Advanced MQTT For Developers

Jeff Lowrey, IBM

Introduction

- **This Talk IS :**
  - A discussion of advanced topics in MQTT application design and architecture to help you turn an application into a highly scalable, performant and secure solution.
  - A discussion of the on-the-wire structure of MQTT packets.

- **This Talk Is NOT:**
  - A lengthy discussion of MQTT Server internals
  - A lengthy discussion of any particular MQTT Server
  - A lengthy discussion of any particular MQTT Client
  - A comparison of performance between different Clients, Servers, or language bindings

Concepts Covered

- Quality of Service : Performance impact and selection criteria
- Topic Tree design for security, managability and performance
- Connection scaling and pooling/sharing
- Wire level packet information and flow

Concepts not covered

- Client language bindings and available Clients
- Mobile versus web versus embedded devices
- Server management
- Message Ordering

A quick review of basics

- **MQTT is small!**
  - Packets are small, Clients are small.
- **Therefore good for embedded devices and mobile apps**
- **Pub/Sub messaging only**
  - no queue based messaging
  - The MQTT standard does not allow it.
- **Open standard – Official OASIS standard available at**
  http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html
- **Multiple qualities of service – at most once, at least once, once and only once.**
- **Many Client implementations, many Server choices.**

Advanced Application design and architecture

Client ID

- The Client ID is the primary identifier for the Client.
- Every Client must have a unique ClientID.
- A zero length ClientID causes the server to create one, but only with CleanSession = 1 (set).
- The ClientID MUST be used to tie the Client to the session in the Server.
- If a second Client presents the same ClientID on a new connection, the Server MUST disconnect the other session. The server assumes this is the same client reconnecting.

Session State

- The Client can create a new session by asking for CleanSession. Both the Server and Client MUST delete their session. The Server returns a new session with new state. This session lasts until disconnect and state cannot be reused in a new session.
- If you want a new session that you can reuse, you must connect with CleanSession = 1, disconnect, and then connect with CleanSession = 0
- The Client and the Server are BOTH required to maintain the session and it's state. The Session Present tells the Client that it's state is consistent with the Server's state.

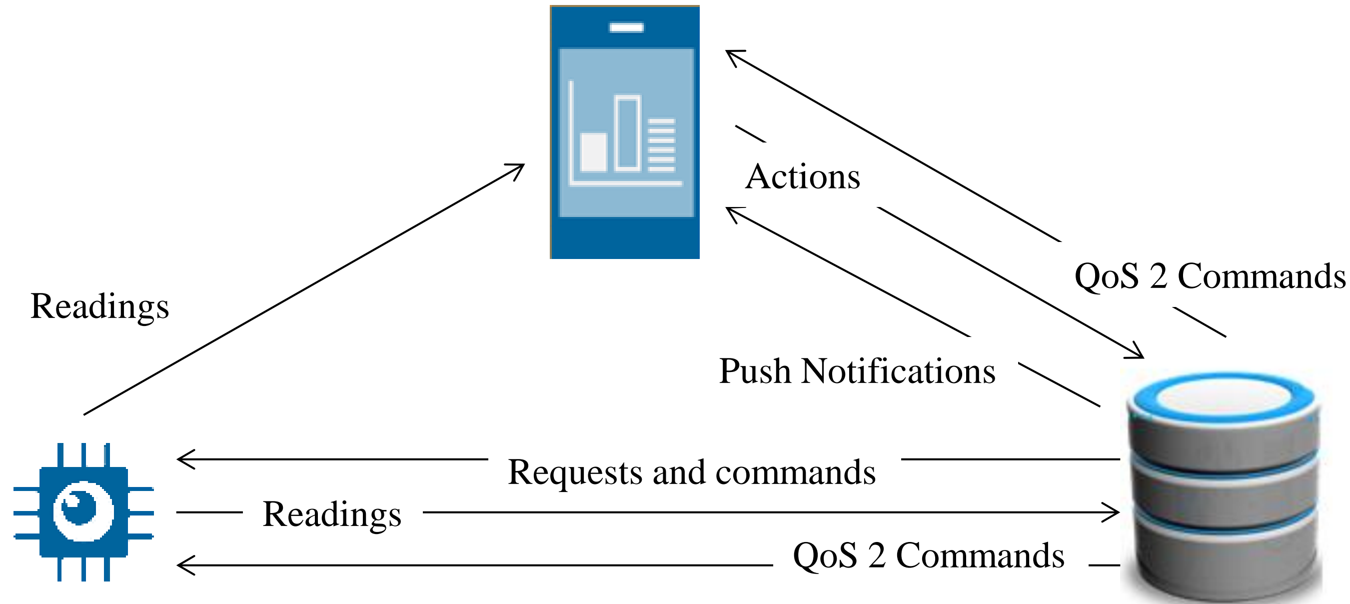
Quality of Service: Performance Impact

- MQTT supports three Qualities of Service – 0, 1, and 2.
- The performance of the different QoSes varies significantly.
- MQTT Servers need to store QoS 1 and 2 messages on disk or other persistent storage. This adds performance overhead.
- This is very similar to MQ, where persistent messages can be much slower than non-persistent messages. And NPMSpeed can speed up non-persistent messages even more.

Choosing and using QoS For your Applications

- Use QoS 0 messages for everything – use application level responses to notify of message receipt or processing.
- Use QoS 1 when QoS 0 is not sufficient.
- Use QoS 2 messages ONLY when you *have* to.
- Design your messages to tolerate QoS 0 and QoS 1 behavior – include historical data, sequence or group information, duplicate messages and etc.
- QoS is negotiated by the Server when a subscription is made. The subscriber requests a specific QoS, and the Server returns what the maximum allowed is. The spec makes no comments on how this decision is made.

Scenario: Sensor, Main Office, Mobile



- **Sensor emits QoS 0 readings and responses to commands and requests**
- **The mobile app uses QoS 0 messages for everything.**
- **The Main Office uses QoS 2 Messages *only* for high priority commands – shutdown, unregister, clear data, warp core ejection, etc.**

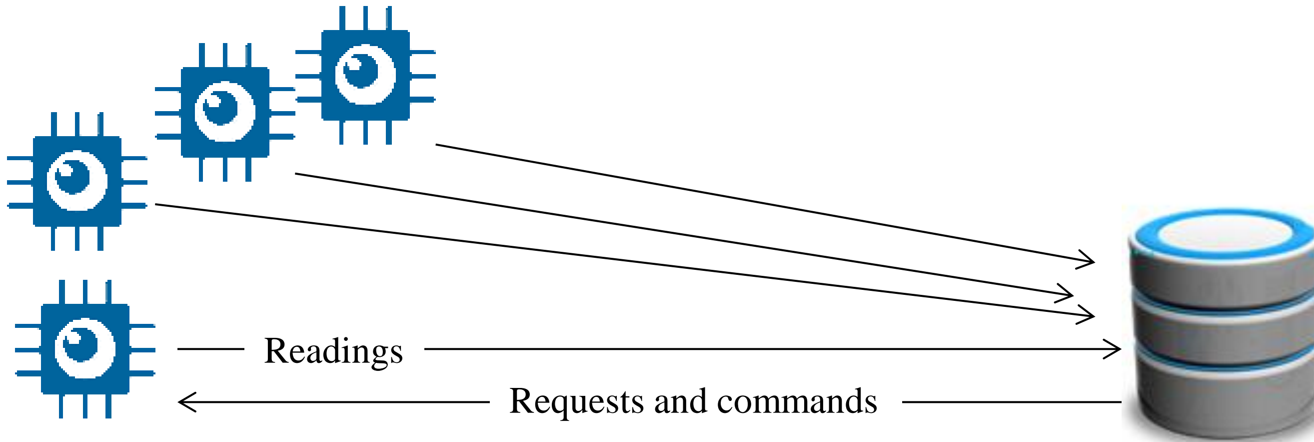
Topic Tree Design
Security considerations

- Your MQTT Server may or may not provide topic level security. Topic Level security is not required by the spec!
- The spec doesn't even require security on the connection – but allows it.
- MessageSight, and Mosquitto provide security on topics, but RabbitMQ does not. Mosqitto requires a plugin.   MessageSight has lots of good information on it's security features and how to use them in the Knowledge Center.
- You should design your topics so that specific parts of the topic tree can be authorized to the right people.
- Example: A set of devices will be subscribing to notifications from a main office. The topic tree should include a device identifier to ensure that each device is only allowed to get updates that belong to it.

Topic Trees and subscription wildcards

- The MQTT standard specifies two wildcard characters : “#” and “+” (in UTF-8). The “#” is a multi-level wildcard and the “+” is a single level. So “Topic/abc/#” will match “Topic/abc/name” and “Topic/abc/def/Client” and etc. “Topic/+/name” will only match “Topic/abc/name”.
- The “#” must be the last character specified in the Topic Filter
- The “+” can be specified at any level in the Topic Filter and can be used more than once.
- Your Topic tree structure should allow you to specify authorization rules based on these wildcards.
- Example: If your topic tree looks like “/App/Customer/CustID/Sensor/SensorType/SensorID”, then you can specify a Topic Filter that limits a specific sensor to both it’s CustID and it’s SensorID using “/App/Customer/CustID/+/+/SensorID”.

Scenario: Topic Security
Sensor to Main Office App



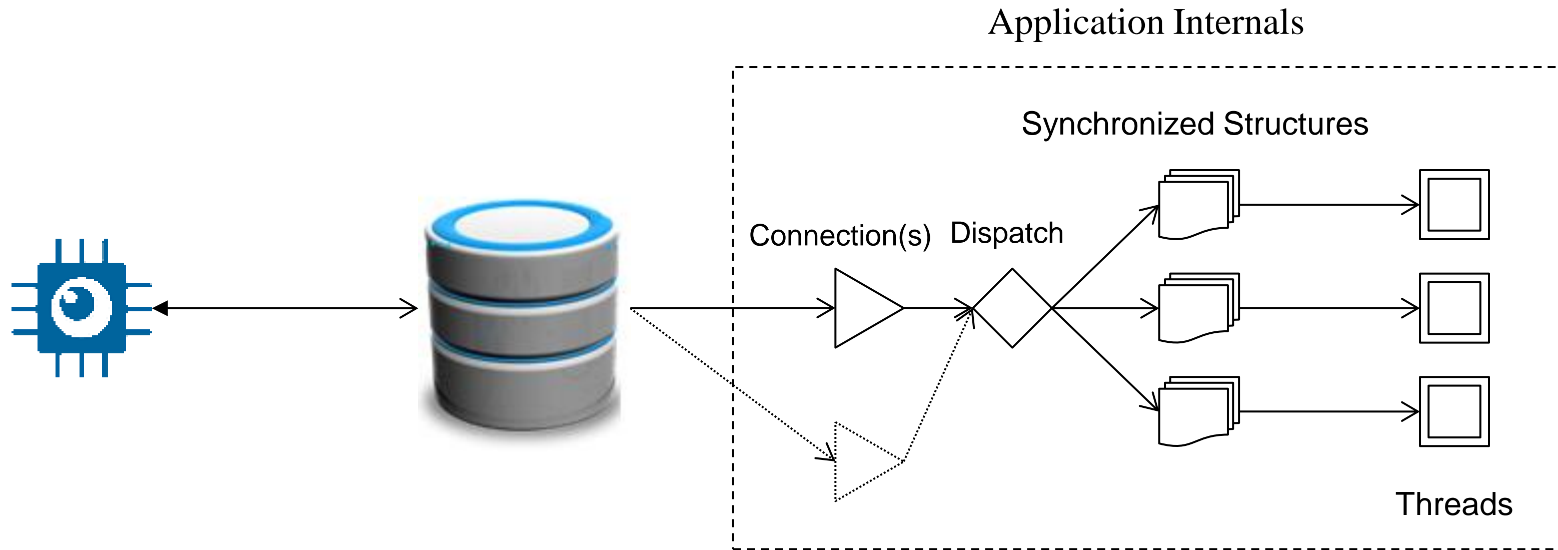
Well Designed Topic Structures allow granular access control:

- /App/Customer/CustID/Sensor/SensorType/SensorID – Root topic
  - ../Reading - any device can publish, only Main Office (and specific mobile devices) can subscribe
  - ../Command – only Main app can publish, Sensors can only subscribe to their specific customer ID and SensorID topic.

Connection scaling and pooling/sharing

- MQTT Connections are full duplex. Publishing and subscribing on the same connection is only a performance consideration when the workload is very high.
- Pooling or dispatching subscriptions to multiple threads will remove the MQTT ability to maintain message ordering. For specific applications, this may be an issue.
- Application logic can use Client id or message data to preserve message ordering.
- Take advantage of the MQTT Client asynchronous dispatch of subscriptions.

Scenario: Connection Scaling



- The sensor uses a single connection to send and receive messages to the main app.
- The main app uses a single connection to receive publications and dispatches them to internal synchronized data structures (queue,stack,etc) based on ClientID or business identifier in message data to preserve message order.
- The main app then processes the messages asynchronously using internal threads.
- If volume of incoming messages grows too large, additional connections can be used to dispatch to new instances of the internal structures.

Questions & Answers

MQTT
On the Wire
Tools, Packet flow, high level
Packet structures

MQTT on the wire - Tools

Some Tools for wire level debugging/non-repudiation (proving “it’s not MQTTs fault!”)

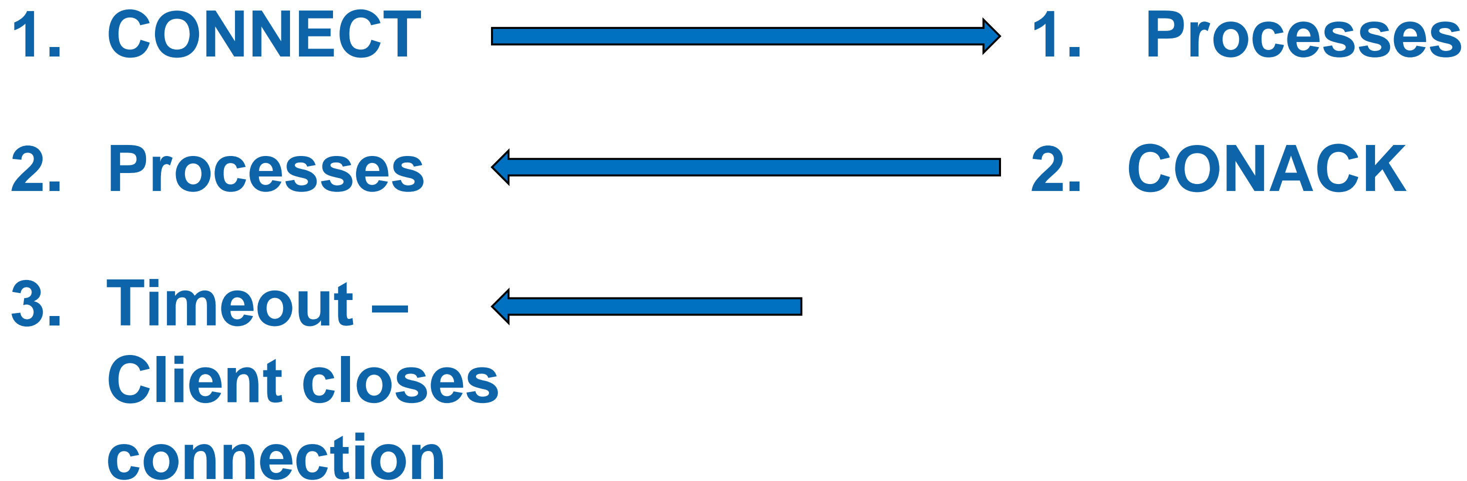
All of these require dealing with SSL first, to state the obvious.

- **Wireshark MQTT Decoder: http://false.ekta.is/2011/06/mqtt-dissector-decoder-for-wireshark/**
- **Node.js MQTT Encode/Decode https://github.com/mqttjs/mqtt-packet**
- **Perl decoder: http://search.cpan.org/~beanz/Net-MQTT-1.130190/bin/net-mqtt-trace**
- **Many more, just search for “mqtt packet decoder”**

MQTT on the wire – Flow of Packets

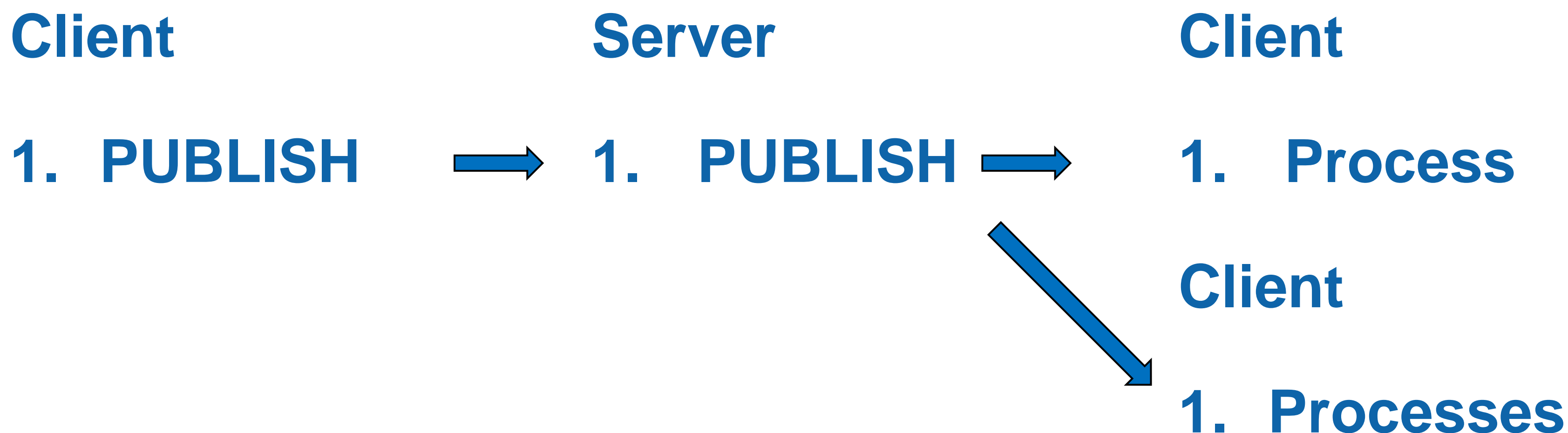
- **Some obvious points:**
  - MQTT packets flow in both directions between Client and Server
  - Only the Client can connect and start packet exchanges.
  - Packet flow is described by the standard.
- **Packets used are : connect, connack, publish, puback, pubrec, pubrel, pubcomp, subscribe, suback, unsubscribe, unsuback, pingreq, pingresp,disconnect**

Connection Packet Flow



CONNACK must be the first packet sent back from the Server. It is up to the Client to decide what a reasonable timeout is.

Publish Packet Flow – QoS 0

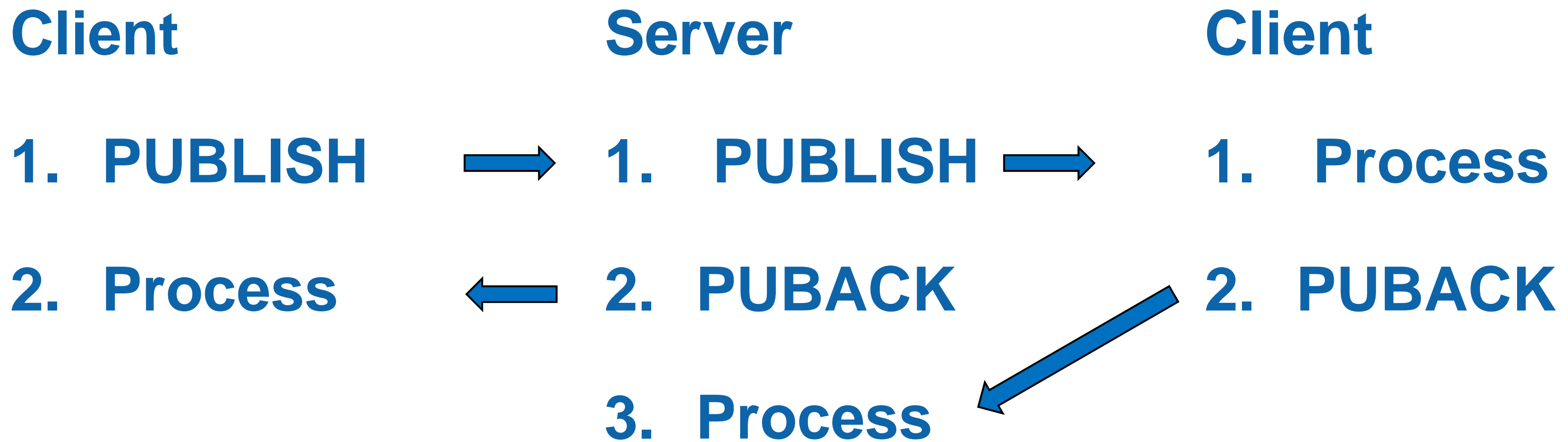


Both Client and Server send PUBLISH packets.

With QoS 0 there is no return packet from PUBLISH.

If Server does not authorize the Client to publish, it must close the connection OR make a positive acknowledgement per QoS rules.

Publish Packet Flow – QoS 1

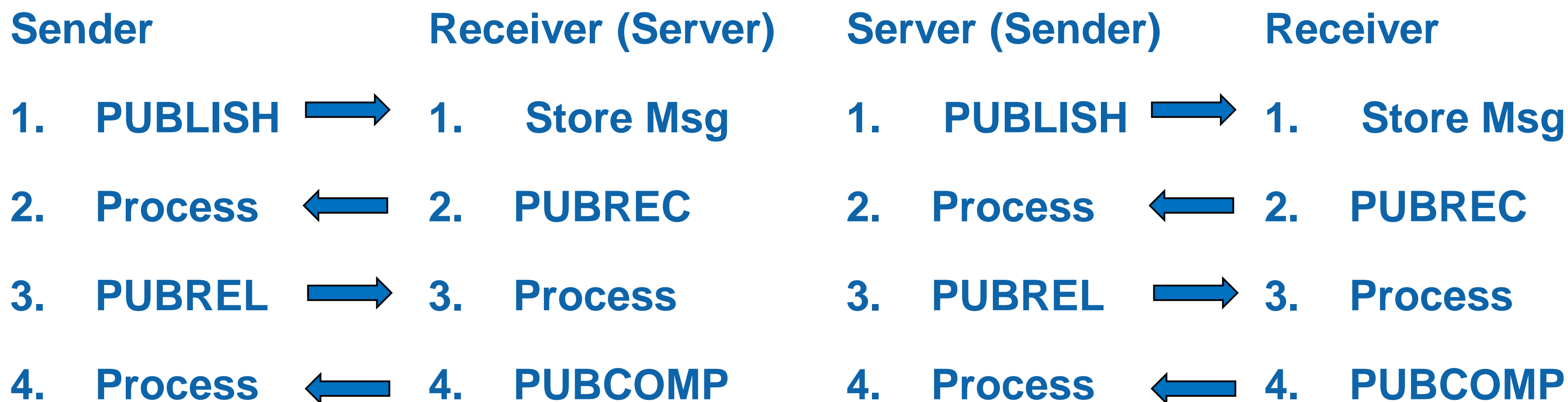


Both Client and Server send PUBLISH packets.

With QoS 1 the Receiver sends back a PUBACK packet.

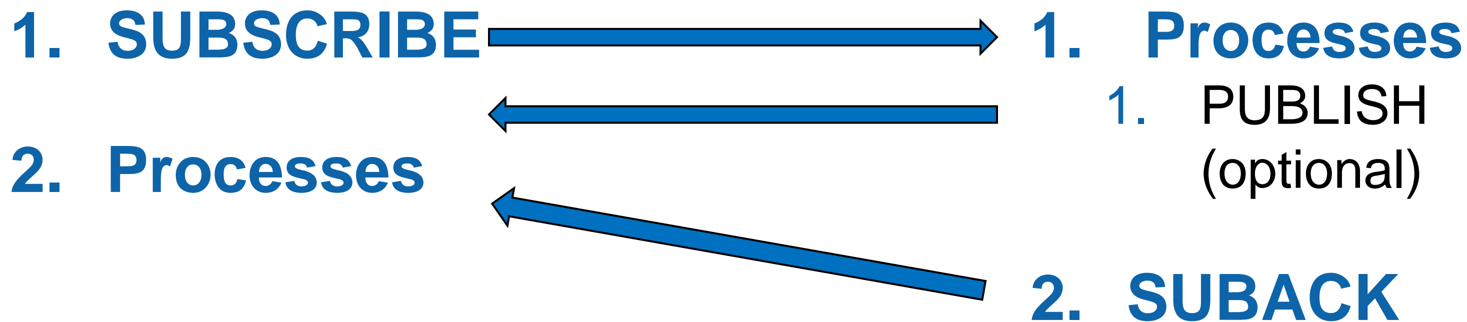
If Server does not authorize the Client to publish, it must close the connection OR make a positive acknowledgement per QoS rules.

Publish Packet Flow – QoS 2



With QoS 2 the Sender emits PUBLISH and PUBREL. The Receiver can either store the message or the packet ID. Each receiver must respond with a PUBREC and then a PUBCOMP. When the sender gets the PUBREC, it should discard the message and store the PUBREC packet ID.

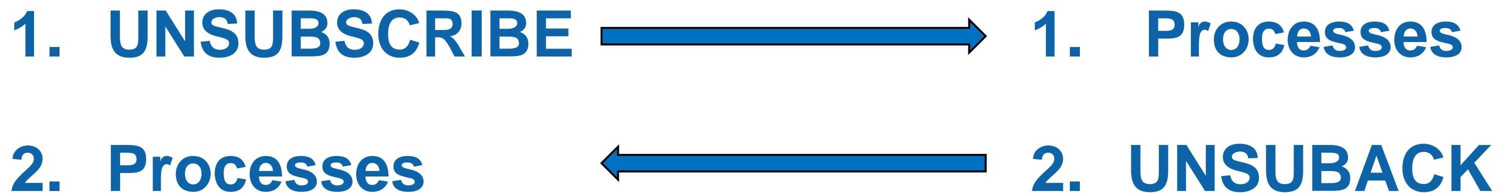
Subscription Packet Flow



SUBACK must be sent. There is no guarantee that SUBACK will come before the first message. QoS is *negotiated* – Client requests a QoS, the Server gives back the maximum allowed.

Each SUBACK contains responses for every Topic Filter/QOS pair in the SUBSCRIBE.

Unsubscribe Packet Flow



UNSUBACK must be sent. The Server removes all Topic Filters that exactly match.

Each UNSUBACK contains responses for every Topic Filter in the UNSUBSCRIBE.

Ping Request Packet Flow

Client		Server
1. PINGREQ		1. Processes
2. Processes		2. PINGRESP

PINGRESP must be sent.

Ping request is used to tell the Server that the Client is alive, tell the Client that the Server is alive, or indicate that the network connection is still active.

Disconnect Packet Flow

Client | **Server**

1. DISCONNECT 1. Processes

No response is sent, no payload is allowed.

The Client closes the network connection.

The Server discards any Will Messages without publishing them. It should close the connection if the Client hasn't done so.

High Level Packet Structures

- Each Packet consists of a fixed header, a variable header and a payload.
- The fixed header is common to all packets but fields vary per packet.
- The variable header and the payload are different for each packet.
- All headers use bit-level fields.
- Payloads vary for each packet – Publish payload is the topic and the application message.

MQTT Fixed Header

- The fixed header consists of two parts: the first byte with flags and the length of the remaining packet data.
- Each byte in the remaining length uses 7 bits and a continuation bit to indicate there's another byte.
- No more than 4 bytes can be used for the remaining length.
- The MQTT Control packet type maps to the individual packets, and both value 0 and value 15 are reserved (so 1-14 are valid packet types)

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							

MQTT Control Packet Flags

- The remaining 4 bits (3 to 0) are used differently for each packet type.
- In MQTT3.1 only the PUBLISH, PUBREL, SUBSCRIBE and UNSUBSCRIBE use any of these bits.
- Publish uses these bits, in order, for the DUP flag, the QoS (2,1) and the Retain flag.
- PUBREL, SUBSCRIBE, and UNSUBSCRIBE use 0,0,1,0 for these bits. Any other values cause network disconnects.

Variable Length Headers

- Each packet can have a variable length header to indicate additional flags about the data.
- Details of the individual fields in these headers are best left to the standards.
- The following packets don't use variable length headers: PINGREQ, PINGRESP, and DISCONNECT.
- The CONNECT packet has the most complex set of flags in the variable length header – indicating everything from the protocol name ("MQTT" in UTF-8) to indicators of what the contents of the payload include. The full set of fields is : Protocol Name, Protocol Level, Connect Flags: {User Name, Password, Will Retain, Will QoS, Will Flag, Clean Session}, and Keep Alive.

MQTT Packet Payload

- The length of the MQTT packet is the bytes of the fixed length header and then the value of the Remaining Length bytes in that header.
- The Remaining Length can only use 4 bytes and can only use 7 bits of those bytes. This limits the size of the message to no more than 250MB (still bigger than MQ!)
- The following Packets have no payload : CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UNSUBACK, PINGREQ, PINGRESP and DISCONNECT (Disconnect doesn't even have a reply message)

Questions & Answers

Thank you!