

Application Layer Protocols : MQTT

Dr. Bibhas Ghoshal

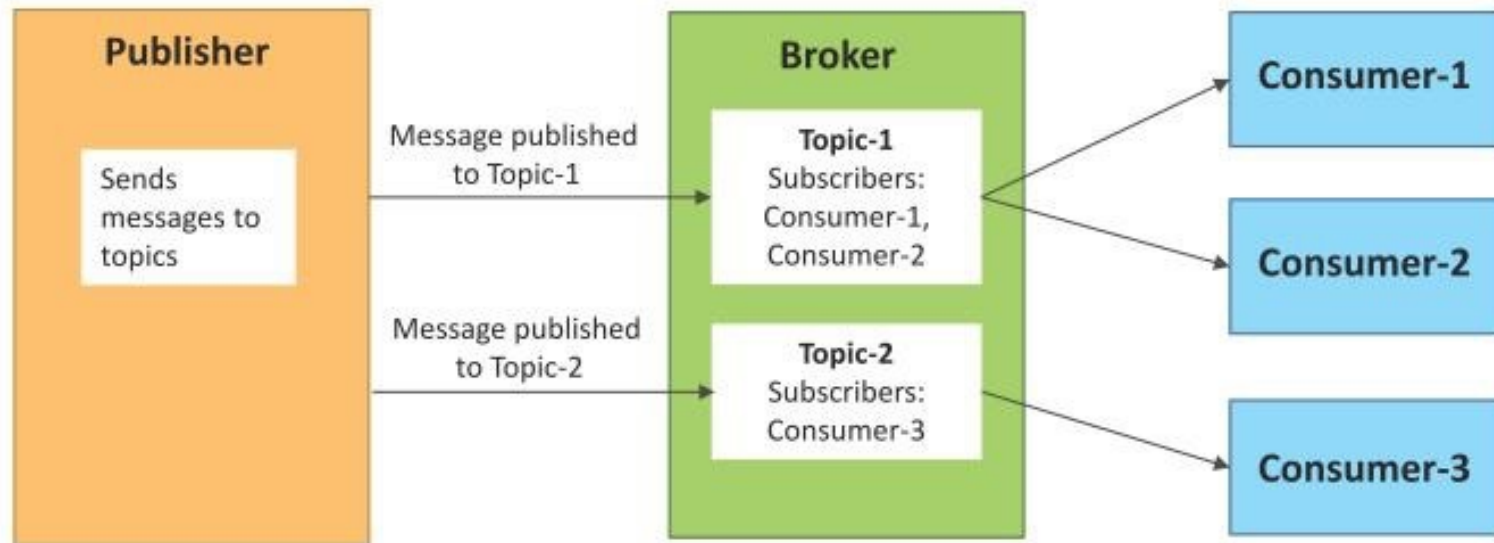
IIIT Allahabad

Data Protocols in IoT

- Devices talk to each other – **devices northbound and southbound**
- Gateways talk to the **cloud northbound and devices southbound**
- Device to Device (**D2D**)
- Device to Cloud (**D2C**)

MQTT Protocol

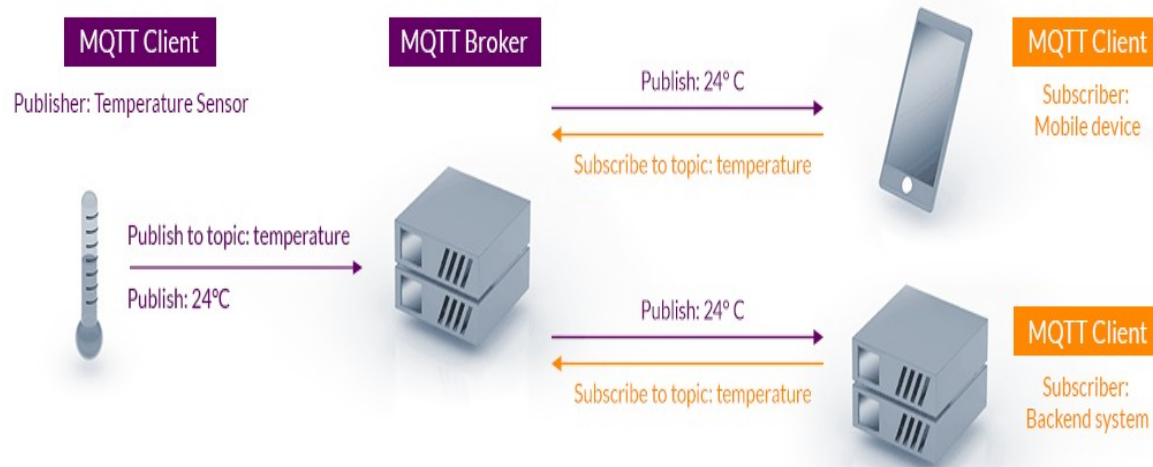
- MQTT - Message Queuing Telemetry Transport
- Enables a publish/subscribe messaging model in an extremely lightweight way.
- Useful for connections with remote locations where a small code footprint is required
- Publish Subscribe Model :



MQTT : History and Requirements

- Invented in **1999** by **Andy Stanford-Clark (IBM)** and **Arlen Nipper (Arcom, now Cirrus Link)**
- **Protocol for minimal battery loss and minimal bandwidth to connect with oil pipelines via satellite**
- **Core Features of MQTT:**
 - Simple implementation – arbitrary messages upto 256MB
 - Quality of Service data delivery – in order deliver per publisher
 - Lightweight and bandwidth efficient – little client state, TCP/Websockets
 - Data agnostic
 - Continuous session awareness

MQTT Architecture



- **Space Decoupling** - Pub. And Sub. do not know each other (ip/port)
- **Time Decoupling** – do not have to be actively connected all time
- **Synchronization Decoupling** – sending/ receiving at own speed
- **Scalability**
- **Message Filtering**

MQTT Quality of Service (QoS) Levels

QoS 0 : At most once “ Fire and Forget”, no confirmation

Guaranteed delivery

QoS 1 : At least once , with confirmation required

(msgs may delivered more than once)

QoS 2 : Exactly once , 2-phase commit

MQTT supports *Persistent Messages*

Ideal for internet connectivity

Automatic keep alive messages

QoS 1 and 2 messages are queued for clients which may be offline

But not timed out

Disconnect, Last Will & Testament and Retained Messages

Clients which disconnect intentionally use Disconnect message

MQTT broker will automatically publish Last Will and Testament message on behalf of clients with unintentionally terminated messages

MQTT supports Retained messages which are automatically delivered when clients subscribes to a topic

MQTT Vs. HTTP

	MQTT	HTTP
Purpose	Messaging	Documents
Protocol Efficiency	High	Average
Power Efficiency	Yes	No
Client Languages	Many	Many

Message Filtering by Broker

- **Subject based Filtering**

filtering is based on the subject or topic that is part of each message. The receiving client subscribes to the broker for topics of interest

- topics are strings with a hierarchical structure that allow filtering based on a limited number of expressions

- **Content based Filtering**

- the broker filters the message based on a specific content filter-language.
- The receiving clients subscribe to filter queries of messages for which they are interested.
- content of the message must be known beforehand and cannot be encrypted or easily changed.

- **Type based Filtering**

- filtering based on the type/class of a message (event) in case Object Oriented languages

MQTT Vs. Message Queues

- **A message queue stores message until they are consumed**
- **A message is only consumed by one client**
- **Queues are named and must be created explicitly**

Client, Broker and Connection Establishment

• **Client :**

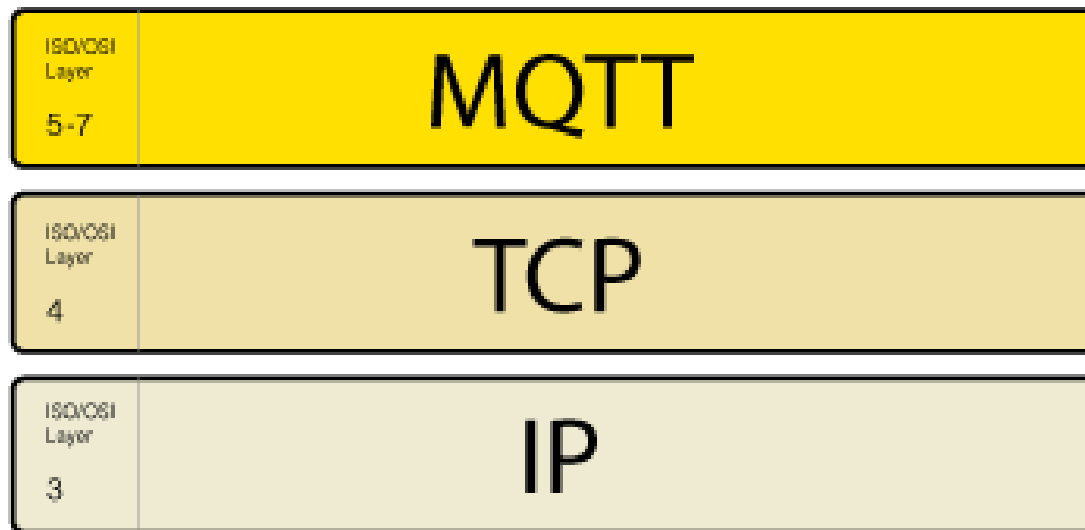
- any device (from a micro controller up to a full-fledged server) that runs an MQTT library and connects to an MQTT broker over a network. Any device that speaks MQTT over a TCP/IP stack can be called an MQTT client.
- MQTT client libraries are available for a huge variety of programming languages. For example, Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, and .NET.

• **Broker :**

- responsible for receiving all messages, filtering the messages, determining who is subscribed to each message, and sending the message to these subscribed clients.
- holds the session data of all clients that have persistent sessions, including subscriptions and missed messages
- authentication and authorization of clients.

MQTT Connection

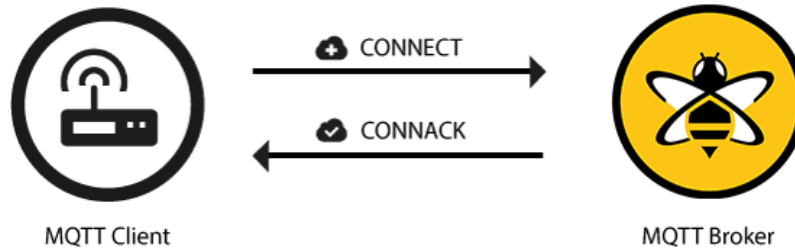
MQTT protocol is based on TCP/IP. Both the client and the broker need to have a TCP/IP stack.




MQTT connection is always between one client and the broker. Clients never connect to each other directly

MQTT Connection

Initiation :



- client sends a **CONNECT** message to the broker.
- broker responds with a **CONNACK** message and a status code.
- Once the connection is established, the broker keeps it open until the client sends a disconnect command or the connection breaks

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

MQTT Packet Components

ClientId: identifies each MQTT client that connects to an MQTT broker. The broker uses the ClientId to identify the client and the current state of the client. Therefore, this Id should be unique per client and broker.

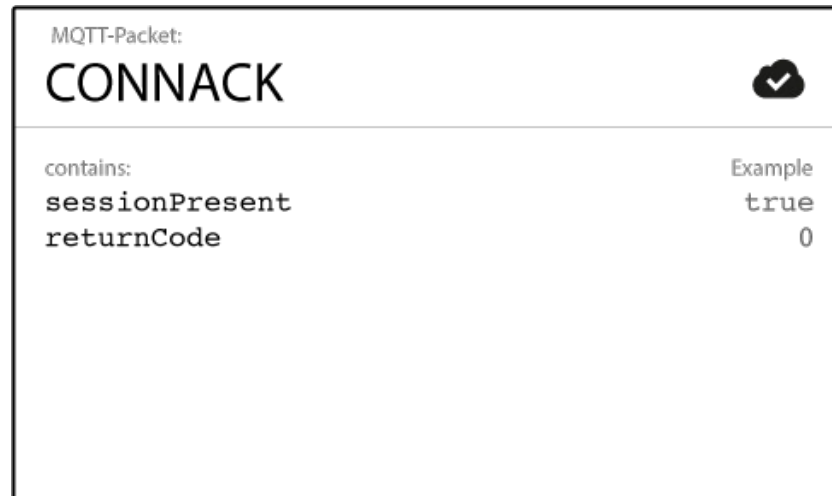
Clean Session : The clean session flag tells the broker whether the client wants to establish a persistent session or not. In a persistent session (CleanSession = false), the broker stores all subscriptions for the client and all missed messages for the client

Username/Password : MQTT can send a user name and password for client authentication and authorization.

Will Message : notifies other clients when a client disconnects ungracefully.

KeepAlive : time interval in seconds that the client specifies and communicates to the broker when the connection established. It defines the longest period of time that the broker and client can endure without sending a message.

Broker Resposne



Session Present flag : The session present flag tells the client whether the broker already has a persistent session available from previous interactions with the client.

Connect return code : tells the client whether the connection attempt was successful or not. (0 = connection accepted)

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

MQTT client can publish messages as soon as it connects to a broker.

MQTT utilizes topic-based filtering of the messages on the broker

Each message must contain a topic that the broker can use to forward the message to interested clients. Typically, each message has a payload which contains the data to transmit in byte format.

use case of the client determines how the payload is structured.

The sending client (publisher) decides whether it wants to send binary data, text data, or even full-fledged XML or JSON.

Publish Message Components

Topic Name : simple string that is hierarchically structured with forward slashes as delimiters. For example, “myhome/livingroom/temperature”

QoS : indicates the Quality of Service Level (QoS) of the message. There are three levels: 0, 1, and 2. The service level determines what kind of guarantee a message has for reaching the intended recipient (client or broker)

Retain Flag : defines whether the message is saved by the broker as the last known good value for a specified topic. When a new client subscribes to a topic, they receive the last message that is retained on that topic.

Payload : actual content of the message. MQTT is data-agnostic. It is possible to send images, text in any encoding, encrypted data, and virtually every data in binary.

Packet Identifier : uniquely identifies a message as it flows between the client and broker. The packet identifier is only relevant for QoS levels greater than zero. The client library and/or the broker is responsible for setting this internal MQTT identifier.

DUP flag : indicates that the message is a duplicate and was resent because the intended recipient (client or broker) did not acknowledge the original message. This is only relevant for QoS greater than 0.

Subscribe

Publishing a message doesn't make sense if no one ever receives it. In other words, if there are no clients to subscribe to the topics of the messages. To receive messages on topics of interest, the client sends a SUBSCRIBE message to the MQTT broker. This subscribe message is very simple, it contains a unique packet identifier and a list of subscriptions.



Packet Identifier : uniquely identifies a message as it flows between the client and broker. The client library and/or the broker is responsible for setting this internal MQTT identifier.

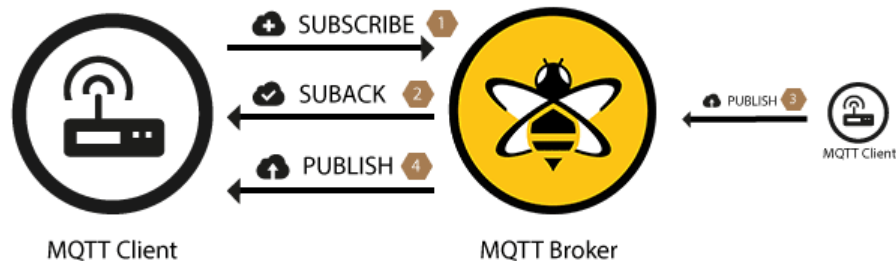
List of Subscriptions : A SUBSCRIBE message can contain multiple subscriptions for a client. Each subscription is made up of a topic and a QoS level. The topic in the subscribe message can contain wildcards that make it possible to subscribe to a topic pattern rather than a specific topic. If there are overlapping subscriptions for one client, the broker delivers the message that has the highest QoS level for that topic.

Subscribe Acknowledgement

MQTT-Packet:	
SUBACK	
contains:	
packetId	Example
returnCode 1 (one returnCode for each topic from SUBSCRIBE, in the same order)	4313
returnCode 2	2
returnCode 3	0
...	...

Packet Identifier : unique identifier used to identify a message, same as in the SUBSCRIBE msg

Return Code : The broker sends one return code for each topic/QoS-pair that it receives in the SUBSCRIBE message.



Unsubscribe

MQTT-Packet:

UNSUBSCRIBE

contains:

<code>packetId</code>	Example
<code>topic1</code> } (list of topics)	4315
<code>topic2</code>	"topic/1"
<code>...</code>	"topic/2"
	...

MQTT-Packet:

UNSUBACK

contains:

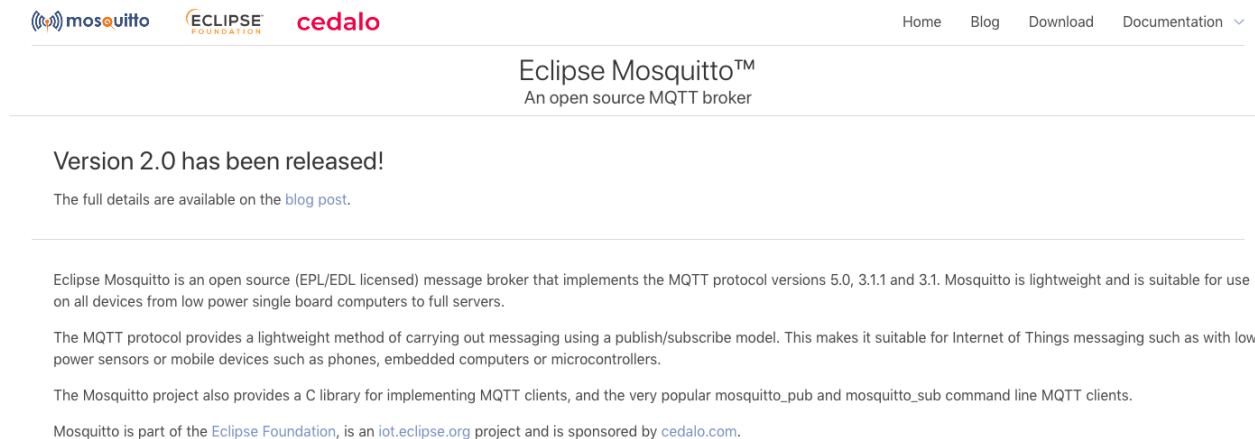
<code>packetId</code>	Example
	4316

Mosquitto MQTT Broker

Mosquitto : lightweight open source message broker

Implements MQTT versions 3.1.0, 3.1.1 and version 5.0

Main website : <https://mosquitto.org/>



The screenshot shows the Eclipse Mosquitto website homepage. At the top, there are logos for Mosquitto, Eclipse Foundation, and Cedalo. A navigation menu includes links for Home, Blog, Download, and Documentation. The main heading reads "Eclipse Mosquitto™ An open source MQTT broker". A prominent announcement states "Version 2.0 has been released!" with a link to a blog post for full details. Below this, a paragraph describes Mosquitto as an open source (EPL/EDL licensed) message broker implementing MQTT versions 5.0, 3.1.1, and 3.1, suitable for various devices. Another paragraph explains the MQTT publish/subscribe model. A third paragraph mentions the C library and command-line clients. The footer notes that Mosquitto is part of the Eclipse Foundation project and is sponsored by Cedalo.com.

A nice tutorial on MOSQUITTO is available at :

<https://www.switchdoc.com/2018/02/tutorial-installing-and-testing-mosquitto-mqtt-on-raspberry-pi/>