MQTT
Message Queue Telemetry Transport

Master 's Degree in Informatics Engineering

Contents

1. What is MQTT?
2. Why MQTT?
3. MQTT Components
4. Mosquitto
5. ESP8266 - MQTT

Contents

What is MQTT?

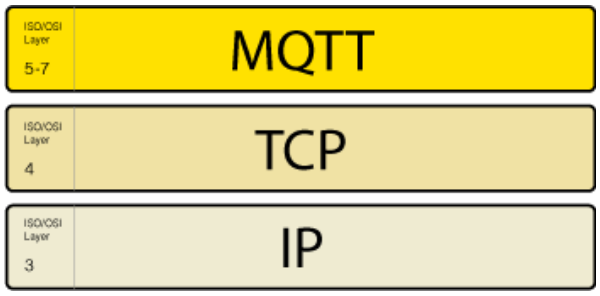
MQTT (Message Queuing Telemetry Transport) is a machine-to-machine connectivity protocol that runs over TCP/IP.

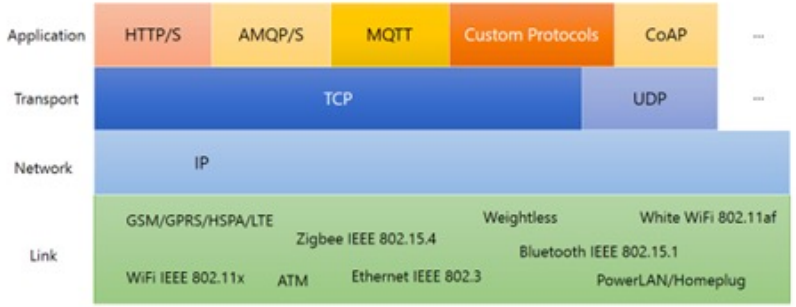
Lightweight, simple, MQTT is based on a publish- subscribe structure:

- A **Publisher** sends messages according to Topics, to specified Brokers.
- A **Broker** acts as a switchboard, accepting messages from publishers on specified topics, and sending them to subscribers to those Topics.
- A **Subscriber** receives messages from connected Brokers and specified Topics.

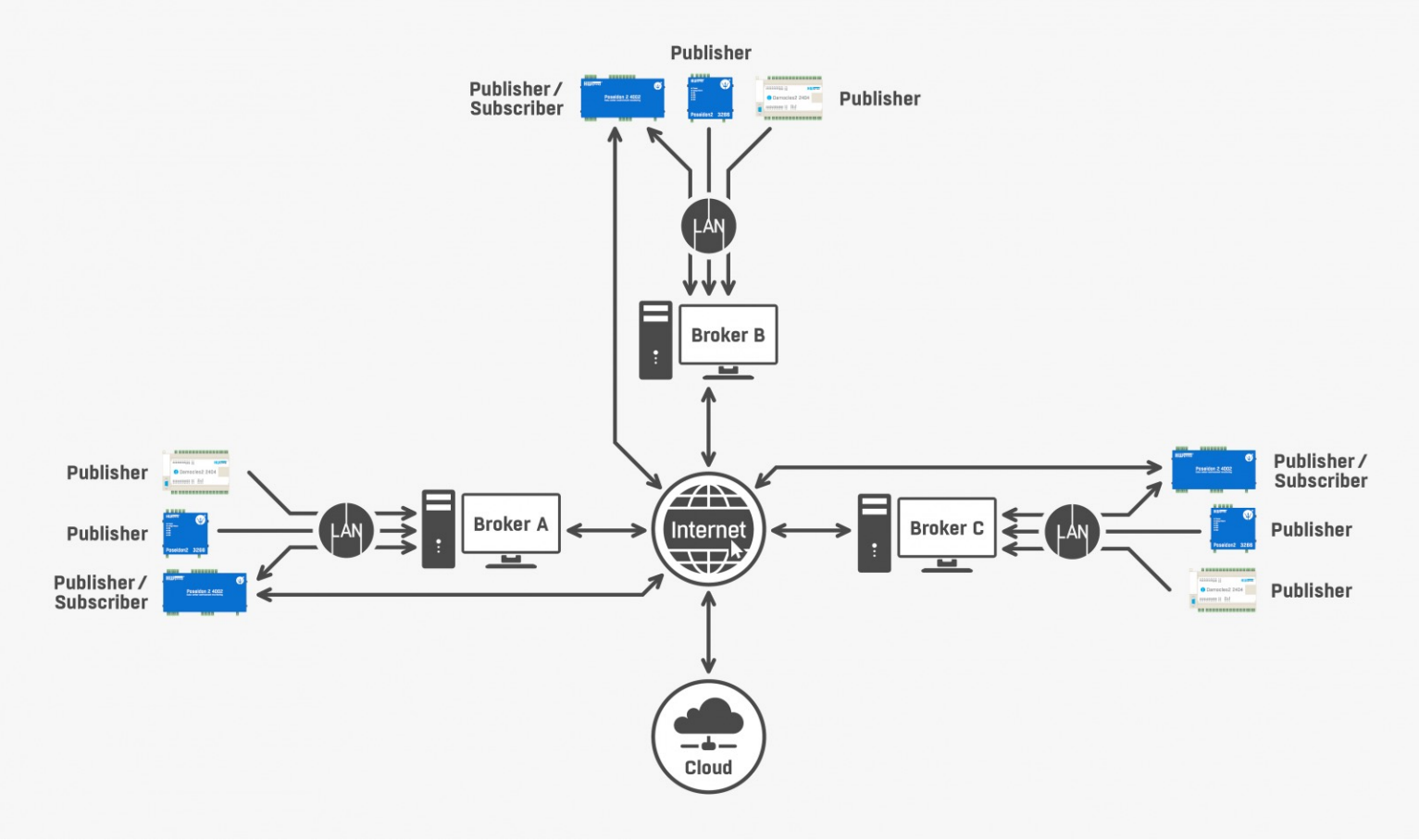
What is MQTT?

- Introduced by IBM 1999
- Main goal: Optimize battery and minimal bandwith
- Standardized as ISO/IEC 20922:2016
- Uses Publish/Subscribe mechanism controlled by Broker
- Provides a Quality of Service Data Delivery
- Broker
  - Software component
  - Responsible for distributing messages from Publishers to interested Subscribers





What is MQTT?


Publisher
Publisher / Subscriber
Publisher
LAN
Broker B
Publisher
Publisher
Publisher / Subscriber
LAN
Broker A
Internet
Broker C
LAN
Publisher / Subscriber
Publisher
Publisher
Cloud

Contents

Why MQTT?

- Most popular application layer protocols used nowadays:
  - **CoAP:** Constrained Application Protocol
  - **MQTT:** Message Queuing Telemetry Transport
  - **XMPP:** Extensible Messaging and Presence Protocol
  - **AMQP:** Advanced Message Queuing Protocol
  - **WebSocket:** Computer Communications Protocol
  - **Alljoyn:** Full stack of protocols intended for IoT. Not separable application layer protocol

Protocol	QoS	Communication Pattern	Target Devices
CoAP	YES	Req/Resp	Very constrained
MQTT	YES	Pub/Sub	Generic, small header
XMPP	NO	Req/Resp Pub/Sub	High memory consumption
HTTP	NO	Req/Resp	High performance
AMQP	YES	Pub/Sub	Ser-2-Ser communication
Web Socket	NO	Client/Server Pub/Sub	needs less power than HTTP still needs high power
AllJoyn	NO	Client/Server Pub/Sub	High computational power

Why MQTT?

- Used on constrained devices and server applications.
- It keeps bandwidth requirements to an absolute minimum.
- It handles unreliable networks.
- It requires little implementation effort for developers.
- It was designed for machine-to-machine (M2M) communication.

Contents

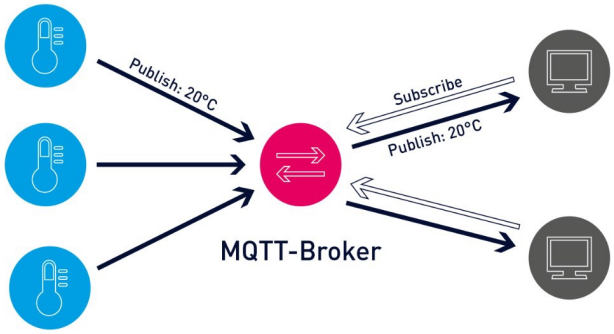
MQTT Components

- Many-to-many Sub to Pub relationship
- One Broker for every system
- Subs authenticated to Broker
- Subs/Pubs can be very constrained
- Pub can be even only a sensor
- Broker has to provide more computational power

Client
(STM32, Arduino, Rapberry Pi)

Broker
(en ligne ou Raspberry Pi)

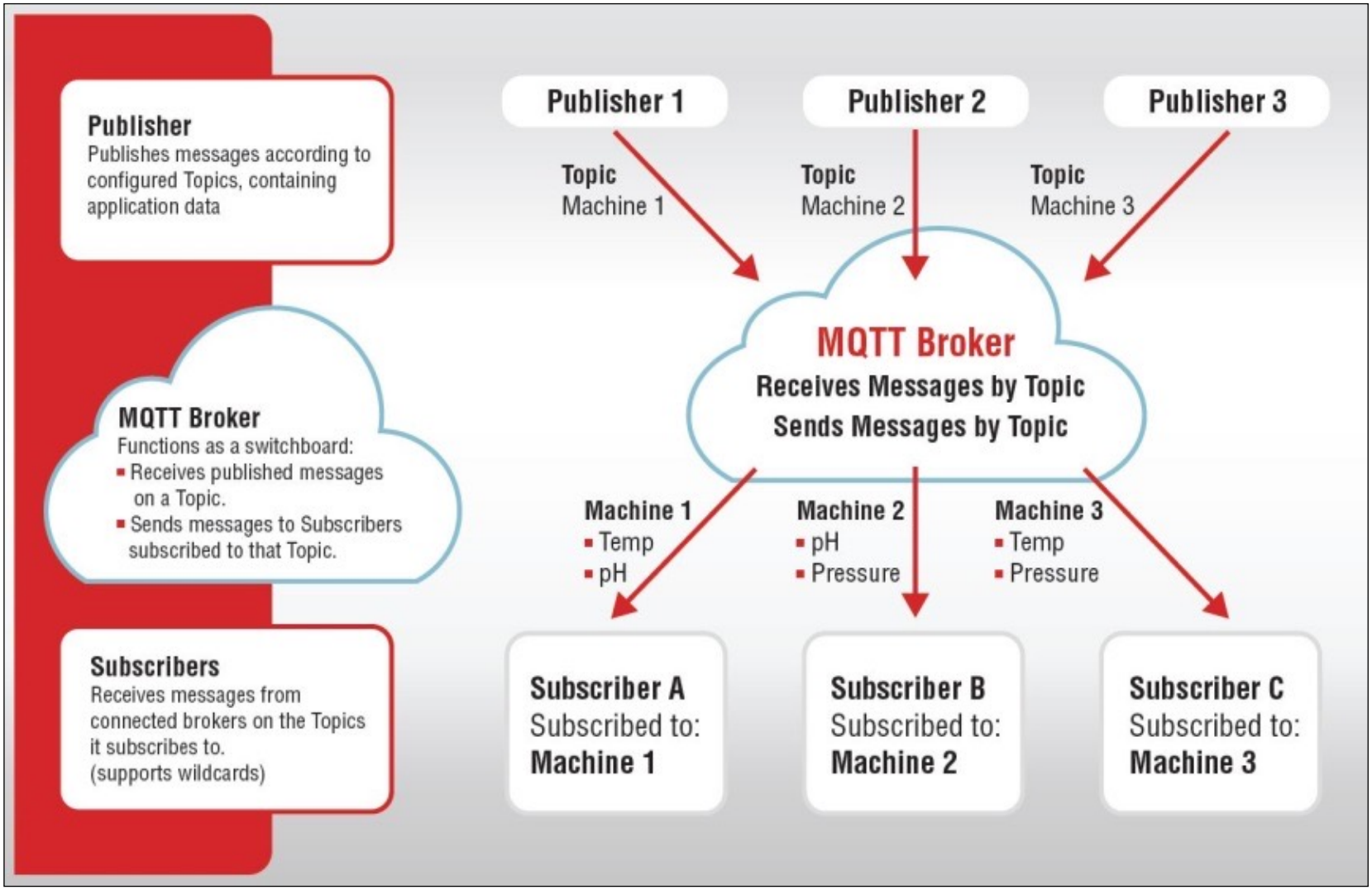
THE THINGS NETWORK



Application
(PC, Smartphone)

https://www.paessler.com/

MQTT Components

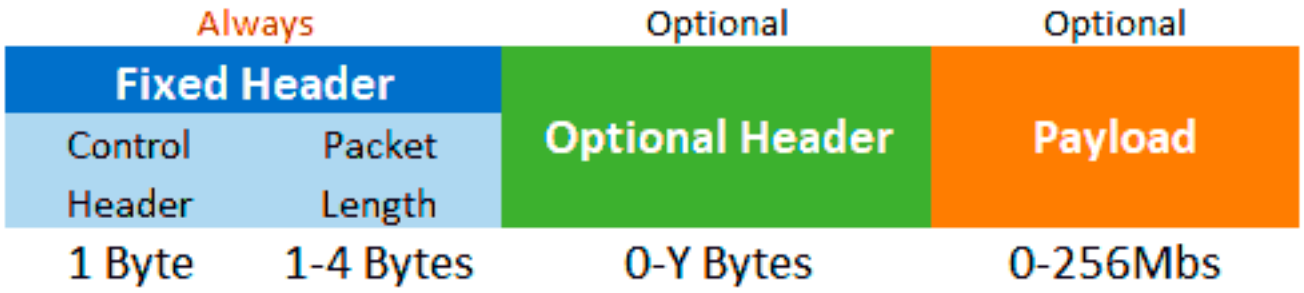


MQTT – Message Structure

Fixed header — This is a mandatory part of the message comprising a control header and a packet size. The minimum size is 2 bytes and the maximum size is 5 bytes.

Variable header — This is an optional part of the message that provides additional information. Its size may vary, depending on the message type.

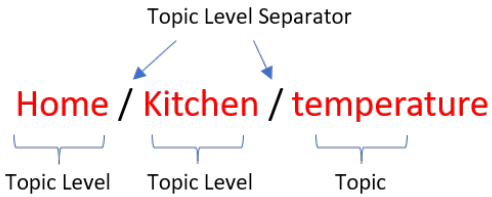
Payload — This is an optional part of the message with a maximum size of 256 MB. It may include different commands, like switching on/off, data exchange, and reading sensor data.



MQTT - Topics

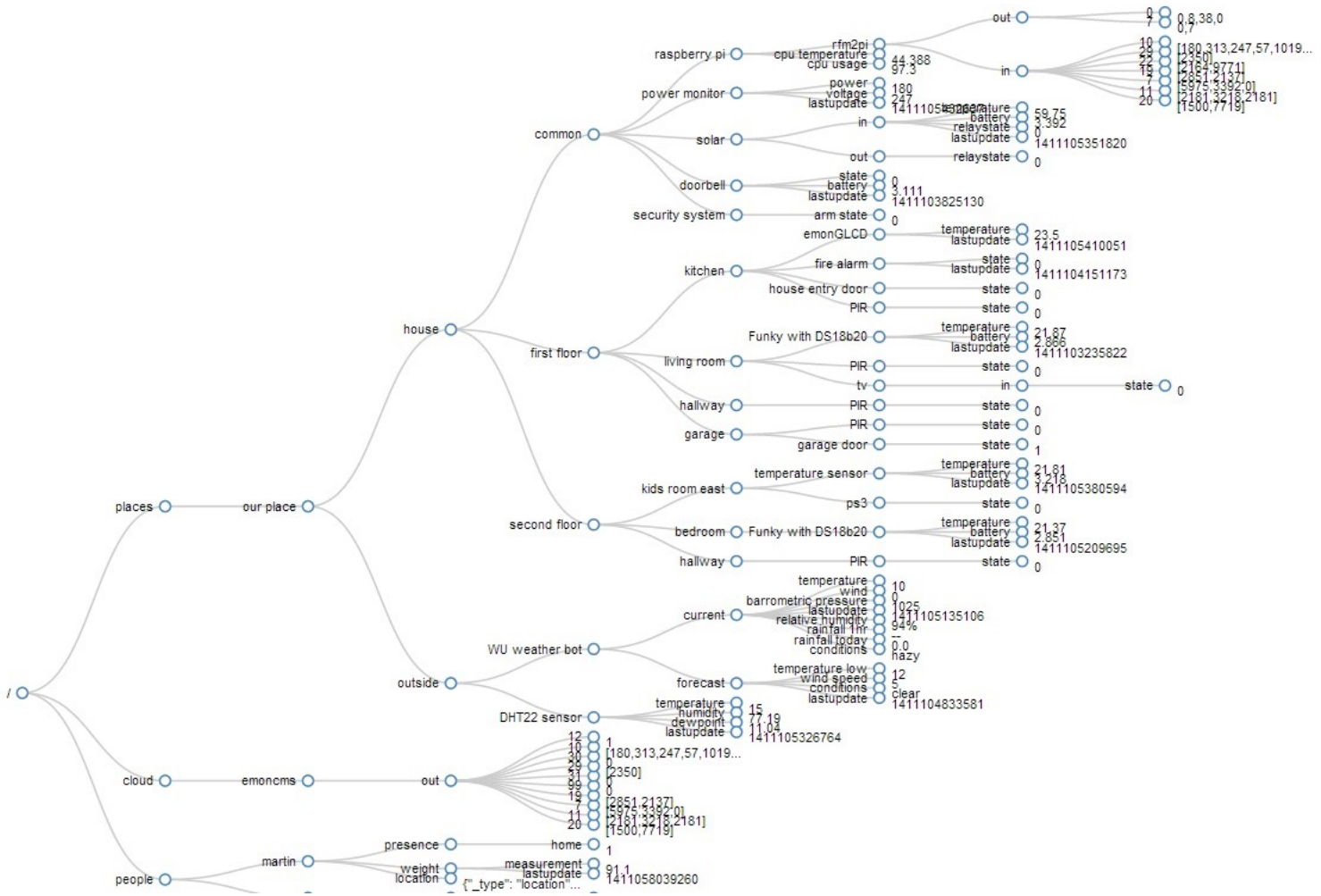
Topics allow MQTT clients to **share information**.

- MQTT Topics are structured in a hierarchy, similar to folders and files in a file system, using the forward slash (/) as a delimiter.



- It creates a self descriptive naming structures
- All topics are created by a subscribing or publishing client, and they are not permanent.
- A topic only exists if a client has subscribed to it, or a broker has a retained messages for that topic.
- Broker do not create topics, except the reserved $SYS topic that it is used to publish information about the broker.
  - Last Will – Each client can specify its last will message when it connects to a broker. This message will be send to other clients about an ungracefully disconnected client (it runs out of power, crashes,…)

MQTT - Topics


/
places
our place
house
common
raspberry pi
rfm2pi
cpu temperature
44.388
cpu usage
97.3
out
0
0,8,38,0
7
0,7
in
10
[180,313,247,57,1019...
power monitor
power
180
voltage
247
lastupdate
1411105326764
solar
in
temperature
59.75
battery
3.392
relaystate
0
lastupdate
1411105351820
out
relaystate
0
doorbell
state
0
battery
3.111
lastupdate
1411103825130
security system
arm state
0
first floor
kitchen
emonGLCD
temperature
23.5
lastupdate
1411105410051
fire alarm
state
0
lastupdate
1411104151173
house entry door
state
0
PIR
state
0
living room
Funky with DS18b20
temperature
21.87
battery
2.866
lastupdate
1411103235822
PIR
state
0
tv
in
state
0
hallway
PIR
state
0
garage
PIR
state
0
garage door
state
1
second floor
kids room east
temperature sensor
temperature
21.81
battery
3.218
lastupdate
1411105380594
ps3
state
0
bedroom
Funky with DS18b20
temperature
21.37
battery
2.851
lastupdate
1411105209695
hallway
PIR
state
0
outside
WU weather bot
current
temperature
10
wind
0
barrometric pressure
1025
lastupdate
1411105135106
relative humidity
94%
rainfall 1hr
--
rainfall today
0.0
conditions
hazy
forecast
temperature low
12
wind speed
5
conditions
clear
lastupdate
1411104833581
DHT22 sensor
temperature
15
humidity
77.19
dewpoint
11.03
lastupdate
1411105326764
cloud
emoncms
out
12
1
10
[180,313,247,57,1019...
30
0
25
0
31
[2350]
99
0
19
0
5
[2851,2137]
7
[5975,3392,0]
11
[2181,3218,2181]
20
[1500,7719]
people
martin
presence
home
1
weight
measurement
91.1
location
lastupdate
1411058039260
{"_type": "location"...

MQTT - Topics

A client can subscribe to individual or multiple topics.

When subscribing to multiple topics two **wildcard** characters can be used. They are:

(hash character) – Multi level wildcard
+ (plus character) – Single level wildcard

* Wildcards can only be used to denote a level or multi-levels i.e /house/# and not as part of the name to denote multiple characters e.g. hou# is not valid

Sample Topics:

Home/LivingRoom/DHT22/Humidity
Home/BedRoom/DHT22/Temperature
Home/BedRoom/DHT22/Humidity
Home/Balcony/LDR/DayLight
Home/Kitchen/SmokeSensor/Smoke

Single Level Wildcard (+) :

Topic with Wild Card:
Home/+/DHT22/Humidity

Matches from Sample Topics:
Home/LivingRoom/DHT22/Humidity
Home/BedRoom/DHT22/Humidity

Sample Topics:

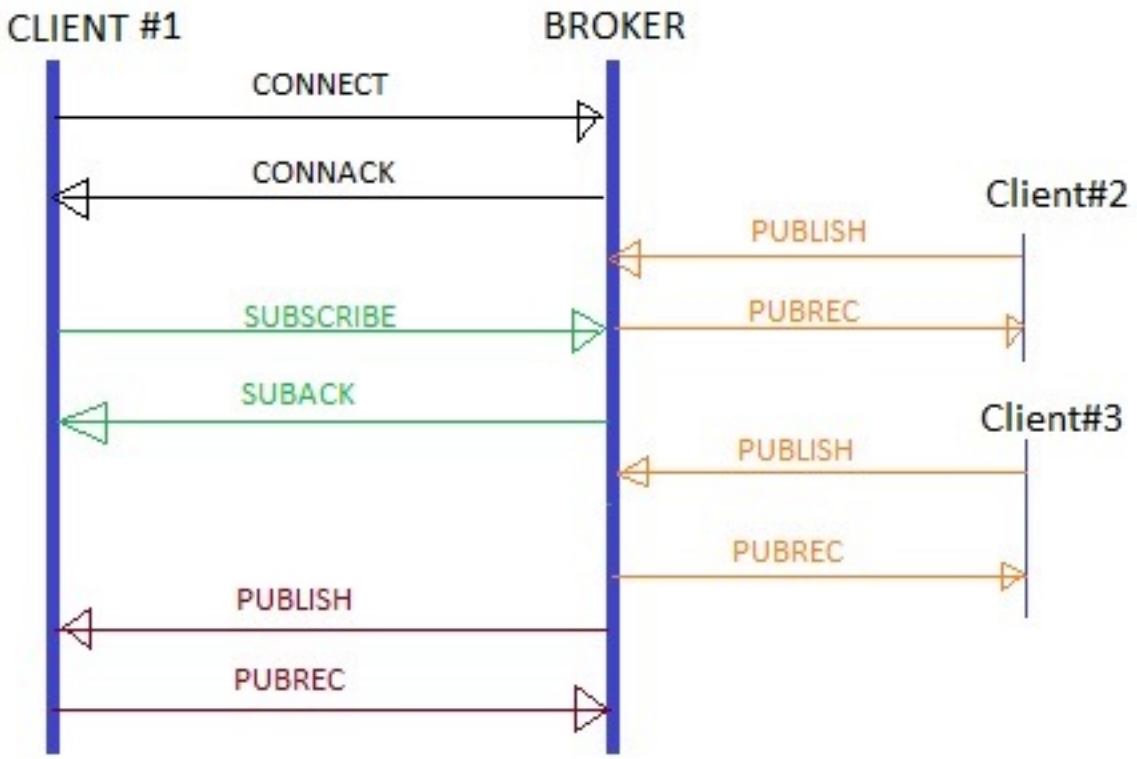
Home/LivingRoom/DHT22/Humidity
Home/BedRoom/DHT22/Temperature
Home/BedRoom/DHT22/Humidity
Home/Balcony/LDR/DayLight
Home/Kitchen/SmokeSensor/Smoke

Multi Level Wildcard (#) :

Topic with Wild Card:
Home/BedRoom/#

Matches from Sample Topics:
Home/BedRoom/DHT22/Temperature
Home/BedRoom/DHT22/Humidity

MQTT – Message Flow



MQTT Case Studies

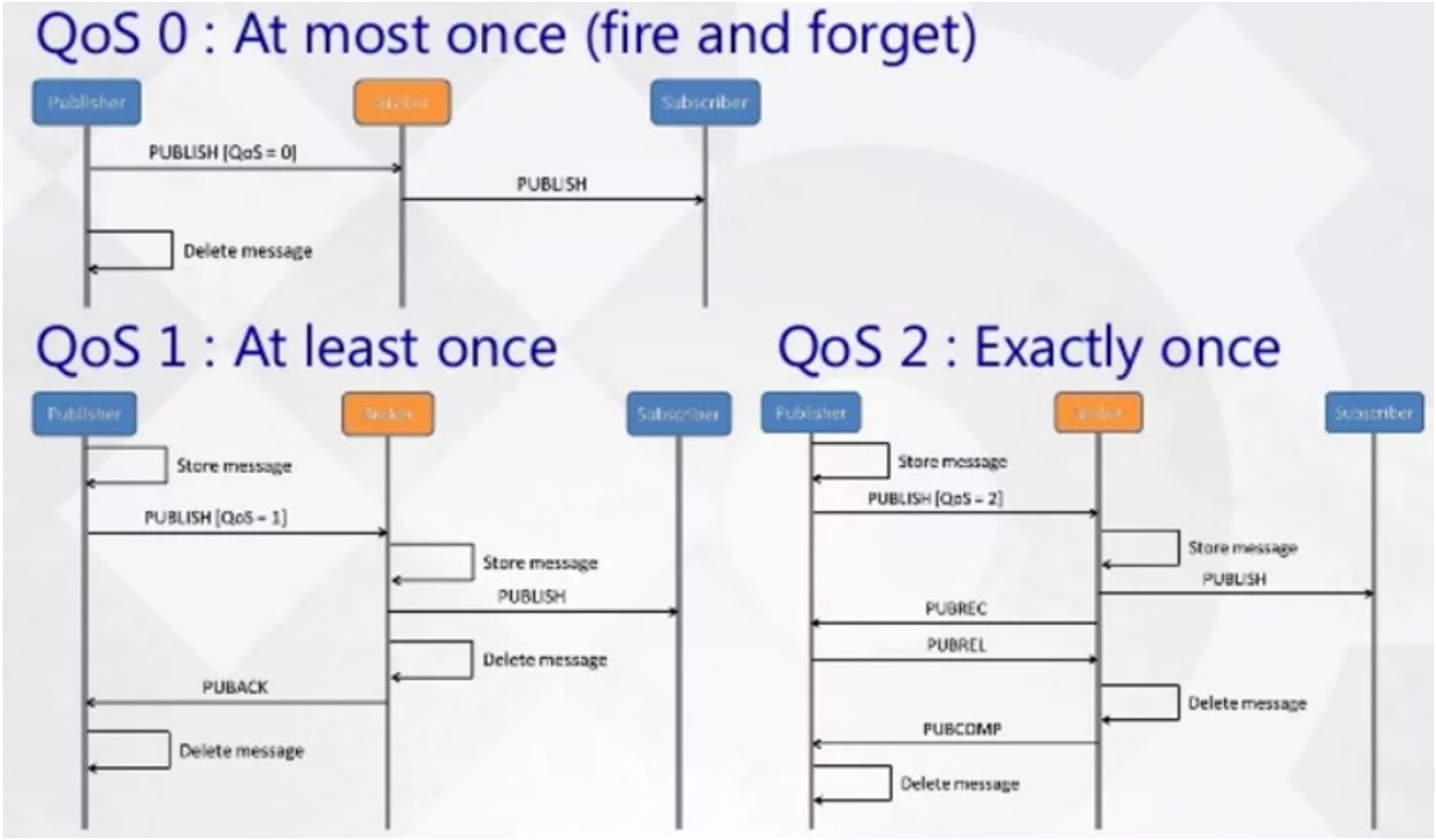
MQTT - QoS

- **QoS 0 (set as default)**: A publisher sends a message without requesting guaranteed delivery. You can use it when the information transmitted is not critical, and the connection is stable.

- **QoS 1**: A publisher sends a message until it gets a delivery confirmation. You can use it when the information transmitted is critical, and the connection is not stable. QoS 1 makes sure the subscriber receives the message.

- **QoS 2**: A publisher sends a message only once with guaranteed delivery. You can use it when the information transmitted is critical, and the connection is not stable. QoS 2 makes sure the subscriber receives the message only once without its duplicates and overhead.

MQTT - QoS


QoS 0 : At most once (fire and forget)
Publisher
Broker
Subscriber
PUBLISH [QoS = 0]
PUBLISH
Delete message
QoS 1 : At least once
Publisher
Broker
Subscriber
Store message
PUBLISH [QoS = 1]
Store message
PUBLISH
Delete message
PUBACK
Delete message
QoS 2 : Exactly once
Publisher
Broker
Subscriber
Store message
PUBLISH [QoS = 2]
Store message
PUBLISH
PUBREC
PUBREL
Delete message
PUBCOMP
Delete message

Contents

Mosquitto - is an open source message broker

mosquitto_pub > is a simple MQTT client that will publish a single message on a topic and exit.

Synopsis

```
mosquitto_pub { [ -h hostname ] [ --unix socket path ] [ -p port-number ] [ -u username ] [ -P password ] -t message-topic ... | -L URL } [ -A
    bind-address ] [ -c ] [ -d ] [ -D command identifier value ] [ -i client-id ] [ -I client-id-prefix ] [ -k keepalive-time ] [ --nodelay ] [ -q
    message-QoS ] [ --quiet ] [ -r ] [ --repeat count ] [ --repeat-delay seconds ] [ -S ] [ -V protocol-version ] [ -x session-expiry-interval ] { -f
    file | -l | -m message | -n | -s } [ --will-topic topic [ --will-payload payload ] [ --will-qos qos ] [ --will-retain ] ] [[ { --cafile
    file | --capath dir } [ --cert file ] [ --key file ] [ --ciphers ciphers ] [ --tls-version version ] [ --tls-alpn protocol ] [ --tls-engine
    engine ] [ --keyform { pem | engine }] [ --tls-engine-kpass-sha1 kpass-sha1 ] [ --tls-use-os-certs ] [ --insecure ] ] | [ --psk hex-key --psk-
   identity identity [ --ciphers ciphers ] [ --tls-version version ] ]] [ --proxy socks-url ]
mosquitto_pub [ --help ]
```

Examples:

> Publish temperature information to localhost with QoS 1:

```
mosquitto_pub -t sensors/temperature -m 32 -q 1
```

> Publish timestamp and temperature information to a remote host on a non-standard port and QoS 0:

```
mosquitto_pub -h 192.168.1.1 -p 1885 -t sensors/temperature -m "1266193804 32"
```

> Publish light switch status. Message is set to retained because there may be a long period of time between light switch events:

```
mosquitto_pub -r -t switches/kitchen_lights/status -m "on"
```

https://mosquitto.org/man/mosquitto_pub-1.html

Mosquitto - is an open source message broker

mosquitto_sub > is a simple MQTT client that that will subscribe to topics and print the messages that it receives.

Synopsis

```
mosquitto_sub { [ -h hostname ] [ --unix socket path ] [ -p port-number ] [ -u username ] [ -P password ] -t message-topic ... | -L URL [ -t
    message-topic ...] } [ -A bind-address ] [ -c ] [ -C msg-count ] [ -d ] [ -D command identifier value ] [ -E ] [ -i client-id ] [ -I client-id-
   prefix ] [ -k keepalive-time ] [ -N ] [ --nodelay ] [ --pretty ] [ -q message-QoS ] [ --random-filter chance ] [ --remove-retained ] [ -R | --retained-
   only ] [ --retain-as-published ] [ -S ] [ -T filter-out ...] [ -U unsub-topic ...] [ -v ] [ -V protocol-version ] [ -W message-processing-timeout ] [ -
   x session-expiry-interval ] [ --proxy socks-url ] [ --quiet ] [ --will-topic topic [ --will-payload payload ] [ --will-qos qos ] [ --will-retain ]
   ] [[ { --cafile file | --capath dir } [ --cert file ] [ --key file ] [ --tls-version version ] [ --tls-alpn protocol ] [ --tls-engine engine ]
   [ --keyform { pem | engine }] [ --tls-engine-kpass-sha1 kpass-sha1 ] [ --tls-use-os-certs ] [ --insecure ] ] | [ --psk hex-key --psk-identity
    identity [ --tls-version version ] ]]
mosquitto_sub [ --help ]
```

Examples:

> Subscribe to temperature information on localhost with QoS 1:

```
mosquitto_sub -t sensors/temperature -q 1
```

> Subscribe to temperatures updates on multiple machines/hard drives to sensors/machines/HOSTNAME/temperature/HD_NAME:

```
mosquitto_sub –t sensors/machines/+/temperature/+
```

> Subscribe to all broker status messages:

```
mosquitto_sub -v -t \$SYS/#
```

https://mosquitto.org/man/mosquitto_sub-1.html

Contents

ESP8266 – MQTT Publisher

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "ssid";
const char* password = "password";
const char* mqtt_server = "192.168.4.1";

WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

```
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      client.publish("outTopic", "hello world");
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {

  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  unsigned long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;
    snprintf (msg, MSG_BUFFER_SIZE, "hello world #%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("broker/counter", msg);
  }
}
```

Required → ESP8266WiFi and PubSubClient libraries

ESP8266 – MQTT Subscriber

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "ssid";
const char* password = "password";
const char* mqtt_server = "192.168.4.1";

const char* clientID = "ESP-01";
const char* clientUserName = "ESP-01";
const char* clientPassword = "ESP-01";


WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
int value = 0;

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

}

void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  client.connect(clientID,clientUserName,clientPassword);
  client.subscribe("broker/counter");
}

void loop() {

  client.loop();
  delay(10);
}
```

Required → ESP8266WiFi and PubSubClient libraries

ESP8266 – MQTT Broker (uMQTTBroker Library)

```
/*
 * uMQTTBroker demo for Arduino
 *
 * Minimal Demo: the program simply starts a broker and waits for any client to connect.
 */

#include <ESP8266WiFi.h>
#include "uMQTTBroker.h"

uMQTTBroker myBroker;

/*
 * Your WiFi config here
 */
char ssid[] = "ssid";      // your network SSID (name)
char pass[] = "password"; // your network password

int counter = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println();
  Serial.println();


  WiFi.softAP(ssid, pass);
  Serial.println("AP started");
  Serial.println("IP address: " + WiFi.softAPIP().toString());

  // Start the broker
  Serial.println("Starting MQTT broker");
  myBroker.init();

  myBroker.subscribe("#");
}

void loop()
{
  myBroker.publish("broker/counter", (String)counter++);

 Serial.print("Clients:");
 Serial.println(myBroker.getClientCount());
 // wait a second
 delay(1000);
}
```

Required → ESP8266WiFi and uMQTTBroker libraries

https://github.com/martin-ger/uMQTTBroker