MQ – Telemetry Transport or Message Queue Telemetry Transport - MQTT

Messaging Protocol for IoT devices

MQTT - Chapter Contents

- What is MQTT
- MQTT Architecture
- MQTT Client Operations
- MQTT QoS
- MQTT Topics
- MQTT Header and Payload
- MQTT Messages
- MQTT Contiki APIs

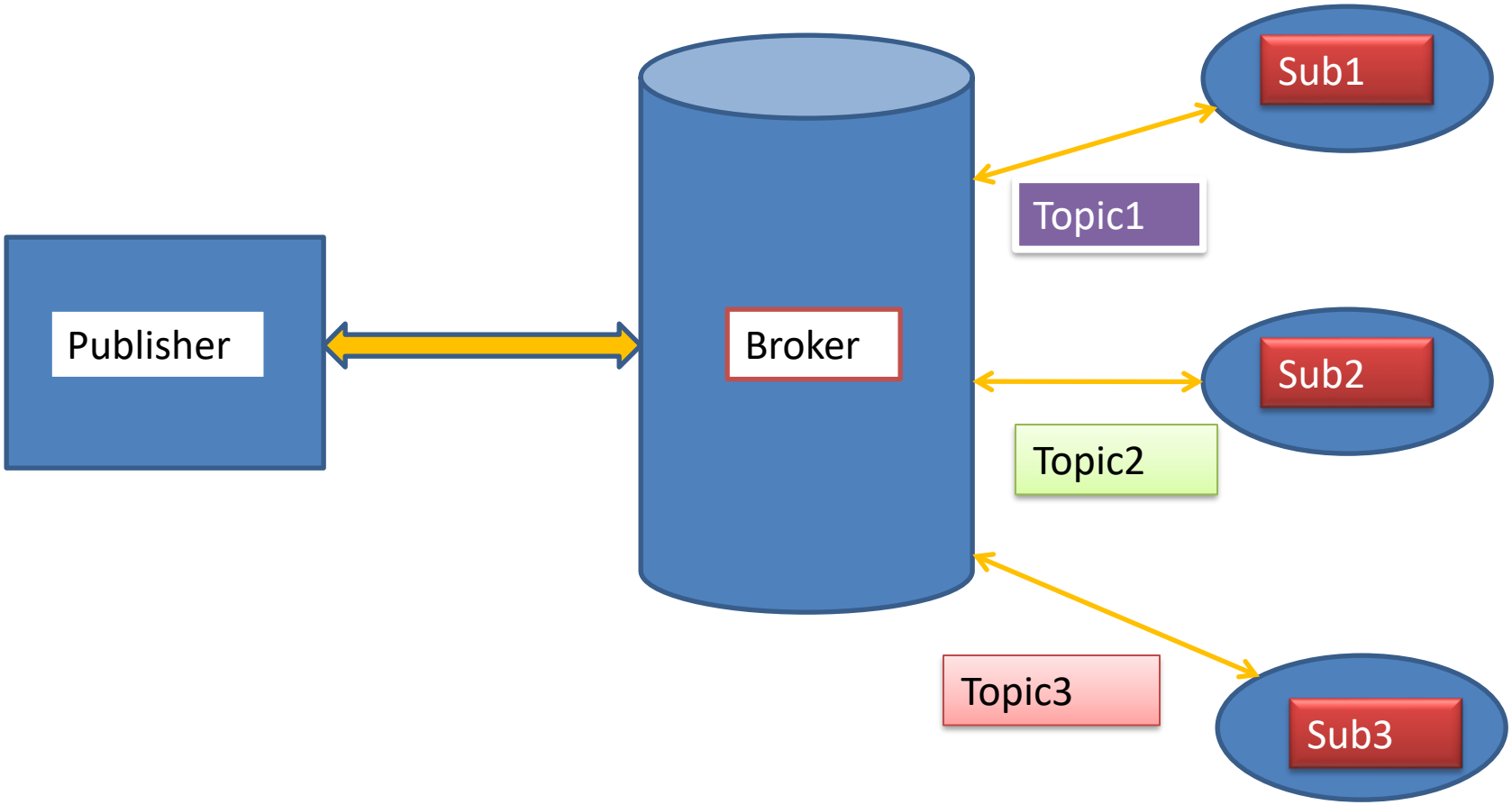
What is MQTT

- Protocol runs over TCP/IP
- Uses Publish Subscribe messaging model
- Message broker distributes topics with clients
- topics are UTF-8 string based, with hierarchical structure
- Decouples clients
- Three quality of service defined for data delivery

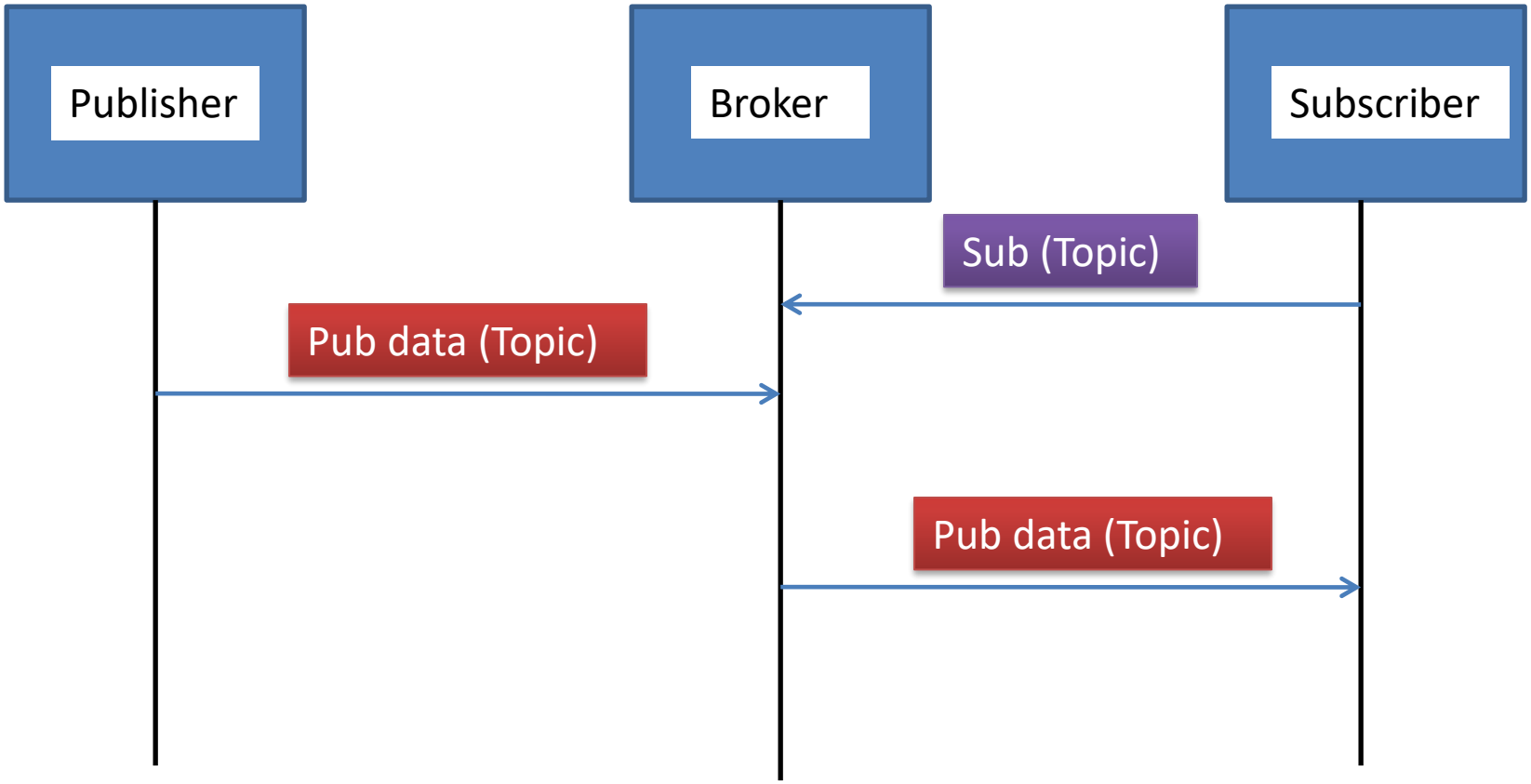
What is MQTT contd.

- Binary header and lightweight protocol
- A messaging transport that is agnostic to the content of payload
- Retain Flag – New subscribed clients shall receive last value
- Last Will – Notify other clients when disconnected ungracefully
- Keep Alive – Ping request message to broker

MQTT Publish – Subscribe Model



MQTT Architecture



Publishers and Subscribers are called Clients

MQTT Client Operations

Connect : Wait for the connection to be established with server/broker

Disconnect : Wait for the MQTT Client to finish any pending tasks and then closes the TCP connection

Subscribe: Requests the server/broker to subscribe the client to one or more topics

Unsubscribe : Requests the server/broker to unsubscribe the client on one or more topics

Publish : Client updates the server/broker with data on a topic

MQTT Specification

- MQTT protocol specifications defined by OASIS
- MQTT ver. 3.1.1 and MQTT ver. 5.0
- MQTT is application layer messaging protocol
- When an application uses MQTT protocol to communicate, it carries application payload, a quality of service (QoS), a collection of properties and a Topic Name
- Publishers and Subscribers are called clients and are interacting with broker also called as server
- One to many message distribution and helps in decoupling the applications

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Specification contd.

- Three QoS are defined for message delivery
- QoS 0 : “At most once” – Best effort delivery
- QoS 1 : “At least once” – guaranteed delivery, but duplicates can occur
- QoS 2 : “Exactly once” – guaranteed delivery, ensures that messages are delivered exactly once without duplication
- Small protocol overhead, message exchanges minimized to reduce network traffic

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Packet Header

Fixed header present in all MQTT Control Packets

Variable header present in some MQTT Control Packets

Payload present in some MQTT Control Packets

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Fixed Header – 2 Bytes

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Control Packet type - 16

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Control Packet type - 16

UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Control Packet Flag bits

MQTT 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 v5.0	DUP	QoS		RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0
AUTH	Reserved	0	0	0	0

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Header contd.

- Remaining length in the MQTT fixed header starting at 2nd byte is used to define number of bytes remaining within the current MQTT control packet that is inclusive of variable header & the payload.

- **Variable Header :** variable header component in the MQTT control packed is optional and is present in only some kind of control packets.

- **Payload :** Some MQTT control packets consists of payload for example PUBLISH

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Header contd.

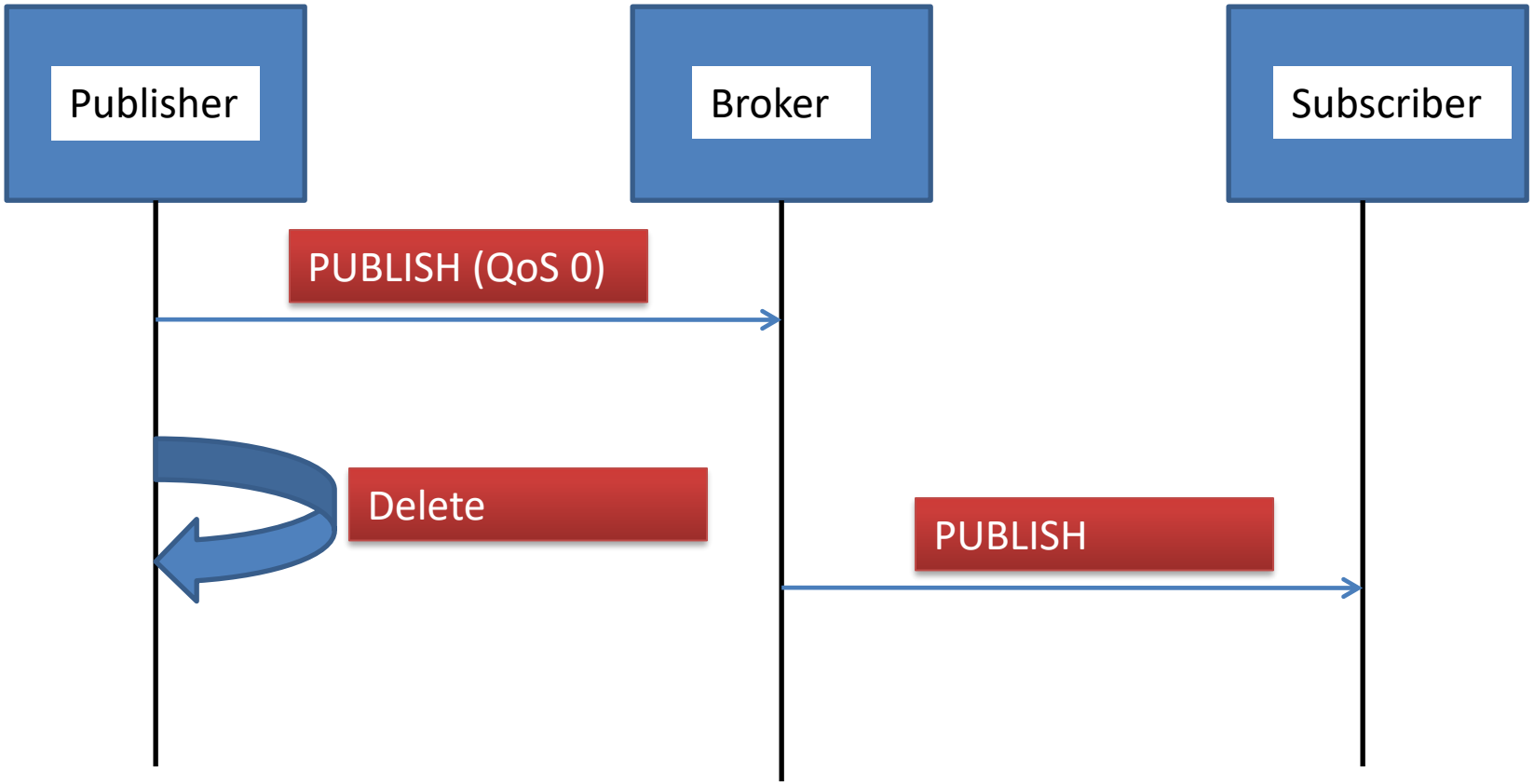
- Remaining length in the MQTT fixed header starting at 2nd byte is used to define number of bytes remaining within the current MQTT control packet that is inclusive of variable header & the payload.

- **Variable Header :** variable header component in the MQTT control packed is optional and is present in only some kind of control packets.

- **Payload :** Some MQTT control packets consists of payload for example PUBLISH.

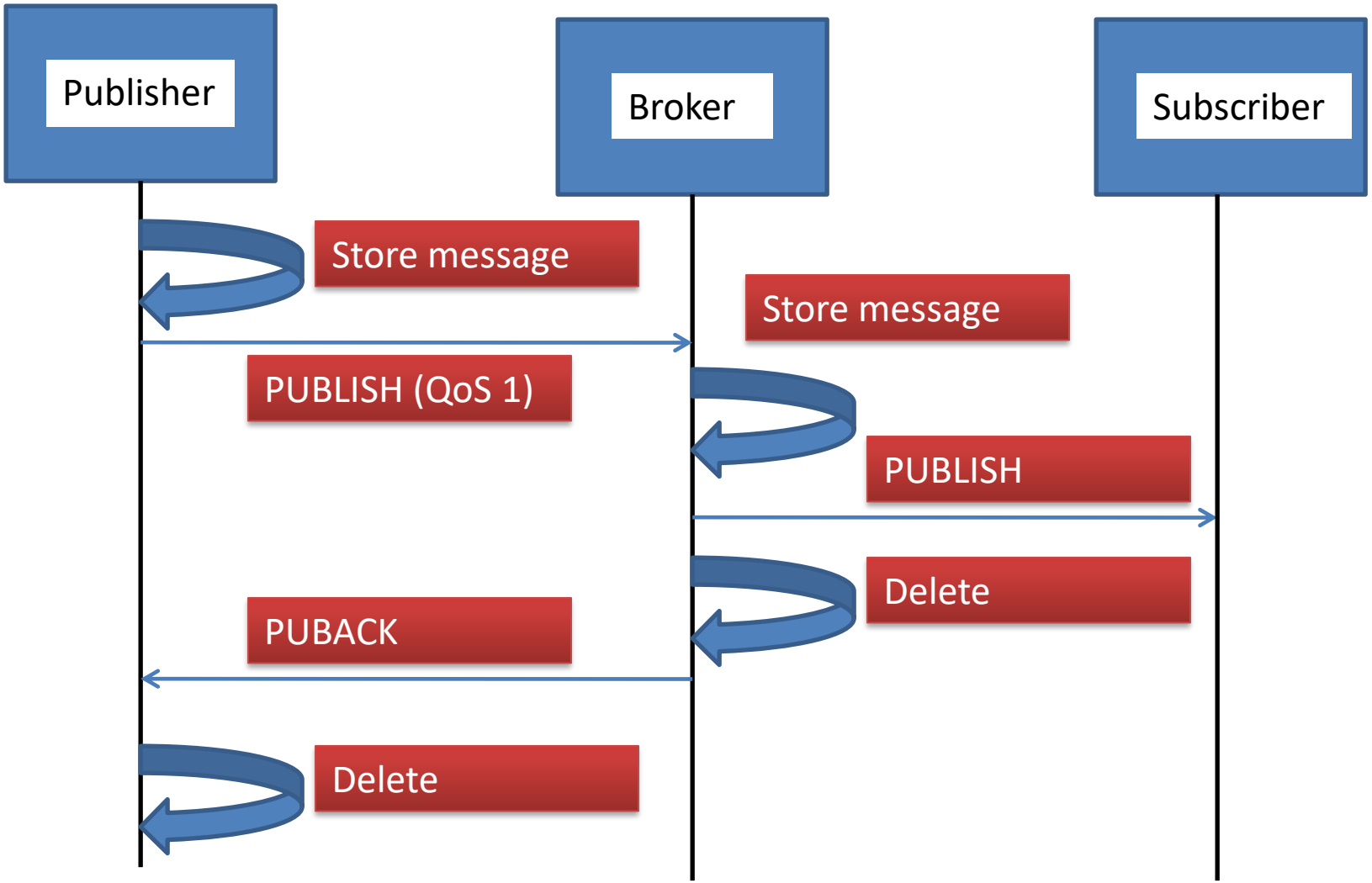
Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT QoS 0 : Fire and Forget



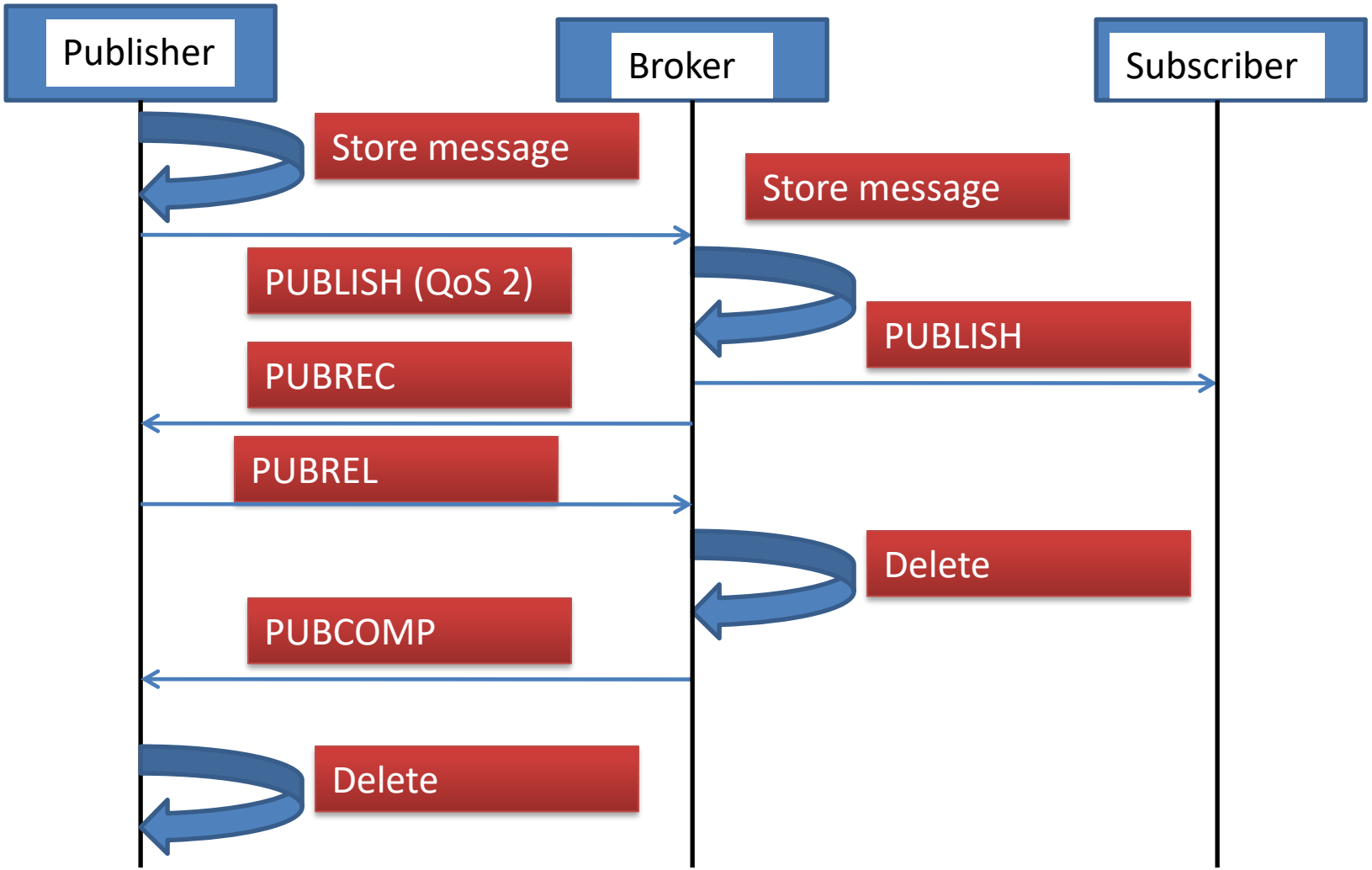
Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT QoS 1 : Atleast Once



Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT QoS 2 : Exactly Once



Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Topics

- Topics are UTF-8 encoded strings, topics can be a simple string or it can have multiple levels with level separators introducing hierarchical representation of topic.

- eg. : myoffice/groundfloor/lab1/temperature

- **Wildcards :** A subscription's Topic Filter can contain special wildcard characters, which allow a Client to subscribe to multiple topics at once

- eg. : myoffice/groundfloor/lab1/# --- **Multilevel Wildcard**

  myoffice/groundfloor/lab1/temperature
  myoffice/groundfloor/lab1/humidity
  myoffice/groundfloor/lab1/light

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Topics contd.

- **Single level Wildcards :** The plus sign “+” is a wildcard character that matches only one topic level

- eg. : myoffice/groundfloor/+/temperature

  myoffice/groundfloor/lab1/temperature
  myoffice/groundfloor/lab2/temperature
  myoffice/groundfloor/lab3/temperature

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT Topics contd.

- **Single level Wildcards :** Topics starting with $ are reserved for special operations like server/broker specific information

- eg. : $ SYS/broker/clients/connected
  $ SYS/broker/clients/disconnected
  $ SYS/broker/clients/total
  $ SYS/broker/clients/total
  $ SYS/broker/messages/sent
  $ SYS/broker/uptime

Ref : https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

MQTT APIs in Contiki-OS

MQTT is implemented in contiki in apps/mqtt

Current version of Contiki supports QoS – 0 and 1

A walk through with contiki APIs for MQTT applications

To start MQTT client this function should be called first

```
/**
 * \brief Initializes the MQTT engine.
 * \param conn A pointer to the MQTT connection.
 * \param app_process A pointer to the application process handling the MQTT
 *        connection.
 * \param client_id A pointer to the MQTT client ID.
 * \param event_callback Callback function responsible for handling the
 *        callback from MQTT engine.
 * \param max_segment_size The TCP segment size to use for this MQTT/TCP
 *        connection.
 * \return MQTT_STATUS_OK or MQTT_STATUS_INVALID_ARGS_ERROR
 *
 * This function initializes the MQTT engine and shall be called before any
 * other MQTT function.
 */
mqtt_status_t mqtt_register(struct mqtt_connection *conn,
                            struct process *app_process,
                            char *client_id,
                            mqtt_event_callback_t event_callback,
                            uint16_t max_segment_size);
```

Ref : https://github.com/contiki-ng/contiki-ng

This function connects to an MQTT broker

```
/**
 * \brief Connects to a MQTT broker.
 * \param conn A pointer to the MQTT connection.
 * \param host IP address of the broker to connect to.
 * \param port Port of the broker to connect to, default is MQTT port is 1883.
 * \param keep_alive Keep alive timer in seconds. Used by broker to handle
 *        client disc. Defines the maximum time interval between two messages
 *        from the client. Shall be min 1.5 x report interval.
 * \return MQTT_STATUS_OK or an error status
 *
 * This function connects to a MQTT broker.
 */
mqtt_status_t mqtt_connect(struct mqtt_connection *conn,
                           char *host,
                           uint16_t port,
                           uint16_t keep_alive);
/**
 * \brief Disconnects from a MQTT broker.
 * \param conn A pointer to the MQTT connection.
 *
 * This function disconnects from a MQTT broker.
 */
void mqtt_disconnect(struct mqtt_connection *conn);
```

Ref : https://github.com/contiki-ng/contiki-ng

MQTT Subscription

```
/**
 * \brief Subscribes to a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to subscribe to.
 * \param qos_level Quality Of Service level to use. Currently supports 0, 1.
 * \return MQTT_STATUS_OK or some error status
 *
 * This function subscribes to a topic on a MQTT broker.
 */
mqtt_status_t mqtt_subscribe(struct mqtt_connection *conn,
                             uint16_t *mid,
                             char *topic,
                             mqtt_qos_level_t qos_level);
```

Message ID mid = 0 for QoS 0

```
/**
 * \brief Unsubscribes from a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to unsubscribe from.
 * \return MQTT_STATUS_OK or some error status
 *
 * This function unsubscribes from a topic on a MQTT broker.
 */
mqtt_status_t mqtt_unsubscribe(struct mqtt_connection *conn,
                               uint16_t *mid,
                               char *topic);
```

Ref : https://github.com/contiki-ng/contiki-ng

MQTT Publication

```
/**
 * \brief Publish to a MQTT topic.
 * \param conn A pointer to the MQTT connection.
 * \param mid A pointer to message ID.
 * \param topic A pointer to the topic to subscribe to.
 * \param payload A pointer to the topic payload.
 * \param payload_size Payload size.
 * \param qos_level Quality Of Service level to use. Currently supports 0, 1.
 * \param retain If the RETAIN flag is set to 1, in a PUBLISH Packet sent by a
 *        Client to a Server, the Server MUST store the Application Message
 *        and its QoS, so that it can be delivered to future subscribers whose
 *        subscriptions match its topic name
 * \return MQTT_STATUS_OK or some error status
 *
 * This function publishes to a topic on a MQTT broker.
 */
mqtt_status_t mqtt_publish(struct mqtt_connection *conn,
                           uint16_t *mid,
                           char *topic,
                           uint8_t *payload,
                           uint32_t payload_size,
                           mqtt_qos_level_t qos_level,
                           mqtt_retain_t retain);
```

Ref : https://github.com/contiki-ng/contiki-ng

MQTT Authorizing and Last Will

```
/**
 * \brief Set the user name and password for a MQTT client.
 * \param conn A pointer to the MQTT connection.
 * \param username A pointer to the user name.
 * \param password A pointer to the password.
 *
 * This function sets clients user name and password to use when connecting t
 * a MQTT broker.
 */
void mqtt_set_username_password(struct mqtt_connection *conn,
                                char *username,
                                char *password);

/**
 * \brief Set the last will topic and message for a MQTT client.
 * \param conn A pointer to the MQTT connection.
 * \param topic A pointer to the Last Will topic.
 * \param message A pointer to the Last Will message (payload).
 * \param qos The desired QoS level.
 *
 * This function sets clients Last Will topic and message (payload).
 * If the Will Flag is set to 1 (using the function) this indicates that,
 * if the Connect request is accepted, a Will Message MUST be stored on the
 * Server and associated with the Network Connection. The Will Message MUST
 * be published when the Network Connection is subsequently closed.
 *
 * This functionality can be used to get notified that a device has
 * disconnected from the broker.
 *
 */
void mqtt_set_last_will(struct mqtt_connection *conn,
                        char *topic,
                        char *message,
                        mqtt_qos_level_t qos);
```

MQTT Client App