**Message Queuing Telemetry Transport Protocol Project**

ECE Department  
Kyle J Cooper

Student ID: 0003304926

kyjcoope@iu.edu

11/28/2018

# Introduction

Message Queuing Telemetry Transport protocol (MQTT) was first made in 1999 but was not made an ISO standard until 2013 (ISO/IEC PRF 20922). MQTT is a messaging protocol designed for connections in remote locations that require limited bandwidth and is built on top of the TCP/IP Protocol. It uses a publisher/subscriber messaging model, where one device acts as a “broker” who will broadcasts to all subscribers everything that is published to the broker. A subscriber will only get updates from the broker about topics they are subscribed to.   
  
The publisher/subscriber model makes MQTT very good at gathering sensor data or for remote devices to send telemetry data. This makes MQTT a very good protocol to enable Internet of Things (IOT) solutions.

# How MQTT works

This section will go over MQTT’s packet structure, and how MQTT is formatted. MQTT uses a command and command acknowledgement format, where each command has an associated acknowledgement.

**MQTT Client MQTT Broker**

Connect🡪

🡨Connect Acknowledgement

Subscribe🡪

🡨Subscribe Acknowledgement

MQTT messages use a format of three control packets. A Fixed header, which is always present, and is 2 bytes long. A Variable header and payload which are sometimes present and have no fixed size, and content is dependent on message type indicated in the fixed header. The total packet size of an MQTT packet is anywhere from 2 bytes to 256MB.

**Fixed Header**

| **bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| byte 1 | Message Type | | | | DUP flag | QoS level | | RETAIN |
| byte 2 | Remaining Length | | | | | | | |

**Message Type:** is what type of message is being sent, *connect acknowledgement*, *publish message*, or *client subscribe request* are all examples of valid message types.

**Duplication Flag:** is set when a client or broke attempts to re-deliver a publish message and requires a QoS level greater than 0.

**QoS Level:** indicates the level of QoS show in the table below.

| **QoS value** | **bit 2** | **bit 1** | **Description** | | |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | At most once | Fire and Forget | <=1 |
| 1 | 0 | 1 | At least once | Acknowledged delivery | >=1 |
| 2 | 1 | 0 | Exactly once | Assured delivery | =1 |
| 3 | 1 | 1 | Reserved | | |

**Retain flag:** indicates that the broker should hold the message and send it as an initial message to new subscribers of this topic.

**Remaining Length:** This is the last byte of the fixed header and is used to indicate the remaining length of the packet, which includes both the variable header and the payload.

**Variable Header**

This contains information about the message type being sent and is different depending on what type of message is being sent. For example, the CONNECT message has a “will” flag so that a client can create a “last will and testament” which is a message the broker can send to subscribers if that client becomes unreachable. If you want to learn more about the variable header and its different types the MQTT manual is available online.

**Payload**

The payload is encoded as UTF-8 strings and contains the user message, as well as any additional information specified in the variable header, such as the clients “will” in a CONNECT message. The only message types that contain a payload are CONNECT, SUBSCRIBE, and SUBACK. Like the variable header the information being sent in the payload is depended on the message type.

# Mosquitto Broker setup

For this project a Mosquitto (yes it has two t’s) server will be used as the Broker for the MQTT protocol and will be run off a Raspberry Pi 3 B+. Mosquitto was picked as the Broker software because it is light-weight and very easy to install and setup. Another benefit to using Mosquitto is that usernames and passwords can be secured using an SSL certificate, which can be setup inside Mosquitto. While I will not be setting up the SSL certificate in this example, it is an important step if you wish to secure your MQTT network, since MQTT does not encrypt usernames or passwords, and only encodes them as UTF-8 strings.

**Setup:** Assuming you already have a Raspberry Pi and its running Rasbrain all you must do to install and setup an MQTT Broker using Moquitto is enter the following commands into your Raspberry Pi.

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install mosquitto

sudo apt-get install mosquitto-clients

This is all you need to do to install and setup MQTT Moquitto Broker with the default settings. By default, MQTT uses port 1883 and will only be available on your local network unless, you have port 1883 forwarded to your MQTT Mosquitto Broker on your external gateway (your router).

You will also need to know your external IP address (google “my IP”), this is most likely a dynamic IP address, and will change over time. To make it so your MQTT Mosquitto Broker is available even if your IP address changes you will need to setup Dynamic DNS on your router if it supports it.

# Paho-MQTT client setup

Paho-MQTT is a python module that is used for subscribing and publishing topics to a MQTT broker. Like Mosquitto, Paho-MQTT is very simple to setup and install, which is why it will be used in this project.

**Setup:** You should have Python 3.6 or later installed. First thing you must do is install paho-mqtt, to do this issue the below command in command prompt.

pip install paho-mqtt

The Python code to setup a MQTT client is show below.

Note: the following code examples are modified versions of  
“https://pypi.org/project/paho-mqtt/” MQTT tutorial.

import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

# Subscribing in on\_connect() means that if we lose the connection and

# reconnect then subscriptions will be renewed.

client.subscribe("test")

# The callback for when a PUBLISH message is received from the server.

def on\_message(client, userdata, msg):

print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect("192.168.254.59", 1883, 60) #connect(hostname, port, timeout)

After connecting with the MQTT Broker, the client can listen for updates from the Broker or publish topics to the Broker.

To listen use,

# Blocking call that processes network traffic, dispatches callbacks and

# handles reconnecting.

# Other loop\*() functions are available that give a threaded interface and a

# manual interface.

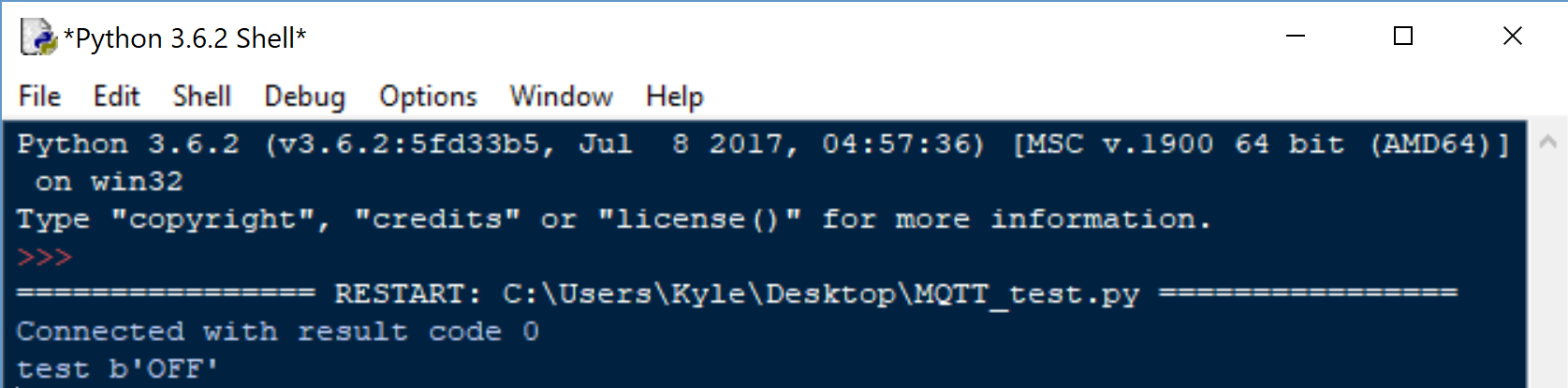
client.loop\_forever()

To publish use,

client.publish(“topic”, “payload”)

#example, client.publish(“test”, “OFF”)

Example result,



Any client connected to my Mosquitto Broker at 192.168.254.59 and subscribed to the test topic will be updated that test’s value is now “OFF” by the broker.

# Conclusion

MQTT is a powerful tool that is used by many IoT (internet of things) software to run backend communication between devices. While MQTT is very powerful, it is not without its disadvantages. The fact that MQTT relies on a Broker server to communicate between devices means that scalability is limited by the amount of traffic the Broker can support. Also since MQTT does not come with any built in encryption to protect against unauthorized publishing and subscribing, you have to rely on third party software to implement encryption security.

# References

C. Hillar, G. (2018). *Hands-On MQTT Programming with Python. [electronic resource]: Work with the*

*Lightweight IoT Protocol in Python*. Birmingham : Packt Publishing, Limited May 2018. Retrieved

fromhttp://www.ulib.iupui.edu/cgibin/proxy.pl?url=https://search.ebscohost.com/login.aspx?direct=true&db=cat06315a&AN=iui.17199664&site=eds-live

Jun-Hong Park, Hyeong-Su Kim, & Won-Tae Kim. (2018). DM-MQTT: An Efficient MQTT Based on SDN

Multicast for Massive IoT Communications. *Sensors, Vol 18, Iss 9, p 3071 (2018)*, (9), 3071.

<https://doi.org/10.3390/s18093071>

20922:2016, I. (2018). *ISO/IEC 20922:2016*. [online] ISO. Available at:

https://www.iso.org/standard/69466.html [Accessed 25 Nov. 2018].

MQTT Version 3.1.1. (2018). Retrieved from http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-

os.html

paho-mqtt. (2018). Retrieved from https://pypi.org/project/paho-mqtt/