# Lab 4 - MQTT Event Hub - Garage Door Sensor

#### Online Link

The details and code to this lab can be found at: <https://github.com/Rhemsley/MQTT-Garage-Distance-Stoplight>

#### Objective

The purpose of this lab is to learn how to integrate a MQTT system with a broker and multiple clients to create a stoplight LED system controlled by a distance sensor and a magnet sensor. This is to emulate a distance sensor that is used for parking your car in a garage and only turns on and runs the LEDS when the garage is open through the magnet sensor. This is accomplished as follows:

* Implement an Event Hub for publish/subscribe notifications between devices.
* Develop a communications protocol for devices across the event bus.
* Establish more complex conditions for the actuator involving multiple sensor

#### Materials

I built this MQTT system off of the Stoplight Sensor system found at <https://github.com/Rhemsley/Distance_Stoplight>. I also used the following materials to create the needed garage door magnet sensor system:

1 x D1 mini

2 x Male to Male Jumper Cables

1 x Magnet switch

1 x Breadboard

#### References

I used the following resources to complete this lab:

<https://mosquitto.org/download/> The main resource used in installing Mosquitto on the laptop using the Ubuntu method on Ubuntu Windows SubSystem

<http://www.steves-internet-guide.com/install-mosquitto-broker/> Additional Mosquitto tips and config file help

<https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/> MQTT explanation

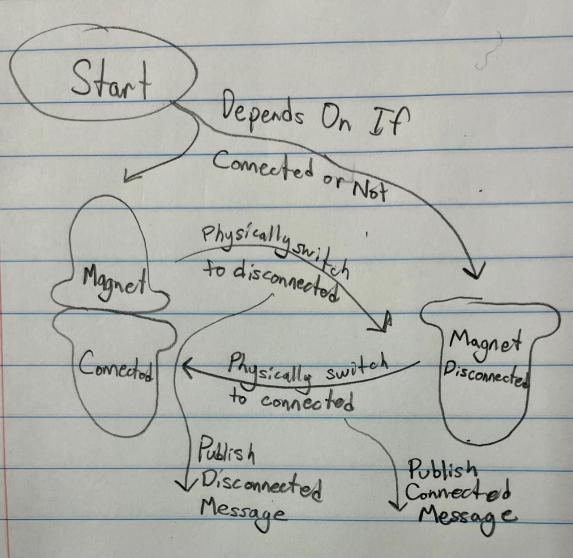
<https://www.vultr.com/docs/install-mosquitto-mqtt-broker-on-ubuntu-20-04-server/> Mosquitto and Mosquitto client install tips

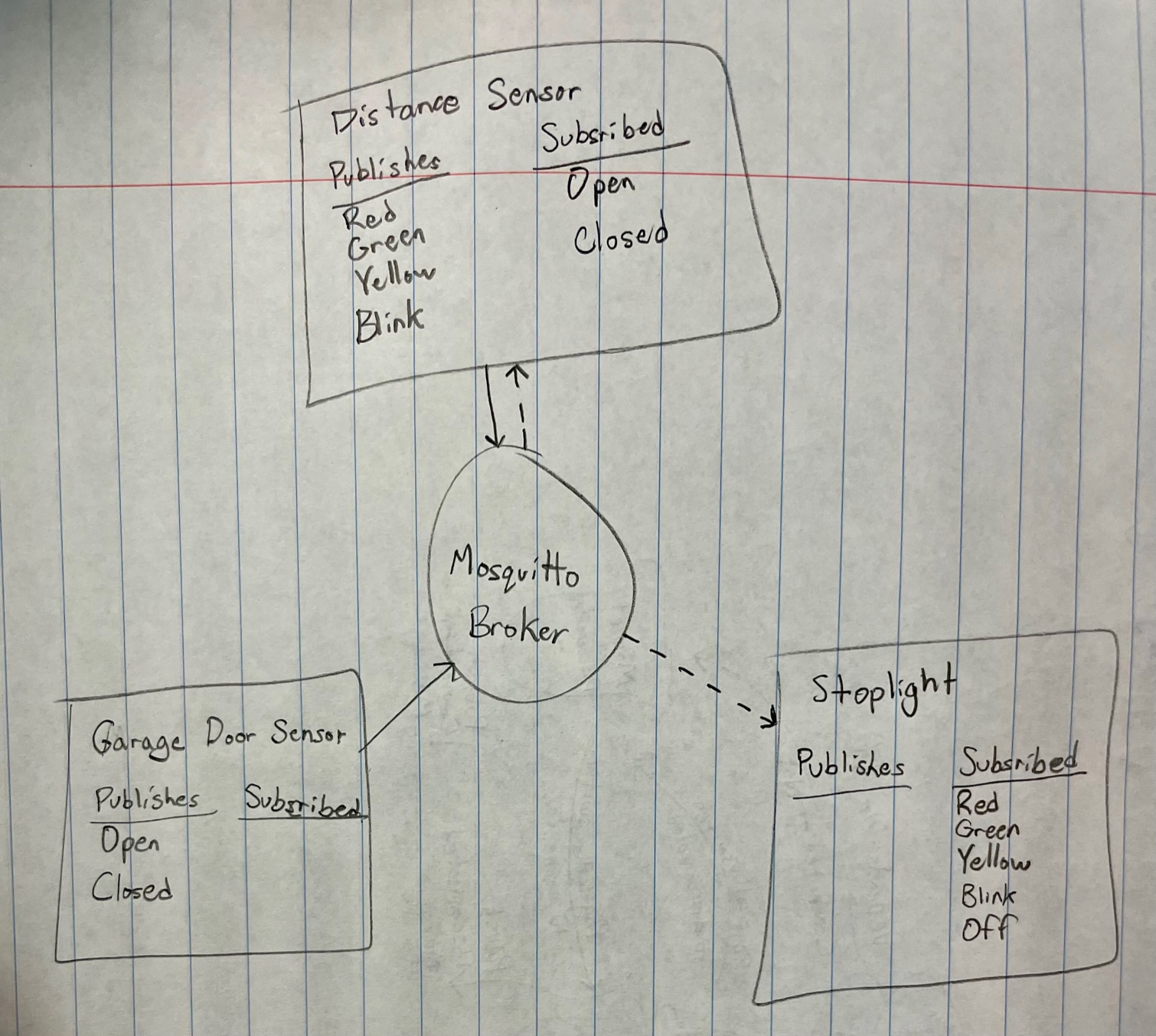
<https://docs.arduino.cc/tutorials/uno-wifi-rev2/uno-wifi-r2-mqtt-device-to-device> Main resource for getting MQTT rolling on an Arduino

<https://iot.stackexchange.com/questions/3397/view-the-messages-sent-to-the-local-mosquitto-server> Mosquitto Client tips

#### Baseline Information

Please see the previous lab linked above for the Flow Charts for all of the LEDs. Below is the Flow Chart for the Magnet Garage Door System. The Magnet simply has the states of connected to indicate the Garage is Closed and disconnected to indicate that the Garage is Open.

Below the Magnet Flow Chart is a general overview diagram of how MQTT works. Then on the right is the diagram for how this system has been implemented using MQTT. For this system we have one broker being ran with Mosquitto on a laptop or a Raspberry Pi and 3 Clients each ran on a D1 Mini Arduino setup.



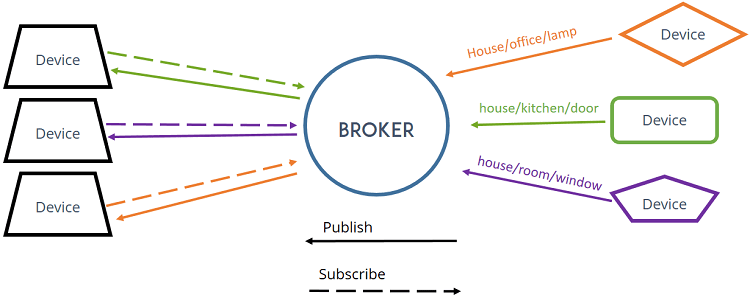
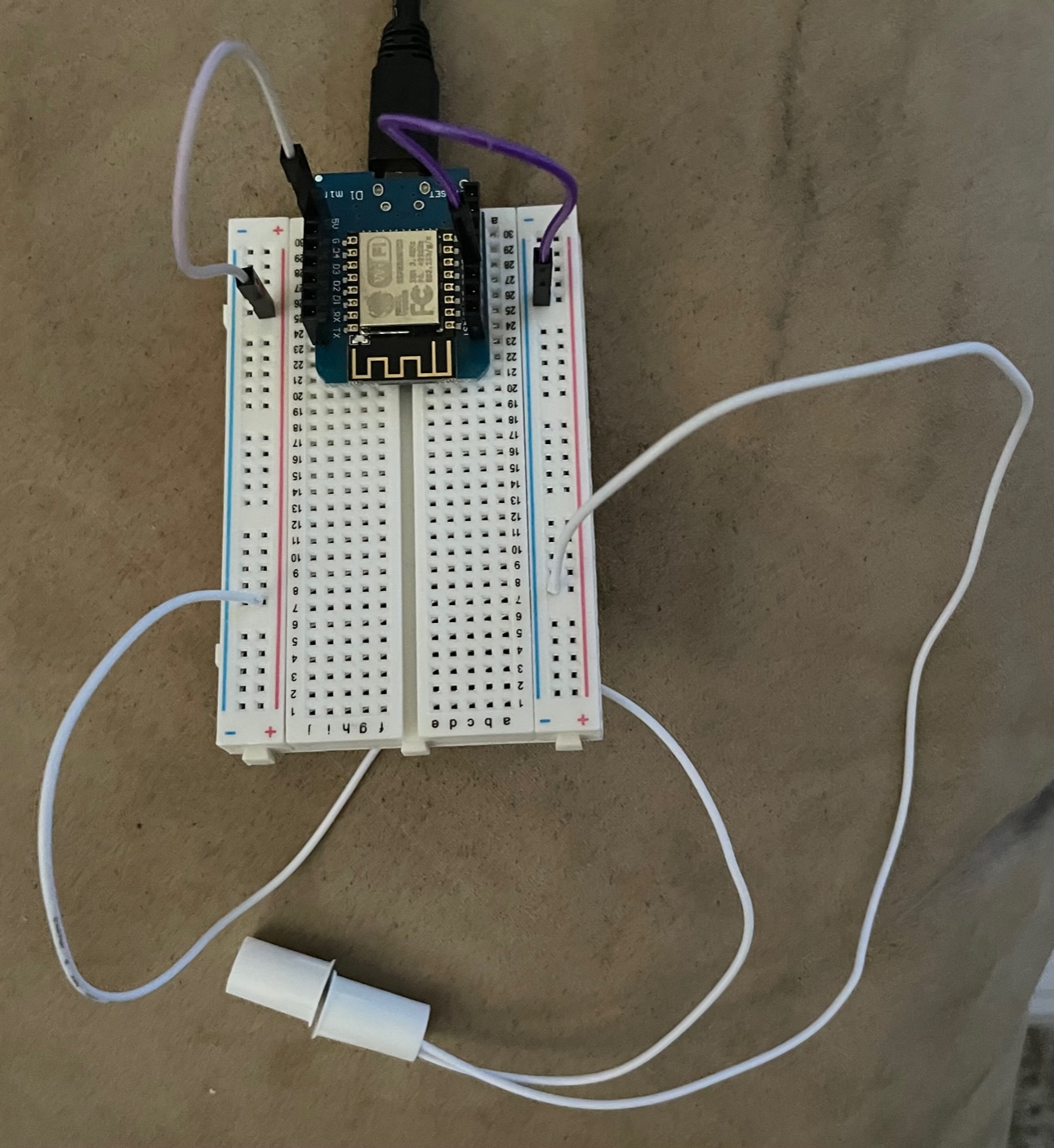


Diagram credit of <https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/>



To the right we have the view of the build with the “Garage Door” Magnet connected to the breadboard in either side of the long power rails as they seemed to hold the ends of the wire better. Then connected to those same rails are the Grey wire on the left which goes to the ground pin and the Purple wire on the right which goes to the D5 pin used as an input pin for this system. This is virtually identical to how a button could be setup and the results of the button would be virtually the same in this system.

#### Procedures

I will assume most if not all of the steps given in the above Distance Sensor Stoplight Git repository have already been followed to create the Stoplight and Distance sensor and what is taking place is understood and I will work off of that baseline knowledge to write these Procedures. The main thing to note though is that all of the code has been modified to switch the system to a MQTT system so the code given in this Git repository and below should be used and not the old code.

1. Get Mosquitto MQTT Broker working on either Laptop or Raspberry Pi
   1. This guide will not get deeply into any potential issues that may occur to get Mosquitto up and running. Many small things could cause issues and that issue should be Googled as there are a lot of good guides and resources to help.
   2. See the Links provided and instructions below to go install Mosquitto. The Mosquitto Operating System can be used but these instructions will be based off of the Mosquitto Linux tool on either a Linux machine like a Raspberry Pi or the Ubuntu Subsystem for Windows.
      1. If using a windows laptop, go to the Microsoft store and download Ubuntu LTS of the version of your choice. For this lab I used version 20.04.5.
      2. Run this application and you should now be able to use the Ubuntu Terminal system on your device. If you run into any issues, Google it because it is common for several issues to occur.
   3. Now in the Terminal run “sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa” then “sudo apt-get update.” This will download Mosquitto it self. If you would like you can also run “sudo apt install mosquitto-clients” to install the client tool and allow your local machine to run client commands like “mosquitto\_pub” and “mosquitto\_sub”.
   4. Now that Mosquitto has been installed, it can be ran with simply typing “mosquitto” however I would recommend running it with the steps below.
      1. First we will make a config file to let the D1 Minis easily work with your version of Mosquitto. To do this, make a config file called “mosquitto.conf” and then within this file put”listener 1883” followed on the next line by “allow\_anonymous true”. This will set the correct listening port and allow the D1 Minis to connect anonymously.
      2. After the config file is started we could start Mosquitto with “mosquitto -c mosquitto.conf” to run Mosquitto using that new config file we just created. On top of this I would actually recommend running “mosquitto -c mosquitto.conf -d” to run mosquitto with the config file in the background which allows you to do other commands like the client commands.
2. Modify code to match your Wifi setup
   1. In the Arduino IDE, get all the code copied in and then in all three of the arduino codes, make sure your Pins are labeled correctly and the WiFi SSID and password is set to match the same network that your host is on.
   2. With this, it is also critical that on each of your codes, it is key that you set the broker[] constant to the IP address of your broker machine. If this is not set correctly then you will not get a MQTT connection on that device and the device will just not even get past that point.

3. Upload the code and check the serial monitor to make sure they connect to WiFi and MQTT

* 1. Now verify and then upload the code to each of the D1 Minis, making sure the correct D1 Mini systems are receiving the correct code.
  2. So long as they all connect to the same network as the Mosquitto broker and then connect to it, they should all be set. If you have any issues connecting to the broker, again double check that the config file is running allowing anonymous connections and check the previous instructions or Google the issue.

4. For distances and instructions on the old systems see the previous labs found in the sensor stoplight git link mentioned a couple times now

5. Working with the Garage Door Magnet sensor

* 1. Simply put, when the magnets are put together, its as if the garage door is closed and therefore the distance sensor is not calling any LED commands, nor is it even reading any distances.
  2. However, if the magnets are separated, it is as if the garage is open and therefore the distance sensor is activated and will start sending the stoplight its requests using different MQTT topics as identified below.

1. MQTT setup
   1. For a quick overview, make sure to review the picture up above over viewing this MQTT setup.
   2. The new Garage Door Sensor is in charge of Publishing the Open or Closed Topics and it is not Subscribed to any topics as it does not need to listen for anything.
   3. Then the Distance Sensor is Subscribed to both the Open and Closed Topics so that it knows when it should be running or not. On top of this, it is then in charge of Publishing the Red, Yellow, Green, Off, and Blink Topics so that the Stoplight knows what LED to run.
   4. Lastly, the Stoplight does not Publish any Topics but it is Subscribed to Red, Yellow, Green, Off, and Blink so that it can know to set the associated LED(s) when it receives a message.
   5. One thing to note is that the sensors are only looking for the Topic calls and do not care what the message is though they are receiving the message and could be coded to do something with those messages.
2. Manual MQTT Publish and Subscribe calls.
   1. Like mentioned, the mosquito-clients tool can be installed to allow the broker device to also be able to Publish or Subscribe to Topics of its choosing. This can be especially useful for debugging or just overviewing what is happening without being connected directly to the D1 Minis and watching the Serial Monitor.
   2. The Publish calls look like “mosquitto\_pub -h localhost -t [Topic] -m “[Message]”” where the [Topic] is replaced with the Topic name of your choosing, like Off, and the [Message] is replaced with your desired message. In this setup the message it self does not really matter but the Topic name is key. Again the possible topic calls are Red, Yellow, Green, Off, and Blink.
   3. The Subscribe calls are simply “mosquitto\_sub -h localhost -t [Topic]” with again the Topic of choice.
   4. If not hosting the broker on the device that you are using mosquitto-client, you could replace the localhost part of those commands with the ip of your broker and it should also work.

#### Observations

#### This lab seemed real scary at first with diving into this new MQTT world but by the end I think it become one of my favorite labs. It involved quite a bit of work as I had to learn about MQTT, setup the broker, and then recode all of the sensors to work with MQTT but luckily the solid understanding of the sensors and MQTT made this very doable and fun.

MQTT seems like a very reasonable and expandable solution to IOT devices as really all it is listening for specific Topics and sending specific topics as each sensor needed. It also seems like there is quite a bit of flexibility in how it is approached and the wide range of devices capable of doing so.

The Garage Door Magnet was a piece of cake once I understood that it was just basically just a button and could be configured about the same way.

As for coding everything, I first got the broker rolling which was actually probably my biggest struggle with the project as I will explain below in a thought question. Then I decided to code up the Garage Door Magnet system and that is when I started figuring out how to implement a Topic system with MQTT. I made just the simple topics of Open and Closed then sent those out and tried to receive them on the Distance Sensor. To my surprise they were received after only very slight tweaking. Then I setup the topic selection to determine what function to call. And lastly it was just about copy, paste, and modify operation on the remaining sensor. Luckily my previous code was very easy to modify to fit a MQTT approach.

#### Thought Questions

1. How does the communication between devices change with the shift to using an event hub? How does this facilitate greater scalability? What things did you do to learn to use MQTT?

It shifts from direct client connections and Get calls to a MQTT system of a broker and clients. On these clients it consisted of Subscribing to the Topics of choice, taking the needed action when the Topic and message is received, and Publishing the messages. So definitely different but still holding a similar code structure to activate the Sensor or LEDs as desired. An absolute key difference is that the first system required direct connections from one client to the other while this system only required a connection to the broker. This did two main things to me, first it made it simpler to connect each device as they had standard connection details and didn’t rely on one devices IP address to be input into the code of the other. The second and more important thing is it allowed for a much more scalable approach as devices just needed to connect to the broker and then they could be Publishing or Subscribing to messages as they pleased. With the system now in place, loads more devices could be added super easily. My main resource on learning was first the lecture as that gave me a good rough understanding and then a couple of the URLs listed above, in particular some of the explanations in the Mosquitto documentation and the explanation given on the arduino URL. I did mess around a bit with the Mosquitto client though not much as I didn’t have too much trouble with the D1 Minis connecting and Publishing data.

1. What are strengths and weaknesses of the direct communication between the sensor and actuator? What are strengths and weaknesses of the event hub? Which do you feel is better for this application?

I think I already hit on some of these in my long answer to the previous question but I would say a main strength for the direct connect version was how it didn’t require a third device/broker. The Broker isn’t too crazy but it is a separate device that the direct communication method didn’t need and another potential vulnerability though in our setup it is not built for security. Then one of the main weaknesses of the event hub could be how much a device could break things if it were to connect and start Publishing false information to mess with the other devices on the network. I could see like a light system setup in MQTT and then a device that goes and messes with all those lights and it could be difficult to stop. I think that the MQTT is definitely better because it allowed the garage door to easily be added without needing to make the sensor also host a page and listen for Get request. As well as it means much more scalability and a easier setup process.

3. What was the biggest challenge you overcame in this lab?

By far my biggest road bump was getting Mosquitto up and running as desired on my Laptop. Interestingly though, this wasn’t really because of a Mosquitto issue as much as it was an issue with my laptops distaste for the Ubuntu Linux SubSystem. I have had the Ubuntu Linux SubSystem on here before and it gave me a real tough time with all sorts of weird error codes and having me go into Turn Windows Features On and Off and do Updates and stuff like that. It would have taken me a lot longer this time if I didn’t have a decent idea of where to mess with things to finally get it going. Once Mosquitto was successfully installed and then running, it was actually mostly smooth sailing.

4. Please estimate the total time you spent on this lab and report.

I would estimate I spent about 5 hours coding/building the project and then 4 hours creating the lab write up.

#### Certification of Work

I certify that the results and solution to this lab were my own work. For the resources of information I found through exploring the internet, I referenced the websites and what I pulled from it. All code written was of my own writing.

-Rylan Hemsley

#### Appendix

Code for reference:

------------------MQTT Garage Door Magnet code-------------------

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <ArduinoMqttClient.h>

// WiFi config

const char\* ssid = "RYLANSLAPTOP"; // Set Wifi Name

const char\* password = "Onyourleft100"; // Set Password

// Mqtt config

const char broker[] = "192.168.137.1";

int port = 1883;

const char topicOff[] = "Off";

const char topicGreen[] = "Green";

const char topicYellow[] = "Yellow";

const char topicRed[] = "Red";

const char topicBlink[] = "Blink";

const char topicOpen[] = "GarageOpen";

const char topicClosed[] = "GarageClosed";

// Wifi and Mqtt instantiation

WiFiClient wifiClient;

MqttClient mqttClient(wifiClient);

// Garage Door status

bool garageOpen = false;

// Send Open Status

void sendGarageOpen() {

mqttClient.beginMessage(topicOpen);

mqttClient.print("Open");

mqttClient.endMessage();

Serial.println("Open status sent");

}

// Send Closed Status

void sendGarageClosed() {

mqttClient.beginMessage(topicClosed);

mqttClient.print("Closed");

mqttClient.endMessage();

Serial.println("Closed status sent");

}

void setup() {

// Setup Serial Monitor with 9600

Serial.begin(9600);

// Create Wifi connection and delay until connected

WiFi.begin(ssid, password);

Serial.println("Connecting to WiFi");

while(WiFi.status() != WL\_CONNECTED) { // Loop until connected to WiFi

delay(500);

Serial.print(".");

}

// If connection successful show IP address in serial monitor

Serial.println("");

Serial.print("Connected to ");

Serial.println(ssid);

Serial.print("IP address: ");

Serial.println(WiFi.localIP()); // Show your assigned IP address

// Connect to MQTT

Serial.print("Attempting to connect to the MQTT broker: ");

Serial.println(broker);

// If unable to connect, output error

if (!mqttClient.connect(broker, port)) {

Serial.print("MQTT connection failed! Error code = ");

Serial.println(mqttClient.connectError());

while (1);

}

Serial.println("You're connected to the MQTT broker!");

Serial.println();

// Setup Magnet pin

pinMode(D5, INPUT\_PULLUP);

// Send starting value

garageOpen = digitalRead(D5);

if (garageOpen) {

Serial.println("Garage Open");

sendGarageOpen();

delay(100);

}

else {

Serial.println("Garage Closed");

sendGarageClosed();

delay(100);

}

}

void loop() {

// Regular calls to send MQTT keep alive

mqttClient.poll();

// Get Magnet/Garage door input status

int pinValue = digitalRead(D5);

delay(10);

// If it has changed status

if (pinValue != garageOpen) {

// Set new status

garageOpen = pinValue;

// If Open, send open, if closed, send closed

if (garageOpen) {

Serial.println("Garage Open");

sendGarageOpen();

delay(100);

}

else {

Serial.println("Garage Closed");

sendGarageClosed();

delay(100);

}

}

}

------------------MQTT Distance Sensor code-------------------

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <ArduinoMqttClient.h>

// WiFi config

const char\* ssid = "RYLANSLAPTOP"; // Set Wifi Name

const char\* password = "Onyourleft100"; // Set Password

// Mqtt config

const char broker[] = "192.168.137.1";

int port = 1883;

const char topicOff[] = "Off";

const char topicGreen[] = "Green";

const char topicYellow[] = "Yellow";

const char topicRed[] = "Red";

const char topicBlink[] = "Blink";

const char topicOpen[] = "GarageOpen";

const char topicClosed[] = "GarageClosed";

// Wifi and Mqtt instantiation

WiFiClient wifiClient;

MqttClient mqttClient(wifiClient);

// Set Sensor Pins

unsigned char echoPin = D1; // Set Echo Pin

unsigned char trigPin = D2; // Set Trig Pin

// Needed variables for sensor and light status

long duration; // sensor duration measurement

int distance1 = 0; // Current distance measurement

int distance2 = 0; // Previous distance measurement 2

int distance3 = 0; // Previous distance measurement 3

int distance4 = 0; // Previous distance measurement 4

int distance5 = 0; // Previous distance measurement 5

int avgDistance = 0; // Sensor duration measurement

bool greenOn = false; // Green status

bool yellowOn = false; // Yellow status

bool redOn = false; // Red status

bool blinkOn = false; // Blink status

bool garageOpen = false; // Garage status

// Subscribed Message handling

void onMqttMessage(int messageSize) {

// Received a message, print out the topic and message size

Serial.println("Received a message with topic '");

Serial.print(mqttClient.messageTopic());

Serial.print("', length ");

Serial.print(messageSize);

Serial.println(" bytes:");

// Save open or close garage door status

if (mqttClient.messageTopic() == topicOpen) {

Serial.println("success for Open");

garageOpen = true;

}

else if (mqttClient.messageTopic() == topicClosed) {

Serial.println("success for Closed");

garageOpen = false;

}

// Use the Stream interface to print the contents

while (mqttClient.available()) {

Serial.print((char)mqttClient.read());

}

Serial.println();

Serial.println();

}

// Send Off request

void lightsOff() {

mqttClient.beginMessage(topicOff);

mqttClient.print("Off");

mqttClient.endMessage();

Serial.println("Off request sent");

}

// Send Green light request

void cycleGreen() {

mqttClient.beginMessage(topicGreen);

mqttClient.print("Green");

mqttClient.endMessage();

Serial.println("Green request sent");

}

// Send Yellow light request

void cycleYellow() {

mqttClient.beginMessage(topicYellow);

mqttClient.print("Yellow");

mqttClient.endMessage();

Serial.println("Yellow request sent");

}

// Send Red light request

void cycleRed() {

mqttClient.beginMessage(topicRed);

mqttClient.print("Red");

mqttClient.endMessage();

Serial.println("Red request sent");

}

// Send Blink light request

void startBlink() {

mqttClient.beginMessage(topicBlink);

mqttClient.print("Blink");

mqttClient.endMessage();

Serial.println("Blink request sent");

}

void setup() {

// Setup Serial Monitor with 9600

Serial.begin(9600);

// Create Wifi connection and delay until connected

WiFi.begin(ssid, password);

Serial.println("Connecting to WiFi");

while(WiFi.status() != WL\_CONNECTED) { // Loop until connected to WiFi

delay(500);

Serial.print(".");

}

// If connection successful show IP address in serial monitor

Serial.println("");

Serial.print("Connected to ");

Serial.println(ssid);

Serial.print("IP address: ");

Serial.println(WiFi.localIP()); // Show your assigned IP address

// Connect to MQTT

Serial.print("Attempting to connect to the MQTT broker: ");

Serial.println(broker);

// If unable to connect, output error

if (!mqttClient.connect(broker, port)) {

Serial.print("MQTT connection failed! Error code = ");

Serial.println(mqttClient.connectError());

while (1);

}

Serial.println("You're connected to the MQTT broker!");

Serial.println();

// Set Received message action to call selector function

mqttClient.onMessage(onMqttMessage);

// Subscribe to desired topics, others commented out for viewing

Serial.print("Subscribing to topics: ");

//Serial.println(topicOff);

//Serial.println(topicGreen);

//Serial.println(topicYellow);

//Serial.println(topicRed);

//Serial.println(topicBlink);

Serial.println(topicOpen);

Serial.println(topicClosed);

Serial.println();

//mqttClient.subscribe(topicOff);

//mqttClient.subscribe(topicGreen);

//mqttClient.subscribe(topicYellow);

//mqttClient.subscribe(topicRed);

//mqttClient.subscribe(topicBlink);

mqttClient.subscribe(topicOpen);

mqttClient.subscribe(topicClosed);

// Setup Sensor Pins

pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output

pinMode(echoPin, INPUT); // Sets the echoPin as an Input

// Start with lights Off

lightsOff(); // Makes sure all the lights are off to start

}

void loop() {

// Regular calls to send MQTT keep alive

mqttClient.poll();

// Only use sensor if Garage is open

if (garageOpen) {

// Sensor distance checking

digitalWrite(trigPin, LOW); // Clears the trigPin

delayMicroseconds(2); // Small delay

digitalWrite(trigPin, HIGH); // Sets the trigPin on HIGH state

delayMicroseconds(10); // For 10 microseconds

digitalWrite(trigPin, LOW); // trigPin back to LOW state

// Reads the echoPin, returns the sound wave travel time in microseconds

duration = pulseIn(echoPin, HIGH);

// Calculating the measured distance in inches. Remove "\* 0.393701" to get cm.

distance1 = duration \* 0.034 / 2 \* 0.393701;

// If a bad distance read, just print a ` and don't add the new distance

if (distance1 >= 120) {

Serial.print("`");

}

else {

// add delay to be measuring roughly 9-10 times a second

delay(100);

// Average the previous 5 distances together to get a rolling average

avgDistance = (distance1 + distance2 + distance3 + distance4 + distance5) / 5;

// Shift all of the measured distances once

distance5 = distance4;

distance4 = distance3;

distance3 = distance2;

distance2 = distance1;

// Prints the avgDistance on the Serial Monitor

Serial.print("Average Distance (inch): ");

Serial.println(avgDistance);

}

// Now to check the distance and set the appropriate LEDs with Get requests

// Distance between 120 and 20 inches, set LED to green if not already green

if ((avgDistance <= 120) && (avgDistance > 20) && greenOn == false) {

// Lights Off then Green on

lightsOff();

cycleGreen();

// All status false but green

greenOn = true;

yellowOn = false;

redOn = false;

blinkOn = false;

}

// Distance between 20 and 12 inches, set LED to yellow if not already yellow

else if ((avgDistance <= 20) && (avgDistance > 12) && yellowOn == false) {

// Lights Off then Yellow on

lightsOff();

cycleYellow();

// All status false but yellow

greenOn = false;

yellowOn = true;

redOn = false;

blinkOn = false;

}

// Distance between 12 and 5 inches, set LED to red if not already red

else if ((avgDistance <= 12) && (avgDistance > 5) && redOn == false) {

// Lights Off then Red on

lightsOff();

cycleRed();

// All status false but red

greenOn = false;

yellowOn = false;

redOn = true;

blinkOn = false;

}

// Distance between 5 and 0 inches, set LED to blinking red if not already blinking

else if ((avgDistance <= 5) && (avgDistance > 0) && blinkOn == false) {

// Lights Off then Blink Red on

lightsOff();

startBlink();

// All status false but blink

greenOn = false;

yellowOn = false;

redOn = false;

blinkOn = true;

}

}

// If garage is closed, turn off lights if any are on

else if (greenOn || yellowOn || redOn || blinkOn) {

// Lights Off

lightsOff();

// All status false, lights are off

greenOn = false;

yellowOn = false;

redOn = false;

blinkOn = false;

}

}

------------------MQTT LED Stoplight Code------------------------

#include <ESP8266WiFi.h>

#include <WiFiClient.h>

#include <ArduinoMqttClient.h>

// Needed variables for LED Pin association

unsigned char greenLed = D3; // Create greenLed Label for pin 3

unsigned char yellowLed = D2; // Create yellowLed Label for pin 2

unsigned char redLed = D1; // Create redLed Label for pin 1

// Needed variables for looping and blinking

unsigned long previousTime = 0; // Create previousTime int for looping

unsigned long startTime = 0; // Create startTime int for looping

bool runStoplight = false; // Create runStoplight bool for looping

bool runBlink = false; // Create runBlink bool for blinking

bool redStatus = false; // Create Red bool for cycling red

bool yellowStatus = false; // Create Yellow bool for cycling yellow

bool greenStatus = false; // Create Green bool for cycling green

// WiFi config

const char\* ssid = "RYLANSLAPTOP"; // Set Wifi Name

const char\* password = "Onyourleft100"; // Set Password

// Mqtt config

const char broker[] = "192.168.137.1";

int port = 1883;

const char topicOff[] = "Off";

const char topicGreen[] = "Green";

const char topicYellow[] = "Yellow";

const char topicRed[] = "Red";

const char topicBlink[] = "Blink";

const char topicOpen[] = "GarageOpen";

const char topicClosed[] = "GarageClosed";

// Set interval for sending messages (milliseconds)

const long interval = 8000;

unsigned long previousMillis = 0;

int count = 0;

// Wifi and Mqtt instantiation

WiFiClient wifiClient;

MqttClient mqttClient(wifiClient);

// Subscribed Message handling

void onMqttMessage(int messageSize) {

// Received a message, print out the topic and message size

Serial.println("Received a message with topic '");

Serial.print(mqttClient.messageTopic());

Serial.print("', length ");

Serial.print(messageSize);

Serial.println(" bytes:");

// If Topic matches a stoplight functioncall, call that function

if (mqttClient.messageTopic() == topicOff) {

handleOff();

}

else if (mqttClient.messageTopic() == topicGreen) {

handleGreen();

}

else if (mqttClient.messageTopic() == topicYellow) {

handleYellow();

}

else if (mqttClient.messageTopic() == topicRed) {

handleRed();

}

else if (mqttClient.messageTopic() == topicBlink) {

handleBlink();

}

// Use the Stream interface to print the contents

while (mqttClient.available()) {

Serial.print((char)mqttClient.read());

}

Serial.println();

Serial.println();

}

// Handle Off call and reset needed variables and LEDs

void handleOff() {

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, LOW); // Turn Red LED off

greenStatus = false; // Indicate green is off

yellowStatus = false; // Indicate yellow is off

redStatus = false; // Indicate red is off

runStoplight = false; // Stop stoplight looping

runBlink = false; // Stop blink looping

previousTime = 0; // Reset previousTime

startTime = 0; // Reset startTime

Serial.println("Turning Off LEDs");

}

// Handle Red call and cycle red LED depending on status

void handleRed() {

// If off turn on and if on turn off

if (redStatus == false) {

digitalWrite(redLed, HIGH); // Turn Red LED on

redStatus = true; // Flip status

}

else {

digitalWrite(redLed, LOW); // Turn Red LED off

redStatus = false; // Flip status

}

Serial.println("Cycling Red");

}

// Handle Yellow call and cycle yellow LED depending on status

void handleYellow() {

// If off turn on and if on turn off

if (yellowStatus == false) {

digitalWrite(yellowLed, HIGH); // Turn Yellow LED on

yellowStatus = true; // Flip status

}

else {

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

yellowStatus = false; // Flip status

}

Serial.println("Cycling Yellow");

}

// Handle Green call and cycle green LED depending on status

void handleGreen() {

// If off turn on and if on turn off

if (greenStatus == false) {

digitalWrite(greenLed, HIGH); // Turn Green LED on

greenStatus = true; // Flip status

}

else {

digitalWrite(greenLed, LOW); // Turn Green LED off

greenStatus = false; // Flip status

}

Serial.println("Cycling Green");

}

// Handle Loop call and setup needed variables

void handleLoop() {

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, LOW); // Turn Red LED off

greenStatus = false; // Indicate green is off

yellowStatus = false; // Indicate yellow is off

redStatus = false; // Indicate red is off

runStoplight = true; // Start stoplight looping

previousTime = millis(); // Set previousTime to current runtime

startTime = millis(); // Then set startTime to current runtime

Serial.println("Starting Stoplight Loop");

}

void handleBlink() {

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, LOW); // Turn Red LED off

greenStatus = false; // Indicate green is off

yellowStatus = false; // Indicate yellow is off

redStatus = false; // Indicate red is off

runBlink = true; // Start blink looping

}

void setup() {

// Setup Serial Monitor with 9600

Serial.begin(9600);

// Create Wifi connection and delay until connected

WiFi.begin(ssid, password);

Serial.println("Connecting to WiFi");

while(WiFi.status() != WL\_CONNECTED) { // Loop until connected to WiFi

delay(500);

Serial.print(".");

}

// If connection successful show IP address in serial monitor

Serial.println("");

Serial.print("Connected to ");

Serial.println(ssid);

Serial.print("IP address: ");

Serial.println(WiFi.localIP()); // Show your assigned IP address

// Connect to MQTT

Serial.print("Attempting to connect to the MQTT broker: ");

Serial.println(broker);

// If unable to connect, output error

if (!mqttClient.connect(broker, port)) {

Serial.print("MQTT connection failed! Error code = ");

Serial.println(mqttClient.connectError());

while (1);

}

Serial.println("You're connected to the MQTT broker!");

Serial.println();

// Set Received message action to call selector function

mqttClient.onMessage(onMqttMessage);

// Subscribe to desired topics, others commented out for viewing

Serial.print("Subscribing to topics: ");

Serial.println(topicOff);

Serial.println(topicGreen);

Serial.println(topicYellow);

Serial.println(topicRed);

Serial.println(topicBlink);

//Serial.println(topicOpen);

//Serial.println(topicClosed);

Serial.println();

mqttClient.subscribe(topicOff);

mqttClient.subscribe(topicGreen);

mqttClient.subscribe(topicYellow);

mqttClient.subscribe(topicRed);

mqttClient.subscribe(topicBlink);

//mqttClient.subscribe(topicOpen);

//mqttClient.subscribe(topicClosed);

// Setup all 3 LED pins and make sure they are off

pinMode(greenLed, OUTPUT); // Set Green LED to Output pin

pinMode(yellowLed, OUTPUT); // Set Yellow LED to Output pin

pinMode(redLed, OUTPUT); // Set Red LED to Output pin

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, LOW); // Turn Red LED off

}

void loop() {

// Regular calls to send MQTT keep alive

mqttClient.poll();

// Stoplight Controlling only runs when set to true

if (runStoplight == true) {

// compare last run to starting time, turn to seconds, then

// check if within first second of three second loop

if ((((previousTime - startTime) / 1000)%3) == 0) {

digitalWrite(greenLed, HIGH); // Turn Green LED on

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, LOW); // Turn Red LED off

Serial.println("Loop Green ON");

previousTime = millis(); // Set previous time to current time

}

// compare last run to starting time, turn to seconds, then

// check if within second second of three second loop

else if ((((previousTime - startTime) / 1000)%3) == 1) {

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, HIGH); // Turn Yellow LED on

digitalWrite(redLed, LOW); // Turn Red LED off

Serial.println("Loop Yellow ON");

previousTime = millis(); // Set previous time to current time

}

// compare last run to starting time, turn to seconds, then

// check if within third second of three second loop

else if ((((previousTime - startTime) / 1000)%3) == 2) {

digitalWrite(greenLed, LOW); // Turn Green LED off

digitalWrite(yellowLed, LOW); // Turn Yellow LED off

digitalWrite(redLed, HIGH); // Turn Red LED on

Serial.println("Loop Red ON");

previousTime = millis(); // Set previous time to current time

}

}

// Blink operation only runs when set to true

if (runBlink == true) {

// If off turn on and if on turn off

if (redStatus == false) {

digitalWrite(redLed, HIGH); // Turn Red LED on

redStatus = true; // Flip status

}

else {

digitalWrite(redLed, LOW); // Turn Red LED off

redStatus = false; // Flip status

}

Serial.println("Cycling Red");

delay(100); // Delay for 100ms

}

}