CHAPTER 3

MQTT: End – to – End IoT Integration with Practical Examples

Suresh Kumar, Palanivelrajan and Yogesh Iggalore

CONTENTS

In the recent years, many applications of Internet of Things (IoT) use internet to monitor, analyses and control the physical objects. MQTT stands for Message Queuing Telemetry Transport. It is light weight publish subscribe system where the data can be published or subscribed by MQTT client. It is originally developed by IBM and it is freely available to the user. MQTT is a simple messaging protocol is designed based on low bandwidth application in mind. It is a perfect protocol for IoT devices. This protocol runs over TCP/IP or over other protocol that provides ordered, loss less, bidirectional connections. It uses publish/subscribe message pattern which provides one to many message distributions and decoupling of applications. The publish/subscribe messages are agnostic to the content of payload. Various quality of services like at most once, at least once and exactly once are used to receive payload without the loss of critical information by IoT devices. The practical implementation of MQTT in IoT application is discussed in this chapter. MQTT is used to reduce transport overhead, protocol exchanges with minimized

information, to reduce network traffic and it includes mechanism to notify interested parties when an abnormal disconnection occurs. MQTT client and MQTT broker (server) exchange information using messages called as topics. Every sensor is a client and connects to server or broker over TCP/IP. The Client publishes discrete chunk of data, opaque to the broker. Every message is published to an address, known as topic. Clients can subscribe to multiple topics. Every client subscribes to different topics and receives publish messages with respect to the corresponding topic/topics.

3.1 INTRODUCTION

Before getting into MQTT protocol and its practical implementation, let us understand what IoT is all about. IoT stands for Internet of Things, which means physical things that are connected with each other or to a group of others either through internet or intranet. For example, let us consider that a temperature sensor, a humidity sensor, a PIR sensor, a bulb and a fan are connected to home Wi-Fi router [1]. PIR sensor broadcasts the information about the presence of human the in room to all other devices. Temperature and humidity sensors broadcast temperature and humidity values to all other devices. Based on PIR sensor signal, temperature and humidity value: blub and fan present in room will be decided to be switched ON or OFF. This may look like automation, but actually, it is not. The example which was discussed above is about how well we use the resources available to us by conserving energy. It is not mere automation.

3.1.1 Definition of IoT

The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction [2].Figure 1 shows the goal of IoT and Figure 2 shows its benefit to humanity.

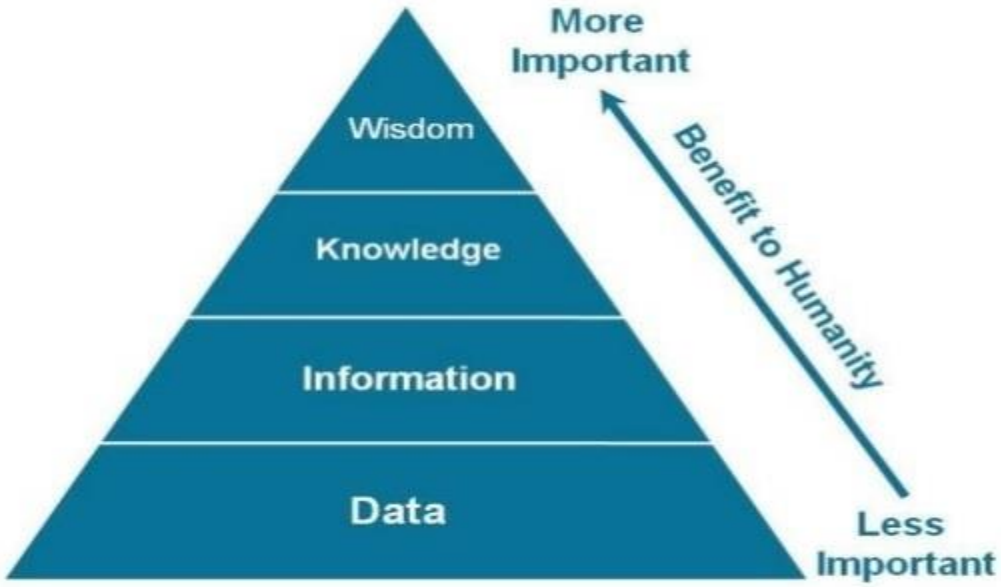


FIGURE 1 Goal of IoT

We can achieve benefit to humanity by the following

- Collecting data from all physical devices.
- Converting this raw data into information
- Using that information, we need to gain knowledge
- With that knowledge we need to attain wisdom

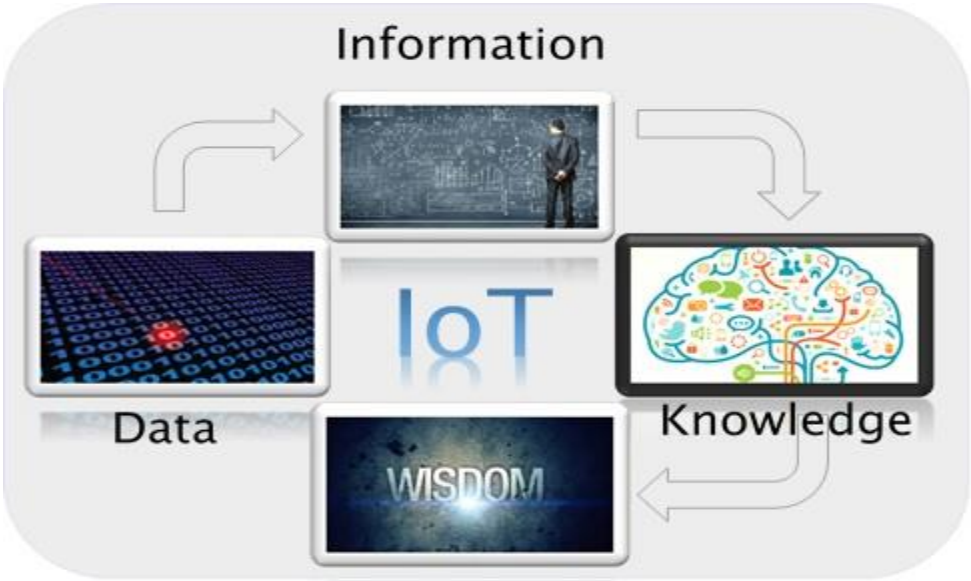


FIGURE 2 Benefit of IoT Humanity

By end of 2023, 1.3 billion IoT devices will be connected to internet. To handle these huge number of devices you need a solid, simple IoT protocol. To address these issues so many IoT protocols are created namely MQTT, CoAP, DDS, AMQP, Sigfox and so on. In this chapter we shall look into MQTT protocol only.

3.2 MQTT

MQTT stands for Message Queuing Telemetry Transport protocol, in the year 1999, IBM employee Andy and Eurotech employee Nipper developed first version of MQTT for monitoring and controlling oil pipeline in desert. As HTTP protocol is heavy for small hardware devices and also the devices are connected via satellite link, the requirement of this monitoring hardware is that, it should use low bandwidth, low power and it should be lightweight. Hence MQTT. In year 2013 IBM made MQTT open source.

3.2.1 Definition of MQTT

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed for easy implementation. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

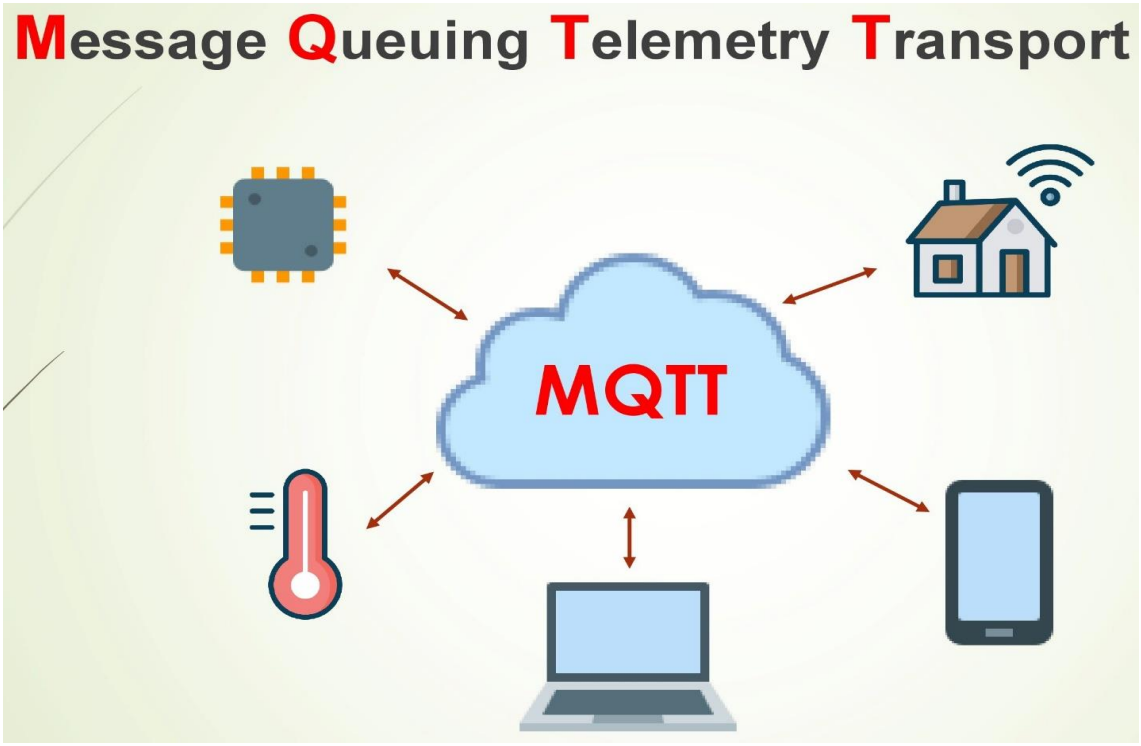


FIGURE 3 Interface of MQTT

As it is mentioned in definition, MQTT is a Client Server publish/subscribe messaging transport protocol which means there will be a centralized server for all MQTT clients which handle all connections and communications. There can be multiple MQTT clients connected to a single server. As shown in above Figure 3, a temperature sensor, home devices, IoT devices, phone and laptop are connected to centralized MQTT server.

3.2.2 Pub/Sub architecture

MQTT works on Pub Sub architecture which means Publish and subscribe architecture. This is different from traditional client server architecture as shown in Figure 4. In client server architecture client directly talks to end server where as in MQTT publisher and subscriber never contact each other directly [3].

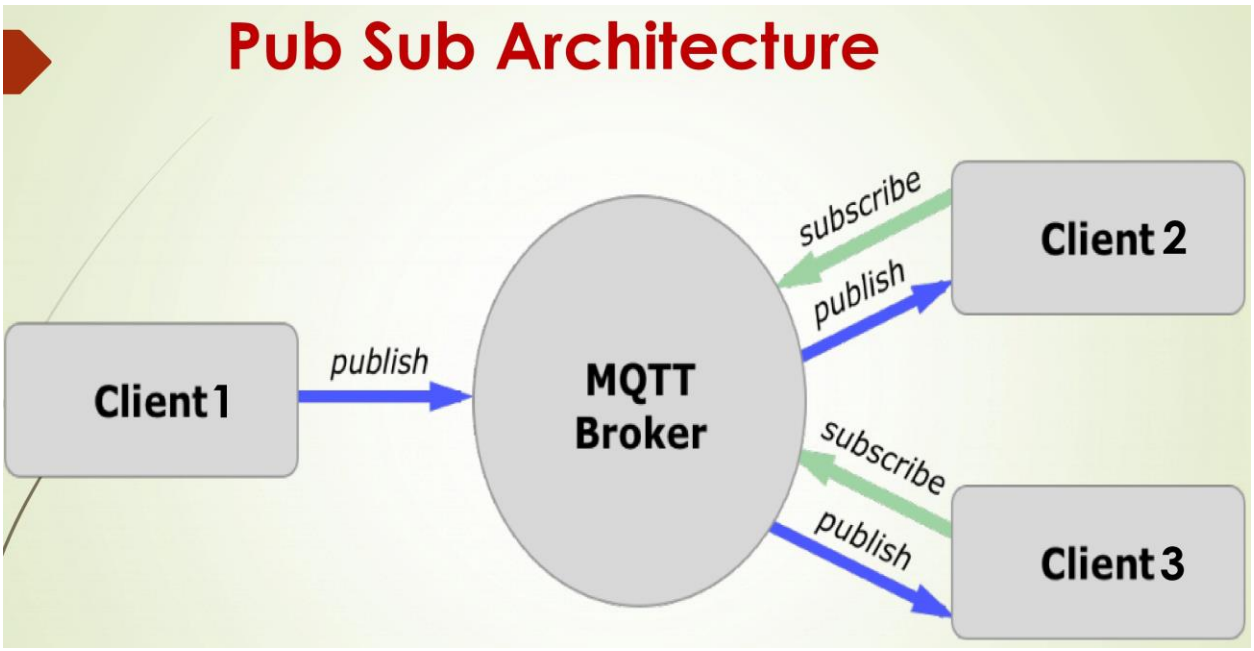


FIGURE 4 Pub / Sub Architecture

3.2.3 Benefits of pub/sub model

1. Client devices need not remember all other clients they need to communicate to.
2. It helps resource constraint devices to communicate with each other.
3. No restriction in number client connections.
4. All communications are central, hence less overhead for clients.
5. All communications work under path called “topics”.
6. No exchange of IP address of publisher and subscriber is required.
7. Time decoupling can be avoided.
8. Synchronization not required since publish and subscribe are independent actions.

3.2.4 Important terminology used in pub/sub architecture

1. **Client**: these are called end devices in MQTT or simply MQTT client devices. Both publisher and subscribers are MQTT clients. These devices communicate with centralized server called broker. Clients can communicate with other clients indirectly by subscribing

to a topic or by publishing a topic to broker.

2. Broker: It is the brain of MQTT protocol where all the connections and communications happen. It is responsible for

    a. Establishing connection with client devices.
    b. Registering publish and subscribe topics.
    c. Responsible for receiving all the client messages
    d. Filtering of received messages
    e. Receiving published messages from client
    f. Sending publish messages to subscribed client

For better understanding let us take Figure 5 as an example. Here we have 3 MQTT clients and centralized MQTT broker. Here MQTT broker handles all communications.

- Temperature sensor sends its temperature value to broker using publish method.
- Phone requests the broker for temperature value using subscribe method and sends fan on/off control signal to broker using publish method.
- Fan requests for on/off control signals from broker using subscribe method.
- Whenever broker receives temperature message from sensor it sends the same to phone.
- Whenever broker receives on/off signal from phone it sends the same to fan.
- Here all clients don’t know each other but work together with publish subscribe model.

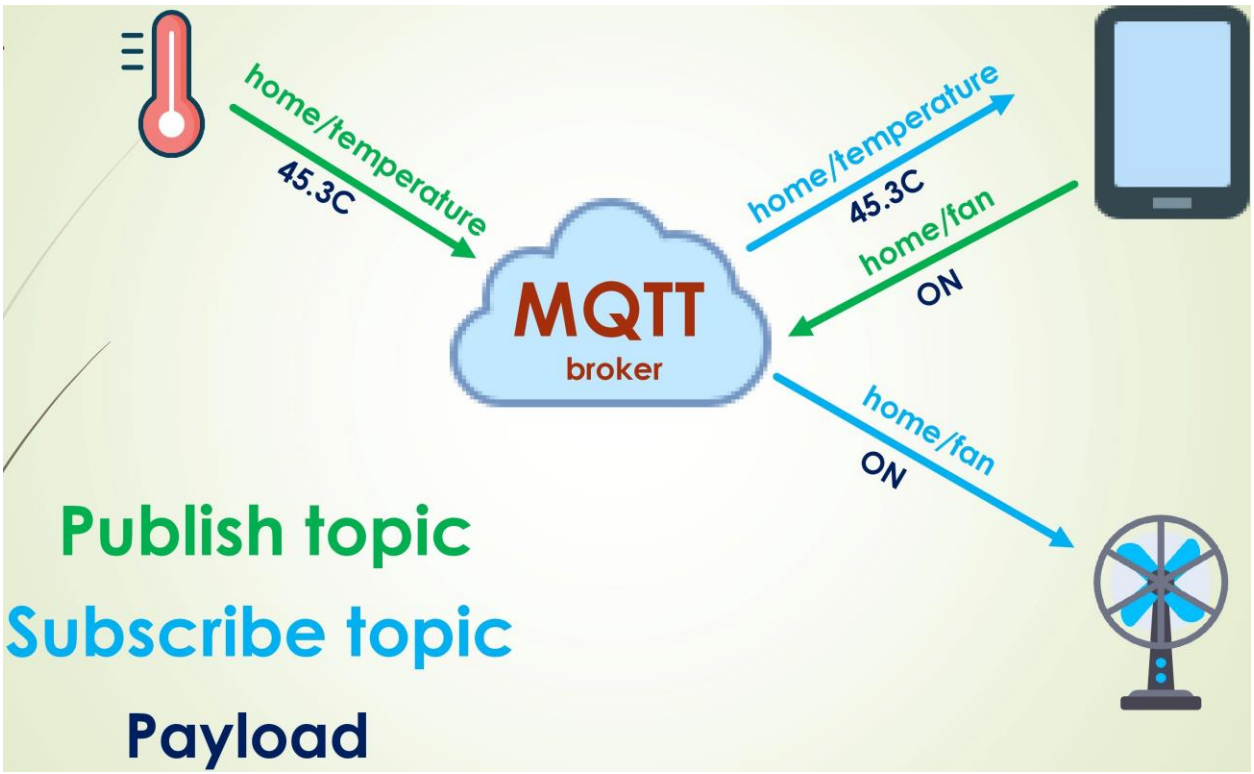


FIGURE 5 Example of MQTT Communication

Any number of these devices can be connected to broker depending on broker handling capacity as shown in figure 6.

FIGURE 6 Number of devices connected to broker

3.2.5 MQTT client broker communication

MQTT protocol works on top of TCP/IP protocol. As shows in Figure 7 client always sends connect packet to server first. Here broker will never initiate connection. The connection is always initiated by the client. Broker acknowledges the connection request. Once the connection is established, client can publish as many messages as it wants and every publish massage will be acknowledged by the broker. If client is interested in any messages, then it can subscribe to that message and broker sends back subscribe acknowledgement. Whenever broker receives a related subscribe message from any another client, it will forward the same to this client. Client can disconnect from broker, whenever it wants and it will receive disconnect acknowledgement.(This feature is only available in MQTT 5 but not MQTT 3)

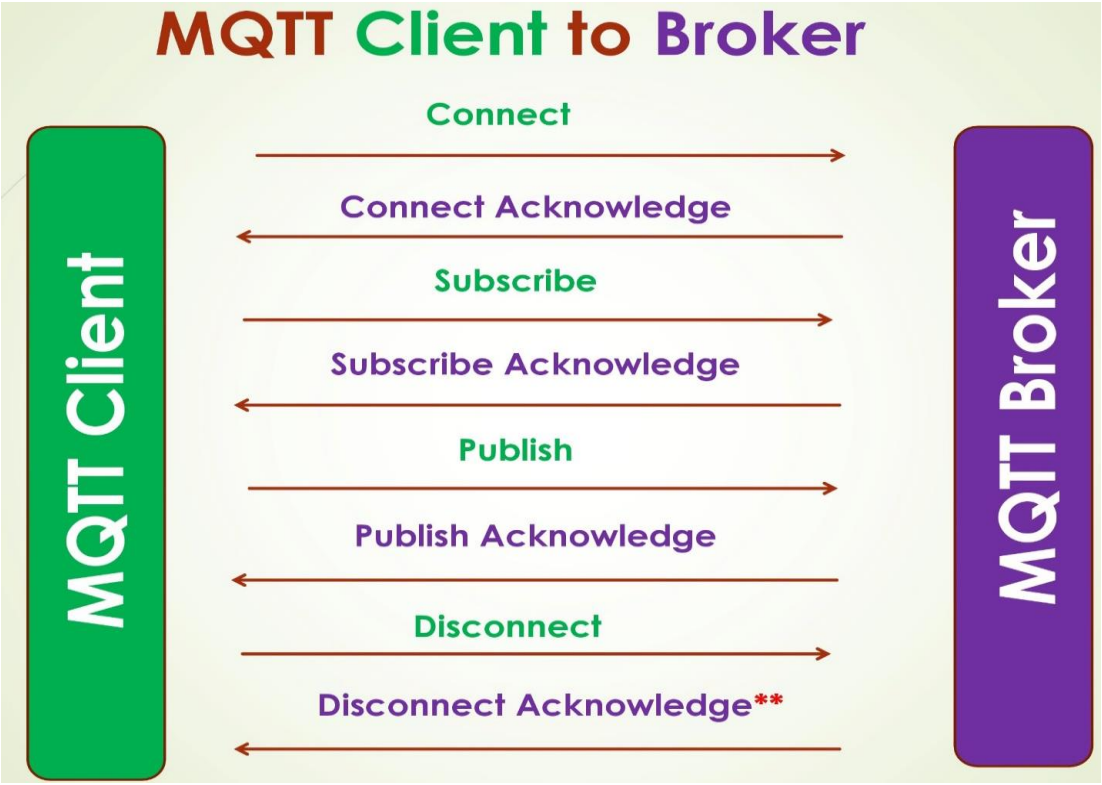


FIGURE 7 MQTT Client to Broker connection

3.2.6 MQTT standard header packet

MQTT works with exchanging series of packets, Figure 8 shows the standard packet structure [5].

Each packet contains 3 sections:

1. Fixed Header (Which is always present in all MQTT packets)
2. Variable Header
3. Payload

Variable Header and Payload are not always present in all the packets.

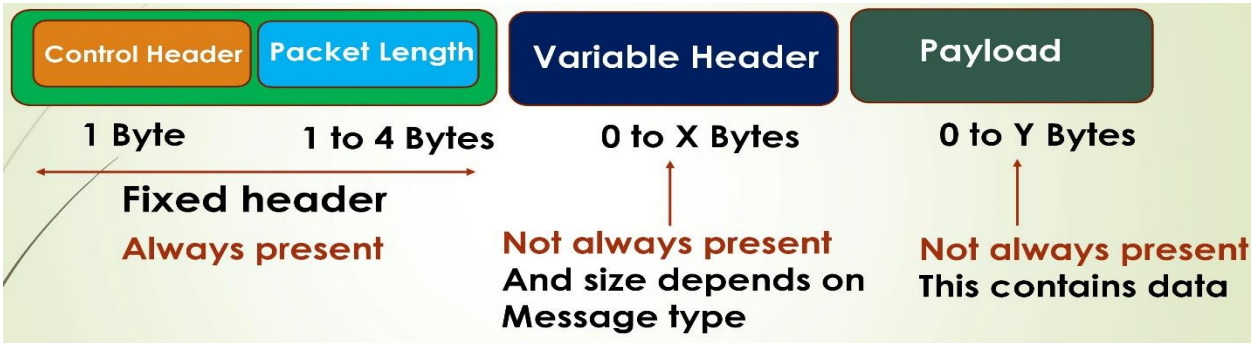


FIGURE 8 MQTT Standard Packet Structure

3.2.6.1 Fixed header

Each packet contains fixed header as shown in Figure 9. First byte in fixed header is control packet. In this, last four bits are allocated for control packet type and first four bits are allocated for control packet flags. Depending on the variable header and payload, packet length varies from one to four bytes

Control Packet	Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
	Control packet type				Control packet flags			

Name	value	direction	description	Flags			
CONNECT	1	Client to server	Client request server to connect	0	0	0	0
CONNACK	2	Server to client	Connect acknowledge	0	0	0	0
PUBLISH	3	Client to server	Publish message	DUP	QoS	QoS	RETN
PUBACK	4	Server to client	Publish acknowledge	0	0	0	0
PUBREC	5	C to S or S to C	Publish received	0	0	0	0
PUBREL	6	C to S or S to C	Publish release	0	0	1	0
PUBCOMP	7	C to S or S to C	Publish complete	0	0	0	0
SUBSCRIBE	8	Client to server	Client subscribe request	0	0	1	0
SUBACK	9	Server to client	Subscribe acknowledge	0	0	0	0
UNSUBSCRIBE	10	Client to server	Client unsubscribe request	0	0	1	0
UNSUBACK	11	Server to client	Unsubscribe acknowledge	0	0	0	0
PINGREQ	12	Client to server	Ping request	0	0	0	0
PINGRESP	13	Server to client	Ping response	0	0	0	0
DISCONNECT	14	Client to server	Client disconnecting	0	0	0	0

FIGURE 9 Control Packet

3.2.6.2 Variable header and payload

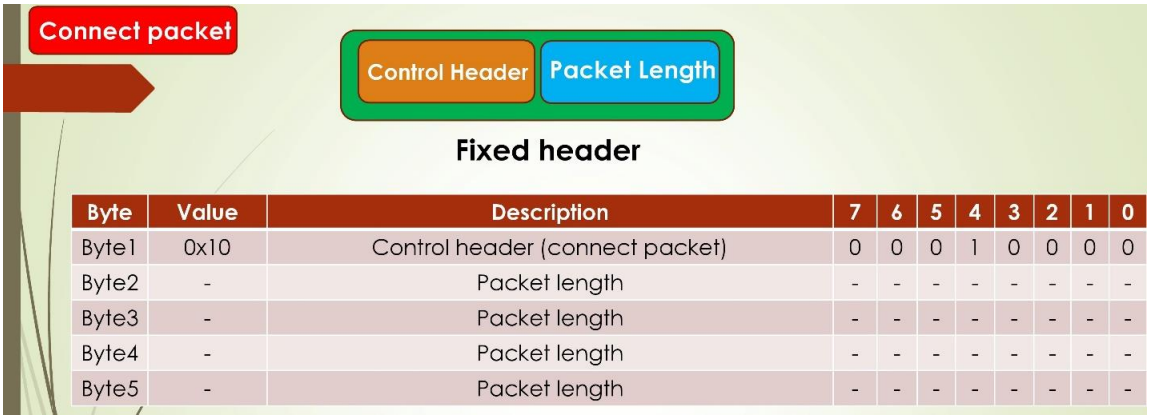
Since these two packets are dynamic, we will discuss them later.

3.2.7 Connect packet

Connect packet contains fixed header, variable header and payload, Once the TCP/IP connection is established, client will send connect packet as its first packet to broker.

3.2.7.1 Fixed header

In the fixed header, first byte is a connect packet with a value of 0x10. Here no control packets. values of bytes 2, 3, 4 and 5 depend on variable header and payload as shown in Figure 10.



Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x10	Control header (connect packet)	0	0	0	1	0	0	0	0
Byte2	-	Packet length	-	-	-	-	-	-	-	-
Byte3	-	Packet length	-	-	-	-	-	-	-	-
Byte4	-	Packet length	-	-	-	-	-	-	-	-
Byte5	-	Packet length	-	-	-	-	-	-	-	-

FIGURE 10 Connect Packet Fixed Header

3.2.7.2 Variable header

Variable header for connect packet contains four fields in the following order Protocol name, protocol level, connect flags and keep alive as shown in Figure 11.

Protocol name: first 2bytes contain protocol name length with a value of 0x04 next 4 bytes are protocol name as “MQTT”.

Protocol level: Byte7 is protocol level, value is 0x04 which means MQTT version 3.1.1

Connect flags: connect flag byte contains a number of parameters specifying the behaviour of MQTT connection. They also indicate the presence and absence of their specific fields in payload.

Connect flags are given as below:

Username: if username flag is set to 0, it must not be present in payload.

Password: if password flag is set to 0, it must not be present in payload.

Will retain, Will QoS and Will flag: these flags decide the will message to publish to another client when this client disconnects ungracefully. If **will flag** is set to 1 then **will message** should be present in the payload. For example, consider temperature sensor sets it **will message** as "temperature sensor in room is offline" then if temperature sensor is not sending its values on regular intervals, the broker will transmit **will message** of temperature to all its subscribers (clients).

Situations in which will message gets published include for example, and many more

1. An I/O error or network failure detected by the MQTT broker.

2.The Client fails to communicate within the Keep Alive time.

3.The Client closes the Network Connection without first sending a DISCONNECT Packet.

4.The MQTT broker closes the Network Connection because of a protocol error.

If the **Will Flag** is set to 1, the **Will QoS** and **Will Retain** fields in the Connect Flags will be used by the MQTT broker, and the **Will Topic** and **Will Message** fields MUST be present in the payload.

The Will Message must be removed from the stored Session state in the MQTT broker once it has been published or the MQTT broker has received a disconnect packet from the Client.

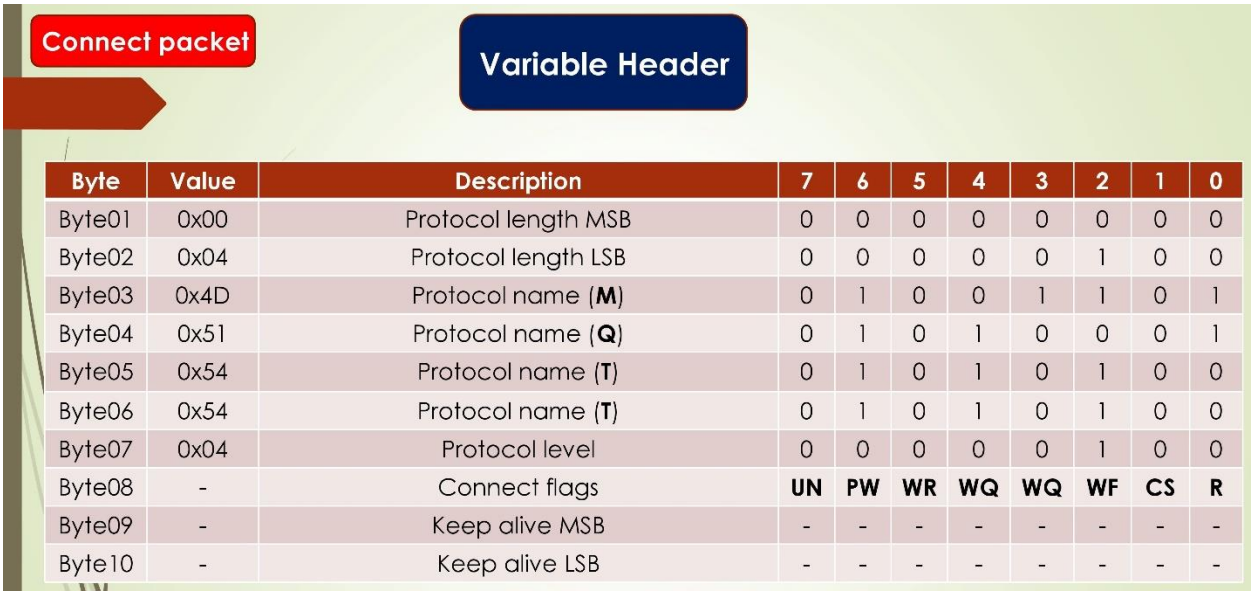
If the **Will Flag** is set to 0, the **Will QoS** and **Will Retain** fields in the Connect Flags MUST be set to zero and the **Will Topic** and **Will Message** fields must not be present in the payload.

If the Will Flag is set to 0, a Will Message must not be published when this Network Connection ends

Keep alive: 2 bytes of keep alive value in seconds define the longest period of time that the broker and client can endure without sending a message. Within this time client must send a ping packet to say that it is alive. Max time can this value can be set is 18 hours 20 minutes.

Clean session: if clean session flag is set to 0, broker must resume the previous session connection otherwise, it must discard the previous session connection.

Session state in client consists of **Quality of Service (QoS)** and existence of session. which will be discussed later in this chapter.



Byte	Value	Description	7	6	5	4	3	2	1	0
Byte01	0x00	Protocol length MSB	0	0	0	0	0	0	0	0
Byte02	0x04	Protocol length LSB	0	0	0	0	0	1	0	0
Byte03	0x4D	Protocol name (**M**)	0	1	0	0	1	1	0	1
Byte04	0x51	Protocol name (**Q**)	0	1	0	1	0	0	0	1
Byte05	0x54	Protocol name (**T**)	0	1	0	1	0	1	0	0
Byte06	0x54	Protocol name (**T**)	0	1	0	1	0	1	0	0
Byte07	0x04	Protocol level	0	0	0	0	0	1	0	0
Byte08	-	Connect flags	**UN**	**PW**	**WR**	**WQ**	**WQ**	**WF**	**CS**	**R**
Byte09	-	Keep alive MSB	-	-	-	-	-	-	-	-
Byte10	-	Keep alive LSB	-	-	-	-	-	-	-	-

FIGURE 11 Connect Packet Variable Header

3.2.7.3 Payload

Connect packet payload contains the following fields Client id, Last will topic, Last will message, username and password these are based on the flags set in connect flags. Client id is an identification name given to IoT device, which can be same or unique. Usually unique ID is preferred. If client sets will topic flag, the payload must include its Last will topic and Last will message. Username and password field must be included based on their respective flags

As shown in figure 12

1. first 2 bytes will be client id length and followed by client id.
2. Followed by 2 bytes of will topic length and will topic.
3. Followed by 2 bytes of will message length and will message.
4. Followed by 2 bytes of username length and username.
5. Followed by 2 bytes of password length and password.

Connect packet

Payload

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte A1	-	Client id length MSB	-	-	-	-	-	-	-	-
Byte A2	-	Client id length LSB	-	-	-	-	-	-	-	-
Byte A3-An	-	Client id	-	-	-	-	-	-	-	-
Byte B1	-	Will topic length MSB	-	-	-	-	-	-	-	-
Byte B2	-	Will topic length LSB	-	-	-	-	-	-	-	-
Byte B3-Bn	-	Will topic	-	-	-	-	-	-	-	-
Byte C1	-	Will message length MSB	-	-	-	-	-	-	-	-
Byte C2	-	Will message length LSB	-	-	-	-	-	-	-	-
Byte C3-Cn	-	Will message	-	-	-	-	-	-	-	-
Byte D1	-	Username length MSB	-	-	-	-	-	-	-	-
Byte D2	-	Username length LSB	-	-	-	-	-	-	-	-
Byte D3-Dn	-	Username	-	-	-	-	-	-	-	-
Byte E1	-	Password length MSB	-	-	-	-	-	-	-	-
Byte E2	-	Password length LSB	-	-	-	-	-	-	-	-
Byte E3-En	-	Password	-	-	-	-	-	-	-	-

FIGURE 12 Connect Packet Payload

Once the broker receives connect packet from client it will respond back with **CONNACK** packet.

3.2.8 CONNACK Packet

The CONNACK packet is sent by broker in response to a connect packet received from client. It is connection acknowledgement packet from broker to client. Fate of further communication depends on this. This packet includes only fixed header and variable header as shown in Figure 13. Payload is not included here.

3.2.8.1 Fixed header

First byte of fixed header is connection acknowledge packet with a value of 0x20. Second byte is

length of remaining bytes with a value of 0x02 which means 2 bytes will be followed by this.

3.2.8.2 Variable header

Variable header includes 2 bytes of connection flags and connection return code.

If the broker accepts the connection with clean session set to 1 then broker must send the response with the flag set to 0. If broker accepts connection with clean session set to 0 then broker must send the response with flag set based on the previous session connection.

The value of connection return code indicates the reason for connection failure or success. Following are only possible return codes from broker.

1. Value 0x00 connection accepted.
2. Value 0x01 connection refused, unacceptable protocol version.
3. Value 0x02 connection refused, identifier rejected.
4. Value 0x03 connection refused, server error
5. Value 0x04 connection refused, bad username or password
6. Value 0x05 connection refused, unauthorised

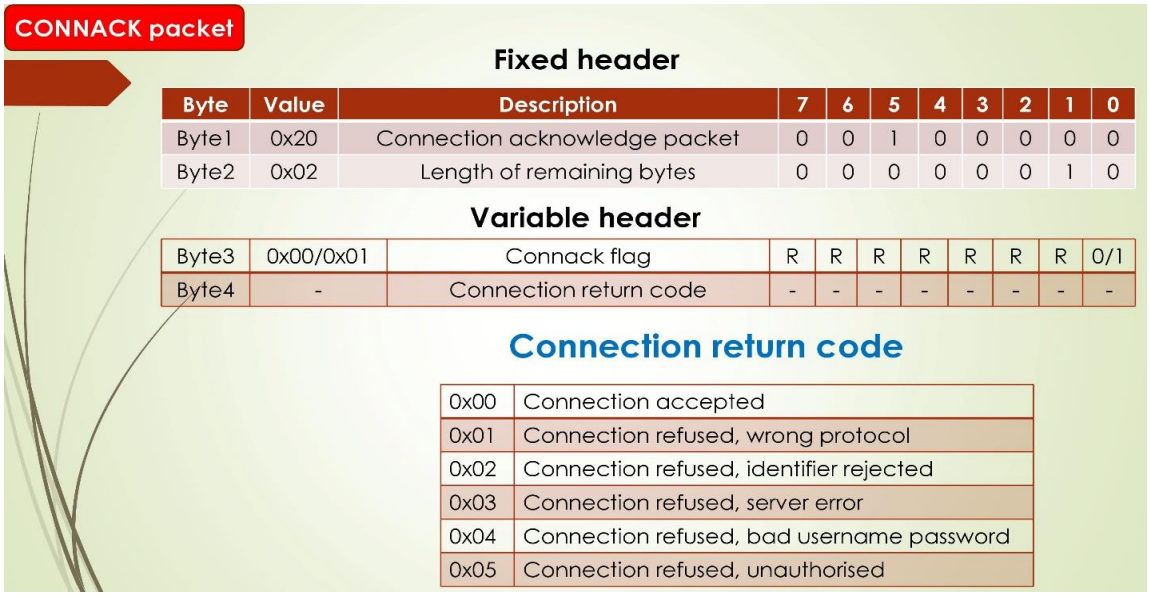


FIGURE 13 CONNACK Packet Fixed Header & Variable Header

Let us understand the above mentioned (CONNECT and CONNACK) packets with the following example.

Connect packet from client to broker

Connect Packet example:

As shown in figure 14 fixed header of connect packet value is 0x10 and it has 0x50 bytes as remaining bytes. Variable header includes protocol length, name, level and keep alive value. Connect flags with **QoS** 1 and **will flag** and **retain flag** set to 1. Keep Alive: 0x00 0x3C (60 Seconds) server client timeout here is 60 seconds if no communication between client and broker, broker will disconnect with client after 90 seconds which is one and half of keep alive time. max time from client side is 18:12:15 hours and for sever it is 27:24:30 hours (one and half of 18:12:15).

Fixed header

Byte	Description	Value
01	connect packet	0x10
02	Packet length	0x50

Variable header

03,04	Protocol length	0x00 0x04
05 – 08	Protocol name	0x4D 0x51 0x54 0x54 (M Q T T)
09	Protocol level	0x04 version 3.1.1
10	Connect flags	0xEE b'11101110 UN,PW,WR,WQ,WQ,WF,CS,R
11-12	Keep alive	0x00 0x3C {60 seconds }

FIGURE 14 Example of Connect Packet

Payload:

For connect packet, Payload order is client id, will topic, will message, username and password as shown in Figure 15. In this example client id length is 17 bytes and client id is ***CC:50:E3:9B:F7:84*** (mac address of device). will topic length is 24 bytes and will topic is

(***CC:50:E3:9B:F7:84/status)***. Will message length is 7 bytes and message is ***offline***. Username length is 6 bytes and username is ***yogesh***. Password length is 6 bytes and password is ***yogesh***.

In order to respond back MQTT broker sends CONNACK packet. let us now look into it.

Payload

Byte	Description	Value
13,14	Client id length	0x00 0x11 (17 bytes)
15 – 32	Client id	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34{CC:50:E3:9B:F7:84} mac address
33, 34	Will topic length	0x00 0x18 (24 bytes)
35 – 59	Will topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x73 0x74 0x61 0x74 0x75 0x73 {CC:50:E3:9B:F7:84/status}
60, 61	Will message length	0x00 0x07
62 - 69	Will message	0x6F 0x66 0x66 0x6C 0x69 0x6E 0x65 {offline}
70,71	Username length	0x00 0x06
72 - 78	Username	0x79 0x6F 0x67 0x65 0x73 0x68 {yogesh}
79,80	Password length	0x00 0x06
81 - 87	Password	0x79 0x6F 0x67 0x65 0x73 0x68 {yogesh}

FIGURE 15 Connect Packet Payload

CONNACK packet is simple and it has only 4 bytes. Byte1 is connect packet identifier with a value 0x20. Byte2 is remaining length of packet. Here it is 0x02 as shown in figure 16. Byte3 is connect flag with value 0. It indicates that the connection success and it is clean session. Byte4 is return code value 0x00 indicates connection accepted.

Fixed header

Byte	Description	Value
01	Connack packet	0x20
02	Packet length	0x02

Variable header

03	Connack flag	0x00 { 0 since it is clean session }
04	Connection return code	0x00 {connections Accepted}

FIGURE 16 CONNACK Packet Example

3.2.9 Publish Packet

MQTT client can publish messages as soon as it is connected to broker. MQTT works on topic-based filtering of the incoming messages. Each publish message must include a topic so that the broker can filter it out and forward it to interested client. Each message should have payload which contains the data to transmit in byte format. MQTT is data-agnostic, the client can send binary data, text data, or even full-fledged XML or JSON. Publish message has several attributes. Let us discuss them in detail.

3.2.9.1 Topic Name

The word **topic** refers to a UTF-8 string that the broker uses to filter messages for each connected client [4]. It includes one or more topic level. Each topic level is separated by forward slash as shown in figure 17.

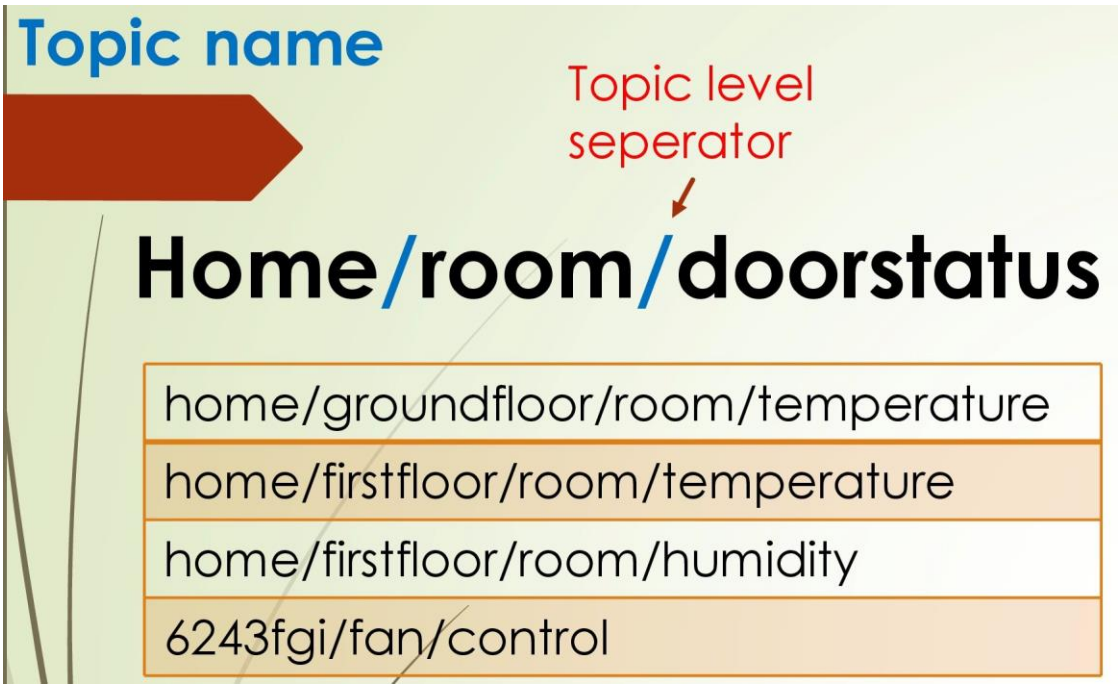


FIGURE 17 Topic Name examples

Topics are case sensitive which means “Home/room” is different from “home/room”. Space should not be used in topic. Topic uses wildcard (using + and #) as well; single level and multi-

level as shown in figure 18 & 19.

FIGURE 18 Topic wildcard usage

FIGURE 19 Topic wildcard usage

Topics beginning with **$** symbol have different purpose. These are not part of the subscription, when you subscribe to multi-level wildcards. $ symbol reserved for internal statistics purpose is shown in figure 20.

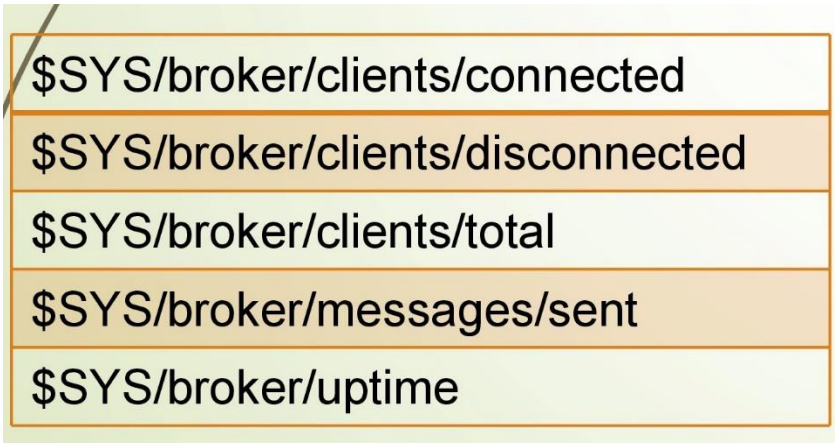


FIGURE 20 Different Purposes of $ Symbol

Best practices to use topics are

1. Never use a leading forward slash since it introduces unnecessary topic levels with zero character.
2. Never use space in topic.
3. Keep topic names short and concise.
4. Use only ascii characters, avoid non printable characters.
5. Embed a unique identifier or the client id into the topic.
6. Don't subscribe to multi-level wild cards (#) as we do not now how many topics are present
7. Don't forget extensibility while choosing topic name.
8. Use specific topic not a general one.

3.2.9.2 Quality of Service (QoS)

Quality of service is a key feature of MQTT. QoS gives client the power to choose a level of service that matches its network reliability and application logic. It is basically an agreement between sender of the message and receiver of the message that defines the guarantee of delivery for the specific message. There are 3 QoS levels in MQTT

1. At most once (0)
2. At least once (1)
3. Exactly once (3)

while we talk about QoS in MQTT, we need to consider two things One: message received by

broker from publishing client and Two: message received by subscribing client from broker. We will look QoS separately, broker transmits the message to subscribing client with QoS either equal to or less than that of publishing client.

3.2.9.2.1 QoS 0 – at most once

This is the minimal quality of service and fastest in MQTT. This service level guarantees a best-effort delivery but there is no guaranty in delivery. No acknowledgment will be sent by broker or subscribing client as shown in figure 21. By default, all publishing messages will have QoS 0 unless mentioned otherwise. QoS is fire and forget and guaranties the delivery of messages depending on TCP protocol.

FIGURE 21 QoS 0 send to broker

3.2.9.2.2 QoS 1 – at least once

This is second level of quality of service provided by MQTT. QoS 1 provides guarantee that message gets delivered at least once to receiver. There can also be a possibility of receiving the same message multiple times. The sender stores the message until receiver acknowledges. The sender uses the packet identifier in each packet to match the publish packet with the corresponding PUBACK packet as shown in figure 22. If the sender does not receive a PUBACK packet in a reasonable amount of time, the sender resends the PUBLISH packet. When a receiver gets a message with QoS 1, it can process it immediately. For example, if the receiver is a broker, the broker sends the message to all subscribing clients and then replies with a PUBACK packet.

If the publishing client sends the message again, it sets a duplicate (DUP) flag. In QoS 1, this DUP flag is only used for internal purposes and is not processed by broker or client. The receiver

of the message sends a PUBACK, regardless of the DUP flag.

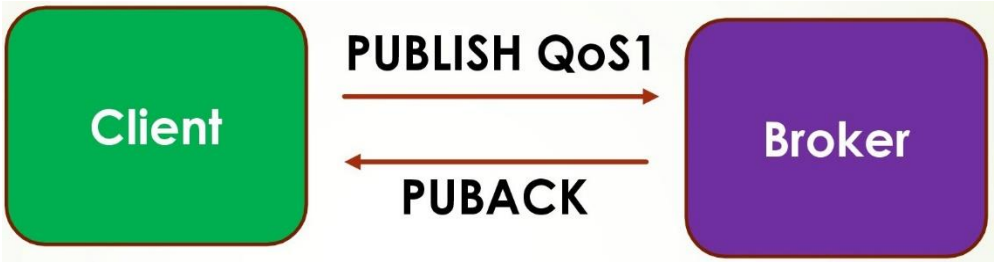


FIGURE 22 QoS 1 send to broker and received PUBACK

3.2.9.2.3 QoS 2 – exactly once

QoS 2 is highest level of service from MQTT. This level guarantees 100% that each message is received only once by intended recipients. QoS 2 is the safest and slowest quality of service level. The guarantee is provided by at least two request/response flows, A four-part handshake as shown in figure 23. The sender and receiver use the packet identifier of the original publish message to coordinate delivery of the message.

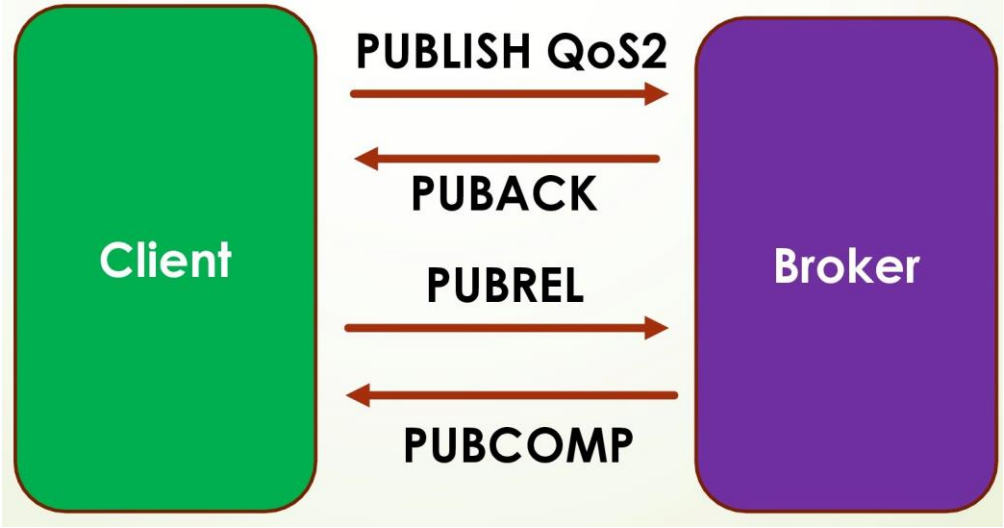


FIGURE 23 QoS 2 send to broker

Use QoS 0 when clients have stable internet connection and messages are not critical.

Use QoS 1 when client need to send all the messages to broker, fast and stable service.

Use QoS 2 when messages are critical and client don’t want his receiver to miss any messages or

duplications.

3.2.9.3 Retain Flag

This flag decides whether message is saved in broker as the **known good value** for specific topic. When new client subscribes to a topic, this saved message will be transmitting by broker to newly subscribed client.

3.2.9.4 DUP flag

This flag indicates the message which is already sent and it is duplicate.

Now let us look into PUBLISH packet, publish packet includes fixed header, variable header and payload.

3.2.9.5 Payload

This is the 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.

3.2.9.6 Packet Identifier

The 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.

3.2.9.7 Fixed header

Figure 24 shows fixed header, in that first byte has PUBLISH packet code in higher nibble and publish flags in lower nibble. While publishing message to broker, client must set its Quality of service, duplicate flag and retain flag as required by the client. Byte 2 to byte 4 will decide the length of packet.

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x3X	Control header (publish packet)	0	0	1	1	D	Q	Q	R
Byte2	-	Packet length	-	-	-	-	-	-	-	-
Byte3	-	Packet length	-	-	-	-	-	-	-	-
Byte4	-	Packet length	-	-	-	-	-	-	-	-
Byte5	-	Packet length	-	-	-	-	-	-	-	-

QoS	Bit2	Bit1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved

D: Duplicate flag **R: Retain flag**

FIGURE 24 Fixed Header

3.2.9.8 Variable header and payload

As shown in figure 25, first 2 bytes gives the length of topic followed by topic and at packet identifier at the end (this is present only if QoS in fixed header set QoS 1 or QoS 2). Payload contains the message to be sent to broker by the client or vice versa.

Variable header

Byte	value	Description	7	6	5	4	3	2	1	0
ByteA1	-	Topic length MSB	-	-	-	-	-	-	-	-
ByteA2	-	Topic length LSB	-	-	-	-	-	-	-	-
ByteA3-An	-	Topic	-	-	-	-	-	-	-	-
ByteB1	-	Packet identifier MSB	x	x	x	x	x	x	x	x
ByteB2	-	Packet identifier LSB	x	x	x	x	x	x	x	x

Payload

ByteC1-Cn	-	Topic payload	-	-	-	-	-	-	-	-

QoS	Packet identifier
0	Absent
1	Present
2	Present

FIGURE 25 Variable Header

Example 1:

topic name "CC:50:E3:9B:F7:84/hall" payload "test"

Publish topic with QoS0

In this example payload message ***test*** is publishing with QoS 0. First byte is publishing command

with QoS set to 0 and no duplicate flag. Retain flag set to 1 is shown in figure 26a.

topic name "CC:50:E3:9B:F7:84/hall" payload "test" QoS0

Fixed header

Byte	Description	Value
01	Publish packet	0x31 b'00110001 {QoS = 0,dup = 0, retain=1}
02	Packet length	0x1C (28 bytes)

Variable header

03,04	Topic length	0x00 0x16 (22 bytes)
05 - 26	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x68 0x61 0x6C 0x6C {CC:50:E3:9B:F7:84/hall}

Payload

27 - 31	payload	0x74 0x65 0x73 0x74 {test}

FIGURE 26 a. Publish topic with QoS0

Publish topic with QoS1

In this example payload message ***test*** is publishing with QoS 1. First byte is publishing command with QoS set to 1 and no duplicate flag. Retain flag set to 1 is shown in figure 26b.

topic name "CC:50:E3:9B:F7:84/hall" payload "test" QoS1

Fixed header

Byte	Description	Value
01	Publish packet	0x33 b'00110011 {QoS = 1,dup = 0, retain=1}
02	Packet length	0x1E (30 bytes)

Variable header

03,04	Topic length	0x00 0x16 (22 bytes)
05 - 26	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x68 0x61 0x6C 0x6C {CC:50:E3:9B:F7:84/hall}
27,28	Packet identifier	0x00 0x02

Payload

29 - 32	payload	0x74 0x65 0x73 0x74 {test}

FIGURE 26 b. Publish topic with QoS1

Publish topic with Qos2

In this example payload message ***test*** is publishing with QoS2. First byte is publishing command with QoS set to 2 and no duplicate flag. Retain flag set to 1 is shown in figure 26c.

topic name "CC:50:E3:9B:F7:84/hall" payload "test" QoS2

Fixed header

Byte	Description	Value
01	Publish packet	0x35 b'00110101 {QoS = 2,dup = 0, retain=1}
02	Packet length	0x1E (30 bytes)

Variable header

03,04	Topic length	0x00 0x16 (22 bytes)
05 - 26	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x68 0x61 0x6C 0x6C {CC:50:E3:9B:F7:84/hall}
27,28	Packet identifier	0x00 0x02

Payload

29 - 32	payload	0x74 0x65 0x73 0x74 {test}

FIGURE 26 c. Publish topic with QoS2

3.2.10 PUBACK packet

This packet is response to the publish packet sent with QoS 1. It includes fixed header and variable header but not payload is shown in figure 27.

PUBACK packet

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x40	Publish acknowledge packet	0	1	0	0	0	0	0	0
Byte2	0x02	Length of remaining bytes	0	0	0	0	0	0	1	0

Variable header

Byte3	-	Packet identifier MSB	-	-	-	-	-	-	-	-
Byte4	-	Packet identifier LSB	-	-	-	-	-	-	-	-

FIGURE 27 PUBACK Packet

First byte is a publish acknowledge packet and value is 0x40, second byte is length of remaining bytes it is always 2 and third and fourth bytes are packet identifier used for QoS 1.

Example 2:

PUBACK contains 4 bytes as shown in the figure 28. First byte is **PUBACK** packet code, value

0x40 followed by length of remaining bytes that is 0x02 and at last 2 bytes of packet identifier. **PUBACK** packet has no payload.

Fixed header

Byte	Description	Value
01	PUBACK packet	0x40
02	Packet length	0x02

Variable header

03,04	Packet identifier	0x00 0x02

FIGURE 28 Example of PUBACK

3.2.11 PUBREC Packet

Publish received is the first response to the publish packet of QoS 2. It includes fixed header and variable header but not payload as shown in figure 29.

PUBREC packet

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x50	Publish received packet	0	1	0	1	0	0	0	0
Byte2	0x02	Length of remaining bytes	0	0	0	0	0	0	1	0

Variable header

Byte3	-	Packet identifier MSB	-	-	-	-	-	-	-	-
Byte4	-	Packet identifier LSB	-	-	-	-	-	-	-	-

FIGURE 29 PUBREC Packet

First byte is a publish receive packet and value is 0x50, second byte is length of remaining bytes it is always 2 and third and fourth bytes are packet identifier used for QoS2.

Example 3:

PUBREC contains 4 bytes. First byte **PUBREC** packet code, value 0x50 followed by length of remaining bytes that is 0x02 and at last 2 bytes of packet identifier. **PUBREC** packet has no payload is shown in figure 30.

Fixed header

Byte	Description	Value
01	PUBREC packet	0x50
02	Packet length	0x02

Variable header

03,04	Packet identifier	0x00 0x02

FIGURE 30 Example of PUBREC Packet

3.2.12 PUBREL Packet

Publish release packet is the response to the publish received packet. This is third packet in QoS 2. It includes fixed header and variable header but not payload as shown in figure 31.

PUBREL packet

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x60	Publish release packet	0	1	1	0	0	0	0	0
Byte2	0x02	Length of remaining bytes	0	0	0	0	0	0	1	0

Variable header

Byte3	-	Packet identifier MSB	-	-	-	-	-	-	-	-
Byte4	-	Packet identifier LSB	-	-	-	-	-	-	-	-

FIGURE 31 PUBREL Packet

First byte is a publish release packet and value is 0x60, second byte is length of remaining bytes it is always 2 and third and fourth bytes are packet identifier used for QoS2.

Example 4:

PUBREL contains 4 bytes. First byte **PUBREL** packet code, value 0x60 followed by length of remaining bytes that is 0x02 and at last 2 bytes of packet identifier is shown in figure 32. **PUBREL** packet has no payload.

Fixed header		
Byte	**Description**	**Value**
01	PUBREL packet	0x60
02	Packet length	0x02
Variable header		
03,04	Packet identifier	0x00 0x02

FIGURE 32 Example of PUBREL Packet

3.2.13 PUBCOMP Packet

Publish complete packet is the response to the publish release packet. This is fourth and last packet in QoS 2 is shown in the figure 33. It includes fixed header and variable header but not payload.

PUBCOMP packet

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x70	Publish complete packet	0	1	1	1	0	0	0	0
Byte2	0x02	Length of remaining bytes	0	0	0	0	0	0	1	0

Variable header

Byte3	-	Packet identifier MSB	-	-	-	-	-	-	-	-
Byte4	-	Packet identifier LSB	-	-	-	-	-	-	-	-

FIGURE 33 PUBCOMP Packet

First byte is a publish complete packet and value is 0x70, second byte is length of remaining bytes it is always 2 and third and fourth bytes are packet identifier used for QoS2.

Example 5:

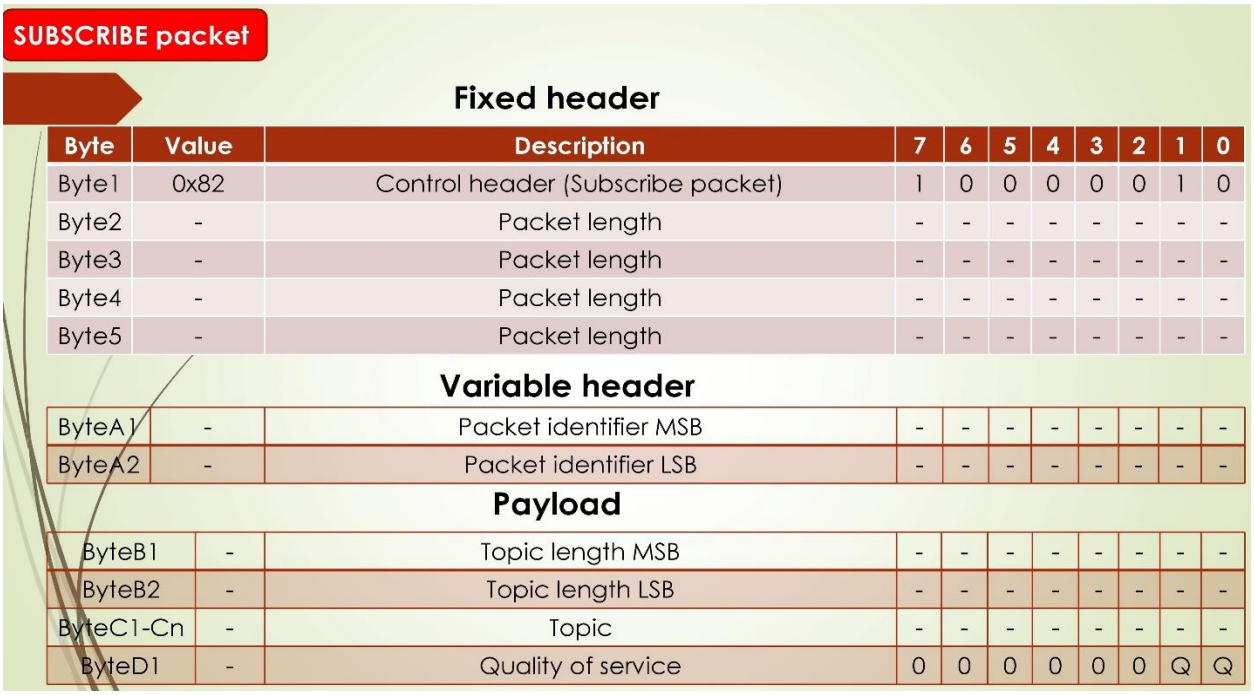
PUBCOMP contains 4 bytes. First byte **PUBCOMP** packet code, value 0x70 followed by length of remaining bytes that is 0x02 and at last 2 bytes of packet identifier is shown figure 34. **PUBCOMP** packet has no payload.

Fixed header		
Byte	**Description**	**Value**
01	PUBCOMP packet	0x70
02	Packet length	0x02
Variable header		
03,04	Packet identifier	0x00 0x02

FIGURE 34 PUBCOMP Packet

3.2.14 Subscribe packet

Publishing messages do not make sense if no one ever receives it. It means that no client has subscribed to topic of the message. 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 list of subscriptions.



Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0x82	Control header (Subscribe packet)	1	0	0	0	0	0	1	0
Byte2	-	Packet length	-	-	-	-	-	-	-	-
Byte3	-	Packet length	-	-	-	-	-	-	-	-
Byte4	-	Packet length	-	-	-	-	-	-	-	-
Byte5	-	Packet length	-	-	-	-	-	-	-	-
ByteA1	-	Packet identifier MSB	-	-	-	-	-	-	-	-
ByteA2	-	Packet identifier LSB	-	-	-	-	-	-	-	-
ByteB1	-	Topic length MSB	-	-	-	-	-	-	-	-
ByteB2	-	Topic length LSB	-	-	-	-	-	-	-	-
ByteC1-Cn	-	Topic	-	-	-	-	-	-	-	-
ByteD1	-	Quality of service	0	0	0	0	0	0	Q	Q

FIGURE 35 Subscribe Packet

Subscribe packet includes fixed header, variable header and payload as shown figure 35.

3.2.14.1 Fixed header

First byte of fixed header is a subscribe packet code and value is 0x82.

Bytes 2 to 5 denote the packet length and varies depending on packet size.

3.2.14.2 Variable header

Variable header will have 2 bytes of unique packet identifier.

3.2.14.3 Payload

First 2 bytes of payload will be length of topic/s and followed by topic/s. Last byte decides quality of service in which client is interested.

Example 6:

Subscribe Topic is CC:50:E3:9B:F7:84/led

QoS 0:

In this example client subscribes to topic ***CC:50:E3:9B:F7/led*** with QoS 0 and packet identifier 0x00 0x01

topic name "CC:50:E3:9B:F7:84/led" QoS0

Fixed header

Byte	Description	Value
01	Subscribe packet	0x82
02	Packet length	0x1A (26 bytes)

Variable header

03,04	Packet identifier	0x00 0x01

Payload

05,06	Topic length	0x00 0x15
07 - 28	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x6C 0x65 0x64 {CC:50:E3:9B:F7:84/led}
29	QoS	0

FIGURE 36 a. Example of Subscribe Topic QoS0

QoS 1:

In this example client subscribes to topic ***CC:50:E3:9B:F7/led*** with QoS 1 and packet identifier 0x00 0x01.

topic name "CC:50:E3:9B:F7:84/led" QoS1

Fixed header

Byte	Description	Value
01	Subscribe packet	0x82
02	Packet length	0x1A (26 bytes)

Variable header

03,04	Packet identifier	0x00 0x01

Payload

05,06	Topic length	0x00 0x15
07 - 28	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x6C 0x65 0x64 {CC:50:E3:9B:F7:84/led}
29	QoS	1

FIGURE 36 b. Example of Subscribe Topic QoS1

QoS 2:

In this example client subscribes to topic ***CC:50:E3:9B:F7/led*** with QoS 1 and packet identifier 0x00 0x01.

topic name "CC:50:E3:9B:F7:84/led" QoS2

Fixed header

Byte	Description	Value
01	Subscribe packet	0x82
02	Packet length	0x1A (26 bytes)

Variable header

03,04	Packet identifier	0x00 0x01

Payload

05,06	Topic length	0x00 0x15
07 - 28	Topic	0x43 0x43 0x3A 0x35 0x30 0x3A 0x45 0x33 0x3A 0x39 0x42 0x3A 0x46 0x37 0x3A 0x38 0x34 0x2F 0x6C 0x65 0x64 {CC:50:E3:9B:F7:84/led}
29	QoS	2

FIGURE 36 c. Example of Subscribe Topic QoS2

3.2.15 SUBACK packet

Subscribe acknowledge packet is the response to the subscribe packet. Based on quality of service requested in subscribe packet, the content in **SUBACK** packet varies. **SUBACK** packet includes fixed header, variable header and payload is as shown in figure 37.

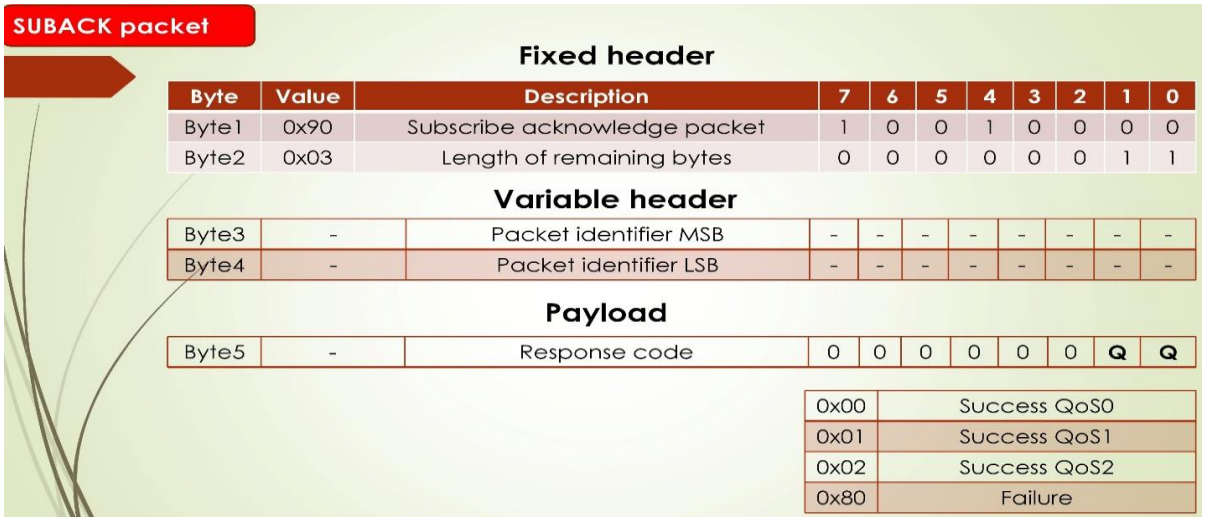


FIGURE 37 SUBACK Packet

3.2.15.1 Fixed header

First byte of fixed header is a subscribe acknowledge packet code and its value 0x90 second byte is remaining length of the packet and it is 0x03.

3.2.15.2 Variable header

Variable header includes 2 bytes of packet identifier.

3.2.15.3 Payload

SUBACK packet includes single byte as payload. Based on quality of service requested, the broker sends QoS which is applicable to particular subscribe request. If response is 0x80 it means the subscribe packet is failed to register in broker. Client must pay attention when it receives response 0x80.

Example 7:

SUBACK contains 5 bytes. First byte **SUBACK** packet code, value 0x90 followed by length of remaining bytes that is 0x03 and followed by 2 bytes of packet identifier and last QoS supported is shown in figure 38.

Fixed header

Byte	Description	Value
01	SUBACK packet	0x90
02	Packet length	0x03

Variable header

03,04	Packet identifier	0x00 0x01

Payload

05	QoS	0x00

FIGURE 38 Example of SUBACK Packet

3.2.16 UNSUBSCRIBE Packet

The client can unsubscribe to a topic in the same way in which it subscribes. Unsubscribe packet will help the client in achieving the same. As shown in figure 39 it includes fixed header, variable header and payload.

UNSUBSCRIBE packet

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0xA2	Control header (Subscribe packet)	1	0	1	0	0	0	1	0
Byte2	-	Packet length	-	-	-	-	-	-	-	-
Byte3	-	Packet length	-	-	-	-	-	-	-	-
Byte4	-	Packet length	-	-	-	-	-	-	-	-
Byte5	-	Packet length	-	-	-	-	-	-	-	-

Variable header

ByteA1	-	Packet identifier MSB	-	-	-	-	-	-	-	-
ByteA2	-	Packet identifier LSB	-	-	-	-	-	-	-	-

Payload

ByteB1	-	Topic length MSB	-	-	-	-	-	-	-	-
ByteB2	-	Topic length LSB	-	-	-	-	-	-	-	-
ByteC1-Cn	-	Topic	-	-	-	-	-	-	-	-

FIGURE 39 UNSUBSCRIBE Packet

3.2.16.1 Fixed header

First byte of fixed header is a unsubscribe packet code and value is 0xA2.

Byte 2 to 5 are packet length and varies depending on packet size.

3.2.16.2 Variable header

Variable header will have 2 bytes of unique packet identifier.

3.2.16.3 Payload

First 2 bytes of payload will be length of topic/s and followed by topic/s.

Example 8:

Unsubscribe topic: CC:50:E3:9B:F7:84/led

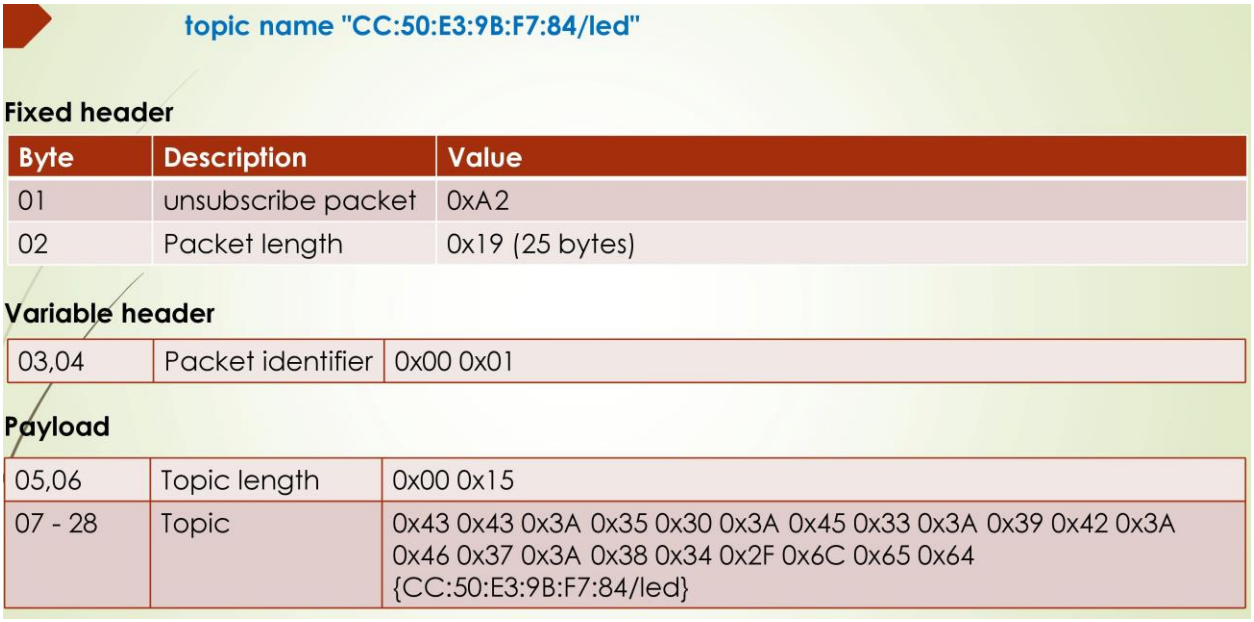


FIGURE 40 Example of Unsubscribe Topic

As shown in figure 40 image unsubscribe packet includes packet identifier as variable header and topic name as payload.

3.2.17 UNSUBACK packet

Unsubscribe acknowledge packet is the response to unsubscribe packet. This packet has fixed

header and variable header and no payload as shown in figure 41.

Fixed header

Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0xB0	Subscribe acknowledge packet	1	0	1	1	0	0	0	0
Byte2	0x03	Length of remaining bytes	0	0	0	0	0	0	1	0

Variable header

Byte3	-	Packet identifier MSB	-	-	-	-	-	-	-	-
Byte4	-	Packet identifier LSB	-	-	-	-	-	-	-	-

FIGURE 41 UNSUBACK packet

3.2.17.1 Fixed header

First byte of fixed header is an unsubscribe acknowledge packet code and its value 0xB0 second byte is remaining length of the packet and is 0x02.

3.2.17.2 Variable header

Variable header includes 2 bytes of packet identifier.

Example 9:

Unsuback packet is four-byte packet which includes packet identifier and has no payload as shown in figure 42.

Fixed header

Byte	Description	Value
01	UNSUBACK packet	0xB0
02	Packet length	0x02

Variable header

03,04	Packet identifier	0x00 0x02

FIGURE 42 Example of Unsuback Packet

3.2.18 PINGREQ Packet

Ping request packet is used when client has no message to send to broker but need to inform the broker that client is alive. On configured intervals, ping request is sent to broker and broker will

respond to it. Ping request packet includes only fixed header packet as shown in figure 43.

Fixed header									
Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0xD0	Ping Response packet	1	1	0	1	0	0	0	0
Byte2	0x00	Length of remaining bytes	0	0	0	0	0	0	0	0

FIGURE 43 PINGREQ Packet

3.2.19 PINGRES Packet

Ping response packet is response packet from broker to client when client sends **PINGREQ** packet. Ping request packet includes only fixed header packet as shown in figure 44.

Fixed header									
Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0xD0	Ping Response packet	1	1	0	1	0	0	0	0
Byte2	0x00	Length of remaining bytes	0	0	0	0	0	0	0	0

FIGURE 44 PINGRES Packet Fixed Header

3.2.20 DISCONNECT Packet

Disconnect packet is the last packet send by client to broker in order to disconnect from MQTT protocol is shown in figure 45.

Fixed header									
Byte	Value	Description	7	6	5	4	3	2	1	0
Byte1	0xE0	Disconnect packet	1	1	1	0	0	0	0	0
Byte2	0x00	Length of remaining bytes	0	0	0	0	0	0	0	0

FIGURE 45 DISCONNECT Packet Fixed Header

3.3 Sample code

Following is the sample code written on c++ using esp32 microcontroller to test MQTT protocol.

```
/*************************************************************************
```

```
 * Project Name: Mqtt_Demo
 *
 * Version: 1.0
 *
 * Description:
 * In this project ESP32 handles Mqtt connections
 * Owner:
 * Yogesh M Iggalore
 *
 ******************************************************************************
 * no Copyright included
 ******************************************************************************/
/* include wifi library for esp32 module */
#include <WiFi.h>
/* additional libraries for esp32 */
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#ifdef __cplusplus
extern "C" {
#endif
uint8_t temprature_sens_read();
#ifdef __cplusplus
}
#endif
uint8_t temprature_sens_read();
/* include mqtt asynchronous client */
#include <AsyncMqttClient.h>

/* wifi ssid and password */
#define WIFI_SSID "IOTDev"
#define WIFI_PASSWORD "IOTDev1234"
//#define WIFI_SSID "METI BSNL"
//#define WIFI_PASSWORD "metibsnl$"

/* mqtt server/host */
#define MQTT_HOST         "silicosmos.in"
/* mqtt port secure communication use 8883 */
#define MQTT_PORT         1883
/* mqtt credentails for connection username and password */
#define MQTT_USERNAME     "yogesh"
#define MQTT_PASSWORD     "yogesh"
/*keep alive in second decides how long server should
  keep client connection */
```

```
#define MQTT_KEEPALIVE    60
/* mqtt Quality of Service
  0 : At most once
  1 : At least once
  2 : Exactly once
*/
#define MQTT_QOS          2

/* retain flag if client disconnect from server */
#define MQTT_RETAIN       1
/* onboard led pin number */
#define LED_PIN    2
/* cleintid should be unique */
String sClientId;
/* will messege on client disconnection and connection to other client */

String sMqttWillTopic;
/* mqtt topic for demo */
String sMqttTemperatureTopic;
String sMqttHallsensorTopic;
String sMqttLedTopic;
uint16_t ui16HallsensorValue;
float    fTemperatureValue;
uint16_t ui16TempPacketId;
uint16_t ui16HallPacketId;
uint16_t ui16LedPacketId;
uint8_t  ui8MqttStatus=0;
char buff[10];
/* mqtt client instance */
AsyncMqttClient mqttClient;
/* timer handling for mqtt and wifi reconnection */
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;
/* timer interval for reading temperature and hall sensor */
TimerHandle_t ReadPublishIntervalTimer;
/**************************************************************************
* Function Name: Connect_To_Wifi
**************************************************************************
* Summary:
*  This function connects esp32 module to wifi with given ssid and password

*
* Parameters:
*  None
*
```

```
 * Return:
 *  None
 *
 ******************************************************************************
void Connect_To_Wifi(void) {
    Serial.println("Connecting to Wi-Fi...");
    /* this line begins the wifi connction */
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    /* for client id we are using mac address */
    sClientId = WiFi.macAddress();
    /* will message topic  */
    sMqttWillTopic = WiFi.macAddress() + "/status";
    Serial.println(sMqttWillTopic);
    /* temperature read topic */
    sMqttTemperatureTopic = WiFi.macAddress() + "/temp";
    Serial.println(sMqttTemperatureTopic);
    /* hall sensor read topic */
    sMqttHallsensorTopic = WiFi.macAddress() + "/hall";
    Serial.println(sMqttHallsensorTopic);
    /* led control topic */
    sMqttLedTopic = WiFi.macAddress() + "/led";
    Serial.println(sMqttLedTopic);
}
/******************************************************************************

 * Function Name: Read_Publish_Temp_Hall
 ******************************************************************************

 * Summary:
 *  This function reads temperature and hall values and publish to server
 *
 *
 * Parameters:
 *  None
 *
 * Return:
 *  None
 *
 ******************************************************************************/

void Read_Publish_Temp_Hall(void){
    uint16_t ui16UnScb;
    if(ui8MqttStatus){
        /* read hall sensor value */
        ui16HallsensorValue = hallRead();
```

```
            /* read temperature value */
            fTemperatureValue = (temprature_sens_read()-32)/1.8;
            /*print hall and temperature values */
            Serial.print("HALL:");
            Serial.printf("%d\n",ui16HallsensorValue);
            Serial.print("TEMP:");
            Serial.printf("%f\n",fTemperatureValue);
            /* publish temperature topic to server */
            snprintf (buff, sizeof(buff), "%f", fTemperatureValue);
            ui16TempPacketId = mqttClient.publish(sMqttTemperatureTopic.c_str()
                               ,MQTT_QOS, MQTT_RETAIN, buff);
            Serial.printf("Publishing at QoS %d, packetId: %d\n",MQTT
                          _QOS,ui16TempPacketId);
            /* publish hall sensor topic to server */
            snprintf (buff, sizeof(buff), "%d", ui16HallsensorValue);
            ui16HallPacketId = mqttClient.publish(sMqttHallsensorTopic.c_str(),2,
                               MQTT_RETAIN, buff);

              Serial.printf("Publishing at QoS %d, packetId: %d\n",MQTT_QOS,
                            ui16HallPacketId);
        }
    }
    /**************************************************************************
    * Function Name: Connect_To_Mqtt
    **************************************************************************
    * Summary:
    *  This function connects esp32 module to wifi with given ssid and password

    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
    *
    **************************************************************************/
    void Connect_To_Mqtt(void) {
        Serial.println("Connecting to MQTT...");
        /* set mqtt client id */
        mqttClient.setClientId(sClientId.c_str());
        /* set username and password */
        mqttClient.setCredentials(MQTT_USERNAME,MQTT_PASSWORD);
        /* set server and port */
        mqttClient.setServer(MQTT_HOST, MQTT_PORT);
        /* set keepalive value */
```

```
    mqttClient.setKeepAlive(MQTT_KEEPALIVE);
    /* set will message */
    mqttClient.setWill(sMqttWillTopic.c_str(),MQTT_QOS,MQTT_RETAIN,
                       "offline",strlen("offline"));
    /* connect to mqtt */
    mqttClient.connect();
}
/*******************************************************************************
* Function Name: WiFiEvent
********************************************************************************
*
* Summary:
*  This function calls when any wifi event like connection and
*  disconnection happens
*
* Parameters:
*  None
*
* Return:
*  None
*
*******************************************************************************/
void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event){
        /* If modules got IP from router */
        case SYSTEM_EVENT_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            /* connect to mqtt */
            Connect_To_Mqtt();
            break;
        /* if module disconnects from router */
        case SYSTEM_EVENT_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            /* stop mqqt reconnect timer */
            xTimerStop(mqttReconnectTimer, 0);
            /* start wifi re connect timer */
            xTimerStart(wifiReconnectTimer, 0);
            break;
        default:
            break;
    }
}
```

```
/*****************************************************************************
* Function Name: On_Mqtt_Connect
*****************************************************************************
*
* Summary:
*  This function calls when mqtt connects to server
*
*
* Parameters:
*  None
*
* Return:
*  None
*
*****************************************************************************/
void On_Mqtt_Connect(bool sessionPresent) {
    uint16_t ui16LoopCounter=0;
    Serial.println("Connected to MQTT.");
    /* read hall sensor value */
    ui16HallsensorValue = hallRead();
    /* read temperature value */
    fTemperatureValue = (temprature_sens_read()-32)/1.8;
    /*print hall and temperature values */
    Serial.print("HALL:");
    Serial.printf("%d\n",ui16HallsensorValue);
    Serial.print("TEMP:");
    Serial.printf("%f\n",fTemperatureValue);
    /* print session */
    Serial.print("Session present: ");
    Serial.println(sessionPresent);

    /* publish temperature topic to server */
    ui16TempPacketId = mqttClient.publish(sMqttTemperatureTopic.c_str(),
                       MQTT_QOS, MQTT_RETAIN, "test");
    Serial.printf("Publishing at QoS %d, packetId: %d\n",MQTT_QOS,
                  ui16TempPacketId);
    /* publish hall sensor topic to server */
    ui16HallPacketId = mqttClient.publish(sMqttHallsensorTopic.c_str(),
                       MQTT_QOS, MQTT_RETAIN, "test");
    Serial.printf("Publishing at QoS %d, packetId: %d\n",2,ui16HallPacketId);

    /* publish will topic to server */
    mqttClient.publish(sMqttWillTopic.c_str(),MQTT_QOS, MQTT_RETAIN,
                       "online");
    Serial.printf("Publishing will topic\n");
```

```
        /* subscribe to led control packet */
        uint16_t ui16LedPacketId = mqttClient.subscribe(sMqttLedTopic.c_str(),
                                   MQTT_QOS);
        Serial.printf("Subscribing to led topic QoS:%d\n", ui16LedPacketId);
        ui8MqttStatus = 1;
    }
    /**************************************************************************
    * Function Name: On_Mqtt_Disconnect
    **************************************************************************
    *
    * Summary:
    *  This function calls when mqtt disconnects to server
    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
    *
    **************************************************************************/

    void On_Mqtt_Disconnect(AsyncMqttClientDisconnectReason reason) {
        Serial.println("Disconnected from MQTT.");
        /* if wifi is connected then reconnect to mqtt */
        if (WiFi.isConnected()) {
            xTimerStart(mqttReconnectTimer, 0);
        }
        ui8MqttStatus = 0;
    }
    /**************************************************************************
    * Function Name: On_Mqtt_Subscribe
    **************************************************************************
    *
    * Summary:
    *  This function calls when client subscribe to topic
    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
    *
    **************************************************************************/
    void On_Mqtt_Subscribe(uint16_t packetId, uint8_t qos) {
        uint16_t ui16LoopCounter=0;
```

```
        Serial.print("Mqtt subscribe pkt: ");
        for(ui16LoopCounter=0;ui16LoopCounter<
               mqttClient.ui16DebugPacketCountLength;ui16LoopCounter++){
            Serial.print(mqttClient.aui8DebugPacket[ui16LoopCounter],HEX);
            Serial.print(" ");
        }
        Serial.println(" ");
        Serial.println("Subscribe acknowledged.");
        Serial.print("  packetId: ");
        Serial.println(packetId);
        Serial.print("  qos: ");
        Serial.println(qos);
    }
    /*****************************************************************************
    * Function Name: On_Mqtt_Unsubscribe
    *****************************************************************************
    *
    * Summary:
    *  This function calls when client unsubscribe to topic
    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
    *
    *****************************************************************************/

    void On_Mqtt_Unsubscribe(uint16_t packetId) {
        uint16_t ui16LoopCounter=0;

        Serial.print("Mqtt unsubscribe pkt: ");
        for(ui16LoopCounter=0;ui16LoopCounter<
              mqttClient.ui16DebugPacketCountLength;ui16LoopCounter++){
            Serial.print(mqttClient.aui8DebugPacket[ui16LoopCounter],HEX);
            Serial.print(" ");
        }
        Serial.println(" ");
        Serial.println("Unsubscribe acknowledged.");
        Serial.print("packetId: ");
        Serial.println(packetId);
    }
    /*****************************************************************************
    * Function Name: On_Mqtt_Message
    *****************************************************************************
```

```
     *
     * Summary:
     *  This function calls when client received any message from server
     *
     *
     * Parameters:
     *  None
     *
     * Return:
     *  None
     *
     ************************************************************************/
     void On_Mqtt_Message(char* topic, char* payload,AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total){
          uint16_t ui16LoopCounter=0;
          Serial.print("Mqtt onMessage pkt: ");
          for(ui16LoopCounter=0;ui16LoopCounter<
               mqttClient.ui16DebugPacketCountLength;ui16LoopCounter++){
                Serial.print(mqttClient.aui8DebugPacket[ui16LoopCounter],HEX);
                Serial.print(" ");
          }
          Serial.println(" ");
          Serial.println("Publish received.");
          Serial.print("  topic: ");
          Serial.println(topic);
          Serial.print("  qos: ");
          Serial.println(properties.qos);
          Serial.print("  dup: ");
          Serial.println(properties.dup);
          Serial.print("  retain: ");
          Serial.println(properties.retain);
          Serial.print("  len: ");
          Serial.println(len);
          Serial.print("  index: ");
          Serial.println(index);
          Serial.print("  total: ");
          Serial.println(total);
          Serial.print("  payload: ");
          Serial.println(payload);
          /* check for topic received */
          if(strcmp(topic,sMqttLedTopic.c_str()) == 0){
              /* if payload is "on" then switch on led else switch off led */
              if(strcmp(payload,"on")){
                  digitalWrite(LED_PIN,LOW);
              }else{
```

```
                digitalWrite(LED_PIN,HIGH);
            }
        }
    }
    /*****************************************************************************
    * Function Name: On_Mqtt_Publish
    *****************************************************************************
    *
    * Summary:
    *  This function calls on sucess of mqtt publish
    *
    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
    *
    *****************************************************************************/
    void On_Mqtt_Publish(uint16_t packetId) {
        uint16_t ui16LoopCounter=0;
        Serial.print("Mqtt publish pkt: ");
        for(ui16LoopCounter=0;ui16LoopCounter<
               mqttClient.ui16DebugPacketCountLength;ui16LoopCounter++){
            Serial.print(mqttClient.aui8DebugPacket[ui16LoopCounter],HEX);
            Serial.print(" ");
        }
        Serial.println(" ");

        Serial.println("Publish acknowledged.");
        Serial.print("  packetId: ");
        Serial.println(packetId);
    }
    /*****************************************************************************
    * Function Name: setup
    *****************************************************************************
    * Summary:
    *  This function calls on start of program
    *
    *
    * Parameters:
    *  None
    *
    * Return:
    *  None
```

```
     *
     ******************************************************************************/
     void setup() {
         /* set serial baudrate to 115200 and start serial port */
         Serial.begin(921600);
         Serial.println();
         Serial.println();
         /* set mqtt timer with 2 second interval */
         mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFAL
SE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(Connect_To_Mqtt));
         /* set wifi timer with 2 second interval */
         wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFAL
SE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(Connect_To_Wifi));
         /* set timer interval to 60 second to read and publish data */
         ReadPublishIntervalTimer = xTimerCreate("ReadPublishIntervalTimer", pdMS_
TO_TICKS(10000), pdTRUE, (void*)0, reinterpret_cast<TimerCallbackFunction_t>(Read_
Publish_Temp_Hall));
         /* set callback function on wifi event */
         WiFi.onEvent(WiFiEvent);
         /* set callback function on mqtt connect */
         mqttClient.onConnect(On_Mqtt_Connect);
         /* set callback function on mqtt disconnect */
         mqttClient.onDisconnect(On_Mqtt_Disconnect);
         /* set callback function on client subscribe to topic */
         mqttClient.onSubscribe(On_Mqtt_Subscribe);

         /* set callback function on client unsubscribe to topic */
         mqttClient.onUnsubscribe(On_Mqtt_Unsubscribe);
         /* set callback function client on received mqtt messege */
         mqttClient.onMessage(On_Mqtt_Message);
         /* set callback function on client publish topic */
         mqttClient.onPublish(On_Mqtt_Publish);
         /* connect to wifi */
         Connect_To_Wifi();
         /* set onbaord led pin as output */
         pinMode(LED_PIN, OUTPUT);
         /* start read publish interval timer */
         xTimerStart(ReadPublishIntervalTimer, 0);
     }
     void loop() {
     }
```

3.4 Server-side implementation

Eclipse mosquitto is an open source message broker that implements the MQTT protocol. It supports MQTT version 5,3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low single board computer to full server. The MQTT protocol provides a lightweight method of carrying out messaging using a publish/subscribe model. This makes it suitable for Internet of Things messaging such as with low power sensors or mobile devices such as phones, embedded computers or microcontrollers.

The Mosquitto project also provides a C library for implementing MQTT clients, and the very popular mosquitto_pub and mosquitto_sub command line MQTT clients. Mosquitto is part of the Eclipse Foundation, is an iot.eclipse.org project and is sponsored by cedalo.com.

Here mosquitto broker installation procedure is given below

Step 1

Require a Linux server

```
.320
sudo apt-get update
sudo apt-get install mosquitto
```

using these two commands mosquitto broker is installed

Step 2

install mqtt clients

```
sudo apt-get install mosquitto-clients
```

Mosquitto clients help us easily test MQTT through a command line utility

Subscribe test in server

```
mosquitto_sub -t "test"
```

Publish test in server

Open another ssh server connection and publish a message in mosquitto broker with the command

```
mosquitto_pub -m "message from mosquitto_pub client" -t "test"
```

Step 3

Secure with the password

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd dave
```

after this command type the password

Create a configuration file for Mosquitto pointing to the password file we have just created.

```
sudo nano /etc/mosquitto/conf.d/default.conf
```

This will open an empty file. Paste the following into it.

```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Save and exit the text editor with "`Ctrl+O`", "`Enter`" and "`Ctrl+X`".

Now restart Mosquitto server and test our changes.

```
sudo systemctl restart mosquitto
```

In the subscribe client window, press "Ctrl+C" to exit the subscribe client and restart it with following command.

```
mosquitto_sub -t "test" -u "dave" -P "password"
```

Note the capital -P here.

In the publish client window, try to publish a message without a password.

```
mosquitto_pub -t "test" -m "message from mosquitto_pub client"
```

The message will be rejected with following error message.

```
Connection Refused: not authorised.
Error: The connection was refused.
```

Now publish a message with the username and password.

```
mosquitto_pub -t "test" -m "message from mosquitto_pub client" -u
"dave" -P "password"
```

Step 4

To add extra user

```
mosquitto_passwd -b passwordfile username password
```

use this command for adding an extra user

Note: Some servers don't allow direct port forwarding. So, it is suggested to allow firewall for particular port of MQTT in the server. Default port for MQTT 1883.

Bibliography

[1] https://www.codeinstitute.net/blog/what-is-iot/

[2] https://www.iso.org/standard/69466.html

[3] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

[4] https://www.hivemq.com/mqtt-essentials/

[5] *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.