**MQTT Protocol**

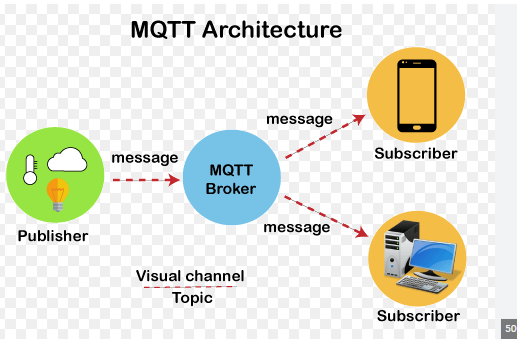
MQTT stands for Message Queuing Telemetry Transport.

MQTT is a machine to machine internet of things connectivity protocol.

It is an extremely lightweight and publish-subscribe messaging transport protocol.

**Characteristics of MQTT:**

1. It is a machine to machine protocol, i.e., it provides communication between the devices.
2. It is designed as a simple and lightweight messaging protocol that uses a publish/subscribe
3. system to exchange the information between the client and the server.
4. It provides faster data transmission, like how WhatsApp/messenger provides a faster delivery. It's a real-time messaging protocol.
5. It does not require that both the client and the server establish a connection at the same time.



**How does MQTT work?**

1. An MQTT client establishes a connection with the MQTT broker.
2. Once connected, the client can either publish messages, subscribe to specific messages, or do both.
3. When the MQTT broker receives a message, it forwards it to subscribers who are interested.

**What are MQTT components?**

1. MQTT Broker: The MQTT broker is the central hub of the MQTT network. It receives messages from publishers and distributes them to subscribers. It also maintains the state of all connected clients and manages subscriptions.
2. Publishers: Publishers are clients that send messages to the MQTT broker.They publish messages to topics, which are used to identify the content of the message.
3. Subscribers: Subscribers are clients that receive messages from the MQTT broker. They subscribe to one or more topics and receive messages that are published to those topics.
4. Topics: Topics are used to identify the content of MQTT messages. They are hierarchical in nature and can be thought of as a tree structure.

For example, a topic might be "sensors/temperature/room1" to indicate the temperature of a sensor in room 1.

1. QoS (Quality of Service) Levels: MQTT supports three different levels of QoS, which determine the level of reliability and delivery guarantees for messages. These levels include:

* QoS 0: (At most once)specifies at most once, or once and only once without requiring an acknowledgment of delivery. This is often referred to as fire and forget.
* QoS 1: (At least once delivery) specifies at least once. The message is sent multiple times until an acknowledgment is received, known otherwise as acknowledged delivery.
* QoS 2: (Exactly once delivery) specifies exactly once. The sender and receiver clients use a two level handshake to ensure only one copy of the message is received, known as assured delivery.

1. Retained Messages: MQTT supports retained messages, which are messages that are stored on the MQTT broker and delivered to new subscribers when they subscribe to a topic.

**Key Functions in MQTT**

1)void on\_message(mosquitto \*mqtt, void \*userdata, const mosquitto\_message \*msg)

{

cout << "[" << msg->topic << "] " << (char \*) msg->payload << std::endl;

}

This is a function definition for a callback function that is intended to be called when a message is received over the MQTT protocol. The function is named on\_message and takes three parameters

1.mqtt: a pointer to a mosquitto struct that represents the MQTT client that received the message

2.userdata: a void pointer that allows arbitrary user data to be passed to the callback function (it is not used in this function)

3. msg: a pointer to a mosquitto\_message struct that contains information about the received message, including its payload and topic.

2) mosquitto\_lib\_init();

The mosquitto\_lib\_init() function initializes the library and prepares it for use by an application. It should be called once at the beginning of the application before any other Mosquitto functions are used.

If the library is not properly initialized before use, it may result in undefined behavior or crashes. Therefore, it is important to ensure that the mosquitto\_lib\_init() function is called before any other Mosquitto functions are used in an application.

3) mosquitto \*mqtt = mosquitto\_new("subscriber", true, NULL);

1. id: a string identifier for the client. In this case, it is set to "subscriber".
2. clean\_session: a boolean flag that specifies whether the client wants to start with a clean session or resume a previous session.
3. userdata: a pointer to arbitrary user data that can be associated with the client instance. In this case, it is set to NULL as no user data is being associated.

4)mosquitto\_connect(mqtt, MQTT\_HOSTNAME, MQTT\_PORT, 0)

This code snippet attempts to connect the MQTT client represented by the mqtt pointer to an MQTT broker using the mosquitto\_connect() function.

The mosquitto\_connect() function takes four arguments:

1. mqtt: a pointer to a mosquitto struct that represents the MQTT client instance.
2. host: a string representing the hostname or IP address of the MQTT broker.
3. port: an integer representing the port number on which the MQTT broker is listening.
4. keepalive: an integer specifying the keep-alive interval in seconds.

If the connection is successful, mosquitto\_connect() returns MOSQ\_ERR\_SUCCESS.

If the connection attempt fails, it returns an error code.

5)void on\_publish(struct mosquitto \*mqtt, void \*userdata, int mid)

1. mqtt: a pointer to the Mosquitto client instance.
2. userdata: a pointer to user-defined data that was set using mosquitto\_userdata\_set().
3. mid: the message ID of the published message.

6)mosquitto\_destroy(mqtt);

mosquitto\_destroy(mqtt) is a function call to clean up and free resources associated with an instance of the Mosquitto library.

It should be called at the end of the program or when the client no longer needs to use the Mosquitto library.

1)created separate file for Publisher and Subscriber

In that client publish message on broker and another client can subscriber that same topic and get message. But this is single way.

We can see the following screenshots:

