AARHUS UNIVERSITY

Internet of Things Technology
Project 10 - Internet of Things Weather Station

7 December 2018

Ong Jing Yin (201800935)
201800935@uni.au.dk

Tiong Yaocong (201801239)
201801239@uni.au.dk

Zhou Zhuyun(201801015)
201801015@uni.au.dk

1. Introduction

Weather affects a wide range of activities - agriculture, transportation and leisure. There are many useful applications of weather monitoring, such as predicting the weather for air traffic in the aviation industry, giving crop warnings for farmers, and forecasting natural disasters like cyclones and hurricanes. Weather stations keep track of different climatic components such as temperature, pressure, humidity and light for these purposes.

The main concept behind IoT is to connect various devices through a network and retrieve or distribute data from the devices as required. IoT in weather monitoring will enable wireless transfer of information or data over a distance from devices (sensors) to a cloud service, which can then be gathered for analysis and then used to alert people or devices that subscribe to the service. The integration of IoT and weather stations thus bring the benefits of scalability and flexibility.

For this project, we have implemented a weather monitoring station with a Sense Hat sensor that monitors temperature, pressure, humidity and light, which transmits the data captured to a Raspberry Pi 3 that it is connected to. We are using the CloudMQTT server as our cloud service, and the MQTT protocol between Pi to cloud and cloud to clients. We will explain in detail the protocol used in our IoT weather station and the hardware of the sensor (Section 2). The Raspberry Pi is connected to the internet which transmits data from the sensor to the cloud (Section 3), and we monitor the environmental changes in the sensor readings from clients subscribed to the weather topics. Our experiment is set up indoors (Section 4) and changes to climatic conditions are artificially created to test for the expected changes in readings recorded by the sensor. Finally, we conclude that our experimental results confirm the successful deployment of the IoT weather station records and distributes weather data to the respective end-points (Section 5).

2. Background

The project is been implemented using Python 2.7 as most IoT devices are pre-installed with Python 2.7 instead of Python 3. This is to support several projects that had been released to the public. Hence, in order to ensure compatibility of our project with others or future work, we have decided to use Python 2.7 instead of Python 3.

We have used Raspberry Pi equipped with Sense Hat to act as our weather station. Hence, it has the capability to monitor environmental changes from its surrounding which will be further discussed in Section 2.1. Next, we have implemented our weather station with MQTT protocol that allows us to transmit relevant messages to respective clients which will be further discussed in Section 2.2. and 2.3.

2.1. Sense Hat

The Raspberry Pi Sense Hat is attached on top of the Raspberry Pi via the 40 GPIO pins to create an ‘Astro Pi’. The Sense Hat has several integrated circuit based sensors which enables it to detect it's general surroundings. It can be used for many different types of experiments applications. The sensors enable us to read

- Pressure
- Humidity
- Temperature

In this project, we will use Sense Hat as our weather sensor to detect crucial information about its surroundings, such as Temperature, Humidity and Pressure.

2.2. CloudMQTT

CloudMQTT are managed Mosquitto servers in the cloud. Mosquitto executes the MQ Telemetry Transport convention, MQTT, which gives lightweight methods of carrying out messaging using a publish/subscribe message queueing model.

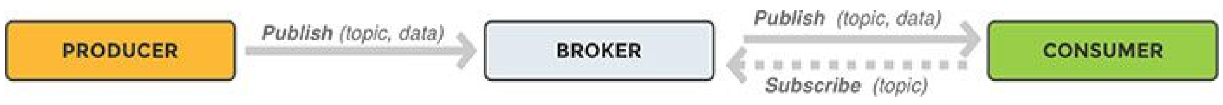


Figure 1: MQTT Protocol

MQTT is the machine-to-machine protocol of the future. It is ideal for the “Internet of Things” world of connected devices. Its minimal design makes it perfect for built-in systems, mobile phones and other memory and bandwidth sensitive applications. Message queues provide an asynchronous communications protocol, the sender and receiver of the message do not need to interact with the message queue at the same time. Messages placed onto the queue are stored until the recipient retrieves them or until the messages times out. MQTT and Mosquitto are for good use by bandwidth sensitive applications. This will be further discussed in Section 2.3.

By using CloudMQTT, it lets us to focus on our application instead of spending time on scaling the broker or patching the platform. Hence, after much deliberation, we decided to choose to use CloudMQTT as our broker for this project.

2.3. MQTT

MQTT is the abbreviation of Message Queuing Telemetry Transport, which is a messaging protocol based on publish-subscribe. A typical MQTT protocol consists of a broker, some publishers and some clients who subscribe to certain topics. The broker is the server and the publishers and clients are able to connect to the broker. Topics are classifications of information that can be subscribed by clients.

The core of this protocol is the MQTT - Broker, who is in charge of communications between publishers and clients. More precisely, publishers do not need information of clients' number or location; conversely, clients do not need to configure with publishers neither. When a publisher has got some new data of a certain topic, it sends a control message with the data to the connected broker. The broker then distributes the newly-received data to the clients and only the clients that have subscribed to that topic.

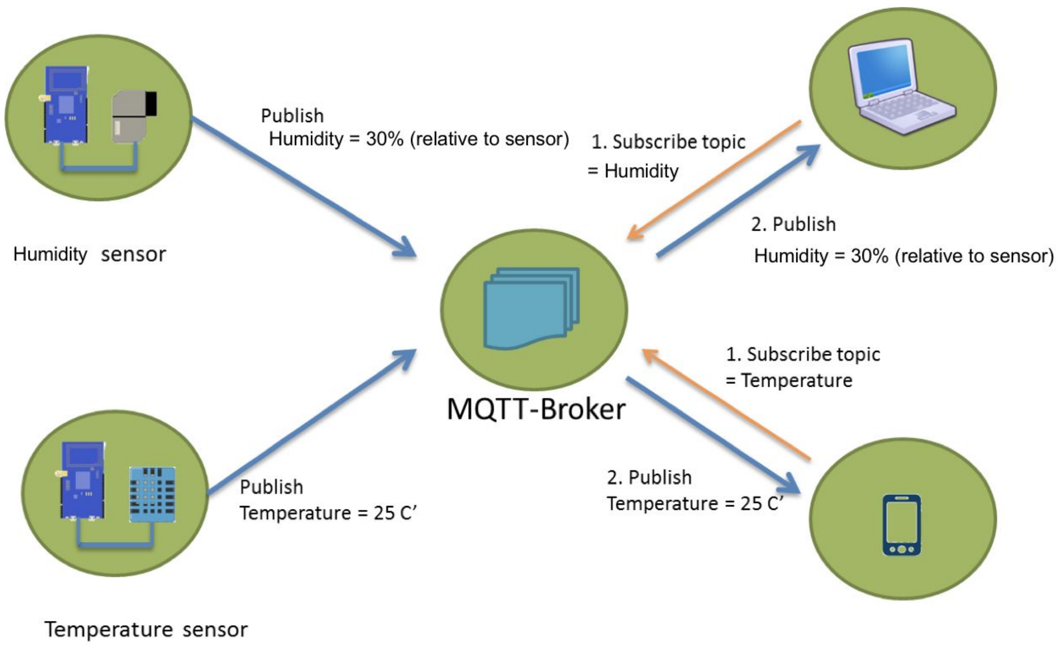


Figure 2: Example of MQTT Protocol

To get into the operating details of MQTT, take the example shown above: There are 2 sensors - humidity sensor and temperature sensor - as publishers. They upload humidity messages and temperature messages to the MQTT - Broker, respectively. We also have got a computer that subscribes to the topic humidity while a mobile phone subscribes to the topic temperature. The main function of MQTT is to transmit the right messages to the right recipients. In this example, the computer will receive and only receive the information of humidity, similar for the mobile phone.

2.4. Setting up the project

2.4.1. Prerequisite

Text editor - Vim (Any text editor will be sufficient)

1. `sudo apt-get install vim`

Python 2.7

1. Download from https://www.python.org/download/releases/2.7/ for respective OS.

2.4.2. Setting up static IP on Raspberry Pi

This needs to be done for the recent Jessie or later update. **/etc/network/interfaces should be left alone**. Open browser and enter the router address (192.168.1.1 for most) and check home network to make sure the Raspberry Pi shows up as 'Static'.

For a static IP address on an Ethernet connection:

1. `sudo vim /etc/dhcpcd.conf`
2. Type in the following lines on the top of the file (This only applies if the router address is 192.168.1.1, otherwise change accordingly):
3. `interface eth0 static ip_address=192.168.1.XX/24 static routers=192.168.1.1 static domain_name_servers=192.168.1.1 8.8.8.8`
4. `sudo reboot`

2.4.3. Setting up Sense Hat on Raspberry Pi

1. `sudo apt-get update`
2. `sudo apt-get upgrade`
3. `sudo apt-get install sense-hat`
4. `sudo reboot`

2.4.4. Setting up MQTT.paho

1. `pip install paho-mqtt`

3. Design, implementation, and experiment setup

3.1. Design & implementation

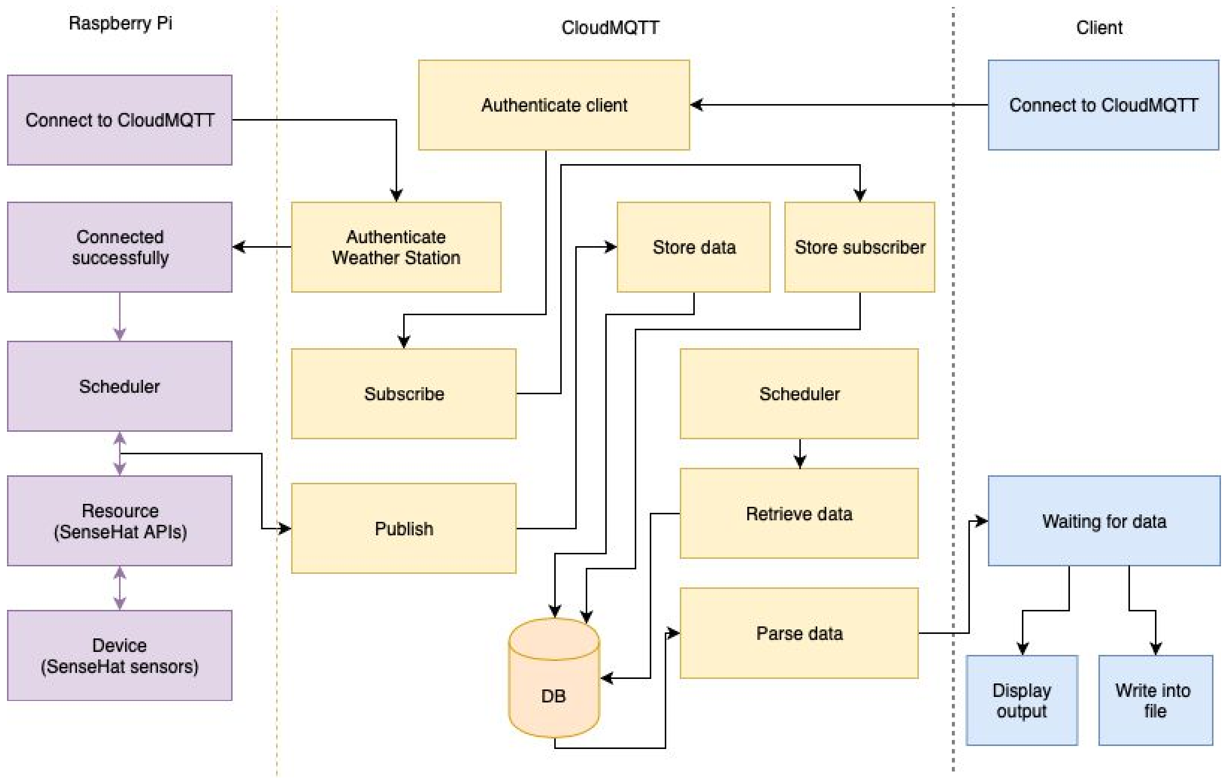


Figure 3: Project's Architecture

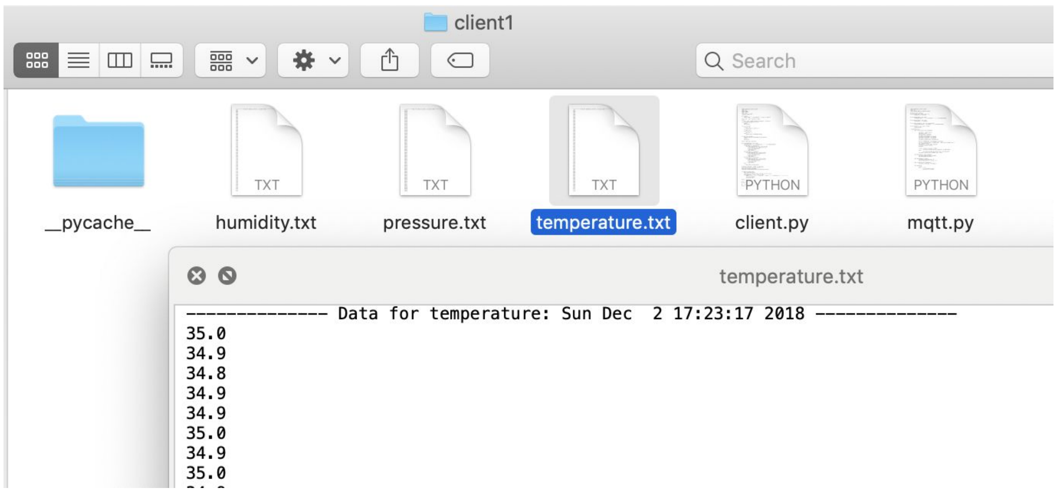
For this project, we have divided it up into 3 major components: Raspberry Pi as the weather station and the publisher, Client as the subscriber and CloudMQTT as MQTT - Broker.

CloudMQTT as a service provider provides a set of APIs and configurations that allows us to easily set up our MQTT - Broker. It allows us to create different credentials for different users for each instance. Hence, allowing different clients to be authenticated and subscribed to their respective topics. We can also configure a scheduler in CloudMQTT to publish the data captured to the respective clients every few seconds or minutes. The data will be either dropped if the clients are in idle state or published to them. Thus, this could help in the projects where power consumption is a concern. In our project, we have set it to publish immediately for the purpose of capturing the data of the effect of simulated "Foehn wind".

Using the DRY principle in Software Engineering, we have implemented a python file, `mqtt.py`, as a service provider that uses the set of CloudMQTT's open source APIs and provides the services to authenticate, connect, publish and subscribe into

CloudMQTT. Weather station and clients will authenticate through this service and to either publish or subscribe to their respective topics. You can refer to Appendix A for a clearer diagram on the interaction process.

Next, in the Raspberry Pi, we have implemented a python script, `weather_script.py` that uses `mqtt.py` to connect to CloudMQTT and Sense Hat's APIs to retrieve information such as temperature, humidity, pressure and light. Inside the script, we have also implemented a scheduler that will prompt the sensor at every 5 minutes to fetch the data from Sense Hat and publish it to CloudMQTT. The purpose of setting the scheduler for every 2 seconds for our experiment where we will be able to simulate different kind of scenarios to change and log the data that was captured by Sense Hat.



```
-------------- Data for temperature: Sun Dec  2 17:23:17 2018 --------------
35.0
34.9
34.8
34.9
34.9
35.0
34.9
35.0
```

Figure 4: Logging of Data in Clients

Lastly, in our `client.py`, it will be used as one of the subscribers that uses `mqtt.py` to connect to the CloudMQTT and subscribe to the relevant topics. We have also implemented a logger in the client to log the data that it will be receiving from CloudMQTT as shown above.

3.2. Experiment setup

Our experiment was conducted indoors. With the sensor attached to the Raspberry Pi, we connected it to a router for internet access. On our PC, we prepared two clients, one of which was subscribed to all three weather topics provided by the sensor - Temperature, Humidity, Pressure, while the other client was subscribed to only the Temperature topic. We accessed the Raspberry Pi through SSH and started the weather station on it.

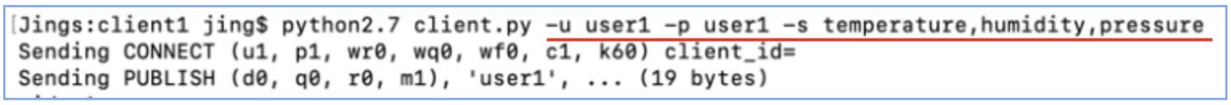
From the MQTT - Broker, we were able to see that the data transmitted from the Raspberry Pi was successfully received by the broker. From the clients' end, we also logged the data received from the MQTT - Broker into the respective weather component files. Our scheduler on the Raspberry Pi is set to capture and transmit data every 2 seconds.

To create changes to our environment, we used a hairdryer on the sensor. We would expect the readings on the sensor to record an increase in temperature and decrease in humidity over time. Our motivation behind this experiment will be explained in Section 4.

4. Experiments

Our experiment is a simulation of foehn monitoring, using a hairdryer. Foehn wind is a type of wind that is warm and dry, and can cause some serious natural disasters in some countries. In Taiwan for example, foehn winds can bring droughts, dry up farmlands and exacerbate forest fires. Foehn wind can also melt snow, causing avalanche and floods. The motivation for our experiment is to be able to use foehn wind as an example, to test the detection of environmental changes and the transmission of data through the MQTT protocol. Foehn wind is sometimes known also as “hairdryer” wind because the wind is warm and dry like the hairdryer wind, hence we used a hairdryer in this experiment to emulate the effects of a foehn wind well. We would expect to record an increase in temperature and a drop in humidity when the hairdryer is used on the sensor.

On the Raspberry Pi (server) side, we ran our weather station script to retrieve data from the sensor. Upon a successful connection with CloudMQTT (MQTT - Broker), the readings captured on the Raspberry Pi will published on CloudMQTT every interval as scheduled on the Raspberry Pi. On CloudMQTT, we configure Users and ACL to create client’s credentials and enable read and write permissions. This allows only authenticated clients to receive data from CloudMQTT. On the client's’ side, we first ran the client’s script with the client’s credentials (username, password) as registered in CloudMQTT and topics which the client is to be subscribed to.



```
[Jings:client1 jing$ python2.7 client.py -u user1 -p user1 -s temperature,humidity,pressure
Sending CONNECT (u1, p1, wr0, wq0, wf0, c1, k60) client_id=
Sending PUBLISH (d0, q0, r0, m1), 'user1', ... (19 bytes)
```

Figure 5: Starting of Script on Client with Authentication

Our readings of weather data was collected over a time frame of 250 seconds. With our weather station scheduler set to retrieve and publish data every two seconds, we collected a total of 125 readings of temperature, humidity and pressure. The diagram below shows the weather data in the form of a line chart.

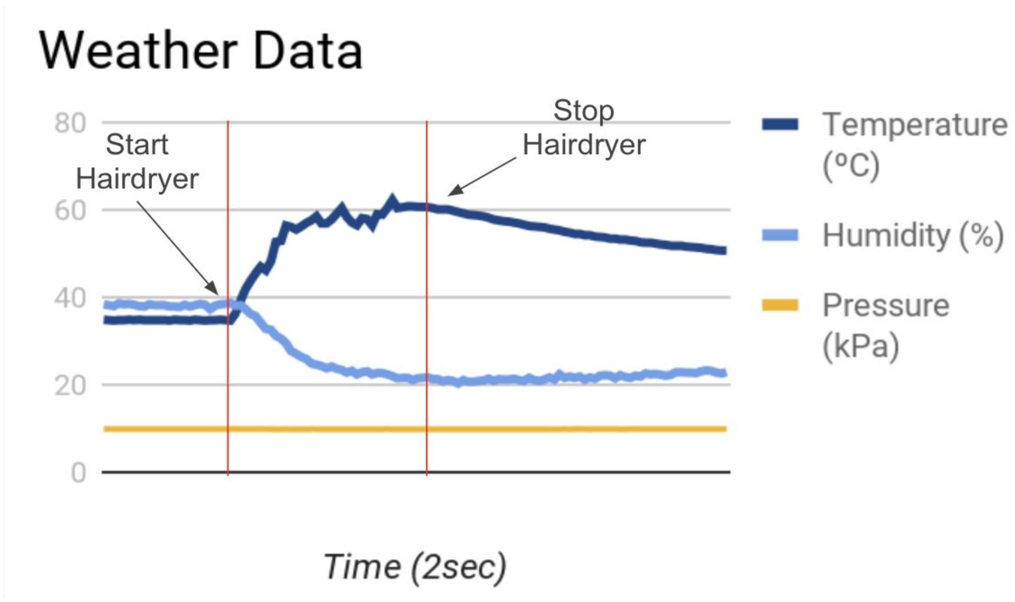


Figure 6: Weather Data received by Clients

The start and stop points of the hairdryer are indicated in the graph. We can see that the temperature, humidity and pressure are approximately constant before the start of the hairdryer. After the hairdryer is started, we saw a sudden increase in temperature, and a decrease in humidity. When we stopped the hairdryer, the temperature and humidity gradually decreased and increased respectively. Throughout the experiment, pressure seemed to be unaffected.

We were also able to show that clients only receive data that they had subscribed to. As shown below, the client that had subscribed to all topics were given data for all topics, and the client that had subscribed to only the temperature topic was given only the temperature data.

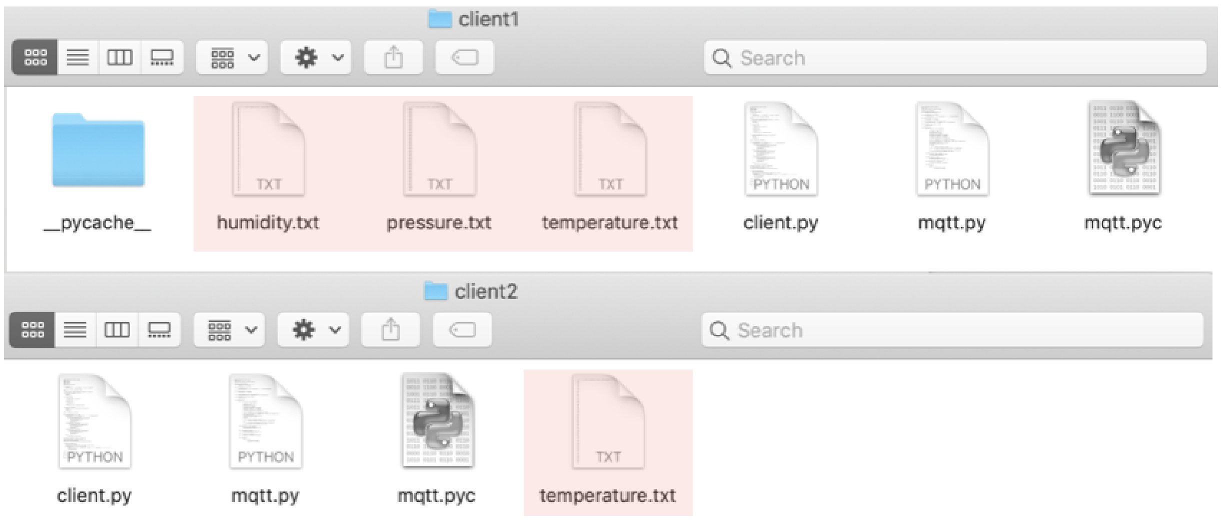


Figure 7: Data written into Clients

From our experiment, we are able to show two things:

1. The results from our experiment were consistent with the results we expected. Using a hairdryer has the effects of increasing the temperature and decreasing the humidity, both of which were reflected in our recordings.
2. The implementation of MQTT protocol in our IoT weather station was successful. The MQTT broker was able to receive data from our Raspberry Pi server and publish only the relevant data to the clients.

5. Conclusion and future work

We presented our set-up, implementation and experimentation of the IoT Weather Station using MQTT protocol. Our results in the simulation displays the expected changes in environmental components when the hairdryer is applied, as well as demonstrates the MQTT protocol in the publishing of data.

5.1. Weather forecasting

Weather forecasting is one useful application of the IoT Weather Station. Sensors can be placed at certain locations to collect data of the weather in those areas over time. The data can be sent wirelessly to a research centre, for example, to do analysis of the data for weather forecasting. This is especially essential in countries that are prone to natural disasters or have extreme climate changes. Alerts can be sent in advance and the necessary actions can be taken in advance.

5.1.1. Foehn monitoring

In particular, one application of weather forecasting is Foehn monitoring. Foehn winds can affect agriculture as the wind is warm and dry. With IoT sensors to detect the presence of such winds, one further extension is to deploy devices that reduce the negative effects when an alert is received. For example, in farms, if Foehn wind is detected, sprinklers can be immediately activated to cool and increase humidity to protect plants from damage.

5.2. Smart homes

IoT Weather Stations can also be used in Smart homes. Data for different weather components can be collected in one’s homes, and then sent to a mobile phone. A person can check and adjust the temperature of his home when he is about to arrive home, so that the house is at a comfortable temperature. More than just convenience, if wild temperature swings are unusual readings are detected, then an alert can be sent to the home’s owner. For example, if a fire broke out in the house, the home’s owner will be able to receive the alert and contact the fire station to solve the problem quickly.

6. References

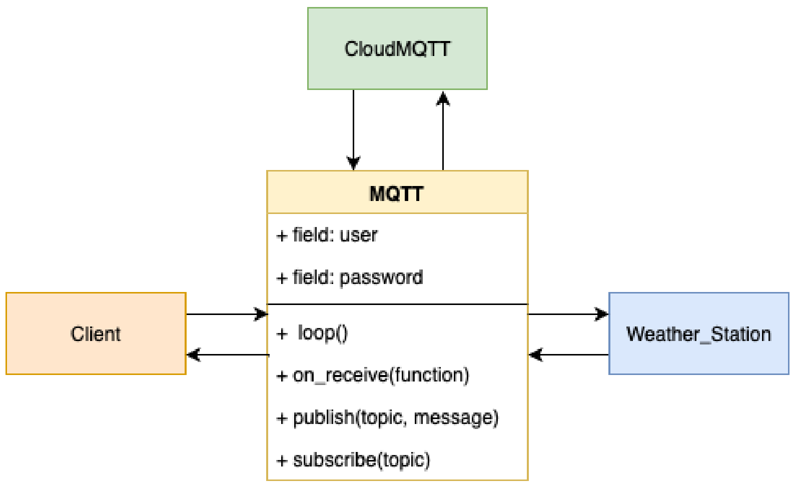
[1] 'Hairdryer wind' melts snow in Antarctica in winter as well. (2018, May 02). Retrieved from https://www.sciencedaily.com/releases/2018/05/180502104045.htm

[2] Documentation Getting started. (n.d.). Retrieved from https://www.cloudmqtt.com/docs.html

[3] Raspberry Pi Weather Station using the Sense HAT. (2018, September 05). Retrieved from https://pimylifeup.com/raspberry-pi-weather-station/

7. Appendix

Appendix A: Client and Server Interaction



We have three python files, `client.py` and `mqtt.py` and `weather_script.py`. We have a MQTT class that act as a service provider to reduce code duplication. Both clients and weather station will be able to access CloudMQTT with their configured unique credentials, as mentioned in Section 3.1 and to publish or subscribe to CloudMQTT through MQTT class.

Appendix B: Source code of the project

You will be able to download the source code in zip format under the release tab from: https://github.com/jingyinno/IoT-Weather-Station

Appendix C: Video demo of the project

You will be able to refer to the project demo that we have presented during our presentation from: https://youtu.be/NqlFmDK53vg