CME466 - Design of an Advanced Digital System

MQTT and Its Implementation

Instructor: Prof. Khan A Wahid
TA: Omid Yaghoobian
Winter 2024

Copyright notice

- These slides are intended to be used in CME466 course only.
- The materials presented in this entire document are protected by copyright laws.
- You shall not reuse, publish, or distribute any of these slides without obtaining permission from the presenter and individual copyright owners.

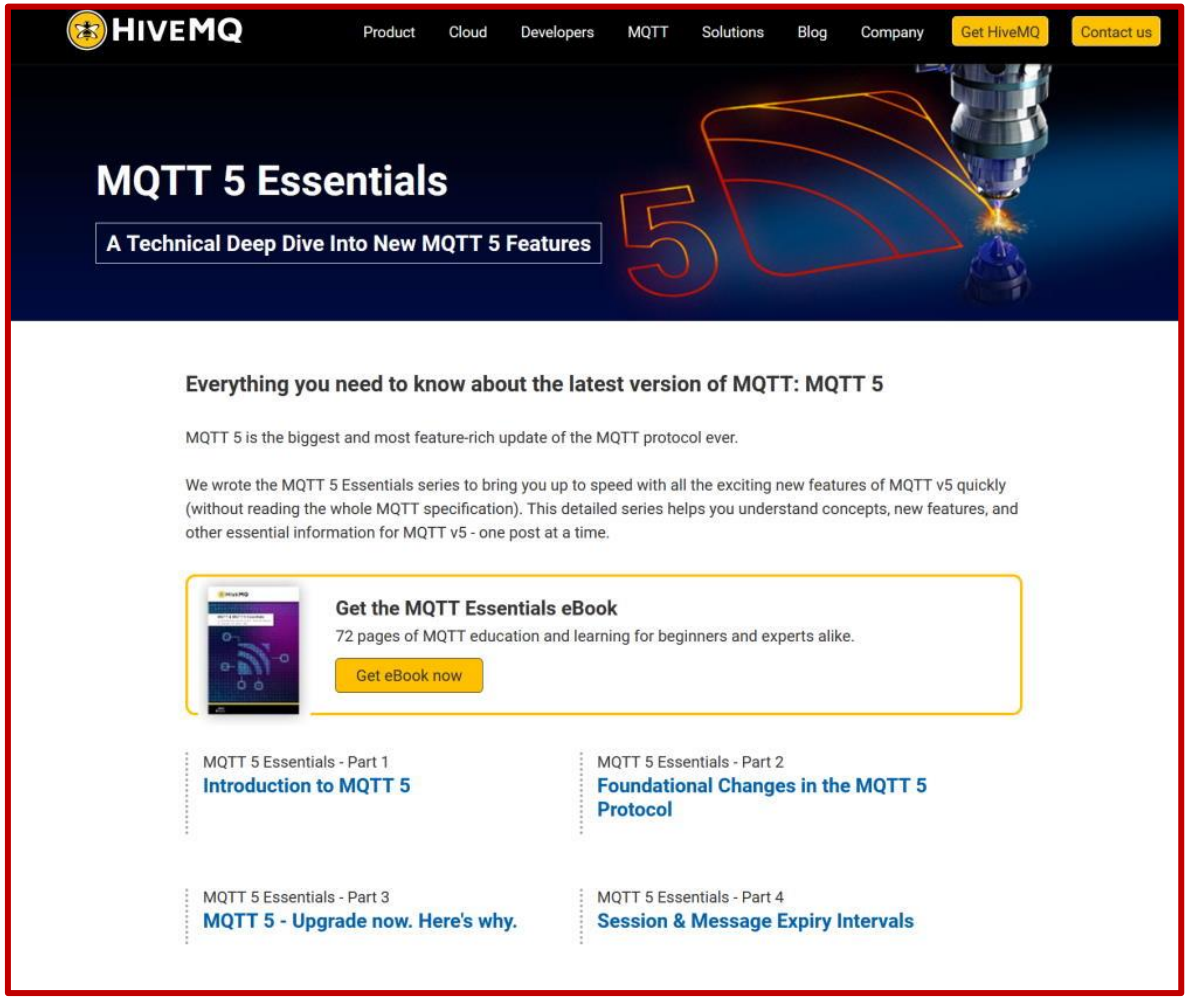
https://copyright.laws.com/copyright-symbol

MQTT: Download free e-book

MQTT & MQTT 5 Essentials

A comprehensive overview of MQTT facts and features for beginners and experts alike

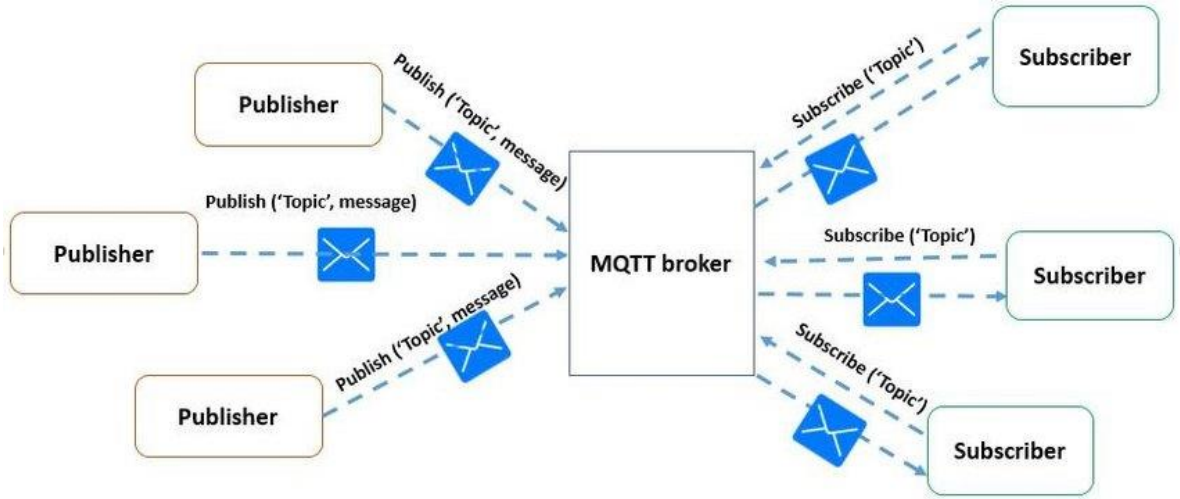
https://www.hivemq.com/mqtt-5/



MQTT (MESSAGE QUEUING TELEMETRY TRANSPORT)

- MQTT, designed in 1999, is a lightweight machine to machine (M2M) communication protocol that supports the publish/subscribe architecture with minimal bandwidth requirements, power consumption, and message data overhead
- An MQTT client publishes messages to a broker through an address known as Topic. Another client can receive the messages by subscribing to that Topic. Clients can subscribe to multiple topics.
- MQTT publish/subscribe protocol provides a scalable and reliable way to connect devices over the Internet. Today, MQTT is used by many companies to connect millions of devices to the Internet

- https://mosquitto.org/
- www.mqtt.org/



https://www.hivemq.com/

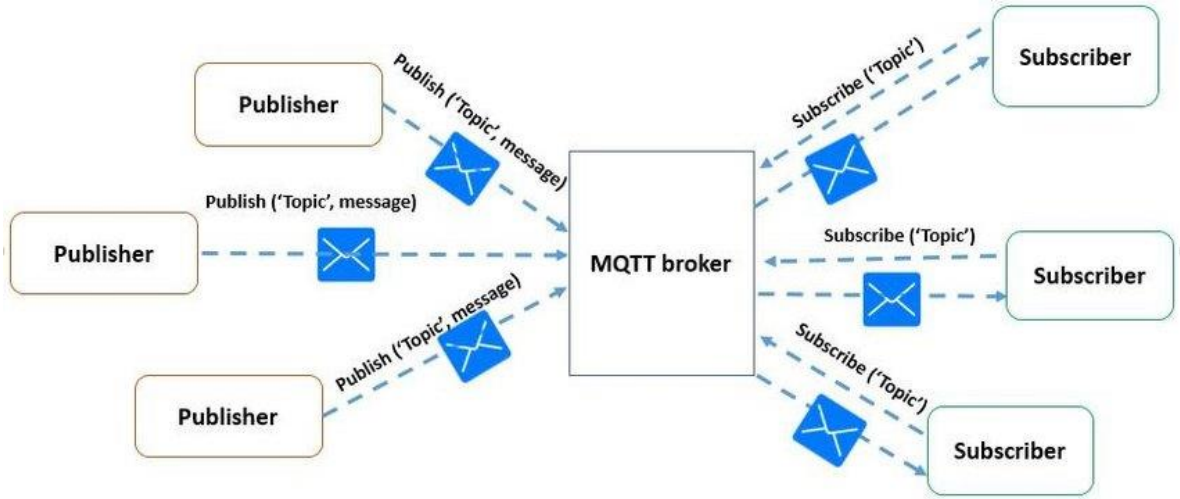
MQTT (MESSAGE QUEUING TELEMETRY TRANSPORT)

MQTT Clients

- MQTT clients publish a message to an MQTT broker and other MQTT clients subscribe to messages they want to receive
- Implementation of MQTT clients typically requires a minimal footprint, so it is well suited for deployment on small constrained devices and are very efficient in their bandwidth requirements

MQTT Broker

- MQTT brokers receive published messages and dispatch the message to the subscribing MQTT clients
- An MQTT message contains a message topic that MQTT clients subscribe to and MQTT brokers use these subscription lists for determining the MQTT clients to receive the message



https://www.hivemq.com/

MQTT Implementation in Python - Publish

- Install MQTT package
  - *pip install paho-mqtt*
  - More info here: https://pypi.org/project/paho-mqtt/
- Import paho library
  - *import paho.mqtt.client as mqtt*
- Use a public broker, declare objects/methods
  - *mqttBroker = "broker.hivemq.com" // public broker*
  - *client = mqtt.Client("fake_temp1") // your name as client*
  - *client.connect(mqttBroker)*
- Publish a message
  - *client.publish("room_temp", randN) // topic is room_temp*

MQTT Implementation in Python - Subscribe

- Import paho library
  - *import paho.mqtt.client as mqtt*
- Use a public broker, declare objects/methods
  - *mqttBroker = "broker.hivemq.com"*
  - *client = mqtt.Client("fake_temp2") // your name as client*
  - *client.connect(mqttBroker)*

- Subscribe to receive a message
  - *client.loop_start()*
  - *client.subscribe("room_temp")*
  - *client.on_message = on_message // call a function*

- Callback function
  - *def on_message(client, userdata, message):*
    - *b = message.payload.decode("utf-8")*
    - *print ("message received", b)*

Quality of Service Levels (QoS)

- The Quality of Service (QoS) level is an agreement between the sender of a message and the receiver of a message that defines the guarantee of delivery for a specific message.
- These QoS levels allow for more reliable IoT applications since the underlying messaging infrastructure and adapt to unreliable network conditions.

- MQTT implements 3 levels of Quality of Service
  1. At most once (0),
  2. At least once (1),
  3. Exactly once (2).

https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

Quality of Service Levels (QoS)

- At most once (0) (fire and forget)
  - QoS0, no guarantee of delivery.
  - The recipient does not acknowledge receipt of the message and the message is not stored and re-transmitted by the sender.
  - provides the same guarantee as the underlying TCP protocol.

- At least once (1)
  - QoS1 guarantees that a message is delivered at least one time to the receiver. The sender stores the message until it gets a PUBACK packet from the receiver that acknowledges receipt of the message.
  - It is possible for a message to be sent or delivered multiple times.

- Exactly once (2)
  - QoS 2 mode guarantees that the message is delivered exactly once. MQTT has a facility for a variable length header. It does not provide any MQTT ACK response, but its default transport protocol (TCP) provides TCP ACK for each packet sent.

Retained Messages

- MQTT clients that subscribe to a new topic have no insight into when to expect the first message they will receive.
- However, an MQTT broker can store a retained message that can be sent immediately upon a new MQTT subscription.
- In this case, the MQTT client will receive at least one message upon subscribing to the topic.

Publish:

```
client.publish("room_temp", randN, qos = 0, retain = True)

# you need to send a null message to cancel the retained message flag
client.publish("room_temp", payload = None, retain = True)
```

Subscribe:

```
print ("topic", message.topic, "retained flag?", message.retain)
```

Last Will and Testament

An MQTT client can specify to an MQTT broker a message, called the last will, that will be sent if the MQTT client ungracefully disconnects. This allows for a more graceful system wide notification that a client has been disconnected.

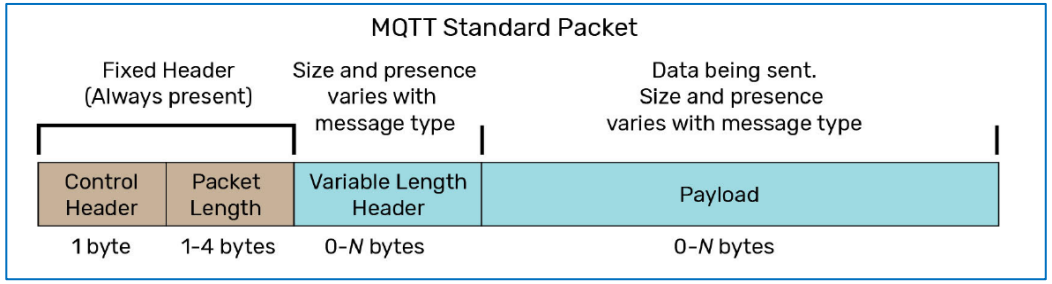
Persistent Sessions

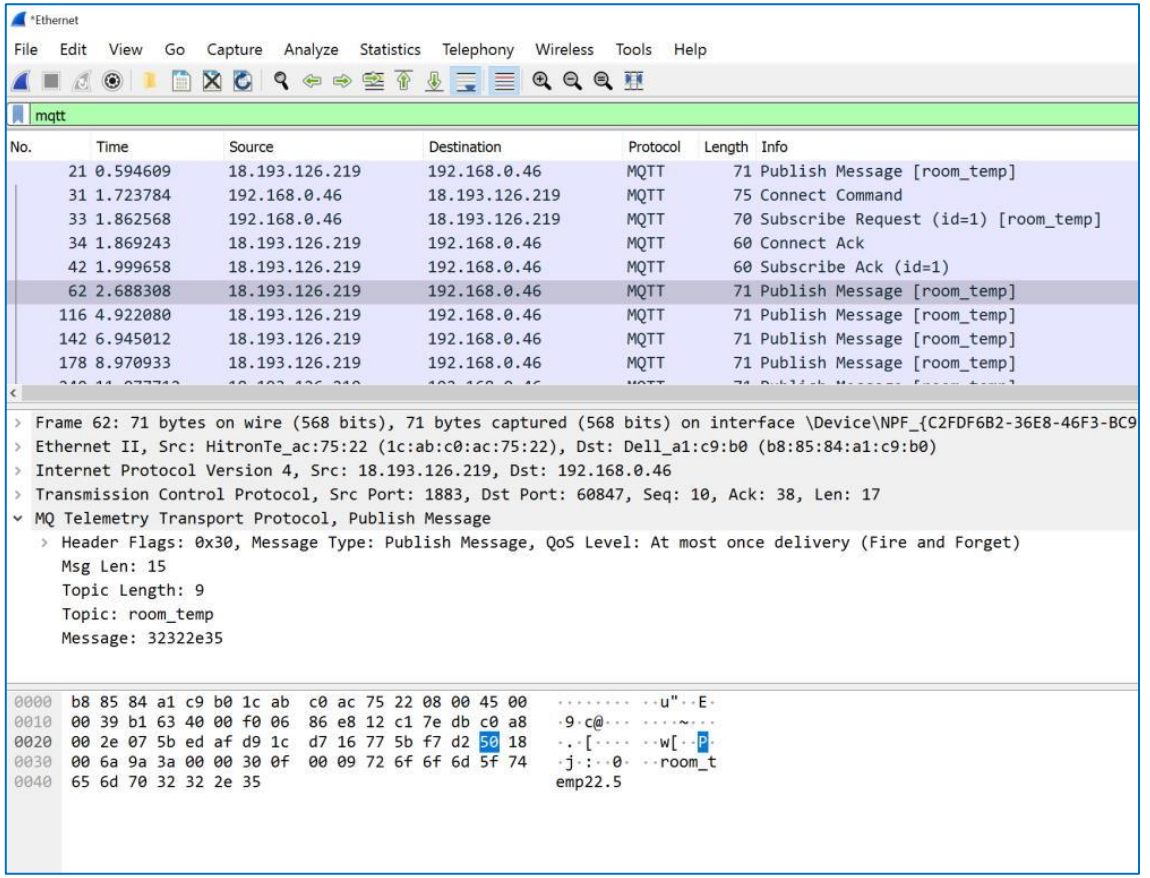
MQTT allows for a persistent session between the client and the broker. This allows for sessions to persist even if the network is disconnected. Once the network is reconnected, the information to reconnect the client to the broker still exists. This is one of the key features that makes the MQTT protocol more efficient than HTTP for use over unreliable cellular networks.

MQTT: Packet

- Unlike HTTP, MQTT was designed for machine-to-machine communication. While HTTP is famously heavyweight, with a long list of message headers used to describe and respond to resources, MQTT is data-agnostic, with a streamlined on-the-wire footprint that can be processed efficiently by devices with limited power and processing capabilities.

- It uses a simple byte array payload with a fixed 2-byte header and variable-length header fields (up to a few additional bytes) to indicate packet length or control codes. A packet can be up to 256 MB in size





https://www.hivemq.com/blog/mqtt5-essentials-part8-payload-format-description/

MQTT strengths

- It requires minimal resources since it is lightweight and efficient
- Support bi-directional messaging between device and cloud
- Can scale to millions of connected devices
- Support reliable message delivery through 3 QoS levels
- Works well over unreliable networks
- Security enabled, so it works with TLS and common authentication protocols

- Note: you cannot identify the clients (pub or sub) using this protocol

FAQs on MQTT Payload Format Description and Content Type

- Format of MQTT payload:
  - The format of the MQTT payload is flexible and can contain any data or information that you want to send. MQTT treats the payload as a raw sequence of bytes and does not impose any specific format or structure on it. It can be any arbitrary data, such as strings, JSON, binary data, or even custom formats specific to your application.
- Max payload size:
  - The maximum allowable size of a packet sent by the client or server, before any MQTTS or other framing is added, is 256MB.
- Can we use JSON?
  - Yes, you can use JSON as the payload format in MQTT. MQTT treats the payload as a raw sequence of bytes, allowing you to send any data format you prefer, including JSON.

https://www.hivemq.com/blog/mqtt5-essentials-part8-payload-format-description/

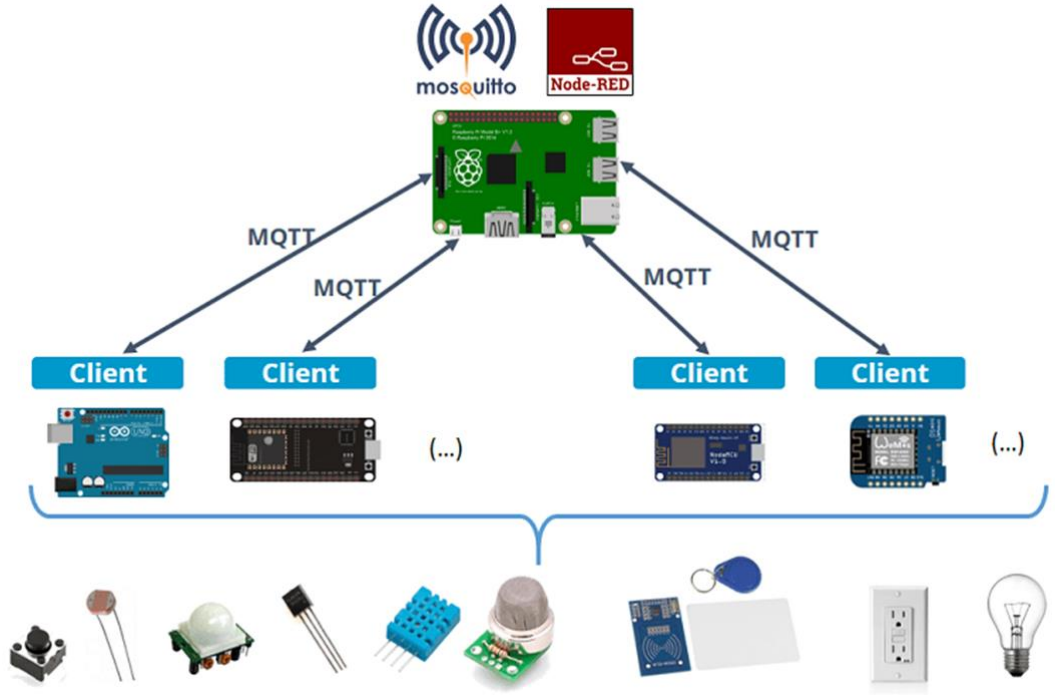
Private MQTT Broker on RPI

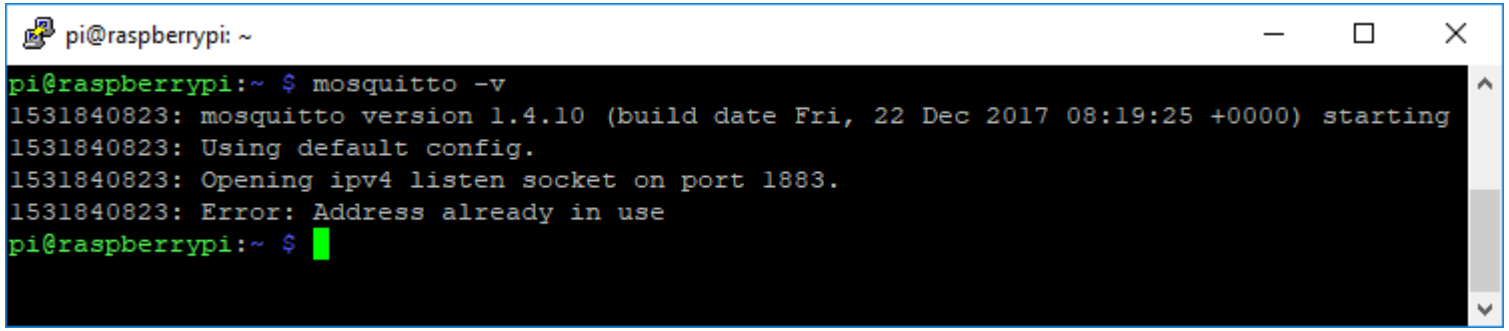
Instead of using public MQTT brokers, you setup a Broker on your PC or RPI.

sudo apt update
sudo apt install -y mosquitto mosquitto-clients

sudo systemctl enable mosquitto.service

mosquitto -v

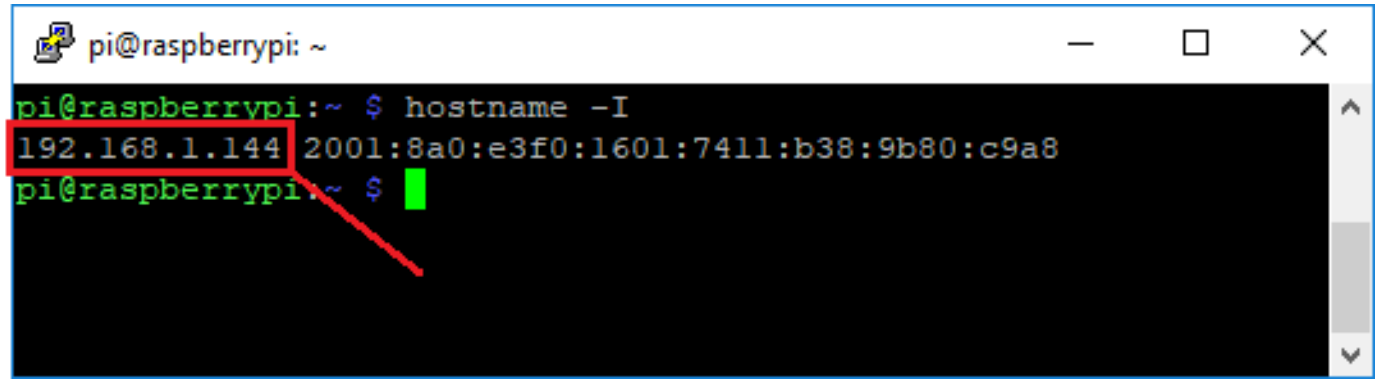




Private MQTT Broker on RPI

To check the RPI IP address use the following command.
You will use the RPI IP address as the broker IP.

hostname -I



Secure MQTT

- Authentication with Username and Password
  - The MQTT protocol provides username and password for authentication on the application level
  - https://www.hivemq.com/blog/mqtt-security-fundamentals-authentication-username-password/
  - *username_pw_set(username, password=None)* from https://pypi.org/project/paho-mqtt/#publishing

  - Mosquitto 2.0 and up offers three choices for authentication: password files, authentication plugins, and unauthorised/anonymous access
  - https://mosquitto.org/documentation/authentication-methods/

- Encrypting the MQTT payload
  - The advantage is that the data is encrypted end to end and not just between the broker and the client (which is the data link)
  - Use python libraries such as *cryptography*, etc.
  - *from cryptography.fernet import Fernet*
  - *cipher_key = Fernet.generate_key()*
  - *cipher = Fernet(cipher_key)*

  - encrypted_message = cipher.encrypt(randN)
  - out_message = encrypted_message.decode()
  - *client.publish("room_temp", out_message)*

http://www.steves-internet-guide.com/encrypting-the-mqtt-payload-python-example/

How to send large files using MQTT

- There are many ways we can publish larger file, like an image or excel file.
- For example, for publishing an image file:

```
#'‘’
# open image in binary read format as file
with open("./image.jpg",'rb') as file:
        filecontent = file.read()
        byteArr = bytearray(filecontent)
        print(byteArr)
        print (len(byteArr))
#’‘’

client.publish("room_temp", byteArr)
print(f"just published byteArr")
```

How to send large files using MQTT

- Then subscribing the same image file:

```
# '''
f = open('image_r.jpg', 'wb’)
# open a file for the image and create a name for the received image
f.write(message.payload)
# write the received bytes to the opened file named "f"
f.close()
#’‘’
```

- To view the received image:

```
'‘’
import cv2from matplotlib
import pyplot as plt%matplotlib inline
#img = cv2.imread("image_r.jpg")
# the above will show BGR image, not typical RGB). below is the fix
img = cv2.imread("image_r.jpg")[...,::-1]
plt.imshow(img)
plt.show()
'''
```

AMQP Implementation

See instructions posted in canvas

CoAP Implementation

See instructions posted in canvas