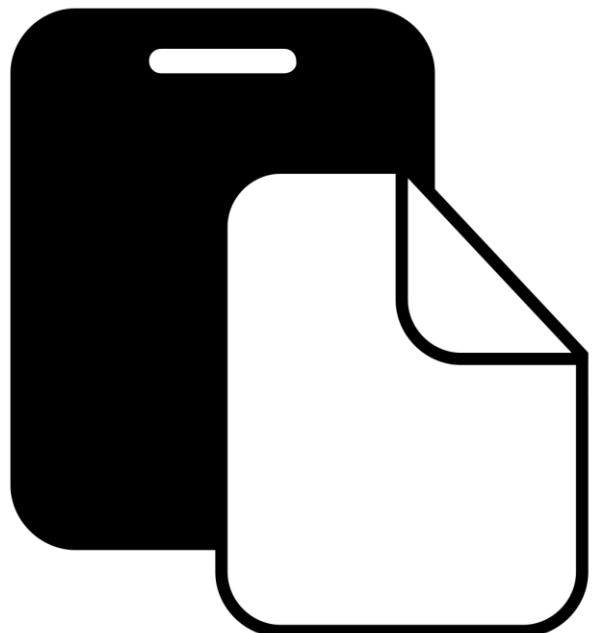


Indirect Messaging

Frank Walsh

Agenda



- Indirect Messaging
 - MQTT
 - MQTT Security
- Example:
 - Publish Env data to broker.

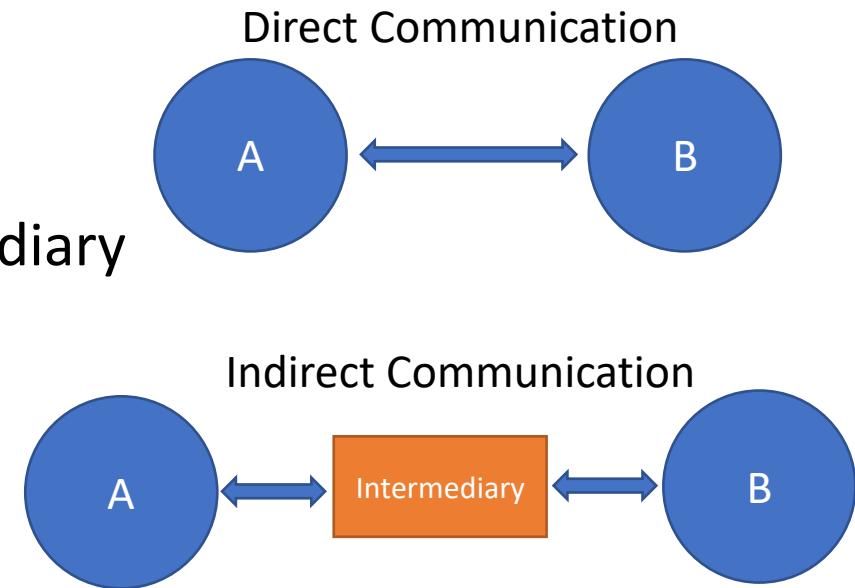
[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)

Indirect Messaging

Publish Subscribe

Using the “Middleman”

- Communication between processes using an intermediary
 - Sender → “The middle-man” → Receiver
 - No direct coupling
- Up to now, only considered Direct Coupling
 - Introduces a degree of rigidity
- Consider...
 - What happens if a device fails during communication in Direct Coupling?
 - What if you'd like to add
- Two important properties of intermediary in communication
 - ***Space uncoupling*** (*devices don't need to "know" about each other*)
 - ***Time uncoupling*** (*devices don't need to be "available" at the same time*)



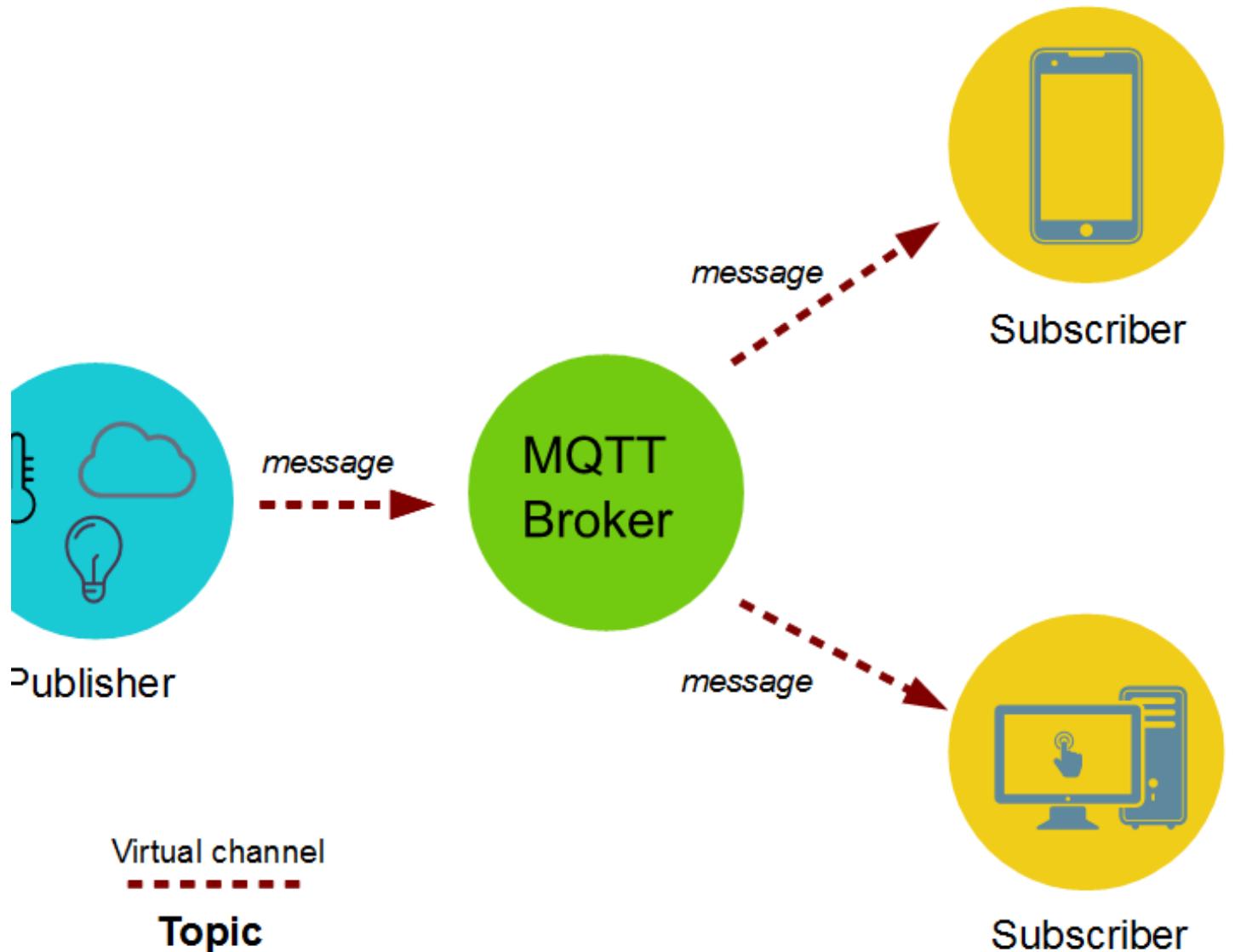


MQTT

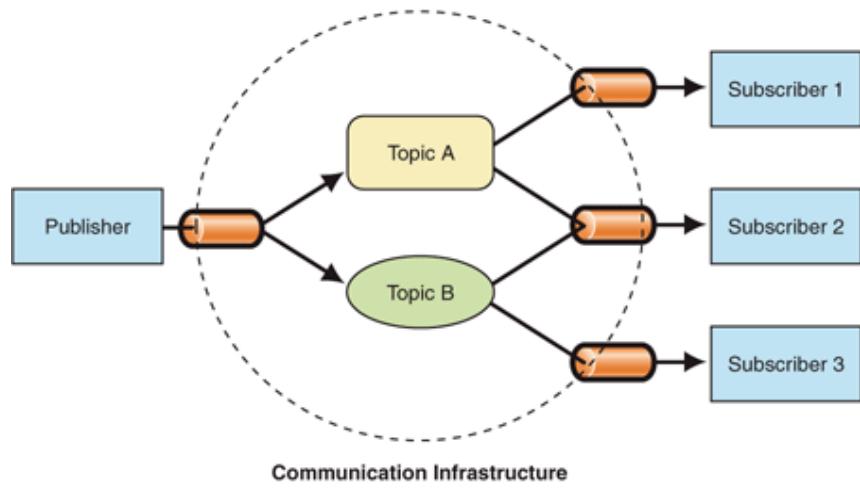
- MQ Telemetry Transport (MQTT)
- Telemetry
 - Remote measurements
- Created by IBM
 - from message queueing (MQ) architecture used by IBM for service oriented networks.
 - Telemetry data goes from devices to a server or broker.
 - Uses a **publish/subscribe** mechanism.
- Lightweight both in bandwidth and code footprint

MQTT – publish subscribe

- **Topics/Subscriptions:** Messages are published to topics.
 - Clients can subscribe to a topic or a set of related topics
- **Publish/Subscribe:** Clients can subscribe to topics or publish to topics.



Publish Subscribe Process



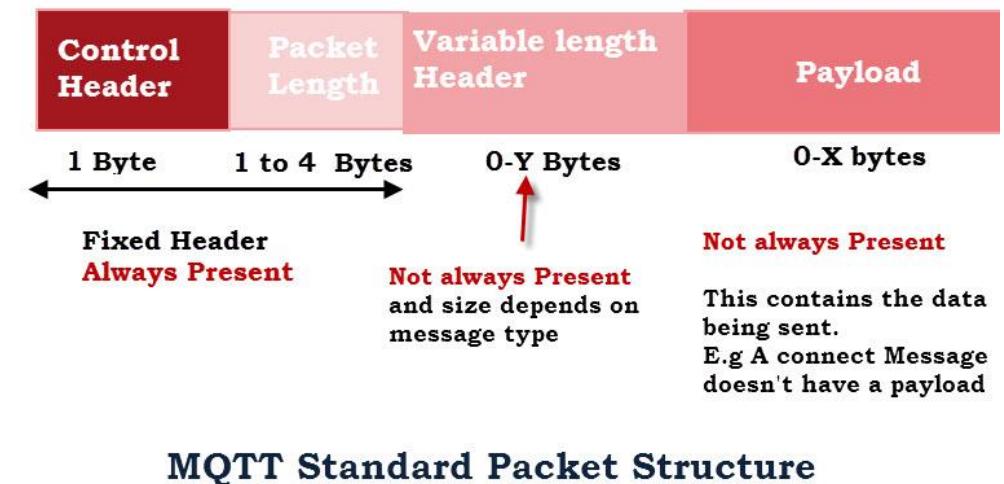
- A message is published once by a publisher.
- Many things can receive the message.
- The messaging service, or “broker”, provides decoupling between the producer and consumer(s)
- A producer sends (publishes) a message (publication) on a topic (subject)
- A consumer subscribes (makes a subscription) for messages on a topic (subject)
- A message server / broker matches publications to subscriptions
 - If no matches the message is discarded
 - If one or more matches the message is delivered to each matching subscriber/consumer

Publish-Subscribe Characteristics

- A published messages may be **retained**
 - A publisher can mark a message as “retained”
 - The broker / server remembers the last known good message of a retained topic
 - The broker / server gives the last known good message to new subscribers
- A Subscription can be **durable or non-durable**
 - Durable: messages forwarded to subscriber immediately, if subscriber not connected, message is stored and forwarded when connected
 - Non-Durable: subscription only active when subscriber is connected to the server / broker

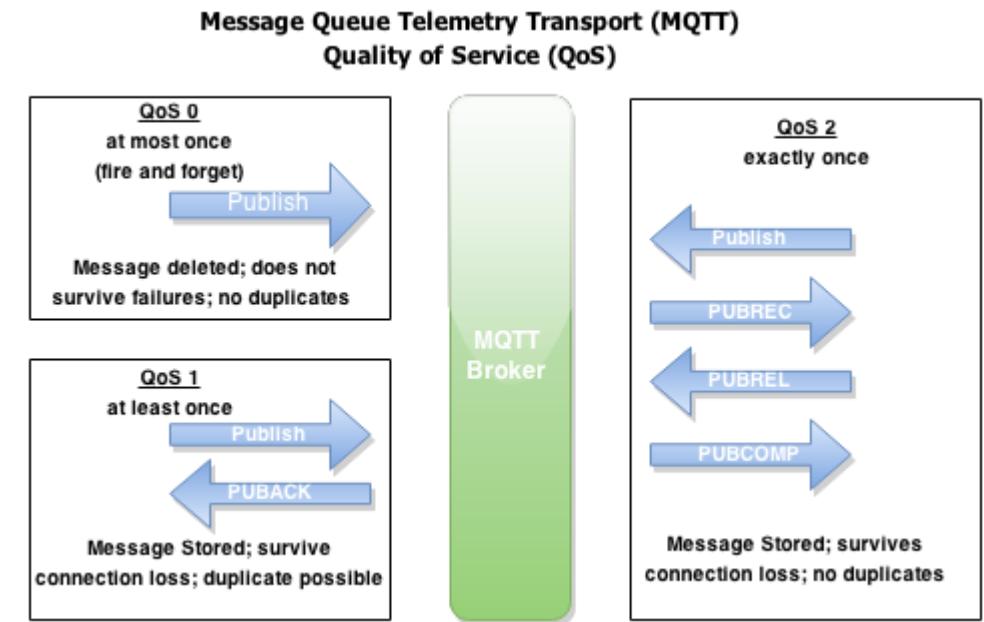
MQTT Characteristics

- MQTT protocol compresses to small number of bytes
 - Smallest packet size 2 bytes
 - Supports always-connected and sometimes connected
 - Provides Session awareness
 - “Last will and testament” enable applications to know when a client goes offline abnormally
 - Typically utilises TCP-based networks.



MQTT Characteristics

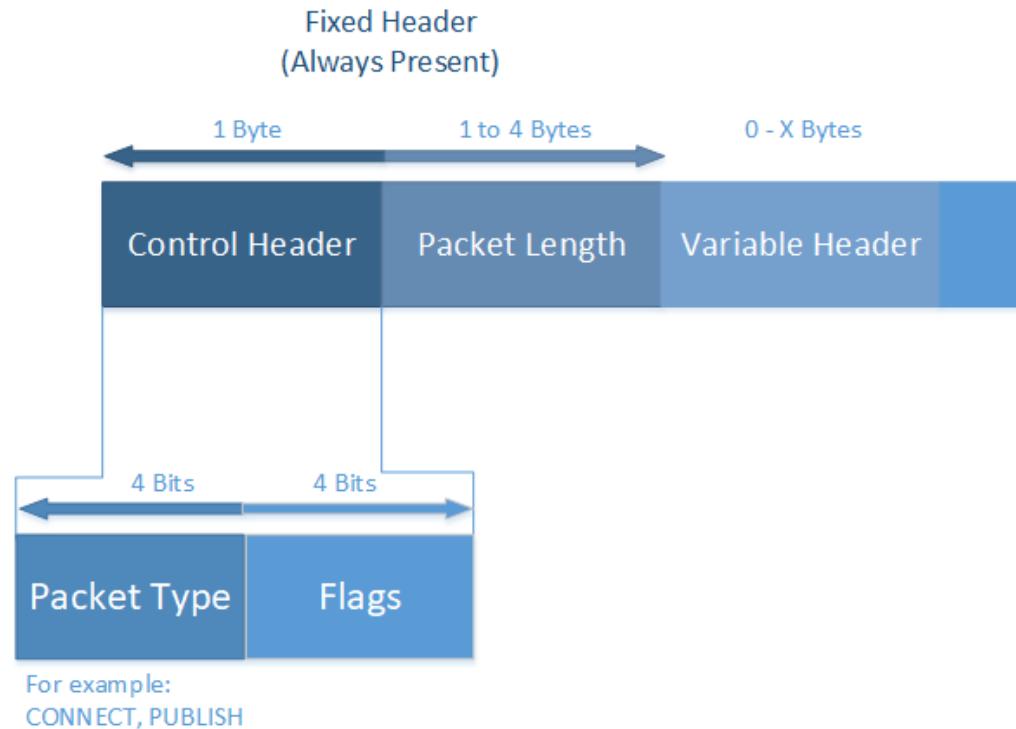
- Three quality of service levels:
 - 0 = At most once (Best effort, No Ack),
 - 1 = At least once (Acked, retransmitted if ack not received),
 - 2 = Exactly once [Request to send (Publish), Clear-to-send(Pubrec), message (Pubrel), ack (Pubcomp)]



<https://dzone.com/articles/internet-things-mqtt-quality>

MQTT Control Packets

- The MQTT protocol uses MQTT Control Packets using the Control Header.
- Control Packet consists of three parts:
 - *fixed header*(present in all packets),
 - *variable header*(optional)
 - *Payload*(optional)



MQTT Control Packets

- MQTT Control Packet Types

| Packet Type | Description | Value | Direction of flow |
|-------------|--|-------|--------------------------------------|
| CONNECT | Client requests a connection to a Server | 1 | Client to Server |
| CONNACK | Acknowledge connection request | 2 | Server to Client |
| PUBLISH | Publish message | 3 | Client to Server or Server to Client |
| PUBACK | Publish acknowledgment | 4 | Client to Server or Server to Client |
| SUBSCRIBE | Subscribe to topics | 8 | Client to Server |
| SUBACK | Subscribe acknowledgment | 9 | Server to Client |
| UNSUBSCRIBE | Unsubscribe from topics | 10 | Client to Server |
| UNSUBACK | Unsubscribe acknowledgment | 11 | Server to Client |
| PINGREQ | Ping request | 12 | Client to Server |
| PINGRESP | Ping response | 13 | Server to Client |
| DISCONNECT | Disconnect notification | 14 | Client to Server |

MQTT Last Will and Testament

- Notify other clients about an ungraceful disconnected client.
- Client can specify its last will message when it connects to a broker.
 - normal MQTT message with a topic, retained message flag, QoS, and payload
 - When an “ungraceful disconnect” occurs, the broker sends the last-will message to all subscribed clients of the last-will message topic.

| MQTT-Packet: | CONNECT |  |
|---|---------|---|
| contains: | | Example |
| <code>clientId</code> | | “client-1” |
| <code>cleanSession</code> | | true |
| <code>username</code> (optional) | | “hans” |
| <code>password</code> (optional) | | “letmein” |
| <code>lastWillTopic</code> (optional) | | “/hans/will” |
| <code>lastWillQos</code> (optional) | | 2 |
| <code>lastWillMessage</code> (optional) | | “unexpected exit” |
| <code>lastWillRetain</code> (optional) | | false |
| <code>keepAlive</code> | | 60 |

MQTT is Open Source



- Lots of implementations:
 - Mosquitto
 - Micro broker
 - Really small message broker (RSMB): C
 - Cloud broker services



MQTT vs HTTP

- Push delivery of messages / data / events
 - MQTT – low latency push delivery of messages from client to server and **server to client**. Helps bring an event oriented architecture to the web
 - HTTP – push from client to server but poll from server to client
- Reliable delivery over fragile network
 - MQTT will deliver message to QOS even **across connection breaks**
 - Decoupling and publish subscribe – **one to many delivery**

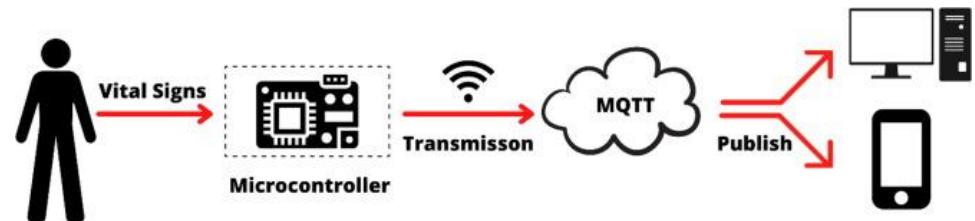
MQTT vs HTTP

| | MQTT | HTTP |
|--------------------|--|---|
| Design orientation | Data centric | Document centric |
| Pattern | Publish/subscribe | Request/response |
| Complexity | Simple | More complex |
| Message size | Small, with a compact binary header just two bytes in size | Larger, partly because status detail is text-based |
| Service levels | Three quality of service settings | All messages get the same level of service |
| Extra libraries | Libraries for C (30 KB) and Java (100 KB) | Depends on the application (JSON, XML), but typically not small |
| Data distribution | Supports 1 to zero, 1 to 1, and 1 to n | 1 to 1 only |

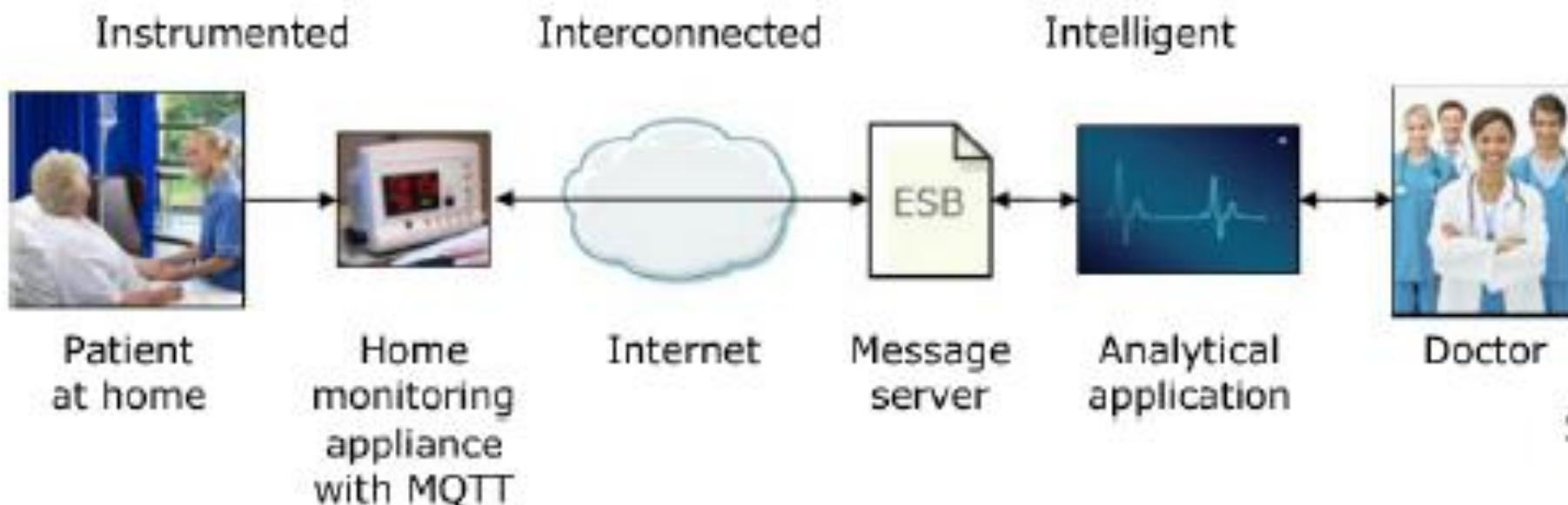
| Characteristics | | 3G | | WIFI | |
|------------------|-------------------------------------|------------|-------------|------------|-------------|
| | | HTTPS | MQTT | HTTPS | MQTT |
| Receive Messages | Messages / Hour | 1,708 | 160,278 | 3,628 | 263,314 |
| | Percent Battery / Hour | 18.43% | 16.13% | 3.45% | 4.23% |
| | Percent Battery / Message | 0.01709 | 0.00010 | 0.00095 | 0.00002 |
| | Messages Received (Note the losses) | 240 / 1024 | 1024 / 1024 | 524 / 1024 | 1024 / 1024 |
| Send Messages | Messages / Hour | 1,926 | 21,685 | 5,229 | 23,184 |
| | Percent Battery / Hour | 18.79% | 17.80% | 5.44% | 3.66% |
| | Percent Battery / Message | 0.00975 | 0.00082 | 0.00104 | 0.00016 |

*sending and receiving 1024 messages of 1 byte each.
 (source: <https://www.ibm.com/developerworks>)

Application Example



- Home care monitoring solution
 - Home and patient instrumented with sensors.
 - E.g. door motion, blood pressure, pacemaker/defib.
 - Collected by monitoring service (broker) using MQTT
 - Subscribed by a health care service in the hospital
 - Alerts relations/health care profs. if anything is out-of-order



Code Example: Python

```
import paho.mqtt.client as mqtt

mqttc = mqtt.Client()

try:
    logging.info("Connecting to " + url_str)
    mqttc.connect(url.hostname, url.port)
    mqttc.loop_start()
except Exception as e:
    logging.error("Connection failed: " + str(e))
    exit(1)

def publish_temperature():
    temp_json = json.dumps({"deviceID": "RPi1", "temperature": 23.4, "timestamp": time.time()})
    mqttc.publish("/fxwalsh/temperature", temp_json)

# Schedule the task every 10 seconds
schedule.every(10).seconds.do(publish_temperature)
```

```
try:
    while True:
        schedule.run_pending()
        time.sleep(1)
except KeyboardInterrupt:
    logging.info("Script termination requested, shutting down.")
finally:
    mqttc.loop_stop()
    mqttc.disconnect()
```

MQTT Security

Frank Walsh

Not good.....

The screenshot shows a list of MQTT messages in a viewer. The top message is from topic /niisten4/862078078713156/off/pub2 with QoS 0. The message content is a binary string of zeros. The timestamp is 2024-02-07 15:46:47.721.

The second message is highlighted with a large red oval. It is from topic /sahuifa with QoS 0. The message content is an SQL insert statement: `insert into tjddcs.t_node1_group1_TAG121212 using tjddcs.stjddcs tags (1) values (1707320807272, 0)`. The timestamp is 2024-02-07 15:46:47.729.

The third message is from topic /neuron/11/modbus-tcp-test/group2 with QoS 0. The message content is a JSON object: `{"node": "modbus-tcp-test", "group": "group2", "timestamp": 1707320792603, "values": {}, "errors": {"tag7": 3002, "tag8": 3002, "tag9": 3002, "tag10": 3002, "tag5": 3002, "tag6": 3002}}`. The timestamp is 2024-02-07 15:46:47.730.

The fourth message is from topic /merakimv/Q2GV-EY5H-6TFX/light with QoS 0. The message content is a JSON object: `{"lux": 646.6}`. The timestamp is 2024-02-07 15:46:47.749.

The fifth message is partially visible at the bottom, from topic /neuron/MQTT/机床/西门子PLC with QoS 0. The message content is not fully shown.

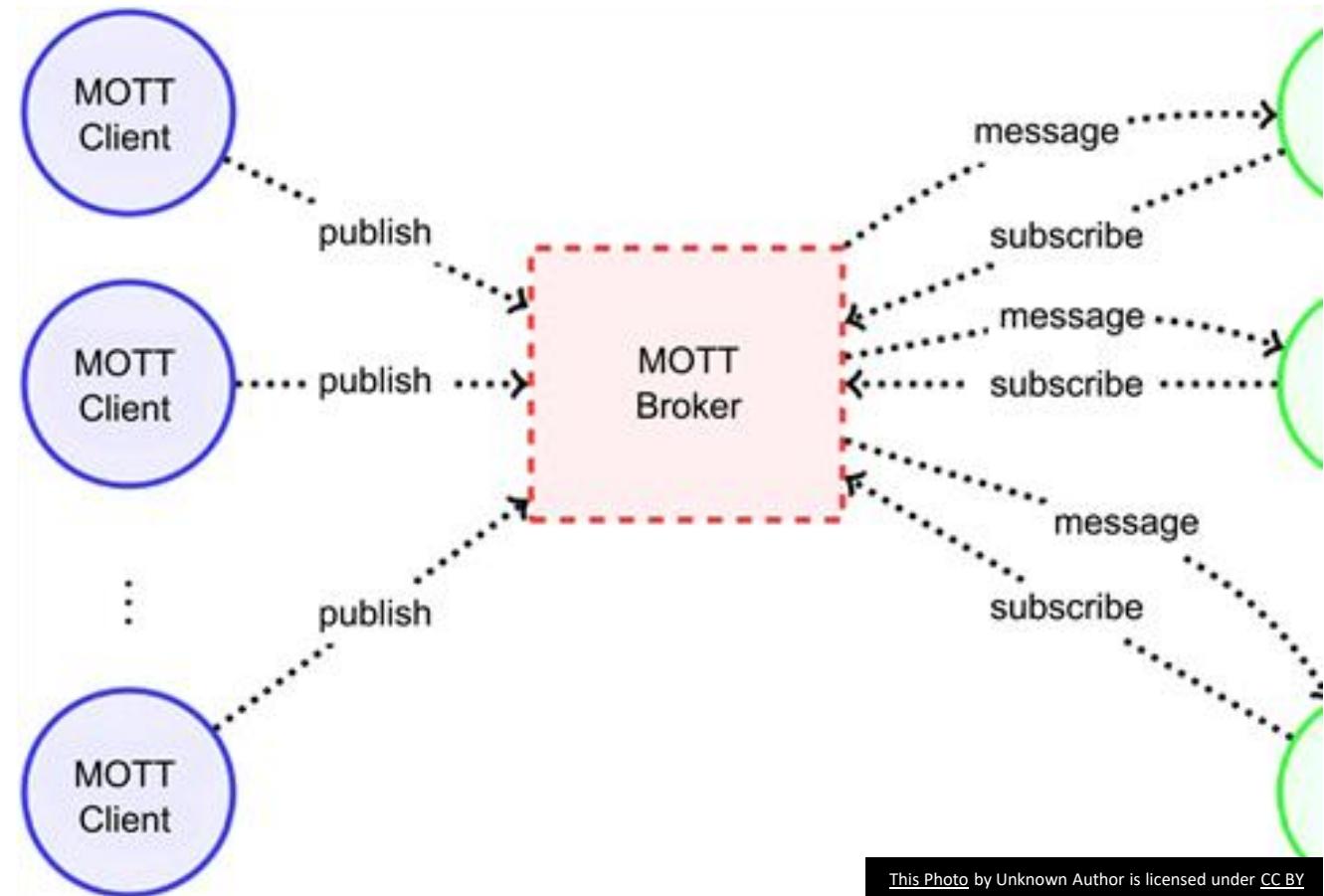


Security in IoT

- Internet of Things presents new security implementation challenges
 - IoT devices have limited computing power and memory capacity
 - Cryptographic algorithms may require more resources than constrained IoT devices possess
 - Critical security issues may require updates to be rolled out to all devices simultaneously can be affected by unreliable networks on which many IoT devices
- **Security is a critical concern for developers of IoT applications.**

Security in MQTT

- We can take a “layered” approach over the protocol stack
- MQTT specifies only a few security mechanisms
- MQTT uses other accepted security standards at the various levels:
 - E.g. SSL/TLS for transport security.
- **Network Layer:** Use a VPN for all communication between clients and brokers. Gateway-based applications, the gateway is connected to devices on the one hand and with the broker over VPN on the other side.
- **Transport Layer:** TLS/SSL
- **Application Layer:** Authorisation using username/password provided by the Protocol



This Photo by Unknown Author is licensed under CC BY

MQTT Authentication

- The MQTT protocol provides username and password fields in the CONNECT message for authentication
 - Username is an UTF-8 encoded string.
 - Password is binary data with a maximum of 65535 bytes.
- The MQTT broker evaluates credentials and returns one of the following return codes:
 - 0 – Connection Accepted
 - 4 – Connection Refused, bad username/password
 - 5 – Connection Refused, not authorised
- Username/password sent as plain text - **Secure transmission of usernames and passwords requires transport encryption**

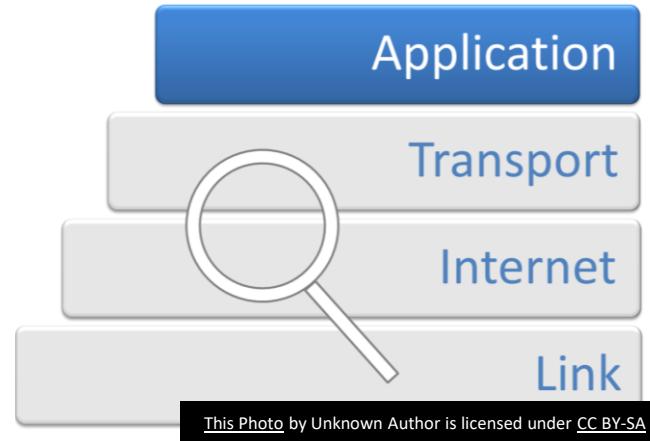
Transport Layer Security (TLS)

- TLS provide a secure communication channel between a client and a server **at Transport Layer**.
- TLS is a cryptographic protocol which use a handshake mechanism to negotiate various parameters to create a secure connection between the client and the server.
- **Servers provide a certificate** (typically issued by a trusted authority) that clients use to verify the identity of the server.
- While the additional CPU usage is typically negligible on the broker, **small constrained devices are not designed for computation-intensive tasks**.
- If TLS is not possible, **payload encryption and at least hash or encrypt the password in the CONNECT message of your client**.



Application Level Encryption

- End to End encryption between publisher and subscriber
- Encrypt the payload without having to configure the MQTT broker
 - Means publisher and subscriber applications have to “know” encryption/decryption method.
Has to be coded
- Possible alternative for constrained devices that cannot support TLS
- Possible (less secure) alternative for using public brokers.



```
from cryptography.fernet import Fernet

def encrypt_payload(payload):
    #HAVING A HARD CODED KEY IS INSECURE - SHOULD BE ENV/EXTERNAL VARIABLE
    cypher_key=b'xqi9zRusHkcv30m050HwX82eMTO-LbeW4YlqVVEzpw8='
    cypher=Fernet(cypher_key)
    encrypted_payload=cypher.encrypt(payload.encode('utf-8'))
    return(encrypted_payload.decode())
```

```
mqttc.publish(base_topic+ "/temperature", encrypt_payload(temp_json))
```

Raspberry Pi: Creating MQTT Clients in Python: Publishing Client

- We'll use **Paho** MQTT python client from Eclipse
- Import the **client class** to provide functions to publish messages and subscribe to topics on MQTT brokers.
- Create instance and connect to a broker:

```
import paho.mqtt.client as mqtt #import the client1
broker_address="192.168.1.184"
#broker_address="iot.eclipse.org" #use external broker
client = mqtt.Client("P1") #create new instance
client.connect(broker_address) #connect to broker
client.publish("house/main-light","OFF")#publish
```

Raspberry Pi: Creating MQTT Clients in Python: Subscribing Client Callback

- When the client receives messages it generate the `on_message` callback.

```
import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print("message received " ,str(message.payload.decode("utf-8")))
    print("message topic=",message.topic)
    print("message qos=",message.qos)

def main():
    broker_address="192.168.1.184"
    client = mqtt.Client("P1") #create new instance
    client.connect(broker_address) #connect to broker
    client.subscribe("house/main-light")

if __name__ == "__main__":
    main()
```

Arduino: Creating MQTT Clients in a Sketch: Publishing Client

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

Uses PubSubClient from Arduino Library

```
WiFiClient wifiClient;  
PubSubClient client(wifiClient);
```

Construct MQTT client using wifiClient

```
client.setServer(mqttServer, mqttPort);
```

Set MQTT Server and port

```
client.connect(connectionId)
```

Create Connection

```
client.publish("/fxwalsh/temperature", message);
```

Publish message to topic!

Javascript/Node.js MQTT Client

```
const mqtt = require('mqtt');

const client  = mqtt.connect('mqtt://broker.emqx.io');
const topic = '/fxwalsh/temperature';

client.on('connect', function () {
  client.subscribe(topic, function (err) {
    if (!err) {
      console.log(`Subscribed to ${topic}`);
    }
  });
});

client.on('message', function (topic, message) {
  // message is a buffer
  console.log(message.toString());
});
```