MQTT

- **Message Queue Telemetry Transport.**
- ISO standard (ISO/IEC PRF 20922).
- It is a publish-subscribe-based lightweight messaging protocol for use in conjunction with the TCP/IP protocol.
- MQTT was introduced by IBM in 1999 and standardized by OASIS in 2013.
- Designed to provide connectivity (mostly embedded) between applications and middle-wares on one side and networks and communications on the other side.

MQTT

- A message broker controls the publish-subscribe messaging pattern.
- A topic to which a client is subscribed is updated in the form of messages and distributed by the message broker.
- Designed for:
  - Remote connections
  - Limited bandwidth
  - Small-code footprint

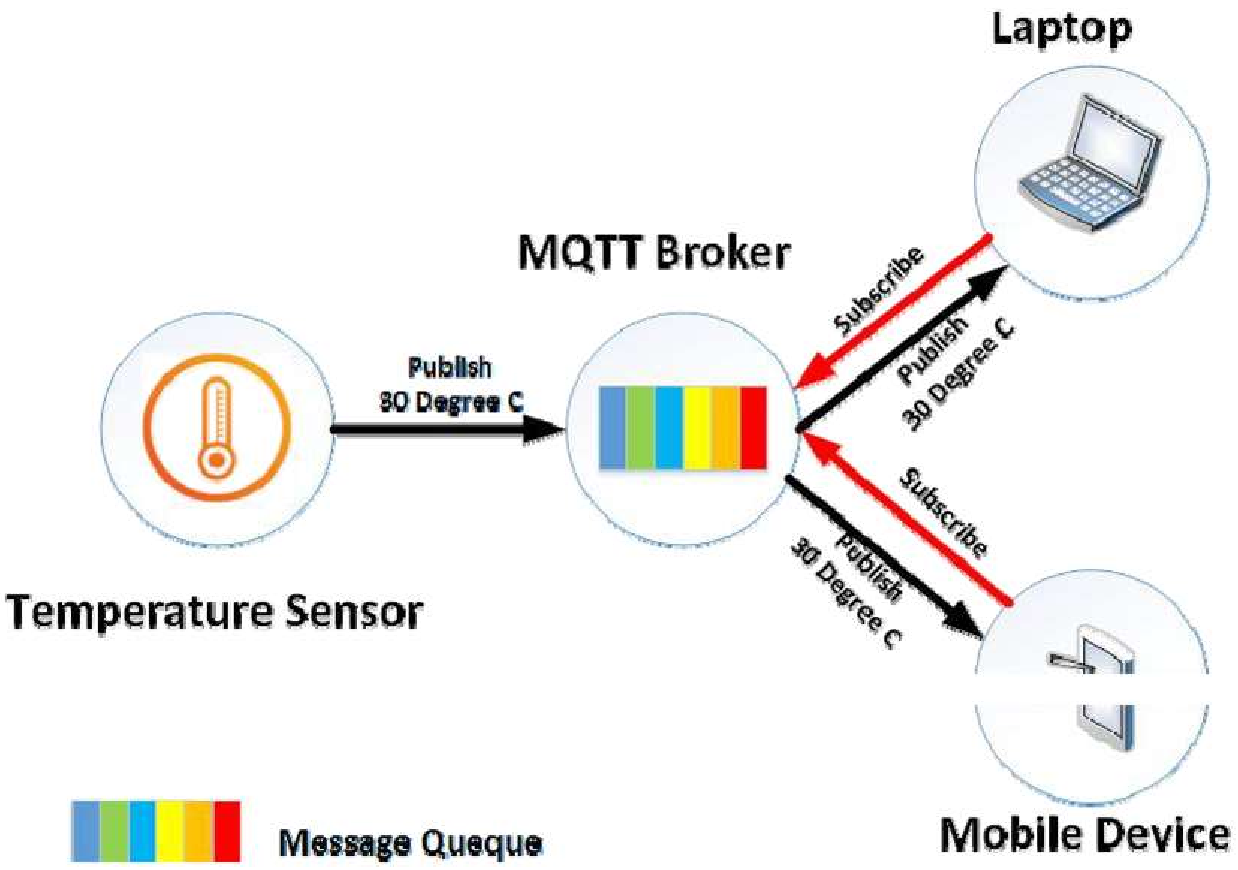
MQTT Components

- Publishers
  - Lightweight sensors
- Subscribers
  - Applications interested in sensor data
- Broker
  - Connect publishers and subscribers
  - Classify sensor data into topics

MQTT Methods

- Connect
- Disconnect
- Subscribe
- Unsubscribe
- Publish

MQTT Communication Model



MQTT Communication

- The protocol uses a **publish/subscribe** architecture (HTTP uses a request/response paradigm).
- Publish/subscribe is **event-driven** and enables messages to be pushed to clients.
- The central **communication point is the MQTT broker**, which is in charge of dispatching all messages between the senders and the rightful receivers.
- Each client that publishes a message to the broker, includes a **topic** into the message. The **topic is the routing information for the broker.**

MQTT Communication

- Each client that wants to receive messages subscribes to a certain topic and the broker delivers all messages with the matching topic to the client.
- Therefore the clients don’t have to know each other. They only communicate over the topic.
- This architecture enables highly scalable solutions without dependencies between the data producers and the data consumers.

MQTT Topic

- A topic is a **simple string** that can have more hierarchy levels, which are separated by a slash.
- A sample topic for sending temperature data of the living room could be *house/living-room/temperature*.
- On one hand the client (e.g. mobile device) can subscribe to the exact topic or on the other hand, it can use a **wildcard**.

MQTT Topic

- The subscription to *house/+/temperature* would result in all messages sent to the previously mentioned topic *house/livingroom/ temperature,* as well as any topic with an arbitrary value in the place of living room, such as *house/kitchen/temperature*.
- The plus sign is a **single level wild card** and only allows arbitrary values for one hierarchy.
- If more than one level needs to be subscribed, such as, the entire sub-tree, there is also a **multilevel wildcard** (#).
- It allows to subscribe to all underlying hierarchy levels.
- For example *house/#* is subscribing to all topics beginning with *house*.

Applications

- **Facebook Messenger** uses MQTT for online chat.
- **Amazon Web Services** use Amazon IoT with MQTT.
- **Microsoft Azure** IoT Hub uses MQTT as its main protocol for telemetry messages.
- The **EVRYTHNG IoT platform** uses MQTT as an M2M protocol for millions of connected products.
- **Adafruit** launched a free MQTT cloud service for IoT experimenters called Adafruit IO.

SMQTT

- **Secure MQTT** is an extension of MQTT which uses encryption based on lightweight attribute.
- The main advantage of using such encryption is the broadcast encryption feature, in which one message is encrypted and delivered to multiple other nodes, which is quite common in IoT applications.
- In general, the algorithm consists of four main stages: setup, encryption, publish and decryption.

SMQTT

- In the setup phase, the subscribers and publishers register themselves to the broker and get a master secret key according to their developer's choice of key generation algorithm.
- When the data is published, it is encrypted and published by the broker which sends it to the subscribers, which is finally decrypted at the subscriber end having the same master secret key.
- The key generation and encryption algorithms are not standardized.
- SMQTT is proposed only to enhance MQTT security features.

CoAP

- CoAP – **Constrained Application Protocol**.
- **Web transfer protocol** for use with constrained nodes and networks.
- **Designed for Machine to Machine** (M2M) applications such as smart energy and building automation.
- Based on **Request-Response model** between end-points
- Client-Server interaction is **asynchronous over a datagram oriented transport protocol** such as UDP

Where are we?

We are here!

HTTP, Websockets, DNS, XMPP, MQTT, CoAp	Application layer
TLS, SSL	Application Layer (Encryption)
TCP, UDP	Transport
IP(V4, V6), 6LowPAN	Internet Layer
Ethernet, 802.11 WiFi, 802.15.4	Link Layer

- The Constrained Application Protocol (CoAP) is a session layer protocol designed by IETF Constrained RESTful Environment (CoRE) working group to provide lightweight RESTful (HTTP) interface.
- Representational State Transfer (REST) is the standard interface between HTTP client and servers.
- Lightweight applications such as those in IoT, could result in significant overhead and power consumption by REST.
- CoAP is designed to enable low-power sensors to use RESTful services while meeting their power constraints.

- Built over UDP, instead of TCP (which is commonly used with HTTP) and has a light mechanism to provide reliability.
- CoAP architecture is divided into two main sub-layers:
  - ○ Messaging
  - ○ Request/response.
- The messaging sub-layer is responsible for reliability and duplication of messages, while the request/response sub-layer is responsible for communication.
- CoAP has four messaging modes:
  - ○ Confirmable
  - ○ Non-confirmable
  - ○ Piggyback
  - ○ Separate

Constrained Application Protocol (CoAp)

Who uses or supports CoAP?

- Open Mobile Alliance M2M
- IPSO Alliance (IP for Smart Objects)
- European Telecom Standards Institute M2M / OneM2M
- Lighting systems for smart cities
- Device management for network operators.
- Copper is a Firefox plugin – treat devices as REST services
- Main Java project on github : Californium

CoAP

Has a scheme coap://

Has a well known port.

GET, POST, PUT, DELETE encoded in binary (1 == GET)

Block transfer support.

Confirmable messages requires an ACK with message ID. The message ID of the ACK matches the message ID of the confirmable message.

Non-confirmable messages do not require an ACK. Less reliable.

Responses are matched with requests via the client generated Token.

Example:

```
  CoAP Client                         CoAP Server
           ---->   CON  {id} GET  /basement/light             Confirmable request has an ID
           <----   ACK  {id} 200 Content {“status” : “on”}     Piggy back response and same ID
```

CoAP Uses Timeouts over UDP

```
CoAP Client                        CoAP Server
            ---->   CON  {id} GET  /basement/light        lost request
 timeout -- -->   CON  {id} GET /basement/light         finally arrives
            <----   ACK  {id} 200 Content {“status” : “on”}
```

The {id} allows us to detect duplicates.
What happens if the ACK is also lost?

CoAP Request/Acknowledge/Callback

```
CoAP Client                          CoAP Server
    ---->   CON  {id} PUT  /basement/cleanFloor   Token:  0x22   Needs time
    <----   ACK  {id}       I am on it!
    <-----  CON {newID} 200 Content /basement/cleanFloor Token: 0x22 Done
    ---->   ACK {newID}
```

In this example, the same token is used to identify this request and the service response.

The id’s are used at the message level.

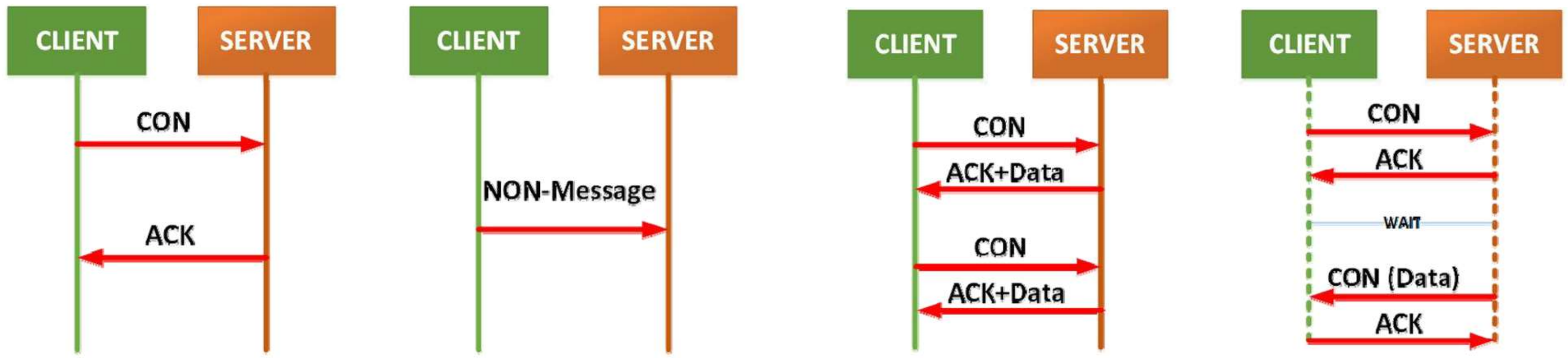
CoAP Publish/Subscribe

- The GET includes an “Observe” message to establish a subscription request.
- The response includes an “Observe” to say this is a publication.
- The value included with Observe response is there for possible re-orderings. The client
- should take the most recent sent and not the most recent to arrive.

```
CoAP Client                      CoAP Server
   ---->   CON  {id} GET  /basement/light Observe: 0   Token:  0x22
   <----   ACK  200 {id}  Observe: 27 Token 0x22
   <----   CON 200 Observe: 28 Token: 0x22 {“light” : ”off”}
   ----->  ACK Token: 0x22
   <----   CON 200 Observe: 30 Token: 0x22 {“light” : ”on”}
           :
           :
           etc.
```

Block transfer is similar. We may request a transfer (one block at a time).

CoAP Request Response Model



Confirmable Message

Non-Confirmable Message

Piggyback Message

Separate Message

Features

- Reduced overheads and parsing complexity.
- URL and content-type support.
- Support for the discovery of resources provided by known CoAP services.
- Simple subscription for a resource, and resulting push notifications.
- Simple caching based on maximum message age.

XMPP

- **XMPP – Extensible Messaging and Presence Protocol.**
- A communication protocol for **message-oriented middleware** based on XML (Extensible Markup Language).
- Real-time exchange of structured data.
- It is an open standard protocol.

XMPP

- XMPP uses a **client-server architecture**.
- As the model is **decentralized**, no central server is required.
- XMPP provides for the **discovery of services** residing locally or across a network, and the **availability information** of these services.
- Well-suited for cloud computing where virtual machines, networks, and firewalls would otherwise present obstacles to alternative service discovery and presence-based solutions.
- Open means to support machine-to-machine or peer-to-peer communications across a diverse set of networks.

Features

- Decentralization – No central server; anyone can run their own XMPP server.
- Open standards – No royalties or granted permissions are required to implement these specifications
- Security – Authentication, encryption, etc.
- Flexibility – Supports interoperability

Application

- Publish-subscribe systems
- Signaling for VoIP
- Video
- File transfer
- Gaming
- Smart grid
- Social networking services

Weakness

- Does not support QoS.
- Text based communications induces higher network overheads.
- Binary data must be first encoded to base64 before transmission.