

Laboratory 5: MQTT

Introduction

This laboratory aims to equip you with knowledge on how to:

- Use **MQTT** to **subscribe** to feeds. This will allow for the Aston IoT application trainer to receive data from another source.
- Use **MQTT** such that the Aston IoT application trainer to **publish** information and visualise this data to the end user.

Equipment and resources

Hardware:

- ESP8266 Aston IoT hardware platform.
- USB A to micro B cable.
- A smart phone to act as a wireless hotspot.

Software & documentation:

- Arduino Integrated Development Environment (IDE).



A reminder on saving and file name convention

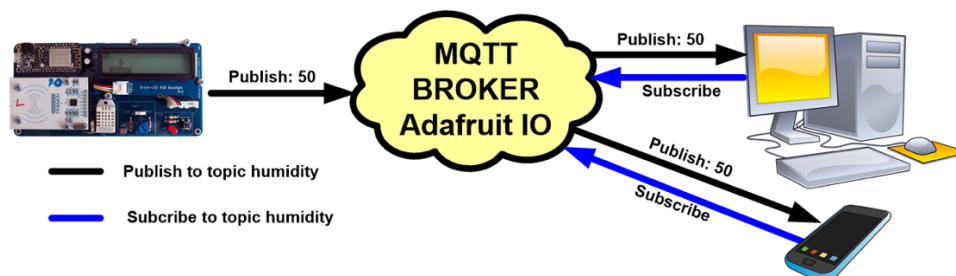
You should keep a copy of every task that you complete on your **student network drive (H:)**, in the **folder H:\EE3IOT\labs**. Sketches should be saved in the format of **surnameinitialLTX**, where **L** is the lab number and **X** is the task number.

For example, if Fred Smith has completed task 1 of Lab 1, the sketch should be saved as **smithfL1T1**.

Task 0: An introduction to MQTT

MQTT (MQ Telemetry Transport) is a publish/subscribe protocol that is designed for memory constrained devices, such as the ESP8266 microcontroller. The protocol also aims to minimise bandwidth utilisation (i.e. the number of bytes transmitted per unit time such that power consumption of the microcontroller can be minimised) and to ensure reliable communications in unreliable networks. As of such, MQTT is a protocol which builds upon TCP/IP.


Unlike other communication protocols which typically use IP addresses and port numbers to route messages between various devices, MQTT utilises **topics or feeds** to route information between various devices. Referring to below:



- Devices can **subscribe** to a topic/feed and the MQTT broker will automatically push all new data to all subscribers of the topic/feed as needed.
- Any device can **publish** data to a particular feed/topic. The broker handles receiving this data and storing it.

- Most MQTT APIs are designed for short messages. The API we are using from Adafruit is limited to 20 byte messages or 20 ASCII characters.

Task 1: Controlling a digital output (LED) via **subscription**

1. Create a WiFi hotspot on your mobile phone. Bridged connections are unfortunately not allowed at the University as the network administration no longer knows whom is using a particular connection.
2. Visit <https://io.adafruit.com> and create an account via clicking on “**Get Started for Free**” towards the top right of the page, making a note of your username.
3. Each source of data (**to or from** the ESP8266 board) requires its own **feed**. We need to create a feed for our ESP8266 platform to subscribe to such the LED can be controlled.
 - Click on **Feeds**.
 - Click on **Actions > Create a New Feed**.
 - Give it a name **LED** and a suitable description.
 - Click on **Create**.
4. A dashboard is required to control the feed such that the LED on the board can be controlled. Dashboards are also utilised to visualise data published to a feed from the Aston IoT application trainer.
 - Click on **Dashboards**.
 - Click on **Actions > Create a New Dashboard**.
 - Give your Dashboard a suitable name and description.
5. Create a block to control data presented on the feed.
 - Click on the Dashboard you just created.
 - Click on **Create a new block**, .
 - Select a **Toggle** block.
 - Select the **LED feed** you created in step 3.
 - Click Next Step.
 - Set the **button on** text to **HIGH**.
 - Set the **button off** text to **LOW**.
6. Blocks on the dashboard can be edited via the Edit button.
7. In Arduino, click on **Sketch | Include Library | Manage Libraries**. Search for “adafruit mqtt library” and install the latest version of the library.
 - Note that you **should** only have to do this once.
8. Download task1.ino from blackboard to use as a basis for this laboratory. This code includes the majority of the setup completed for you.
 - We create a subscription via creating a subscription object/instance as follows:

```
// Setup a feed called LED to subscribe to HIGH/LOW changes
Adafruit_MQTT_Subscribe LED = Adafruit_MQTT_Subscribe(&MQTT, ADAUSERNAME "/feeds/led");
```

Note that we only need to refer to the MQTT class instance, our Adafruit username and the feed (topic) we wish to subscribe to. For Adafruit IO, all feeds are denoted by /feeds/feedname.
 - At the end of the setup() function, we subscribe to a particular topic/feed, in this case the LED feed. `MQTT.subscribe(&LED);` // subscribe to the LED feed
9. Add the **ssid** and **password** for your mobile hotspot between the quotation marks, as highlighted in the code.
10. Find the **View AIO key** hyperlink on the Adafruit IO page and copy the active AIO key into the sketch as shown in the sketch for the **ADAKKEY** #define macro.
 - Note that this key is secondary password for embedded devices to access your feeds. Hence, treat it with the same respect as any of your other passwords!

11. Also copy your **Adafruit username** into the **ADAUSERNAME** #define macro as shown in the sketch.
12. The code is already setup to subscribe to the LED feed you created earlier. However, in the subscription update checking code, shown below in Figure 1 needs additional code to modify the state of the LED on pin 15 appropriately.
 - Comparisons can be made against the **(char *)LED.lastread** char array using **strcmp**, as in previous laboratories.
 - When the feed is updated to **HIGH**, turn the LED on.
 - When the feed is updated to **LOW**, turn the LED off.

```
// an example of subscription code
Adafruit_MQTT_Subscribe *subscription;           // create a subscriber object instance
while ( subscription = MQTT.readSubscription(5000) ) // Read a subscription and wait for max of 5 seconds.
{                                                  // will return 1 on a subscription being read.
  if (subscription == &LED)                      // if the subscription we have received matches the one we are after
  {
    Serial.print("Recieved from the LED subscription:"); // print the subscription out
    Serial.println((char *)LED.lastread);              // we have to cast the array of 8 bit values back to chars to print

    // ADD code here to compare (char *)LED.lastread against "HIGH" or "LOW". Refer to previous labs
    // for how to use STRCMP. You should use two ifs rather than an if then else structure.
    // we could also convert characters received into integer variables via the use of atoi for passing values from the
    // the broker to the ESP8266

  }
}
```

Figure 1: Subscription update checking code

13. Verify that your sketch works by controlling the LED state via the Adafruit IO MQTT online broker.

SHOW A LECTURER/DEMONSTRATOR TO BE SIGNED OFF.

Task 2: Monitoring potentiometer values remotely via **publishing**

1. Create a new sketch and copy the code from task 1 to use as a starting point.
2. Using the **Adafruit IO MQTT broker system**, create a new feed called **pot**.
3. Add a **Gauge** to the **dashboard** you created earlier and setup the Gauge to read the pot feed.
 - Set the minimum value to 0 and the maximum value to 1023.
4. Add a MQTT global publish object called **POT** as shown below:
 - Note that we pass the address of the MQTT instance, our **Adafruit username** and the feed name. Feeds are always located in **/feeds/feedname**.


```
//Feeds we publish to
Adafruit_MQTT_Publish POT = Adafruit_MQTT_Publish(&MQTT, ADAUSERNAME "/feeds/pot");
```
 - Place this instance in the place highlighted in the sketch.
5. Now we have a MQTT publishing object, we can publish to the feed pot. Measure the potentiometer ADC value and publish the value to the feed **pot**.
 - This can be achieved via the publish class function/method. For example:


```
POT.publish( analogRead(ANALOGUE) );
```

 would publish the ADC value read by **analogRead** to the POT MQTT feed.
 - The analogue pin needs to be defined, refer to previous laboratories.
 - It is good to practice to check if publishing to a feed was successful. **The publish function returns 1 on success and 0 on failure. Create an if structure to print messages over serial to verify if publishing was successful.**
6. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.

7. Verify that your sketch works and displays the value of the potentiometer approximately every 5 seconds on your Adafruit IO dashboard.

SHOW A LECTURER/DEMONSTRATOR TO BE SIGNED OFF

Task 3: Publishing and plotting temperature and humidity values

1. Create a new sketch and copy the code from task 2 to use as a starting point.
2. Create two new feeds to store this data on the Adafruit IO broker system.
3. Create two line chart objects on your dashboard to display the temperature and humidity.
4. Modify your sketch to publish the temperature and humidity values to these feeds once every 5 seconds.
 - You will need to create two new **Adafruit_MQTT_Publish** class instances/objects.
5. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.
6. Verify that your sketch works and updates the line plots on your Adafruit IO dashboard with new values of temperature and humidity approximately every 5 seconds.

Task 4: Making it secure (optional for EE3IOT)

It is relatively easy to modify the code to be **relatively** secure. All previous tasks send all information in **plain text** across the network, this includes the username and password. This obviously has numerous implications! However, it is relatively easy to modify your sketch to use encryption.

1. Change the line `WiFiClient client;` to `WiFiClientSecure client;`.
2. We need a thumbprint of Adafruit's certificate to enable the use of SSL/TLS as a client. Add the following as a global variable to your code (i.e. place it outside of a function).

```
static const char *fingerprint PROGMEM = "77 00 54 2D DA E7 D8 03 27 31
23 99 EB 27 DB CB A5 4C 57 18";
```

3. Change the port used for MQTT (ADAPORT) to **8883** from **1883**.
4. At the end of the `setup()` function but before code configuring subscriptions, add the following line:

```
client.setFingerprint(fingerprint);
```

This configures the API to use the thumbprint for SSL/TLS. All other code can remain the same.

5. Reprogram the Aston AVR and verify the code still works.

SHOW A LECTURER/DEMONSTRATOR TO BE SIGNED OFF