IOT COMMUNICATION PROTOCOLS- (MQTT)

Publish Subscribe Communication Model

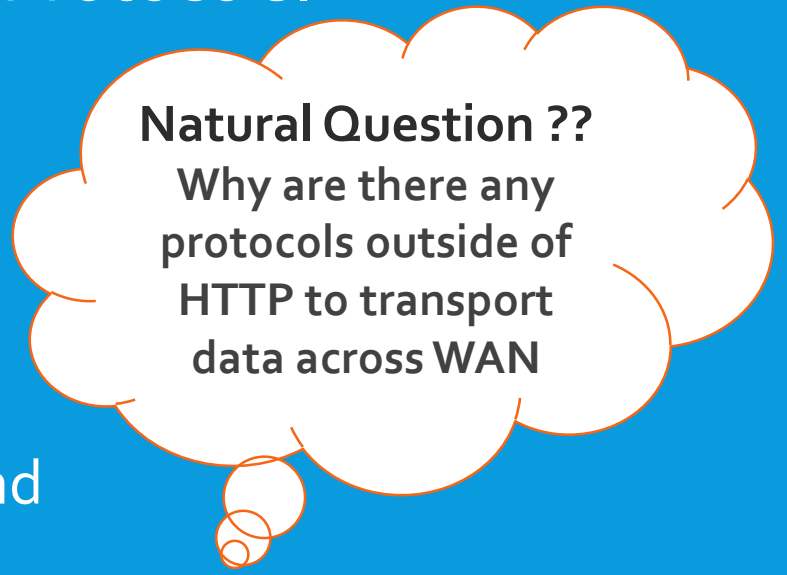
Bhupendra Pratap Singh,

CDAC, ACTS, Pune

PROTOCOLS -

Set of rules and guidelines for communicating data

- Many protocols like ftp , RTP and HTTP are present from traditions what is need of specific IoT Protocols.



- HTTP has provided significant services and abilities for the internet over 20 years,

yet was designed and architected for **General purpose Computing** in Client –Server model.

NEED OF PROTOCOLS?

- **Transfer Data**
  - From end device/nodes/motes to end device/nodes/motes
  - From end device/nodes/motes to gateway/middleware
  - From end device/nodes/motes to server
  - From Gateway/middleware to server

CONT. .....

- **Represent Data in**
  - Readable Format
  - Encrypted
- **Resources**
  - Computational Capacity
  - Power

NEED OF IOT PROTOCOLS

- IoT Devices can be very constrained in computing resources, remote and bandwidth limited.
- Due to above reason **more efficient, secure and scalable protocols are necessary to manage a plethora of devices** in various network topologies such as mesh network.

MQTT (MQ TELEMETRY TRANSPORT)

HISTORY

- IBM WebSphere Message Queue technology was first conceived in 1993 to address problems in independent and non-concurrent distributed systems to securely communicate.
- A derivative of the WebSphere Message Queue was authored by **Andy Stanford-Clark** and **Arlen Nipper** at IBM in 1999 to address the particular constraints of connecting remote oil and gas pipelines over a satellite connection. **That protocol termed as the MQTT**.

CONT. .....

- MQTT was an internal and proprietary protocol for IBM for many years until being released in version 3.1 in 2010 as a royalty-free product. In 2013, MQTT was standardized and accepted in to OASIS consortium. In 2014, OASIS released it publically as version 3.1.1. MQTT is also an ISO standard (ISO/IEC PRF 20922).
- Latest release for review MQTT Version 5.0
- http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.pdf

THE GOAL OF IP BASED TRANSPORT PROTOCOL ARE

- It must be simple to implement
- To provide a form of Quality of Service
- To be very lightweight and bandwidth efficient.
- To be data agnostic.
- To have continuous session awareness.
- To address security issues.

A WELL DEFINED SUMMARY FROM MQTT.ORG

- MQTT stands for MQ Telemetry Transport. **It is a publish/subscribe, extremely simple and lightweight messaging protocol**, designed for **constrained devices and low-bandwidth**, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements while also attempting to ensure reliability and some degree of assurance of delivery.

CONT.....

- These principles also turn out to make the protocol ideal of the emerging "machine-to-machine" (M2M) or "Internet of Things" world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.
- TODO : refer http://mqtt.org/faq
  - http://mqtt.org/2014/11/mqtt-v3-1-1-now-an-oasis-standard

Organization for the Advancement of Structured Information Standards (**OASIS**)

INTRODUCTION

- Publish Subscribe model also known as pub/sub, mapping done through topic names.
- **Topic names can be hierarchical, cards(+, #) Allowed.**
- Works on TCP Transport layer by default …. TLS/SSL are possible.
- **Broker runs on port 1883 while over TLS it runs on 8883 by default.**
- Unlike a traditional client-server model, the clients are not aware of any physical identifiers such as the IP address and port.

CONT.....

- MQTT has three main components –
  - Publisher
  - Subscriber
  - Broker
- **Publisher publish the data over a TOPIC and subscriber has to subscribe to that topic in order to receive the data published over that topic.**

PACKET STRUCTURE

- Binary encoding
- Comes with min two bytes for essential header for payload upto 127 bytes.
- Max size of essential header is 5 bytes (1+4).

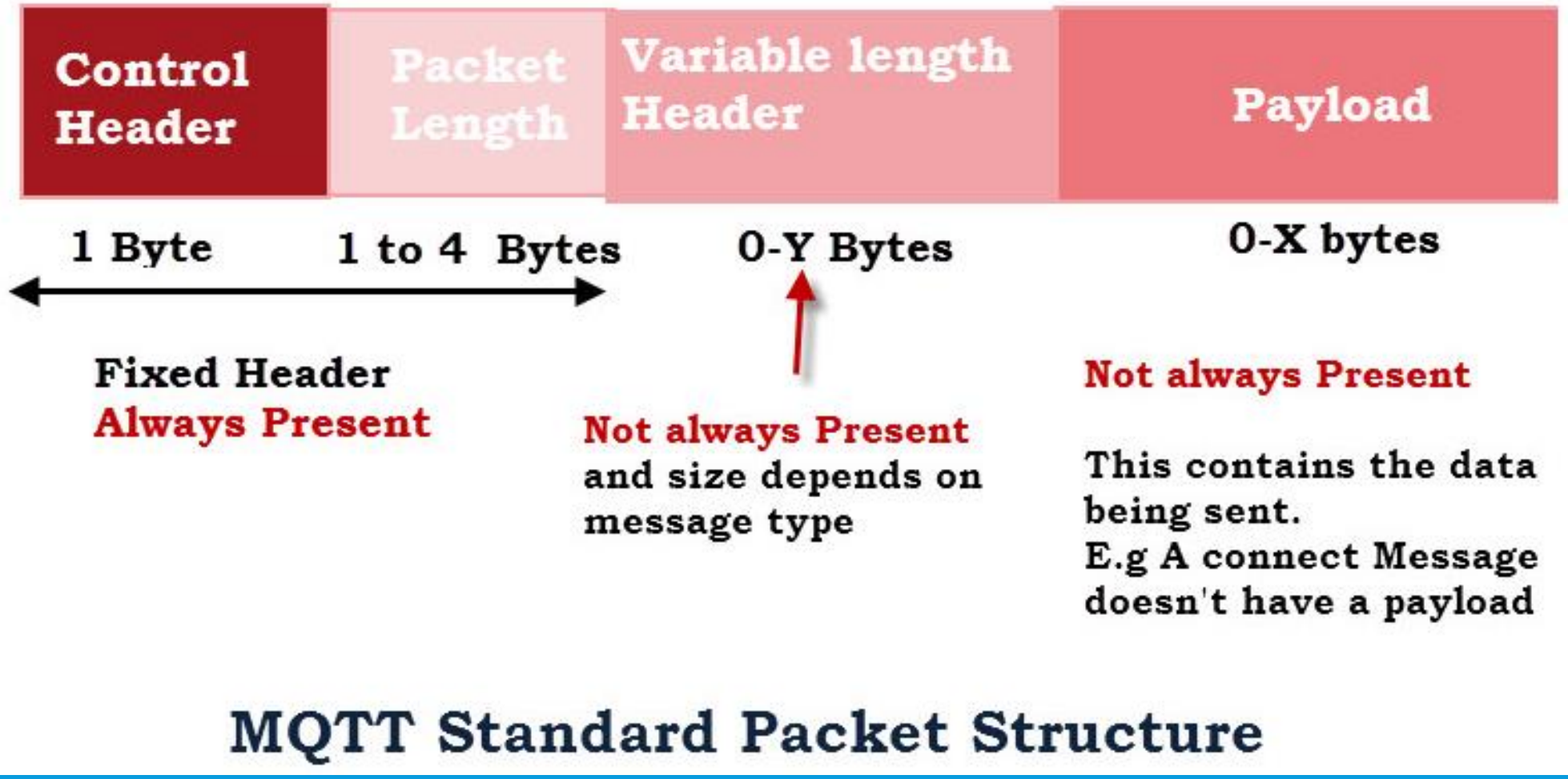
MQTT HEADERS

- Topic names, Client ID, User names and Passwords are encoded as **UTF-8 strings**.
- The **Payload** excluding **MQTT protocol information** like **Client ID** etc. is binary data and the content and format is application specific.
- The MQTT packet or message format consists of a 2 byte fixed header (always present) + Variable-header (not always present)+ payload (not always present).

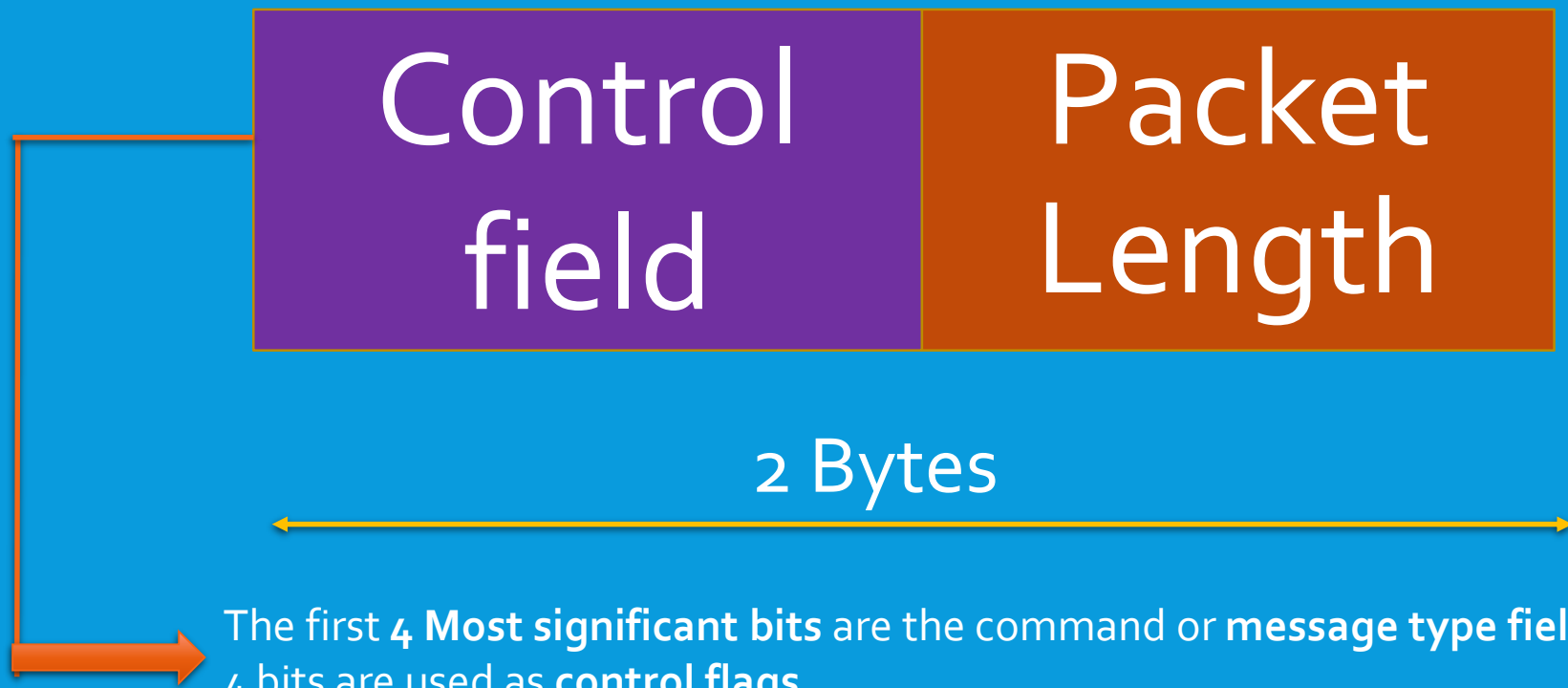
Possible Packet Formats:

- Fixed Header (Control field + Length) – Example CONNACK
- Fixed Header (Control field + Length) + Variable Header -Example PUBACK
- Fixed Header (Control field + Length) + Variable Header + payload -Example CONNECT

PACKET STRUCTURE CONT.....



MINIMUM PACKET SIZE



packet length field is 1 byte which is for messages with a total length less than 127 bytes

The first **4 Most significant bits** are the command or **message type field** and the other 4 bits are used as **control flags**.

CONTROL PACKET TYPES AND FLAG BITS

Table 2.1 - Control packet types

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server	Publish message

Table 2.2 - Flag Bits

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP[1]	QoS[2]	QoS[2]	RETAIN[3]
PUBACK	Reserved	0	0	0	0

CONT. .. (SUMMARY)

- The **fixed header field** consists of the **control field** and the variable length **packet length field.**
- The **minimum size** of the **packet length field** is **1 byte** which is for messages with a **total length** less than 127 bytes.(not including control and length fields).
- The maximum packet size is 256MB. Small packets less than **127 bytes** have a **1 byte** packet length field.
- Packets larger than 127 and less than 16383 will use 2 bytes. etc.
- The **minimum packet size** is just **2 bytes** with a **single byte control field** and a **single byte packet length field**. E.g. the **disconnect message** is only 2 bytes.

WHAT IS PUBLISHER, SUBSCRIBER AND BROKER?

PUBLISHER-

- A client transmitting a message is called a publisher.

SUBSCRIBER-

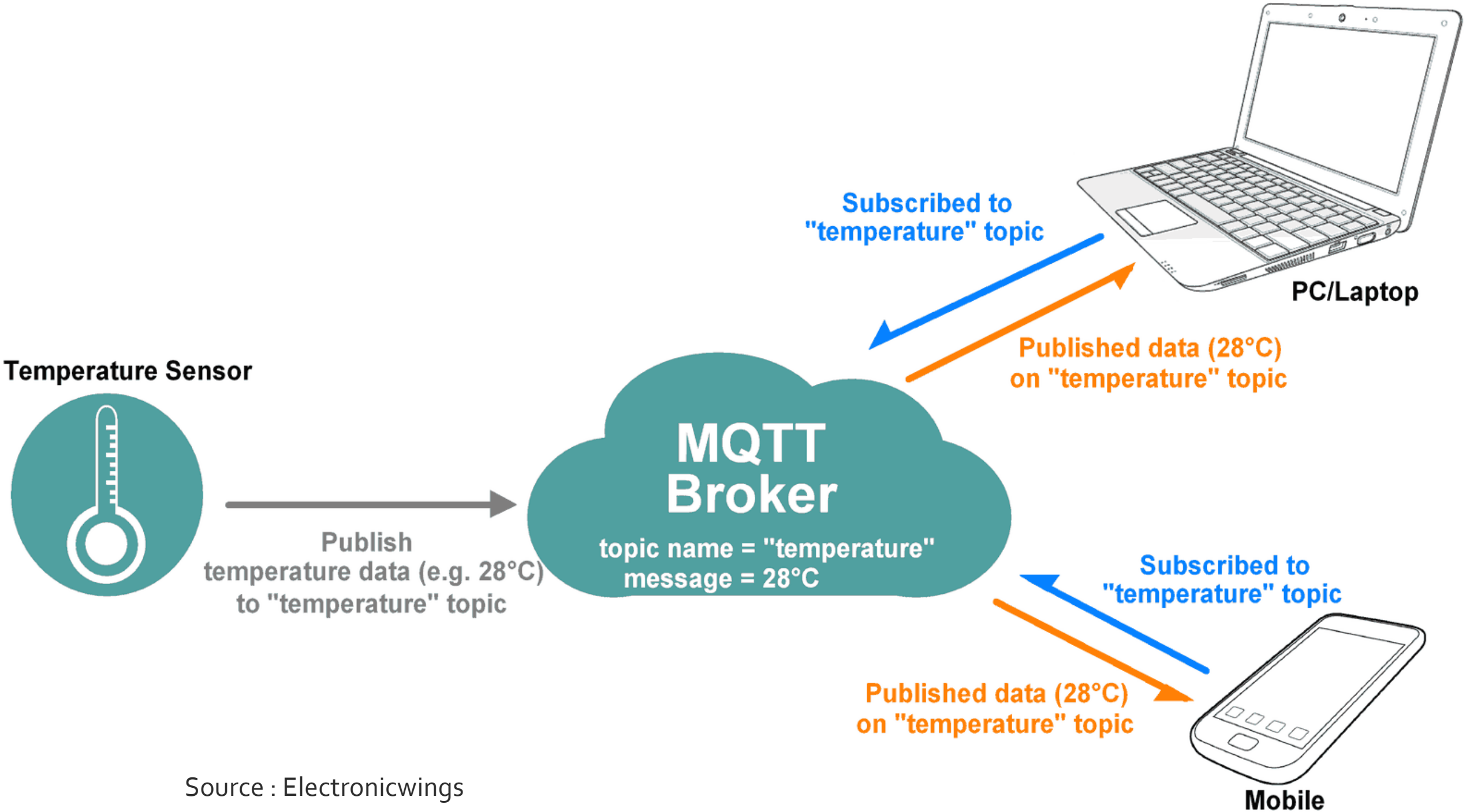
- A client receiving a message is called a receiver.

BROKER-

It is basically used to connect clients and filter data.

CONCEPT OF DECOUPLING

- The most important aspect of pub/sub is the **decoupling** of the publisher of the message from the recipient(subscriber). This decoupling has several dimensions:
- **Space decoupling**: Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).
- **Time decoupling**: Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling:** Operations on both components do not need to be interrupted during publishing or receiving.



Source : Electronicwings

WILD CARDS (APPLICABLE ONLY FOR SUBSCRIPTION)

- When a client subscribes to a topic, it can subscribe to the exact topic of a published message or it can use wildcards to subscribe to multiple topics simultaneously. A wildcard can only be used to subscribe to topics, not to publish a message. There are two different kinds of wildcards: ***single-level*** and ***multi-level***.
- **Single Level: +**

Any topic matches a topic with single-level wildcard if it contains an arbitrary string instead of the wildcard. For example a subscription to ***myhome/groundfloor/+/temperature*** can produce the following results:

- ✅ myhome / groundfloor / livingroom / temperature
- ✅ myhome / groundfloor / kitchen / temperature
- ❌ myhome / groundfloor / kitchen / brightness
- ❌ myhome / firstfloor / kitchen / temperature
- ❌ myhome / groundfloor / kitchen / fridge / temperature

MULTILEVEL (#)

- The multi-level wildcard covers many topic levels. The hash symbol represents the multi-level wild card in the topic. For the broker to determine which topics match, the multi-level wildcard must be placed as the last character in the topic and preceded by a forward slash.

SYSTOPIC

- Generally there is no such restriction kept on the naming culture of the topic, however there is an exception –
- **The $-symbol topics are reserved for internal statistics of the MQTT broker**
- For example :
- **$SYS/broker/load/bytes/received**: The total number of bytes received since the broker started.
- **$SYS/broker/load/bytes/sent**: The total number of bytes sent since the broker started.
- **$SYS/broker/clients/connected**: The number of currently connected clients
- **Explore more on :** https://github.com/mqtt/mqtt.github.io/wiki/SYS-Topics

MAXIMUM ALLOWABLE PACKET SIZE

- The maximum allowable packet size is **256 MB**, which allows for an extremely large payload.

Points to Remember:

A. Maximum data payload size is cloud and broker dependent.

IBM Watson allows for payload sizes up to 128 KB, while Google supports 256 KB.

MQTT ATTRIBUTES

- **QOS (Quality of service)**
  - This number 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)
  - There are three types/levels of quality of service –
  - **QOS – ZERO (Fire and forget – At most once)**
  - **QOS – 1 (At least Once)**
  - **QOS – 2 (Exactly Once)**

QOS – ZERO

- The minimal QoS level is zero. This service level guarantees a best-effort delivery. There is no guarantee of delivery. The recipient does not acknowledge receipt of the message and the **message is not stored and re-transmitted by the sender**.

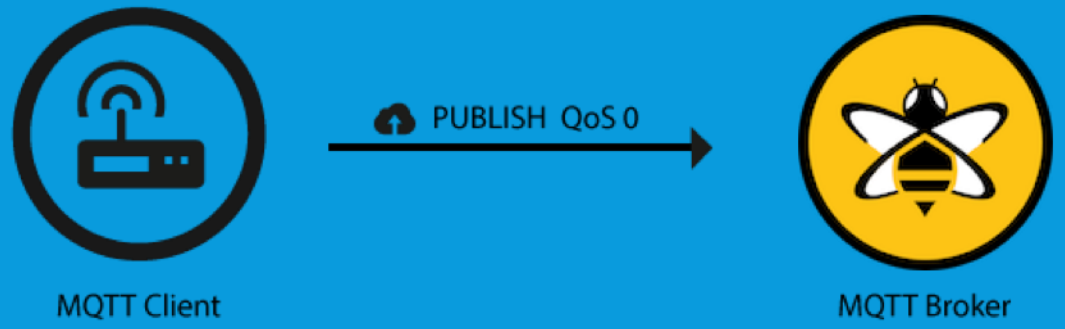
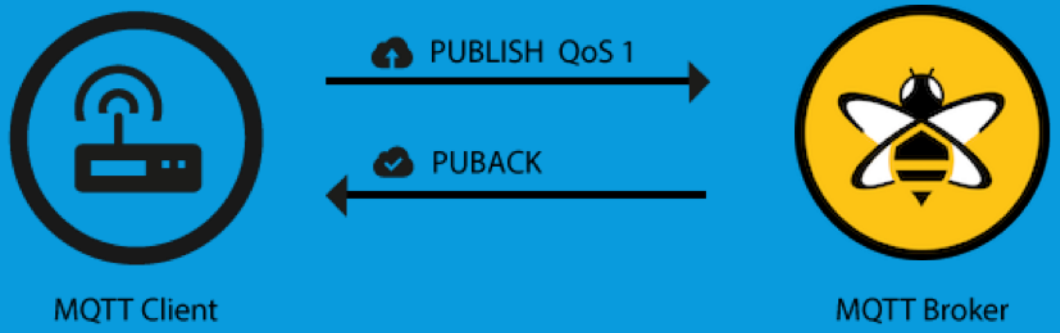


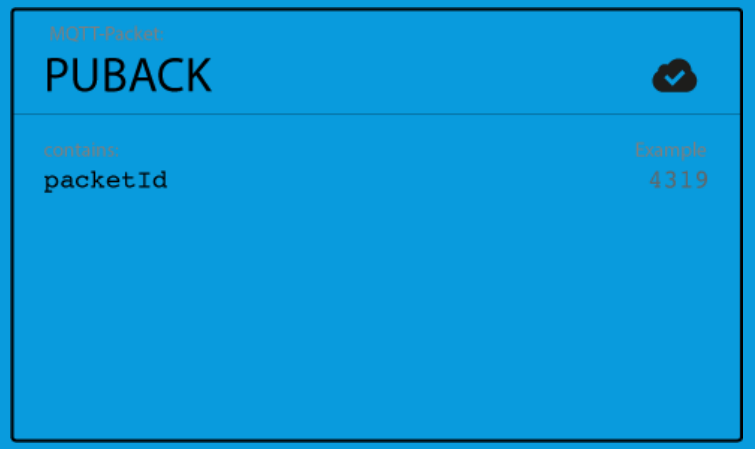
Image source- HiveMQ

QOS - 1

- It guarantees that a message is delivered at least one time to the receiver. The sender stores the message until it gets a **PUBACK** packet from the receiver that acknowledges receipt of the message. It is possible for a message to be sent or delivered multiple times.
- 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.

CONT. …





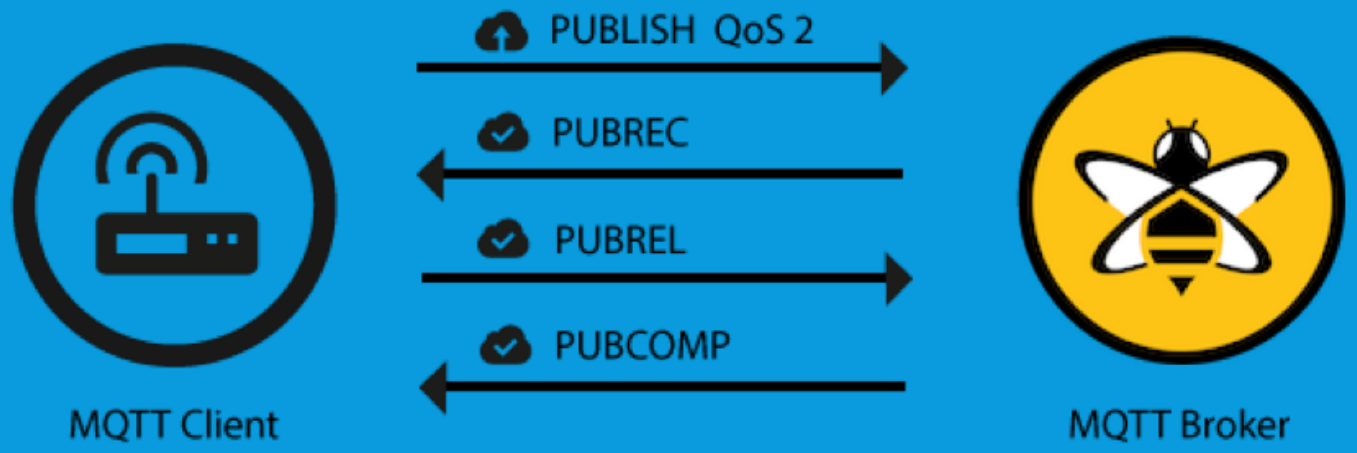
The sender uses the packet identifier in each packet to match the PUBLISH packet to the corresponding PUBACK packet. If the sender does not receive a PUBACK packet in a reasonable amount of time, the sender resends the PUBLISH packet. When a receiver gets a message with QoS 1, it can process it immediately. For example, if the receiver is a broker, the broker sends the message to all subscribing clients and then replies with a PUBACK packet

Image source- HiveMQ

QOS – 2 (FOUR WAY HANDSHAKING)

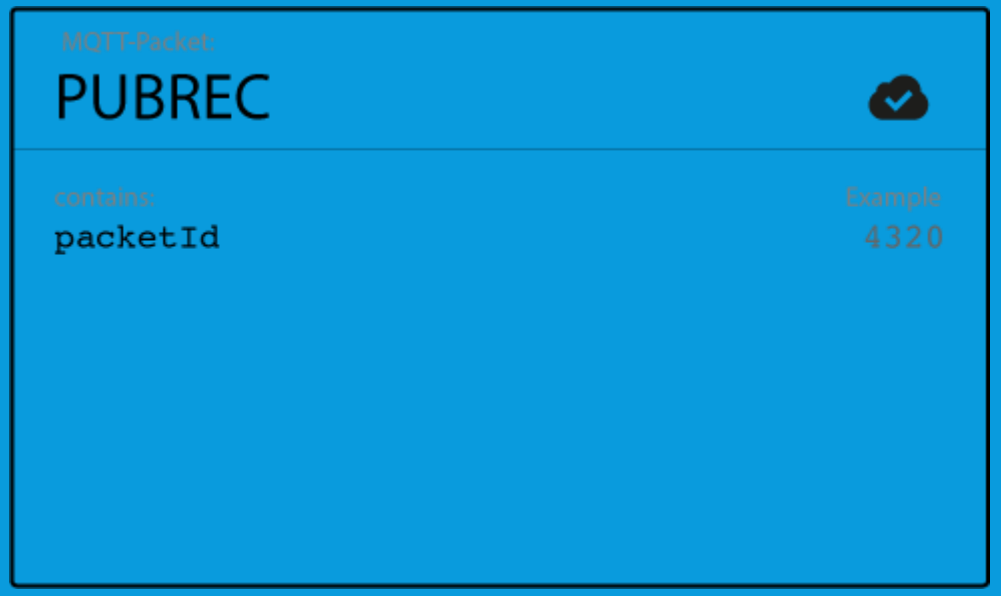
- QoS 2 is the highest level of service in MQTT. This level guarantees that each message is received only once by the intended recipients. **QoS 2 is the safest and slowest quality of service level**.
- The guarantee is provided by at least two request/response flows (a four-way handshake) between the sender and the receiver. The sender and receiver use the packet identifier of the original PUBLISH message to coordinate delivery of the message.

CONT.....



When a receiver gets a QoS 2 PUBLISH packet from a sender, it processes the publish message accordingly and replies to the sender with a **PUBREC** packet that acknowledges the PUBLISH packet. If the sender does not get a PUBREC packet from the receiver, it sends the PUBLISH packet again with a duplicate (DUP) flag until it receives an acknowledgement.

CONT.....

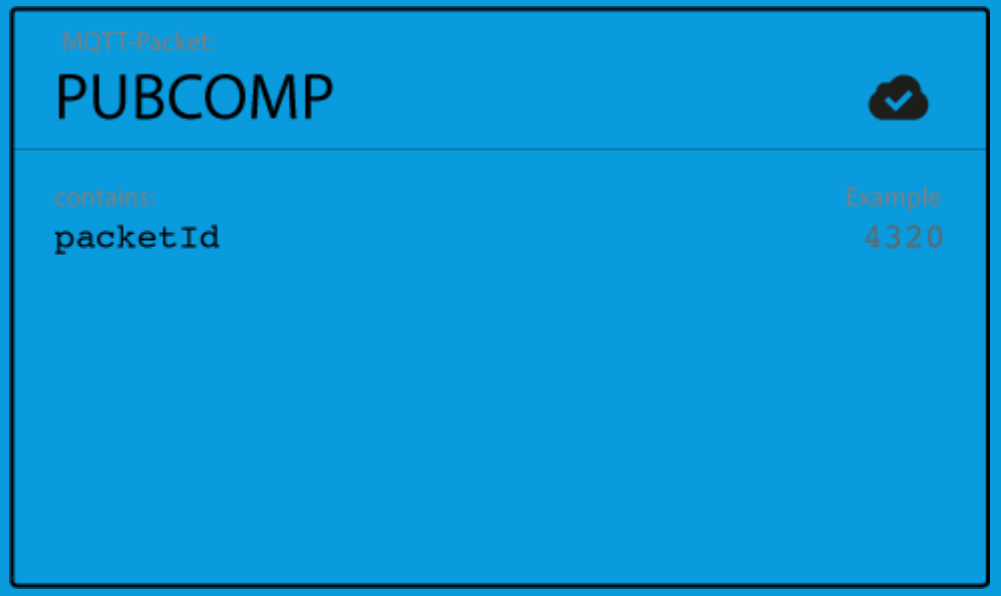


Once the sender receives a PUBREC packet from the receiver, the sender can safely discard the initial **PUBLISH** packet. The sender stores the PUBREC packet from the receiver and responds with a **PUBREL** packet.

CONT.....

After the receiver gets the PUBREL packet, it can discard all stored states and answer with a PUBCOMP packet (the same is true when the sender receives the PUBCOMP). Until the receiver completes processing and sends the PUBCOMP packet back to the sender, the receiver stores a reference to the packet identifier of the original PUBLISH packet. This step is important to avoid processing the message a second time.

CONT....



POINTS TO REMEMBER:

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

QOS – 1 is provide faster data transmission than QOS- 2

After the sender receives the PUBCOMP packet, the packet identifier of the published message becomes available for reuse.

ALWAYS REMEMBER

- Packet identifiers are unique per client
- The packet identifier that MQTT uses for QoS 1 and QoS 2 is unique between a specific client and a broker within an interaction. This identifier is not unique between all clients. Once the flow is complete, the packet identifier is available for reuse. This reuse is the reason why the **packet identifier does not need to exceed 65535**. It is unrealistic that a client can send more than this number of messages without completing an interaction.

MQTT CONTROL PACKET DESCRIPTION

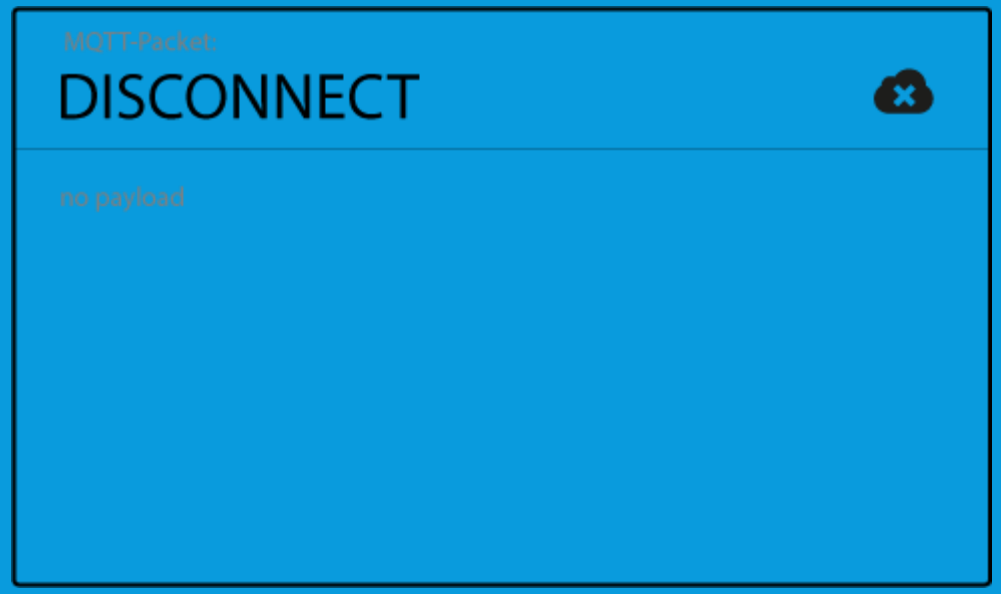
Control packet	Direction of flow	Description
CONNECT	Client to Server	Client request to connect to Server
CONNACK	Server to Client	Connect acknowledgment
PUBLISH	Client to Server or Server to Client	Publish message
PUBACK	Client to Server or Server to Client	Publish acknowledgment
PUBREC	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	Client to Server	Client subscribe request
SUBACK	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	Client to Server	Unsubscribe request
UNSUBACK	Server to Client	Unsubscribe acknowledgment
PINGREQ	Client to Server	PING request
PINGRESP	Server to Client	PING response
DISCONNECT	Client to Server	Client is disconnecting

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1

LAST WILL AND TESTAMENT

- A Last Will and Testament (LWT) message can be specified by an MQTT client when connecting to the MQTT broker. If that client does not disconnect gracefully, the broker sends out the LWT message on behalf of the client when connection loss is detected.

In response to the ungraceful disconnect, the broker sends the last-will message to all subscribed clients of the last-will message topic. If the client disconnects gracefully with a correct DISCONNECT message, the broker discards the stored LWT message



WHAT ARE THE SCENARIOS WHEN BROKER SEND LWT MESSAGE

- The broker detects an I/O error or network failure.
- The client fails to communicate within the defined Keep Alive period.
- The client does not send a DISCONNECT packet before it closes the network connection.
- The broker closes the network connection because of a protocol error.

RETAINED MESSAGE

- A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding QoS for that topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic.
- In other words, a retained message on a topic is the ***last known good value***. The retained message doesn't have to be the last value, but it must be the last message with the retained flag set to true
- **Retained messages help newly-subscribed clients get a status update immediately after they subscribe to a topic. The retained message eliminates the wait for the publishing clients to send the next update**

CLEAN/PERSISTENT SESSION

- When a client connects to an MQTT broker, it has the choice of requesting a persistent session. The broker is responsible for storing session information of the client if the client requested a persistent session. The session information of a client includes:
- All subscriptions of the client
- All QoS 1 / 2 messages which are not processed yet
- All QoS 1 / 2 messages the client missed while being offline

CONT..

- People often ask how long the broker stores the session.

The easy answer is:
The broker stores the session until the clients comes back online and receives the message. However,
what happens if a client does not come back online for a long time?
Usually, the memory limit of the operating system is the primary constraint on message storage.
There is no standard answer for this scenario

MQTT KEEP ALIVE TIME (TO AVOID HALF OPEN CONNECTIONS)

- "The Keep Alive … is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other Control Packets, the Client MUST send a PINGREQ Packet."

MQTT SECURITY

- **Username / Password Authentication**
- **Transport Security: TLS**

LISTING OF MQTT BROKERS

1. Mosquitto
2. HiveMQ
3. Mosca

REFERENCES

- HiveMQ
- OASIS MQTT 3.1.1/5.0
- IoT for architects by Perry Lea

For any further queries feel free to write@ bhupendra.jmd@gmail.com