IoT Engineering
7: Messaging Protocols and Data Formats

CC BY-SA, Thomas Amberg, FHNW
(unless noted otherwise)
Slides: tmb.gr/iot-7

Overview

These slides present the *MQTT messaging protocol.*

How to publish messages to a topic on a broker.

How to subscribe to messages about a topic.

How data formats encode the payload.

Prerequisites

Set up SSH access to the Raspberry Pi, install Node.js.

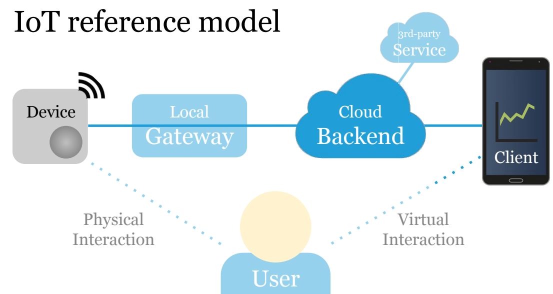
Check the Wiki entry on Raspberry Pi Zero W Setup.

And follow the steps to install the Node.js runtime.

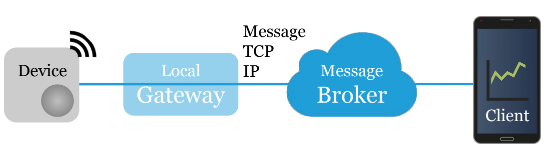
Set up the Feather Huzzah ESP8266 for Arduino.

Get access to a Wi-Fi network without a portal.

IoT reference model



Messaging protocols



Messaging protocols enable lightweight, bidirectional data exchange between devices and client apps.

We will look at the MQTT messaging protocol.

MQTT

MQTT is a standard protocol to transfer data packets.

In the OSI model, MQTT sits on the application layer.

It uses TCP/IP as a transport, on port 1883 and 8883.

The transferred data packets are called *messages*.

Current version is MQTT v5.0, replacing v3.1.1.

Publish/subscribe

MQTT is based on the *Publish*/*Subscribe* pattern.

This pattern decouples the sender and receiver.

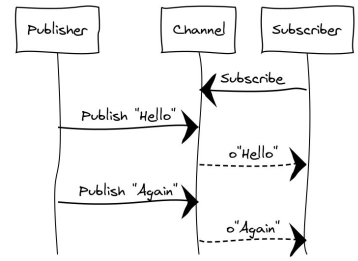
Publishers send messages to a specific channel.

Subscribers of a channel receive the messages.

Pub/Sub, 1:1

Publisher sends message to a channel.

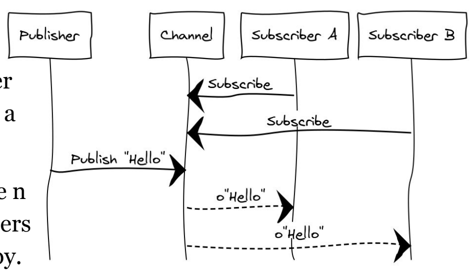
Subscriber gets the published message.



1:n

Publisher sends to a channel.

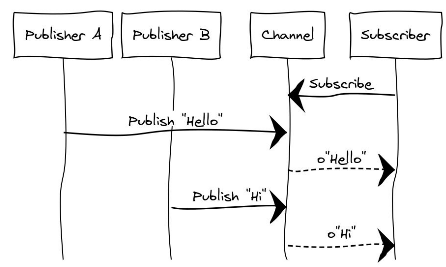
All of the n subscribers get a copy.



n:1

Publishers send to a channel.

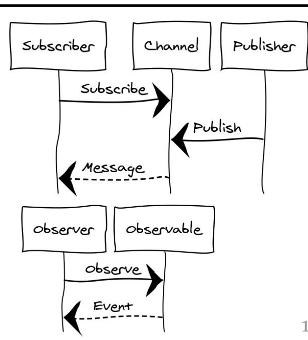
Subscriber gets each message.



Decoupling

With Pub/Sub the channel decouples the two parties.

Compare this to the *Observer* pattern, where the receiver knows the sender.



Clients and brokers

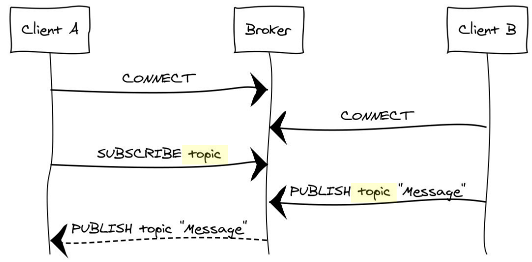
In MQTT, *clients* exchange messages via a *broker*.

Clients can be publishers, subscribers or both.

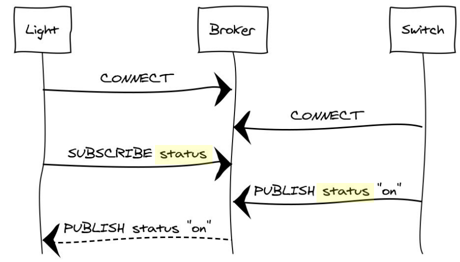
Brokers offer multiple channels, or *topics*.

Brokers can cache or store messages.

MQTT is session-based.



Connect to a broker, publish/subscribe to a topic.



Connected light with "broadcast" functionality.

Node.js MQTT with *mqtt*

Install the mqtt Node.js library & command line tool:

```
$ npm install mqtt # installs Node.js library
$ sudo npm install mqtt -g # adds tool to path
```

To publish/subscribe with the command line tool, try:

```
$ mqtt sub -t 'mytopic' -h 'test.mosquitto.org'
$ mqtt pub -t 'mytopic' \
-h 'test.mosquitto.org' \
-m 'Hello, world!'
```


Hands-on, 10': MQTT command line

Install the *mqtt* CLI tool on the Raspberry Pi.

Connect to the broker `test.mosquitto.org`

Subscribe to the topic `fhnw-iot/names`

Send* your name to the same topic.

*Open a second terminal.

Node.js MQTT subscriber client .js

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

const broker = "mqtt://test.mosquitto.org/";
const client = mqtt.connect(broker);
client.on("connect", () => {
  client.subscribe("hello"); // topic "hello"
});
client.on("message", (topic, message) => {
  console.log(message.toString());
});
```


Node.js MQTT publisher client .js

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

const broker = "mqtt://test.mosquitto.org/";
const client = mqtt.connect(broker);
client.on("connect", () => {
  const topic = "hello";
  const message = "Hello, World!";
  client.publish(topic, message);
});
```

Hands-on, 10': MQTT pub/sub clients

Install the mqtt Node.js library on the Raspberry Pi.

Run the previous MQTT pub/sub* client examples.

Use the *.js* link on each page or check the main repo.

To run a Node.js program *my.js*, type: `$ node my.js`

*Open a second terminal.

ESP8266 MQTT publisher client .ino

```
#include <ESP8266WiFi.h> // v2.4.2
#include <ESP8266MQTTClient.h> // v1.0.4

MQTTClient client;

void handleConnected() {
  client.publish("hello", "Hello, World!");
}

client.onConnect(handleConnected);
client.begin("mqtt://test.mosquitto.org/");
```


ESP8266 MQTT subscriber client .ino

```
#include <ESP8266WiFi.h> // v2.4.2
#include <ESP8266MQTTClient.h> // v1.0.4

MQTTClient client;

void handleC…() { client.subscribe("hello"); }
void handleD…(String topic, String data,…) {…}

client.onConnect(handleConnected);
client.onData(handleDataReceived);
client.begin("mqtt://test.mosquitto.org/");
```


Topics

The broker organises messages into multiple topics.

Clients send each message to a specific topic.

Clients subscribe to one or more topics.

Topics are hierarchical, like paths.

Wildcards replace topic levels.

Home

```
home
  /room
    /light
      /status             "on"
      /color              "255,0,64"
    /sensor
      /temperature        "23.0"
      /humidity           "42"

home/room/light/status    "off"
```


mqtt+json

```
home
  /room
    /light
      {
        "status": "on",
        "color": "255,0,64"
      }
    /sensor …

home/room/light~~/status~~ {"status":"off"}
```

Broker

```
$SYS
  /broker
    /load
      /bytes
        /received/+    "1024", "3280", "31415"
        /sent/1min     "2048"  (5min)  (15min)
    /clients
      /connected       "3"
      /total           "99"
```


Hands-on: 15' local MQTT broker

Install and run the *mosquitto* broker on Raspberry Pi:

```
$ sudo apt-get update
$ sudo apt-get install mosquitto # port 1883
```

Test with the ESP8266 publisher/subscriber clients.

Use the *.ino* link on the page or check the main repo.

Check `$SYS/broker/clients/connected` on the Pi.

Quality of Service

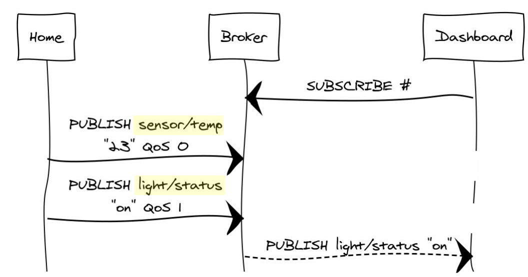
Clients indicate desired *QoS* when publishing.

QoS 0 — At most once delivery

QoS 1 — At least once delivery

QoS 2 — Exactly once delivery*

*QoS 2 is hard to implement reliably, in practice.

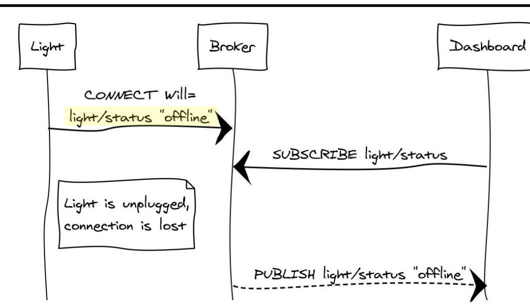


Will message

MQTT allows to set a "last will" when connecting.

The client specifies a will topic and a will message.

The will is published as soon as the client is offline.



Client libraries and tools

Paho is an open source library in Java, Python, ...

MQTT.js is Node.js library and command line tool.

Node-RED ist a dataflow-based, rule-based client.

HiveMQ is a MQTT client with Websocket support.

There are many other clients/libraries at mqtt.org.

Broker software

Shiftr.io visualises topics and messages in real-time.

Mosquitto is small and runs on the Raspberry Pi.

VerneMQ supports clustering and it's open source.

AWS and Azure IoT are scalable and highly reliable.

Additional broker software is listed on mqtt.org.

MQTT security

MQTT over TCP/IP can rely on (point-to-point) TLS.

For testing with TLS, see http://test.mosquitto.org/

End-to-end encryption is offered, e.g. by Teserakt*.

*See also Is MQTT Secure?

Reasons to use MQTT

Clients don't have to know each other, just the broker.

Messages can be cached, while a client stays offline.

Subscribing to hierarchies of topics with wildcards.

Last-will message, as soon as a client goes offline.

New features in MQTT v5.0

Reason code in the case of errors (on CONNACK).

Payload format and content type (MIME type).

Session expiry interval (from disconnect).

Optional broker feature availability.

There is a detailed summary in the v5.0 spec.

Data formats

Two parties need to agree on what is valid content.

Parsing means reading individual "content tokens".

Record-based formats, e.g. CSV, are good for tables.

Text-based formats, e.g. JSON are easily readable.

Binary formats, e.g. Protobuf, take less space.

Data formats are often specified in EBNF.

CSV

Comma Separated Values (CSV), defined in RFC4180.

```
file = record *(CRLF record) [CRLF];
record = field *(COMMA field);
field = *TEXTDATA;
CRLF = CR LF;
COMMA = %x2C; CR = %x0D; LF = %x0A;
TEXTDATA = %x20-21 / %x23-2B / %x2D-7E;
```

Header and escaped fields omitted for shortness.

JSON

JSON is a simple data format based on Unicode text:

```
{"temp": 23} // try ddg.co/?q=json+validator
```

On the Raspberry Pi, Node.js offers the JSON object:

```
const obj = JSON.parse("{\"temp\": 23}");
const data = JSON.stringify(obj);
```

On Arduino, use e.g. the Arduino_JSON library:

```
JSONVar obj = JSON.parse("{\"temp\": 23}");
String data = JSON.stringify(obj);
```


Protobuf

Protocol Buffers (Protobuf) is a binary data format:

```
message Measurement {
  required int32 temp = 1;
  optional int32 humi = 2;
}
```

Message schemas are compiled to a target language, i.e. a parser is generated, re-generated upon changes.

Hands-on, 15': Data formats

Choose one of the Grove sensors listed in the Wiki.

Define a suitable JSON format to transmit its data.

Translate the format into a Protobuf .proto file.

Done? Build the parser for Node.js or Arduino.

Summary

MQTT is a messaging protocol based on pub/sub.

Clients exchange messages by topic, via a broker.

Advantages are decoupled clients, will message.

Data formats allow to write and read payloads.

Next: Long Range Connectivity with LoRaWAN.

Feedback or questions?

Write me on https://fhnw-iot.slack.com/

Or email thomas.amberg@fhnw.ch

Thanks for your time.