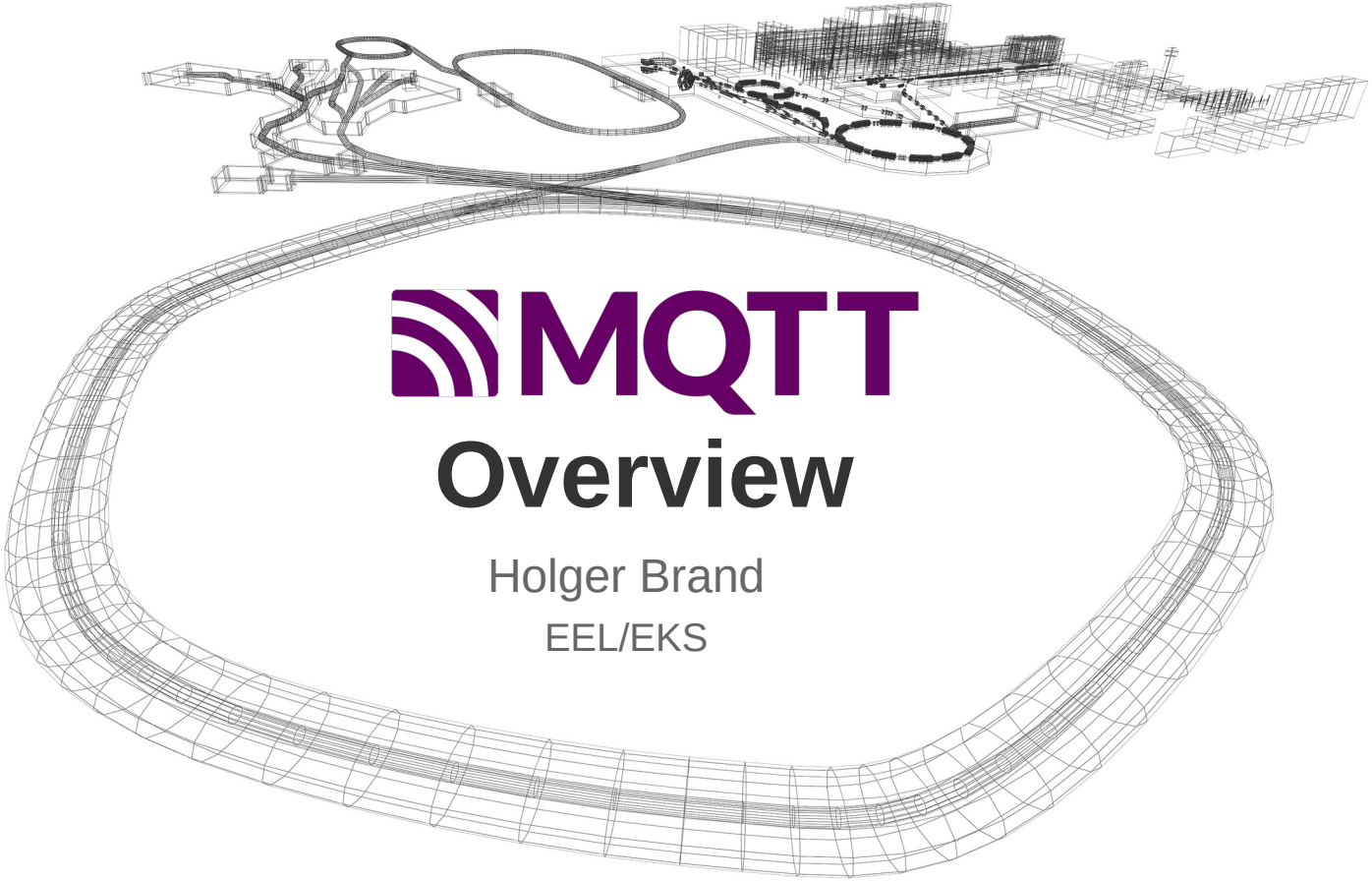


MQTT
Overview

Holger Brand
EEL/EKS

Literature

- MQTT https://mqtt.org/
  - Getting started, Specifications, Software, Use Cases, FAQ
- Introduction by Stephen Cope
  - Most MQTT related text and images copied from http://www.steves-internet-guide.com
    - With kind courtesy of Stephen Cope.
  - eBooks
    - MQTT for Complete Beginners - Learn the Basics of the MQTT Protocol
      - ASIN: B08HDCWDSG
    - WORKING WITH THE PAHO MQTT CLIENT
      - ASIN: B08M48B2T6
    - THE MOSQUITTO BROKER FOR COMPLETE BEGINNERS
      - ASIN: B09DVN9CDZ

What is MQTT?

- MQTT is a lightweight publish/subscribe messaging protocol designed for M2M (machine to machine) telemetry in low bandwidth environments.
- It was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 for connecting Oil Pipeline telemetry systems over satellite.
- Although it started as a proprietary protocol it was released Royalty free in 2010 and became an OASIS standard in 2014.
- MQTT is fast becoming one of the main protocols for IOT (internet of things) deployments.

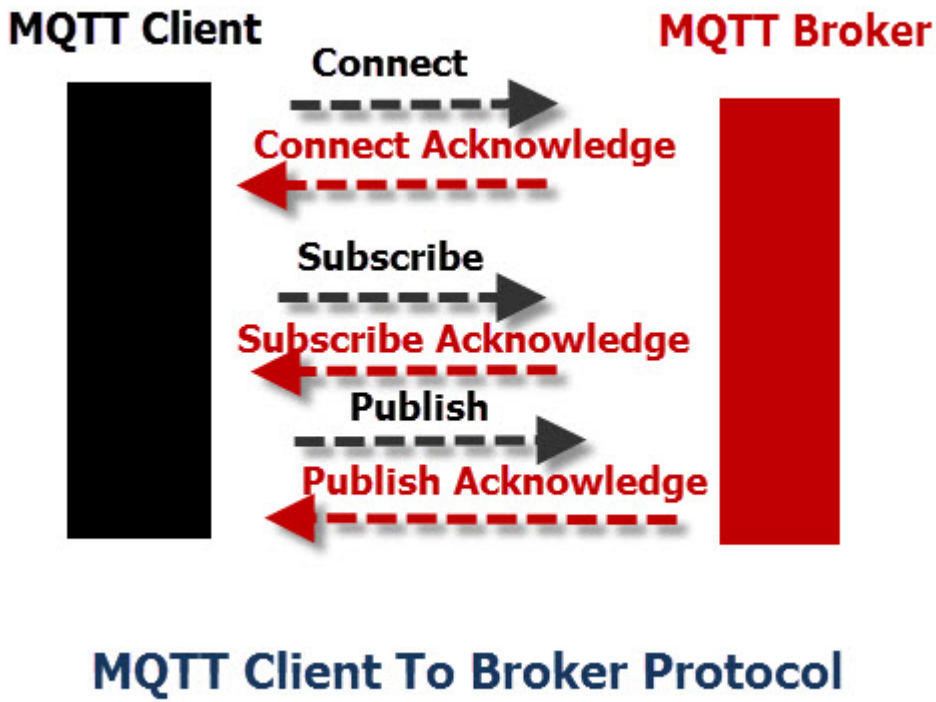
MQTT Versions

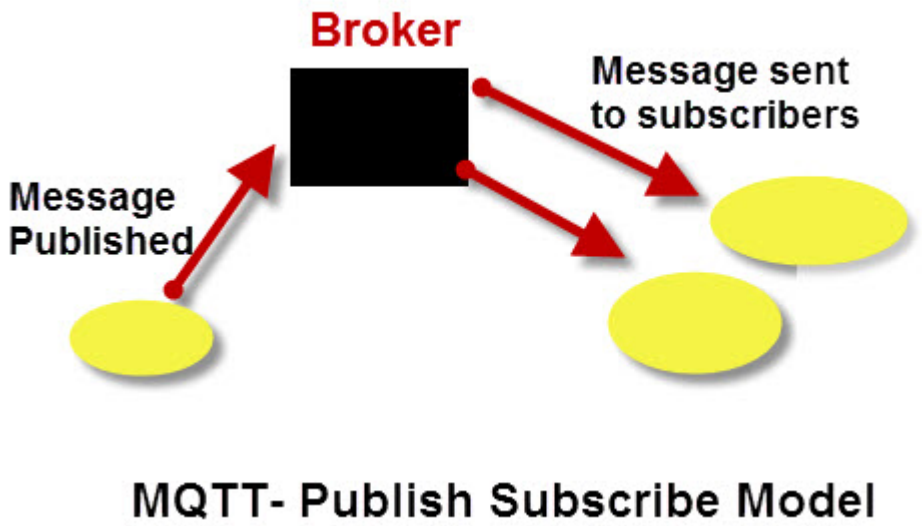
- MQTT v3.1.1 – In Common Use
- MQTT v5 – Currently Limited use
- The original MQTT which was designed in 1999 and has been in use for many years and is designed for TCP/IP networks.
  - Web Sockets have been added.

Introduction

- MQTT is a binary based protocol were the control elements are **binary bytes** and **not text strings**.
- MQTT uses a **command** and **command acknowledgment** format.
  - That means each command has an associated acknowledgement.
- Topic names, Client ID, User names and Passwords are encoded as UTF-8 strings.
- The Payload excluding MQTT protocol information like Client ID etc is binary data and the content and format is application specific.

MQTT Client to Broker Protocol





How MQTT Works -Beginners Guide

- MQTT is a messaging protocol i.e it was designed for transferring messages, and uses a publish and subscribe model.
- In MQTT a publisher publishes messages on a topic and
- A subscriber must subscribe to that topic to view the message.
- MQTT **requires the use of a central Broker** as shown in the diagram on previous page.

Important Point to Note

- Clients do not have addresses like in email systems, and messages are not sent to clients.
- Messages are published to a broker on a topic.
- The job of an MQTT broker is to filter messages based on topic, and then distribute them to subscribers.
- A client can receive these messages by subscribing to that topic on the same broker
- **There is no direct connection between a publisher and subscriber.**
- **All clients can publish (broadcast) and subscribe (receive).**
- **MQTT brokers do not normally store messages.**
  - Refer to QoS-Level and Retain-Flag.

MQTT Client-Broker Connections

- MQTT uses TCP/IP to connect to the broker.
- TCP is a connection orientated protocol with error correction and guarantees that packets are received in order.
  - You can consider a TCP/IP connection to be similar to a telephone connection.
  - Once a telephone connection is established you can talk over it until one party hangs up.
- Most MQTT clients will connect to the broker and remain connected even if they aren’t sending data.
- Connections are acknowledged by the broker using a Connection acknowledgement message.
  - You cannot publish or subscribe unless you are connected.

Client Name or Client ID

- All clients are required to have a client name or ID.
- The client name is used by the MQTT broker to track subscriptions etc.
- **Client names must also be unique.**
  - If you attempt to connect to an MQTT broker with the same name as an existing client then the existing client connection is dropped.
  - Because most MQTT clients will attempt to reconnect following a disconnect this can result in a loop of disconnect and connect.

Clean Sessions

- MQTT clients by default establish a clean session with a broker.
- A clean session is one in which the broker isn’t expected to remember anything about the client when it disconnects.
- With a non clean session the broker will remember client subscriptions and may hold undelivered messages for the client.
- However this depends on the Quality of service used when subscribing to topics, and the quality of service used when publishing to those topics.

Last Will Messages

- The idea of the last will message is to notify a subscriber that the publisher is unavailable due to network outage.
- The last will message is set by the publishing client, and is set on a per topic basis which means that each topic can have its own last will message.
- This means that each topic can have its own last will message associated with it.
- The message is stored on the broker and sent to any subscribing client (to that topic) if the connection to the publisher fails.
  - If the publisher disconnects normally the last Will Message is not sent.
- The actual will messages is including with the connect request message.

MQTT Over WebSockets

- Web-socket is a computer communications protocol, providing full-duplex communication channels over a single TCP/IP connection. Wiki
- It is closely associated with http as it uses http for the initial connection establishment..
- The client and server connect using http and then negotiate a connection upgrade to web-sockets, the connection then switches from http to web-sockets
- The client and server can now exchange full duplex binary data over the connection.Web-sockets allows you to receive MQTT data directly into a web browser.
- This is important as the web browser may become the DE-facto interface for displaying MQTT data.
- MQTT web-socket support for web browsers is provided by the Javascript MQTT Client.

MQTT Security

- MQTT supports various authentications and data security mechanisms.
- It is important to note that these security mechanisms are configured on the MQTT broker, and it is up to the client to comply with the mechanisms in place.
- Client Authentication
  - There are three ways that a Mosquitto broker can verify the identity of an MQTT client:
    - Client ids
    - Usernames and passwords.
      - The username/password combination is transmitted in clear text and is not secure without some form of transport encryption.
      - The username used for authentication can also used in restricting access to topics.
      - On the Mosquitto broker you need to configure two settings for this to work. Again you will find these settings in the security section of the mosquitto.conf file*.
    - Client Certificates*
- Securing Data*

* Refer to documentation

Publishing Topics and Subscription

- Publishing to Topics
  - A client can only publish to an individual topic. That is, using wildcards when publishing is not allowed.
  - E.g. to publish a message to two topics you need to publish the message twice.
- When are Topics Created?
  - Topics are created dynamically when:
  - Someone subscribes to a topic
  - Someone publishes a message to a topic with the retained message set to True.
- When are Topics Removed from a Broker?
  - When the last client that is subscribing to that broker disconnects, and clean session is true.
  - When a client connects with clean session set to True.

Using Topic Wildcards # and +

- Subscribing to topic house/# covers
  - house/room1/main-light
  - house/room1/alarm
  - house/garage/main-light
  - house/main-door
  - **but doesn’t cover**
    - house/
    - House
- Subscribing to topic house/**+**/main-light covers
  - house/room1/main-light
  - house/room2/main-light
  - house/garage/main-light
  - **but doesn’t cover**
    - house/room1/side-light
    - house/room2/side-light

Simple Topic Guidelines

- Because topics are case sensitive use lower case only.
- Separate command and response topics using a prefix e.g. command/ and response/
- Don’t use topics with spaces.
- Use letters numbers and dashes only
- Include routing information in the topic structure.

MQTT Quality of Service (QoS)

- MQTT provides 3 QoS levels
  - QoS 0 – Once (not guaranteed)
  - QoS 1 – At Least Once (guaranteed)
  - QoS 2 – Only Once (guaranteed)
- The QoS levels are a way of guaranteeing message delivery and they refer to the connection between a broker and a client.

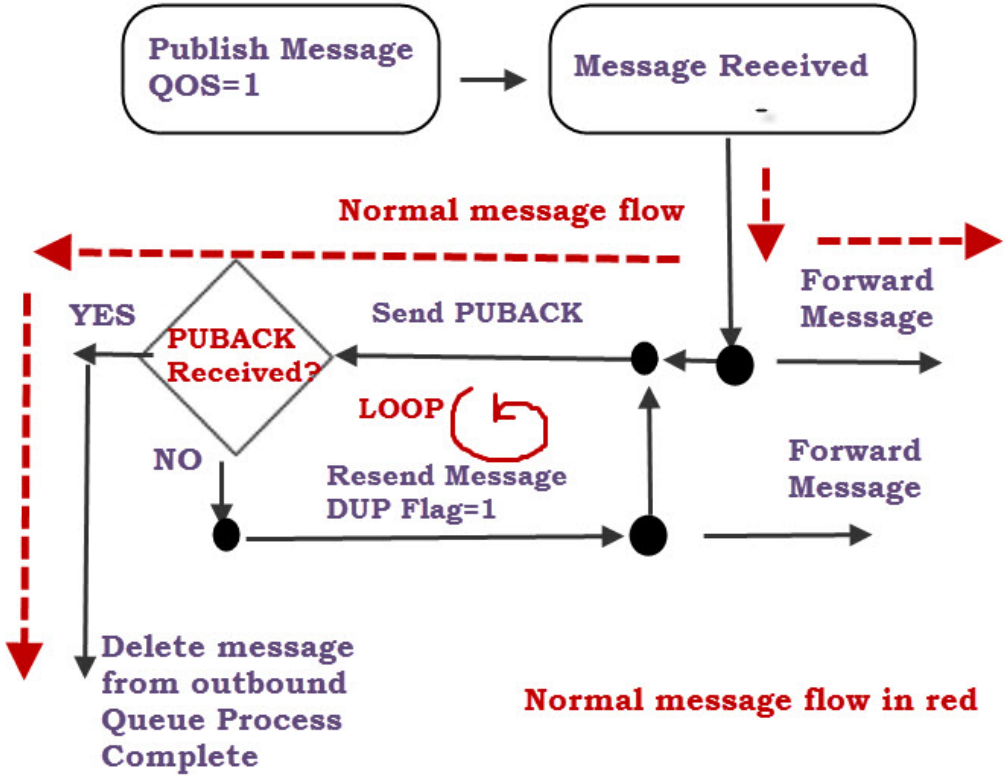
MQTT QoS 0 - Once

- This is the fastest method and requires only 1 message.
  - It is also the most unreliable transfer mode.
- The message is not stored on the sender, and is not acknowledged.
- The message will be delivered **only once, or not at all**.
- Once the message has been sent by the client it is deleted from the outbound message queue.
  - Therefore with this QoS level there is no possibility of duplicate messages.

MQTT QoS 1 – At Least Once

- This level guarantees that the message will be delivered at **least once, but may be delivered more than once**. (See Flow Diagram on right.)
- Publishing with **QoS of 1 requires 2 messages**.
- The sender sends a message and waits for an acknowledgement (PUBACK).
  - If it receives an acknowledgement then it notifies the client app, and deletes the message from the outbound queue.
  - If it doesn't receive an acknowledgement it will resend the message with the DUP flag set (Duplicate Flag).
  - The message will continue to be resent at regular intervals, until the sender receives an acknowledgement.
  - If the message is being sent to the broker then the broker will forward that message to subscribers even though the duplicate flag is set.
  - Therefore subscribers can receive the same message multiple times.

MQTT QOS 1 – At Least Once - Message Flow

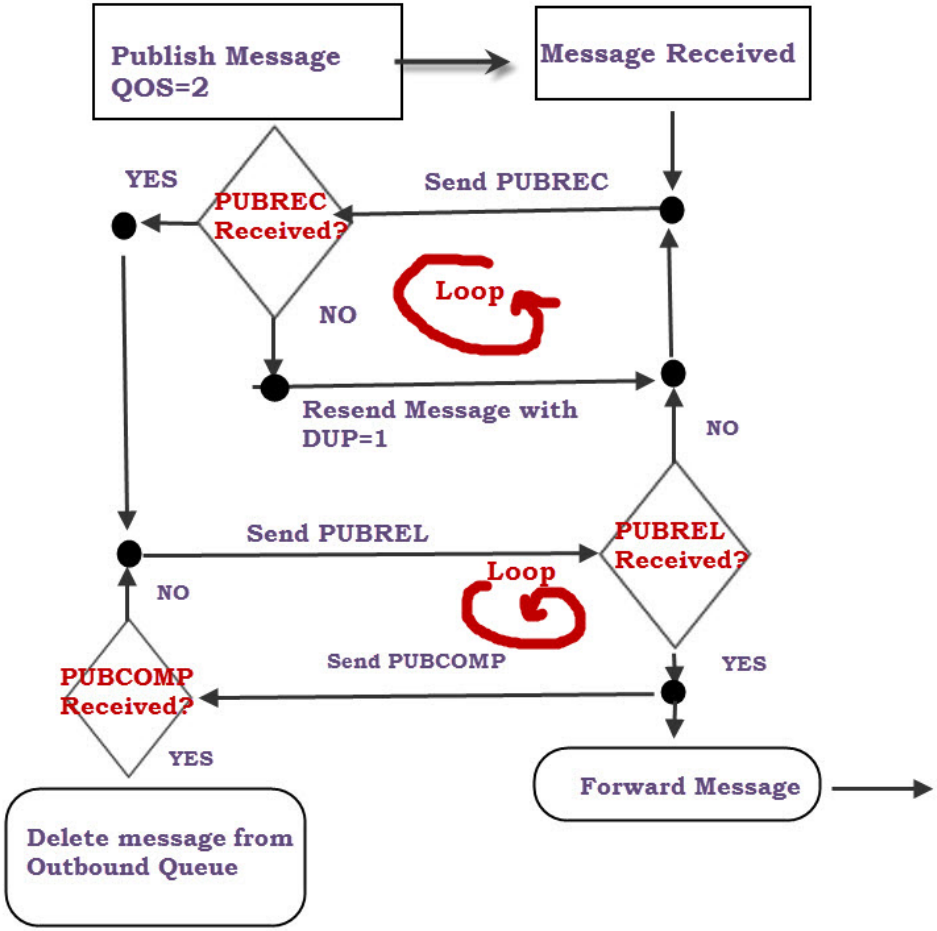


MQTT QOS 1 Message Flow Diagram

MQTT QoS 2 – Only Once

- This level guarantees that the message will be delivered only once.
  - This is the slowest method as it requires 4 messages.
- Message flow:
  1) The sender sends a message and waits for an acknowledgement (PUBREC)
  2) The receiver sends a PUBREC message
     1) If the sender doesn't receive an acknowledgement (PUBREC) it will resend the message with the DUP flag set.
     2) When the sender receives an acknowledgement message PUBREC it then sends a message release message (PUBREL). The message can be deleted from the queue.
        1) If the receiver doesn't receive the PUBREL it will resend the PUBREC message
        2) When the receiver receives the PUBREL message it can now forward the message onto any subscribers.
        3) The receiver then send a publish complete (PUBCOMP) .
        4) If the sender doesn't receive the PUBCOMP message it will resend the PUBREL message.
        5) When the sender receives the PUBCOMP the process is complete and it can delete the message from the outbound queue, and also the message state.

MQTT QOS 2 – Only Once – Message Flow



MQTT QOS 2 Message Flow Diagram

MQTT Retain Flag

- Normally if a publisher publishes a message to a topic, and no one is subscribed to that topic the message is simply discarded by the broker.
- However the publisher can tell the broker to keep the last message on that topic by setting the retained message flag=True.
  - This can be very useful, as for example, if you have sensor publishing its status only when changed e.g. Door sensor. What happens if a new subscriber subscribes to this status?
  - Without retained messages the subscriber would have to wait for the status to change before it received a message.
  - However with the retained message the subscriber would see the current state of the sensor.
- **What is important to understand is that only one message is retained per topic.**
  - The next message published on that topic replaces the last retained message for that topic.

Table: Clean Session, Retain Message, QOS

Retain Message, Clean Session and QOS Table

Clean Session Flag	Retain Flag	Subsribe QOS	Publish QOS	Published Message Always Reeeived
True	False	0	0	No
True	False	0	1	No
True	False	1	0	No
True	False	1	1	No
False	False	0	0	No
False	False	0	1	No
False	False	1	0	No
False	False	1	1	Yes – All messages
True	True	0	0	Yes –Last Message only
True	True	0	1	Yes –Last Message only
True	True	1	0	Yes –Last Message only
True	True	1	1	Yes –Last Message only
False	True	0	0	Yes –Last Message only
False	True	0	1	Yes –Last Message only
False	True	1	0	Yes –Last Message only
False	True	1	1	Yes – All messages

Note: QOS 1 and QOS 2 produce same result. Therefore for QOS 1 in the table read 1 or 2.

Eclipse Paho Project

- A popular client library is provided by the Eclipse Paho libraries.

MQTT Client Comparison

Client	MQTT 3.1	MQTT 3.1.1	MQTT 5.0	LWT	SSL / TLS	Automatic Reconnect	Offline Buffering	Message Persistence	WebSocket Support	Standard MQTT Support	Blocking API	Non-Blocking API	High Availability
Java	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Python	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔	✖
JavaScript	✔	✔	✖	✔	✔	✔	✔	✔	✔	✖	✖	✔	✔
GoLang	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
C	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
C++	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rust	✔	✔	✖	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔
.Net (C#)	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔	✖
Android Service	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
Embedded C/C++	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✔	✔	✖

MQTT Tools and Resources I

- Command Line Tools
  - These tools are part of the mosquitto broker install on Windows, but are a separate install on Linux (Raspberry Pi).
  - Mosquitto_sub – Simple subscribe utility for testing. This comes with the Mosquitto broker.
  - Mosquitto_pub – Simple publish utility for testing. This comes with the Mosquitto broker.
- MQTT Ping
  - This test tool is a simple python script that tests if a broker is up and how long messages take, just like the traditional IP ping utility.
  - The script requires only a single parameter which is the broker IP or domain name.Type python mqtt-ping.py -h
  - ```
    C:\Python34\steve\mqtt-tools\mqtt-ping.py  -b <broker> -p <port> -t <topic>
     -c <count> -d <delay> -u <username> -P <pasword>-s <silent True>
    ```
    ```

MQTT Tools and Resources II

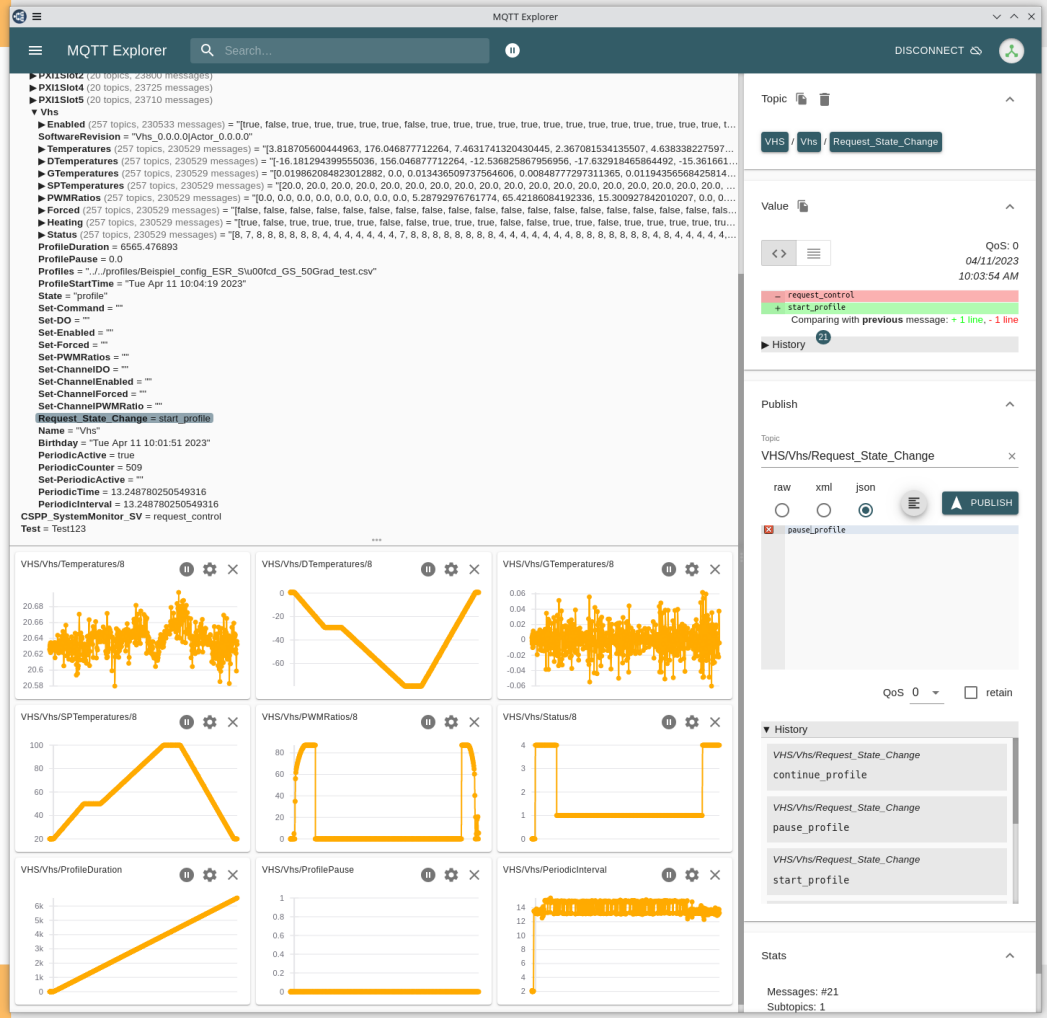
- MQTT- Monitor -Monitors Selected topics
  - This is very useful tool for monitoring topics on a broker. I use it frequently when testing.
  - By default it only display changed messages.
  - Usage: python mqtt-monitor.py -h
  -

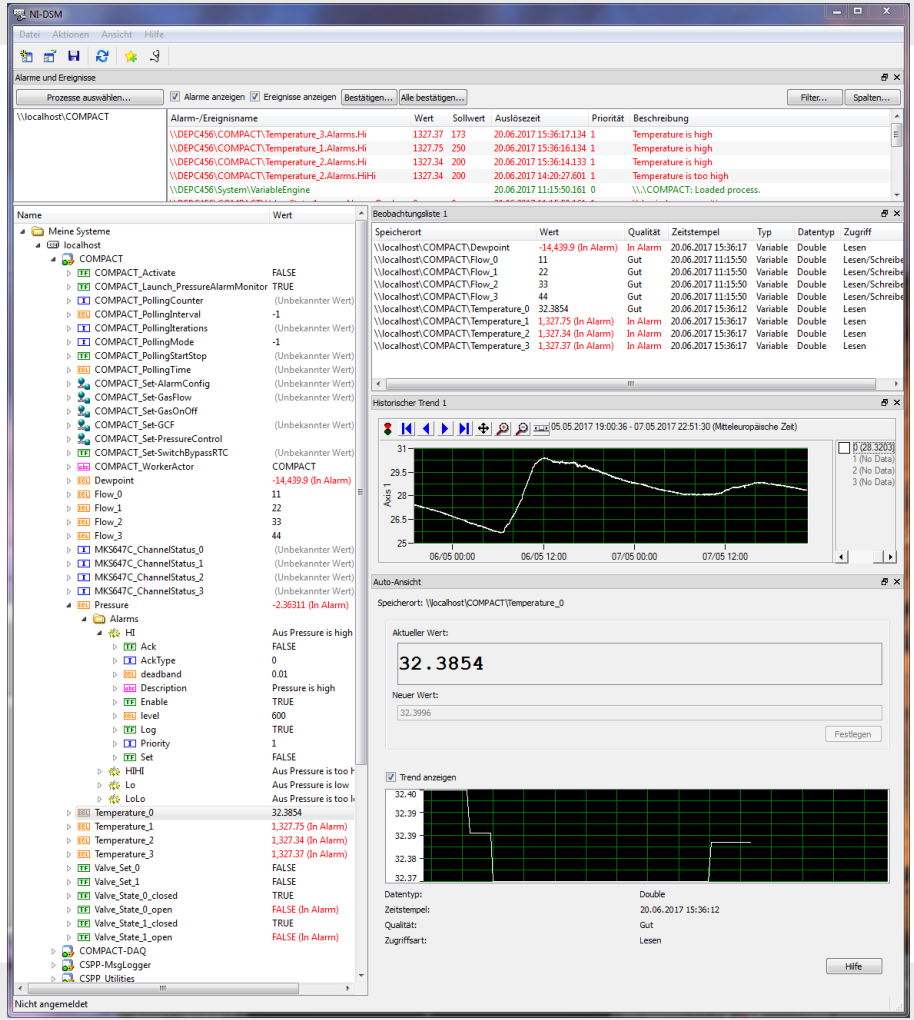
```
C:\Python34\steve\mqtt-tools>python mqtt-monitor.py -h
mqtt-monitor.py  -b <broker> -p <port>-t <topic> -q QOS -v <verbose yes/no> -h <
help>-c <loop Time secs -d logging debug yes/no> >
```

MQTT Explorer

- MQTT Explorer– Explore MQTT topic structure,graph data, delete retained messages, send and receive messages. Supports web-sockets
  - Windows, Linux and Mac Apps
- http://mqtt-explorer.com/
- Author: Thomas Nordquist
  - *MQTT Explorer is a comprehensive MQTT client that provides a structured overview of your MQTT topics and makes working with devices/services on your broker dead-simple.*
- Substitute for NI Distributed System Manager

MQTT Explorer - NI Distributed System Manager





Mosquitto Configuration

- Following a selection of configuration entries
  - assuming the broker is behind a firewall
  - you do not need to fear sabotage
  - Logging
    - connection_messages true
      - If set to true, client connection and disconnection messages will be included in the log.
    - log_timestamp true
      - If set to true, add a timestamp value to each log message.
    - log_timestamp_format %Y-%m-%dT%H:%M:%S
  - Remote access
    - allow_anonymous true
      - Defaults to false, unless there are no listeners defined in the configuration # file, in which case it is set to true, but connections are only allowed from the local machine.
  - password_file /etc/mosquitto/passwd

Mosquitto – Start in Docker

- Thanks to P. Zumbruch for help.
- Set Export environment variables

```
export mosquitto_ListenerPort=1883
export mosquitto_WebSocketPort=9001
export mosquitto_User=eks
workingDir=/home/$USER/Container/Mosquitto-PORT-${mosquitto_ListenerPort:?}-${mosquitto_WebSocketPort:?}-${mosquitto_User:?}
export mosquitto_configDir=/home/$USER/Container/Mosquitto/config && \
export mosquitto_logDir=/home/$USER/Container/Mosquitto/log && \
export mosquitto_dataDir=/home/$USER/Container/Mosquitto/data
```

- Preparation

```
source /home/$USER/Container/Mosquitto/MosquittoEnv.sh
mkdir -p $mosquitto_configDir $mosquitto_dataDir $mosquitto_logDir
cp mosquitto_default.conf ${mosquitto_configDir}/mosquitto.conf
# Create passwd
touch $mosquitto_configDir/passwd &&
sudo docker run --rm \
  --name mosquitto-port${mosquitto_ListenerPort:?}_${mosquitto_WebSocketPort:?}-${mosquitto_User:?} \
  -it \
  -v $mosquitto_configDir/passwd:/mosquitto/config/passwd \
  Eclipse-mosquitto \
  mosquitto_passwd -c /mosquitto/config/passwd ${mosquitto_User:?}
echo " Stop container with: \'sudo docker container stop mosquitto-port${mosquitto_ListenerPort:?}_${mosquitto_WebSocketPort:?}-${mosquitto_User:?}\'"
```

- Run Mosquitto

```
source /home/$USER/Container/Mosquitto/MosquittoEnv.sh
sudo docker or better podman run --rm \
  --name mosquitto-port${mosquitto_ListenerPort:?}_${mosquitto_WebSocketPort:?}-${mosquitto_User:?} \
  -d  -p ${mosquitto_ListenerPort:?}:1883 \
  -p ${mosquitto_WebSocketPort:?}:9001 \
  -v ${mosquitto_configDir}/mosquitto.conf:/mosquitto/config/mosquitto.conf:ro \
  -v ${mosquitto_configDir}/passwd:/etc/mosquitto/passwd:ro \
  -v ${mosquitto_dataDir}:/var/lib/mosquitto/data \
  -v ${mosquitto_logDir}:/mosquitto/log \
  --network=mqtt_network \
  Eclipse-mosquitto \
  mosquitto -v -c /mosquitto/config/mosquitto.conf
```

Example: pyacdaq.actor.mqtt:Mqtt class

https://git.gsi.de/EKS/Python/ACDAQ/PyAcdaq

```
mqtt_proxy = Mqtt.start('Mqtt',
                        start_periodic=True,
                        mqtt_actor=None,  # It is the MQTT actor.
                        broker=mqtt_broker,
                        port=1883,
                        client_id='unique_client_id',
                        clean_session=True,
                        retain=True,
                        will='offline',
                        username=mqtt_user,
                        password=mqtt_password
                        ).proxy()
```

Mqtt._on_start(self): -> Mqtt.__connect_broker(self):

```
self._client = mqtt.Client(self._client_id, self._clean_session, userdata={'actor_ref': self})
        self._client.connected_flag = False
        if self._username is not None and self._password is not None:
            self._client.username_pw_set(username=self._username, password=self._password)
        self._client.will_set(self._client_id, self._will, 1, False)
        # bind call back function
        self._client.on_connect = on_connect
        self._client.on_disconnect = on_disconnect
        # self._client.on_log = on_log
        self._client.on_publish = on_publish
        self._client.on_message = on_message
        self._client.on_subscribe = on_subscribe
        self._client.on_unsubscribe = on_unsubscribe
        self._client.loop_start()
        self._client.connect(self._broker, self._port, keepalive=60, bind_address='')
```

Mqtt.subscribe_topics(self, subscriber, topics, qos=0):

- rcs_mids = (())
- for **topic** in **topics**:
- **prefixed_topic** = self._client_id + '/' + **topic**
- **if prefixed_topic not in self._subscriptions:**
- self._subscriptions.update({prefixed_topic: set((**subscriber**,))})
- else:
- **subs** = self._subscriptions[**prefixed_topic**]
- subs.add(**subscriber**)
- self._subscriptions.update(**{prefixed_topic: subs}**)
- rc, mid = **self._client.subscribe(prefixed_topic, qos=0)**
- rcs_mids = rcs_mids + ((rc, mid),)

Mqtt.publish_topic(self, topic, value, qos=0, retain=False, expand=False):

```
cs_mids = (())
prefixed_topic = self._client_id + '/' + topic
if (isinstance(value, list | tuple | set | numpy.ndarray)):  # Performance improvement proposed by Bjorn Schmitt, B.Schmitt@gsi.de
    rc, mid = self._client.publish(prefixed_topic, json.dumps(value.tolist()), qos, retain or self._retain)
else:
    rc, mid = self._client.publish(prefixed_topic, json.dumps(value), qos, retain or self._retain)
    rcs_mids = rcs_mids + ((rc, mid),)
    if expand and (isinstance(value, list | tuple | set | numpy.ndarray)):  # Performance improvement proposed by Bjorn Schmitt, B.Schmitt@gsi.de
        for index, val in enumerate(value):
            sub_topic = prefixed_topic + '/' + str(index)
            if isinstance(val, numpy.bool_):
                rc, mid = self._client.publish(sub_topic, bool(val), qos, retain or self._retain)
            elif isinstance(val, numpy.short | numpy.ushort | numpy.intc | numpy.uintc | numpy.int_ | numpy.uint | numpy.longlong | numpy.ulonglong):
                rc, mid = self._client.publish(sub_topic, int(val), qos, retain or self._retain)
            elif isinstance(val, numpy.half | numpy.float16 | numpy.single | numpy.double | numpy.longdouble):
                rc, mid = self._client.publish(sub_topic, float(val), qos, retain or self._retain)
            else:
                rc, mid = self._client.publish(sub_topic, val, qos, retain or self._retain)
```

mqtt callback on_message(client, userdata, msg) → Mqtt._on_receive(self, message):

```
match message['mqtt']:  # Messages from MQTT callback functions.
    case 'on_connect':
        self.on_connect()
        for subscriber in self._subscribers:
            subscriber.on_connect()
    case 'on_disconnect':
        for subscriber in self._subscribers:
            subscriber.on_disconnect()
     case 'on_message':
         for subscriber in self._subscriptions[message['topic']]:  # send to topic subscribers only
             subscriber.on_message(message['topic'], message['msg'])
     case _:  # Call super
         super()._on_receive(message)
     pass
```

Discussion

- MQTT is used by many groups in HGF
- Experience with MQTT?
  - LabVIEW Implementations to enable smooth migration?
    - All implementation that I tried have some serious problems.
      - https://github.com/LabVIEW-Open-Source/MQTT-Client
        - Handles QoS > 0 synchronous which is a performance killer.
        - After a while: *„Not enough memory to complete operation.“*
          - Maybe message pileup in queue, but difficult to debug.
    - Can you recommend a working implementation with good performance mybe used as PVConnection in **CS++**?
    - Migration of LabVIEW based Vacuum Heating System successful.
      - Python / pykka / pyacdaq / nidaqmx / simpful / telegraf / InfluxDB / Grafana
      - Commissioning planned for May 2023