Recap asyncio

- Asyncio is a simple framework allowing for **cooperative scheduling**
  - The activities of the program are executed in coroutines running in tasks
  - Coroutines yield (=give up) execution control when they have to wait for some input or output
  - The Event Loop decides which task is run next, when the current task yields.
  - The programmer decides when each coroutine yields
  - However, Asyncio cannot change the fact that in python only one thread can run at a time

Recap: Web Server

- Web Server: Basics of HTTP
  - The HTTP protocol is defined by Requests and Responses
  - Request:
    - Different classes of requests exist: GET and POST are mostly used for web servers
    - A series of ASCII lines form a header
    - Followed by an empty line
    - Some more ASCII data might follow (not the case for simple GET request, but for POST requests)
  - Response:
    - A header with some ASCII lines
    - Followed by an empty line
    - Usuall followed by some data (e.g. the web page to be displayed by the browser)

Exercises

- Always remember:
  - What you need in your program must be uploaded to the microctontroller :

    mpremote cp source :.

- The file style.css for the exercise (to nicely format the page) is now also included in the code template
- One of you yesterday had a nice idea for improvement:
  - It would be better to have the web page also in a separate file and be served with the “sendFile” function
    - Allows for easier editing of the web page contents
    - Cleanly separates web content from program code
    - Need to provide techniques to replace some parts of the web page dynamically (i.e. before sending the page to the browser): in our case the sensor values have to be updated.

Microcontrollers

Part VI

MQTT basics

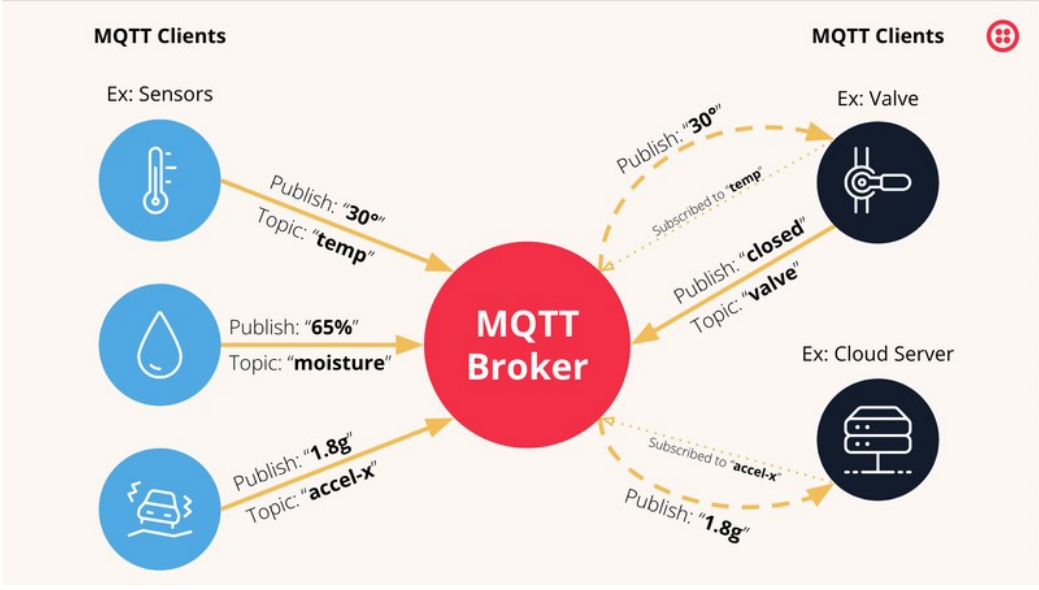
(Christoph.Schwick@cern.ch)

MQTT

- MQTT was originally developed by IBM to monitor oil pipelines over a satellite
- Requirements:
  - Very robust
  - Simple (especially for the senors which need to send their data: they are small devices like microcontrollers)
  - No high data bandwidth required.
- MQTT is an application protocol: it needs a reliable underlying network protocol.
  - Usually TCP/IP is used.

MQTT Basics

- MQTT is a “Publisher Subscriber” Protocol
  - Requires a central entity: the “**broker**”
  - Each client connects to the broker
  - Decoupling of data senders and consumers → **Robustness**
- Messages are sent in “**topics**”
  - They have a topic name (see later)
  - They have payload (the real data)
  - Data can be anything
- Clients **subscribe** to topics they are interested in
- Clients “**publish** topics” to the broker
  - They will then be delivered to all subscribers of the topic
- Clients can joint a leave the network at any time without the system breaking down
  - However, the broker needs to stay up and running



MQTT topics

- Published messages (=data) are called topics and have an unique topic-names
- Topics are organised in a hierarchical structure (via there names)
  - The names are written like a directory path e.g.:
    - home/living-room/light
    - home/living-room/store
    - home/bedroom/temperature
- Topic names are used to publish messages and to subscribe to messages
- When subscribing to topics you can use **wildcards**:
  - home/+/temperature : means subscription to temperature values of all rooms (assuming that the second component of the topic is the room name; the '+' is the placeholder for "any single subtopic" at this point in the hierarchy.) → The '+' stands for anything except for a '/'.
  - home/# : means subscription to all messages from "home/{anything}"
    The '#' has to be always the last character in the expression, if used.
    The '#' stands for anything possibly including '/' and subtopics

MQTT : QoS

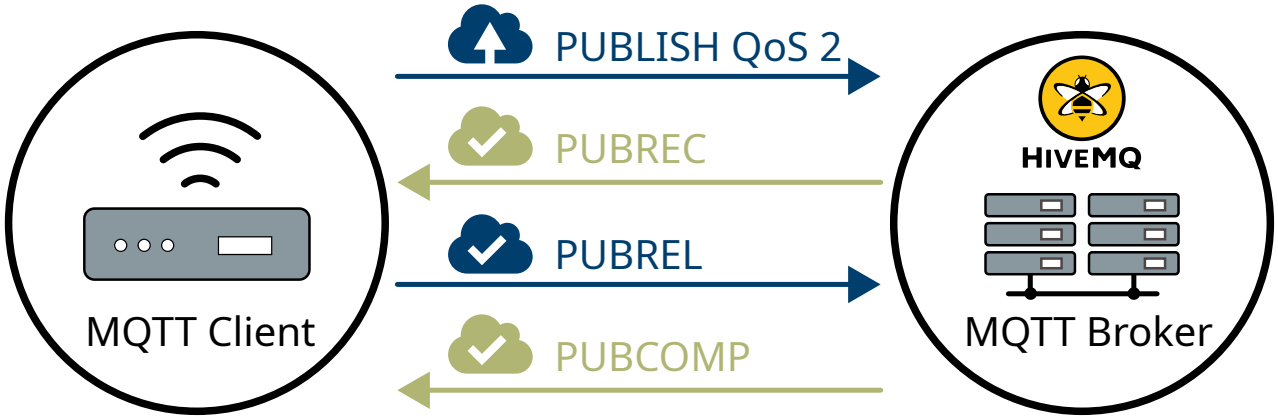
- There are three different QoS levels in MQTT (0,1, 2):
  - QoS = 0: “Fire and Forget” → A message is sent to the broker and from the broker to the client and it is “assumed to arrive”. If for any reason the message does not arrive (e.g. the client has a problem) the message is lost for that client.

MQTT : QoS

- There are three different QoS levels in MQTT (0, 1, 2):
  - QoS = 1: The message is guaranteed to be sent at least once to the subscriber (by the broker)
    - This is done with a protocol requiring the receiver to acknowledge the reception of the message. The receiver can be either the broker or the subscribed client. Messages which are not acknowledged will be re-sent by the sender after a time-out time.

MQTT : QoS

- There are three different QoS levels in MQTT (0, 1, 2):
  - QoS = 2: the message is delivered exactly once to the client.
    - Here a double handshake is done: the receiver acknowledges the reception of a message with a “PUBREC” packet. The sender then discards the message and sends a PUBREL message indicating that the message has been release and from now on the message will not be re-sent anymore. The receiver again acknowledges the reception of the PUBREL message with PUBCOMP packet. At this point the sender can totally forget about the transfer. All messages are being re-sent if the respective acknowlege is not received withing a time-out period.



Connection to the Broker

MQTT-Packet:

CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

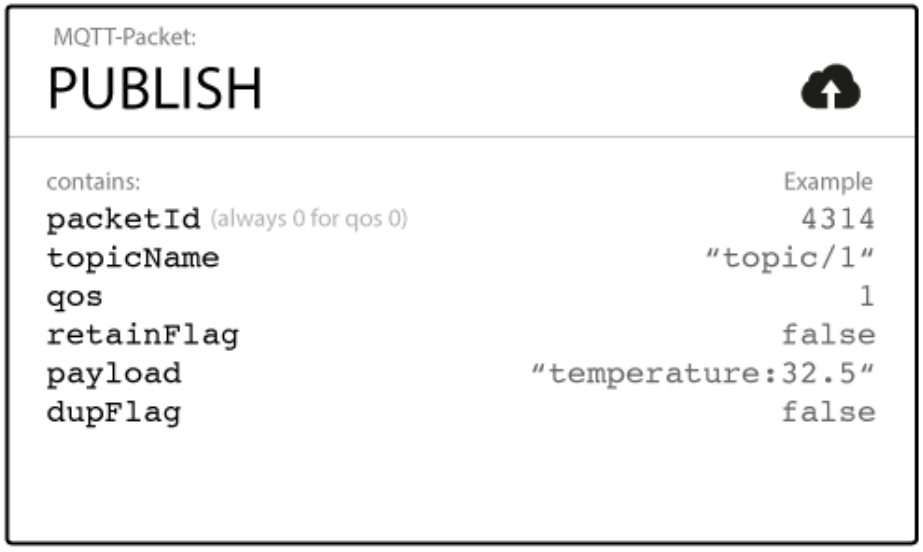
clientId: unique identifier

cleanSession: Information about session will be saved (e.g. subscribed topics). Will be used to restore session on dis-connect → re-connect

lastWill : If client disconnects for any reason the broker will send the lastWillMessage to clients which subscribed to lastWillTopic. (All this only if lastWill fields have been set)

keepAlive : a time-out period after which the connection is shut down if no message is sent during this time. To avoid this clients send "ping" messages in intervals smaller than this timeout (if no other messages are sent)

Publish message

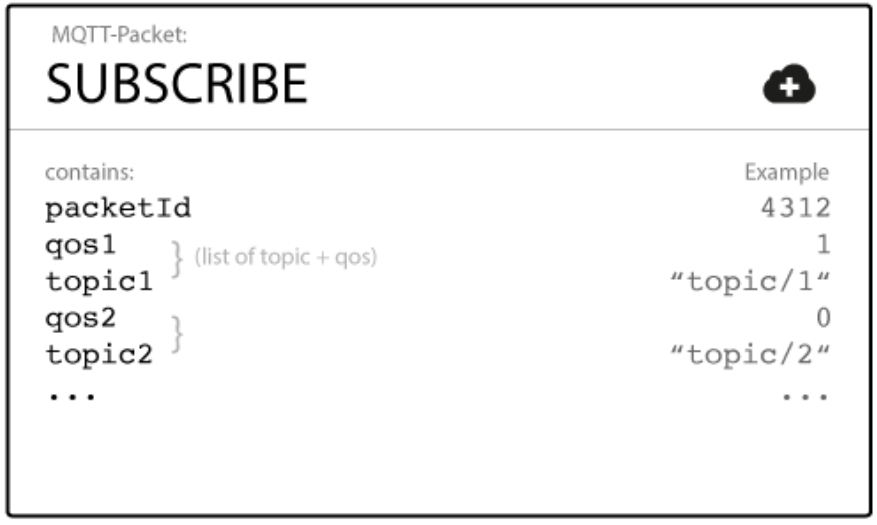


packetId: unique identifier (together with clientId) for the transaction. Generated by client. Used for qos>0 to associate acknowledge messages to specific transaction.

retainFlag: If true broker memorises the last published message for the topic and immediately sends it to clients on subscription to this topic. *Note:* a retained message can only be deleted from the broker by a client publishing a message with a 0-byte payload to that topic.

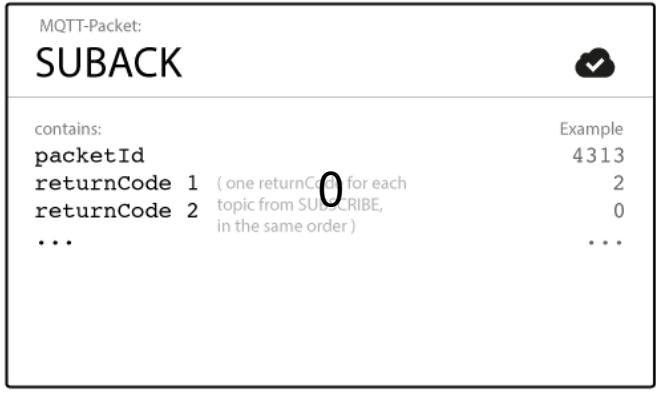
dupFlag : Set by client if a publish message is re-sent after the timeout (relevant for qos>0).

Subscribe message



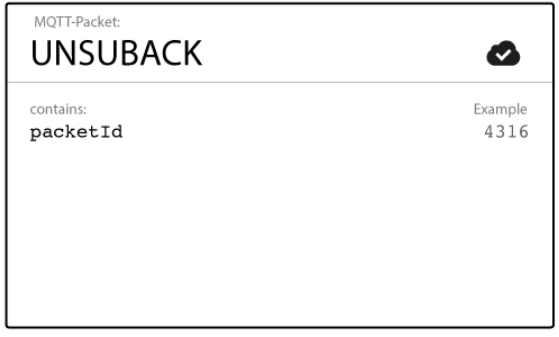
As can be seen subscribe messages can be chained in a single message:

This means with a single message a client can subscribe to multiple topics.



Subscriptions are acknowledged with packets contains a return code for every subscription: The return code is the qos level on success or 0x80 on failure

Un-subscribe message



As can be seen also the un-subscribe messages can be chained in a single message:

This means with a single message a client can un-subscribe from multiple topics.

Un-subscriptions are acknowledged with a very simple acknowlege packet

MQTT v5 enhancements

- Note:

  We gave an overview on MQTT v3 in this course. The latest protocol is MQTT v5 which has some additional features:

  - Custom key-value pairs in headers which opens the possibility to custom enhancements
  - Reason codes are available to allow diagnosing protocol errors if they occur
  - Clean Start flag upon connection allows to request to force a clean start of a session (also if the previous session had the cleanSession flag set to False)
  - Non trivial authentication is supported
  - The broker is allowed to gracefully disconnect from a client
  - Retransmission of packets on TCP networks only occurs if the network connection is compromised

Remarks to the exercise

- Use a json configuration file for setting parameters for

```
{
    "ssid"             : "student12",
    "password"         : "$unilab1",
    "mqtt_server"      : "10.42.0.181"
    "client_id"        : "[yourname]",
    "mqtt_topic"       : "sensors/[yourname]",
    "mqtt_publish_iv" : 10,
    "alt0"             : 12
}
```

  Be aware that “yourname” should be unique, so may be also add the first letter of your surname

- The MQTT server IP will have to be confirmed during the exercise (it is not guaranteed that my computer always gets the same IP, and we do not have a nameserver in our mini lab system, so we cannot use hostnames).
- The alt0 (altitude of Padova) is just given to convert the pressure to 0 sea level. (Well, here in Padova we are almost at sea level, however in Geneva I have to enter here 428m)

Remarks to the exercise

- Please add your coordinates to the json file you send to MQTT:
  - Add a field ‘row’ with the row number of your bench: we start counting from the blackboard to the projector screen. The row where I have my computer is row number 0
  - Add a field ‘column’ with the place number within the row (we start counting from the corridor to the window i.e. 0 to 3.
  - Finally add a field ‘name’ with your name (or “artist name”)

```
message = {
     "temperature" : temp,
     "pressure"    : p0,
     "humidity"    : hum,
     "row"         : {row},
     "col"         : {col},
     "name"        : “{your name}”
         }
```

The mqtt library for ESP32

- In the exercise we use a very simple mqtt client library called

  umqtt.simple

  - This library in part of the micropython installation
  - Documentation is not very detailed:
    https://mpython.readthedocs.io/en/v2.2.1/library/mPython/umqtt.simple.html
  - The best documentation is the code itself:
    https://github.com/micropython/micropython-lib

  - In our exercise we just need the connect() and the publish() calls. (We only use qos=0)

In case you want to play at home...

- If you want to play with MQTT at home you need a MQTT Broker
- A very widely used broker which works very well (we use it during the exercises) is the “mosquitto” broker.
  - Linux distributions usually contain this broker as an installable package
  - On Ubuntu you can do “apt-get install mosquitto”
  - The package also contains command-line tools for publishing and subscribing to topics. This is useful for testing and monitoring what you send to the broker from you microcontroller
  - Mosquitto also runs on other platforms (Windows, Mac)