Internet of Things (IoT) Communication

Topic 4: MQTT Protocol

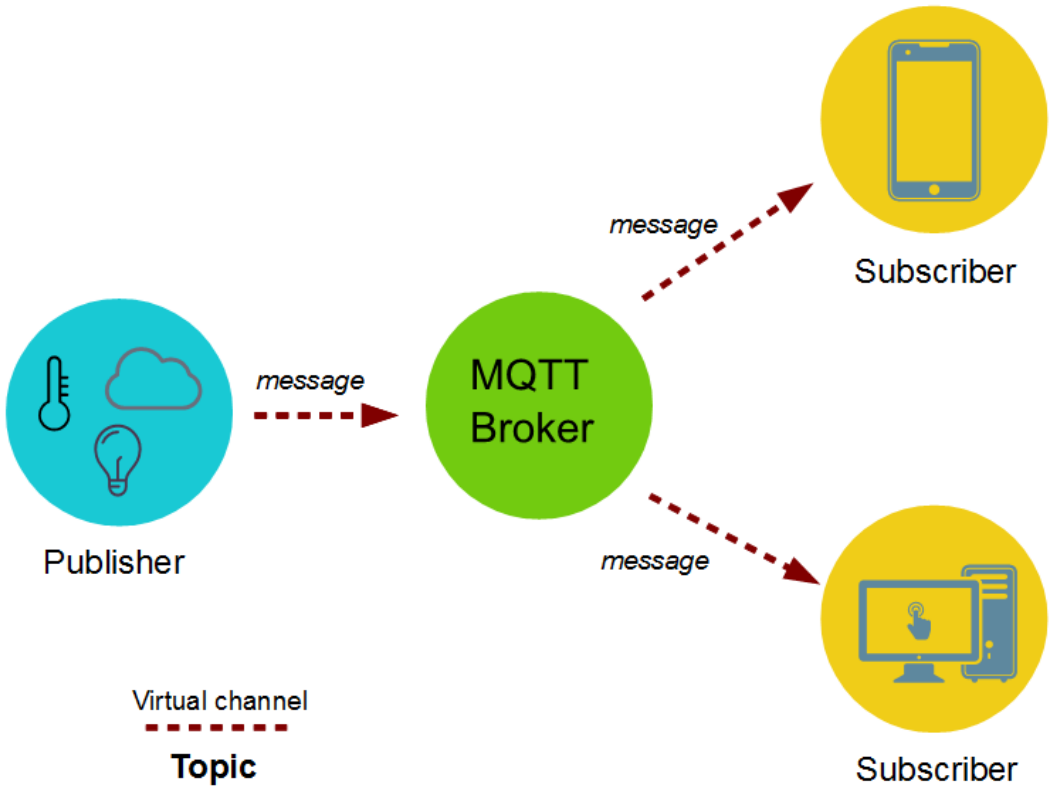
Deliverables

In this exercise, you should complete all the tasks and submit the following Python source codes to the course Moodle page.

- Task 1 – ***task1_mqtt_publish.py*** & ***task1_mqtt_subscribe.py***
- Task 2 – ***task2_mqtt_subscribe.py*** & ***task2_traffic_light_publisher.py***
- Task 3 – Answer Q3 on Moodle
- Task 4 – ***task4_mqtt_publish.py***, ***task4_mqtt_subscribe.py*** & ***task4_traffic_light_subscriber.py***

Introduction

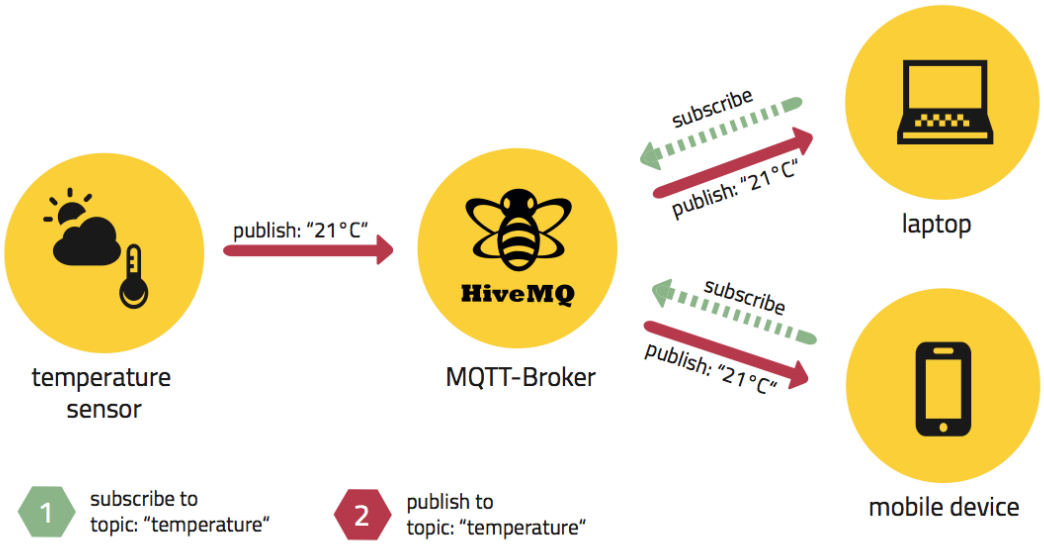
In every IoT application, there is communication between the devices or data sink. **Message Queuing Telemetry Transport (MQTT)** is a lightweight publish-subscribe protocol widely used in machine-to-machine (M2M) communication for IoT applications. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited.



Publish-subscribe is a pattern where senders of messages, called **Publishers**, do not program the messages to be sent directly to some target recipients. Instead, they characterize message payloads into different **topics**, without knowing who will be receiving the messages. On the other hand, interested clients called **Subscribers**, may choose to subscribe to these topics, and they will then receive all the messages related to the subscribed topics without the need to know who the publishers are.

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.
You can think of MQTT as writing stories for a newspaper where you don't know who will be subscribing to the article!

For example:



The temperature sensor (on the left) is the Publisher, and it keeps posting its temperature measurements to the topic called "temperature" to the MQTT-Broker (in the middle). Then, if both the laptop and mobile devices (on the right) are also interested in knowing the latest temperature, they can simply subscribe to the "temperature" topic from the Broker, and the Broker will send them the messages accordingly!

MQTT Client

In this workshop, we are going to use the Paho Python client library to manage MQTT connection, publishing and subscription on Raspberry Pi. First of all, install the library with the command:

sudo apt-get install mosquitto mosquitto-clients

Note: The API reference of this library can be found on https://eclipse.org/paho/clients/python/docs/

Let's try to copy and paste the following codes:

mqtt_publish.py

```
import paho.mqtt.publish as publish

hostname = "test.mosquitto.org"  # Sandbox broker
port = 1883  # Default port for unencrypted MQTT

topic = "PC000/test"  # '/' is used as the delimiter for sub-topics

publish.single(topic, payload="Hello, MQTT!", qos=0,
        hostname=hostname,
        port=port)
```

mqtt_subscribe.py

```
import paho.mqtt.client as mqtt

hostname = "iot.eclipse.org"  # Sandbox broker
port = 1883  # Default port for unencrypted MQTT

topic = "PC000/#"  # Wildcard character '#' indicates all sub-topics (e.g.
PC000/test, PC000/sensor/temperature, etc.)

def on_connect(client, userdata, flags, rc):
        # Successful connection is '0'
        print("Connection result: " + str(rc))
        if rc == 0:
                # Subscribe to topics
                client.subscribe(topic)
```

```
def on_message(client, userdata, message):
        print("Received message on %s: %s (QoS = %s)" %
                (message.topic, message.payload.decode("utf-8"),
str(message.qos)))

def on_disconnect(client, userdata, rc):
        if rc != 0:
                print("Disconnected unexpectedly")

# Initialize client instance
client = mqtt.Client()

# Bind events to functions
client.on_connect = on_connect
client.on_message = on_message
client.on_disconnect = on_disconnect

# Connect to the specified broker
client.connect(hostname, port=port)

# Network loop runs in the background to listen to the events
client.loop_forever()
```

Note: `client.loop_forever()` uses the main thread, and prevent the codes below that line from being executed.

***Task 1**: Study the code in **mqtt_publish.py** and **mqtt_subscribe.py**. Change the topic name and payload such that you can distinguish your messages from your classmates.*

*Run **mqtt_subscribe.py** first, then execute **mqtt_publish.py** in another Terminal window. See if you can receive your messages correctly.*

*Submit to Moodle: demonstration video & **task1_mqtt_publish.py** & **task1_mqtt_subscribe.py***

If you have any questions, please refer to the API reference at https://eclipse.org/paho/clients/python/docs/ or ask the TA for help!

Submit your code for Task 1 to Moodle before working on the next task!

IoT Traffic Light – Publisher

Still remember the traffic light you built? Let's try to turn it into an IoT traffic light using the MQTT protocol!

Make some changes to your ***traffic_light.py*** program, so that it will publish its state (State 1, 2 or 3) to the MQTT broker. Then change the ***mqtt_subscribe.py*** program so that it subscribes to the traffic light states and displays it on the screen.

1. Save ***traffic_light.py*** as a new file called ***traffic_light_publisher.py*** before making any changes!

2. Import the required library and the default MQTT settings
```
import paho.mqtt.client as mqtt
hostname = "iot.eclipse.org"
port = 1883
```

3. Create an appropriate topic name for the traffic light state messages so that the subscribers can know what to be expected!
```
topic_state = "PCxxx/traffic_light/state"
```

4. Due to the high transmission frequency, it is better to maintain a consistent connection to the MQTT broker, rather than using `paho.mqtt.publish.single` as in ***mqtt_publish.py***. To establish a connection, it would be similar as something we had done for ***mqtt_subscribe.py***.
```
mqttc = mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_publish = on_publish
mqttc.on_disconnect = on_disconnect
mqttc.connect(hostname, port=port,
keepalive=60, bind_address="")
mqttc.loop_start()
```

Note that `mqttc.loop_start()` spawns a new thread for connection, unlike `loop_forever()` we saw in the previous task.

5. Complete the function definitions for `on_connect()`, `on_publish()`, and `on_disconnect()`

```
def on_connect(client, userdata, flags, rc):
    # Successful connection is '0'
    print("[MQTT] Connection result: " + str(rc))

def on_publish(client, userdata, mid):
    print("[MQTT] Sent: " + str(mid))

def on_disconnect(client, userdata, rc):
    if rc != 0:
    print("[MQTT] Disconnected unexpectedly")
```

6. To publish a message, use the publish() function: **publish(topic, payload, qos, retain)**. For example:
   `mqttc.publish(topic_state, 1, qos=0, retain=False)  # State 1`

7. Insert the above code segments into their correct positions, and publish a new message for the latest state at the beginning of every state changes

8. Open ***mqtt_subscribe.py*** and make the appropriate changes so that the subscriber can display the real-time traffic light state on the screen.

***Task 2**: Run **mqtt_subscribe.py** first, then execute **traffic_light_publisher.py** in another Terminal window. See if you can receive the traffic states correctly.*

*Submit to Moodle: demonstration video & **task2_mqtt_subscribe.py** & **task2_traffic_light_publisher.py***

Submit your code for Task 2 to Moodle before working on the next task!

QoS level

You may notice that the **publish()** function has a parameter called **qos**. According to the API reference, it means the **Quality of Service** level to use.

The Quality of Service (QoS) level is an agreement between sender and receiver of a message regarding the guarantees of delivering a message. There are 3 QoS levels in MQTT:

- At most once (QoS 0)
- At least once (QoS 1)
- Exactly once (QoS 2)

QoS is a major feature of MQTT. It makes communication in unreliable networks a lot easier because the protocol handles retransmission and guarantees the delivery of the message, regardless how unreliable the underlying transport is. Also, it empowers a client to choose the QoS level depending on its network reliability and application logic.

QoS 0 – at most once

QoS level 0 guarantees a best effort delivery, meaning that a message won't be redelivered by the sender. This is often called "fire and forget" and provides the same guarantee as the underlying TCP protocol.

QoS 1 – at least once

When using QoS level 1, it is guaranteed that a message will be delivered at least once to the receiver. However, the message can also be delivered more than once.

QoS 2 – exactly once

QoS level 2 guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level.

***Task 3**: Answer on Moodle: Suggest a use case scenario for each of the QoS levels, and provide justifications to your choice with reference to the characteristics of the QoS levels.*

IoT Traffic Light – Subscriber

Now, we would like to add a new feature to the traffic light – **Emergency signal**.

First of all, make some changes to your ***mqtt_publish.py*** program, so that it will publish a Boolean value indicating the emergency status (***False*** as Normal, ***True*** as Emergency) to the topic of "**PCxxx/traffic_light/emergency**".

Next, save ***traffic_light_publisher.py*** as a new file called ***traffic_light_subscriber.py***, and add a MQTT subscriber into the program so that it has the following behavior: When the emergency status received is ***False***, everything should be normal; when the emergency status received is ***True***, blink the Signal LED to notify the pedestrians. Also, the Wait button should be disabled during the emergency state.

Finally, modify the ***mqtt_subscribe.py*** program so that it subscribes to both the traffic light states and the emergency status, and displays them on the screen.

Task 4*: Run* ***mqtt_subscribe.py*** *first, then run* ***traffic_light_subscriber.py****, then execute* ***mqtt_publish.py****.*

Submit to Moodle: demonstration video & ***task4_mqtt_subscribe.py*** *&* ***task4_traffic_light_subscriber.py*** *&* ***task4_mqtt_publish.py***

Congratulations! That's all for the MQTT session. ☺

References

1. *http://www.switchdoc.com/2016/02/tutorial-installing-and-testing-mosquitto-mqtt-on-raspberry-pi/*
2. *https://eclipse.org/paho/clients/python/docs/*
3. *https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels*