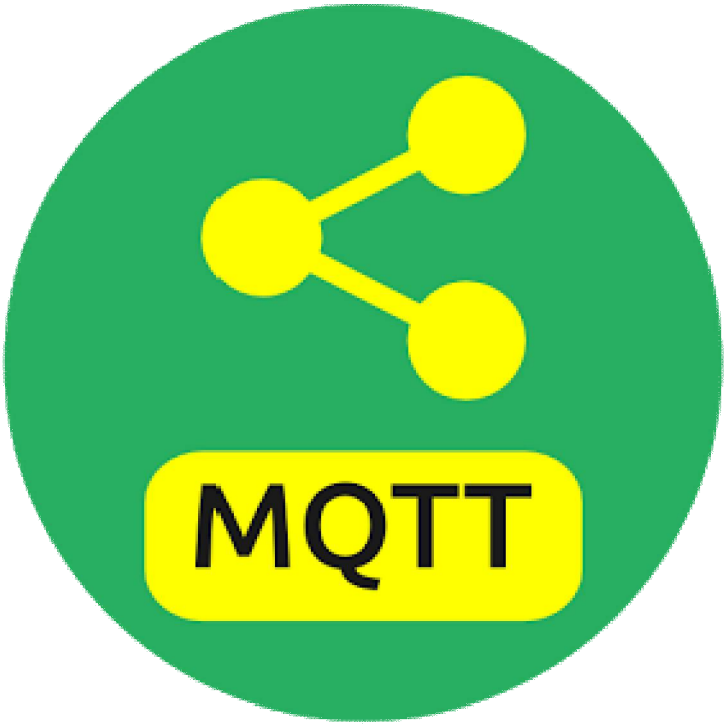
Getting Started With

MQTT
A Practical Guide



By: Satya Sankar Sahoo

Who Should Read This eBook?

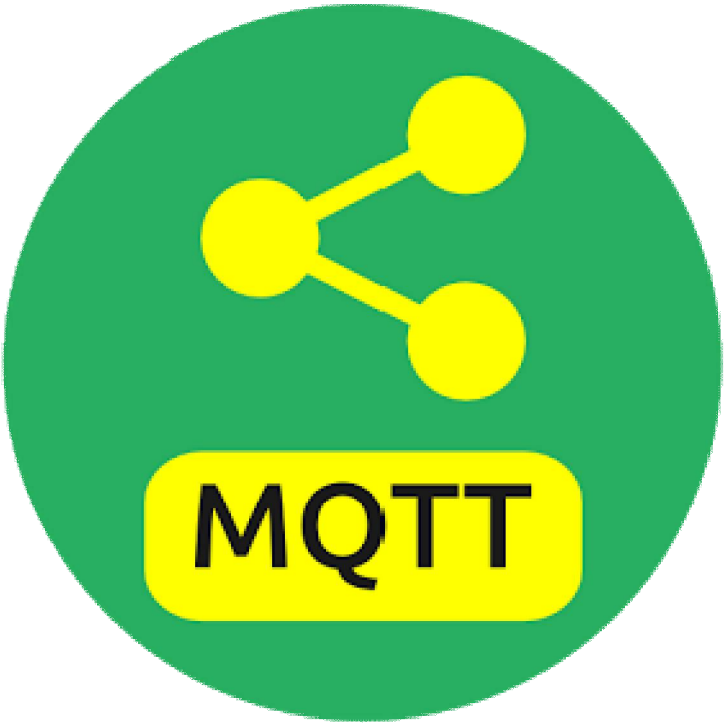
I believe this eBook is perfect for anyone that loves electronics, open-source hardware, home automation , IOT, and the MQTT. This eBook is perfect for:

- Beginners – If you're just starting and wondering about MQTT , this eBook is perfect for you, because I'll start from the very beginning and you'll learn everything you need to know
- Makers/DIYers – If you love making things, this eBook is perfect for you. You'll design several circuits and projects
- Hobbyists – If making things is your hobby and you like to make projects in the weekends or late at night. This eBook is perfect for worth for you to getting started with MQTT.

What You'll Learn In This eBook?

Getting Started With

MQTT
A Practical Guide



Introduction

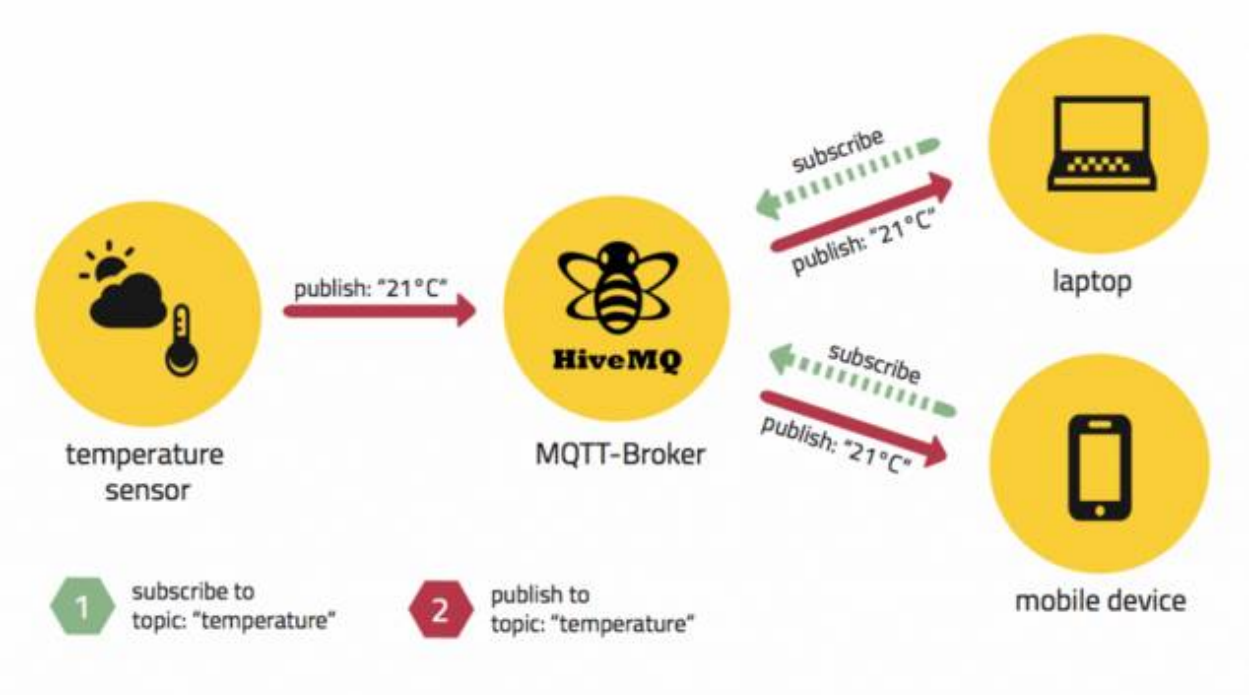
MQTT is a machine-to-machine (M2M)/"Internet of Things" connectivity protocol. It was designed as an extremely lightweight publish/subscribe messaging transport. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. For example, it has been used in sensors communicating to a broker via satellite link, over occasional dial-up connections with healthcare providers, and in a range of home automation and small device scenarios. It is also ideal for mobile applications because of its small size, low power usage, minimised data packets, and efficient distribution of information to one or many receivers.

MQTT was first developed in 1999, but with the exponential growth of the Internet of Things, and the need to connect and communicate between low-powered smart devices, MQTT has recently found a market. MQTT was built to be a low-overhead protocol that strongly considered bandwidth and CPU limitations. It was designed with ability to run in an embedded environment where it would reliably and effectively provide an avenue for communication.

The MQTT design makes it very appealing for the exponential emerging Internet of Things market. As a user, if you are looking for a very small publish/subscribe protocol, MQTT may be for you. If you need more features, or need to scale to hundreds or thousands of connected devices

How It Works

The protocol uses a publish/subscribe architecture in contrast to HTTP with its request/response paradigm.Publish/Subscribe is event-driven and enables messages to be pushed to clients. The central communication point is the MQTT broker, it is in charge of dispatching all messages between the senders and the rightful receivers. Each client that publishes a message to the broker, includes a topic into the message. The topic is the routing information for the broker. Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with the matching topic to the client. Therefore the clients don’t have to know each other, they only communicate over the topic. This architecture enables highly scalable solutions without dependencies between the data producers and the data consumers.



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

Use Case

In order to make the subsequent code more understandable, we will use the transferring of sensor data from a temperature and brightness sensor to a control center over the internet as an example. The sensors will be connected to a Raspberry Pi, which acts as gateway to the MQTT broker, which resides in the cloud. On the other side is a second device, the control center, that also has an MQTT client and receives the data. Additionally we will implement a notification, which alerts the control center if the sensor is disconnected.

NOTE: Pub/Sub decouples a client, who is sending a particular message (called publisher) from another client (or more clients), who is receiving the message (called subscriber).

Client

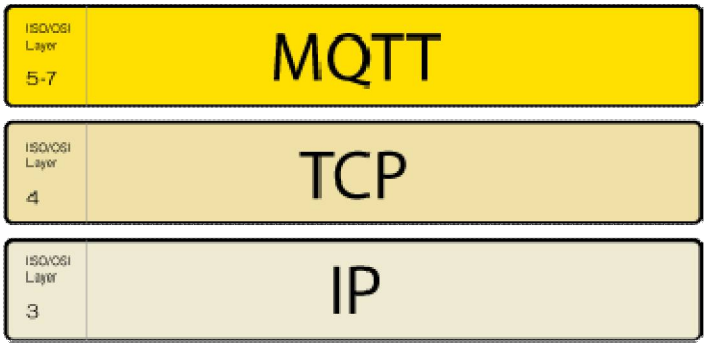
When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). A MQTT client is any device from a micro controller up to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device, that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it. The client implementation of the MQTT protocol is very straight-forward and really reduced to the essence. That's one aspect, why MQTT is ideally suitable for small devices. MQTT client libraries are available for a huge variety of programming languages, for example Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET. A complete list can be found on the MQTT wiki.

Broker

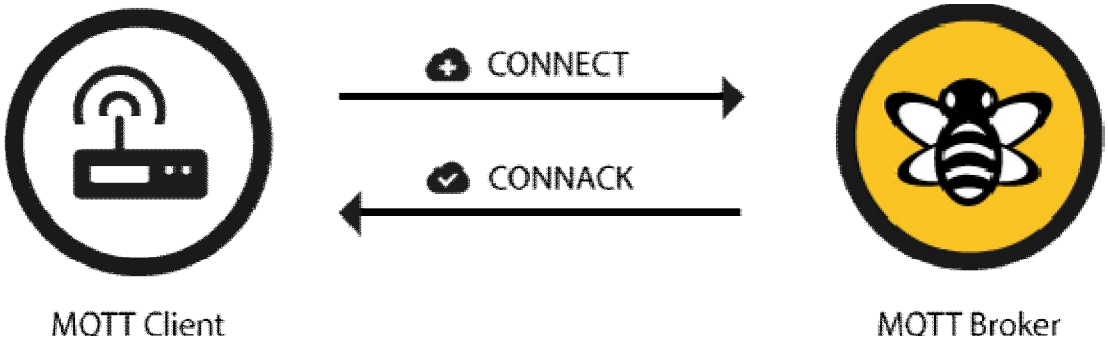
The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. It also holds the session of all persisted clients including subscriptions and missed messages (More details). Another responsibility

of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. As we described in one of our early blog post subscribing to all message is not really an option. All in all the broker is the central hub, which every message needs to pass. Therefore it is important, that it is highly scalable, integratable into backend systems, easy to monitor and of course failure-resistant. For example HiveMQ solves this challenges by using state-of-the-art event driven network processing, an open plugin system and standard providers for monitoring.

MQTT Connection



The MQTT protocol is based on top of TCP/IP and both client and broker need to have a TCP/IP stack.



The MQTT connection itself is always between one client and the broker, no client is connected to another client directly. The connection is initiated through a client sending a CONNECT message to the broker. The broker response with a CONNACK and a status code. Once the connection is initiated MQTT will keep it open as long as the client doesn't send a disconnect command or it looses the connection .

Understanding MQTT By Practical Example

in order to understand it better, let's do practical for more clarity about MQTT. There is a free server, http://test.mosquitto.org/ where you can test your first MQTT message. The default port for MQTT is 1883, but this is not encrypted(Not Secure), So don't use for commercial purpose.

What You Need:

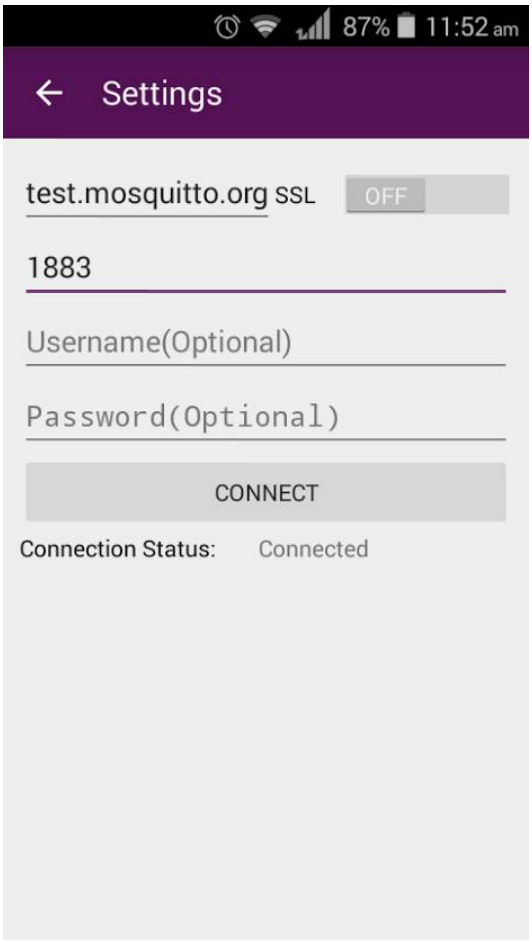
1. Two android Phone or Two PC with Internet Connection

Testing Your MQTT message in Android Phone:

Step 1: Download an application named "MQTT Client" on both android Phone from this below link

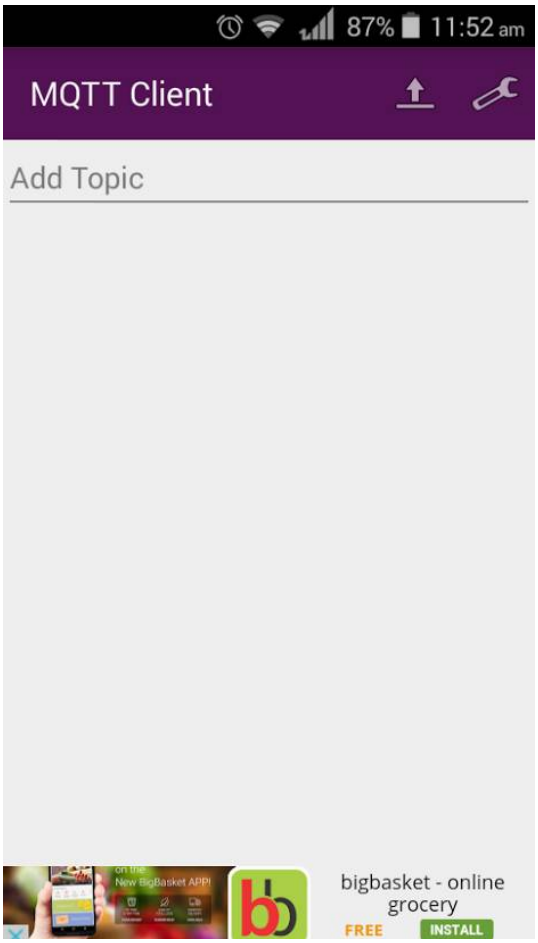
https://play.google.com/store/apps/details?id=com.deepeshc.mqttrec

Step 2: Open application and Go to settings



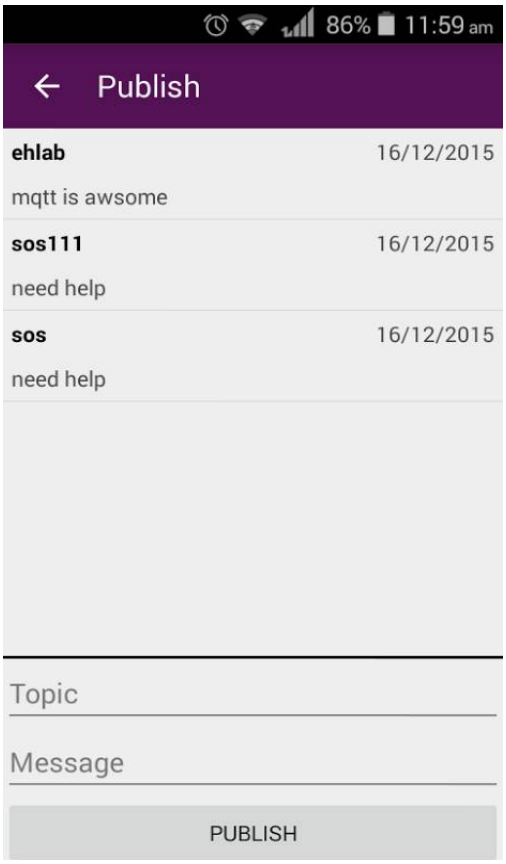
a. In First text box enter URL of MQTT Server/Broker that is test.mosquitto.org
b. Second Text box is for Port, type 1883
c. You have to left username and password blank.
d. Press Connect
e. Do the same thing other phone.

Step 3: Now you have to create a TOPIC in Phone 1

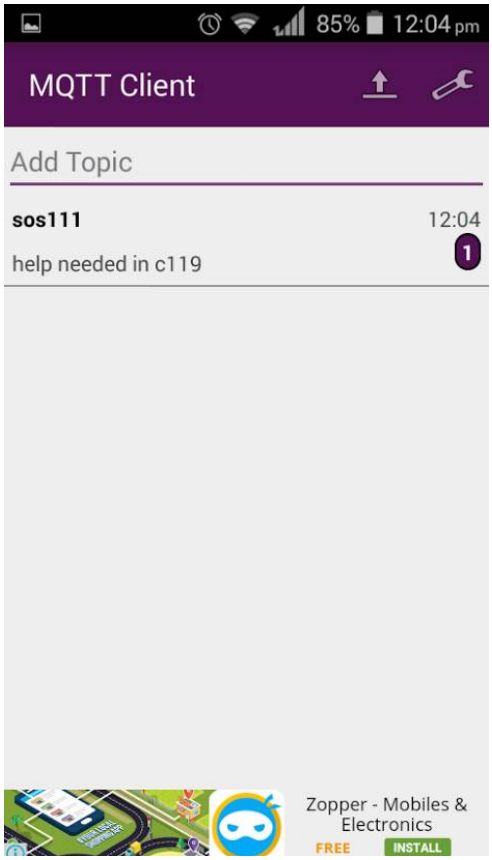


In the top Right corner of application , You can see a upload button symbol near setting button, press on that. You will see a page like below.

Fact: Facebook uses MQTT for its Chat Messenger.



(Create Topic On Phone 1)



(Subscribe to a Topic on Phone 2)

In the above image, you can see there are Two textbox One is Topic, and Other is message. Phone 1 Needs to create a Topic, and Phone 2 should Subscribe a topic.

Let's say I create a Topic Called "SOS111" and Message is "Need Help" from Phone 1, and from other Phone I subscribe to SOS111, then I will get the message "Need Help" . Is that not Cool.

To subscribe to the topic on Phone 2, You need to go to Page 1 of application and in "Add Topic" Text box , you have to type the topic name in phone 1, which is "SOS111" like shown in above screenshot.

Setup Your Own MQTT Server/Broker

You can setup your own MQTT server/Broker on any Cloud Server or on a local machine or everyone's favourite Raspberry PI. In this book , we will learn how to setup a local MQTT server at our Home on Raspberry Pi.

You Need:

a. Raspberry Pi 2
b. Internet Connection
c. 30 Min Time

Step 1:

We need to install few dependencies before we compile and run our own MQTT server.

a. sudo apt-get update
b. sudo apt-get install libssl-dev
c. sudo apt-get install cmake
d. sudo apt-get install libc-ares-dev
e. sudo apt-get install uuid-dev
f. sudo apt-get install daemon

Step 2:

Then we need to download and compile libwebsockets To download use below command

wget http://git.libwebsockets.org/cgi-bin/cgit/libwebsockets/snapshot/libwebsockets-1.4-chrome43-firefox-36.tar.gz

Unpack using following command

*tar zxvf libwebsockets**

then change directory to using: *cd libwebsockets**

Make a build directory inside libwebsockets using bellow command

mkdir build

cd build

Then to build

cmake .. #(note the ..)

Then run following command to install

sudo make install

Then we need to rebuild the library cache

sudo ldconfig

change the directory to Home by command

cd

Step 3:

Then we need to download the source code of MQTT broker, which is Mosquitto using following command

wget http://mosquitto.org/files/source/mosquitto-1.4.1.tar.gz

unpack using following command

tar zxvf mosquitto-1.4.1.tar.gz

cd mosquitto-1.4.1

open and edit config.mk in your favourite editor

change the line “WITH_WEBSOCKETS:=no” to “WITH_WEBSOCKETS:=yes”

Then compile using bellow command

make

Then to install use below command

sudo make install

Then we need to create a config directory inside /etc

sudo mkdir /etc/mosquitto

copy default config file to /etc

sudo cp mosquitto.conf /etc/mosquitto

Add the following two lines to /etc/mosquitto/mosquitto.conf at the end of file

listener 9001

protocol websockets

then add a user for mosquitto using following command

sudo adduser mosquitto

Then reboot and login user as mosquitto

after successful login run below command

mosquitto -v

Congrats, your first MQTT server is up and running. You can do same communication using any MQTT client that you did with your android Phone last time in previous section.

Home Automation Project using MQTT

Objective:

The main objective of project to learn MQTT using it in Home automation using ESP8266. If you want video Demo of project, you can find it below.

https://www.youtube.com/watch?v=86DtfijqJ6M

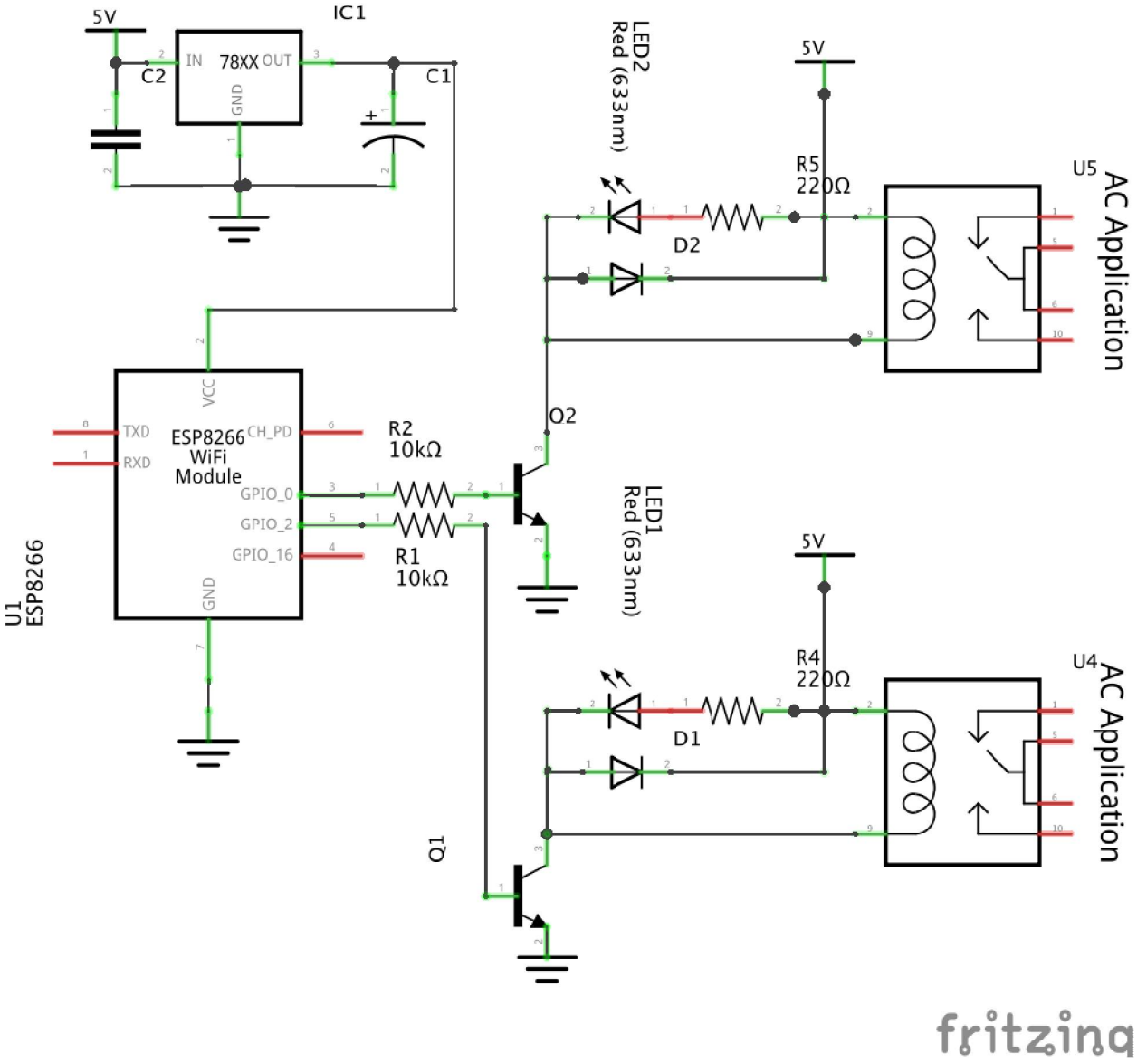
What we will make:

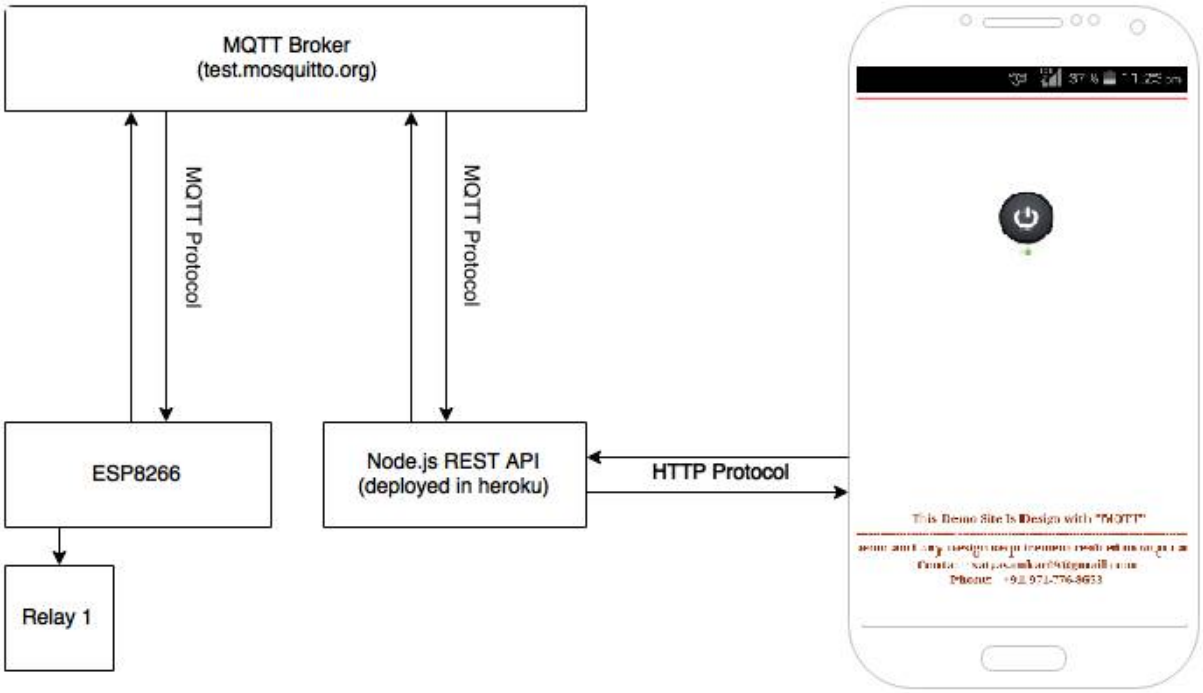
we will make a project, by using which we can control one relay any where from world using a browser of a mobile application.

What You Need:

a. ESP8266 (we have used ESP-12)
b. General Purpose PCB
c. Jumper wire
d. Soldering IRON
e. Multimeter
f. PC/Laptop/Internet Connection
g. An account in Heroku.com

Electrical Connection:


5V
IC1
C2
IN
78XX
OUT
GND
C1
LED2
Red (633nm)
5V
R5
220Ω
U5
AC Application
D2
VCC
TXD
RXD
ESP8266
WiFi
Module
CH_PD
GPIO_0
GPIO_2
GPIO_16
GND
R2
10kΩ
O2
U1
ESP8266
LED1
Red (633nm)
5V
R4
220Ω
U4
AC Application
D1
Q1
fritzing



The Communication Diagram is shown in above Image. This article shows you how to make your own REST API server for the ESP8266 and your Mobile App to communicate to, via MQTT. You can use the existing REST API server that is deployed in heroku(http://mqtt-ehlabs-demo.herokuapp.com) or you can deploy your own. If you don't want to use "test.mosquitto.org" as your MQTT Broker as stated in previous chapter.

After assembling the SCH shown above you need to download the code to ESP8266. You can download the source code from below link.

https://github.com/embeddedhacks/esp8266-relay

To compile the code you need to install and ESP8266 Arduino library and PusbSUBClient Library which can be downloaded from bellow

https://github.com/embeddedhacks/pubsubclient

https://github.com/embeddedhacks/Arduino

Once you successfully installed library and compiled download the code to ESP8266 and You are ready to go.

```
#include                                                <PubSubClient.h>
#include                                                 <ESP8266WiFi.h>

const          char*          ssid          =          "your-wifi-ssid";
const        char*        password        =        "your-wifi-passwd";

char* topic = "device/control";
```

ESP8266 Arduino Code

This code block connects the ESP8266 to the wifi network by specifying the SSID and password. This block also establishes a connection with the MQTT broker.

```
char*  server  =  "85.119.83.194";  //  IP  of  test.mosquitto.org

WiFiClient                                                  wifiClient;
PubSubClient    client(server,    1883,    callback,    wifiClient);
.....
```

Now you need to make rest api based MQTT client You can host rest api on heroku server.

The link for source code

https://github.com/embeddedhacks/MQTT-REST-API

You need to create a account in rest api and host the code in above link to your heroku account like I have hosted my server on

http://mqtt-ehlabs-demo.heroku.com/

You can create your own server locally or by deploying it in the cloud. This node application requires two modules, Hapi.js is a REST API framework while MQTT.js is an mqtt client.

This block initializes the REST API server, as well as establishes a connection with the MQTT broker.

```
var Hapi = require('hapi');
var mqtt = require('mqtt');

var server = new Hapi.Server();
var port = Number(process.env.PORT || 4444);

server.connection({ port: port, routes: { cors: true } });

var client = mqtt.connect('mqtt://test.mosquitto.org:1883');
```

This function is for publishing messages to the broker.

```
var mqttPublish = function(topic, msg){
  client.publish(topic, msg, function() {
    console.log('msg sent: ' + msg);
  });
}
```

This block creates a route for the '/device/control' POST Method, wherein if this route is being called it will execute the mqttPublish function. The 'deviceInfo' variable contains the message for the ESP8266 to translate, wherein 'dev1-on' turns the relay 1 on, 'dev1-off' off and same thing for 'dev2-on' and 'dev2-off'.

```
server.route([
  {
    method: 'POST',
    path: '/device/control',
    handler: function (request, reply) {
      var deviceInfo = 'dev' + request.payload.deviceNum + '-' +
```

```
request.payload.command;
      reply(deviceInfo);
      mqttPublish('device/control',          deviceInfo,          {
        'qos'                              :                      2
      });
    }
  }
]);

server.start();
```

lThe first parameter in the mqttPublish function 'device/control' is the topic wherein our ESP8266 listens to, then the second parameter is where the message 'deviceInfo' is being passed to. The 'qos' means quality of service. It is a level of agreement between sender and receiver of a message regarding the guarantees of delivering a message.

Your server part is ready. Now you need a HTTP client which can make post request to your server.

for that you can use different website or HTTP client. But I have made my own for this project.

ike you see on www.ehlabs.org/mqtt

the souce cod of this demo can be downloaded from here

https://www.dropbox.com/sh/dogug35rzxgvkif/AAAoE4HZiqovA__GhtEmCSDea?dl=0

if you open index.html in notepad you can see, I have doing a post request to my rest api hosted on heroku. like shown in bellow

```
//Function to ON switch
function onSwitch(){
    //alert("Switch ON the button.");
    var url = "https://mqtt-ehlabs-demo.herokuapp.com/device/control";
    var params = "command=on&deviceNum=1";
    var xhr = new XMLHttpRequest();
    xhr.open("POST", url, true);

    //Send the proper header information along with the request
    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    xhr.send(params);

}
```

I have passed two parameter command=on and deviceNum=1

this two parameter are being parsed by esp8266 to make the relay on and off.

This is Cool :)

"END OF FIRST MQTT PROJECT"

I Hope You got a basic Idea of MQTT, and Its Implementation.

If You have any doubt related to setup or any problem you can ask on bellow link direct to auther.

https://github.com/embeddedhacks/Getting-Started-with-MQTT

Conclusion

This Book is written to help readers getting started with MQTT by doing practical. There is lot of other things that we need to consider while working on MQTT, but that is out of scope of this book. I hope now

you have enough knowledge to start and looking forward to learn more about MQTT of your own from internet or other resource.

Thanks for reading. If you have any doubt you can directly reach Me at Below Links

GitHUB- https://github.com/embeddedhacks/

Facebook: https://www.facebook.com/satyasankarsahoo

IOT Facebook Page : https://www.facebook.com/iot.Projects/

Blog : www.embbsys.blogspot.in

Linkedin: https://in.linkedin.com/in/satyasankarsahoo

Thanks

For Reading :)

By: Satya Sankar Sahoo