**[Updated MQTT/ESP8266 sketch to automatically reconnect Wifi and MQTT if the Wifi went down](https://openhardwarecoza.wordpress.com/2015/07/09/updated-mqttesp8266-sketch-to-automatically-reconnect-wifi-and-mqtt-if-the-wifi-went-down/)**

I made a few changes to the m:on(“offline”) function to handle failed Wifi connections better.

I had a few issues where my wifi router restarted, upon restarting (with the old sketch on the ESP07) once the wifi was back up, it reconnected to the Wifi though the magic of NodeMCU’s wifi.sta.autoconnect(1) functionality.  However despite the Wifi reconnecting, the MQTT stopped working.

I added a new function called reconnect() to handle the dropped connection.   It sets up a timer action to check when the Wifi is up again (Status = 5 and IP is not = null) then reruns the MQTT connect/subscribe:

function reconnect()

print ("Waiting for Wifi")

if wifi.sta.status() == 5 and wifi.sta.getip() ~= nil then

print ("Wifi Up!")

tmr.stop(1)

m:connect(Broker, 1883, 0, function(conn)

print("Mqtt Connected to:" .. Broker)

mqtt\_sub() --run the subscription function

end)

end

end

Here’s the full sketch now

* Elegant reconnect after wifi fails
* 7 channels
* /dev1/ch1-7 mqtt names

-- One time ESP Setup --

wifi.setmode(wifi.STATION)

wifi.sta.config ( "openhardwarecoza" , "novell1234" )

print(wifi.sta.getip())

Broker="192.168.1.200"

--GPIO2 is connected to LED via resistor, initially off

gpio.mode(1,gpio.OUTPUT)

gpio.write(1,gpio.LOW)

gpio.mode(2,gpio.OUTPUT)

gpio.write(2,gpio.LOW)

gpio.mode(3,gpio.OUTPUT)

gpio.write(3,gpio.LOW)

gpio.mode(4,gpio.OUTPUT)

gpio.write(4,gpio.LOW)

gpio.mode(5,gpio.OUTPUT)

gpio.write(5,gpio.LOW)

gpio.mode(6,gpio.OUTPUT)

gpio.write(6,gpio.LOW)

gpio.mode(7,gpio.OUTPUT)

gpio.write(7,gpio.LOW)

function reconnect()

print ("Waiting for Wifi")

if wifi.sta.status() == 5 and wifi.sta.getip() ~= nil then

print ("Wifi Up!")

tmr.stop(1)

m:connect(Broker, 1883, 0, function(conn)

print("Mqtt Connected to:" .. Broker)

mqtt\_sub() --run the subscripion function

end)

end

end

m = mqtt.Client("KITCHEN", 180, "user", "password")

m:lwt("/lwt", "ESP8266", 0, 0)

m:on("offline", function(con)

print ("Mqtt Reconnecting...")

tmr.alarm(1, 10000, 1, function()

reconnect()

end)

end)

-- on publish message receive event

m:on("message", function(conn, topic, data)

print("Recieved:" .. topic .. ":" .. data)

if (data=="ON") and (topic=="/dev1/ch1/com") then

print("Enabling Output")

gpio.write(1,gpio.HIGH)

m:publish("/dev1/ch1/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch1/com") then

print("Disabling Output")

gpio.write(1,gpio.LOW)

m:publish("/dev1/ch1/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch2/com") then

print("Enabling Output")

gpio.write(2,gpio.HIGH)

m:publish("/dev1/ch2/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch2/com") then

print("Disabling Output")

gpio.write(2,gpio.LOW)

m:publish("/dev1/ch2/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch3/com") then

print("Enabling Output")

gpio.write(3,gpio.HIGH)

m:publish("/dev1/ch3/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch3/com") then

print("Disabling Output")

gpio.write(3,gpio.LOW)

m:publish("/dev1/ch3/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch4/com") then

print("Enabling Output")

gpio.write(4,gpio.HIGH)

m:publish("/dev1/ch4/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch4/com") then

print("Disabling Output")

gpio.write(4,gpio.LOW)

m:publish("/dev1/ch4/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch5/com") then

print("Enabling Output")

gpio.write(5,gpio.HIGH)

m:publish("/dev1/ch5/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch5/com") then

print("Disabling Output")

gpio.write(5,gpio.LOW)

m:publish("/dev1/ch5/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch6/com") then

print("Enabling Output")

gpio.write(6,gpio.HIGH)

m:publish("/dev1/ch6/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch6/com") then

print("Disabling Output")

gpio.write(6,gpio.LOW)

m:publish("/dev1/ch6/state","OFF",0,0)

elseif (data=="ON") and (topic=="/dev1/ch7/com") then

print("Enabling Output")

gpio.write(7,gpio.HIGH)

m:publish("/dev1/ch7/state","ON",0,0)

elseif (data=="OFF") and (topic=="/dev1/ch7/com") then

print("Disabling Output")

gpio.write(7,gpio.LOW)

m:publish("/dev1/ch7/state","OFF",0,0)

end

end)

function mqtt\_sub()

m:subscribe("/dev1/#",0, function(conn)

print("Mqtt Subscribed to OpenHAB feed for device KITCHEN")

end)

end

tmr.alarm(0, 1000, 1, function()

print ("Connecting to Wifi... ")

if wifi.sta.status() == 5 and wifi.sta.getip() ~= nil then

print ("Wifi connected")

tmr.stop(0)

m:connect(Broker, 1883, 0, function(conn)

print("Mqtt Connected to:" .. Broker)

mqtt\_sub() --run the subscription function

end)

end

end)

[**OpenHAB, MQTT, Arduino and ESP8266: Part 1 – Setting up your environment**](https://openhardwarecoza.wordpress.com/2015/03/29/openhab-mqtt-arduino-and-esp8266-part-1-setting-up-your-environment/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

This article assumes the reader is familiar with WHAT the various components mentioned above is.  You won’t need an in depth understanding – I will try and explain, but as long as you understand what each of the following is you should be able to follow along:

* MQTT: Message Queue Telemetry Transport) is a publish-subscribe based “light weight” messaging protocol for use on top of the TCP/IP protocol. It is designed for connections with remote locations where a “small code footprint” is required and/or network bandwidth is limited. The Publish-Subscribe messaging pattern requires a message broker. The broker is responsible for distributing messages to interested clients based on the topic of a message. Andy Stanford-Clark and Arlen Nipper of Cirrus Link Solutions authored the first version of the protocol in 1999.  See <http://mqtt.org/>
* Mosquitto:  A MQTT Broker. A broker is a service running between the publisher and the subscriber – provides message exchange, QoS, etc.  This is the most important core for OpenHAB to be able to interact with Hardware in this configuration  See <http://mosquitto.org/>
* NodeMCU:  An alternative firmware for ESP8266 Wifi Modules.  Using LUA as a language – very easy to program, comes with good MQTT Support.  See <http://nodemcu.com/index_en.html>
* OpenHAB – Open Source Home Automation platform – ties all the various bits of (vendor agnostic) hardware together into a harmonious control system.  See <http://www.openhab.org/>

The very first thing you would need to do before you can begin, is to install the following software.  In my case I am doing all my configuration a Ubuntu 14.04 server.  I did try a Raspberry Pi V1 but it was just too slow.

Add the OpenHAB repository:

1.  Add the OpenHab Repository

sudo nano /etc/apt/sources.list.d/openhab.list

2.  Insert

deb http://repository-openhab.forge.cloudbees.com/release/1.6.2/apt-repo/ /

and save the file

3.  Install Java

echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee /etc/apt/sources.list.d/webupd8team-java.list

echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu trusty main" | tee -a /etc/apt/sources.list.d/webupd8team-java.list

apt-get install oracle-java8-installer

4. Install OpenHAB

sudo apt-get install openhab-runtime openhab-addon-binding-mqtt openhab-addon-action-mail openhab-addon-binding-bluetooth openhab-addon-binding-serial openhab-addon-binding-weather openhab-addon-persistence-rrd4j

Note there are a LOT more add-ons, bindings, etc:  Have a look at which others may interest you:

sudo apt-cache search openhab

5.  Install Mosquitto

sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa

sudo apt-get install mosquitto mosquitto-clients

6.  Start and Test Mosquitto

sudo /etc/init.d/mosquitto start

On one terminal:  Start a Subscription client to test with:

mosquitto\_sub -d -t hello/world

On a seperate terminal: Push a publication to the hello/world topic:

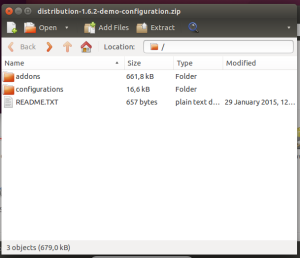
mosquitto\_pub -d -t hello/world -m "Hello from MQTT"

7.  Demo OpenHAB Configuration

You can download a OpenHAB demo configuration from <https://github.com/openhab/openhab/releases/download/v1.6.2/distribution-1.6.2-demo-configuration.zip>

For me this helped quite a bit – since seeing how things get done, often helps getting it done

Open the ZIP:

[](https://openhardwarecoza.files.wordpress.com/2015/03/democonfig.png)

Extract “addons” to /usr/share/openhab (overwrite existing folder)

Extract “configurations” to /etc/openhab/ (overwrite existing folder)

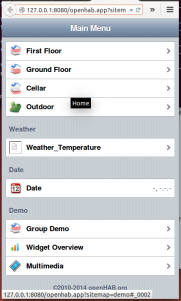
Proceed to start OpenHAB

sudo /etc/init.d/openhab start

8.  Log into your demo sitemap:

<http://127.0.0.1:8080/openhab.app?sitemap=demo>

This will allow you to explore the basic UI.   Yes I agree it looks like an old version of iOS  – blegh – but OpenHAB2 is currently in development and besides OpenHAB1 can be controlled through the much more awesome Android/iOS apps, as well as the fact that once properly configured you shouldn’t have to log into the web interface ever since your smart wall panels, presence detection, automation rules etc is suppose to control your habitat based on Events not clicks (:

[](https://openhardwarecoza.files.wordpress.com/2015/03/demohab.png)

9.  Configure OpenHAB to use a MQTT Binding

sudo vi /etc/openhab/configurations/openhab.cfg

At the very bottom, define a MQTT Broker:

mqtt:broker.url=tcp://192.168.0.100:1883

mqtt:broker.clientId=openhab

Exit, Save, and restart OpenHAB

sudo /etc/init.d/openhab restart

[**OpenHAB, MQTT, Arduino and ESP8266: Part 2 – Publish, Subscribe, Command, State, and WTFs**](https://openhardwarecoza.wordpress.com/2015/03/29/openhab-mqtt-arduino-and-esp8266-part-2-publish-subscribe-command-state-and-wtfs/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

In this part, we will begin to build our own basic configuration (replacing the Demo configuration) but for now, left have a quick talk about MQTTs role in the OpenHAB event bus (or as items)

MQTT allows you to exchange messages, via a broker – by Publishing a message to a Broker.  All the clients that are Subscribed to the topic you are publishing to, will receive the message once published.

A topic can be for example

* /home/openHAB/in/GarageDoor/state
* /home/openHAB/out/GroundFloor\_Office\_Light/command

A message publish to these topics could be:

* /home/openHAB/in/GarageDoor/state = OPEN
* /home/openHAB/out/GroundFloor\_Office\_Light/command = OFF

And a subscription, could be to

* /home/openHAB/in/GarageDoor/state (will receive OPEN from the broker when the published sends OPEN)
* /home/#  – wait what’s this?  I can subscribe to an exact subtopic, OR I can subscribe to a list of topics (:  – very useful for a device with many functions (:

Now before you even begin configuring devices, I want to lay down some advice I learned along the way during this project:

1. Decide on STANDARDS RIGHT NOW!!! Deciding on topic names should make a lot of sense as you will be typing them a lot (short is better) and tracking message flow through Mqtt-spy – you want it to be clear enough that you can immediately see what you are looking for
2. Configure any device to send a confirmation “state” message after it switches or gets queried, this will just make sure your OpenHAB instance stays sane!  The worst thing is when you see a light as switched off in the OpenHAB interface only to be blinded by it still shining in your eyes… Initially I thought I was battling hardware configs, then I thought I had relays latching, then I thought I had a bad network…   Later on I realised that adding simple Status publishes from the device to ensure OpenHAB always knows its real state – is a LOT more important and robust

In my initial playing I followed the naming from the demo configuration  for example GF\_Office\_Lamp for example for a Ground Floor (level),  Office (room) Lamp (device)

As soon as I started designing hardware I realised this standard is going to become HELL!!!

Instead I decided to have a much simpler Numbering scheme and rather configure names in OpenHAB to avoid confusion

So for example instead of trying to address the desk lamp in my office as

/home/openHAB/out/GF\_Office\_Lamp/command=ON

I am rather going for

/home/1/ard1/p1/com=ON

* /home/ is a wide topic set covering all the home automation messages.
* /home/1/ signifies Room 1 (my office) – makes for a LOT less typing
* /home/1/ard1 is a device in my Office – a Arduino with a ethernet shield
* /home/1/ard1/p1 is Port 1 of this Arduino device.   Since the Arduino sketch can handle tens of relays – I can have P1-P14 (:
* /home/1/ard1/p1/com = Command topic – for P1 I want both a Command and a State topic available:  Command being messages that CONTROL the device, and State being messages from the device, telling OpenHAB it’s status (ON or OFF?)

In my OpenHAB item file I can manage this device by defining it as:

Switch lamp1 "Office Lamp" (all) {mqtt=">[broker:/home/1/ard1/p1/com:command:on:ON],>[broker:/home/1/ard1/p1/com:command:off:OFF],<[broker:/home/1/ard1/p1/com/state:state:default"}

You may notice that for this device I expect it to have two Commands (ON or OFF) and one subscription to the State topic (to change the status of the “Switch” object)

[lamp1](https://openhardwarecoza.files.wordpress.com/2015/03/lamp1.png)

Depending on the device (for example in the Arduino sketch) you may want the message to be formatted differently.  In my example sketches later on you may notice my Arduino sketch expects 1 or 0 for on/off.  The ESP8266 sketch expects ON|OFF

You can specify this on the configuration line as well:

Switch lamp1 "Office Lamp" (all) {mqtt=">[broker:/home/1/ard1/p1/com:command:on:1],>[broker:/home/1/ard1/p1/com:command:off:0],<[broker:/home/1/ard1/p/com/state:state:default"}

[**OpenHAB, MQTT, Arduino and ESP8266: Part 3: Hardware: Arduino with Ethernet Shield**](https://openhardwarecoza.wordpress.com/2015/03/29/openhab-mqtt-arduino-and-esp8266-part-3-hardware-arduino-with-ethernet-shield/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

Lets dive right in:  
Take a Arduino Uno, and install a standard Ethernet shield on top.   Wire pin D4 and D5 to a two channel relay breakout or just LEDs (for now – just testing phase)

Wire a I2C LCD to SDA/SCL and VCC/GND  – I used this as a supplementary check of what my code is doing but this may also make it into any final hardware version

#include <SPI.h>

#include <Ethernet.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal\_I2C.h>

// Update these with values suitable for your network.

byte mac[]    = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };

byte server[] = { 192, 168, 0, 100 }; // IP Address of your MQTT Server

byte ip[]     = { 192, 168, 0, 120 }; // IP for this device

EthernetClient ethClient;

PubSubClient client(server, 1883, callback, ethClient);

LiquidCrystal\_I2C    lcd(0x27,2,1,0,4,5,6,7); // 0x27 is the I2C bus address for an unmodified backpack

byte bulb[8] = {

    0b01110,

    0b10001,

    0b10001,

    0b10001,

    0b01110,

    0b01010,

    0b01110,

    0b00100

};

byte plug[8] = {

    0b01010,

    0b01010,

    0b11111,

    0b10001,

    0b10001,

    0b01110,

    0b00100,

    0b00100

};

void callback(char\* topic, byte\* payload, unsigned int length) {

Serial.println("Callback");

Serial.print("TopicSerial.println(topic);

Serial.print("LengthSerial.println(length);

Serial.print("PayloadSerial.write(payload,length);

Serial.println();

  if (strcmp(topic,"/home/1/ard1/p1/com")==0) {

   if (payload[0] == '0')

    {

    digitalWrite(4, LOW);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Fan OFF");

    delay(100);

    client.publish("/home/1/ard1/p1/state","0");

    }

    else if (payload[0] == '1')

    {

    digitalWrite(4, HIGH);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Fan ON");

    delay(100);

    client.publish("/home/1/ard1/p1/state","1");

    }

   }

  if (strcmp(topic,"/home/1/ard1/p2/com")==0) {

   if (payload[0] == '0')

    {

    digitalWrite(5, LOW);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Light OFF");

    delay(100);

    client.publish("/home/1/ard1/p2/state","0");

    }

    else if (payload[0] == '1')

    {

    digitalWrite(5, HIGH);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Light ON");

    delay(100);

    client.publish("/home/1/ard1/p2/state","1");

    }

   }

}

void setup()

{

  Serial.begin(9600);

  lcd.begin (20,4); // for 16 x 2 LCD module

  lcd.createChar(0, bulb);

  lcd.createChar(1, plug);

  lcd.home(); // set cursor to 0,0

  lcd.print(" Starting  ");

  delay(1000);

  Ethernet.begin(mac, ip);

    lcd.home();

    lcd.clear();

    lcd.print("Ethernetected");

  if (client.connect("arduinoClient")) {

    lcd.home();

    lcd.clear();

    lcd.print("MTQQected");

    client.publish("outTopic","hello world");

    client.subscribe("/home/1/ard1/#"); // Subscribe to all messages for this device

  }

  pinMode(4, OUTPUT);

  digitalWrite(4, LOW);

  pinMode(5, OUTPUT);

  digitalWrite(5, LOW);

}

void loop()

{

  client.loop();

}

Now comes the interesting time: Time to start customizing OpenHAB to handle this newly built hardware (:

1.   Create a new Sitemap – sitemaps in OpenHAB define which items to display

sudo vi /etc/openhab/configurations/sitemaps/dolphin.sitemap

Inside this file, paste the following content

sitemap dolphin label="Main Menu"

{

Frame label="MQTT" {

Switch item=mqttsw1 label="MQTT Switch 1"

Switch item=mqttsw2 label="MQTT Switch 2"

Switch item=lamp1 label="Office Lamp"

}

}

Lets configure these three new items

sudo vi /etc/openhab/configurations/items/dolphin.items

Add the following to the new file

Group All

Switch mqttsw1 "Switch 1" (all) {mqtt=">[broker:/home/1/ard1/p1/com:command:on:1],>[broker:/home/1/ard1/p1/com:command:off:0]"}

Switch mqttsw2 "Switch 2" (all) {mqtt=">[broker:/home/1/ard1/p2/com:command:off:0],>[broker:/home/1/ard1/p2/com:command:on:1]"}Restart OpenHAB

sudo /etc/init.d/openhab restart

Once it has started, access your new sitemap:

<http://127.0.0.1:8080/openhab.app?sitemap=dolphin>

[](https://openhardwarecoza.files.wordpress.com/2015/03/mqtt.png)

Lets recap the configuration:

1. We have a device(s) on /home/1/ard1/p1/com and /home/1/ard1/p2/com
2. These devices subscribe to our MQTT broker on 192.168.0.100:1883 (See the arduino Sketch)
3. Our MQTT Broker, Mosquitto, runs on 192.168.0.100:1883
4. OpenHAB has a Broker configuration in /etc/openhab/configurations/openhab.cfg (Where OpenHAB can Publish commands, for example turn ON or OFF /home/1/ard1/p1)
5. We configured a sitemap with two items (sorry screenshot above shows 3 devices – ahead of myself there) under /etc/openhab/configurations/sitemaps/dolphin.sitemap (we’ll add groups and other cool bits later – for now the basics)
6. The items in our sitemap, has relevant configurations under /etc/openhab/configurations/items/dolphin.items
7. These items have a MQTT Binding to send 1 = ON and 2 = OFF

{mqtt=">[broker:/home/1/ard1/p1/com:command:on:1],>[broker:/home/1/ard1/p1/com:command:off:0]"}

[**OpenHAB, MQTT, Arduino and ESP8266: Part 4: Hardware: ESP8266 with NodeMCU firmware**](https://openhardwarecoza.wordpress.com/2015/03/30/openhab-mqtt-arduino-and-esp8266-part-4-hardware-esp8266-with-nodemcu-firmware/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

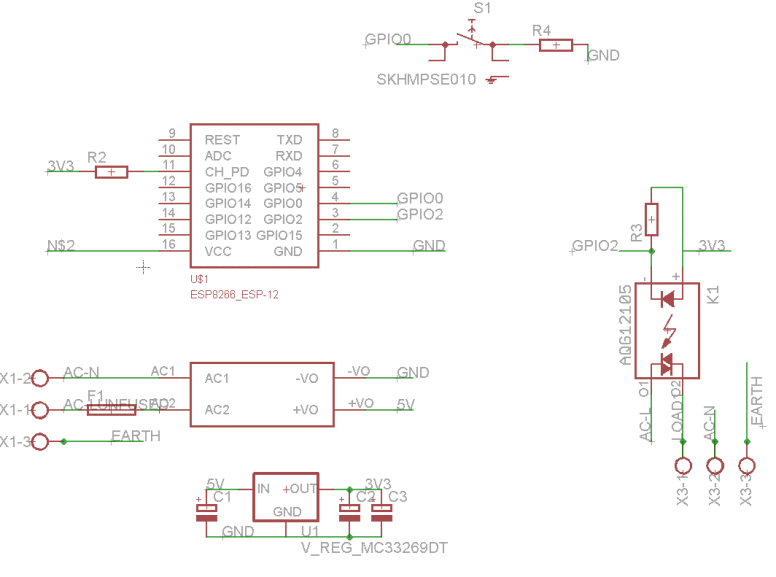
* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

If you are not familiar with the ESP8266, I will be very brief since the NodeMCU Wiki is a huge treasure chest of information!  See  <https://github.com/nodemcu/nodemcu-firmware/wiki>

The ESP8266 is a CHEAP (Undcer $5) Wifi module with built in microcontroller.  NodeMCU is a replacement firmware for the default Espessif firmware, that turns the onboard microcontroller into a LUA interpreter.  In my plans these will be for simple little inline power switches on table lamps, desk fans, and the like.  I am designing a PCB with a ESP8266, 220v->5v PSU (HLK-PM01), 3.3v LDO Regulator, Omron G3MB-202P Solid state relay (2A @240v = 480w max, so ideal for 60w desk lamps, etc) and all the supporting circuitry

So lets dive right in.

Hardware setup:

[](https://openhardwarecoza.files.wordpress.com/2015/03/schematic-node.png)

LUA Sketch: Upload as init.lua (Again – this assumes you have played with NodeMCU before)

Also assumes you have already configured Wifi on the ESP8266

--Init

 DeviceID="esp01"

RoomID="1"

 Broker="192.168.0.100"

--GPIO0 is connected to switch with internal pullup enabled

gpio.mode(3,gpio.INPUT,gpio.PULLUP)

 --GPIO2 is connected to LED via resistor, initially off

 gpio.mode(4,gpio.OUTPUT)

 gpio.write(4,gpio.HIGH)

 m = mqtt.Client("ESP8266".. DeviceID, 180, "user", "password")

 m:lwt("/lwt", "ESP8266", 0, 0)

 m:on("offline", function(con)

    print ("Mqtt Reconnecting...")

    tmr.alarm(1, 10000, 0, function()

      m:connect(Broker, 1883, 0, function(conn)

        print("Mqtt Connected to:" .. Broker)

        mqtt\_sub() --run the subscription function

      end)

    end)

 end)

-- Pin to also toggle the status so you don't have to keep pulling out your phone to turn on device

 gpio.trig(3, "down",function (level)

    local PinValue=gpio.read(4)

    --Change the state

    if (PinValue==0) then

      --The read resets the output to 0, put it back

      gpio.write(4,1)

      print("Light was on, turn off")

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","OFF",0,0)

    else

      gpio.write(4,0)

      print("Light was off, turn on")

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","ON",0,0)

     end

 end)

 -- on publish message receive event

 m:on("message", function(conn, topic, data)

    print("Recieved:" .. topic .. ":" .. data)

      if (data=="ON") then

      print("Enabling Output")

      gpio.write(4,gpio.LOW)

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","ON",0,0)

    elseif (data=="OFF") then

      print("Disabling Output")

      gpio.write(4,gpio.HIGH)

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","OFF",0,0)

    else

      print("Invalid - Ignoring")

    end

 end)

 function mqtt\_sub()

    m:subscribe("/home/".. RoomID .."/" .. DeviceID .. "/p1/com",0, function(conn)

      print("Mqtt Subscribed to OpenHAB feed for device " .. DeviceID)

    end)

 end

 tmr.alarm(0, 1000, 1, function()

  if wifi.sta.status() == 5 and wifi.sta.getip() ~= nil then

    tmr.stop(0)

    m:connect(Broker, 1883, 0, function(conn)

      print("Mqtt Connected to:" .. Broker)

      mqtt\_sub() --run the subscription function

    end)

  end

 end)

You should see some familiarity with the Arduino sketch

1. We connect to a Broker and subscribes to the topic

/home/".. RoomID .."/" .. DeviceID .. "/p1/com

RoomID and DeviceID is configured near the top of the sketch as each device needs a different ID and each room in the house too

1. When the device is turned on or off (either from the physical button on GPIO0, or from a MQTT Command, we send back the status to MQTT by publishing the state to

/home/".. RoomID .."/" .. DeviceID .. "/p1/state

Right, left add this to our OpenHAB configuration:

sudo vi /etc/openhab/configurations/sitemaps/dolphin.sitemap

and add the line for lamp1

sitemap dolphin label="Main Menu"

{

   Frame label="MQTT" {

   Switch item=mqttsw1 label="MQTT Switch 1"

   Switch item=mqttsw2 label="MQTT Switch 2"

   Switch item=lamp1 label="Office Lamp"

  }

}

Lets configure the new item:

sudo vi /etc/openhab/configurations/items/dolphin.items

Add the Switch lamp1 item:

Group All

Switch mqttsw1 "Switch 1" (all) {mqtt=">[broker:/testsw/1:command:on:1],>[broker:/testsw/1:command:off:0]"}

Switch mqttsw2 "Switch 2" (all) {mqtt=">[broker:/testsw/2:command:off:0],>[broker:/testsw/2:command:on:1]"}

Switch lamp1 "Office Lamp" (all){mqtt=">[broker:/home/1/esp01/p1/com:command:on:ON],>[broker:/home/1/esp01/p1/com:command:off:OFF],<[broker:/home/1/esp01/p1/state:state:default]"}

Save and exit, restart OpenHAB

sudo /etc/init.d/openhab restart

Once it has started, access your new sitemap: <http://127.0.0.1:8080/openhab.app?sitemap=dolphin>

[](https://openhardwarecoza.files.wordpress.com/2015/03/mqtt.png)

Now, you may notice that the line in dolphin.items looks a little different…  We have a third parameter in the configuration: (Compare it to the two switches above it)

In a MQTT Binding on OpenHAB we define the direction of the message with a > (outgoing) or a < (Incoming) before the configuration

I color coded the three parameters below to highlight the point.  The first to OUTPUTS (>) a message into **/home/1/esp01/p1/com**, whereas the third parameter INPUTS (<) a message from **/home/1/esp01/p1/state**

{mqtt=">[broker:/home/1/esp01/p1/com:command:on:ON],>[broker:/home/1/esp01/p1/com:command:off:OFF],<[broker:/home/1/esp01/p1/state:state:default]"}

Remember in Part 2 I mentioned you want to stay sane?  Well, this is how simple that is.  By configuring the device to PUBLISH its status whenever it changes, and configuring the OpenHAB item to listen for that change – we ensure the dashboard always matches the physical (malfunction like a dead bulb or stuck relay not withstanding of coarse)

I have the State Publish code in the Arduino sketch too, we just didnt add it on the mqtt binding for the two earlier items for simplicity’s sake.  As an exercise you could use the third as an example and configure it now (:

The second reason for pushing the state back to OpenHAB is that I may not always want OpenHAB to be the ONLY control mechanism.  I mean do you really want to pull out your phone each time you walk into a room just to turn on the light?  No, many people would prefer a physical button on/near the device you want to turn on/off.

If you look at the lua sketch above you should see I already added it into this device:

This section of the sketch above handles just that.  By reading the status of a button on GPIO3, and accordingly toggles GPIO4 to the opposite of its current state  (Note it toggles the pin state) and then also publishes the change to OpenHAB (to update the item’s state)

-- Pin to also toggle the status so you don't have to keep pulling out your phone to turn on device

 gpio.trig(3, "down",function (level)

    local PinValue=gpio.read(4)

    --Change the state

    if (PinValue==0) then

      --The read resets the output to 0, put it back

      gpio.write(4,1)

      print("Light was on, turn off")

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","OFF",0,0)

    else

      gpio.write(4,0)

      print("Light was off, turn on")

      m:publish("/home/".. RoomID .."/" .. DeviceID .. "/p1/state","ON",0,0)

     end

 end)

# [OpenHAB, MQTT, Arduino and ESP8266: Part 5: Hardware – Sensors](https://openhardwarecoza.wordpress.com/2015/03/30/openhab-mqtt-arduino-and-esp8266-part-4-hardware-sensors/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

This one, is quite easy – yet had one twist that had me pulling my hair out.

Lets dive right in again with a Arduino based example:  A simple MQTT Thermometer

Have a look at <http://www.instructables.com/id/ARDUINO-TEMPERATURE-SENSOR-LM35/> for how to wire a LM35 temperature sensor to an Arduino

The following is a mashup between our previous sketch (MQTT Switches), and the LM35 Serial sketch linked to above:

float temp;

int tempPin = 0;

#include <SPI.h>

#include <Ethernet.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal\_I2C.h>

// Update these with values suitable for your network.

byte mac[]    = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };

byte server[] = { 192, 168, 0, 100 };

byte ip[]     = { 192, 168, 0, 120 };

EthernetClient ethClient;

PubSubClient client(server, 1883, callback, ethClient);

// callback function not used really since we are not subscribing to anything in this example

void callback(char\* topic, byte\* payload, unsigned int length) {

Serial.println("Callback");

Serial.print("TopicSerial.println(topic);

Serial.print("LengthSerial.println(length);

Serial.print("PayloadSerial.write(payload,length);

Serial.println();

}

void setup()

{

  Serial.begin(9600);

  Ethernet.begin(mac, ip);

if (client.connect("arduinoClient")) {

    Serial.print("MTQQected");

    // Send initial state

    client.publish("/home/1/ard2/s1/state","0.00");

    }

}

void loop()

{

  client.loop();

  temp = analogRead(tempPin);

  temp = temp \* 0.48828125;

  Serial.print("TEMPRATURE;

  Serial.print(temp);

  Serial.print("\*C Serial.println();

  char temperature[10];

  dtostrf(temp,4,3,temperature);

  client.publish("/home/1/ard2/s1/state",temperature);

  delay(5000);

}

Easy enough, does a Publish to MQTT:/home/1/ard2/s1/state containing the current temperature.

Configuring this into OpenHab is quite easy too… except…

If you look at the examples for adding Switches to OpenHab – you have a Switch item in the Sitemap, and a Switch item in the Items file… easy enough?

This one had me scratching my head though.  I knew that I had to create a Number item in OpenHAB’s Item file, and this was easy enough and in no time at all I was seeing my published temperatures updating the item every 5 seconds  (you can tail -f /var/log/openhab/events.log)

Extract of the log:

2015-03-30 10:10:07 - Office\_temp state updated to 30.762

2015-03-30 10:10:12 - Office\_temp state updated to 30.762

2015-03-30 10:10:17 - Office\_temp state updated to 30.762

2015-03-30 10:10:22 - Office\_temp state updated to 30.762

2015-03-30 10:10:27 - Office\_temp state updated to 30.762

2015-03-30 10:10:32 - Office\_temp state updated to 30.762

2015-03-30 10:10:37 - Office\_temp state updated to 30.762

But! I also added a Number item to the Sitemap – but NOTHING was displayed when I viewed the sitemap/UI

Turns out (and I couldn’t find any documentation stating that, just a noticeable lack of how to setup a Number object in the sitemap and examples) but that you actually create a Text Item in the sitemap, pointed at the name of the Number object in your Items configuration!   Phew…

Here’s the configs to add to get your Arduino+Ethernet Shield+LM35 with the sketch above, into OpenHAB

sudo vi /etc/openhab/configurations/sitemaps/dolphin.sitemap

And add the Text line

sitemap demo label="Main Menu"

{

Frame label="MQTT" {

Switch item=mqttsw1 label="MQTT Switch 1"

Switch item=mqttsw2 label="MQTT Switch 2"

Switch item=lamp1 label="Office Lamp"

Text item=Office\_temp

}

}

Next

sudo vi /etc/openhab/configurations/items/dolphin.items

Add the Numbers item:

Group All

Switch mqttsw1 "Switch 1" (all) {mqtt=">[broker:/testsw/1:command:on:1],>[broker:/testsw/1:command:off:0]"}

Switch mqttsw2 "Switch 2" (all) {mqtt=">[broker:/testsw/2:command:off:0],>[broker:/testsw/2:command:on:1]"}

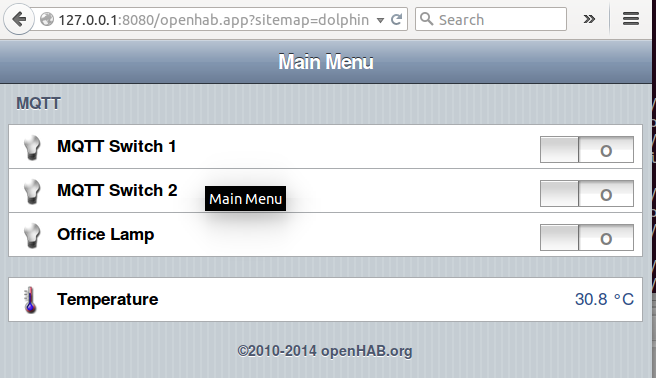
Switch lamp1 "Office Lamp" (all) {mqtt=">[broker:home/openHAB/out/Light\_GF\_Office\_Lamp/command:command:on:ON],>[broker:home/openHAB/out/Light\_GF\_Office\_Lamp/command:command:off:OFF],<[broker:/openHAB/in/Light\_GF\_Office\_Lamp/state:state:default"}

Number Office\_temp "Temperature [%.1f °C]" <temperature> {mqtt="<[broker:/home/1/ard2/s1/state:state:default]"}

Save, exit and restart OpenHab

sudo /etc/init.d/openhab restart

You should see the Temperature item in the UI, and a few seconds later (or as soon as the Arduino sends a MQTT message containing a temperature) it should display the temperature too!

[](https://openhardwarecoza.files.wordpress.com/2015/03/screenshot-from-2015-03-30-101915.png)

[**OpenHAB, MQTT, Arduino and ESP8266: Part 5.1: Graphing Sensor Data**](https://openhardwarecoza.wordpress.com/2015/03/30/openhab-mqtt-arduino-and-esp8266-part-5-1-graphing-sensor-data/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
* Part 6:  OpenHAB Automation Rules

So now that we have a pretty basic “remote control” system – not quite Home Automation yet, but pretty nifty already – you should have enough knowledge to add several switches and sensors at the very least. Keep in mind any value you could have an Arduino find and publish to MQTT can be used, whether its an analogRead() or a simple ON/OFF status message, anything can be used.

OpenHAB comes with a functionality called Persistence, which basically means storing values/states in a database.   This could be as complicated a MySQL, or – the way I prefer for now – a simple RRD deployment.   RRD is used in a LOT open source project to provide graphing – the first time I encountered RRD was on PfSense’s traffic logs for example (:

To configure RRD4J as a Persistance add-on for OpenHAB is extremely simple!

1.  Install the rrd4j add-on:

sudo apt-get install openhab-addon-persistence-rrd4j

2.  Edit the Persistence file:

sudo vi /etc/openhab/configurations/persistence/rrd4j.persist

Add:

/ Configuration file for "rrd4j" persistence module

// persistence strategies have a name and a definition and are referred to in the "Items" section

Strategies {

// for rrd charts, we need a cron strategy

everyMinute : "0 \* \* \* \* ?"

}

Items {

// let's store EVERYTHING - we may need it later (:

\* : strategy = everyMinute

}

You could also choose to only share specific states:

/ Configuration file for "rrd4j" persistence module

// persistence strategies have a name and a definition and are referred to in the "Items" section

Strategies {

// for rrd charts, we need a cron strategy

everyMinute : "0 \* \* \* \* ?"

}

Items {

// let's only store temperature values in rrd

Office\_temp : strategy = everyMinute

}

3.  Add a chart to your sitemap:

Notes:  You can change the Period to h,D,W,M,Y etc (Hour, Day, Week, Month, Year

sitemap dolphin label="Main Menu"

{

        Frame label="MQTT" {

        Switch item=mqttsw1 label="MQTT Switch 1"

        Switch item=mqttsw2 label="MQTT Switch 2"

        Switch item=lamp1 label="Office Lamp"

        Text item=Office\_temp

        }

        Frame label="Temperature Graph" {

        Chart label="Last Hour Temp" item=Office\_temp period=h refresh=30000

        }

}

Save, exit and restart OpenHAB

sudo /etc/init.d/openhab restart

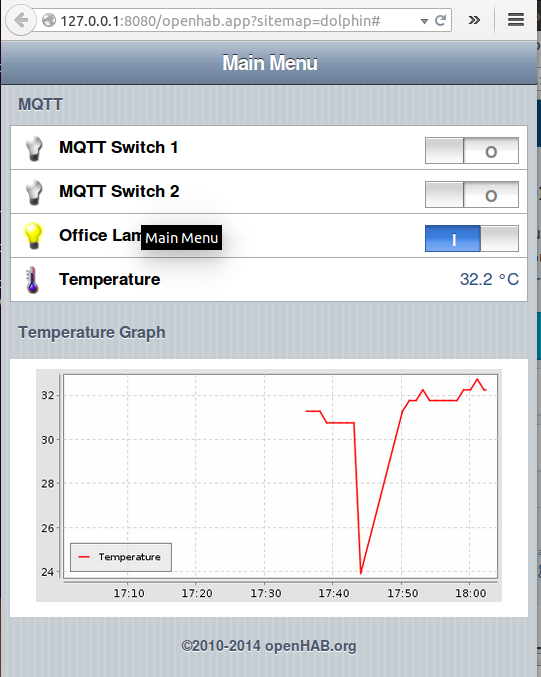
Give it a minute or two, and you should see it has created a RRD database for your items:

peter@www:/$ cd /usr/share/openhab/etc/rrd4j

peter@www:/usr/share/openhab/etc/rrd4j$ ls

Office\_temp.rrd

Next, give it 3-4 minutes and test your Sitemap:

[](https://openhardwarecoza.files.wordpress.com/2015/03/tempgraph.png)

Easy?   Definately the easiest way I have ever used to map a value (:

To recap how this works:

1. Arduino with MQTT Publishes temperature to a MQTT Broker
2. OpenHAB pulls this value into a “Number” Item’s state from a configuration in your .items file
3. We display the current value in a Text item in the .sitemap
4. We enable the RRD4J module by apt-get installing it, and then placing a configuration in /etc/openhab/configurations/persistance/rrd4j.persist
5. It creates .rrd files in /usr/share/openhab/etc/rrd4j/
6. We can access the graph from a Graph entry in the sitemap

[**OpenHAB, MQTT, Arduino and ESP8266: Part 5.2: Combined Switch and Sensor module based on Arduino**](https://openhardwarecoza.wordpress.com/2015/04/05/openhab-mqtt-arduino-and-esp8266-part-5-2-combined-switch-and-sensor-module-based-on-arduino/)

The following series of posts will document my journey to full Home Automation using the MQTT protocol, all custom hardware (Arduino and ESP8266 based), and all tied together using OpenHAB

* Part 1:  Setting up the server / environment
* Part 2:  Publish, Subscribe, Command, State, and WTFs
* Part 3:  Hardware:  Arduino with Ethernet Shield
* Part 4:  Hardware:  ESP8266 with NodeMCU firmware
* Part 5:  Hardware:  Sensors
  + Part 5.1:  Graphing Sensor Data
  + Part 5.2:  Combined Switch and Sensor
* Part 6:  OpenHAB Automation Rules

This is a very quick post just to jot down a very simple mashup of Part 3 and Part 5.  The sketch below listens for MQTT messages to switch, at every second, but only sends temperature updates every 5 seconds…  Good tradeoff = always available to switch immediately, yet still low bandwidth since it doesnt send a temp update on each cycle of the loop:

Another twist here is that I added a physical button on D7 that also turns the device on/off and then publishes the state back to MQTT -> openhab so the virtual switch status matches what the physical is doing

#include <SPI.h>

#include <Ethernet.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <LCD.h>

#include <LiquidCrystal\_I2C.h>

float temp;

int tempPin = 0;

unsigned long interval=5000;  // the time we need to wait

unsigned long previousMillis=0; // millis() returns an unsigned long.

int buttonPin = 7;

boolean currentState = LOW;//stroage for current button state

boolean lastState = LOW;//storage for last button state

boolean ledState = LOW;//storage for the current state of the LED (off/on)

// Update these with values suitable for your network.

byte mac[]    = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };

byte server[] = { 192, 168, 0, 100 };

byte ip[]     = { 192, 168, 0, 120 };

EthernetClient ethClient;

PubSubClient client(server, 1883, callback, ethClient);

LiquidCrystal\_I2C    lcd(0x27,2,1,0,4,5,6,7); // 0x27 is the I2C bus address for an unmodified backpack

byte bulb[8] = {

    0b01110,

    0b10001,

    0b10001,

    0b10001,

    0b01110,

    0b01010,

    0b01110,

    0b00100

};

byte plug[8] = {

    0b01010,

    0b01010,

    0b11111,

    0b10001,

    0b10001,

    0b01110,

    0b00100,

    0b00100

};

void callback(char\* topic, byte\* payload, unsigned int length) {

Serial.println("Callback");

Serial.print("TopicSerial.println(topic);

Serial.print("LengthSerial.println(length);

Serial.print("PayloadSerial.write(payload,length);

Serial.println();

  if (strcmp(topic,"/testsw/1")==0) {

   if (payload[0] == '0')

    {

    digitalWrite(4, LOW);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Fan OFF");

    delay(100);

    client.publish("/testsw/1/state","0");

    }

    else if (payload[0] == '1')

    {

    digitalWrite(4, HIGH);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Fan ON");

    delay(100);

    client.publish("/testsw/1/state","1");

    }

   }

  if (strcmp(topic,"/testsw/2")==0) {

   if (payload[0] == '0')

    {

    digitalWrite(5, LOW);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Light OFF");

    delay(100);

    client.publish("/testsw/2/state","0");

    }

    else if (payload[0] == '1')

    {

    digitalWrite(5, HIGH);

    lcd.home();

    lcd.clear();

    lcd.write((uint8\_t)1);

    lcd.print ("Turning Light ON");

    delay(100);

    client.publish("/testsw/2/state","1");

    }

   }

}

void setup()

{

  Serial.begin(9600);

  pinMode(buttonPin, INPUT);//this time we will set the pin as INPUT

  digitalWrite(buttonPin, HIGH);

  lcd.begin (20,4); // for 16 x 2 LCD module

  lcd.createChar(0, bulb);

  lcd.createChar(1, plug);

  lcd.home(); // set cursor to 0,0

  lcd.print(" Starting  ");

  delay(1000);

  Ethernet.begin(mac, ip);

    lcd.home();

    lcd.clear();

    lcd.print("Ethernetected");

  if (client.connect("arduinoClient")) {

    lcd.home();

    lcd.clear();

    lcd.print("MTQQected");

    client.publish("outTopic","hello world");

    client.subscribe("/testsw/#");

  }

  pinMode(4, OUTPUT);

  digitalWrite(4, LOW);

  pinMode(5, OUTPUT);

  digitalWrite(5, LOW);

}

void loop()

{

  client.loop();

   if ((unsigned long)(millis() - previousMillis) >= interval) {

      previousMillis = millis();

      temp = analogRead(tempPin);

      temp = temp \* 0.48828125;

      Serial.print("TEMPERATURE;

      Serial.print(temp);

      Serial.print("\*C     Serial.println();

      char temperature[10];

      dtostrf(temp,4,3,temperature);

      client.publish("/home/1/ard2/s1/state",temperature);

   }

  currentState = digitalRead(buttonPin);

  if (currentState == HIGH && lastState == LOW){//if button has just been pressed

    Serial.println("pressed");

    delay(10);//crude form of button debouncing

    //toggle the state of the LED

    if (ledState == HIGH){

      digitalWrite(4, LOW);

      ledState = LOW;

      client.publish("/testsw/1/state","0");

    } else {

      digitalWrite(4, HIGH);

      ledState = HIGH;

       client.publish("/testsw/1/state","1");

    }

  }

  lastState = currentState;

}