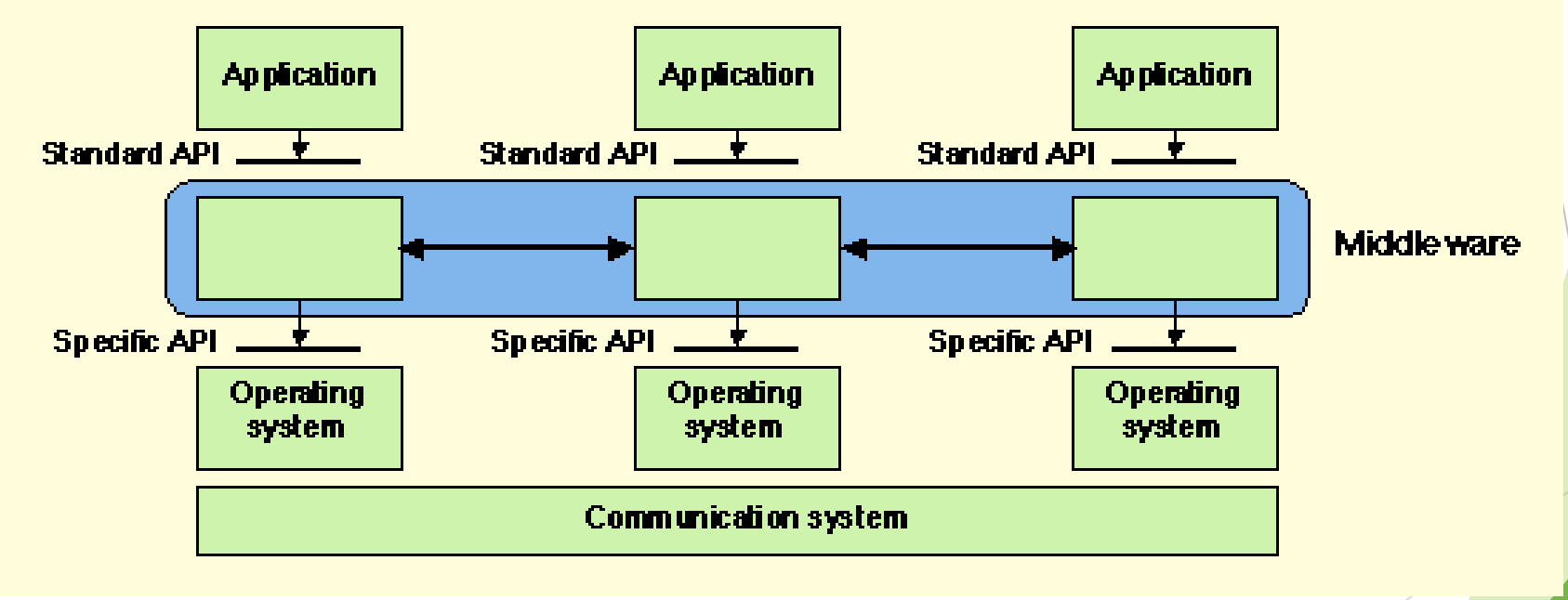
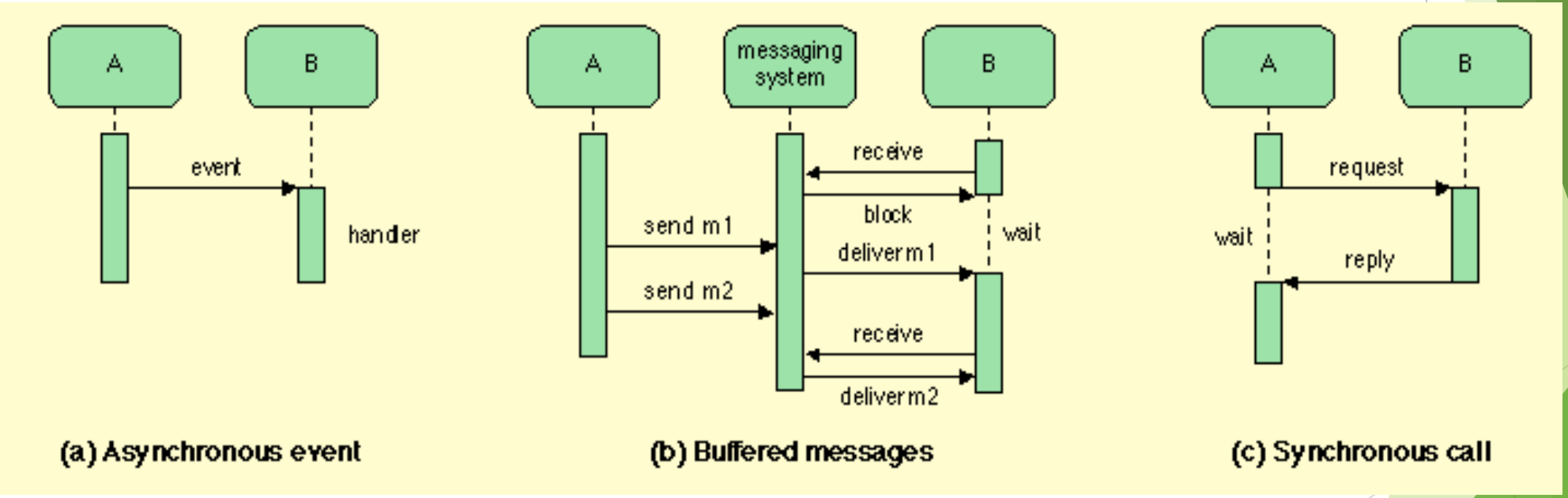
Middleware Introduction

Introduction to Middleware



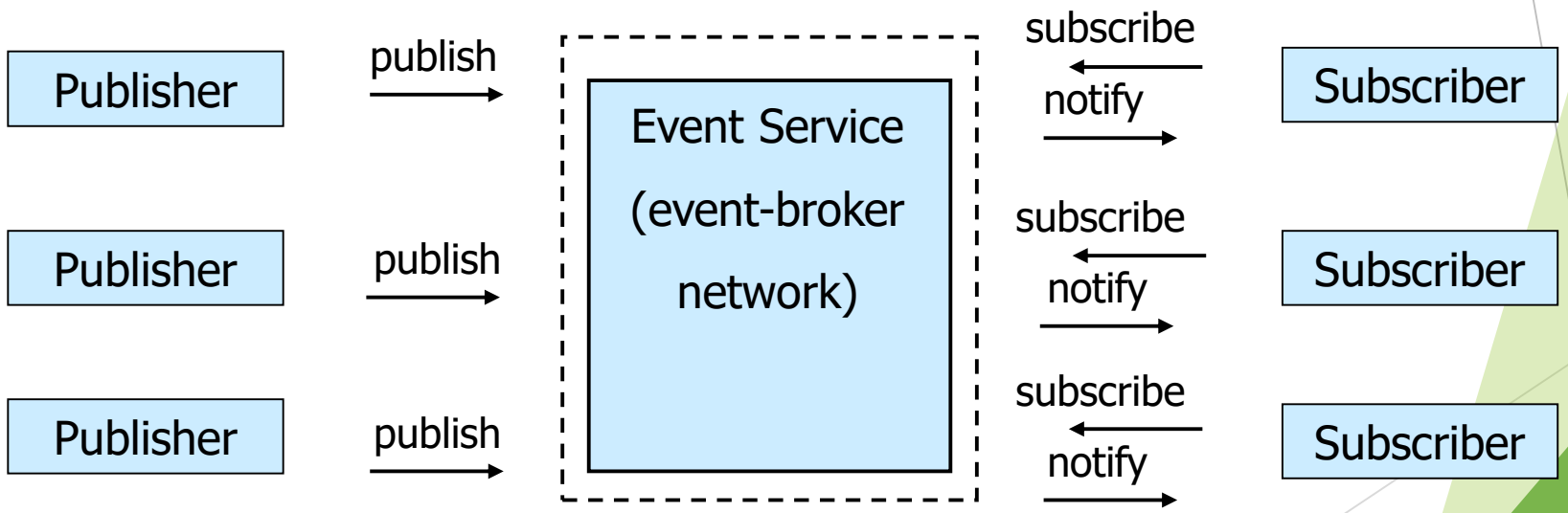
Basic Communication Patterns



Event based Middleware

Event-Based Middleware, ex. Publish/Subscribe Pattern

- Publishers (advertise and) publish events (messages)
- Subscribers express interest in events with subscriptions
- Event Service notifies interested subscribers of published events
- Events can have arbitrary content (typed) and name/value pairs



MQTT

Message Queue Telemetry Transport

http://mqtt.org/

MQTT - Open Connectivity for Mobile, M2M and IoT

- A lightweight publish/subscribe protocol with predictable bi-directional message delivery

1999 Invented by Dr. Andy Stanford-Clark (IBM), Arlen Nipper (now Cirrus Link Solutions)

2004 MQTT.org open community

2011 - Eclipse PAHO MQTT open source project

2013 – MQTT Technical Committee formed

Cimetrics, Cisco, Eclipse, dc-Square, Eurotech, IBM, INETCO Landis & Gyr, LSI, Kaazing, M2Mi, Red Hat, Solace, Telit Comms, Software AG, TIBCO, WSO2

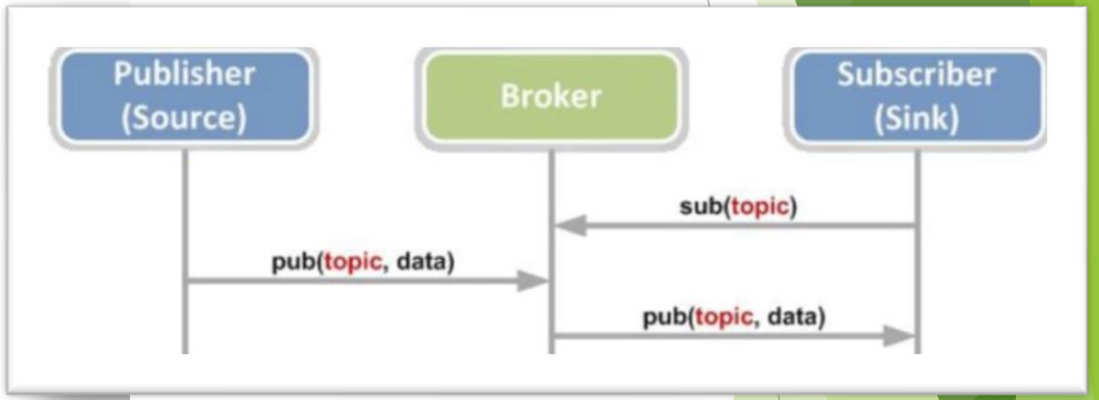
paho

Evolution of an open technology

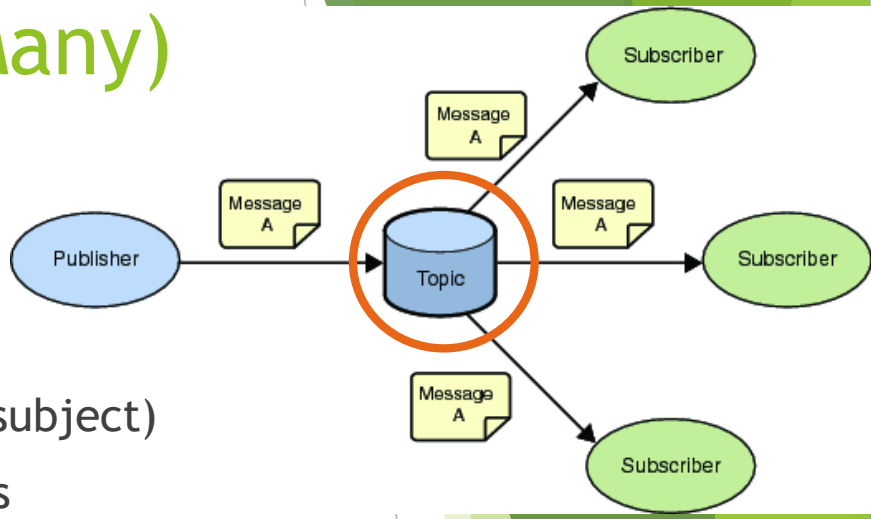
MQTT

Event based IoT Middleware
- Event pattern of communication (one to many)
- Over IP (TCP)

- MQTT is described on the mqtt.org site as a machine-to-machine (M2M) / IoT connectivity protocol.
- MQTT is an Event based IoT middleware (one to many)
  - publish/subscribe messaging transport protocol
  - Over TCP/IP (or MQTT-S over UDP for LAN)
- Its protocol is lightweight
  - it can be supported by some of the smallest measuring and monitoring devices (ex. Arduino)
  - it can transmit data over far reaching networks
  - It can transmit data over sometimes intermittent networks.



Publish / Subscribe Messaging (One to Many)



- A producer publishes a message (publication) on a topic (subject)
- A consumer subscribes (makes a subscription) for messages on a topic (subject)
- A message server (called BROKER) matches publications to subscriptions
  - If none of them match the message is discarded after modifying the topic
  - If one or more matches the message is delivered to each matching consumer after modifying the topic

- Publish / Subscribe has three important characteristics:
  1. It decouples message senders and receivers, allowing for more flexible applications
  2. It can take a single message and distribute it to many consumers
  3. This collection of consumers can change over time, and vary based on the nature of the message.

MQTT Topic and Wildcards

MQTT Topics & Wildcards

- **Topics are hierarchical (like filesystem path):**
  - */wsn/sensor/R1/temperature*
  - */wsn/sensor/R1/pressure*
  - */wsn/sensor/R2/temperature*
  - */wsn/sensor/R2/pressure*
- **A Subscriber can use wildcards in topics:**
  - **/wsn/sensor/+/temperature**
  - **/wsn/sensor/R1/+**
  - **/wsn/sensor/#**

MQTT Topic : Details

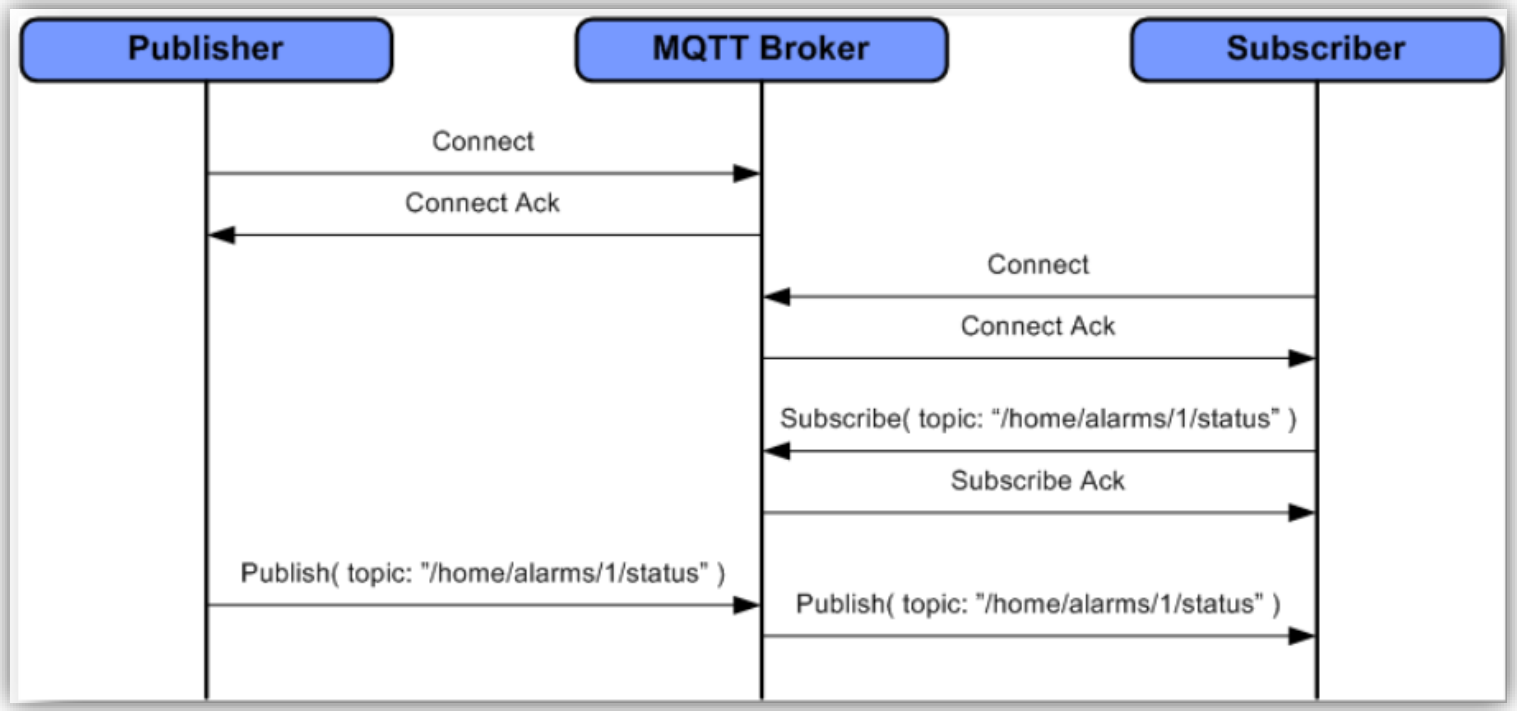
- A topic forms the namespace
  - Is hierarchical with each “sub topic” separated by a /
  - An example topic space
    - A house publishes information about itself on:
      - *<country>/<region>/<town>/<postcode>/<house>/energyConsumption*
      - *<country>/<region>/<town>/<postcode>/<house>/solarEnergy*
      - *<country>/<region>/<town>/<postcode>/<house>/alarmState*
      - *<country>/<region>/<town>/<postcode>/<house>/alarmState*
    - *And subscribes for control commands:*
      - *<country>/<region>/<town>/<postcode>/<house>/thermostat/setTemp*
- A subscriber can subscribe to an absolute topic or can use wildcards:
  - Single-level wildcards “+” can appear anywhere in the topic string
  - Multi-level wildcards “#” must appear at the end of the string
  - Wildcards must be next to a separator
  - Cannot be used wildcards when publishing
  - For example
    - *UK/Hants/Hursley/SO212JN/1/energyConsumption*
      - Energy consumption for 1 house in Hursley
    - *UK/Hants/Hursley/+/+/energyConsumption*
      - Energy consumption for all houses in Hursley
    - *UK/Hants/Hursley/SO212JN/#*
      - Details of energy consumption, solar and alarm for all houses in SO212JN

MQTT publish subscribe architecture

- The MQTT messages are delivered asynchronously (“push”) through publish subscribe architecture.
- The MQTT protocol works by exchanging a series of MQTT control packets in a defined way.
- Each control packet has a specific purpose and every bit in the packet is carefully crafted to reduce the data transmitted over the network.
- A MQTT topology has a MQTT server and a MQTT client.
- MQTT client and server communicate through different control packets. Table below briefly describes each of these control packets.

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

Sample of protocol use



Message Fixed Header

- MQTT messages contain a fixed header that includes flags :
  - in the first byte indicating the message type (four bits),
  - whether the message is being re-sent (one bit),
  - a quality-of-service (QOS) flag (two bits),
  - and a message retention flag (one bit).
- The remaining portion of the fixed header indicates the length of the rest of the message, which includes a header that varies by message type (hence it's called a variable header), and the message payload.

bit	7	6	5	4	3	2	1	0
byte 1	Message type				DUP	QoS level		RETAIN
byte 2	Message length (between one and four bytes)							
byte 3	… if needed to encode message length							
byte 4	… if needed to encode message length							
byte 5	… if needed to encode message length							

Ideal for constrained networks (low bandwidth, high latency, data limits, and fragile connections)

- MQTT control packet headers are kept as small as possible.
- Each MQTT control packet consist of three parts, a fixed header, variable header and payload.
- Each MQTT control packet has a 2 byte Fixed header. Not all the control packet have the variable headers and payload.
- A variable header contains the packet identifier if used by the control packet.
- A payload up to 256 MB could be attached in the packets.
- Having a small header overhead makes this protocol appropriate for IoT by lowering the amount of data transmitted over constrained networks.

Quality of Service (QoS) for MQTT

- Quality of service (QoS) levels determine how each MQTT message is delivered and must be specified for every message sent through MQTT. It is important to choose the proper QoS value for every message, because this value determines how the client and the server communicate to deliver the message. Three QoS for message delivery could be achieved using MQTT:
  - QoS 0 (At most once) - where messages are delivered according to the best efforts of the operating environment. Message loss can occur.
  - QoS 1 (At least once) - where messages are assured to arrive but duplicates can occur.
  - QoS 2 (Exactly once) - where message are assured to arrive exactly once.
- There is a simple rule when considering performance impact of QoS. It is “The higher the QoS, the lower the performance".

Quality of Service (QoS) for MQTT

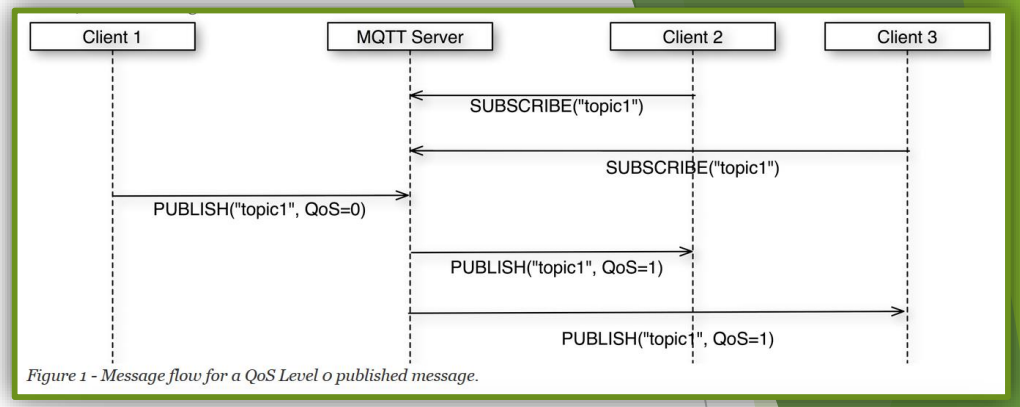


Figure 1 - Message flow for a QoS Level 0 published message.

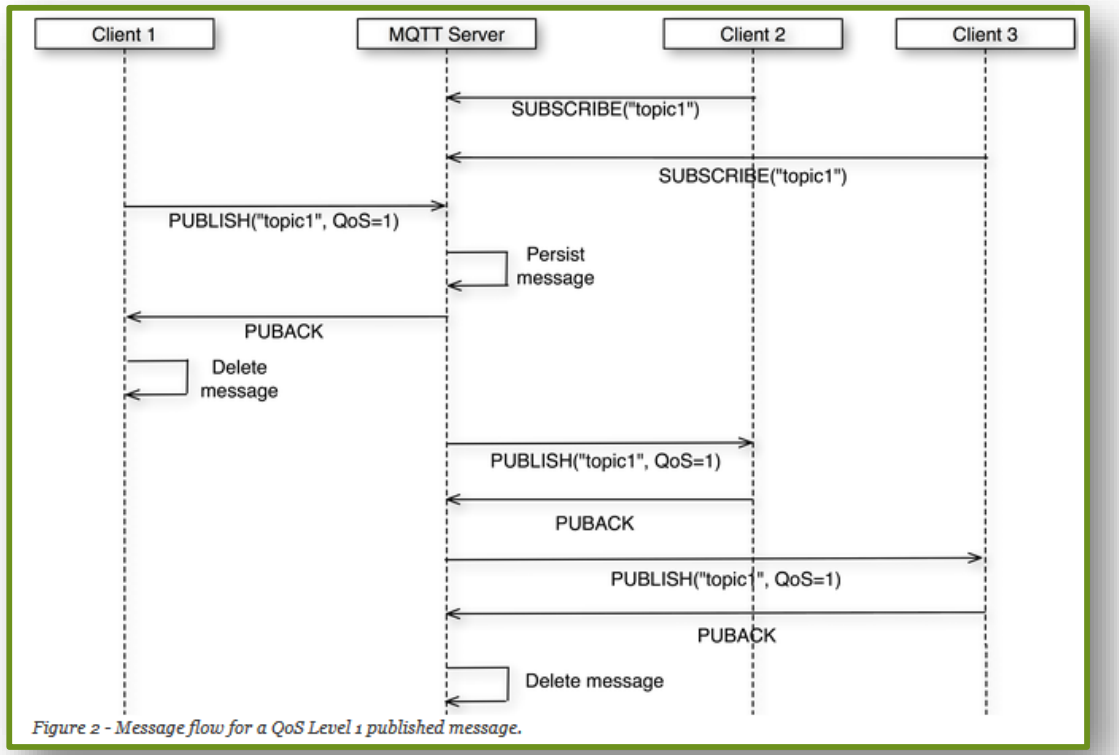


Figure 2 - Message flow for a QoS Level 1 published message.

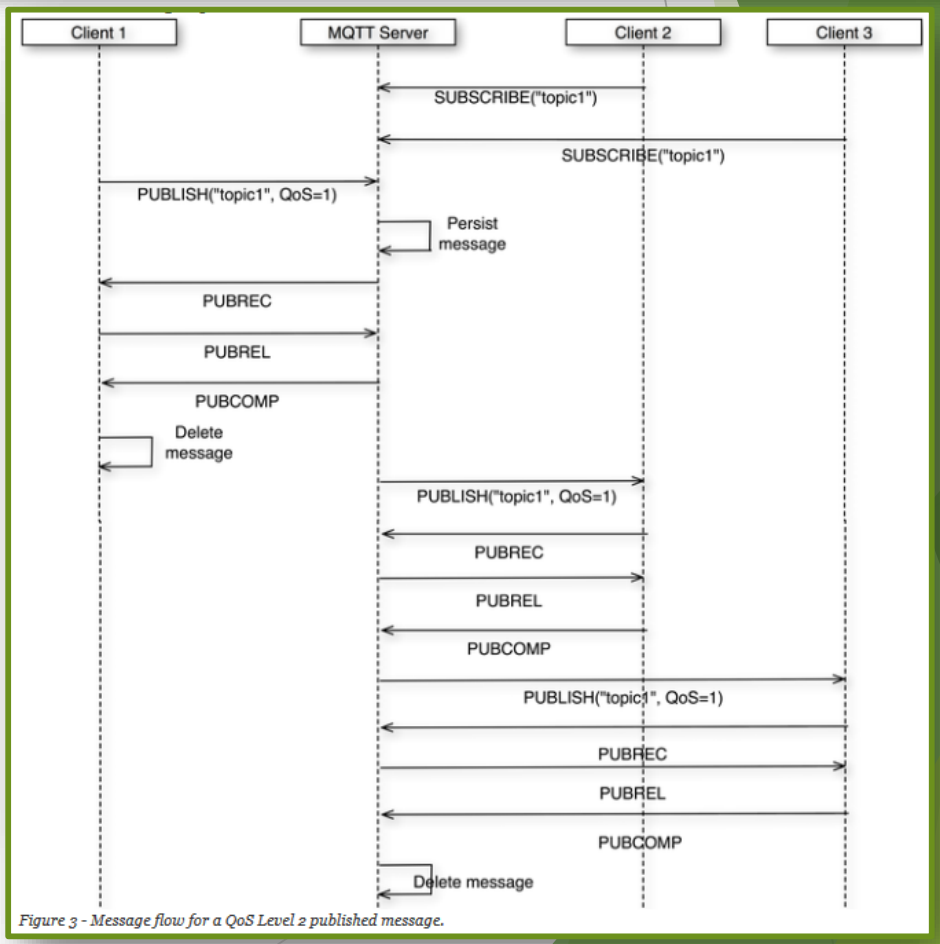


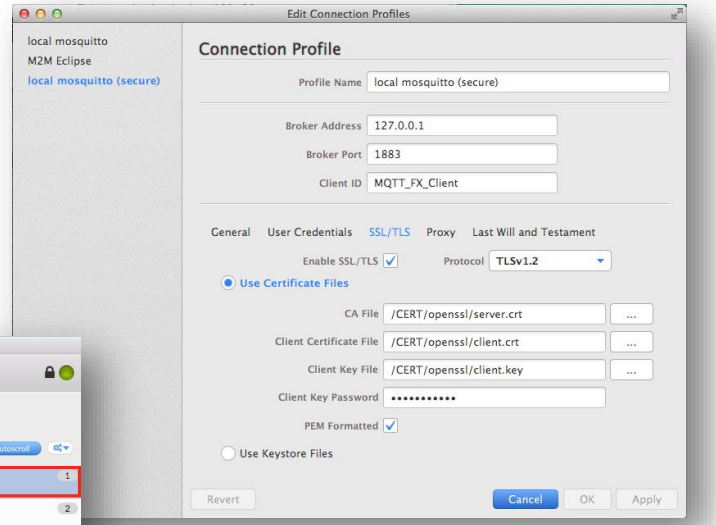
Figure 3 - Message flow for a QoS Level 2 published message.

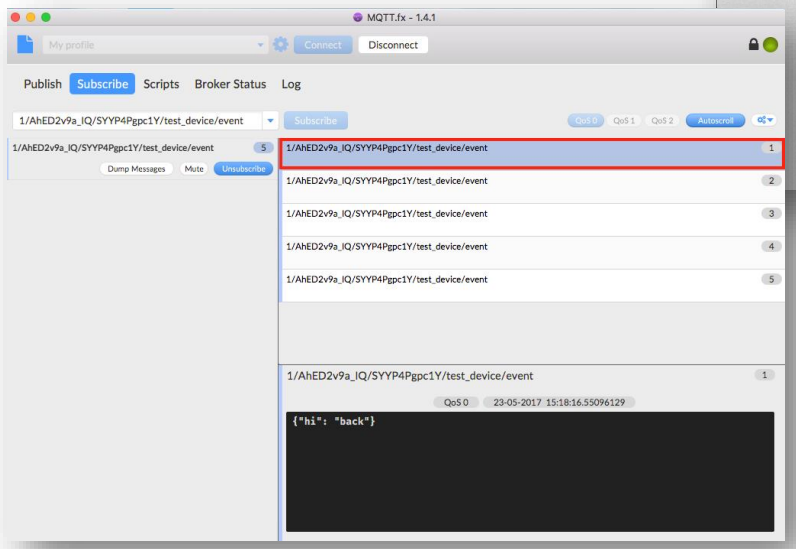
MQTT Brockers in the Cloud

Broker	Free with minimum limitations
ThingMQ	
ThingStudio	X
cloudMQTT	X
IBM Bluemix	X
Heroku	X
Hivemq	
Microsoft Azure IoT	
MQTT.io	

Tools to test MQTT

- https://mqttfx.jensd.de/
- MQTT.fx





MQTT Clients and APIs

- You can develop an MQTT client application by programming directly to the MQTT protocol specification, however it is more convenient to use a prebuilt client
- Client libraries provide some or all of the following:
  - Functions to build and parse the MQTT protocol control packets
  - Threads to handle receipt of incoming control packets
  - QoS 1 and QoS 2 delivery using a local persistence store
  - KeepAlive handling
  - Simple API for developers to use
- Open Source clients available in Eclipse Paho project
  - C, C++, Java, JavaScript, Lua, Python and Go
- Clients for other languages are available, see mqtt.org/software
  - E.g. Delphi, Erlang, .Net, Objective-C, PERL, PHP, Ruby
  - Not all of the client libraries listed on mqtt.org are current. Some are at an early or experimental stage of development, whilst others are stable and mature.