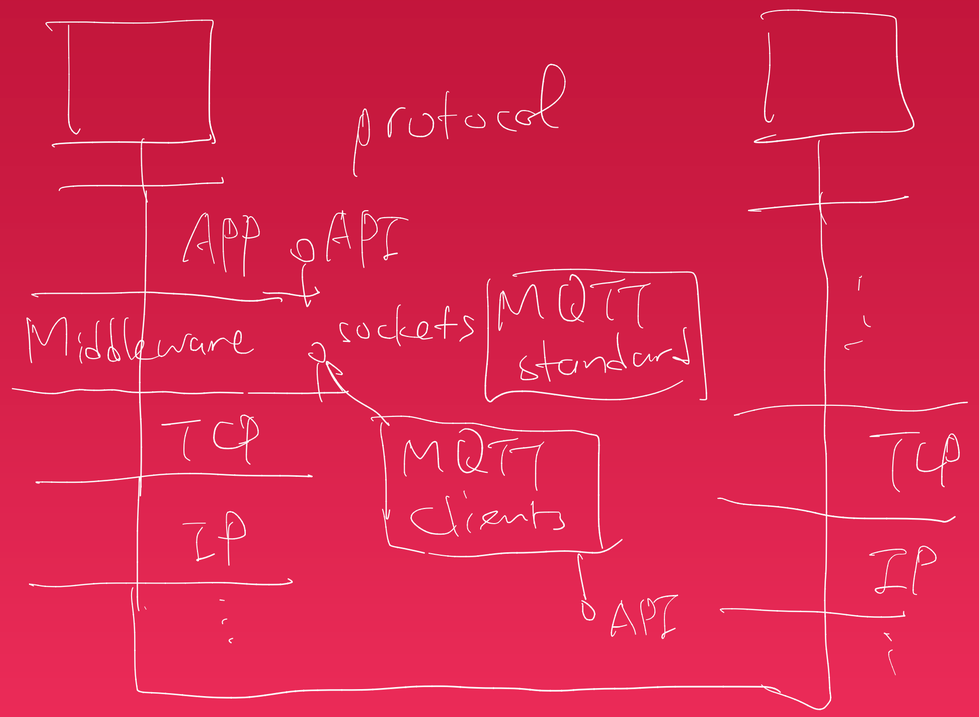
INSTITUT
POLYTECHNIQUE
DE PARIS

MQTT Introduction



Ada Diaconescu

ada.diaconescu@telecom-paris.fr

What is MQTT ?

- Client Server Publish / Subscribe messaging transport protocol
- Characteristics: lightweight (compared to e.g. HTTP) , open, simple, easy to implement (on client side)
- Ideal for constrained environments (small code footprint & limited bandwidth)
  - E.g. Machine to Machine (M2M)
  - E.g. Internet of Things (IoT)
- Homepage: https://mqtt.org

ada.diaconescu@telecom-paris.fr

MQTT History

- Invented in 1999 by Andy Stanford-Clark (IBM) & Arlen Nipper (Arcom → Cirrus Link)
- Needed a protocol for minimal battery loss and minimal bandwidth to connect with oil pipelines via satellite
- Requirements: simple implementation, Quality of Service (QoS) data delivery, lightweight & bandwidth efficient, data agnostic, continuous session awareness
- Focus has shifted from proprietary embedded systems to open IoT
- MQTT acronym
  - Present: simply the protocol name
  - Past: *MQ Telemetry Transport (*based on IBM's MQ Series*)*

ada.diaconescu@telecom-paris.fr

MQTT History (2)

- 1999: proprietary MQTT protocol invention (over TCP/IP)
- 2010: IBM releases royalty-free MQTT v3.1.0
- 2014: MQTT v3.1.1 standard approved by OASIS (open organisation for advancing standards, e.g. AMQP, SAML, ...) & ISO/IEC 20922 → in common use
- 2019: MQTT v5 standard approved by OASIS → currently limited use
  - New features for Cloud platforms
  - More reliability for mission-critical messaging
- 2013: MQTT-SN for Sensor Networks - over UDP, Zigbee, other transports...
  - Suitable for Internet of Things (IoT)

ada.diaconescu@telecom-paris.fr

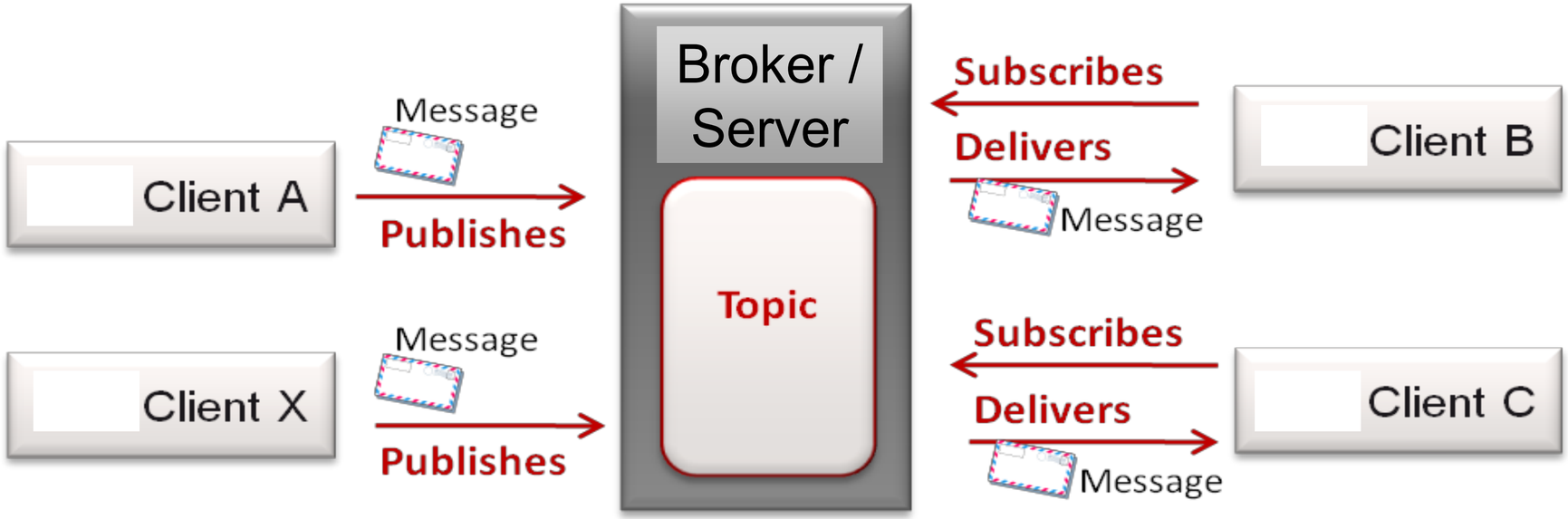
Publish/Subscribe (Pub/Sub) Communication Model

- Alternative to Client-Server model: direct communication
- Pub/Sub decouples message senders and receivers
- Model:
  - Publisher: the Client that sends the message
  - Subscriber: the Client that receives the message

  → Publishers & Subscribers: are never in direct contact; unaware of each-other
  - Broker / Server: third party component that handles Pub/Sub client connections
    - Filter incoming messages from publishers (senders)
    - Distribute messages correctly to subscribers (receivers)

ada.diaconescu@telecom-paris.fr

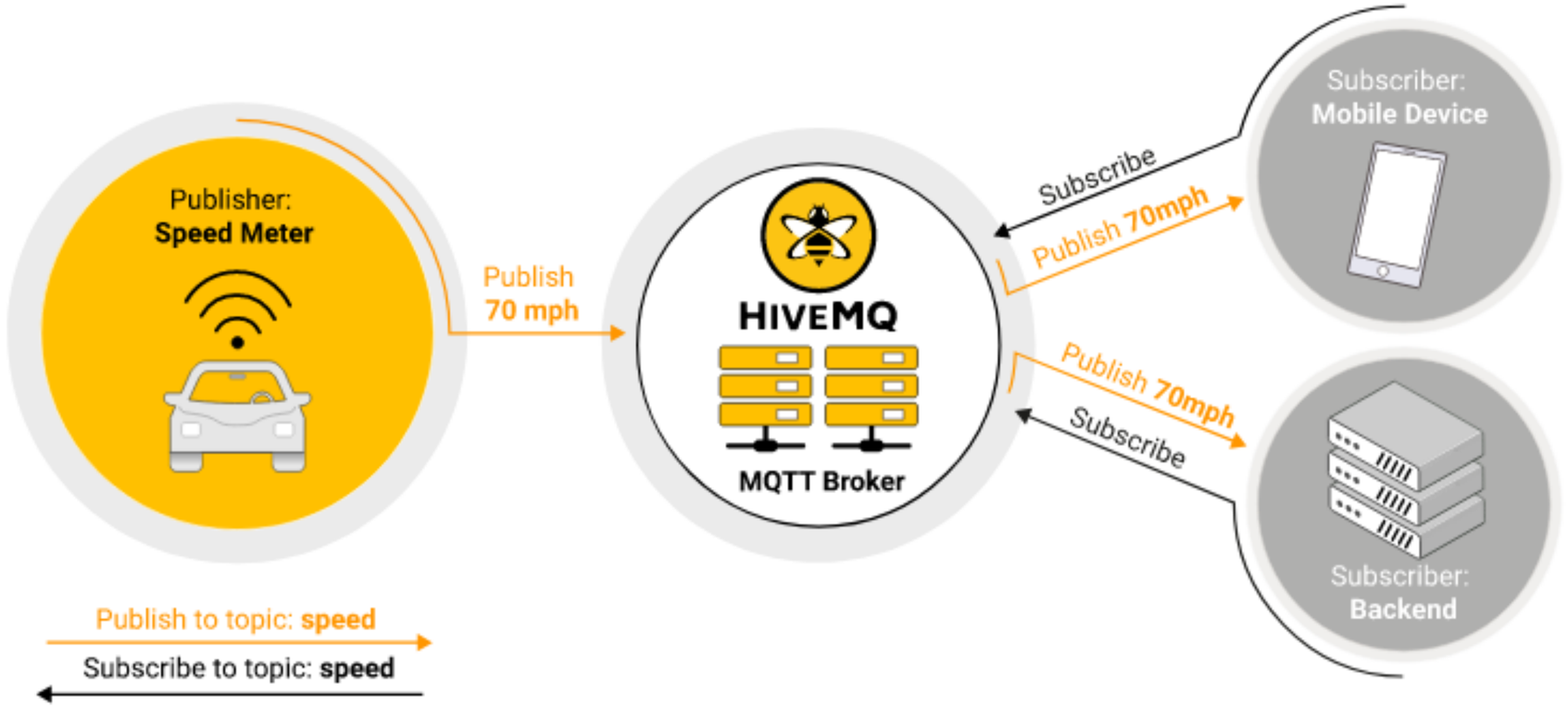
(Pub/Sub) Communication Model (2)



- Space decoupling: clients (Pub/Sub) do not need to know each-other (IP addr. & port)
- Time decoupling: clients do not need to run at the same time
- Synchronisation decoupling: clients not disrupted when sending or receiving messages

ada.diaconescu@telecom-paris.fr

Pub/Sub example → Car Speed Meter



- Image source: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/

ada.diaconescu@telecom-paris.fr

MQTT: differences from Message Queues

- In a Queue
  1. Messages are stored until picked by a client (consumer)

     → A message is always received by a client

  2. A message is only consumed by one client

  3. Queues are named and created explicitly

     → messages can only be published & consumed after the queue is created

- In a Topic (message “subject”)
  1. Messages are transient (i.e. not stored by the server, except for durable subscriptions)

     → A messages may not be received by any client if no prior subscriptions are made

  2. A message is received by all topic subscribers

  3. Topics can be created on the fly

     → topics are created as the messages are published or consumers subscribe

ada.diaconescu@telecom-paris.fr

MQTT Client

- Any device that runs an MQTT library and connects to an MQTT Broker over a network (TCP/IP stack)
  - E.g. A small resource-constraint device connected to a Broker via wireless network
  - E.g. A classic computer running an MQTT Client GUI for testing purposes – e.g. MQTT Spy
- Publisher and/or subscriber
- Libraries for e.g., Android, Arduino, C, C++, C#, Go, iOS, Java, JavaScript, .NET.
  - Complete list: https://github.com/mqtt/mqtt.org/wiki/libraries
  - Eclipse Paho project – open-source MQTT clients: https://www.eclipse.org/paho/

ada.diaconescu@telecom-paris.fr

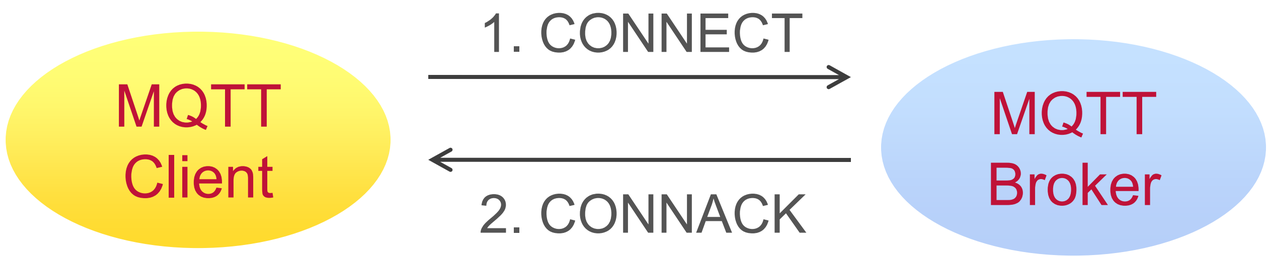
MQTT Broker

- Provides support for
  - Receiving messages
  - Filtering messages
  - Determining the subscribers of each message
  - Sending the message to these subscribers
  - Manage session data for clients with persistent sessions (e.g. subscriptions, missed messages)
  - Client authorisation and authentication
- Provider-specific properties
  - Scalability (e.g. millions of concurrent clients)
  - Failure-resistance
  - Integration into backend systems
  - Monitoring, ...
- Implementations
  - Mosquitto – https://mosquitto.org/
    - Free , open-source (most popular)
    - Windows & Linux
  - Eclipse – Paho MQTT
    - Free
  - HiveMQ – http://www.hivemq.com/
    - Commercial
  - **JoramMQ** – http://jorammq.com
    - Commercial
- Usage
  - Own locally installed Broker / Server
  - Cloud-based Server or Virtual Server
  - Shared Server Application

ada.diaconescu@telecom-paris.fr

MQTT Connection

- MQTT: based on TCP/IP
  - Both Client and Broker need to have a TCP/IP stack
- Initiate Connection:
  1. Client sends Broker a CONNECT message (including KeepAlive parameter)
  2. Broker responds with CONNACK message & status code
  - → Broker keeps connection open until Client sends disconnect command or connection breaks (i.e. if Client fails to send PINGREQ before max delay, set via KeepAlive param.)



ada.diaconescu@telecom-paris.fr

CONNECT message → packet contents

- Client ID
  - Unique per client & broker (e.g. “Client-A”) – duplicate IDs cause disconnection of older Clients
  - Can be empty for stateless clients (must use true “clean session” flag)
- Clean session (flag) clean session = T：新会话，断开不保留信息
  - Persistent session: CleanSession = false

    → broker stores all client’s sessions & non-acknowledged messages (if QoS > 0)
- Username/Password (optional)
  - Authentication (messages should be encrypted, e.g. TSL)
  - Alternative: SSL certificates
- Will message (optional)
  - Notifies other clients when a client disconnects ungracefully (e.g. failure)
- Keep Alive
  - Longest period (seconds) with no communication between Client & Broker (e.g. PING request/reply)

CONNACK message → package contents

- Session present (flag)
  - Does the Borker already have a persistent session for the client?
    - Clients can know whether they are already subscribed to topics
    - Always: SessionPresent = false IF CleanSession == true
- Connect return code
  - Was the connection attempt successful or not?

Return Code	Meaning
0	Connection accepted
1	Connection refused, unacceptable protocol version
2	Connection refused, identifier rejected
3	Connection refused, server unavailable
4	Connection refused, bad user name or password
5	Connection refused, not authorized

ada.diaconescu@telecom-paris.fr

PUBLISH

- Clients can publish messages after connecting to a Broker
- Client is unaware of how many subscribers there are for a topic
- Each message must contain a topic – to determine the corresponding subscribers
- Packet Contains:
  - packetID (e.g. 1234) – only relevant if QoS > 0; use 0 value otherwise
  - topicName: hierarchically structured String (e.g. “work/myOffice/temperature”)
  - QoS: quality of service levels (i.e. 0, 1 or 2)
  - retainFlag: does the Broker save this message as the topic’s last valid value?
  - payload: message content in byte format (client may use text, binary, xml, json, ...)
  - dupFlag: is this a duplicate message? (e.g. lost connection between publish & ack., QoS>0)

ada.diaconescu@telecom-paris.fr

SUBSCRIBE

- To receive messages published on a topic, clients must subscribe to that topic
  - Clients only receive messages published to a topic *after* they subscribe to that topic
- SUBSCRIBE message (packet contents):
  - packetId (e.g. 1234)
  - List of subscriptions
    - qos<n>
    - topic<n>: subscription topic – can contain wildchars
  - E.g. qos1,topic1, qos2,topic2,....
  - E.g. 1,”topic/1”,0,”topic/2”,....

ada.diaconescu@telecom-paris.fr

SUBACK

- Broker acknowledges a client's subscription
- If Successful, the client receives all messages published on that topic
- Package content:
  - packageId: the same as for the Subscribe message
  - List of return codes: for each topic from Subscribe, in the same order

Return Code	Meaning
0	Success - Maximum QoS 0
1	Success - Maximum QoS 1
2	Success - Maximum QoS 2
128	Failure

ada.diaconescu@telecom-paris.fr

UNSUBSCRIBE

- Deletes a Client's indicated subscriptions with the Broker
- Packet contents
  - packetId
  - List of topics: to unsubscribe from (e.g. "topic/1","/topic/2",...)

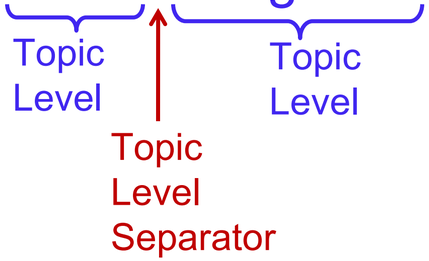
UNSUBACK

- Broker confirms unsubscription
- Pakcet contents
  - packetId: same as unsubscribe message

ada.diaconescu@telecom-paris.fr

Topic

- UTF-8 String, hierarchically structured by a forward slash “/” (topic level separator)
  - E.g. “home/livingroom/light-sensor/1”



- The Broker uses topics to filter messages to connected clients (subscribed)
- Clients do *not* need to create a topic before publishing or subscribing to it
- Wildcards can be used to subscribe to topics (not to publish)
  - Single-level: home/+/light-sensor/1
  - Multi-level: home/livingroom/#

Wildcard (matched by any string)

Wildcard (only at the end; matched by any structured string)

System Topics

- Beginning with “$”
  - Often used for the internal statistics of the MQTT Broker
  - Common topic: $SYS/#
- Not included when subscribing to wildcard topic “#”
- E.g.,
  - $SYS/broker/clients/connected
    $SYS/broker/clients/disconnected
    $SYS/broker/clients/total
    $SYS/broker/messages/sent
    $SYS/broker/uptime

ada.diaconescu@telecom-paris.fr

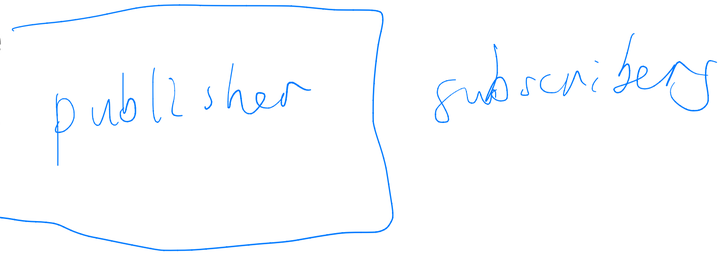
Topics – Best Practices

- **Do *not* use a leading forward slash**: **wasted topic level**
- **Avoid spaces in a topic**: **harder to read & debug**
- **Use concise & precise topic names**: **minimise network load**
  → MQTTv5 uses numeric aliases for topic names
- **Use only ASCII characters (printable)**: **easier to debug**
- **Don’t subscribe to all topics (#)**: **scalability issues**
- **Consider future extensibility**: **how may your topics change?**
- **Use specific topics, rather than generic ones**: **e.g. 1 topic per sensor type**

ada.diaconescu@telecom-paris.fr

Quality of Service (QoS)

- Agreement between a message sender and a message receiver
  - Hence between Publisher (P) and Broker (B) and/or between Broker (B) and Subscriber (S)
  - End-to-end QoS (Publisher to Subscriber) involves both parts (P → B & B → S)
- Defines the guarantees of message delivery
- 3 QoS levels in MQTT:
  - 0: at most once
  - 1: at least once
  - 2: exactly once

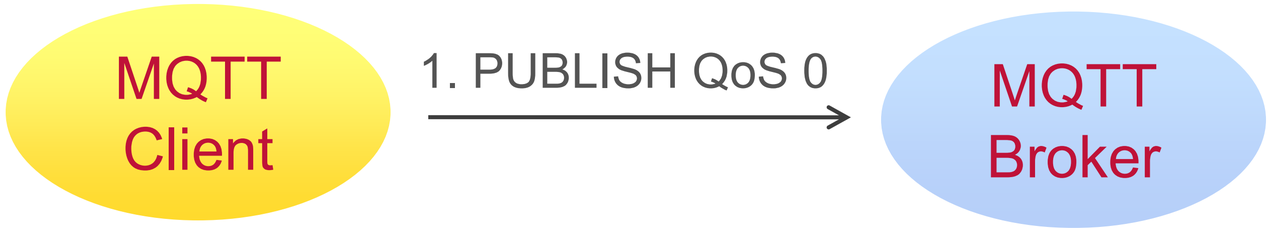


- Concerns two sides of message delivery
  - From Publisher to Broker: QoS defined by the publisher (in PUBLISH message)
  - From Broker to Subscriber: QoS defined by subscriber (in SUBSCRIBE message)
  - → if Subscriber QoS is lower than Publisher QoS, the broker uses the lower QoS

ada.diaconescu@telecom-paris.fr

QoS 0: deliver at most once

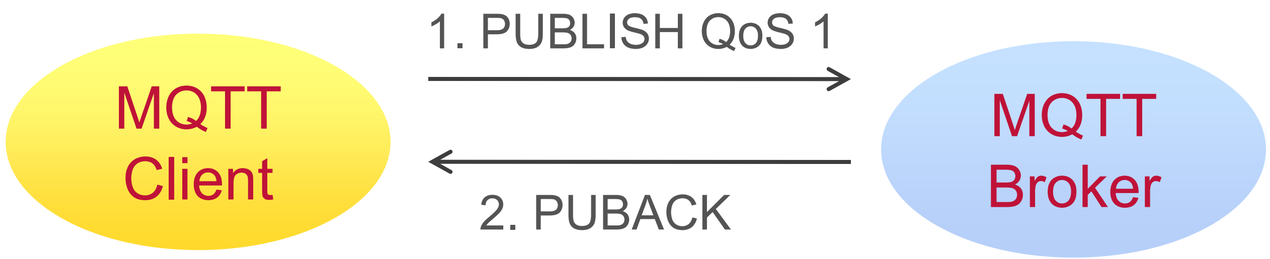
- Best effort delivery – no guarantee, “fire & forget”, same QoS as TCP/IP
- The recipient does *not* acknowledge message reception
- The message is *not* stored & resent by the sender



ada.diaconescu@telecom-paris.fr

QoS 1: deliver at least once

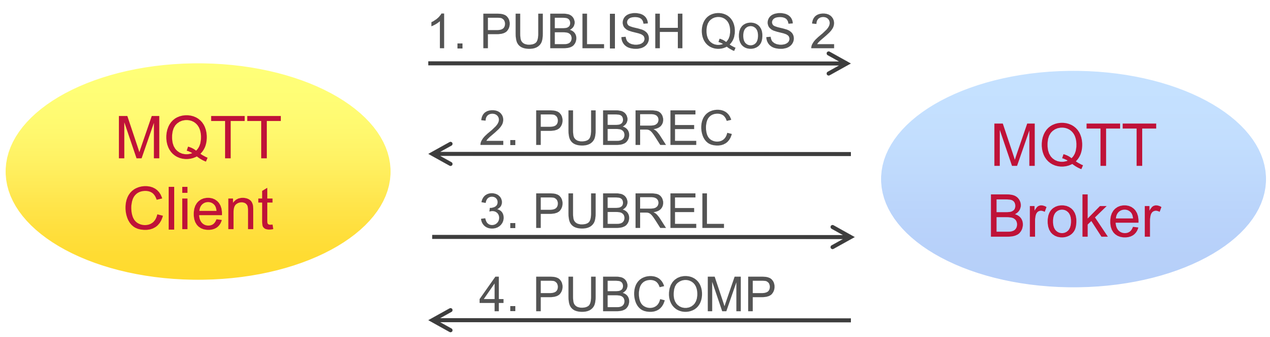
- Guarantees that the message is delivered at least once to the receiver
- The sender stores the message until it gets a PUBACK packet from the receiver
  - Uses the packetId to match the published packet to the acknowledgement
- The sender resends the packet if it does not receive a PUBACK within a delay; or if the connection was lost before the PUBACK arrived
  - Sets the duplicate flag DUP (not processed by broker or subscriber in QoS 1)
- A message may be delivered multiple times



ada.diaconescu@telecom-paris.fr

QoS 2: deliver exactly once

- Guarantees that a message is received once and only once by intended recipients
  - Via 2x request/reply exchanges (4 part handshake) between sender & receiver
  - Uses the packetId of the initial message (e.g. PUBLISH)
- The sender resends the packet if it does not receive a PUBREC within a delay
  - Sets the duplicate flag DUP
- The receiver discards duplicates and sends a PUBCOMP message after receiving a PUBREL message
  - Sender discards all duplicates upon receiving PUBCOMP message
- Safest & slowest QoS level



ada.diaconescu@telecom-paris.fr

QoS Best Practices

- How to select the “correct” QoS level? → Application & use-case dependent
- QoS 0
  - Mostly stable network connection (e.g. wired)
  - Acceptable to lose some messages (e.g. frequent measurements)
  - No need to queue messages (i.e. done for QoS 1 or 2 ; and persistent session)
- QoS 1
  - Need to receive every message & can handle duplicates
  - Cannot support the overhead of QoS 2 (i.e. faster message delivery with QoS 1)
- QoS 2
  - Critical to receive all messages exactly once (i.e. duplicate delivery can harm recipients)
  - Can support overheads and delays in message delivery

ada.diaconescu@telecom-paris.fr

Persistent Session

- The Broker stores information for an offline client; available when client reconnects
  - Session existence (even if no subscription)
  - Client's subscriptions
  - All messages with QoS 1 or 2 that the client has not yet completely confirmed
  - All new messages with QoS 1 or 2 that the client missed when offline
- Clients can demand a persistent session when connecting to the Broker
  - cleanSession == true → non-persistent session;
    → all info is lost if the client disconnects
  - cleanSession == false → persistent session
    → all info is stored until the client requests a cleanSession
- Clients with persistent sessions must also store information (used upon reconnection)
  - All messages with QoS 1 or 2 that are not yet confirmed by the Broker

ada.diaconescu@telecom-paris.fr

Best Practices – Persistent or Clean Session

- Persistent session
  - The Client must receive all messages from a topic, even if offline sometimes
  - The Client has limited resources; the Broker can re-establish communication quickly
  - The Client needs to resume all messages with QoS 1 & 2 after reconnect

- Clean session
  - The Client is only a publisher (not a subscriber)
  - The Client does *not* need to get messages that it misses when offline

ada.diaconescu@telecom-paris.fr

Retained Messages

- A message published with flag **`retain = true`**
- The Broker stores the last retained message and its QoS for a topic
  - A single retained message (i.e. the last one) is stored for each topic
- A Client that subscribes to a topic receives the retained message from that topic
  - Allows subscribed Clients to get a status update immediately (*last known good value*)
- Each new retained message replaces the old one on the same topic
- To delete a retained message, send a retained message with zero payload to the targeted topic

ada.diaconescu@telecom-paris.fr

Last Will & Testament (LWT)

- Unreliable networks, device batteries, ... → ungraceful disconnects of Clients
  - Clients can specify their LWT when connecting to the Broker
  - CONNECT message: lastWillTopic, lastWillQoS, lastWillMessage, lastWillRetain
- LWT allows to notify other Clients about a Client's ungraceful disconnect
  - MQTT message with a topic, retain flag, QoS and payload
- The Broker stores the LWT message until the Client disconnects ungracefully ; then sends the LWT to all Clients subscribed to that topic
  - Broker detects IO error or network failure, no communication within Keep Alive period, connection closed without DISCONNECT message, protocol error
- If the Client disconnects gracefully (i.e. DISCONNECT message) then the Broker discards the LWT message
  - MQTTv5: Client may send DISCONNECT message & ask that the WILL be delivered

Keep Alive

- MQTT relies on Transmission Control Protocol (TCP)
  - Reliable, ordered, error-checked packet transfer over the Internet
- If one communication end fails (e.g. Mobile networks) → half-open connection
  - The functioning end keeps sending messages and waits for acknowledgement
- Keep Alive: assess if a connection is still open (between a Client and a Broker)
- A Client specifies the Keep Alive interval upon connection to the Broker
  - Maximum interval with no communication (client-dependent: min. 0, max. 18h12min15 sec)
  - Client must send at least a PINGREQ packet if it has nothing else to communicate
  - The Broker responds with PINGRESP packet to confirm its availability to the Client
  - Broker disconnects Client if it receives no message within 1.5*KeepAlive interval

ada.diaconescu@telecom-paris.fr

MQTT over Web Sockets

- Send & receive MQTT messages to a browser
  - Any browser that supports WebSockets can be an MQTT Client
  - Needs
    - JavaScript library that enables MQTT over WebSockets – e.g. https://eclipse.org/paho/clients/js/
    - Broker that supports MQTT over WebSockets
- WebSocket
  - Network protocol: bi-directional, ordered, lossless comm. between a Browser & a Web Server
  - Standardised in 2011; supported by all modern browsers
  - Based on TCP
- MQTT over WebSocket
  - MQTT message is encapsulated within one or several WebSocket frames
  - Browsers do not yet support Socket API (direct TCP connections)
  - May use secure WebSockets via Transport Layer Security (TSL) to encrypt messages

JoramMQ

- Developed by: ScalAgent D.T. , Grenoble, France
- Commercial : based on OW2/Joram 5.17, open-source broker (LGPL)
- Java implementation
- http://jorammq.com JoramMQ

ada.diaconescu@telecom-paris.fr

QUESTIONS?

THANK YOU !

MQTT: Useful Information in a Nutshell

- Default MQTT port: 1883
- Used transport protocol: TCP/IP
  - Other protocols available for MQTT-SN (e.g. UDP, Zigbee)
- Data format for communication: binary
- Message encryption: not by default
- Each Client connected to a Broker must have a unique ClientId
- Is it possible to know the identity of a publisher Client? No
  - Unless the publisher includes its Id in the message payload
- Can I get a list of published topics? Not easily – the Broker does not keep a list
  - Topics aren't permanent – if a topic has no subscribers the Broker discards messages published on this topic

ada.diaconescu@telecom-paris.fr

MQTT Resources

- MQTT essentials: https://www.hivemq.com/mqtt-essentials/
  - Text & Video formats
- MQTT Beginners Guide: http://www.steves-internet-guide.com/mqtt/
  - Related MQTT tutorials: http://www.steves-internet-guide.com/category/mqtt/
- MQTT Protocol homepage: https://mqtt.org/
- Eclipse IoT: http://iot.eclipse.org/getting-started
- MQTT Brokers/servers:
  - https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations
  - http://www.steves-internet-guide.com/mqtt-hosting-brokers-and-servers/