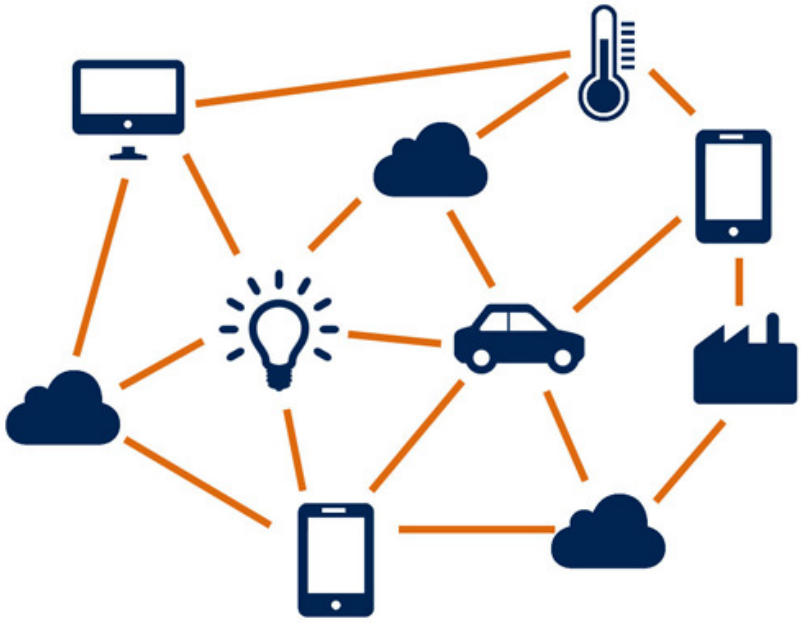
Internet of Things Protocol



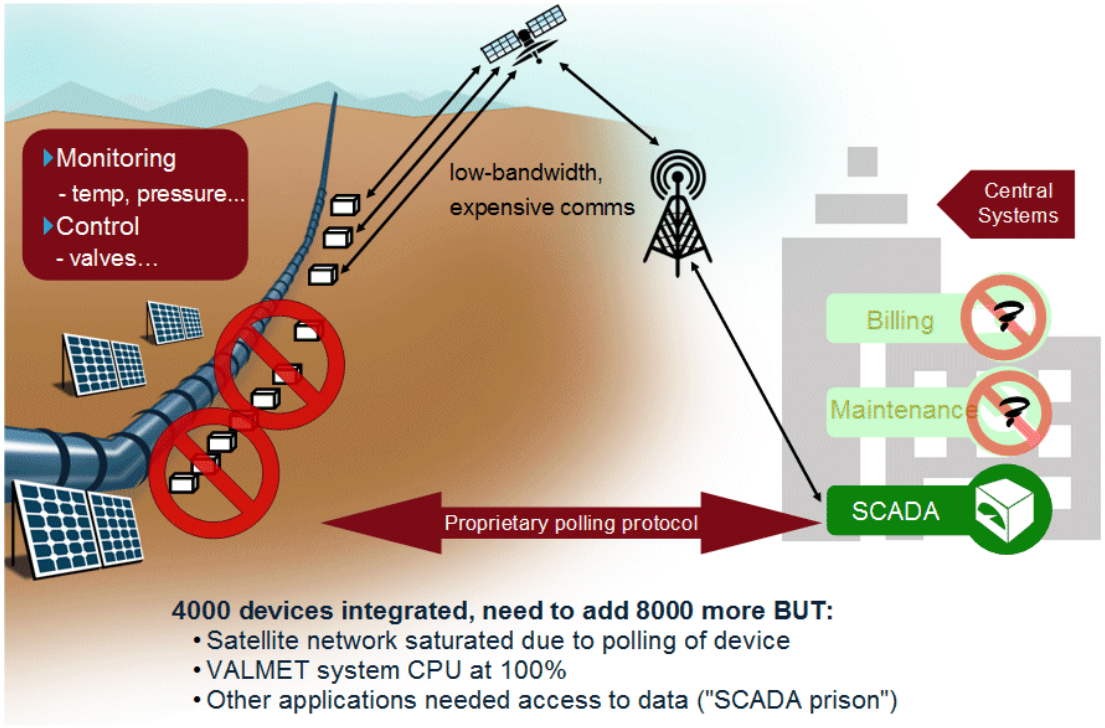
MQTT Protocol

- Invented by Andy Stanford Clark (IBM) and Arlen Nipper (Eurotech) in 1999
- Originally envisioned for use over Satellite link from an oil pipe line

Use case:

- 30.000 devices
- 17.000 km pipeline
- Remote monitoring
- Remote control
- Uses satellite links
- Bandwidth is **very** expensive

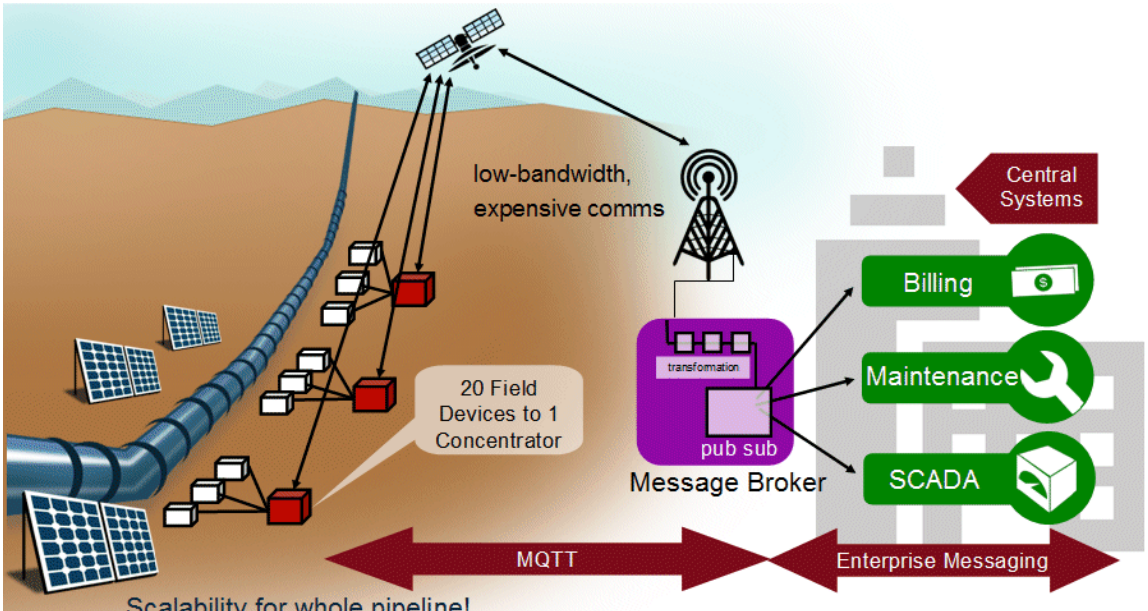
MQTT Protocol



4000 devices integrated, need to add 8000 more BUT:

- Satellite network saturated due to polling of device
- VALMET system CPU at 100%
- Other applications needed access to data ("SCADA prison")

MQTT Protocol



Scalability for whole pipeline!
Network traffic much lower - events pushed to/from devices and report by exception
Network cost reduced
Lower CPU utilization
Broken out of the SCADA prison – data accessible to other applications

MQTT Protocol

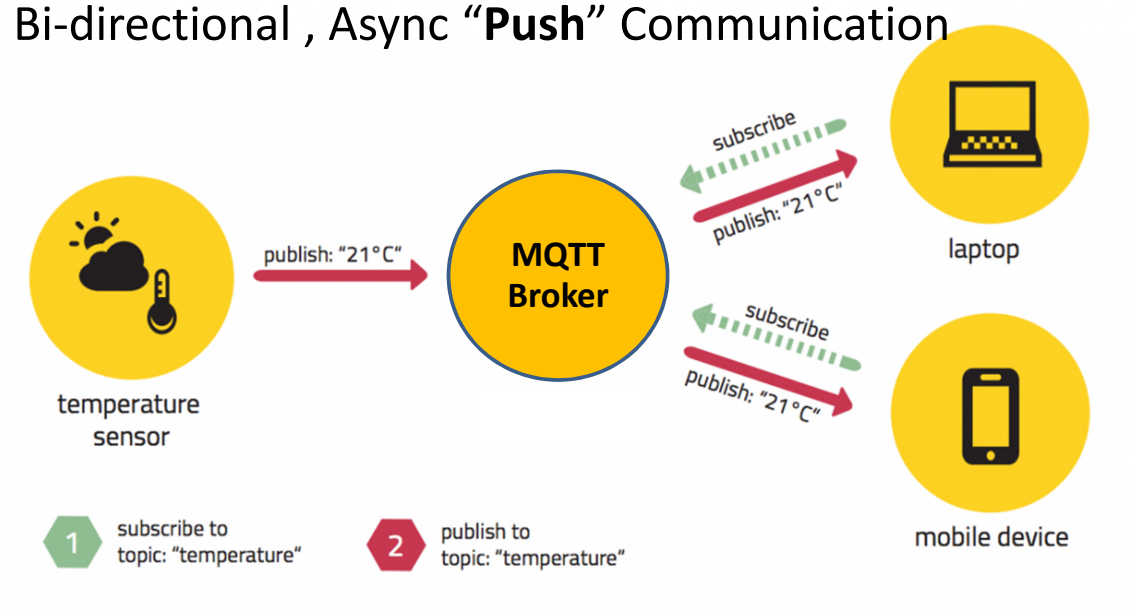
MQTT a lightweight protocol for IoT messaging

- **open** — open spec, standard — 40+ client implementations
- **lightweight** — minimal overhead — efficient format — tiny clients (kb)
- **reliable** — QoS for reliability on unreliable networks
- **simple** — 43-page spec — connect + publish + subscribe

Invented	Published	Eclipse M2M	Standard
EUROTECH IBM	MQTT	paho	OASIS
Late 1990s	Aug 2010	Nov 2011	Sep 2014

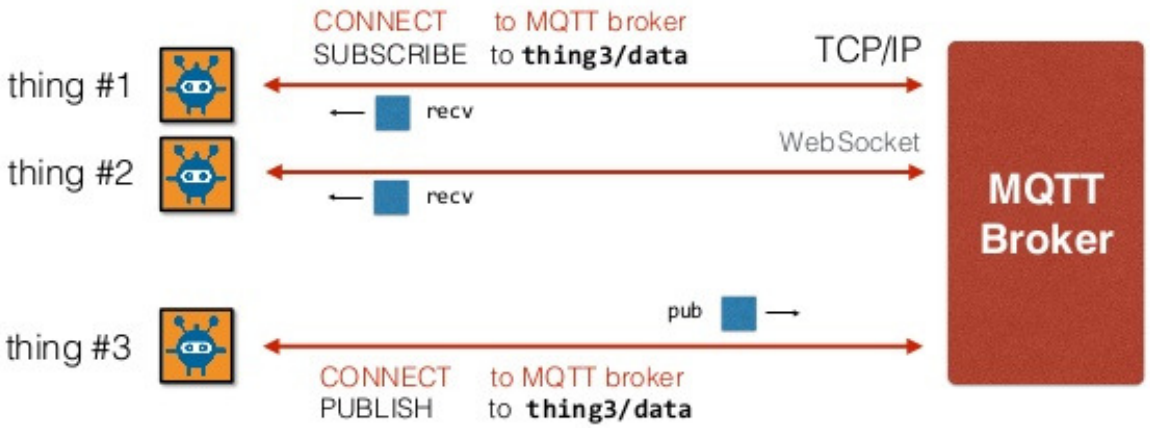
The publish/subscribe pattern

Bi-directional , Async "**Push**" Communication



MQTT Protocol

MQTT bi-directional, async "push" communication



MQTT Protocol

MQTT simple to implement

Connect

Subscribe

Publish

Unsubscribe

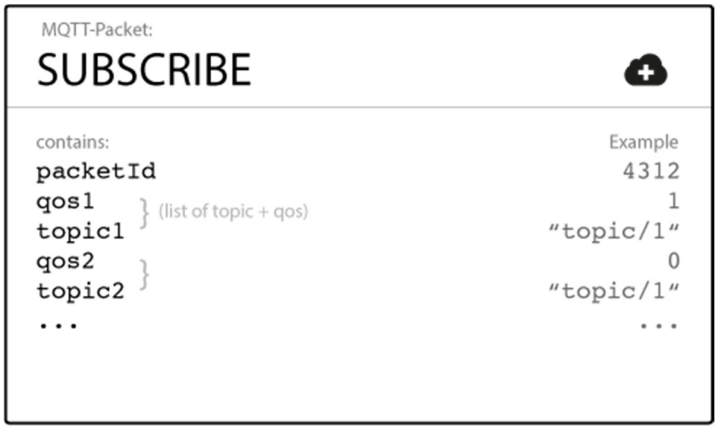
Disconnect

```
client = new Messaging.Client(hostname, port, clientId)
client.onMessageArrived = messageArrived;
client.onConnectionLost = connectionLost;
client.connect({ onSuccess: connectionSuccess });

function connectionSuccess() {
     client.subscribe("planets/earth");
     var msg = new Messaging.Message("Hello world!");
     msg.destinationName = "planets/earth";
     client.publish(msg);
}

function messageArrived(msg) {
     console.log(msg.payloadString);
     client.unsubscribe("planets/earth");
     client.disconnect();
}
```

Eclipse Paho JavaScript MQTT client

Subscribe

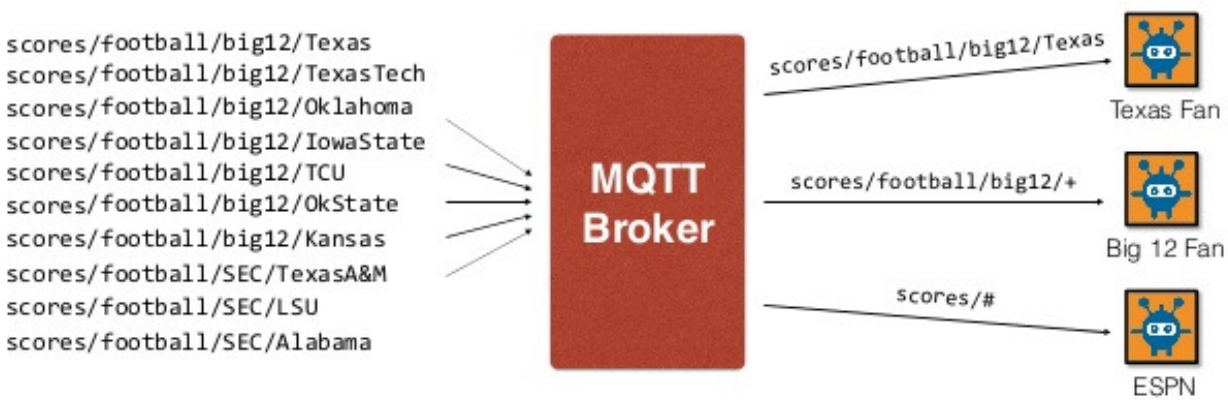
- A client needs to send a SUBSCRIBE message to the MQTT broker in order to receive relevant messages.



Subscribe

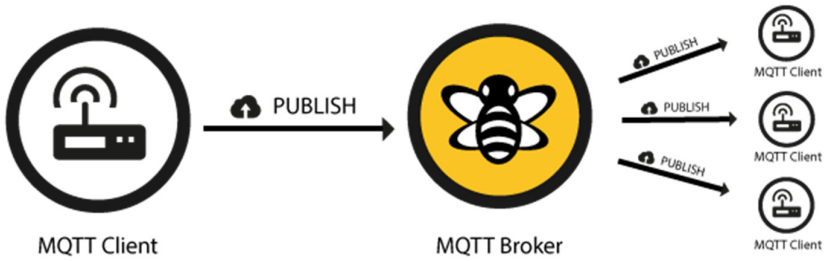
MQTT

allows wildcard subscriptions



single level wildcard: **+** multi-level wildcard: **#**

Publish



MQTT-Packet:

PUBLISH

contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

Publish

Topics
A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

QoS
A Quality of Service Level (QoS) for this message. The level (0,1 or 2) determines the guarantee of a message reaching the other end

Retain-Flag
This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing. (ให้ Broker เก็บ Payload ล่าสุด ที่ส่งขึ้นไปไว้ด้วย ถ้ามีคนใหม่เข้ามา Subscribe ทีหลัง ก็จะได้รับข้อมูลนี้ด้วย)

Payload
This is the actual content of the message.

DUP flag
The duplicate flag indicates, that this message is a duplicate and is resent because the other end didn't acknowledge the original message. This is only relevant for QoS greater than 0

Packet Identifier
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero.

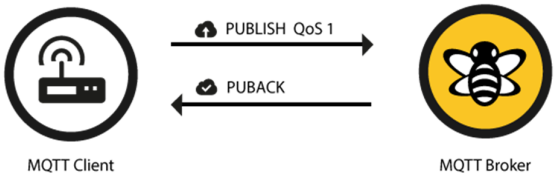
QoS

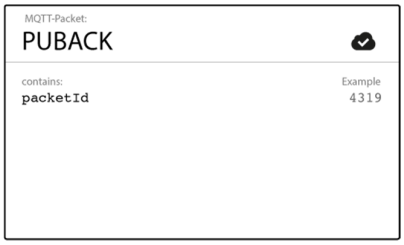
- **QoS 0 – at most once**

  The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender.

- **QoS 1 – at least once**

  it is guaranteed that a message will be delivered at least once to the receiver. The sender will store the message until it gets an acknowledgement in form of a PUBACK command message from the receiver.

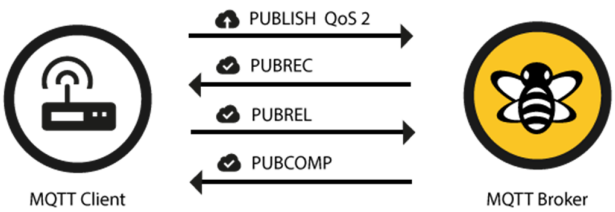




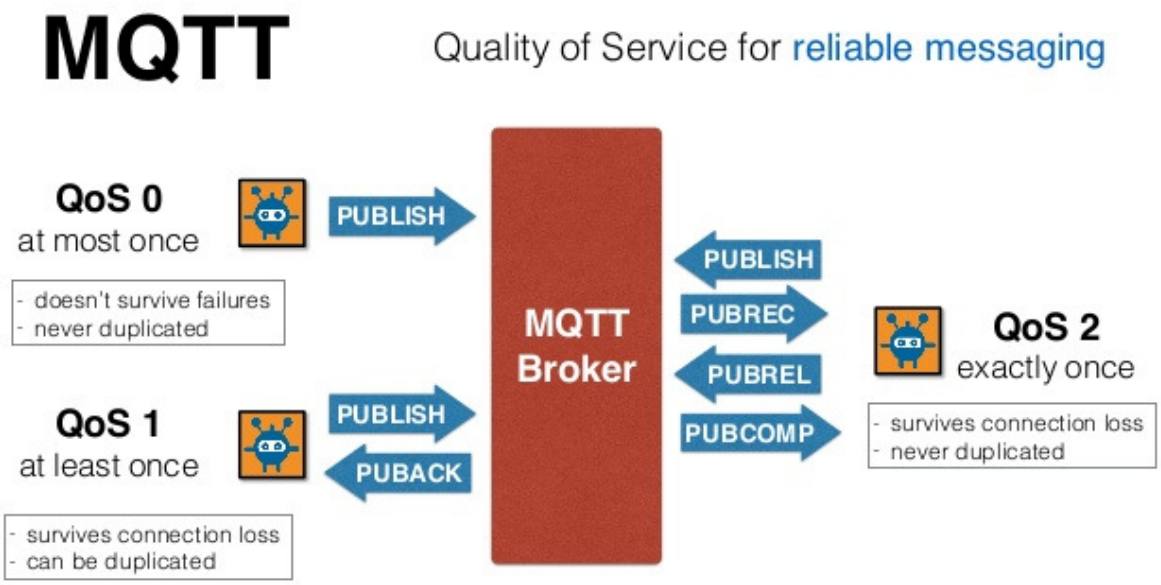
QoS

- **QoS 2 – Exactly once**

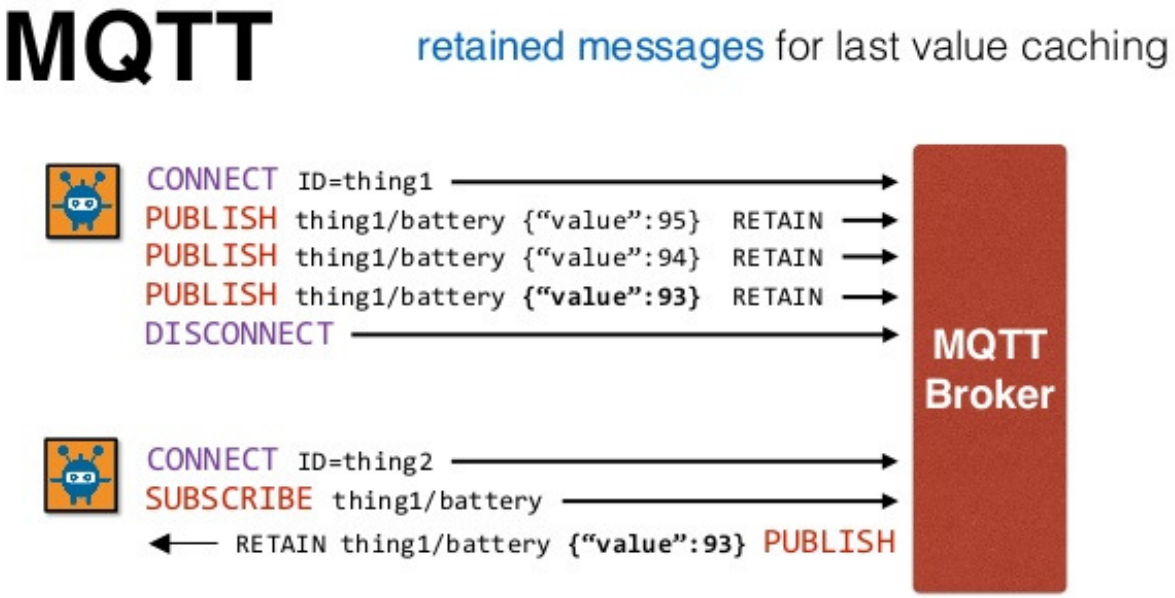
  The highest QoS is 2, it guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver.



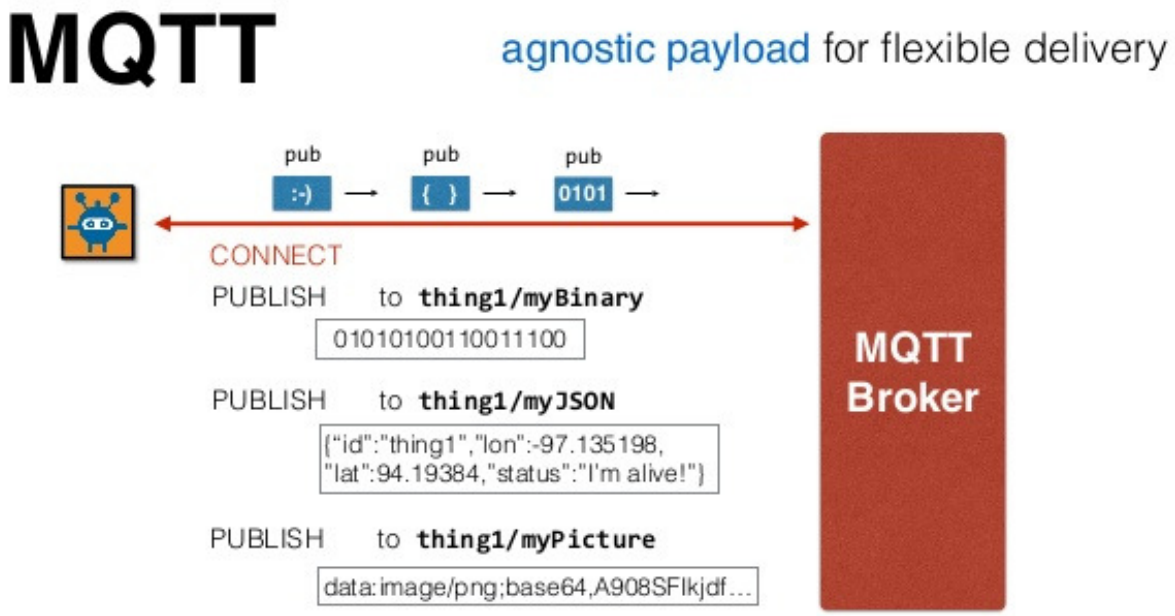
QoS



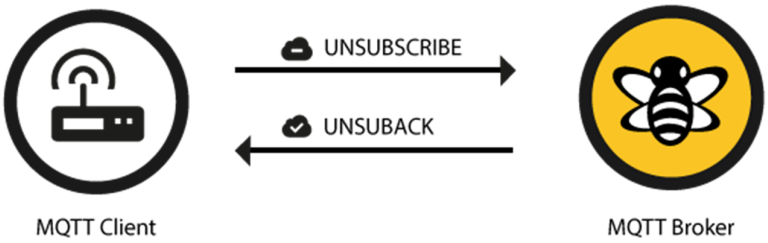
Retain messages



Payload



Unsubscribe



MQTT-Packet:

UNSUBSCRIBE

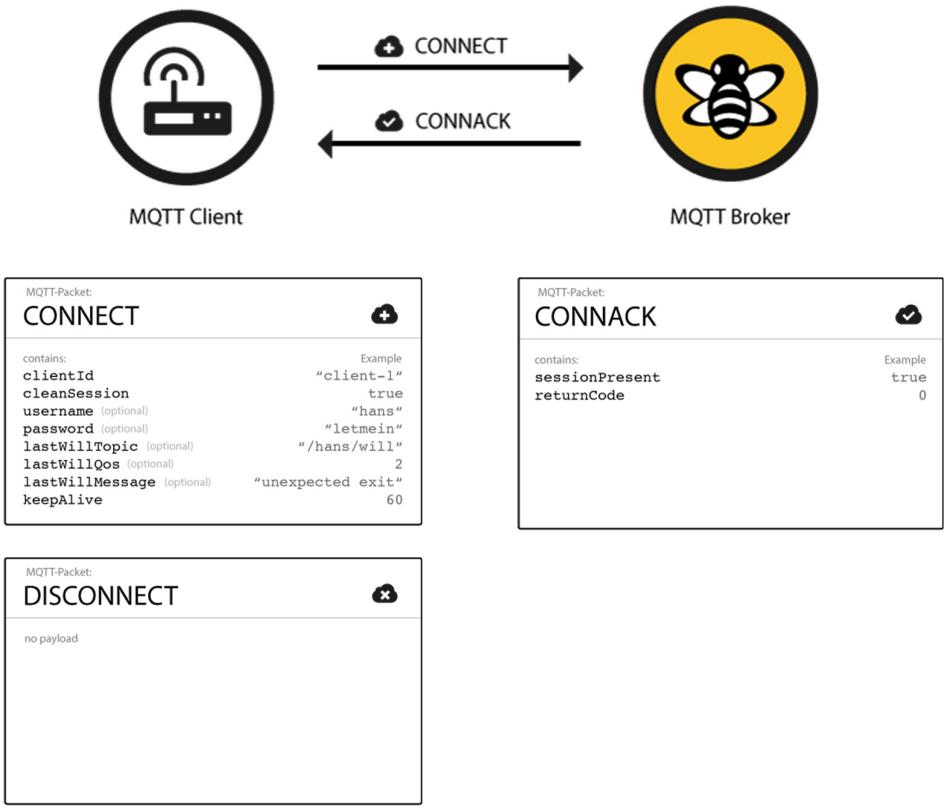
contains:	Example
packetId	4315
topic1 } (list of topics)	"topic/1"
topic2	"topic/2"
...	...

MQTT-Packet:

UNSUBACK

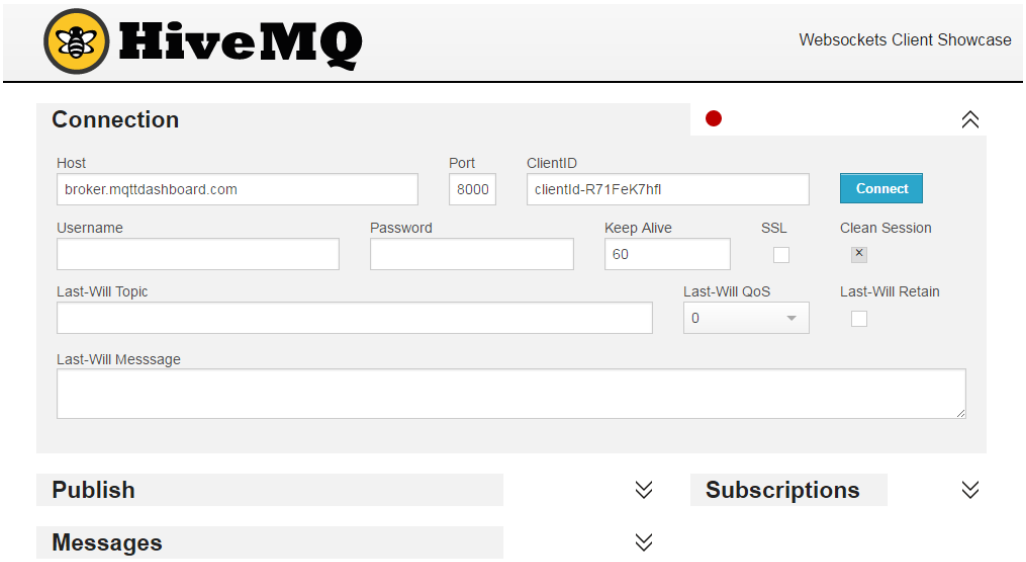
contains:	Example
packetId	4316

Connect & Disconnect



การทดลองที่ 1. Public MQTT Broker

- เข้า Website ที่ URL
  http://www.hivemq.com/demos/websocket-client/



การทดลองที่ 7.1 Public MQTT Broker

- **Connect**
  - กดปุ่ม Connect
- **Subscribe**
  - กด Add Topic Subscription เพื่อ add หัวข้อ ที่ต้องการ subscribe
  - กด Subscribe
- **Publish**
  - กำหนด topic ที่ต้องการจะ publish
  - พิมพ์ ข้อความ ที่จะ publish
  - กด publish

ติดตั้ง Library ของ MQTT Protocol

- Download Library Pub sub จาก

  https://github.com/knolleary/pubsubclient

  โดยการ download เป็น .zip

- แล้วติดตั้งที่ sketch -> Include Library -> Add .zip library
- เมื่อติดตั้งเสร็จแล้ว จะมี Library PubSubClient ปรากฏที่

  sketch -> Include Library

ข้อจำกัด

- It can only publish QoS 0 messages. It can subscribe at QoS 0 or QoS 1.
- The maximum message size, including header, is **128 bytes** by default.

PubSubClient Library

- **Constructors**
  - **PubSubClient** ()
  - **PubSubClient** (client)
  - **PubSubClient** (server, port, [callback], client, [stream]) ***
- **Functions**
  - boolean **connect** (clientID) ***
  - boolean **connect** (clientID, willTopic, willQoS, willRetain, willMessage)
  - boolean **connect** (clientID, username, password)
  - boolean **connect** (clientID, username, password, willTopic, willQoS, willRetain, willMessage)
  - void **disconnect** ()
  - int **publish** (topic, payload) ***
  - int **publish** (topic, payload, retained)
  - int **publish** (topic, payload, length)
  - int **publish** (topic, payload, length, retained)
  - int **publish_P** (topic, payload, length, retained)
  - boolean **subscribe** (topic, [qos]) ***
  - boolean **unsubscribe** (topic)
  - boolean **loop** ()
  - int **connected** ()
  - int **state** ()
  - PubSubClient **setServer** (server, port)
  - PubSubClient **setCallback** (callback)
  - PubSubClient **setClient** (client)
  - PubSubClient **setStream** (stream)
- **Other**
  - Configuration Options
  - Subscription Callback

Connect NodeMCU to Wireless Network

`WiFi.begin()`

Description

Initializes the WiFi library's network settings and provides the current status.

Syntax

```
WiFi.begin();
WiFi.begin(ssid);
WiFi.begin(ssid, pass);
```

Returns

`WL_CONNECTED` when connected to a network

`WL_IDLE_STATUS` when not connected to a network, but powered on

Connect NodeMCU to Wireless Network

`WiFi.status()`

Description

Return the connection status.

Syntax

```
WiFi.status();
```

Returns

`WL_CONNECTED`: assigned when connected to a WiFi network;
`WL_IDLE_STATUS`: it is a temporary status assigned when `WiFi.begin()` is called and remains active until the number of attempts expires
`WL_NO_SSID_AVAIL`: assigned when no SSID are available;
`WL_CONNECT_FAILED`: assigned when the connection fails for all the attempts;
`WL_CONNECTION_LOST`: assigned when the connection is lost;
`WL_DISCONNECTED`: assigned when disconnected from a network;

Connect NodeMCU to Wireless Network

`WiFi.localIP()`

Description

Gets the WiFi shield's IP address

Syntax

```
WiFi.localIP();
```

Returns

the IP address

`WiFi.disconnect()`

Description

Disconnects the WiFi from the current network.

Syntax

```
WiFi.disconnect();
```

Returns

nothing

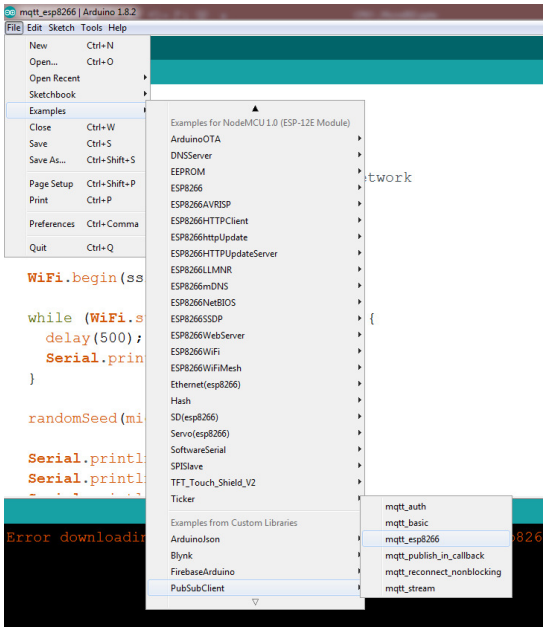
Connect NodeMCU to Wireless Network

ตัวอย่างโปรแกรม

```
#include <ESP8266WiFi.h>

const char* ssid = "SSID_name";
const char* password = "PASSWORD";

void setup() {
  Serial.begin(115200);
  delay(10);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

  Serial.println("");
  Serial.println("WiFi connected");

  // Print the IP address

  Serial.print("IP=");
  Serial.println(WiFi.localIP());

}

void loop() {

}
```

MQTT with NodeMCU

- เปิดโปรแกรมตัวอย่าง
- File => Example => PubsubClient => Mqtt_esp8266



MQTT with NodeMCU

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

// Update these with values suitable for your network.

const char* ssid = "........";
const char* password = "........";
const char* mqtt_server = "broker.mqtt-dashboard.com";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

1. ใส่ค่า ssid และ password

2. แก้ mqtt_server เป็น broker.mqttdashboard.com

MQTT with NodeMCU

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

  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());
}
```

ส่วนนี้เป็นการเชื่อมต่อ WIFI ถ้ายังต่อไม่ติดก็จะขึ้น ...

ถ้าต่อติดก็จะแสดงคำว่า WIFI connected และแสดงค่า IP address ที่ได้รับจาก Access point

MQTT with NodeMCU

```
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();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note that
    // but actually the LED is on; this is because
    // it is acive low on the ESP-01)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off by making
  }
}
```

ส่วนนี้เป็น function callback

ทำหน้าที่แสดงค่าที่ได้จากการ Subscribe

โดยแสดง Topic และ payload ที่ subscribe มา

MQTT with NodeMCU

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

ส่วนนี้เป็น function reconnect ใช้สำหรับเชื่อมต่อกับ Broker
โดย ClientID จะต้องไม่ซ้ำกัน กับคนอื่นๆใน broker นั้นๆ
เมื่อเชื่อมต่อติดแล้ว ก็จะ publish คำว่า hello world ไป 1 ครั้ง

3. แก้ ClientID อย่าให้ซ้ำกัน

4. แก้ Topic ที่ต้องการ Publish/ Subscribe
แก้ข้อความที่จะ publish เมื่อเชื่อมต่อกับ broker สำเร็จ

MQTT with NodeMCU

```
void setup() {
  pinMode(BUILTIN_LED, OUTPUT);      // Initialize
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
```

ส่วนนี้เป็น ส่วน setup
สำหรับตั้งค่าตอนเริ่มต้น

MQTT with NodeMCU

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

  long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;
    snprintf (msg, 75, "hello world #%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("outTopic", msg);
  }
}
```

ส่วนนี้เป็น ส่วนหลักของโปรแกรม
โปรแกรมจะวน loop รอไปเรื่อยๆ
เมื่อมีการ publish มา ก็จะไปทำงานที่
ฟังก์ชั่น Callback เพื่อแสดงค่า

ในขณะเดียวกัน ทุกๆ 2 วินาที
ก็จะ publish ค่า ไปยัง topic ที่กำหนด

5. แก้ Topic ที่ต้องการ Publish

MQTT with NodeMCU

เมื่อแก้โปรแกรมเสร็จแล้ว ทำการทดลองโดยใช้ HiveMQ
http://www.hivemq.com/demos/websocket-client/

ทำการ publish มาที่ topic ที่กำหนด
และใช้ Serial monitor ดูค่าที่ Subscribe ได้

และในทำนองเดียวกัน ใช้ HiveMQ ทำการ Subscribe ข้อความที่ส่งมาจาก NodeMCU