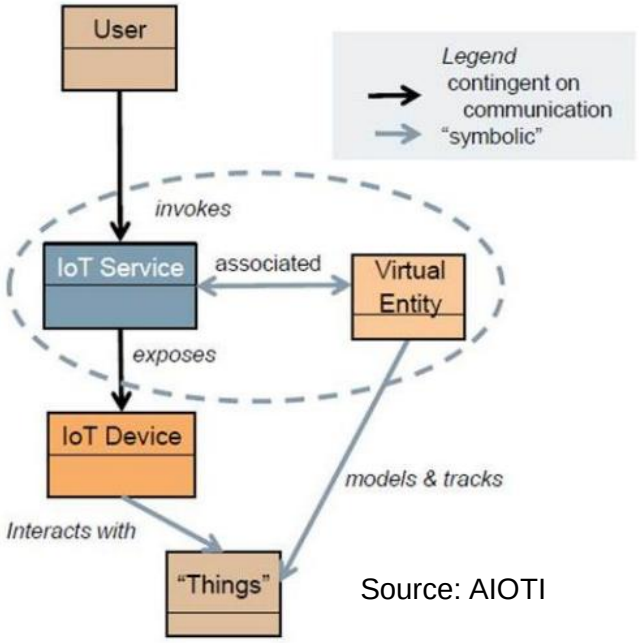
EIOT

MQTT

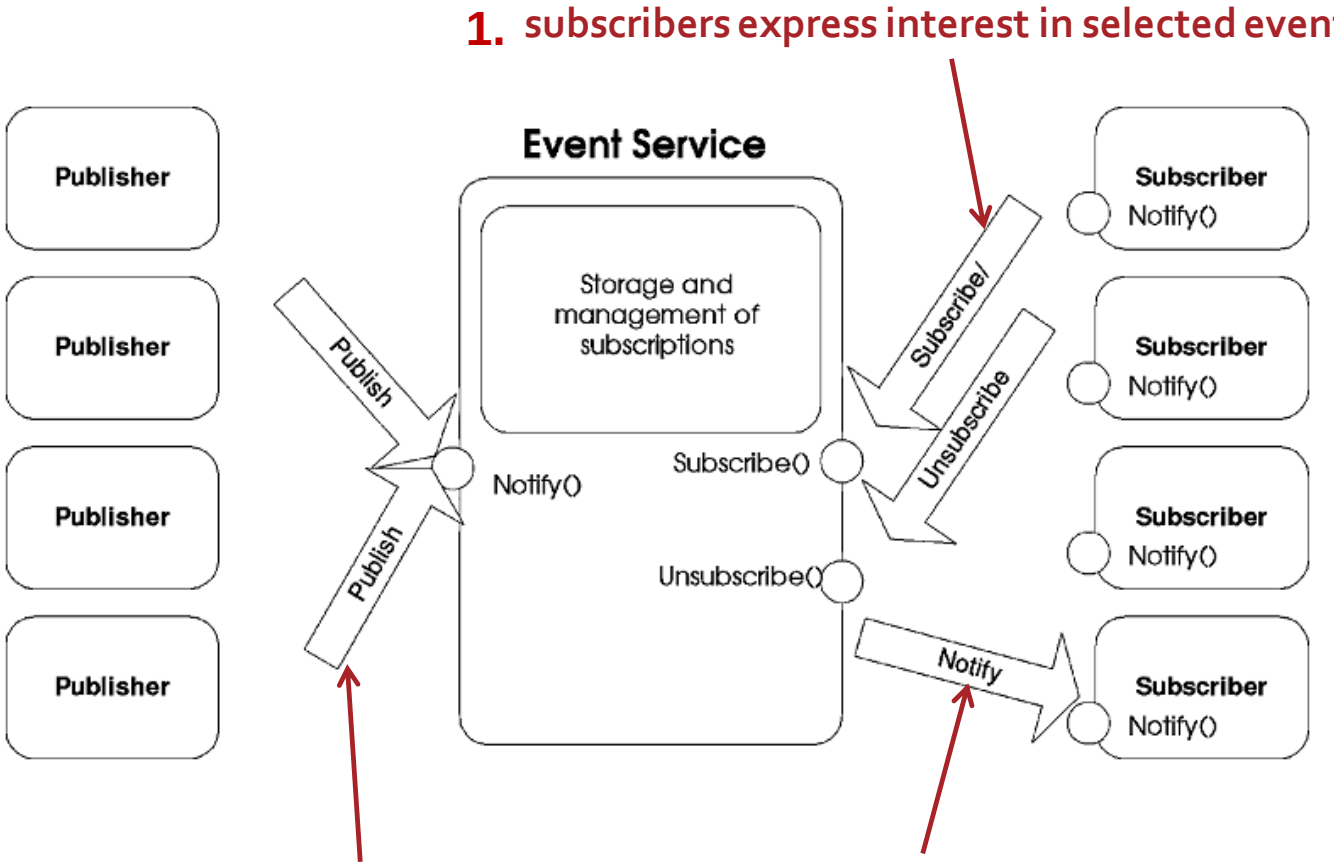
Jarosław Domaszewicz

Instytut Telekomunikacji Politechniki Warszawskiej



Source: AIOTI

PUBLISH/SUBSCRIBE FUNDAMENTALS (1/5)

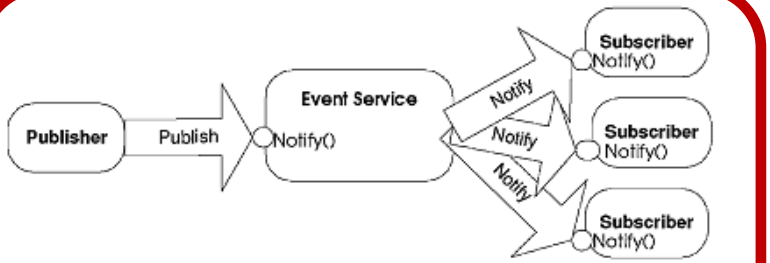


Source:
The Many Faces of Publish/Subscribe
P. Eugster et al.,
ACM Computing Surveys,
Vol. 35, No. 2, 2003

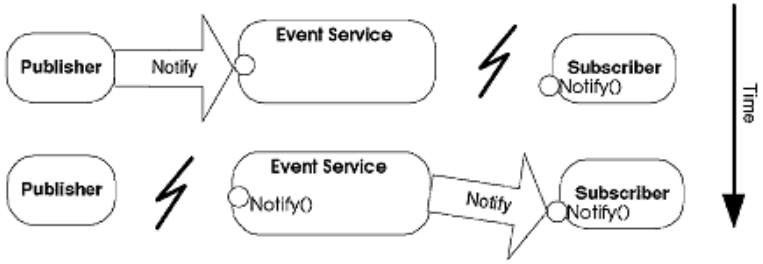
- **publishers**, **subscribers**, **event service** (middleware)
- note: push vs. pull, one-to-many, many-to-one

PUBLISH/SUBSCRIBE FUNDAMENTALS (2/5)

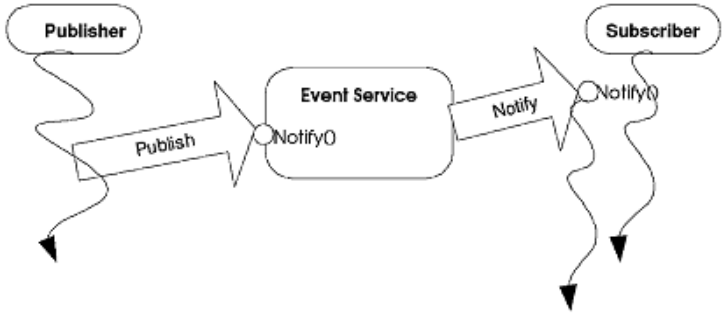
- **space decoupling**
  - publishers and subscribers do not know about one another (compare with client-server)
  - they do need to know about the event service
  - publishers do not know how many subscribers participate and vice versa



Space decoupling



Time decoupling



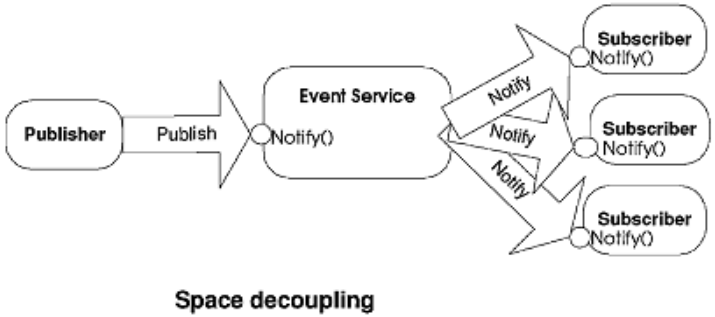
Synchronization decoupling

decoupling is good!

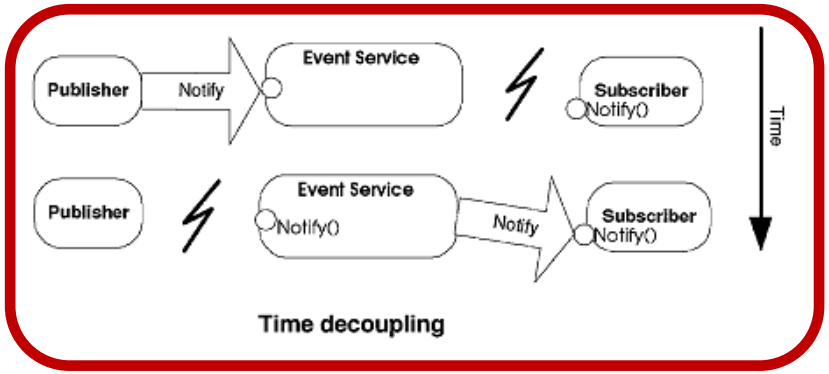
Source: *The Many Faces of Publish/Subscribe*
P. Eugster et al.
ACM Computing Surveys, Vol. 35, No. 2, 2003

PUBLISH/SUBSCRIBE FUNDAMENTALS (3/5)

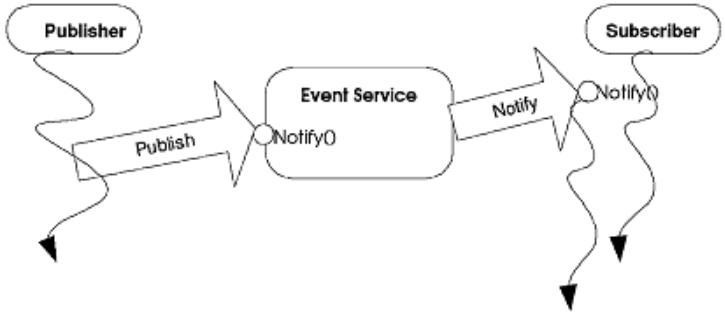
- **time decoupling**
  - publishing and delivery may occur at different times (compare with client-server) …
  - … (but most often they occur close in time)



Space decoupling



Time decoupling



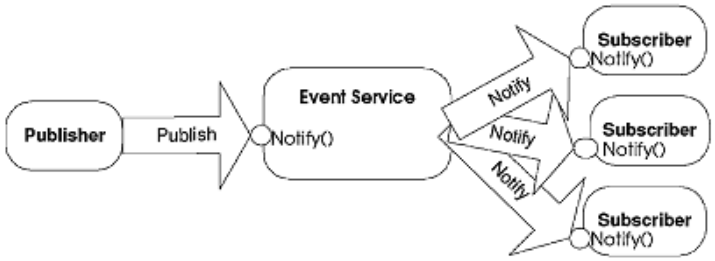
Synchronization decoupling

decoupling is good!

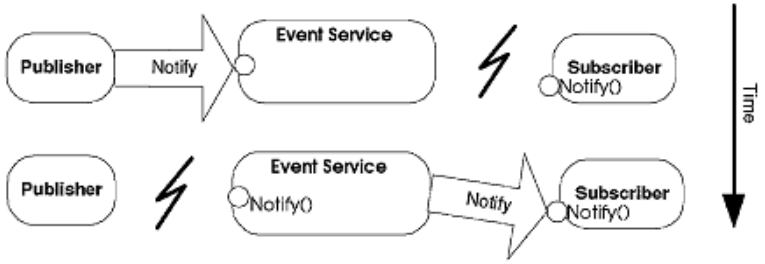
Source: *The Many Faces of Publish/Subscribe*
P. Eugster et al.
ACM Computing Surveys, Vol. 35, No. 2, 2003

PUBLISH/SUBSCRIBE FUNDAMENTALS (4/5)

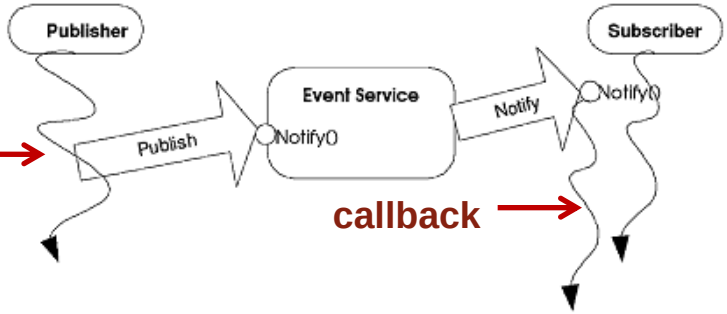
- **synchronization decoupling**
  - when publishing, publishers are not blocked
  - subscribers receive notification asynchronously (via a callback)



Space decoupling



Time decoupling



publishing is non-blocking

callback

Synchronization decoupling

decoupling is good!

Source: *The Many Faces of Publish/Subscribe*
P. Eugster et al.
ACM Computing Surveys, Vol. 35, No. 2, 2003

PUBLISH/SUBSCRIBE FUNDAMENTALS (5/5)

- how to specify events of interest?

- **topic-based** publish/subscribe
  - also called subject-based
  - subscribers subscribe to events published under a given topic
  - publishers tag events with a topic
  - each topic amounts to an individual communication channel

- **content-based** publish/subscribe
  - subscriptions based on the actual content of events
  - a subscriber provides a subscription pattern that refers to the content

- **type-based** publish/subscribe
  - subscription based on the type of (kind of) events

MQTT: KEY FACTS

- an OASIS and ISO standard
  - new: version 5 (but we cover v. 3.1.1)

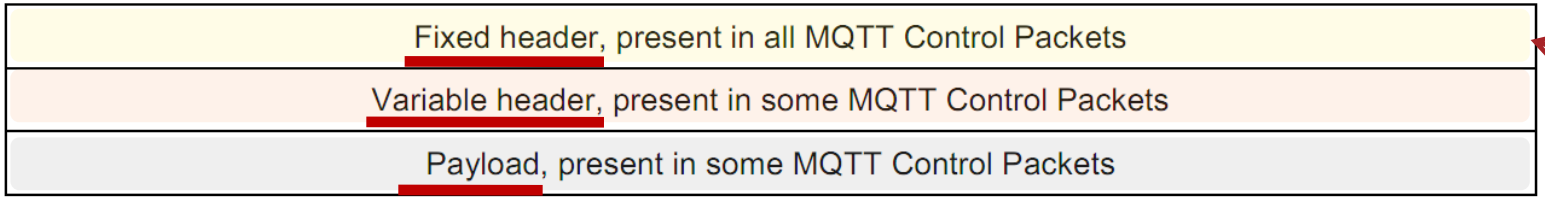
- runs on top of TCP
- implements publish/subscribe
- binary and lightweight
- agnostic as to application data
- some terminology:
  - **MQTT clients**: publishers, subscribers
  - **MQTT broker** (server): event service
  - **MQTT session**: an MQTT-level connection between a client and the broker

MQTT MECHANISMS

- topics and topic filters
- retained messages
- Keep Alive
- will
- persistent sessions
- (application-level) QoS (i.e., reliability)
  - even though it runs on top of TCP
  - takes care of broken TCP connections

MQTT Control Packet format

Figure 2.1 – Structure of an MQTT Control Packet



Source:
MQTT Version 3.1.1
OASIS Standard , October 2014

ASIDE: MQTT DATA REPRESENTATIONS

- Bits
  - bits in a byte are labeled 7 through 0; bit number 7 is the most significant bit
- Integer data values (fixed length)
  - 16 bits in big-endian order: the high order byte precedes the lower order byte
- Variable-length integers
  - least significant byte first →

1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

- UTF-8 strings.

Figure 1.1 Structure of UTF-8 encoded strings

Bit	7	6	5	4	3	2	1	0
byte 1	String length MSB							
byte 2	String length LSB							
byte 3 ….	UTF-8 Encoded Character Data, if length > 0.							

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

MQTT Control Packet fixed header

Figure 2.2 - Fixed header format

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							

length of variable header and payload

Name	Value
Reserved	0
CONNECT	1
CONNACK	2
PUBLISH	3
PUBACK	4
PUBREC	5
PUBREL	6
PUBCOMP	7
SUBSCRIBE	8
SUBACK	9
UNSUBSCRIBE	10
UNSUBACK	11
PINGREQ	12
PINGRESP	13
DISCONNECT	14
Reserved	15

connection establishment

publishing

ensuring QoS in delivering published messages

subscription management

heartbeat

disconnection

DUP[1]	QoS[2]	QoS[2]	RETAIN[3]

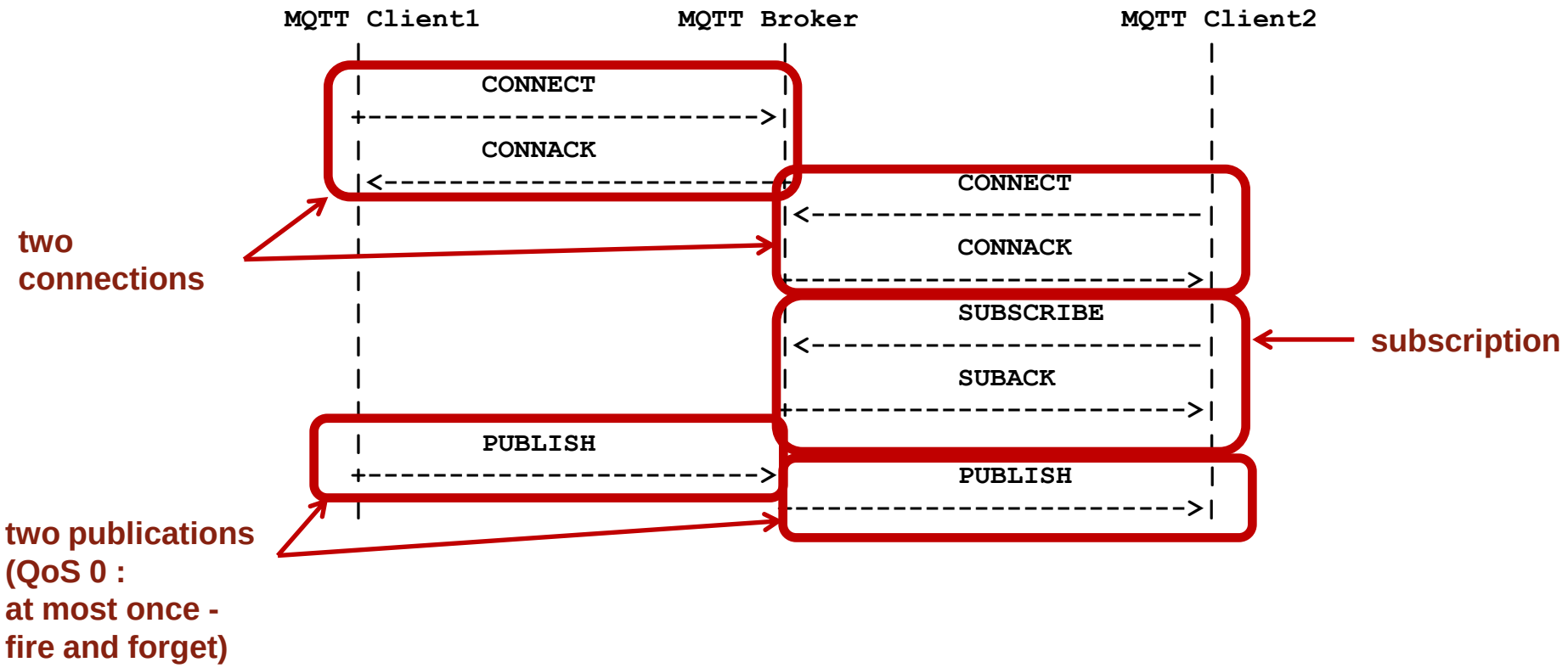
flags used in PUBLISH only

client to broker

broker to client

Source:
MQTT Version 3.1.1
OASIS Standard , October 2014

TYPICAL MESSAGE EXCHANGE



MQTT Control Packet variable header (1/3)

- In CONNECT:
  - Protocol Name (”MQTT”)
  - Protocol Level (rev. level)
  - Connect Flags

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

Figure 3.4 - Connect Flag bits

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

  - Keep Alive time

- In CONNACK:
  - CONNECT return code (accepted or refused with reason)

MQTT Control Packet variable header (2/3)

- In PUBLISH:
  - Topic Name

MQTT Control Packet variable header (3/3)

- In assorted packets:
  - Packet Identifier (2B)

Control Packet	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)
PUBACK	YES
PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES
SUBACK	YES
UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

MQTT Control Packet payload

Control Packet	Payload	
CONNECT	Required	ClientID, Will Topic, Will Message, User Name, Password
CONNACK	None	
PUBLISH	Optional	Application Message (note: MQTT is agnostic as to the format)
PUBACK	None	
PUBREC	None	
PUBREL	None	
PUBCOMP	None	
SUBSCRIBE	Required	(Topic_Filter_1,QoS_1), (Topic_Filter_2,QoS_2),…// requested QoS
SUBACK	Required	QoS_1, QoS_2, … // granted QoS
UNSUBSCRIBE	Required	Topic_Filter_1, Topic_Filter_2, …
UNSUBACK	None	
PINGREQ	None	
PINGRESP	None	
DISCONNECT	None	

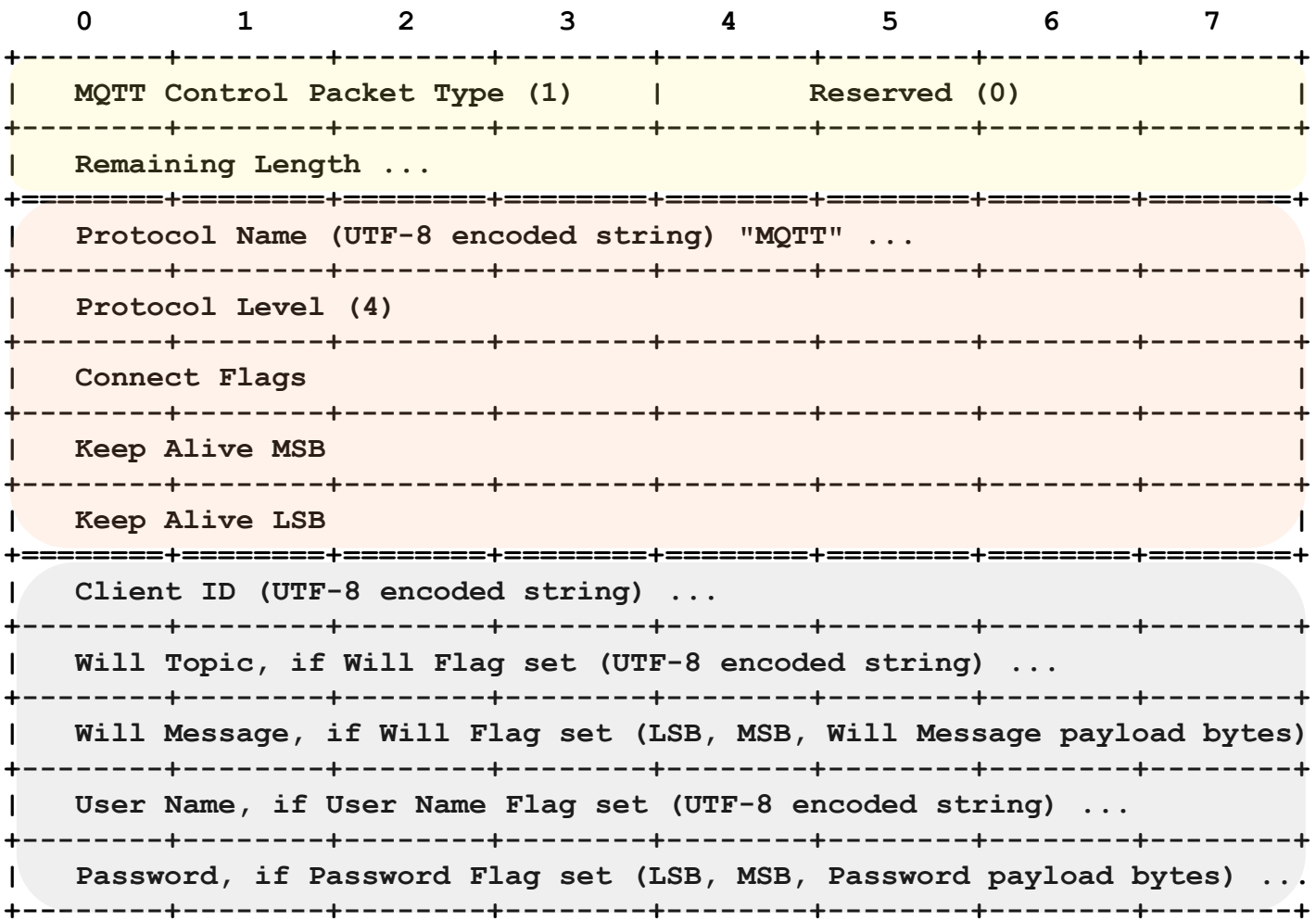
underlined payload elements are mandatory

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

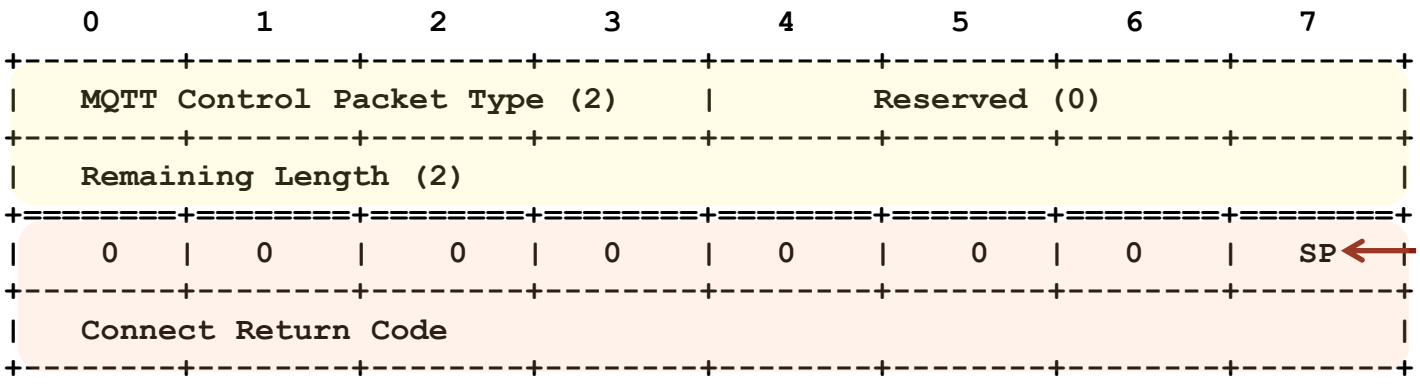
CONNECT

Figure 3.4 - Connect Flag bits

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



CONNACK



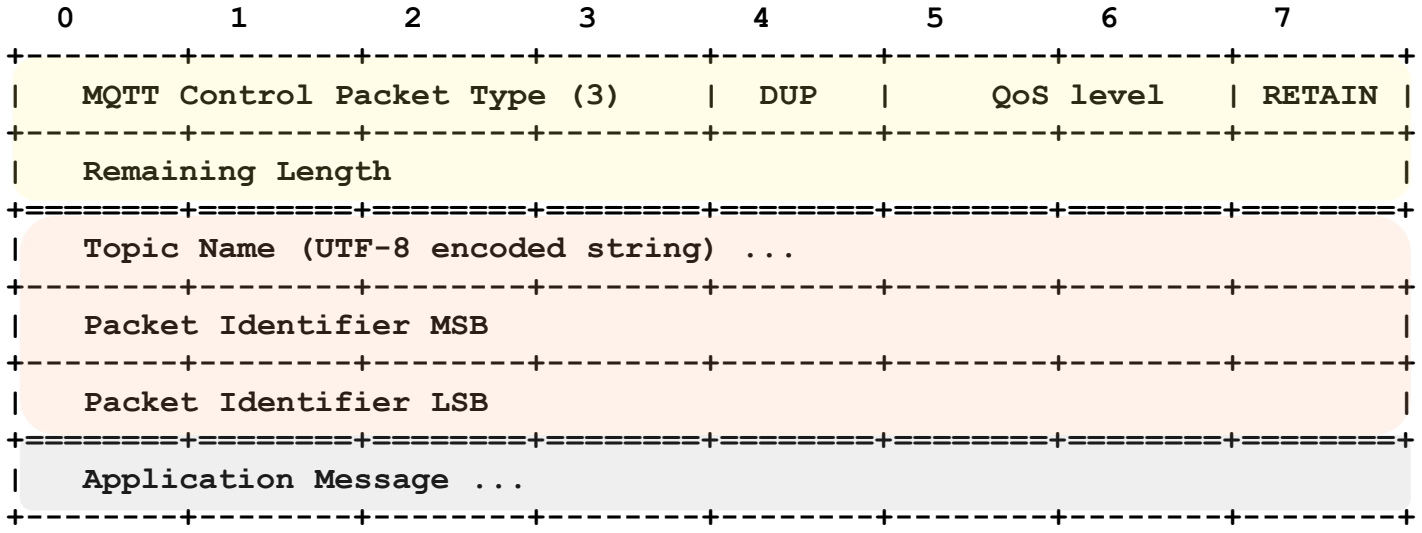
Session Present Flag (see persistent sessions)

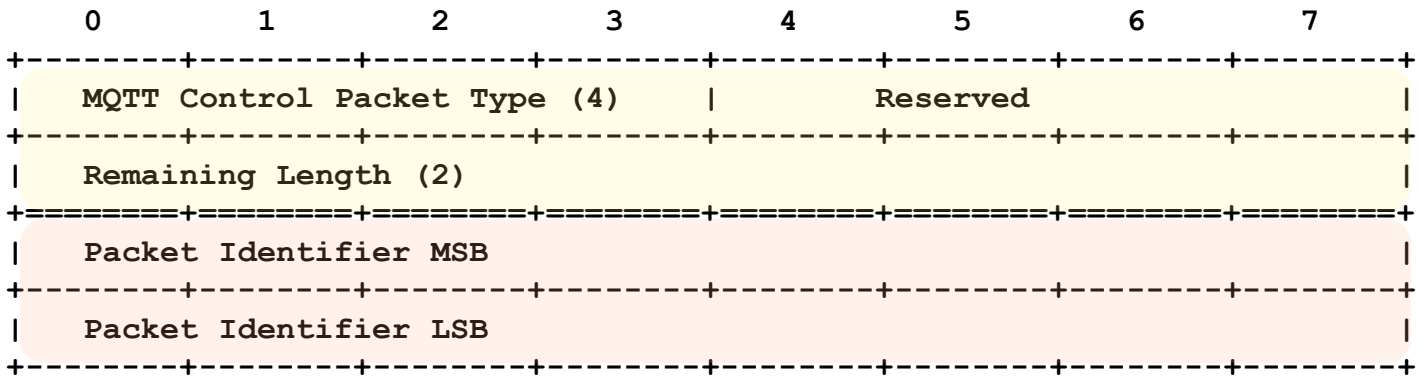
Source:
MQTT Version 3.1.1
OASIS Standard , October 2014

Table 3.1 – Connect Return code values

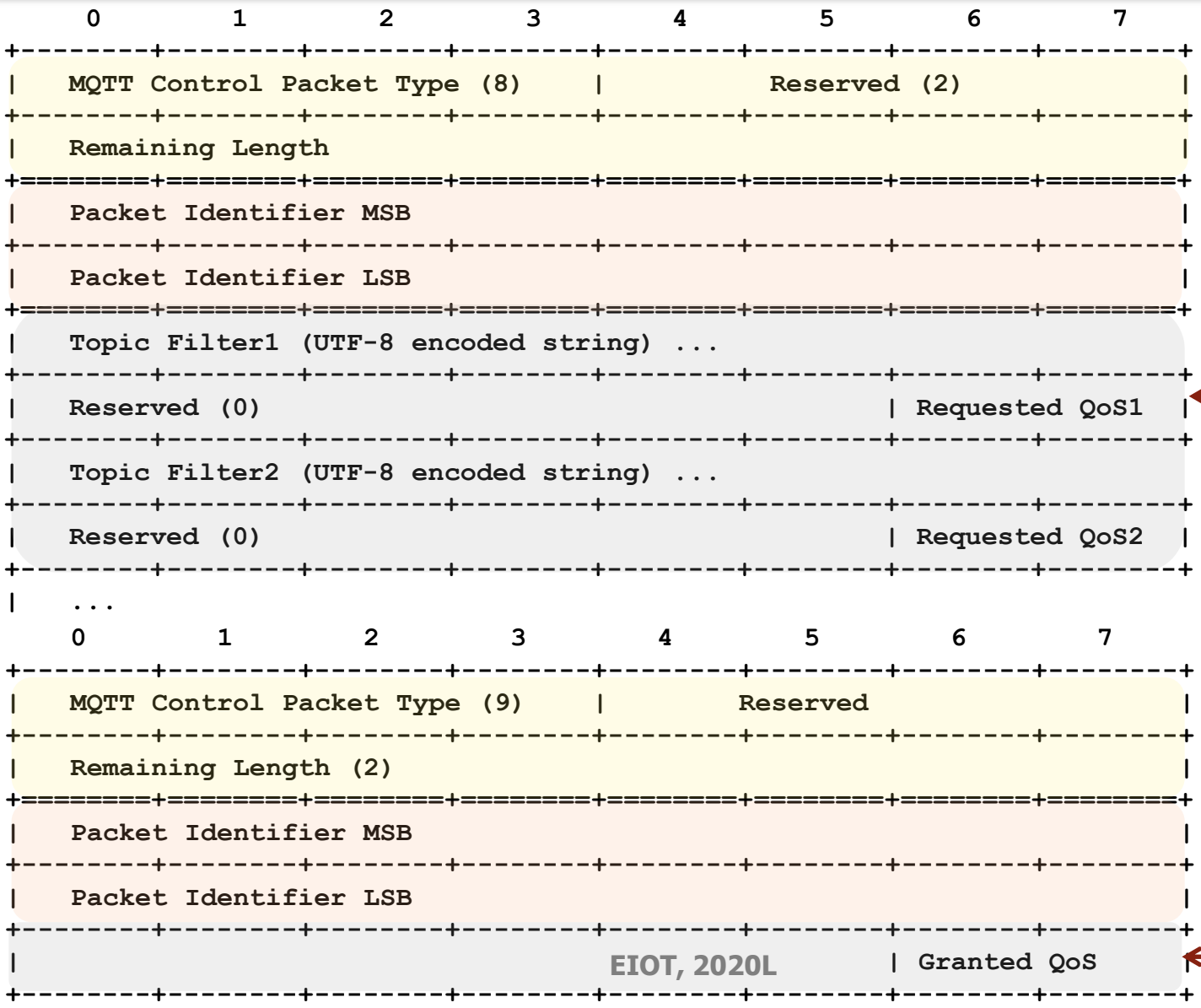
Value	Return Code Response	Description
0	0x00 Connection Accepted	Connection accepted
1	0x01 Connection Refused, unacceptable protocol version	The Server does not support the level of the MQTT protocol requested by the Client
2	0x02 Connection Refused, identifier rejected	The Client identifier is correct 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

PUBLISH AND PUBACK





SUBSCRIBE AND SUBACK



per topic filter: maximum QoS requested ("do not send at a higher QoS")

per topic filter: maximum QoS granted

SUBSCRIBE AND SUBACK

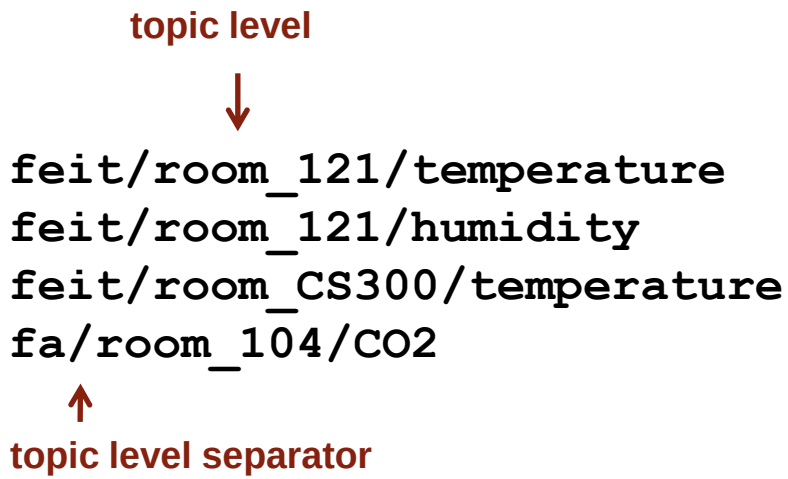
- In SUBACK, the broker might grant a lower maximum QoS than the subscriber requested.

- The QoS of PUBLISH messages sent by the broker to the subscriber is equal to …

  min(QoS of the message originally published by the publisher,
  maximum QoS granted by the broker).

TOPICS

- topics have a hierarchical structure:



- the broker accepts any topic, without earlier registration
  - MQTT is agnostic as to topics (just like with application messages)
- topics starting with ”$” are reserved (e.g., for monitoring purposes)

```
feit/room_121/temperature
feit/room_121/humidity
feit/room_CS300/temperature
fa/room_104/CO2
```

TOPIC FILTERS

- to subscribe to more than one topic at once, use **wildcards**
  - allowed only in topic filters (in SUBSCRIBE)

- **single-level** wildcard (**+**)
  - example: `feiti/+/temperature` (all temperature readings from FEITI)
  - example: `+/+/co2` (all CO2 readings from entire WUT)

- **multi-level** wildcard (**#**)
  - example: `fa/#` (all sensor readings from FA)
  - must be the last character in the topic

TOPICS IN REAL LIFE (1/3)

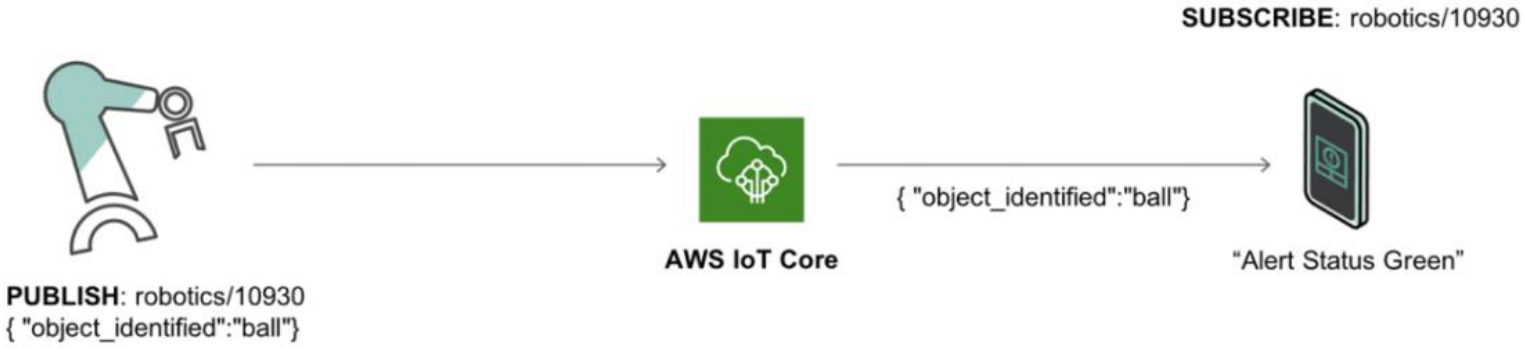


Figure 1: One to One Messaging in Point-to-point Communication

Source: Amazon Web Services, *Designing MQTT Topics for AWS IoT Core*, May 2019

TOPICS IN REAL LIFE (2/3)

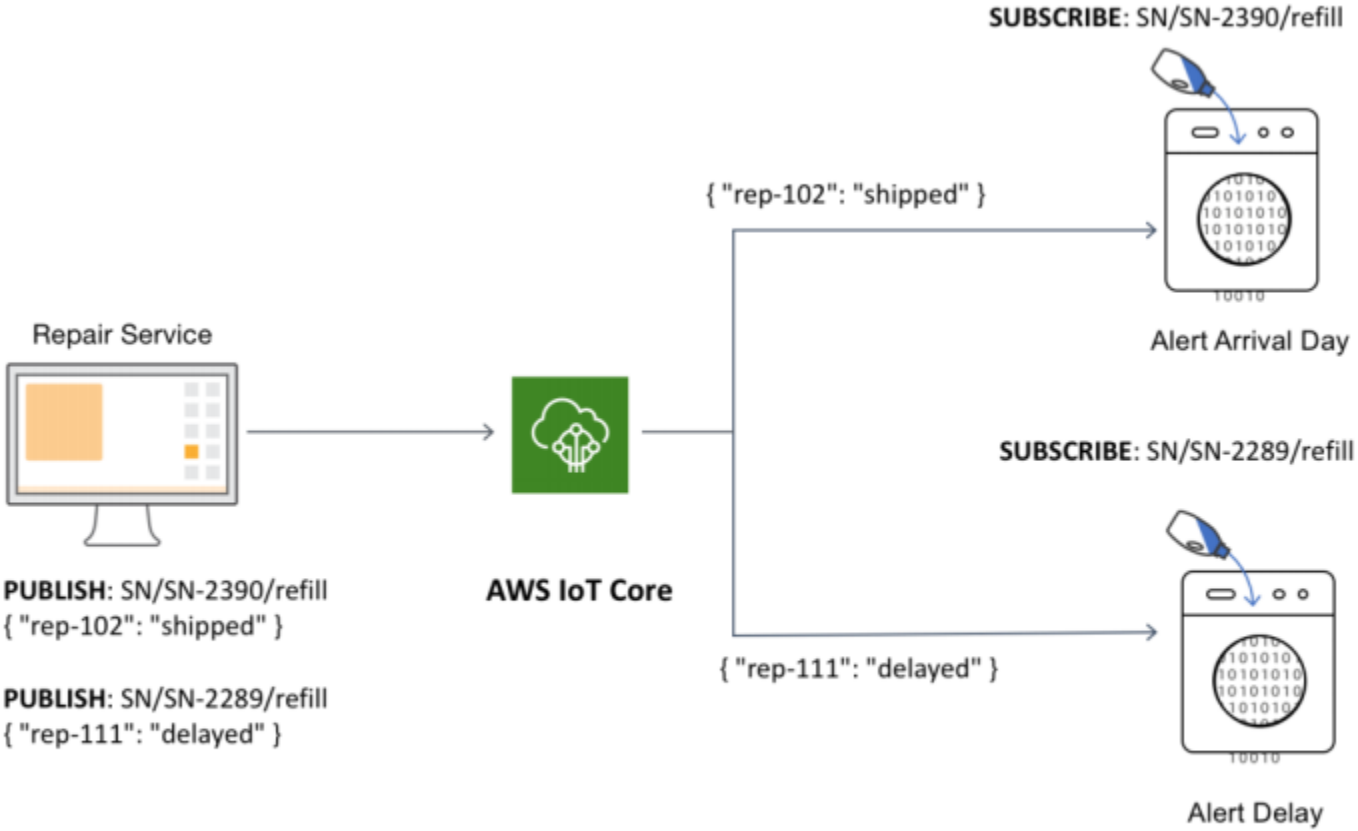


Figure 2: One-to-many messaging in point-to-point communication

Source: Amazon Web Services, *Designing MQTT Topics for AWS IoT Core*, May 2019

TOPICS IN REAL LIFE (3/3)

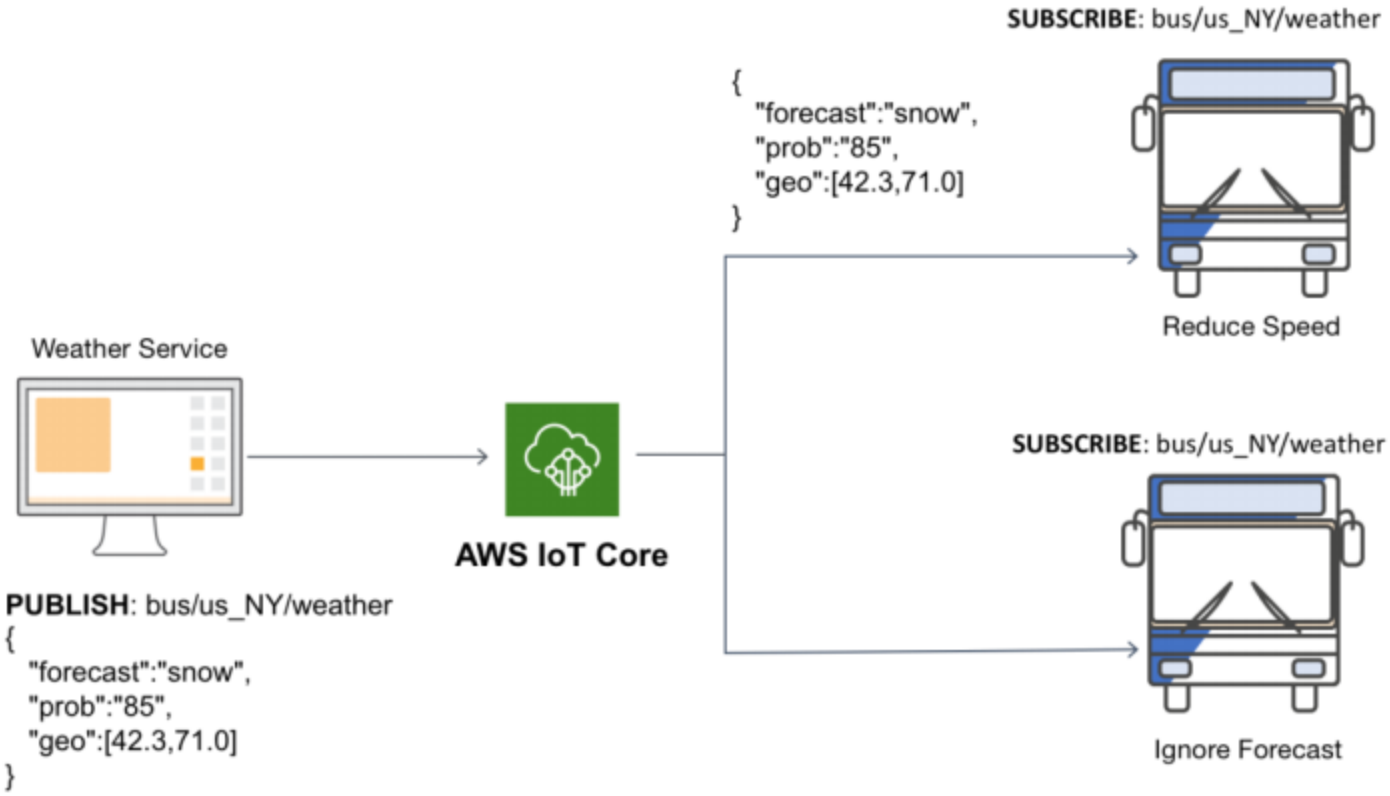


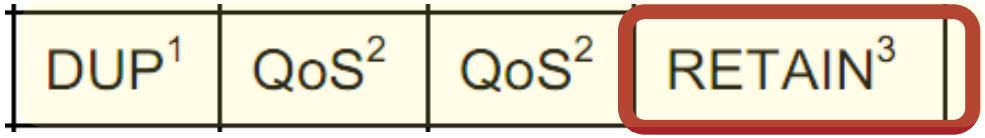
Figure 3: One-to-many messaging in broadcast communication

Source: Amazon Web Services, *Designing MQTT Topics for AWS IoT Core*, May 2019

```
+--------+--------+--------+--------+--------+--------+--------+--------+
|   MQTT Control Packet Type (3)     |  DUP   |     QoS level    | RETAIN |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

RETAINED MESSAGES (1/2)

- a client may subscribe (to a topic) at any time
- how quickly after subscribing will the client receive the first message?
  - publishing client(s) may not have anything to publish for some time, …
  - … and the subscribing client does not know anything ”about the topic”
- recall fixed header flags in PUBLISH



Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

- if RETAIN is set by the publisher, it is a **retained message** (i.e., stored by the broker)

RETAINED MESSAGES (2/2)

- the broker stores only one retained message per topic
  - new retained message overwrites the last one
- a client will receive a topic's retained message directly after subscribing to it
  - RETAIN will also be set, so the client knows it is a retained message
- to delete one, send a retained message with the empty payload

CONNECTIONS: KEEP ALIVE

- how will the broker find out that a client is no longer there?
  - lost connection, battery, crashing
- how will the client find out about a lost connection?

- recall the variable header part of CONNECT:
  - Protocol Name and Level, Connect Flags, Keep Alive time (in seconds)

- **Keep Alive time** (in CONNECT) and **keepalive timer** (at the broker)
  - for how long the server may not hear from the client before disconnection?
  - actually, it's 1.5*keep _alive_time
  - keepalive timer started at the broker each time a message is received
  - if no other messages to send, the client sends PINGREQ
  - the server responds with PINGRESP
  - if PINGREQ not received, the server closes the connection
  - if PINGRESP not received (”within a reasonable amount of time”), the client closes the connection

CONNECTIONS: THE SERVER'S MAY DECIDE

- ”a server is permitted to disconnect a client that it determines to be inactive or non-responsive at any time,
regardless of the Keep Alive value provided by that client.”

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

TESTAMENT (WILL)

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

Figure 3.4 - Connect Flag bits

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

- recall CONNECT
  - variable header: Protocol Name and Level, Keep Alive time, Connect Flags
  - payload: ClientID, Will Topic, Will Message, User Name, Password
- graceful (clean) disconnection: DISCONNECT
- when does the broker send the Will Message?
  - it has not heard from the client for 1.5*keepalive_time
  - the client closes the network connection without DISCONNECT
  - the broker closes the network connection because of a protocol error
  - the broker detects a network error
- the broker will send the Will Message to all that subscribed to the Will Topic

PERSISTENT SESSIONS (1/2)

- a client can lose the connection quite often
  - messages lost when not connected
  - the need to re-subscribe to topics of interest after re-connection
- **persistent session**: the broker keeps a per client session state
  - unacknowledged messages with QoS 1 or 2 (see below)
  - all subscriptions
- on a re-connection (if a client so desires) …
  - all missed messages with QoS 1 or 2 are delivered
  - no need to re-subscribe

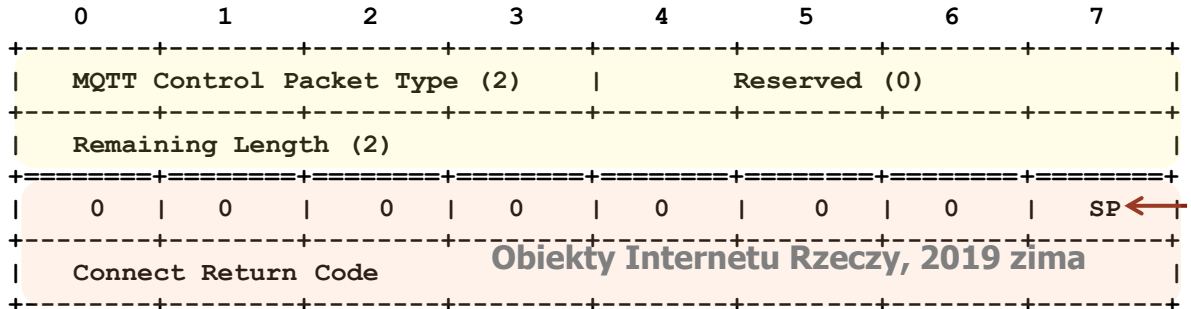
PERSISTENT SESSIONS (2/2)

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

Figure 3.4 - Connect Flag bits

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

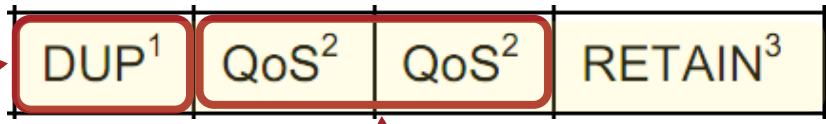
- recall CONNECT
  - variable header: Protocol Name and Level, Keep Alive time, Connect Flags
- Clean Session flag
  - if set, the broker discards the session state (if any) and does not maintain one
  - if not set, a persistent session is continued (or started)
  - if not set, the CONNACK SP flag indicates if the broker has a state for the client ID



CONNACK

Session Present Flag

fixed header flags in PUBLISH
If set, then this is a redelivery



QoS

Table 3.2 - QoS definitions

QoS value	Bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

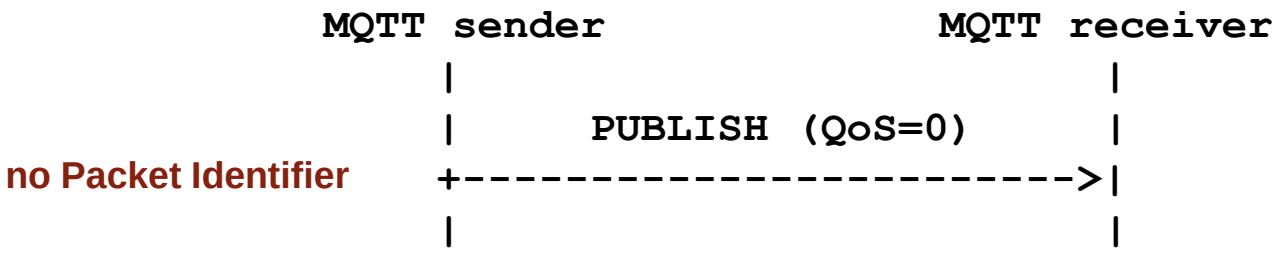
Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

- recall fixed header flags in PUBLISH
- may need reliability on top of TCP
  - a TCP connection may break down
  - there may be no space in the buffers at the receiver

- answer: application-level reliability
- QoS 0, **at most once**: QoS of the underlying network
  - no (MQTT-level) ACK
- QoS 1, **at least once**
  - the receiver responds with ACK
  - the receiver does not detect duplicates
- QoS2, **exactly once**
  - the receiver responds with ACK, etc.
  - the receiver does detect (and discards) duplicates

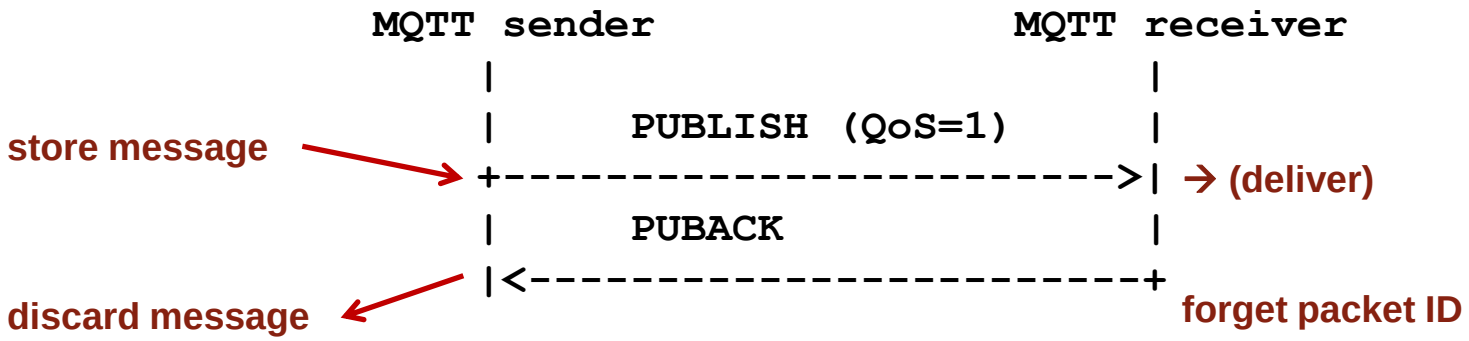
QoS 0 (AT MOST ONCE)

- fire and forget
- the message arrives once or not at all



QoS 1 (AT LEAST ONCE) (1/4)

- the receiver responds with ACK



QoS 1 (AT LEAST ONCE) (2/4)

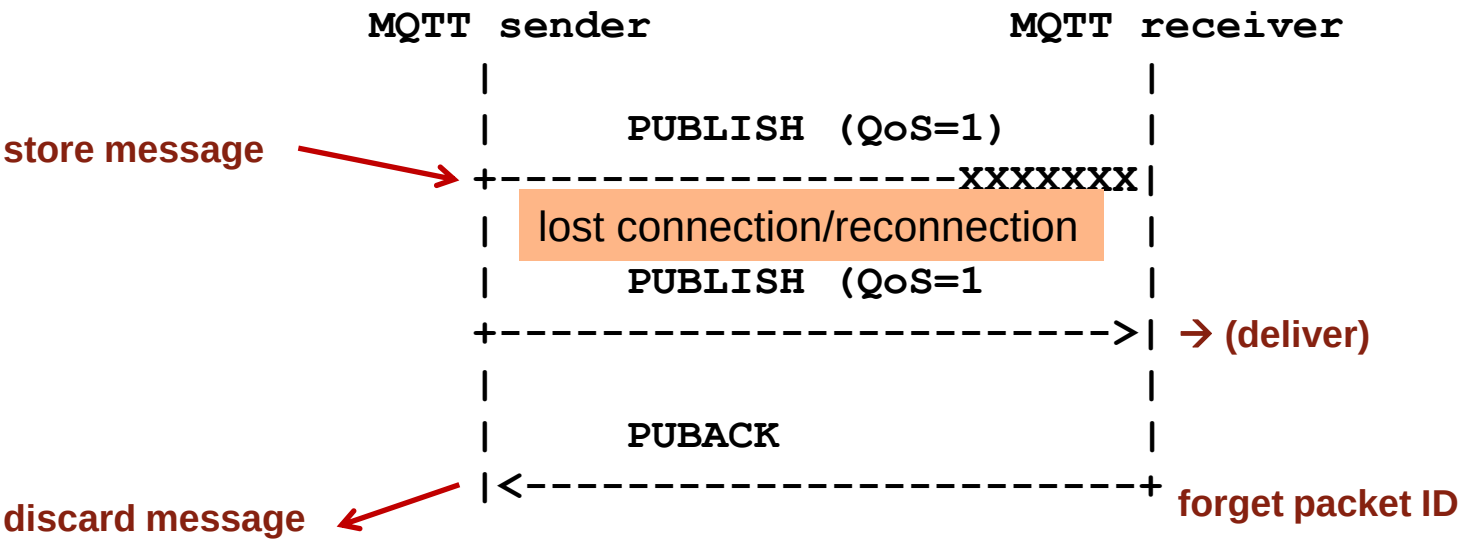
- what if either of the two messages are lost due a lost connection?
- on reconnection, the sender resends if the packet unacknowledged

”When a Client reconnects with CleanSession set to 0,
both the Client and Server MUST re-send any unacknowledged
PUBLISH Packets (where QoS > 0) using their original Packet Identifiers.

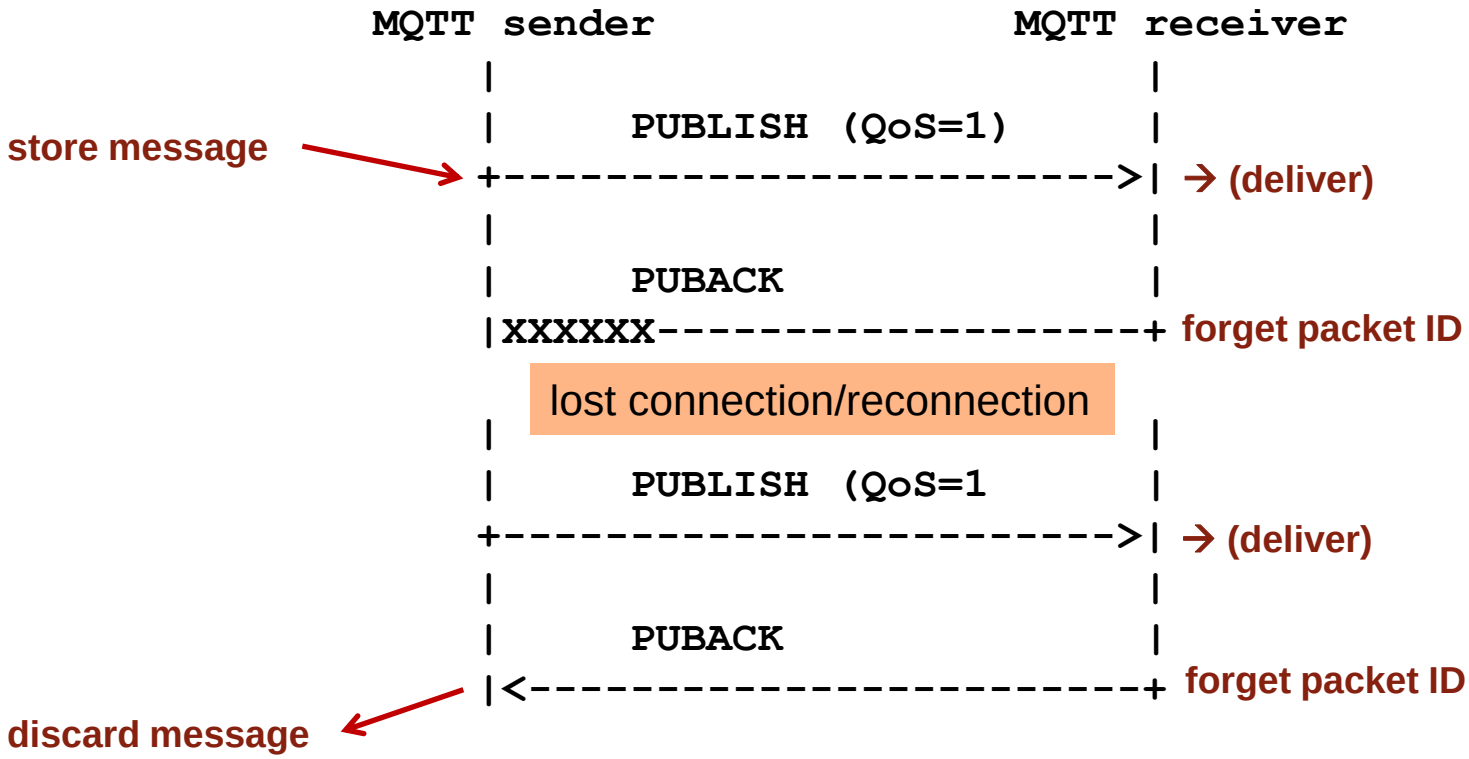
This is the only circumstance where a Client or Server is REQUIRED
to redeliver messages.”

Source: *MQTT Version 3.1.1*
OASIS Standard , October 2014

QoS 1 (AT LEAST ONCE) (3/4)



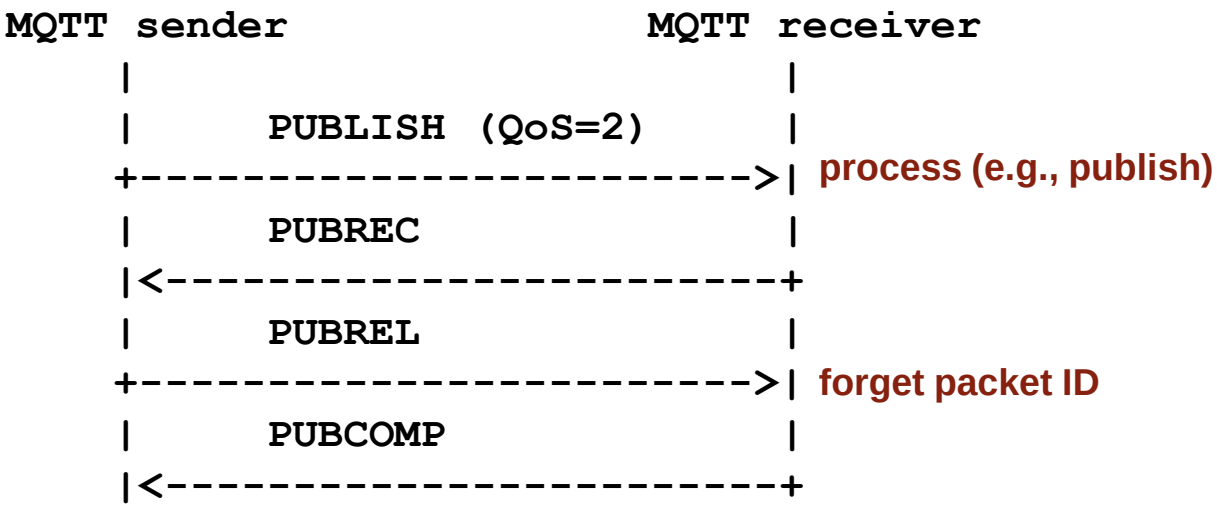
QoS 1 (AT LEAST ONCE) (4/4)



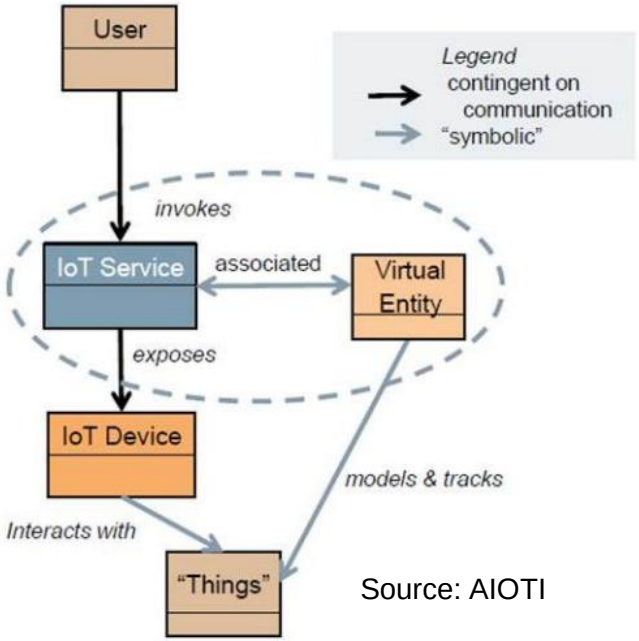
- the receiver does not detect duplicates
- multiple copies of the message may be delivered

QoS 2 (EXACTLY ONCE)

- the receiver does detect (and discards) duplicates



Thank you!



Source: AIOTI