MQTT

IoT Protocols MQTT

Stefan Forsström
Institution of Information Systems and Technology

Overview

- History and Background
- Structure and Features
- MQTT Messages Details
- Tips for the Lab Project

History

- Message Queue Telemetry Transport (MQTT)
  - Was invented in 1999 by
    - Andy Stanford-Clark (IBM)
    - Arlen Nipper (Arcom, now Cirrus Link)
- The use case was to create a protocol for minimal battery loss and minimal bandwidth, to connect oil pipelines over satellite Internet
- They specified the following goals of the protocol:
  - Simple to implement
  - Provide a Quality of Service Data Delivery
  - Lightweight and Bandwidth Efficient
  - Data Agnostic
  - Continuous Session Awareness

History

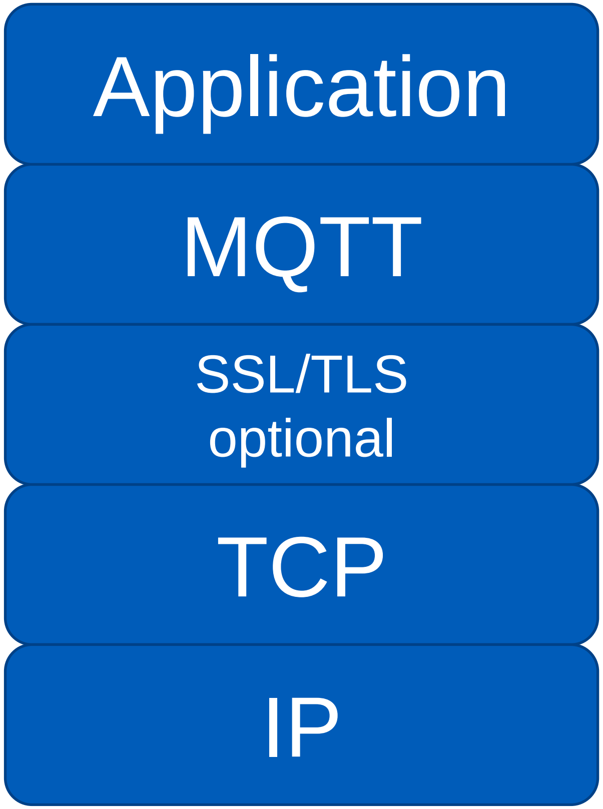
- It is an publish-subscribe "lightweight" messaging protocol
  - But it fits the IoT very well
- Highly centralized with a coordinating broker server
  - Consumers subscribe to topics
  - Which producers can publish to
  - And the broker unicasts the data to the subscribers
- Not really “lightweight”, since it is based on TCP
  - Comparable to REST, about the same
  - But has built in publish subscribe features

Structure

TCP/IP Port: 1883

When running over SSL/TLS port: 8883

When running over Websockets port: 8000



Application

MQTT

SSL/TLS
optional

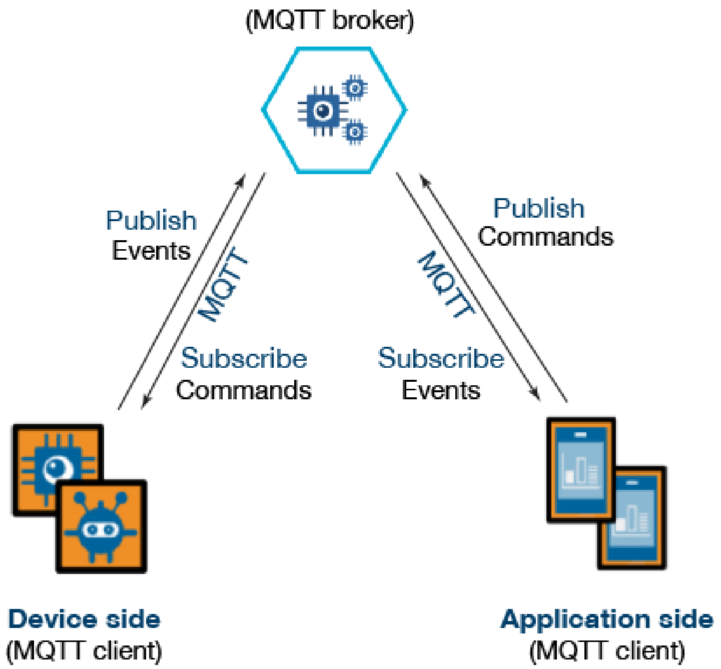
TCP

IP

Mittuniversitetet
MID SWEDEN UNIVERSITY

Structure

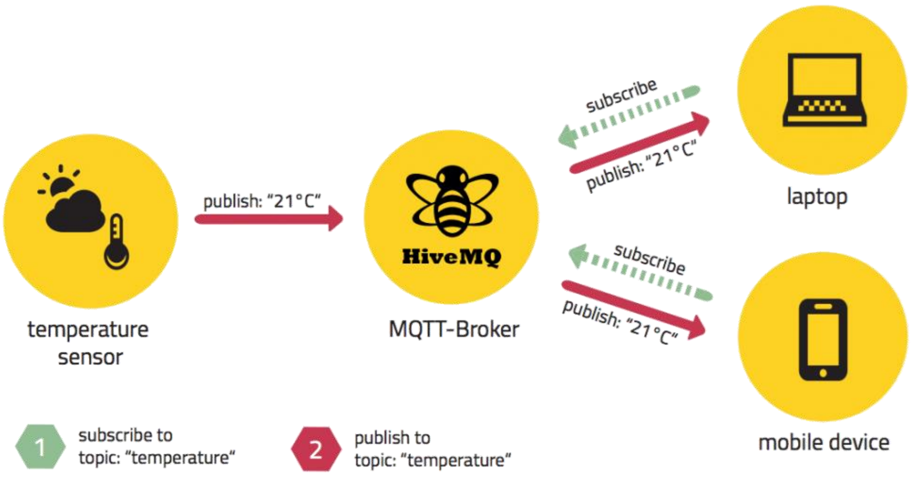
- MQTT consist of three parts:
  - Broker
  - Subscribers
  - Publishers



- Clients connect to a “Broker”
- Clients subscribe to topics e.g.,
- Clients can publish messages to topics:
  - All clients receive all messages published to topics they subscribe to
- Messages can be anything, Text, Images, etc.

Publish/Subscribe Concept

- Decoupled in space and time:
  - The clients does not need to know each others IP address and port and they do not need to be running at the same time

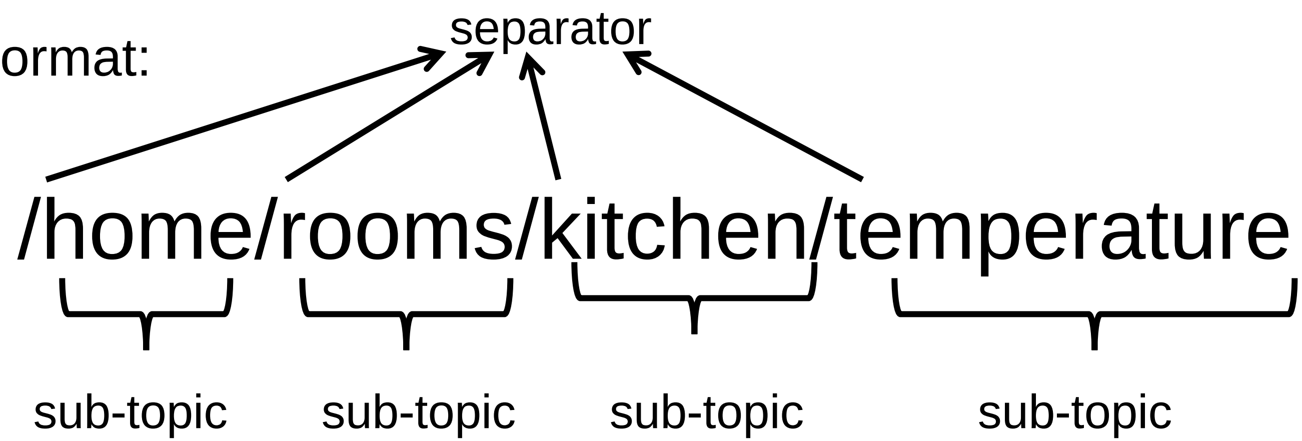


- The broker's IP and port must however be known by all clients

Topics

- Each published data specifies a topic
  - Each subscriber subscribed to that topic will receive it
  - Namespace hierarchy is used for topic filtering

- Topic format:



separator

/home/rooms/kitchen/temperature

sub-topic sub-topic sub-topic sub-topic

Subscriptions

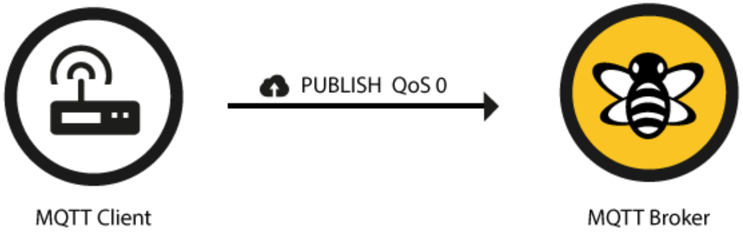
- Subscription types
  - Durable
    - If the subscriber disconnect messages are buffered at the broker and delivered upon reconnection
  - Non-durable
    - Connection lifetime is the subscription lifetime
    - When the TCP session breaks down, the subscription is closed

Publishing

- It might also be the case that a published message never becomes consumed by any subscriber
  - The clients are unaware of the number of subscribers
- Message retention
  - Retained (a type of “persistent” message)
    - The subscriber upon first connection receives the last good publication (i.e., does not have to wait for new publication)
    - Only the most recent persistent message is stored and distributed
- Last Will and Testament (LWT)
  - A message published upon disconnecting a connection
  - Anybody subscribing to the LWT topic will know when a certain device (that registered a LWT) disconnected

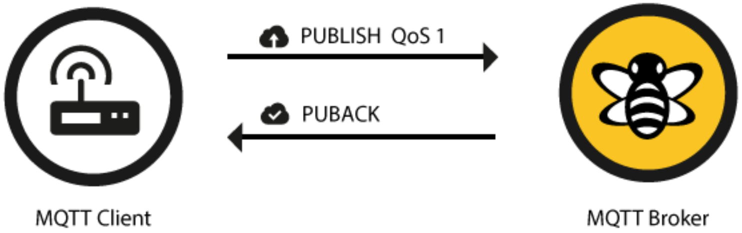
Publishing “QoS” (Reliability)

- 0 – unreliable (aka “at most once”)
  - OK for continuous streams, least overhead (1 message)
  - “Fire and forget”
  - TCP will still provide reliability



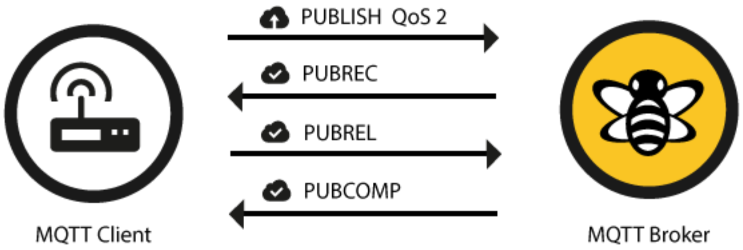
Publishing “QoS” (Reliability)

- 1 – delivery “at least once” (duplicates possible)
  - Used for alarms – more overhead (2 messages)
  - Contains message ID (to match with ACKed message)



Publishing “QoS” (Reliability)

- 2 – delivery “exactly once”
  - Utmost reliability is important – most overhead (4 messages) and slowest



Security?

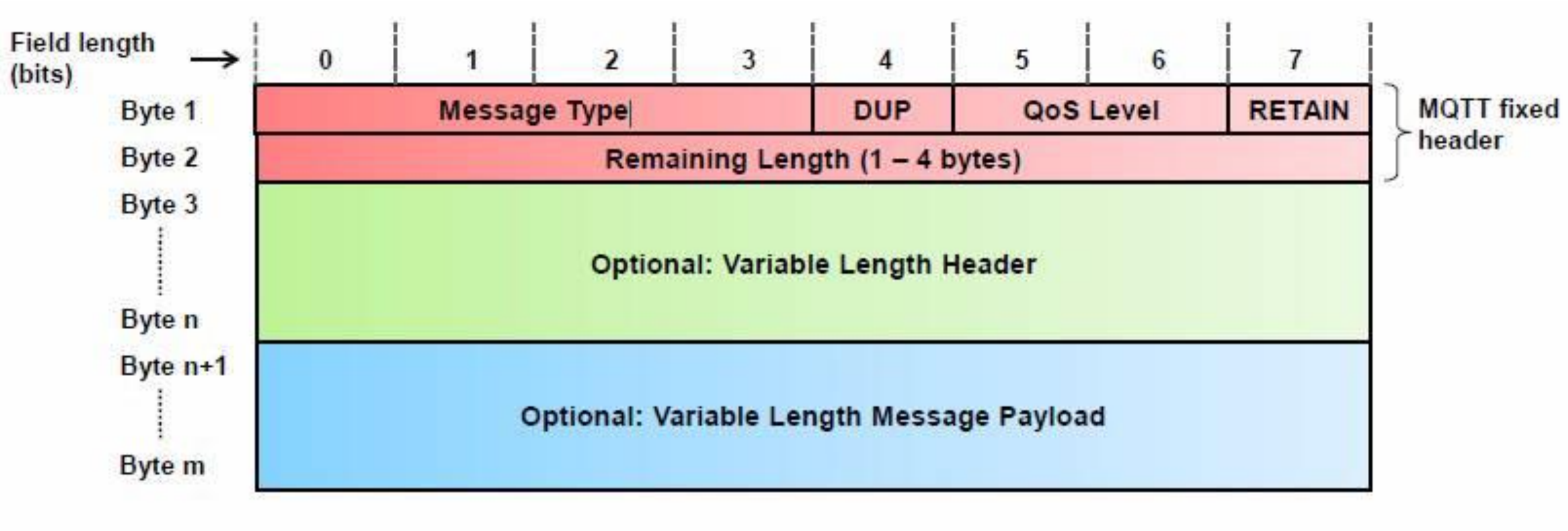
- All communication is done in clear text
  - Unless SSL/TLS is used

- There is a simple Client ID method used for recognizing users
- But there are also functions for username/password authentication
  - For private accessing the broker etc.

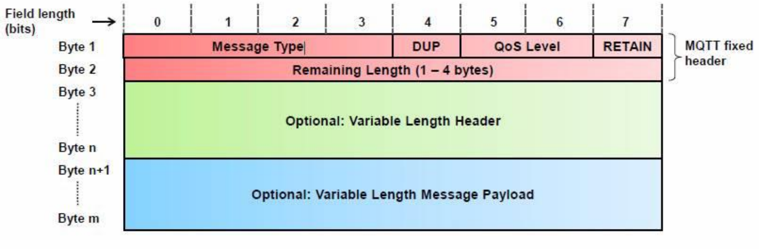
- You can also control which clients are able to subscribe and publish to different topics
  - Using either the ClientID or username/password

MQTT Messages

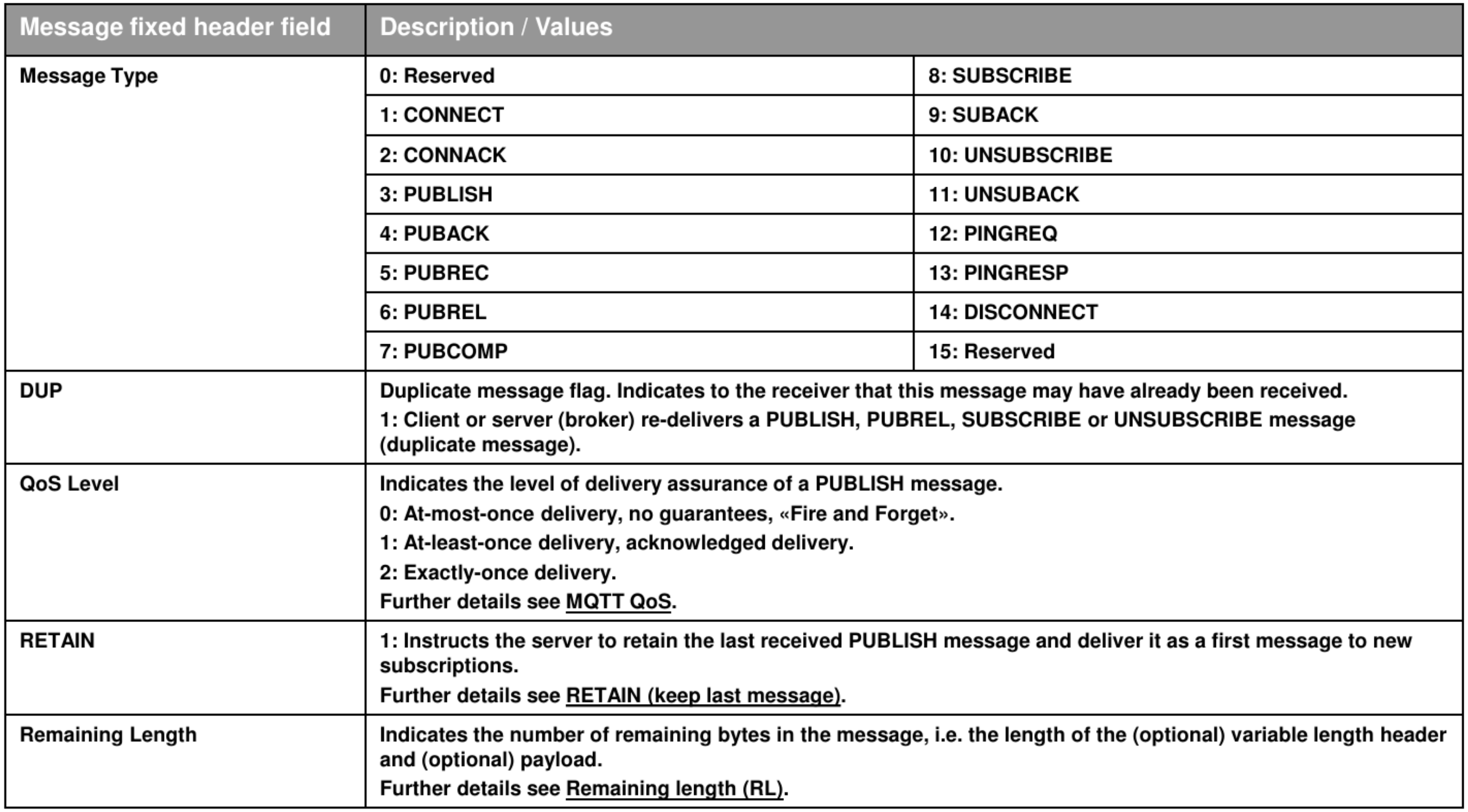
MQTT Message Format



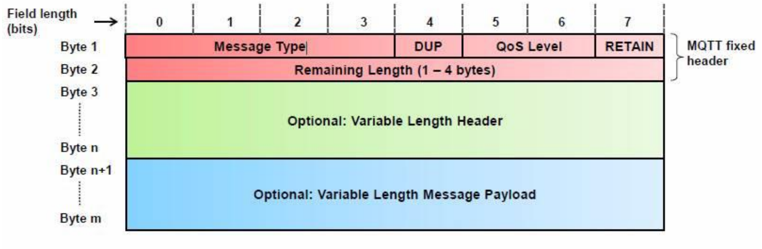
- Shortest message is two bytes (red fields)



Message Types



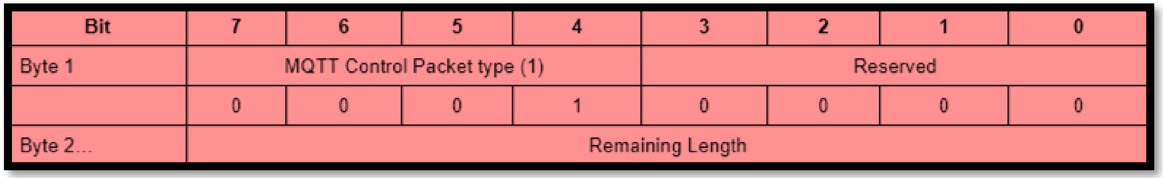
Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	



Field length (bits)	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 Byte n	Optional: Variable Length Header								
Byte n+1 Byte m	Optional: Variable Length Message Payload								

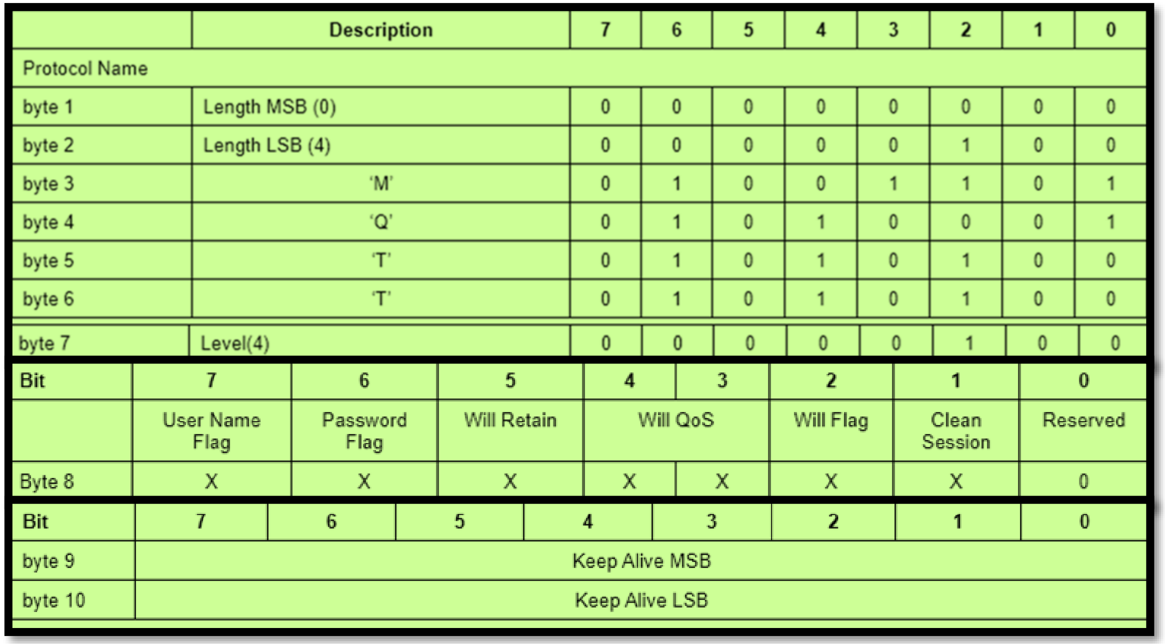
Connect Message

- Message nr 1



Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
Byte 2...	Remaining Length							

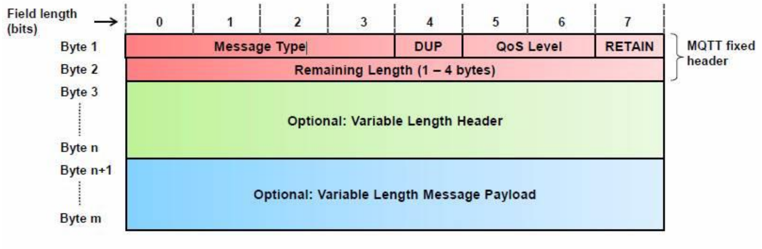
- Protocol name
- Flags
- Keep alive
- Payload
  - can include client ID length and name



	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0
byte 7	Level(4)	0	0	0	0	0	1	0	0

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
Byte 8	X	X	X	X	X	X	X	0

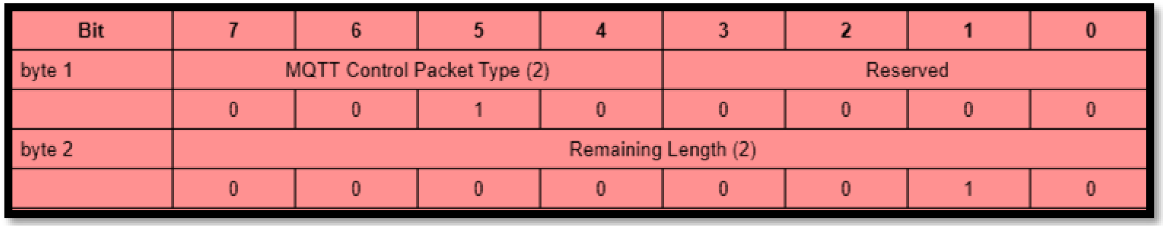
Bit	7	6	5	4	3	2	1	0
byte 9	Keep Alive MSB							
byte 10	Keep Alive LSB							



Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 ⋮ Byte n	Optional: Variable Length Header								
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload								

Connect Ack Message

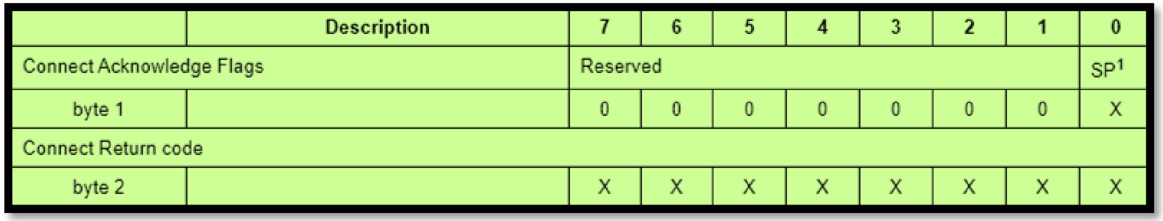
- Message nr 2



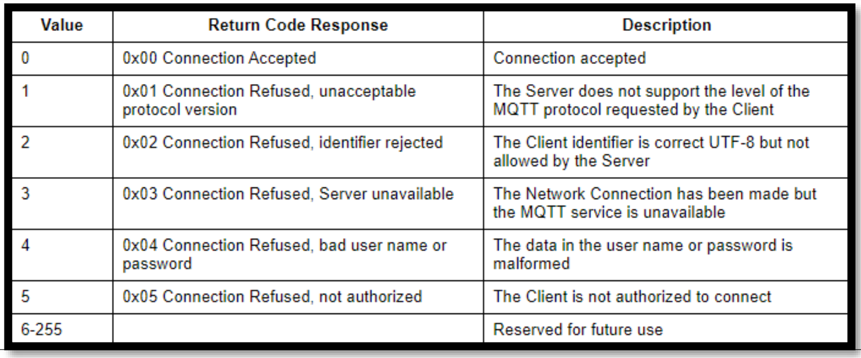
Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet Type (2)				Reserved			
	0	0	1	0	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

- SP: Session Present Bit

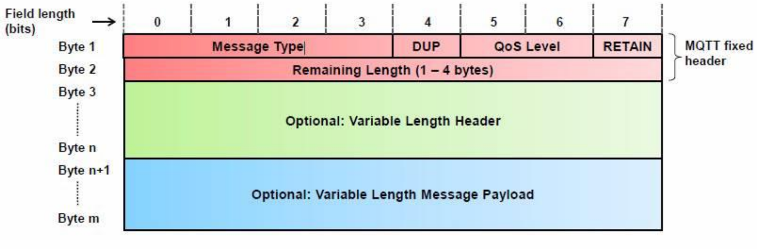
- Return code



	Description	7	6	5	4	3	2	1	0
Connect Acknowledge Flags		Reserved							SP[1]
byte 1		0	0	0	0	0	0	0	X
Connect Return code									
byte 2		X	X	X	X	X	X	X	X



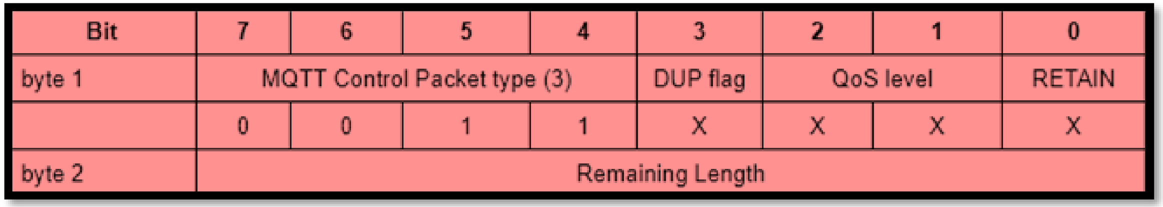
Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct UTF-8 but not allowed by the Server
3	0x03 Connection Refused, Server unavailable	The Network Connection has been made but the MQTT service is unavailable
4	0x04 Connection Refused, bad user name or password	The data in the user name or password is malformed
5	0x05 Connection Refused, not authorized	The Client is not authorized to connect
6-255		Reserved for future use



Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 ⋮ Byte n	Optional: Variable Length Header								
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload								

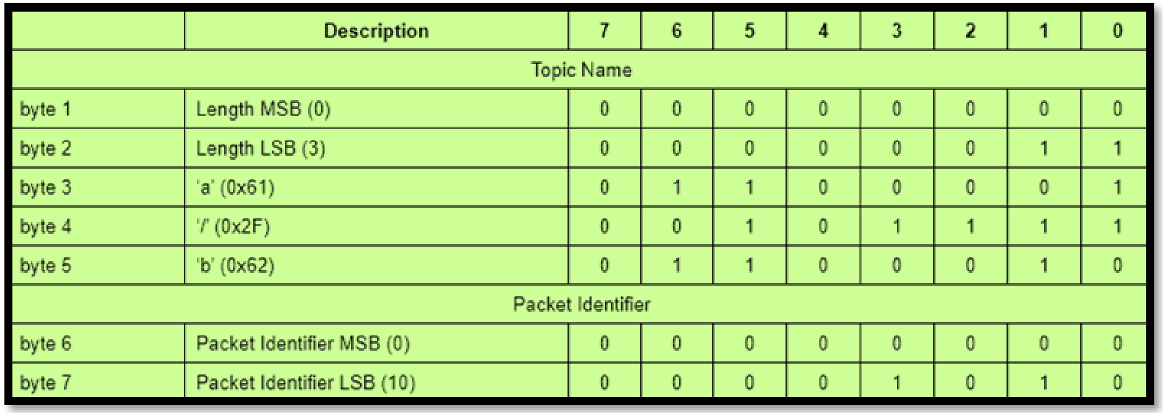
Publish Message

- Message nr 3

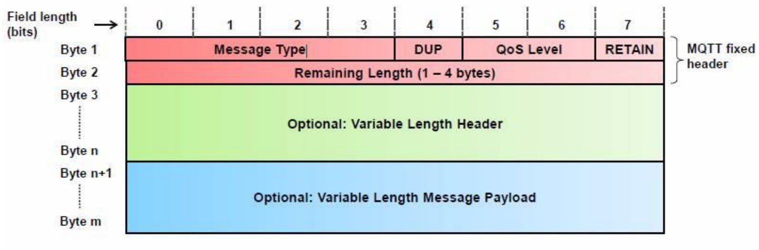


Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2	Remaining Length							

- Topic Length
- The topic characters
- Packet ID

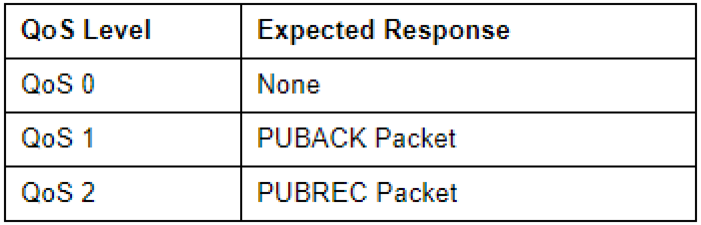


	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Packet Identifier									
byte 6	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 7	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0



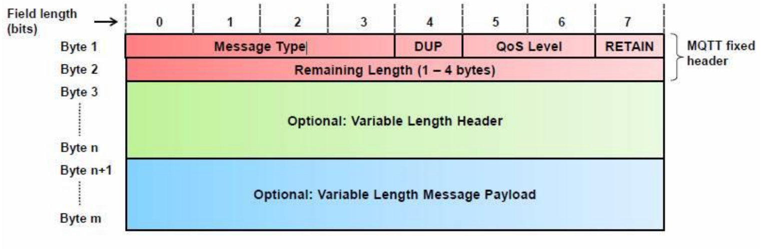
Publish Message

- The QoS level specifies the answer



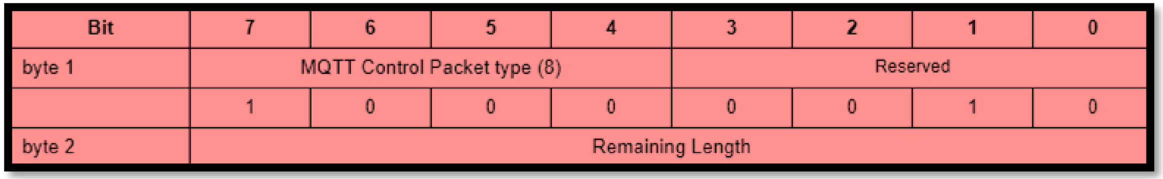
QoS Level	Expected Response
QoS 0	None
QoS 1	PUBACK Packet
QoS 2	PUBREC Packet

- I leave these for self study:
  - PUBACK – Publish acknowledgemen (QoS 1)
  - PUBREC – Publish received (QoS 2 publish received, part 1)
  - PUBREL – Publish release (QoS 2 publish received, part 2)
  - PUBCOMP – Publish complete (QoS 2 publish received, part 3)



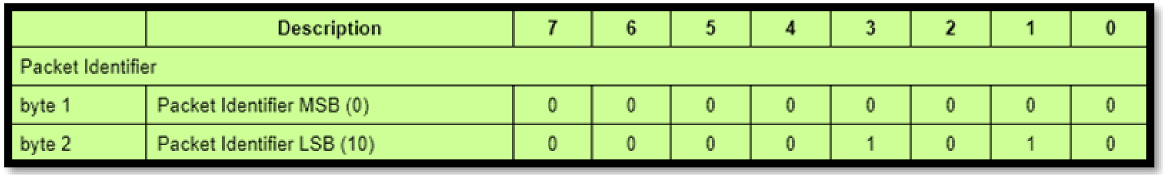
Subscribe Message

- Message nr 8



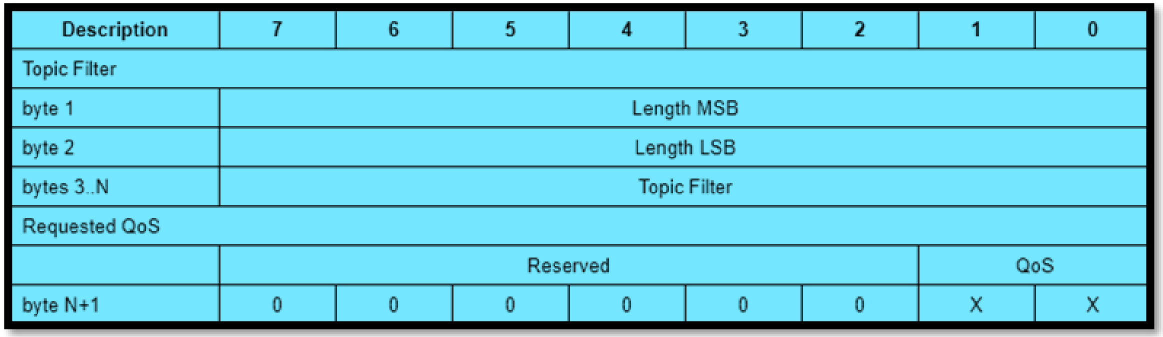
Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
byte 2	Remaining Length							

- Package ID

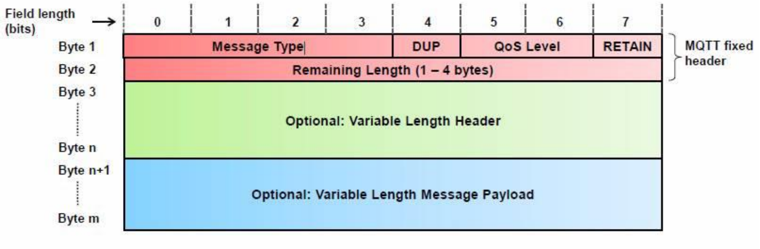


	Description	7	6	5	4	3	2	1	0
Packet Identifier									
byte 1	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 2	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0

- Payload
  - Can have many subs in one message



Description	7	6	5	4	3	2	1	0
Topic Filter								
byte 1	Length MSB							
byte 2	Length LSB							
bytes 3..N	Topic Filter							
Requested QoS								
	Reserved						QoS	
byte N+1	0	0	0	0	0	0	X	X



Subscribe Ack Message

- Message nr 9

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (9)				Reserved			
	1	0	0	1	0	0	0	0
byte 2	Remaining Length							

- Packet ID to ACK

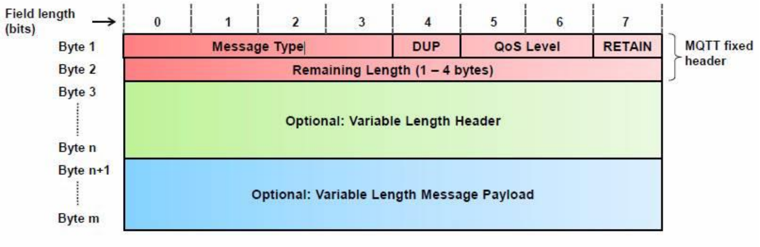
Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

- Return code

Bit	7	6	5	4	3	2	1	0
	Return Code							
byte 1	X	0	0	0	0	0	X	X

Allowed return codes:

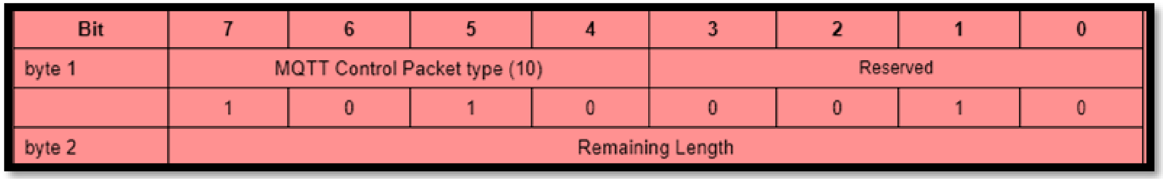
0x00 - Success - Maximum QoS 0
0x01 - Success - Maximum QoS 1
0x02 - Success - Maximum QoS 2
0x80 - Failure



Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 Byte n	Optional: Variable Length Header								
Byte n+1 Byte m	Optional: Variable Length Message Payload								

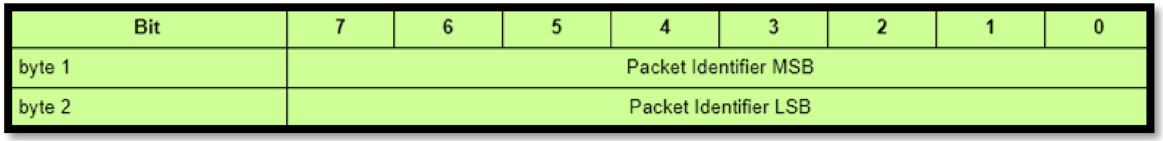
Unsubscribe Message

- Message nr 10



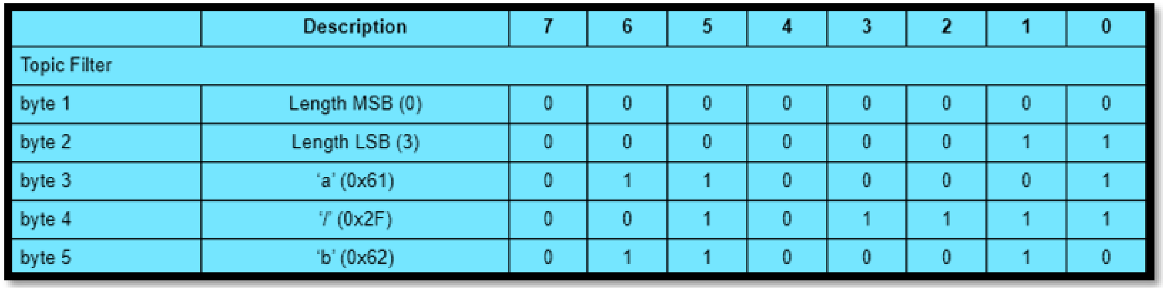
Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (10)				Reserved			
	1	0	1	0	0	0	1	0
byte 2	Remaining Length							

- Packet ID

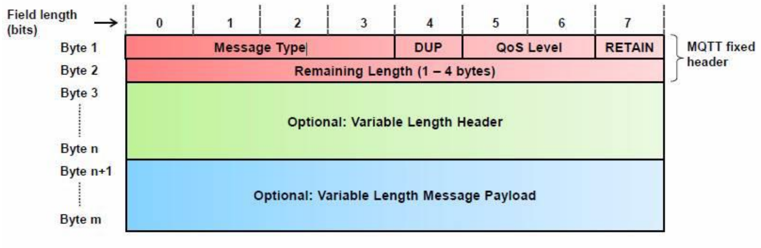


Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

- Payload
  - Can have many unsub in one message

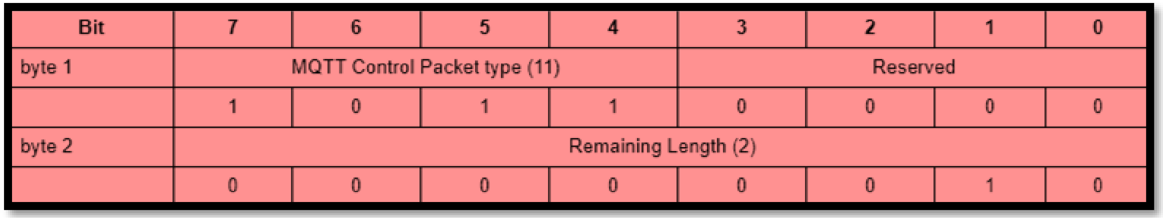


	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	‘a’ (0x61)	0	1	1	0	0	0	0	1
byte 4	‘/’ (0x2F)	0	0	1	0	1	1	1	1
byte 5	‘b’ (0x62)	0	1	1	0	0	0	1	0



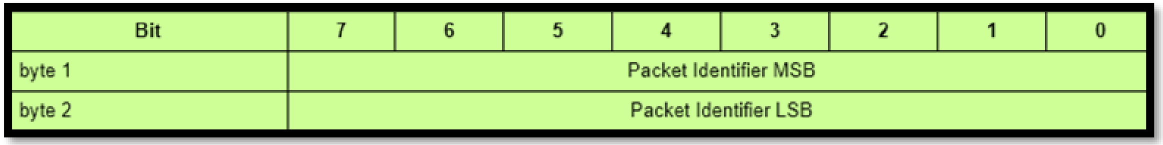
Unsubscribe Ack Message

- Message nr 11

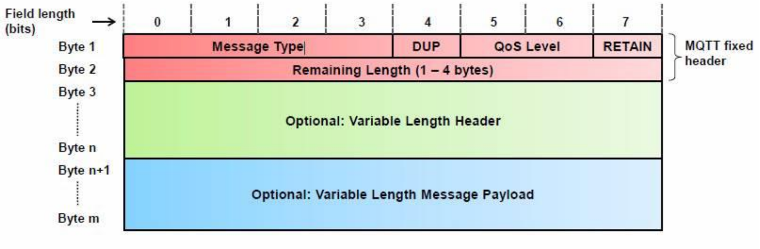


Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (11)				Reserved			
	1	0	1	1	0	0	0	0
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

- Packet ID to ACK

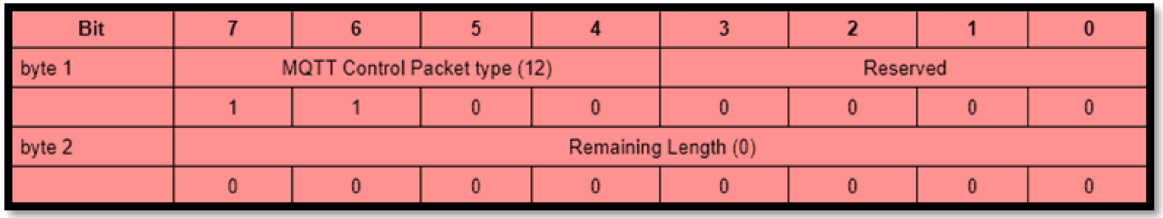


Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							



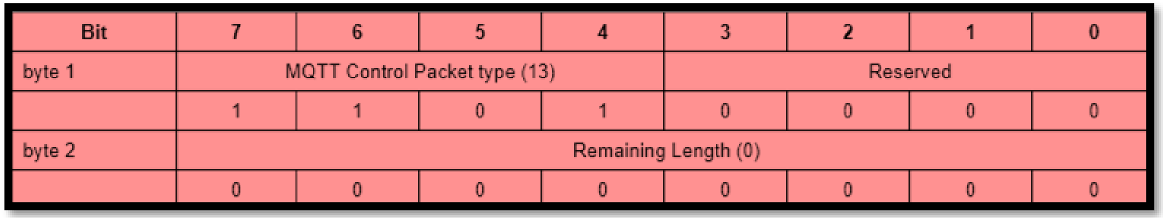
PingReq/PingResp Message

- Ping Request

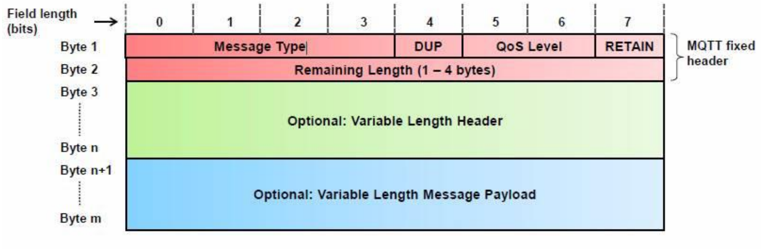


Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (12)				Reserved			
	1	1	0	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

- Ping Response



Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (13)				Reserved			
	1	1	0	1	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0



Field length (bits) →

0 1 2 3 4 5 6 7

Byte 1: Message Type | DUP | QoS Level | RETAIN

Byte 2: Remaining Length (1 – 4 bytes)

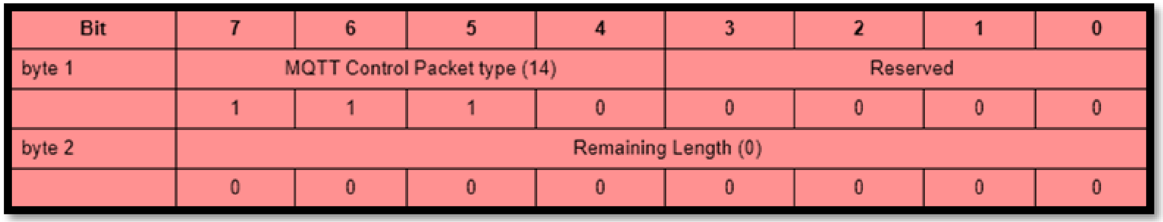
MQTT fixed header

Byte 3 … Byte n: Optional: Variable Length Header

Byte n+1 … Byte m: Optional: Variable Length Message Payload

Disconnect Message

- Disconnect



Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (14)				Reserved			
	1	1	1	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

V3.1.1 and V.5 and MQTT-SN

- MQTT version 3.1.1 was the standard for a long time
  - But has now been updated to v.5 (published 2019)
- In general it functions the same, but some changes to the protocol
  - Clean session/start
  - Client Restrictions/Limitations
  - Server Restrictions/Limitations
  - Will Delay Intervals
  - Server Redirect
  - Payload Format Indicator
  - Topic aliases
  - User Properties
  - Request Response
  - Non local publishing
  - Retained Message Control
  - Subscription Identifier
  - Shared Subscriptions
  - Reason Codes on All ACK Messages
  - Server Disconnect
- MQTT for Sensor Networks (MQTT-SN)
  - A more limited version of MQTT over UDP (not updated since 2013)

Tips for the Lab Project

Tips for the Lab Project

- Start by using a normal MQTT Client, connect to an MQTT broker
  - For example broker.mqttdashboard.com
  - Observe the messages in Wireshark (tcp.port == 1883)
- Then create your own TCP socket listening on port 1883
  - And send your MQTT client to it instead
  - Observe, listening, and answer to incoming packets
- Since you are the broker, most answers will be short ACKs
  - The Connection ACK is only 4 bytes, for example:
    - Byte 1: (0010 0000) Packet Type 2, not DUP, QoS 0, no RETAIN
    - Byte 2: (0000 0010) 2 remaining bytes
    - Byte 3: (0000 0000) No session present
    - Byte 4: (0000 0000) return code 0, success

Tips for the Lab Project

- Start by answering the connect packages
  - And then make your program answer the ping packets
  - Otherwise all your clients will time out after a while
- If you receive a subscribe, save that socket and topic to a list/map
  - When you receive a publish, send it to all sockets on the topic list
  - Unsubscribe removes socket from the topic list
- Wireshark is your friend
  - To see how other MQTT clients messages looks like is very good for you to learn, debug and compare to yourself to them
- There is no delimiter of payload or between messages
  - Make sure you have the right remainder packet length
  - It is the only way for the system to distinguish between two messages

Tips for the Lab Project

- The MQTT 3.1.1 Oasis Standard
  - http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

- A good online reference:
  - http://www.steves-internet-guide.com/mqtt-basics-course/

Contact Information

STEFAN FORSSTRÖM
PhD in Computer and System Sciences

MID SWEDEN UNIVERSITY
Institution of Information Systems and Technology (IST)
Campus Sundsvall, Room L426
Email: stefan.forsstrom@miun.se