# MQTT

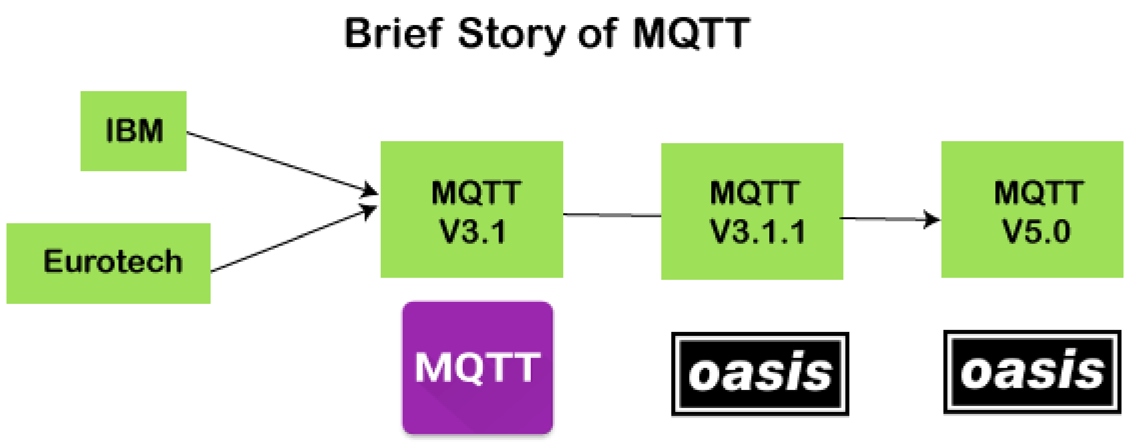
MQTT stands for **Message Queuing Telemetry Transport**. MQTT is a machine to machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications.

## Characteristics of MQTT

The MQTT has some unique features which are hardly found in other protocols. Some of the features of an MQTT are given below:

- It is a machine to machine protocol, i.e., it provides communication between the devices.
- It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe system to exchange the information between the client and the server.
- It does not require that both the client and the server establish a connection at the same time.
- It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
- It allows the clients to subscribe to the narrow selection of topics so that they can receive the information they are looking for.

## History of MQTT



The MQTT was developed by Dr. Andy Stanford-Clark, IBM, and Arlen Nipper. The previous versions of protocol 3.1 and 3.1.1 were made available under MQTT ORG. In 2014, the MQTT was officially published by OASIS. The OASIS becomes a new home for the development of the MQTT. Then, the OASIS started the further development of the MQTT. Version 3.1.1 is backward comfortable with a 3.1 and brought only minor changes such as changes to the connect message and clarification of the 3.1 version. The recent version of MQTT is 5.0, which is a successor of the 3.1.1 version. Version 5.0 is not backward, comfortable like version 3.1.1. According to the specifications, version 5.0 has a significant number of features that make the code in place.

The major functional objectives in version 5.0 are:

- Enhancement in the scalability and the large-scale system in order to set up with the thousands or the millions of devices.
- Improvement in the error reporting

## MQTT Architecture

**To understand the MQTT architecture, we first look at the components of the MQTT.**

- **Message**
- **Client**
- **Server or Broker**
- **TOPIC**

## Message

The message is the data that is carried out by the protocol across the network for the application. When the message is transmitted over the network, then the message contains the following parameters:

1. Payload data
2. Quality of Service (QoS)
3. Collection of Properties
4. Topic Name

**Client**

In MQTT, the subscriber and publisher are the two roles of a client. The clients subscribe to the topics to publish and receive messages. In simple words, we can say that if any program or device uses an MQTT, then that device is referred to as a client. A device is a client if it opens the network connection to the server, publishes messages that other clients want to see, subscribes to the messages that it is interested in receiving, unsubscribes to the messages that it is not interested in receiving, and closes the network connection to the server.

In MQTT, the client performs two operations:

In MQTT, the client performs two operations:

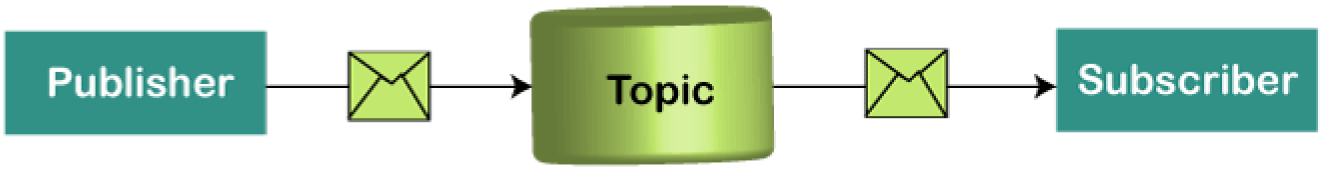
**Publish:** When the client sends the data to the server, then we call this operation as a publish.

**Subscribe:** When the client receives the data from the server, then we call this operation a subscription.

**Server**

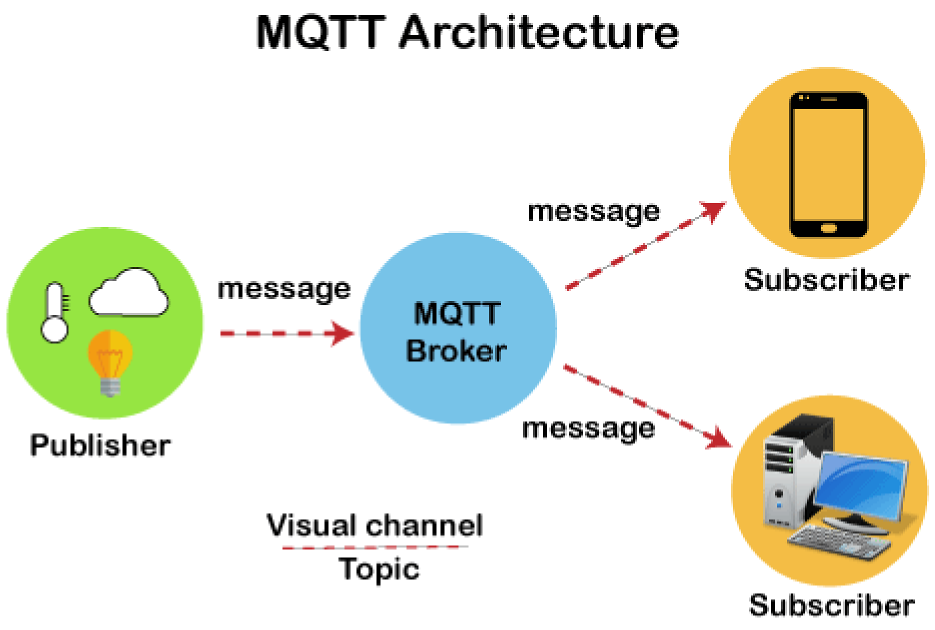
The device or a program that allows the client to publish the messages and subscribe to the messages. A server accepts the network connection from the client, accepts the messages from the client, processes the subscribe and unsubscribe requests, forwards the application messages to the client, and closes the network connection from the client.

**TOPIC**



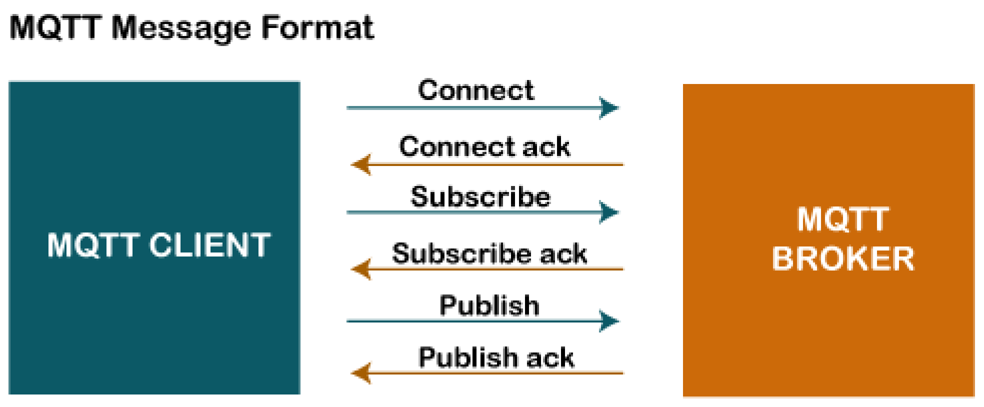
The label provided to the message is checked against the subscription known by the server is known as TOPIC.

## Architecture of MQTT



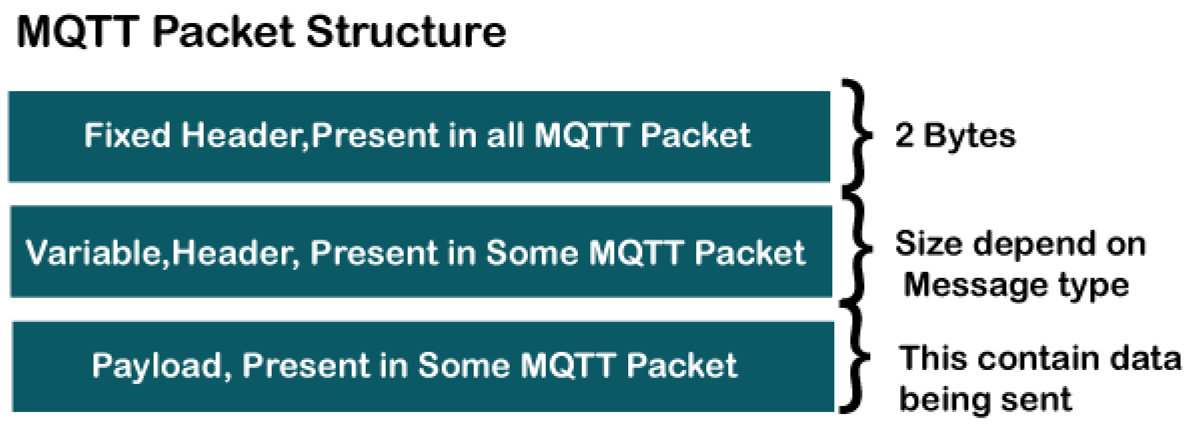
Now we will look at the architecture of MQTT. To understand it more clearly, we will look at the example. Suppose a device has a temperature sensor and wants to send the rating to the server or the broker. If the phone or desktop application wishes to receive this temperature value on the other side, then there will be two things that happened. The publisher first defines the topic; for example, the temperature then publishes the message, i.e., the temperature's value. After publishing the message, the phone or the desktop application on the other side will subscribe to the topic, i.e., temperature and then receive the published message, i.e., the value of the temperature. The server or the broker's role is to deliver the published message to the phone or the desktop application.

## MQTT Message Format



The MQTT uses the command and the command acknowledgment format, which means that each command has an associated acknowledgment. As shown in the above figure that the connect command has connect acknowledgment, subscribe command has subscribe acknowledgment, and publish command has publish acknowledgment. This mechanism is similar to the handshaking mechanism as in TCP protocol.

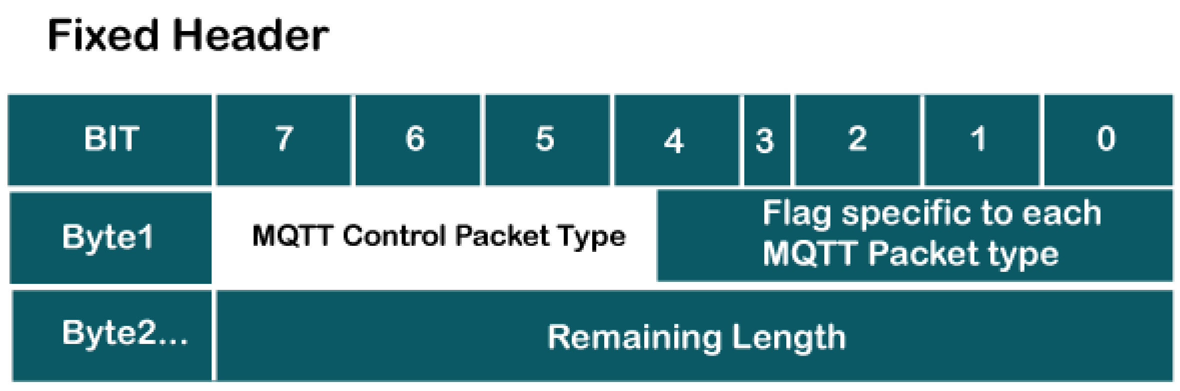
Now we will look at the packet structure or message format of the MQTT.


MQTT Packet Structure
Fixed Header,Present in all MQTT Packet
2 Bytes
Variable,Header, Present in Some MQTT Packet
Size depend on Message type
Payload, Present in Some MQTT Packet
This contain data being sent

The MQTT message format consists of 2 bytes fixed header, which is present in all the MQTT packets. The second field is a variable header, which is not always present. The third field is a payload, which is also not always present. The payload field basically contains the data which is being sent. We might think that the payload is a compulsory field, but it does not happen. Some commands do not use the payload field, for example, disconnect message.

## Fixed Header

Let's observe the format of the fixed header.


Fixed Header

| BIT | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Byte1 | MQTT Control Packet Type | | | | Flag specific to each MQTT Packet type | | | |
| Byte2... | Remaining Length | | | | | | | |

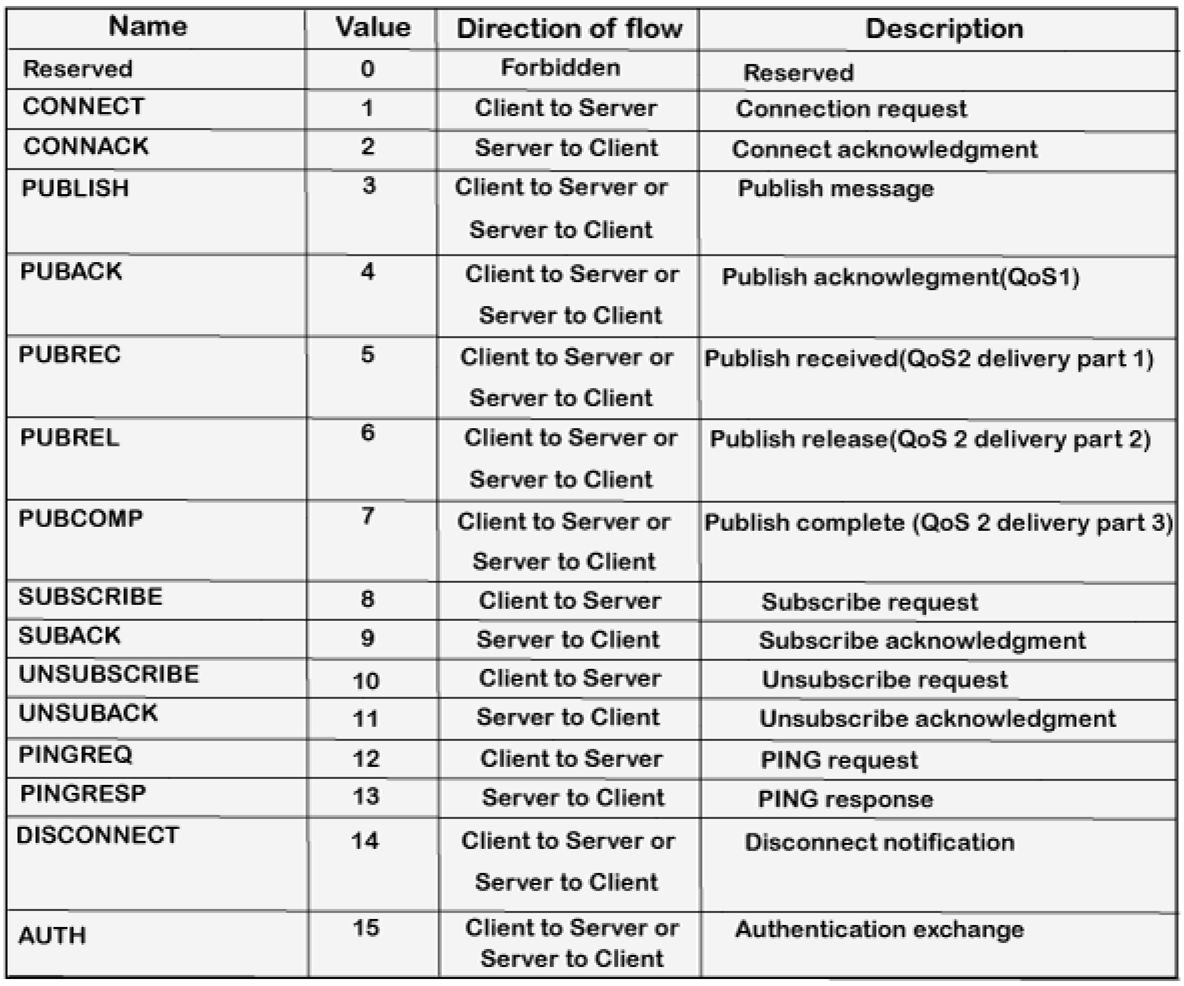
As we can observe in the above format that the fixed header contains two bytes. The first byte contains the following fields:

- **MQTT Control Packet Type:** It occupies 4 bits, i.e., 7 to 4-bit positions. This 4-bit is an assigned value, and each bit represents the MQTT control packet type.
- **Flag specific to each MQTT packet type:** The remaining 4-bits represent flag specific to each MQTT packet type.

The byte 2 contains the remaining length, which is a variable-length byte integer. It represents the number of bytes remaining in a current control packet, including data in the variable header and payload. Therefore, we can say that the remaining length is equal to the sum of the data in the variable header and the payload.

**MQTT Control Packet Types**

MQTT Control Packet Types



| Name | Value | Direction of flow | Description |
|---|---|---|---|
| Reserved | 0 | Forbidden | Reserved |
| CONNECT | 1 | Client to Server | Connection request |
| CONNACK | 2 | Server to Client | Connect acknowledgment |
| PUBLISH | 3 | Client to Server or Server to Client | Publish message |
| PUBACK | 4 | Client to Server or Server to Client | Publish acknowlegment(QoS1) |
| PUBREC | 5 | Client to Server or Server to Client | Publish received(QoS2 delivery part 1) |
| PUBREL | 6 | Client to Server or Server to Client | Publish release(QoS 2 delivery part 2) |
| PUBCOMP | 7 | Client to Server or Server to Client | Publish complete (QoS 2 delivery part 3) |
| SUBSCRIBE | 8 | Client to Server | Subscribe request |
| SUBACK | 9 | Server to Client | Subscribe acknowledgment |
| UNSUBSCRIBE | 10 | Client to Server | Unsubscribe request |
| UNSUBACK | 11 | Server to Client | Unsubscribe acknowledgment |
| PINGREQ | 12 | Client to Server | PING request |
| PINGRESP | 13 | Server to Client | PING response |
| DISCONNECT | 14 | Client to Server or Server to Client | Disconnect notification |
| AUTH | 15 | Client to Server or Server to Client | Authentication exchange |

The above table shows the control packet types with 4-bit value and direction flow. As we can observe that every command is followed by acknowledgment like CONNECT has CONNACK, PUBLISH has PUBACK, PUBREC, PUBREL, and PUBCOMP, SUBSCRIBE has SUBACK, UNSUBSCRIBE has UNSUBACK.