13/101)
W.A.M.N.WEERASOORIYA (E/13/396)

INTRODUCTION

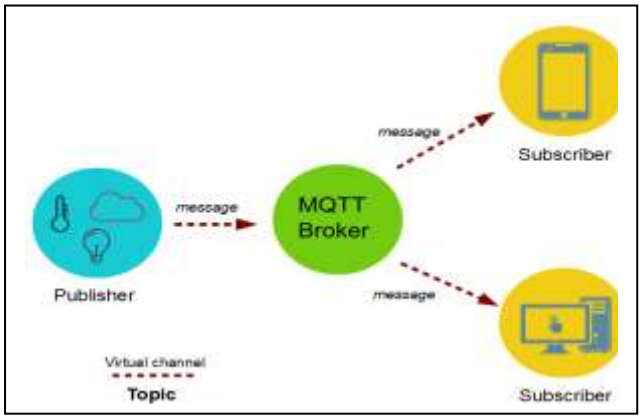
The project is an air quality, sound, temperature and humidity monitoring system. Multiple sensors are embedded into one device to get the above readings. Those type of devices can be located at several locations and all the devices are connected with a main server. Readings obtained from each and every device can be send to the main server. When the standard limits of the above three values are exceeded, the entities who are maintaining the main server, are alerted. Anyone who are intending to monitor the above pollution parameters, can maintain a main server. In our system we are mainly targeting a factory.

1. DESCRIPTION OF CLIENT AND SERVER NETWORK CODE WITH JUSTIFICATION FOR CHOICE TECHNOLOGIES

We used a NodeMCU with ESP8266 inboard, to push data from the devices to the main server. ESP8266 with NodeMCU Firmware can be configured as an Access Point, Wi-Fi Client (Host / Station) or both as Client and AP at the same time. In this implementation we used it as a Wi-Fi client. It has capability to work with 802.11b, 802.11b and 802.11n networks.

TECHNOLOGY USED AND JUSTIFICATIONS

The main technology used in the implementation is **MQTT** (Message Queuing Telemetry Transport) which is a lightweight messaging protocol that provides resource-constrained network clients with a simple way to distribute telemetry information. The protocol, which uses a publish/subscribe communication pattern, is used for machine-to-machine (M2M) communication and plays an important role in the Internet of Things (IoT).



MQTT allows devices to send (publish) information about a given topic to a server that functions as an MQTT message broker. The broker then pushes the information out to those clients that have previously subscribed to the client's topic.

Figure 1: publish/subscribe

- **Difference from other protocols**

The difference to HTTP is that a client doesn't have to pull the information it needs, but the broker pushes the information to the client, in the case there is something new. Therefore each MQTT client has a permanently open TCP connection to the broker. If this connection is interrupted by any

circumstances, the MQTT broker can buffer all messages and send them to the client when it is back online.

- **Error Handling**
  If there is an error in an MQTT connector step, the Error Handler is invoked and handles the error. The first error in the step causes the termination of the step execution. If the step has multiple methods, and an error occurs in one of these methods, all subsequent methods are not executed.

There are some free MQTT brokers available in the internet such as cloudMQTT. We have used the broker which is hosted by "**Adafruit.io"**. We can publish data from the devices to Adafrui.io.
To get started on Adafruit.io initially we have to connect to Adafruit IO's MQTT server (a.k.a broker) and we are connecting via the Internet**.**

- **Quality of Service (QoS) for MQTT**

Quality of service (QoS) levels determine how each MQTT message is delivered and must be specified for every message sent through MQTT. It is important to choose the proper QoS value for every message, because this value determines how the client and the server communicate to deliver the message

MQTT CLIENT

This is the program that establishes connection with the server. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time).

```
#include <math.h>
#include <dht.h>
#include <ESP8266WiFi.h>
#include "MQ135.h"
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
```

Figure 2: Including relevant libraries

"Adafruit_MQTT.h" and "Adafruit_MQTT_Client.h" libraries have to be included along with the other relevant libraries.

```
#define WLAN_SSID       "AndroidG2"
#define WLAN_PASS       "apapapap"


#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "iotair"
#define AIO_KEY         "58d1d1b9c885427193377785d0870d534"
```

Figure 3: Wi-Fi authentication

After some pin definitions and other objects required for the transport layer such as Wi-Fi credentials, Cellular APN details, etc.

```
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

Adafruit_MQTT_Publish photocell = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/photocell");
Adafruit_MQTT_Publish temperaturem = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/temperature");
Adafruit_MQTT_Publish Humidity = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Humidity");
Adafruit_MQTT_Publish smokem = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Smoke");
Adafruit_MQTT_Publish LPGm = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/LPG");
Adafruit_MQTT_Publish COm = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/CO");
Adafruit_MQTT_Publish COx = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/CO2");
Adafruit_MQTT_Subscribe onoffbutton = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/onoff");

void MQTT_connect();
```

Figure 4: Publishing and Subscribing

Publishing

Adafruit IO's MQTT API exposes feed data using special topics. We can publish a new value for a feed to its topic, or we can subscribe to a feed's topic to be notified when the feed has a new value. Any one of the following topic forms is valid for a feed:

- (username)/feeds/(feed name or key)
- (username)/f/(feed name or key)

We published to a topic starting by creating the name.For an example the name of the “photo cell” topic is “AIO_USERNAME/feeds/photocell”**.** That way it doesn’t get confused with anybody else's photocell feed. Only we have access to publish to the feeds under our username. Then we can create the Adafruit_MQTT_Publish object, which we also call photocell.

Subscribing

Subscribing to a topic is similar to publishing. Create a string name to the feed name and create an MQTT subscription object. For an instance “onoffbutton” object in the above figure.

Sketch Setup

We have connected to the internet through a hotspot.

```
void setup() {

  Serial.begin(9600);
  delay(200);//Wait rest of 1000ms recommended delay before

  Serial.println(F("IOT Enviornmental Pollution Monitoring System"));

  // Connect to WiFi access point.
  Serial.println(); Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);

  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.println("WiFi connected");
  Serial.println("IP address: "); Serial.println(WiFi.localIP());

  // Setup MQTT subscription for onoff feed.
  mqtt.subscribe(&onoffbutton);
```

Figure 5: Sketch Setup

Subscribe to the topics

```
// Setup MQTT subscription for onoff feed.
mqtt.subscribe(&onoffbutton);
```

Figure 6: Subscribe to the topics

Check Connection

```
void loop() {

 MQTT_connect();
```

Figure 7: Ensuring the connection to the MQTT server

There is a helper program called MQTT_connect () to make sure the connection to the MQTT server.

Following is the function called above. It checks to make sure that MQTT is connected, and if not, it reconnects.

```
void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(100);           // basically die and wait for WDT to reset me
// wait 5 seconds
       retries--;
       if (retries == 0) {
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
}
```

Figure 8: MQTT connect function

Wait for subscription messages

```
// this is our 'wait for incoming subscription packets' busy subloop
// try to spend your time here

Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(5000))) {
  if (subscription == &onoffbutton) {
    Serial.print(F("Got: "));
    Serial.println((char *)onoffbutton.lastread);
  }
}
```

Figure 9: Wait for subscription messages

After the connection check, we have to wait for subscriptions to come in. We started by creating a pointer to an Adafruit_MQTT_Subscribe object.
We have used this to determine which subscription was received. In this case we only have one subscription, so we don't *really* need it.
Then we have to wait for a subscription message.
mqtt.readSubscription (5000) will sit and listen for up to 5000ms for a message. It will either get a message before the timeout, and reply with a pointer to the subscription **or** it will timeout and return **0**. In this case, it will wait up to 5 seconds for a subscription message.

If the reader times out, the while loop will fail. However, we do get a valid non-zero return. Then we compare what subscription we got to our known subs

For example, here we have compared to the **onoffbutton** feed sub. If they match, we can read the last message. The message is in feedobject.lastread. We have to cast this since MQTT is completely agnostic about what the message data is.

Publish Data

```
    dBvalue=(sum/20)+10;
    sum=0;
Serial.print(F("\nSending dBValue val "));
Serial.print(dBvalue);
Serial.print("...");
if (! photocell.publish(dBvalue)) {
  Serial.println(F("Failed"));
} else {
  Serial.println(F("OK!"));
}
```

Figure 10: Publish Data

If we have got any subscriptions, we should listen for them in the large bulk of the time 'available' in the loop.

Once we're done listening, we can send some data. Publication is much easier than subscribing. Publish can be done by just calling the **publish** function of the feed object. We can send ints, floats, strings, etc.

MQTT SEVER

- We are connecting to Adafruit IO's MQTT server (a.k.a broker) via the internet. We can log in to the Adafruit account by using account username and a key.
- There is a term called “feed” which is basically a set of data that we can read or write from like a sequential file. In MQTT historical data is not accessible (REST does support it) but we can add data and we can receive the latest added data.
- Initially we have to create two feeds;
    1. **photocell** - this feed will store light data from your device to adafruit.io
    2. **onoff** - this feed will act as an on/off switch, sending data *to* your device from adafruit.io

- Then to create a dashboard with a **gauge** connected to **photocell**. We used a thin type gauge with min value 0 and max value 1024 (this could store a 10 bit value)
- After that created a new block and block is added to the dashboard
- Finally another block for on-off toggle switch.

2. DESCRIPTION OF APPLICATION LAYER PROTOCOL STRUCTURE, INCLUDING EXAMPLE (APPLICATION LAYER) TRACES OF CLIENT-SERVER COMMUNICATION

- Collecting data from sensors
- push all the data to the main server using MQTT protocol
- We don't use HTTP or any other application layer protocol
- MQTT is a Pub/Sub centralized-broker protocol that is usually implemented over TCP, but that specification does not force the underlying protocol to be TCP, but is the most used one.
- Web socket implementations are also available
- TCP connection is made when we send the MQTT Connect message, which is the first packet we need to send regardless of what kind of client we are.
- MQTT has ways of detecting if this connection has broken and act in consequence. Sessions can be resumed after disconnections, so closing the TCP/MQTT connection does not mean that the session will be lost.
- The TCP handshake is sent before the MQTT connect message. Most of the libraries will handle this function, but if we are trying to create our own library, then we obviously need to establish a connection before sending the Connect message.

TRACES OF CLIENT-SERVER COMMUNICATION

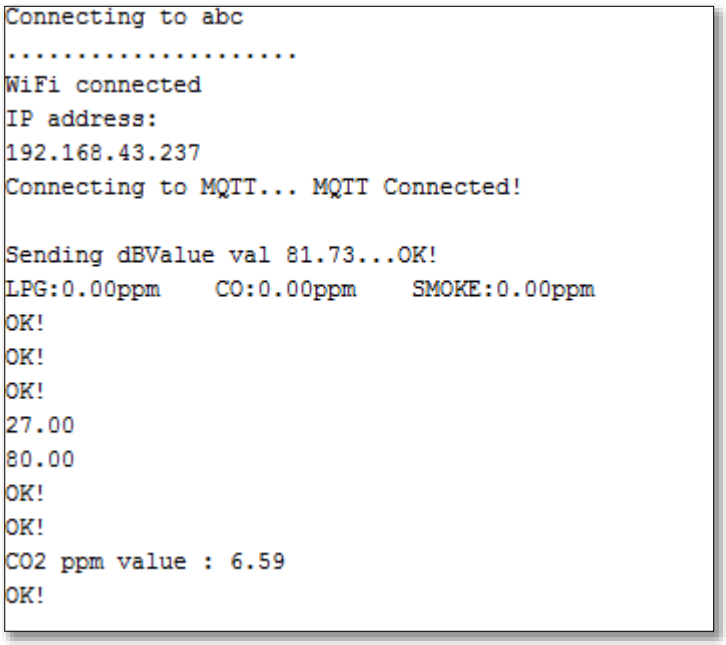


Figure 11: connecting to the server

Figure 12: Values displayed on the server

REFERENCES

- https://www.ibm.com/developerworks/community/blogs/5things/entry/5_things_to_know_about_mqtt_the_protocol_for_internet_of_things?lang=en
- http://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html
- https://learn.adafruit.com/adafruit-io/overview
- https://www.youtube.com/watch?v=VjpONmC2tac