

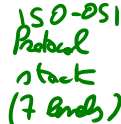
Message Passing Protocols

Tullio Facchinetti
<tullio.facchinetti@unipv.it>

17 maggio 2023

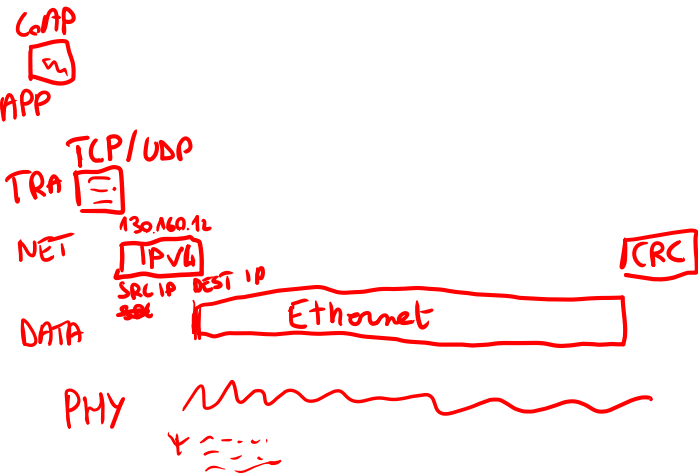
<http://robot.unipv.it/toolleeo>

Wi-Fi (MAC)
STA AP



MAC!

Source: E. Al-Masri et al., "Investigating Messaging Protocols for the Internet of Things (IoT)", in IEEE Access, vol. 8, 2020.



Comparison of some data link layer protocols

IoT network = heterogeneous ~~and~~ existence of many protocols

Technology	Throughput (Approximate) (Kbps)	Range (Approximate) (m)	Mobility Support
NFC	424	0.1	Yes
ANT+	1,000	50	Yes
ZigBee	250	100	Yes
Z-Wave ¹	40	100	Yes
Bluetooth ²	1,000	100	Yes
WiFi ³	54,000	150	Yes
WirelessHART	250	150	Yes
Weightless-W	10,000	5,000	Yes
LTE-M	1,000	11,000	Yes
LoRaWAN ⁴	0.3 ⁴	14,000	Yes
Sigfox ⁵	0.1	17,000	Yes
NB-IoT	200	20,000	No ⁵

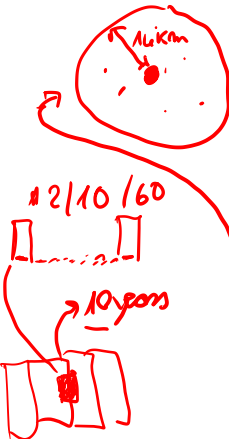
¹ outdoor or open air; indoor is approximately 50m

² data rate may vary depending on the deployed region (up to 600 bps)

³ Bluetooth 5 can support a range of approximately 150m (outdoor) with up to 8x broadcasting capacity

⁴ range up to 50Kbps if using Frequency Shift Keying (FSK) instead of LoRa

⁵ minimal, no full support for mobility as in LTE (possibly during cell reselection - idle state)



WSN

TSCH (IEEE 802.15.4)

12.7 Mbps

50 m

> 56 Kbps
800 Mbps Wi-Fi 7 50



IoT application range requirements



50 microcontroller

Because of the diversity of IoT devices, there exists no single communication technology that is capable of supporting heterogeneous environments.

CAN
TSN
(Ethernet)
(deterministic)

Application	~ Range	Technology
• Industry Automation	10m - 50m	LoRa, ZigBee, WirelessHART
• Smart Metering	15km - 40km	<u>LoRa</u> , Weightless-N
• Smart Buildings	10m - 250m	LoRa, Sigfox
• Asset Tracking	50m - 500m	LoRa, Sigfox, Weightless
• Smart Energy	100m - 15km	LoRa
• Environmental Monitoring	100m - 1.5km	LoRa, Sigfox
• Health Monitoring	10m - 25m	BLE, LoRa, ZigBee, ANT+
• <u>Wearable & Fitness</u>	30m-50m	ANT+, BLE
• Consumer Electronics	10m-25m	ZigBee, Z-Wave, BLE

Latency

Wi-Fi

Specifications, standards and alliances



Standards' governing bodies and alliances that have been formed for enhancing communication technologies for the IoT landscape.

Specifications, standards and alliances: goals

- Initiatives, specifications and standards have different focuses and target specific stakeholders or markets.
- Some initiatives address challenges for Business to Consumer (B2C) or Business to Business (B2B) applications, others were developed to accommodate specific vertical or horizontal domains in the IoT landscape.

Specifications, standards and alliances: goals

Examples:

- E.g., IEEE, ZigBee Alliance, ISO, CEN and ULE are all organizations or alliances that have proposed standards or specifications for a vertical domain that primarily focus on solving a very specific area such as home and building automation.
- IEC, ISO, oneM2M, OPC and OpenIndustry 4.0 Alliance provide specifications or recommendations that are domain-specific or solve problems within the manufacturing and industrial automation vertical domain.
- W3C, ITU, OASIS, OMG, IETF and HyperCat provide standards, specifications and recommendations for a broader support of a number of IoT applications while encompassing many different domains.

Characteristics of initiatives or standards

- 1 ① Architecture
- ② Communication
- ③ Security and privacy
- ④ Interoperability
- ⑤ Integration
- ⑥ Device types and sensor technology
- ⑦ Deployment models
- ⑧ Services' provisioning
- ⑨ Application and device management

• Safety

• Real-time

Some features are also supported by a number of protocols that exist across the link and application layers.

• Reliability

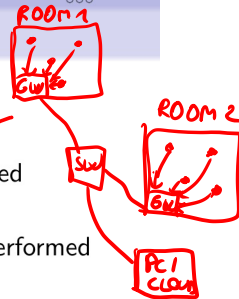
Guidelines to select a message passing protocol

- What are the system requirements and challenges that may influence choosing an application protocol for IoT development?
- What is the extent of the coverage of these challenges in existing literature?
- Which communication types are covered by existing application layer protocols?
- What factors were used or applied in conducting prior research studies?
- What is the depth of the examined literature in terms of coverage, comprehensibility and technical knowledge?
- What is the adoption rate of the existing protocols used for IoT applications?

Types of IoT Environment

• Device-to-human / human-to-device

- 1 Device-to-Device (D2D): the communication is provided between two nodes or devices directly.
- 2 Device-to-Application (D2A): the communication is performed between devices and an IoT application.
- 3 Device-to-Gateway (D2G): the communication is provided through a gateway that resides in close proximity to the edge of the network while interacting with IoT devices.
- 4 Device-to-Cloud (D2C): the communication is achieved directly between IoT devices and cloud service providers.



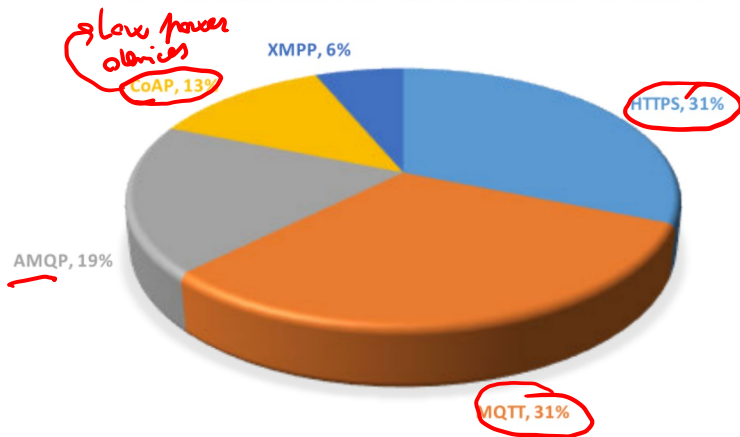
Source: Souri et al., "A systematic review of IoT communication strategies for an efficient smart environment", Transactions on Emerging Telecommunications Technologies, Aug. 2019.

Support by platform providers

IoT platform	Year F.A.	Protocols
<u>Azure</u> IoT Hub	2014	HTTP(S), MQTT, MQTT over WebSocket, AMQP, AMQP over WebSocket, custom protocols via gateway
<u>Google</u> IoT Core	2018	HTTP(S), MQTT, custom protocols via gateway
<u>IBM</u> Watson IoT	2014	HTTP(S), MQTT, MQTT over WebSocket, WebSocket
AWS IoT Core	2015	HTTP(S), MQTT, MQTT over WebSocket
Alibaba IoT	2015	HTTP(S), CoAP, MQTT, MQTT over WebSocket, WebSocket, network types: 3G, 4G, NB-IoT, LoRa
<u>Oracle</u> IoT	2016	HTTP(S), CoAP, MQTT, AMQP, XMPP, WebSocket
<u>Siemens</u> MindSphere	2016	HTTP(S), CoAP, MQTT, AMQP, XMPP, thorough gateways: OPC UA, LoRaWAN, Modbus, 6LoWPAN, LwM2M
Bosch IoT Hub	2017	HTTP(S), MQTT, AMQP, LoRaWAN
<u>Cisco</u> Kinetic	2017	HTTP(S), MQTT, AMQP, WebSocket, custom protocols via gateways
Eclipse Hono	2018	HTTP(S), CoAP, MQTT, AMQP, custom protocols via gateways

Support by platform providers

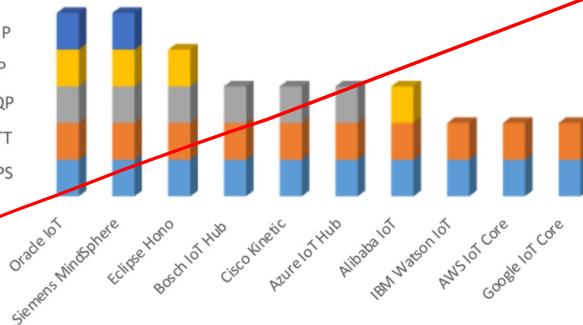
IOT PLATFORM SUPPORT FOR MESSAGING PROTOCOLS



Support by platform providers

Comparison of Messaging Protocol Support by IoT Platforms

■ XMPP
■ CoAP
■ AMQP
■ MQTT
■ HTTPS



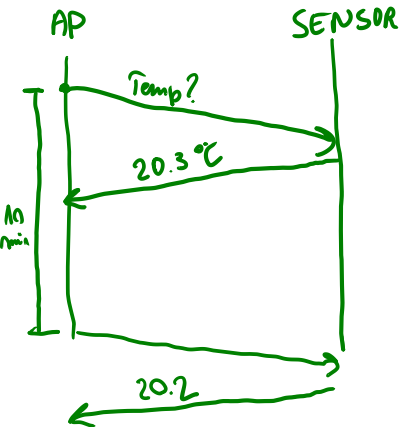
Constrained Application Protocol (CoAP)

- Web transfer protocol intended for devices running on **constrained networks** (e.g., low-power, lossy).
- Designed for Machine-to-Machine (M2M) applications, e.g. factory automation, smart energy.
- **Request-response** interaction model. (Other: Publish / Subscriber)
- Uses **major concepts from the web** such as Uniform Resource Identifiers (URIs) and Internet media types.
- Used over the UDP transport protocol using the *coap* scheme and over **Datagram Transport Layer Security** (DTLS) using the *coaps* scheme.
- Defined in RFC 7252 (and several extensions).

CoAP aims to **bridge HTTP and RESTful services** through simple interfacing

18.
15:40

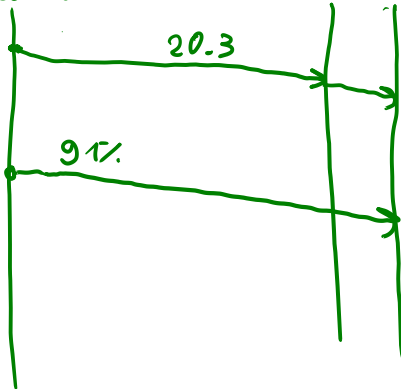
Request/Response



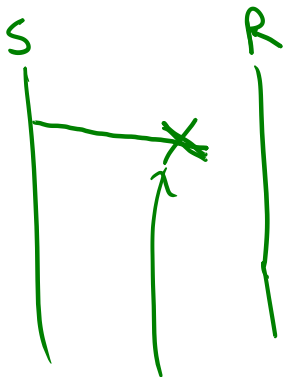
Publish/Subscriber (temp) (temp, hum)

SENSOR

SUB1 SUB2

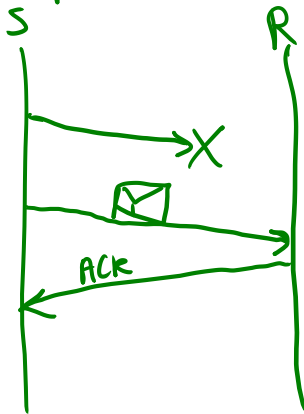


UDP (Not confirmed)

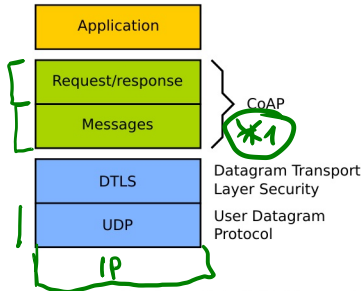


The sender doesn't know that the message was lost

TCP



CoAP stack

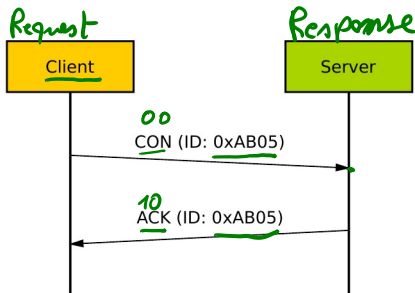


- The **Messages layer** deals with UDP and with asynchronous messages.
- The **Request/Response layer** manages request/response interaction based on request/response messages.

Types of messages *1

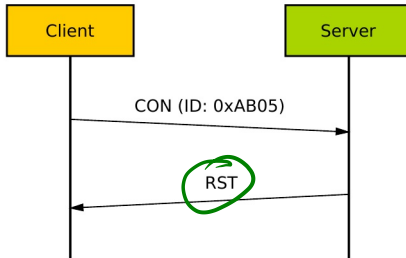
Message type	Bits	Code	Description
<u>Confirmable</u>	<u>00</u>	CON	An acknowledgement is required, this improves the reliability of the UDP protocol
<u>Non-confirmable</u>	<u>01</u>	NON	Acknowledgement is not required, leading to less reliable messages
<u>Acknowledgment</u>	<u>10</u>	ACK	Contains the acknowledgement of a previous message
<u>Reset</u>	<u>11</u>	RST	Indicates that a message was received but it could not be processed

Message model: Confirmable messages



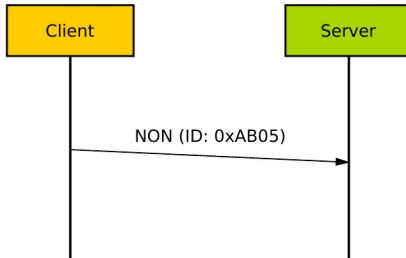
- A **Confirmable message (CON)** is a **reliable message**.
- The sending of a Confirmable message is **repeated** until the other party sends back an **Acknowledge message (ACK)**.
- The ACK message **contains the same ID** of the CON message.
- This overcomes the unreliability of UDP messages.

Message model: Reset messages



- If the server has troubles managing the incoming request, it can send back a **Reset message (RST)** instead of the Acknowledge message (ACK).
- The client stops sending its requests.

Message model: Non-confirmable messages



- **Non-confirmable messages (NON)** do not require an Acknowledge by the server.
- NON messages are **unreliable messages**; they can be **used for non-critical information** that must be delivered to the server.
- **Values read from sensors** typically belong to this category.
- Even if unreliable, NON messages have a **unique identifier**.

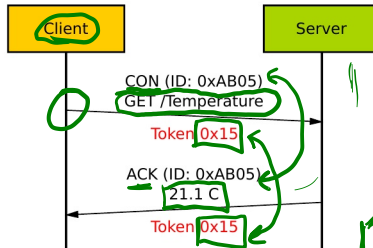
Request/Response model

- The CoAP Request/Response is the **second layer** in the CoAP abstraction layer.
- The request is sent using a **Confirmable (CON)** or **Non-Confirmable (NON)** message.

There are several scenarios depending on if the server **can answer immediately** to the client request or **the answer if not available**.

Request/Response model

CoAP

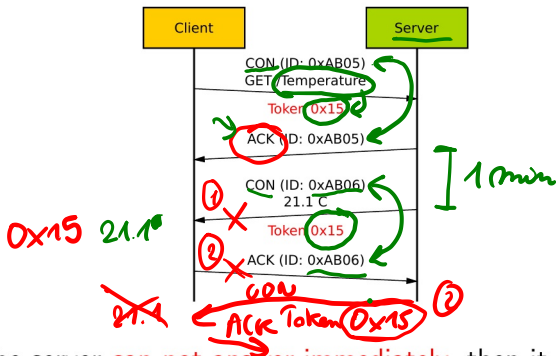


~~/room 1 / temp~~
~~/room 1 / hum~~
~~/room 1 / light 1 / is_on~~
~~/room 1 / light 2 / is_on~~
 ✓ /room 2 / temp

URI

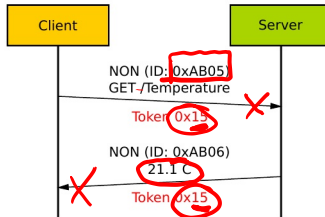
- If the server **can answer immediately** to the client request **AND** the request **was made using a CON message**, the server sends back to the client an **ACK message containing the response** or the error code.
- The **Token** is used to **match the request and the response**.
- The Token is different from the Message identifier.

Request/Response model



- If the server can not answer immediately, then it sends an ACK message with an empty response.
- As soon as the response is available, the server sends a new CON message to the client containing the response, with the corresponding Token.
- The client replies with an ACK message.

Request/Response model



- If the request coming from the client is carried using a **NON-confirmable message**, then the server answer using a **NON-confirmable message**.
- The Token is used to **match the two messages**.

Message format

Octet offset	0																2								3												
Bit offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31					
4	32	ver				type				token length				request/response code								message ID															
8	64																																				
12	96																																				
16	128																																				
20	160	1	1	1	1	1	1	1	1																												
		payload (if available)																																			

token (0..8 bytes) 0x15

options (if available)

1 1 1 1 1 1 1

- **Version** (2 bits): version number of the CoAP protocol.
- **Type** (2 bits): message type (CON, ACK, NON, RST).
- **Token length** (4 bits): size of the variable-length Token field.
- **Request/response code** (8 bits): divided into Class (3 bits) and Code (5 bits) (see next slide).
- **Message ID** (16 bits): Used to detect message duplication and to match messages.
- **Token** (variable): used to match requests and responses.

If the message is correct

Message format

Method Code
000/00001

2 (5 bits)
000/00010

Requests/responses codes.

The format is class.code

Method: 0XX

0 - EMPTY

1 - GET

2 - POST

3 - PUT

4 - DELETE

5 - FETCH

6 - PATCH

7 - iPATCH

Success: 2XX

1 - Created

2 - Deleted

3 - Valid

4 - Changed

5 - Content

31 - Continue

Client Error: 4XX

0 - Bad Request

1 - Unauthorized

2 - Bad Option

3 - Forbidden

4 - Not Found

5 - Method Not Allowed

6 - Not Acceptable

8 - Request Entity Incomplete

9 - Conflict

12 - Precondition Failed

13 - Request Entity Too Large

15 - Unsupported Content-Format

Server error: 5XX

0 - Internal server error

1 - Not implemented

2 - Bad gateway

3 - Service unavailable

4 - Gateway timeout

5 - Proxying not

supported

Signaling Codes: 7XX

0 - Unassigned

1 - CSM

2 - Ping

3 - Pong

4 - Release

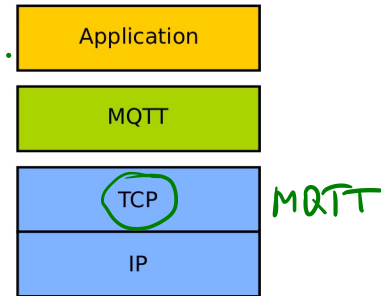
5 - Abort

Message Queuing Telemetry Transport (MQTT)

- Publish-subscribe **lightweight** messaging protocol designed for **constrained devices**.
- Protocol that is designed for **unreliable networks** or **intermittent connectivity**.
- Exchange of data with the cloud in a **real-time manner**.
- Very popular and widespread for **IoT and M2M applications**.
- **OASIS** standard.

CoAP → RFC

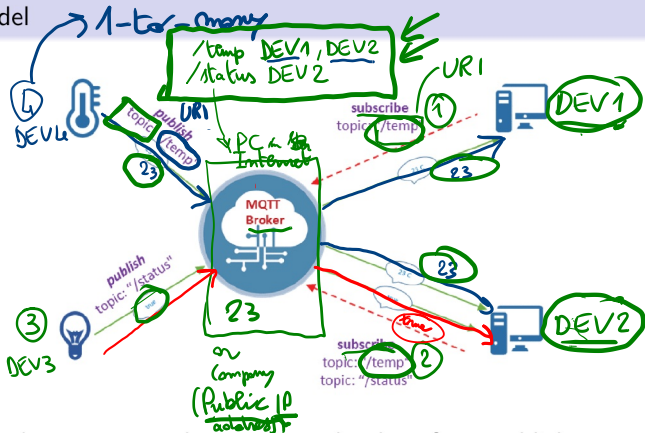
MQTT protocol stack



A variant MQTT-SN (Sensor Networks) can use other transport protocols such as UDP or Bluetooth.

MQTT model

Publisher /
Subscriber



- ① • **Broker:** a **server** that receives the data from publishers and forwards it to the interested subscribers.
- ② • **Publisher:** a client that **sends data** to the Broker.
- ③ • **Subscriber:** a client that is registered on the Broker to **receive updates** from specific sources.

MQTT topics

ASCII (European/Latin char)
 For all the languages.

- The word **"topic"** refers to an **UTF-8** string that the broker uses to **filter messages** for each connected client.
- The topic is the **subject that identifies a data exchange**.
- The topic consists of one or more **topic levels**.
- Each topic level is **separated by a forward slash** (topic level separator).

8 bits/char Latin lang
 Chinese 32 bits
 Farsi

Level 1

Level 2

Level 3

Examples of topics:

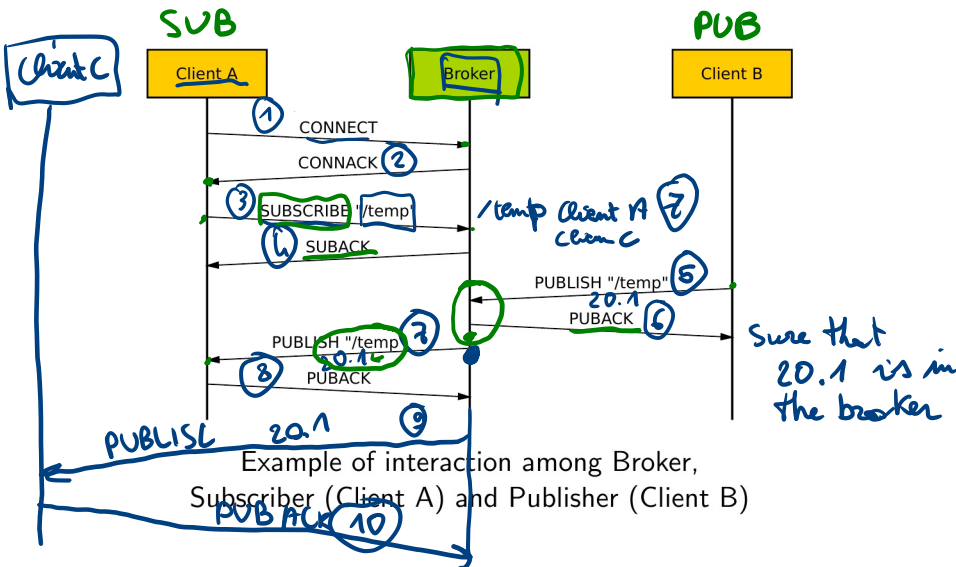
home/first-floor/kitchen/humidity

Italy/Lombardy/Milan/Bicocca

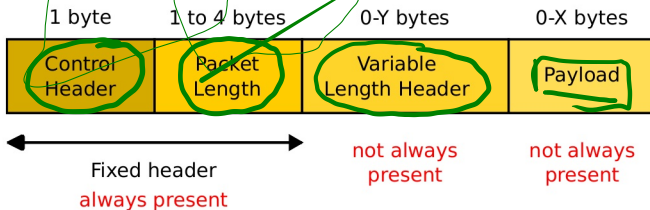
France/Paris/taxi/12748237349723422/longitude

5cc4a8cf-e485-6f30-c728-02398ddcab/status

MQTT message model



MQTT message format



- The size of the Variable Length Header **depends from the message type**.
- The payload contains the **data to send**.
- The payload **may not be present** (e.g., CONNACK does not have payload).

Considerations on scalability

- ¹⁰ n number of clients of an MQTT broker.
- topics average number of topics subscribed by each client. ^{3 topics}

Number of message transmissions (**throughput**):

Worst case: $\overline{\text{topics}} = n$ (every client subscribes every topic):

$n_{\text{msg}} = n \cdot \boxed{\text{topics}}$

^{30/1 sec} If the pub. period of all the clients is the same. Each client publishes a topic every 1 second.

$$\boxed{n_{\text{msg}}} = n^2$$

Total time **time** to deliver the messages:

^{to send all messages}

$$\boxed{\text{time}} = \boxed{n_{\text{msg}}} \cdot t_{\text{msg}}$$

where t_{msg} is the **average time** required to send a single message.

✓ If pub. ~~to~~ pub every 0.5 s
 $\hookrightarrow n_{\text{msg}} = 60/\text{sec}$

Considerations on scalability: numerical example

- $n = 100$
- $\overline{topics} = 10$

Number of message transmissions (**throughput**):

pub sub → $n_{msg} = n \cdot \overline{topics} = 100 \cdot 10 = 1000$

Worst case: $\overline{topics} = n = 100$:

pub → $n_{msg} = 100 \cdot 100 = 10000$

Total time **time** to deliver the messages with $t_{msg} = 10ms$:

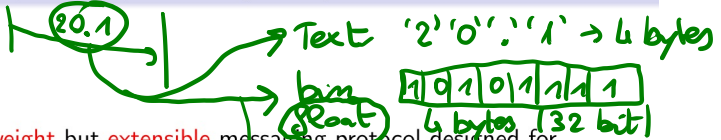
→ **average time** = $1000 \cdot 10ms = 10000ms = 10sec$

worst time = $10000 \cdot 10ms = 100000ms = 100sec = 1.67min$

*real-networks
0.5ms*

Advanced Message Queuing Protocol (AMQP)

N₁ N₂



- **Lightweight** but **extensible** messaging protocol designed for M2M messaging.
- **Binary**, application layer protocol.
- Generally used in **corporate environments**.
- Focuses on **interoperability**.
- Support for both **request-response** and **publish-subscribe** models.
- **OASIS** and **ISO** standard.

CoAP

MQTT

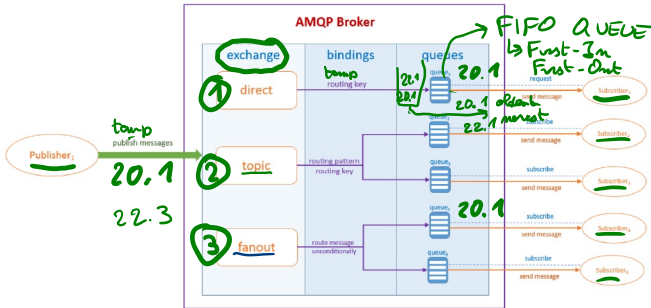
TEXT

float

100257.5 → 8 bytes

4 bytes

Message distribution model



- **Exchange**: a **routing agent** that runs on a virtual host residing on a broker's server.
- **Queue**: **named FIFO buffer** that stores messages on behalf of applications temporarily.
- **Bindings**: relationships between **message exchanges** and **message queues**.

Types of message exchanges: direct

Direct message type



routing-key (sim. to the topic)



- An exchange forwards incoming messages to queues **based on the routing key** associated with each message.

- Each binding contains a **binding key**.



- A publisher provides a **routing key** for each message sent to the direct exchange.

- A message **passes through a message queue** when the binding key from the message queue is **identical** (i.e. exact matching) to that of the publisher's message routing key.



- This method is used to implement **point-to-point messaging**.
- In cases the binding key is associated with multiple queues, it implements **multicasting operations**.



Pub. (routing-key = 20.1)

DIRECT binding-key = temp

DIRECT binding-key = temp

20.1

Types of message exchanges: topic (1/3)

Topic message type

routing key = temp
broker
pattern



Key topic that
might contain
wildcards

- The routing key is considered as a **routing pattern**, i.e., a **topic**.
- The **routing key** is fixed.
- The routing pattern in the topic exchange allows the **use of wildcards**.
- A publisher sends a message to the topic exchange **providing a routing key**.
- The message then passes to the queue if the routing pattern **matches that of the routing key**.
- This method implements a **publish/subscribe messaging pattern**.

Types of message exchanges: topic (2/3)

AMQP Topic message type

- Each keyword is **delimited by a period** (**.**).
- The ***** is used to match a **single keyword**.
- The **#** is used to match **zero or more keywords**.

Generalization of other message types:

- When only the “**#**” binding key is used, the queue receives all the messages, regardless of the routing key → **fanout exchange**.
- When **neither “*” and “#”** are used in bindings → **direct exchange**.

Types of message exchanges: topic (3/3)

Examples of topics:

→ routing - keys

- unipv.engineering.robotics (1)
- unipv.mathematics.analysis (2)
- unibid.engineering.iot (3)
- unimi.engineering.iot (4)

Examples of binding keys:

Binding key / pattern - keys

unipv.#

*.engineering.iot

#.iot

.engineering

unipv.engineering.robotics

#

Corresponding topic

1 2

3 4

3 4

1 3 4

1

1 2 3 4

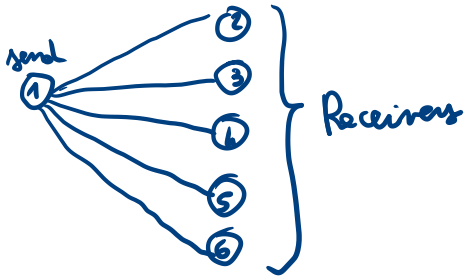
→ DIRECT MESSAGING

Types of message exchanges: fanout

Fanout message type



- This method **does not require** routing keys for binding messages to queues.
- Messages are **broadcasted** to all subscribers unconditionally.
- Used to asynchronously broadcast event notifications **to all endpoints**.



msg $N_1 \rightarrow N_4$ UNICAST

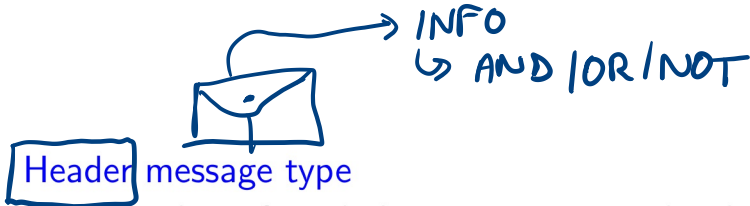
msg $N_1 \rightarrow \begin{matrix} N_2 \\ N_4 \\ N_5 \end{matrix}$ MULTICAST

Subset of nodes

msg $N_1 \rightarrow \begin{matrix} N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \end{matrix}$ BROADCAST

All the nodes

Types of message exchanges: header



- The exchange forwards the message to a queue based on **arguments or properties** specified in the header of a message.
- **X-match expressions** can logically combine multiple properties with **AND and OR conditions**.

Extensible Messaging & Presence Protocol (XMPP)

- Originally known as **Jabber** in 2002, standardized in 2011.
- **Client/server architecture** initially designed to provide application with **instant messaging capabilities**.
- Uses **XML** as the underlying data exchange format (**larger overhead** w.r.t. binary protocols).
- Runs over **TCP/IP**.
- **XML fragments** transmitted by XMPP, and used for basic communication, are called **stanzas**.
- **Point-to-point encryption** by Transport Layer Security (TLS) is built-in in the specifications.

XMPP identifiers

XMPP entities are associated with **Jabber IDs** (JIDs) **in the form of an email address** with a fully qualified domain name and/or a valid resource.

`xmpp_user@xmpp_server/resource`

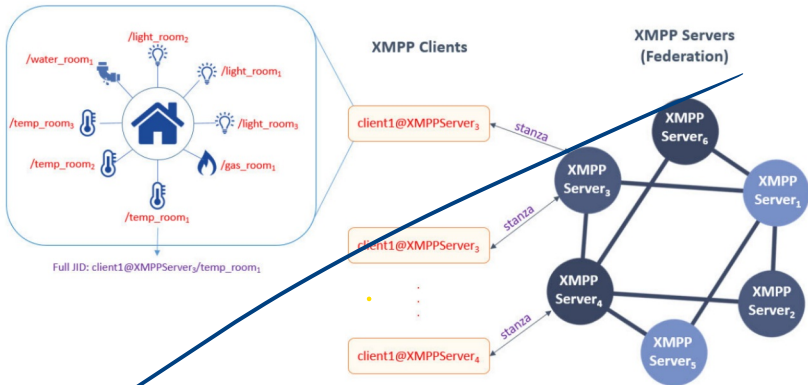
- `xmpp_user` is the **client's username**.
- `xmpp_server` is a **fully qualified domain name**.
- `resource` is an identifier used to identify the **client's device on the network**.

Bare JID : address without the resource.

Full JID : JID that includes a resource identifier.

Multiple resources (i.e. full JIDs) can be **associated with one username** indicating different devices used or associated with the same “account” or user.

XMPP architecture



- **Device-to-device** communication is not allowed.
- XMPP servers **can form a federation**: servers acknowledge each other over the same network.

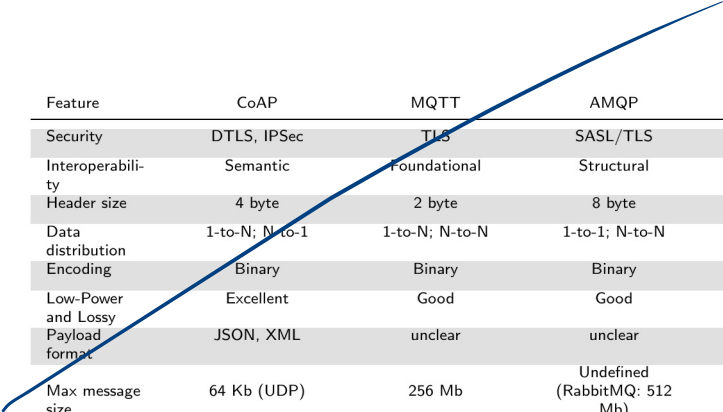
Comparison (1/3)

Feature	CoAP	MQTT	AMQP	XMPP
Year introduced	2013	1999	2003	2002
Standardized	2014 (ongoing)	2013	2014	2004
Messaging pattern	request/response	publish-subscribe	request/response; publish-subscribe	request/response; publish-subscribe
Architecture	tree	tree	star	client-server
Transport	UDP	TCP	TCP	TCP
Network layer	IPv6	IPv4 or IPv6	IPv4 or IPv6	IPv4 or IPv6
M2M communication	✓	✓	✓	✓
Asynchronous messaging	✓	✓	✓	✓
Transaction support	✗	✗	✓	✗
Extensibility	✗	✗	✓	✓

Comparison (2/3)

Feature	CoAP	MQTT	AMQP	XMPP
Data prioritization	✗	✗	✓	✓
QoS support	✓	✓	✓	✗
Message caching	✓	✓	✓	✓
Message caching	✓	✓	✓	✓
RESTful	✓ (observe option)	✗	✗	✗
Dynamic discovery	✓	✗	✗	✓
QoS levels	2 levels	3 levels	3 levels	none
Communication scope	Device to cloud	Device to cloud	Device to device; Device to cloud; Cloud to cloud	Device to cloud; Cloud to cloud
Addressing	URI	topic only	queue, topic, routing key	Jabber identification
Filtering	Resource identifier	Topic	Queue	{user: to, from}, type, iq, presence packets

Comparison (3/3)



Feature	CoAP	MQTT	AMQP	XMPP
Security	DTLS, IPSec	TLS	SASL/TLS	SASL/TLS
Interoperability	Semantic	Foundational	Structural	Structural
Header size	4 byte	2 byte	8 byte	Variable
Data distribution	1-to-N; N-to-1	1-to-N; N-to-N	1-to-1; N-to-N	1-to-1; N-to-N
Encoding	Binary	Binary	Binary	Text
Low-Power and Lossy	Excellent	Good	Good	Fair
Payload format	JSON, XML	unclear	unclear	XML
Max message size	64 Kb (UDP)	256 Mb	Undefined (RabbitMQ: 512 Mb)	Undefined; 64 Kb (stanza size)
Governing body	IETF	OASIS	OASIS	IETF