CoAP
RFC 7252

Internet of Things: Protocols and Networks
(CSC2106)

Constrained Application Protocol

SINGAPORE INSTITUTE OF TECHNOLOGY

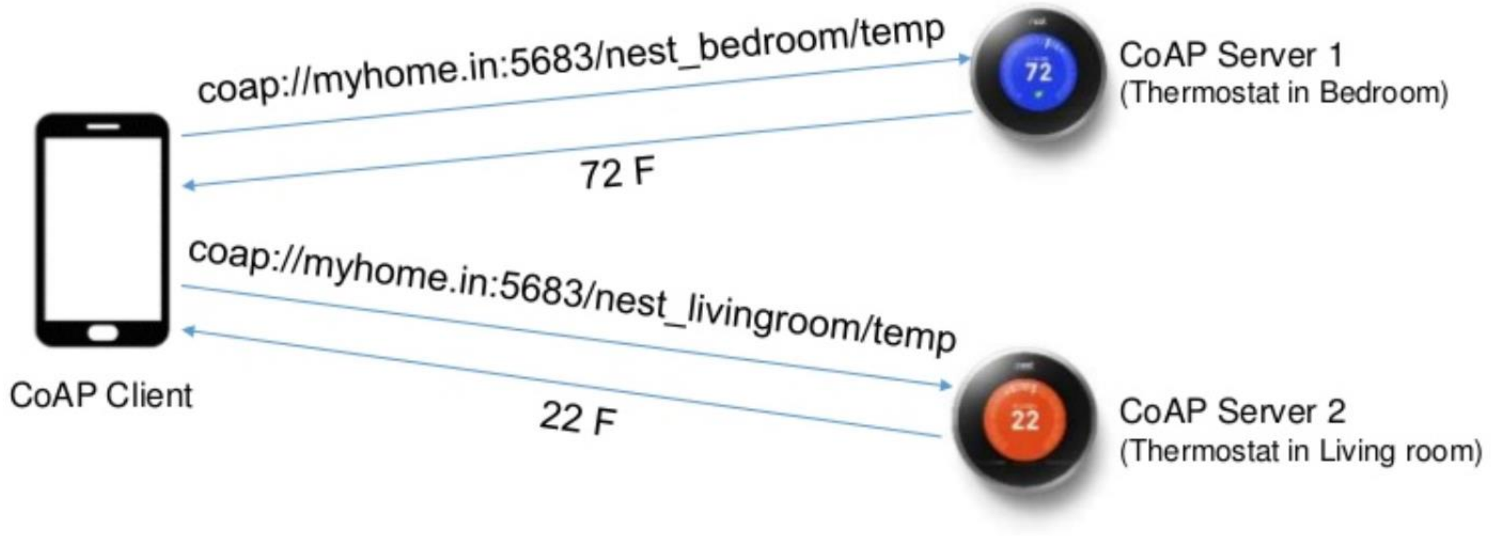
Recent Protocols for IoT

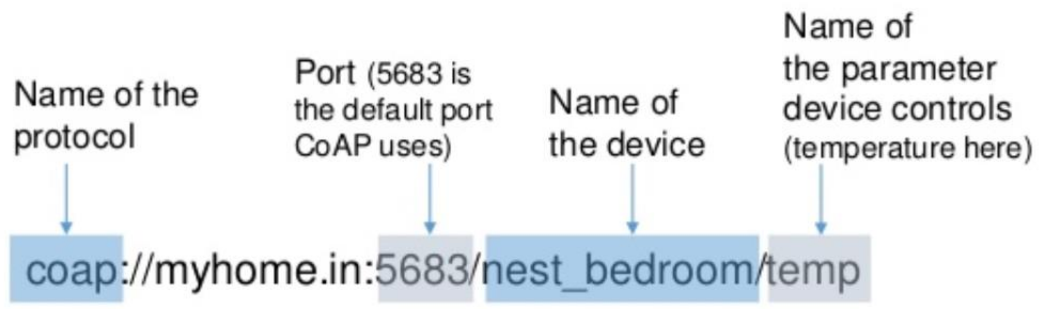
Session	**MQTT**, SMQTT, CoRE, DDS, AMQP , XMPP, **CoAP**, HTTP, REST, IEC,…	**Security**	**Management**
Network	Encapsulation **6LowPAN**, 6TiSCH, 6Lo, Thread…	IEEE 1888.3, TCG, Oath 2.0, SMACK, SASL, EDSA, ace, DTLS, Dice, …	IEEE 1905, IEEE 1451, IEEE 1377, IEEE P1828, IEEE P1856
	Routing RPL, CORPL, CARP		
Datalink	**Wi-Fi**, 802.11ah, **Bluetooth Low Energy**, **Z-Wave**, **ZigBee Smart**, DECT/ULE, 3G/LTE, NFC, Weightless, HomePlug GP, 802.15.4e, G.9959, WirelessHART, DASH7, ANT+, LTE-A, **LoRaWAN**, ISA100.11a, DigiMesh, WiMAX, **NB-IoT, SigFox** …		

Constrained Application Protocol (CoAP)

- Developed by the Constrained Resource Environments (CoRE) IETF group
- REST-based web transfer protocol (RESTful protocol)
- To communicate over the Internet using UDP instead of TCP (while still providing reliability)
- CoAP has been designed to work on resource-constrained microcontrollers with as low as 10 KB of RAM and 100 KB of code space (RFC 7228).
- CoAP could be used for adapting simple HTTP interfaces into a more compact protocol
  - different message types to request resources from server: GET, PUT, POST, DELETE and **OBSERVE**
  - subset of HTTP functionality re-designed for low power embedded devices such as sensors (for IoT and M2M)
- Supports multicast and congestion control

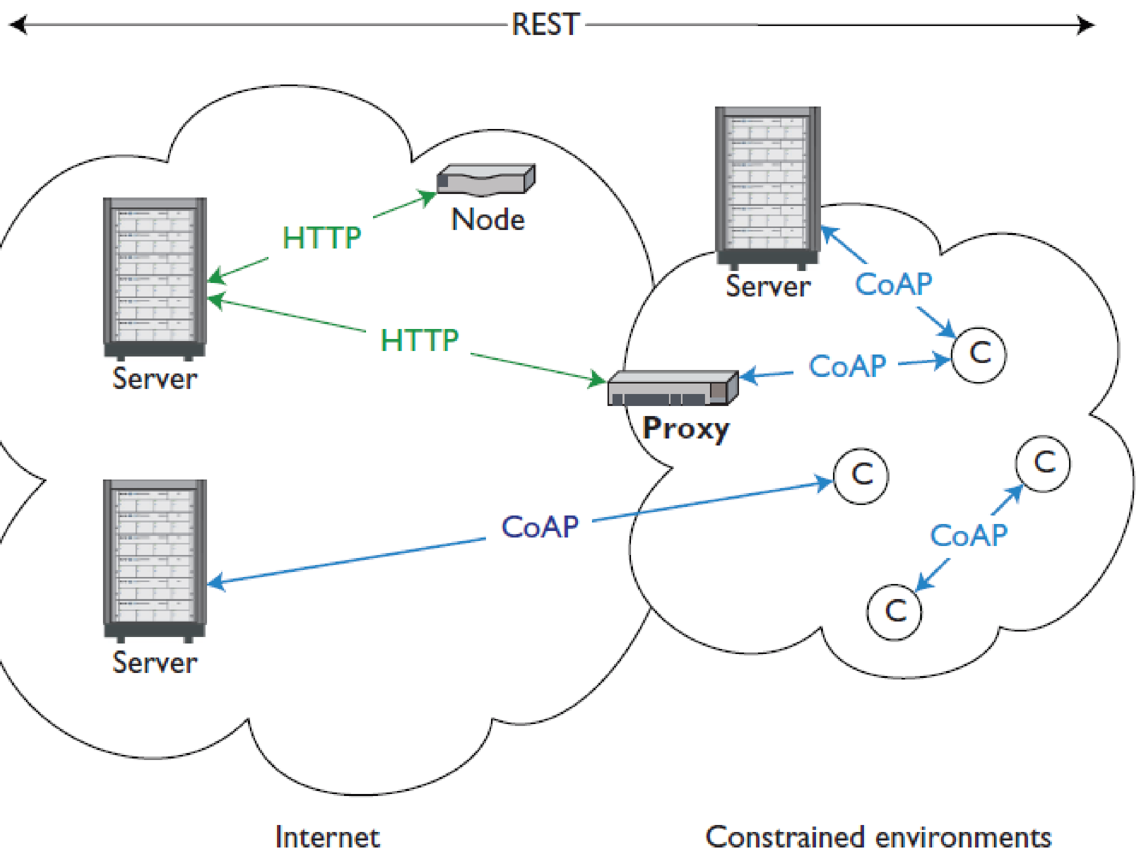
Example of a CoAP Transaction


coap://myhome.in:5683/nest_bedroom/temp
72
CoAP Server 1
(Thermostat in Bedroom)
72 F
coap://myhome.in:5683/nest_livingroom/temp
CoAP Client
22 F
22
CoAP Server 2
(Thermostat in Living room)

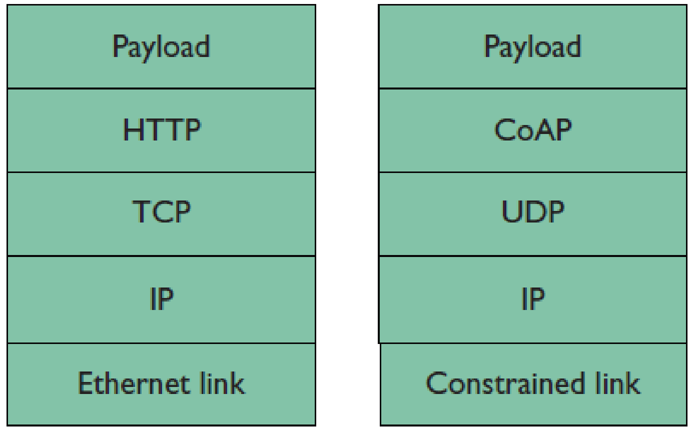

Name of the protocol
Port (5683 is the default port CoAP uses)
Name of the device
Name of the parameter device controls (temperature here)
coap://myhome.in:5683/nest_bedroom/temp

https://devopedia.org/constrained-application-protocol

CoAP Architecture



HTTP and CoAP work together across constrained and traditional Internet environments

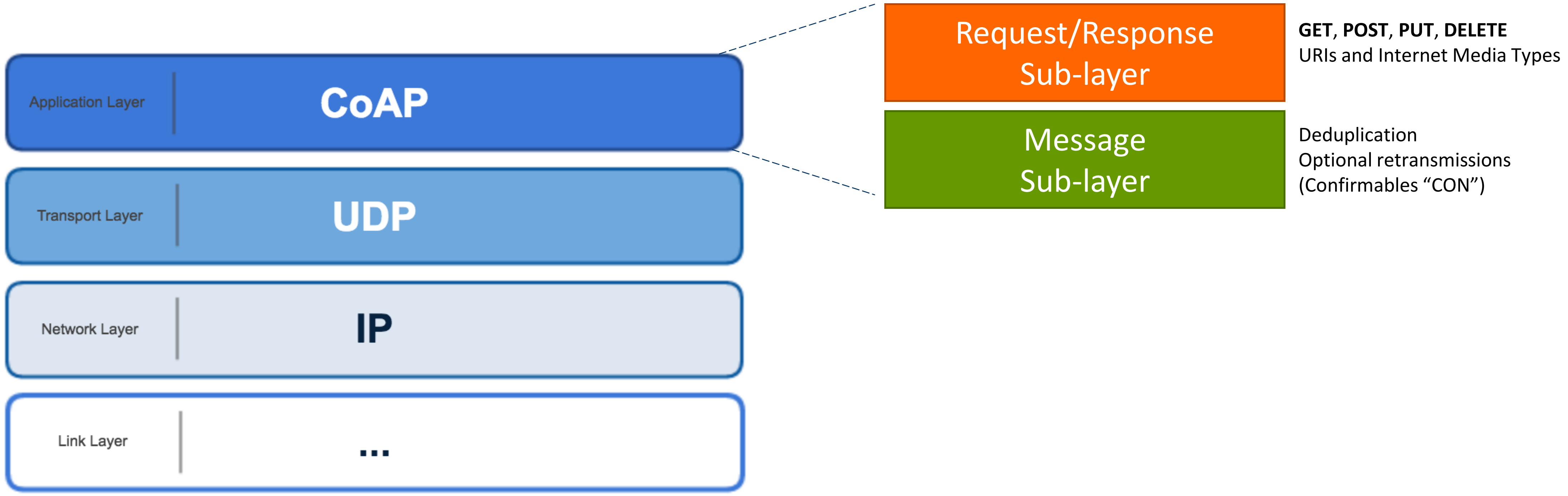


HTTP protocol stack and CoAP protocol stack

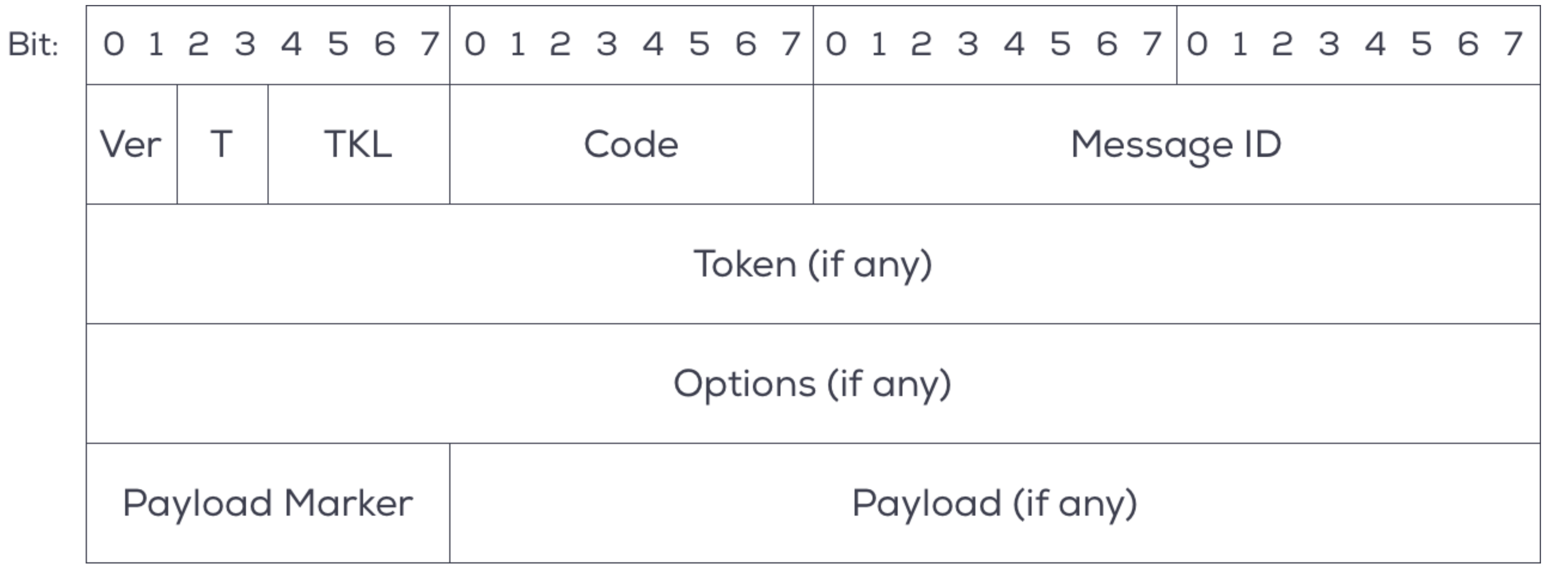
Bormann, Carsten, Angelo P. Castellani, and Zach Shelby. 2012. "CoAP: An application protocol for billions of tiny Internet nodes." IEEE Internet Computing, vol. 1, no. 2, pp. 62–67, Mar/Apr.

TCP vs UDP

- TCP overhead is too high, and its flow control is not appropriate for short-lived transactions
- UDP has lower overhead and supports multicast



Message Header (4 bytes)



Ver - Version (1)

T – Message Type (Confirmable, Non-Confirmable, Acknowledgement, Reset)

TKL- Token Length, if any, the number of Token bytes after this header

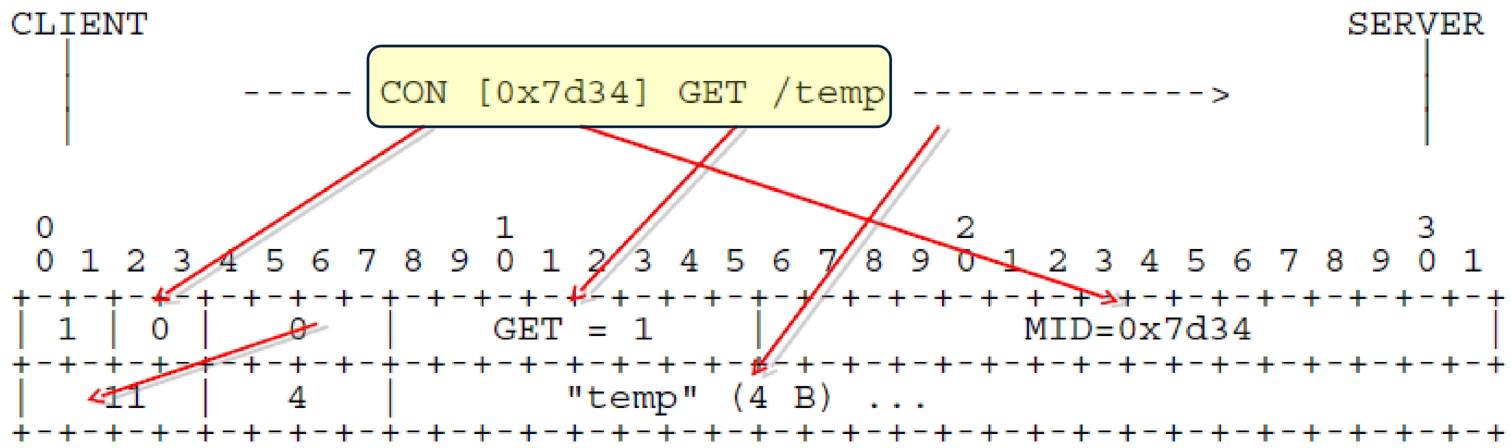
Code - Request Method (1-10) or Response Code (40-255)

Message ID – 16-bit identifier for matching responses

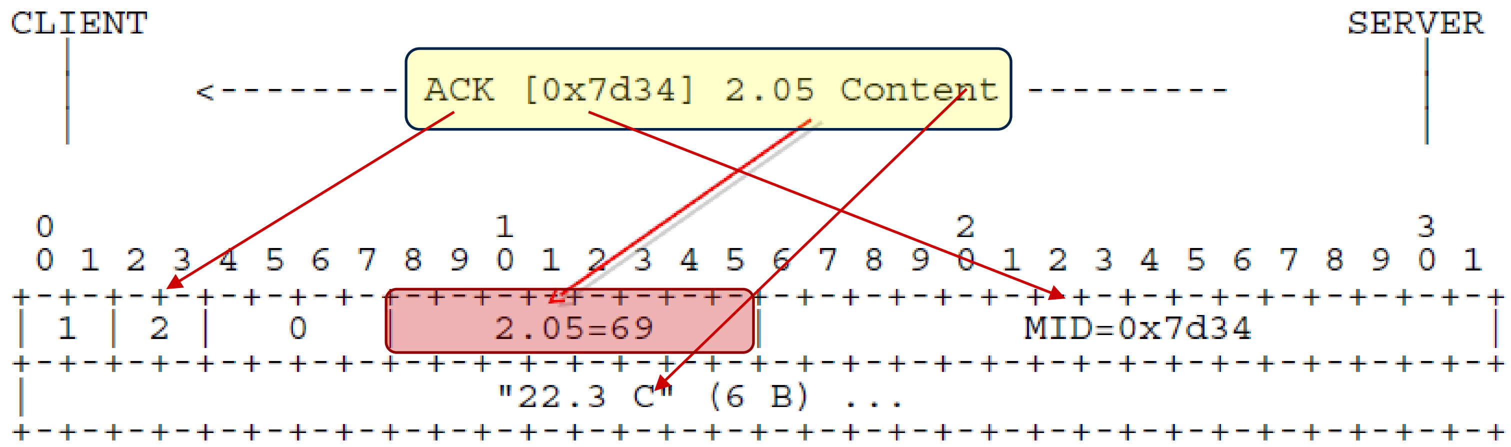
Token – Optional response matching token

https://tools.ietf.org/html/rfc7252
https://tools.ietf.org/id/draft-ietf-core-coap-00.html

Message Mapping



```
CLIENT                                                       SERVER
  |                                                            |
  |       ----- CON [0x7d34] GET /temp -------------->         |
  |                                                            |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 0 |   0   |     GET = 1   |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  11   |   4   |      "temp" (4 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



```
CLIENT                                                       SERVER
  |                                                            |
  |     <-------- ACK [0x7d34] 2.05 Content ---------          |
  |                                                            |

 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 1 | 2 |   0   |    2.05=69    |          MID=0x7d34           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  "22.3 C" (6 B) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Response Code

CoAP Status Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
2.31	Continue
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method Not Allowed
4.06	Not Acceptable
4.08	Request Entity Incomplete
4.12	Precondition Failed
4.13	Request Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported

HTTP Status Code	Description
1xx	Informational
2xx	Successful 200 – OK 201 – Created 202 – Accepted 204 – No Content
3xx	Redirection 301 - Moved Permanently 305 - Use Proxy 307 - Temporary Redirect
4xx	Client Error 400 – Bad Request 401 – Unauthorized 403 – Forbidden 404 - Not Found 405 – Method Not Found 408 – Request Timeout
5xx	500 – Internal Server Error 501 – Not Implemented 503 – Service Unavailable 504 - Gateway Timeout

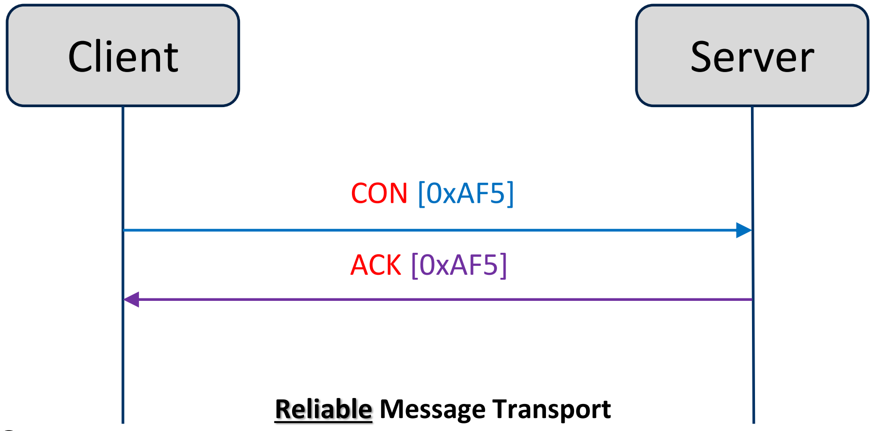
Only mostly used HTTP Status Codes are listed here

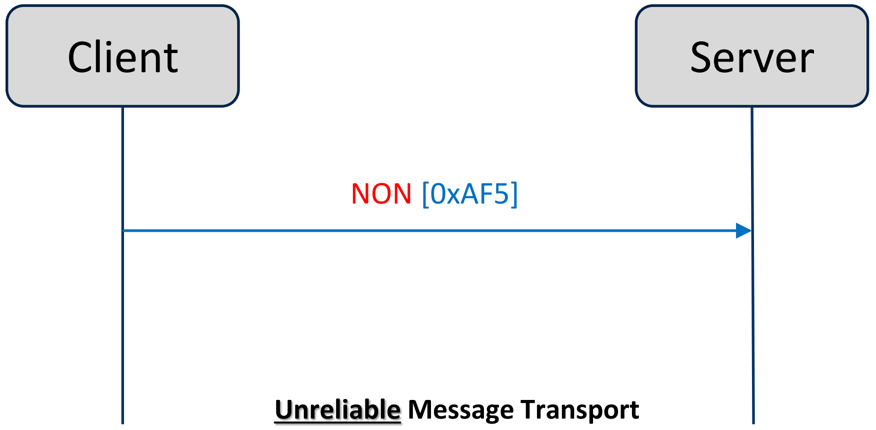
https://www.slideshare.net/HamdamboyUrunov/the-constrained-application-protocol-coap-part-2

Message Layer Model

Four Message Types:

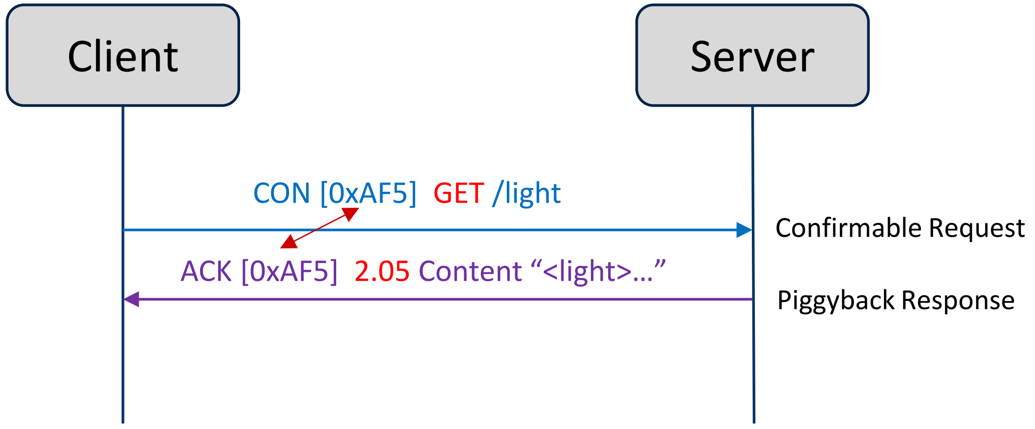
- **Confirmable (CON)** – requires an acknowledgement
- **Non-confirmable (NON)** – no acknowledgement needed
- **Acknowledgement (ACK)** – Acknowledge a Confirmable message
- **Reset (RST)** - indicates a Confirmable message has been received but context is missing for processing



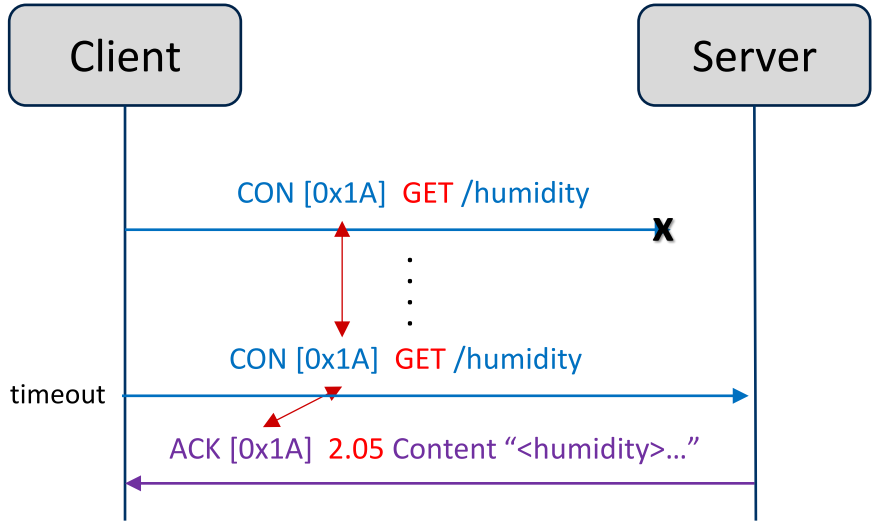


Request / Response Layer Model

Piggy-backed

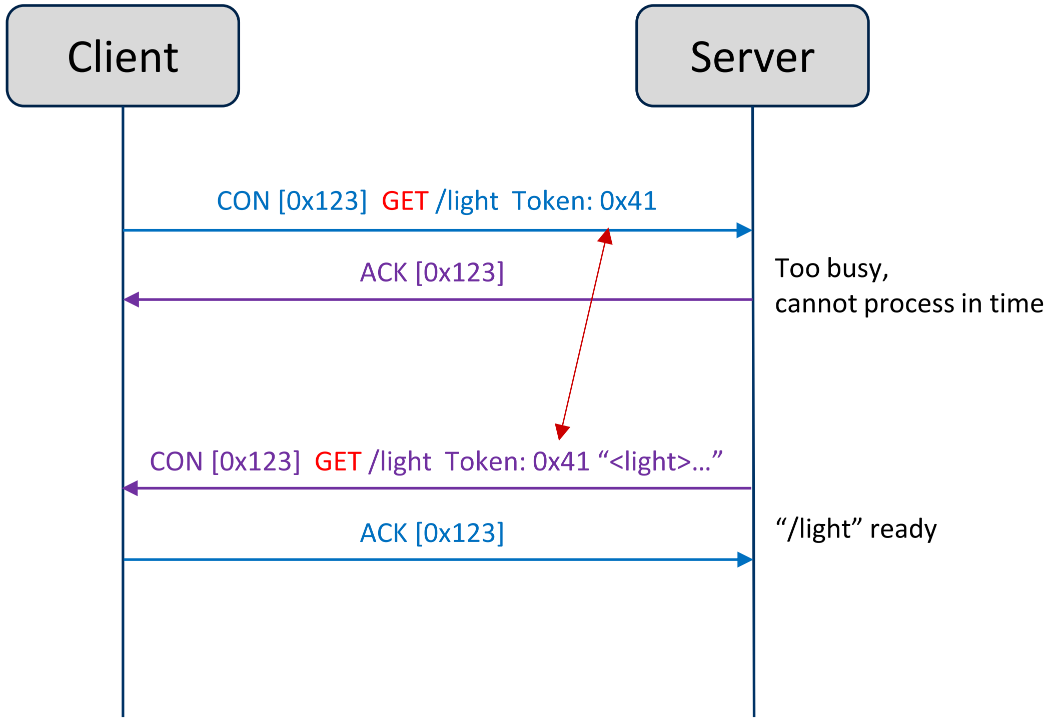


Dealing with Packet Loss



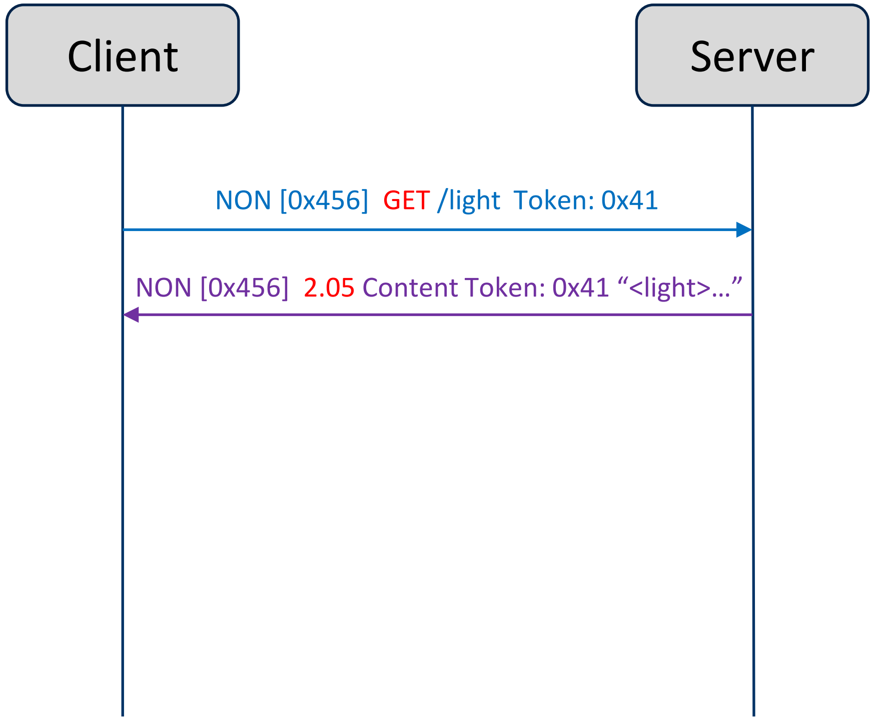
Request / Response Layer Model

Separate Response



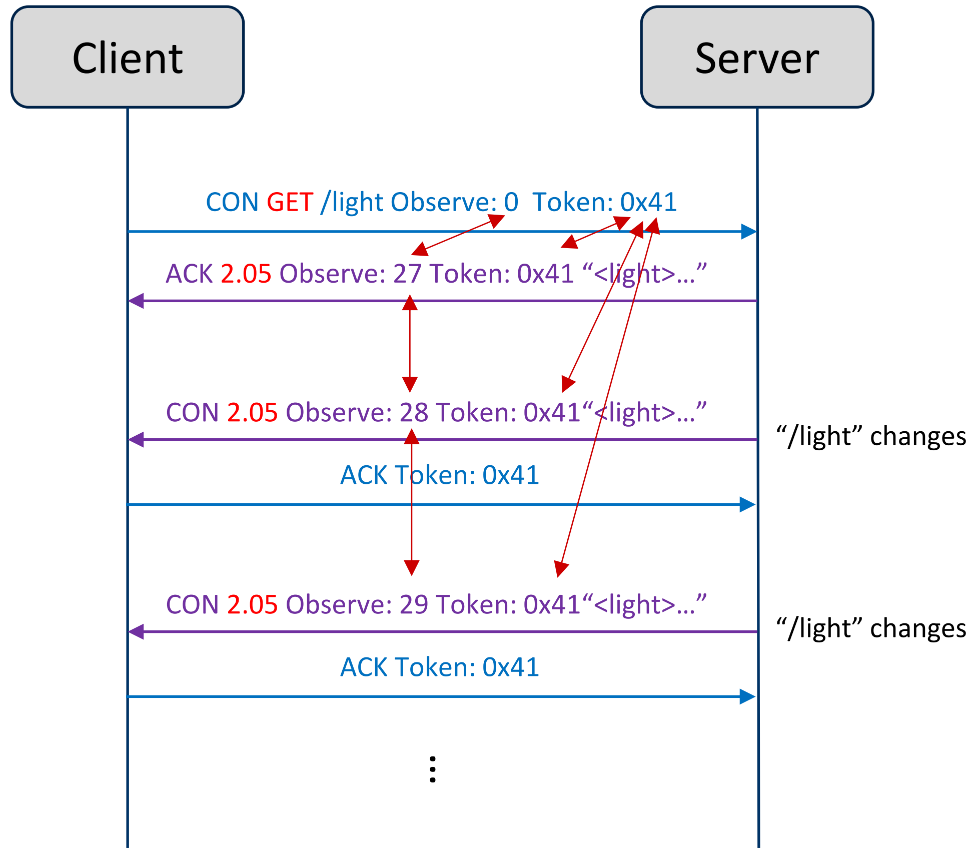
Client
Server
CON [0x123] GET /light Token: 0x41
ACK [0x123]
Too busy,
cannot process in time
CON [0x123] GET /light Token: 0x41 “<light>…”
ACK [0x123]
“/light” ready

Non-confirmable R&R



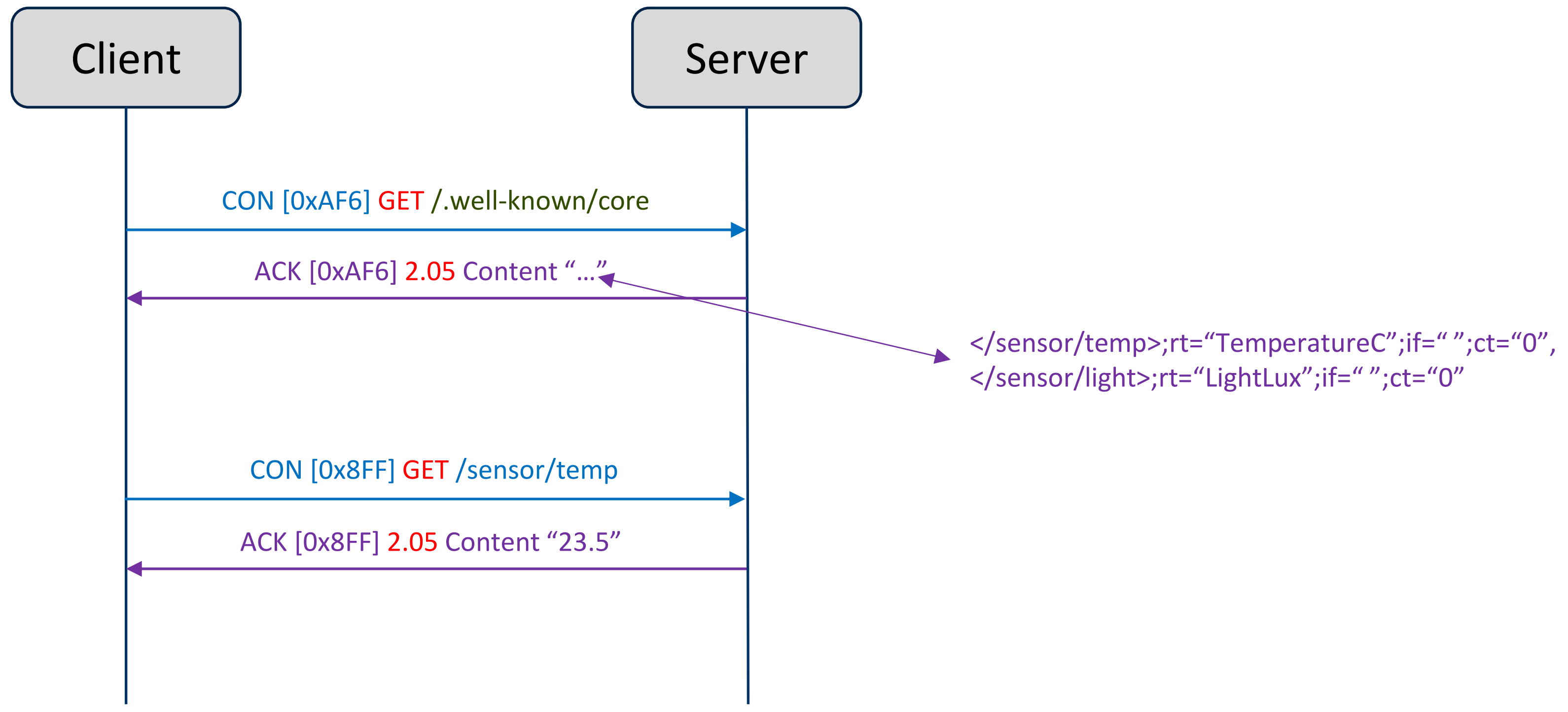
Client
Server
NON [0x456] GET /light Token: 0x41
NON [0x456] 2.05 Content Token: 0x41 “<light>…”

Observation

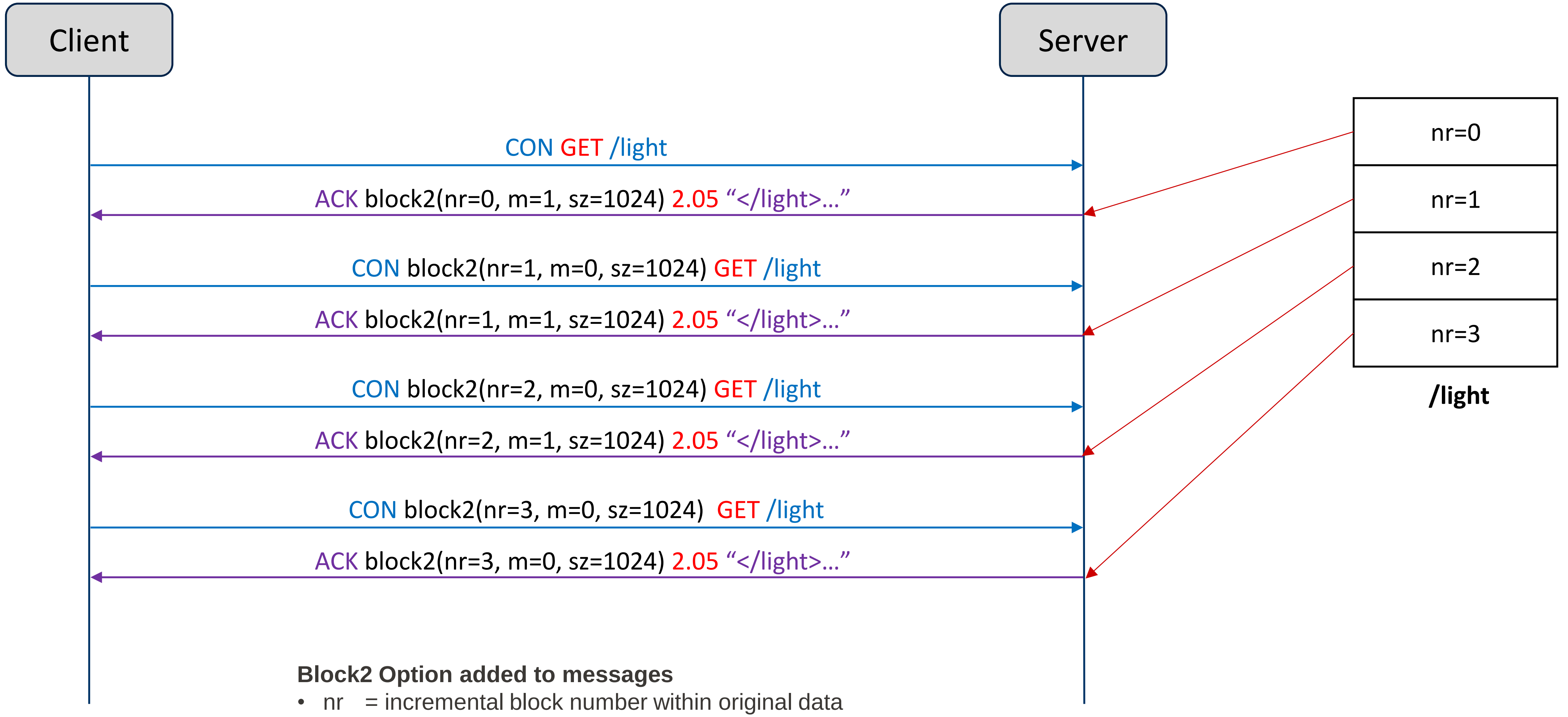

Client
Server
CON GET /light Observe: 0 Token: 0x41
ACK 2.05 Observe: 27 Token: 0x41 "<light>..."
CON 2.05 Observe: 28 Token: 0x41"<light>..."
"/light" changes
ACK Token: 0x41
CON 2.05 Observe: 29 Token: 0x41"<light>..."
"/light" changes
ACK Token: 0x41
⋮

http://tesi.cab.unipd.it/44294/1/TESI.pdf

Resource Discovery


Client
Server
CON [0xAF6] GET /.well-known/core
ACK [0xAF6] 2.05 Content "…"
</sensor/temp>;rt="TemperatureC";if=" ";ct="0",
</sensor/light>;rt="LightLux";if=" ";ct="0"
CON [0x8FF] GET /sensor/temp
ACK [0x8FF] 2.05 Content "23.5"

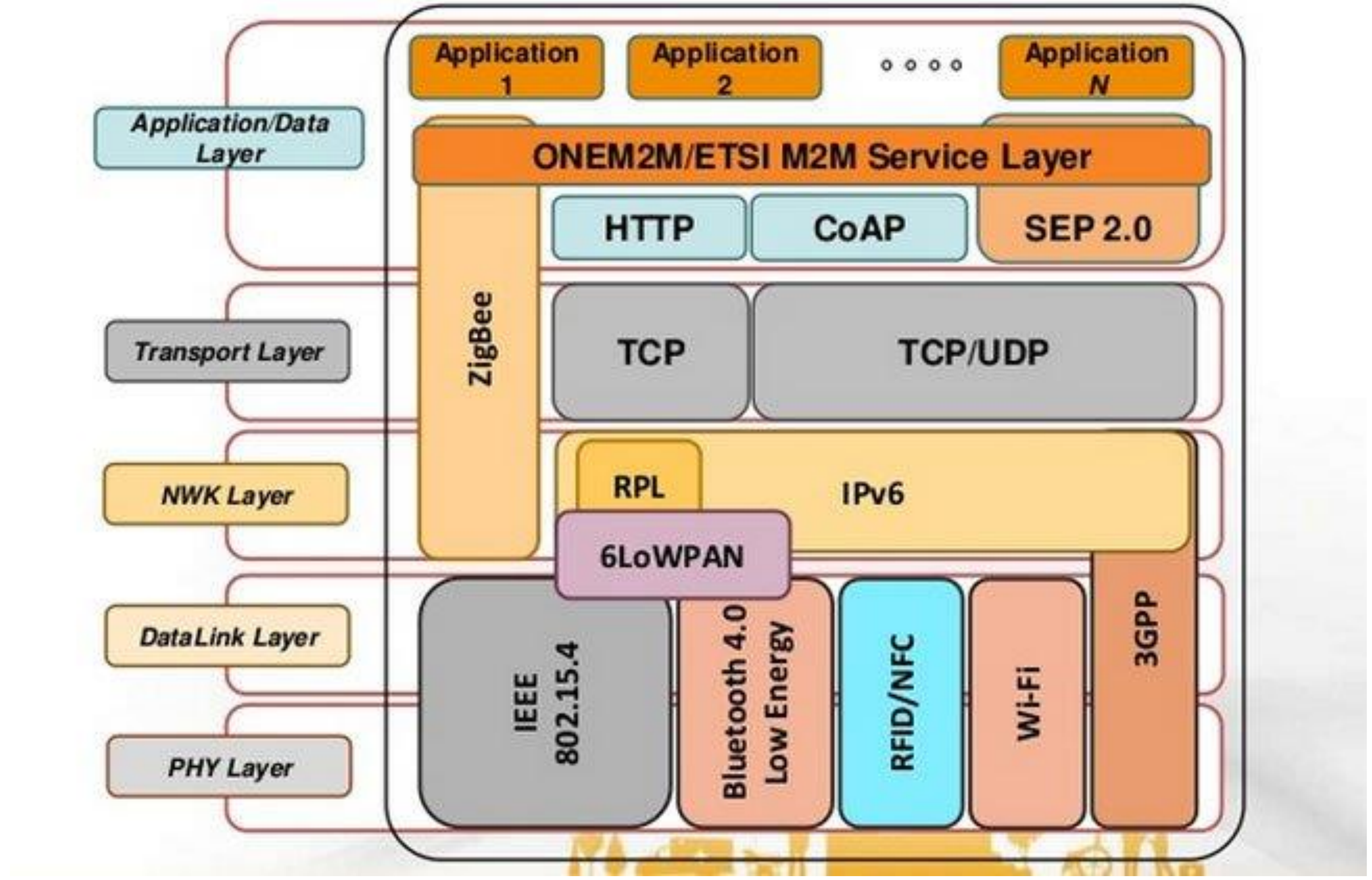
Block Transfer


Client
Server
CON GET /light
ACK block2(nr=0, m=1, sz=1024) 2.05 "</light>..."
CON block2(nr=1, m=0, sz=1024) GET /light
ACK block2(nr=1, m=1, sz=1024) 2.05 "</light>..."
CON block2(nr=2, m=0, sz=1024) GET /light
ACK block2(nr=2, m=1, sz=1024) 2.05 "</light>..."
CON block2(nr=3, m=0, sz=1024) GET /light
ACK block2(nr=3, m=0, sz=1024) 2.05 "</light>..."
nr=0
nr=1
nr=2
nr=3
/light

Block2 Option added to messages

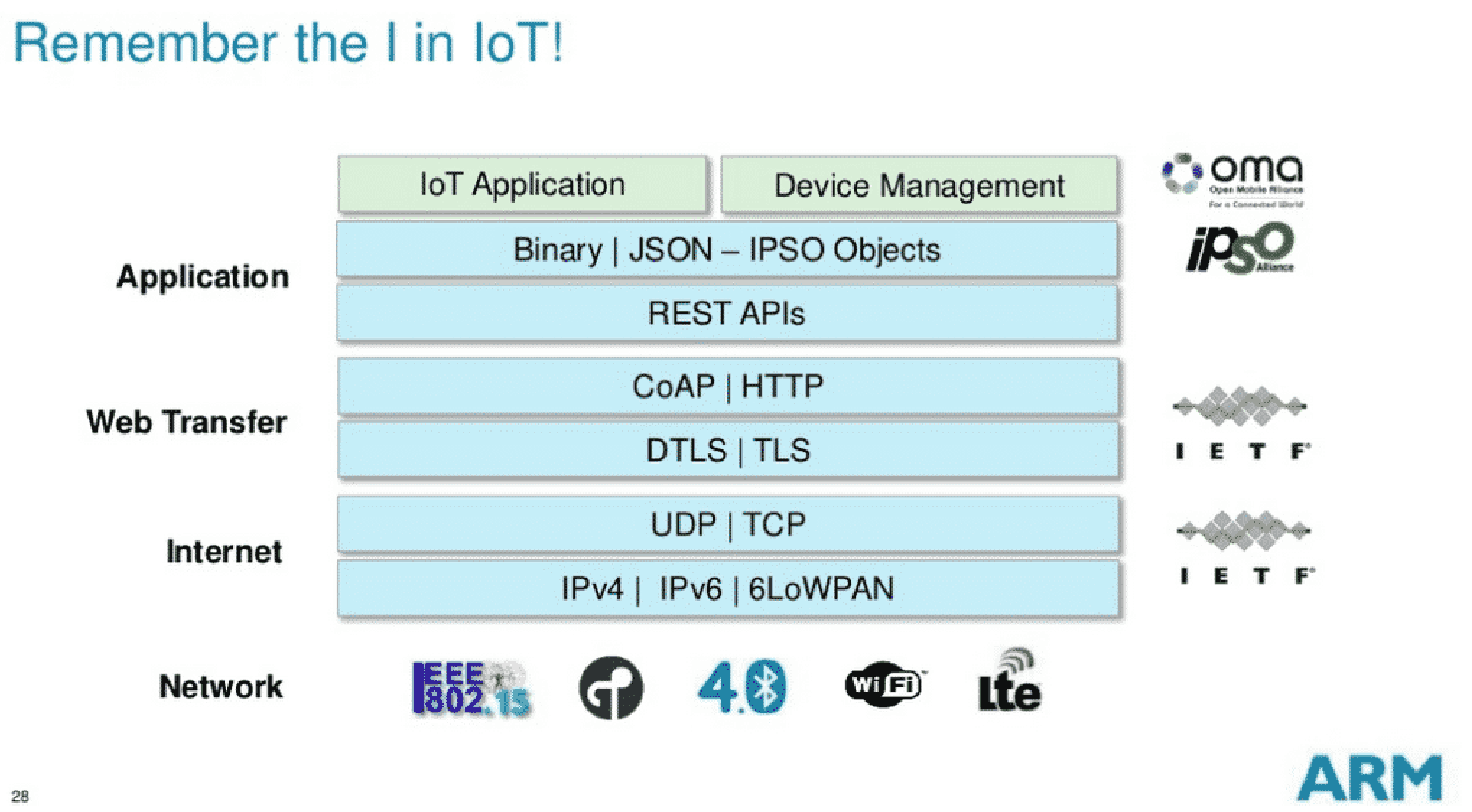
- nr = incremental block number within original data
- m = more blocks flag
- sz = block size

IoT



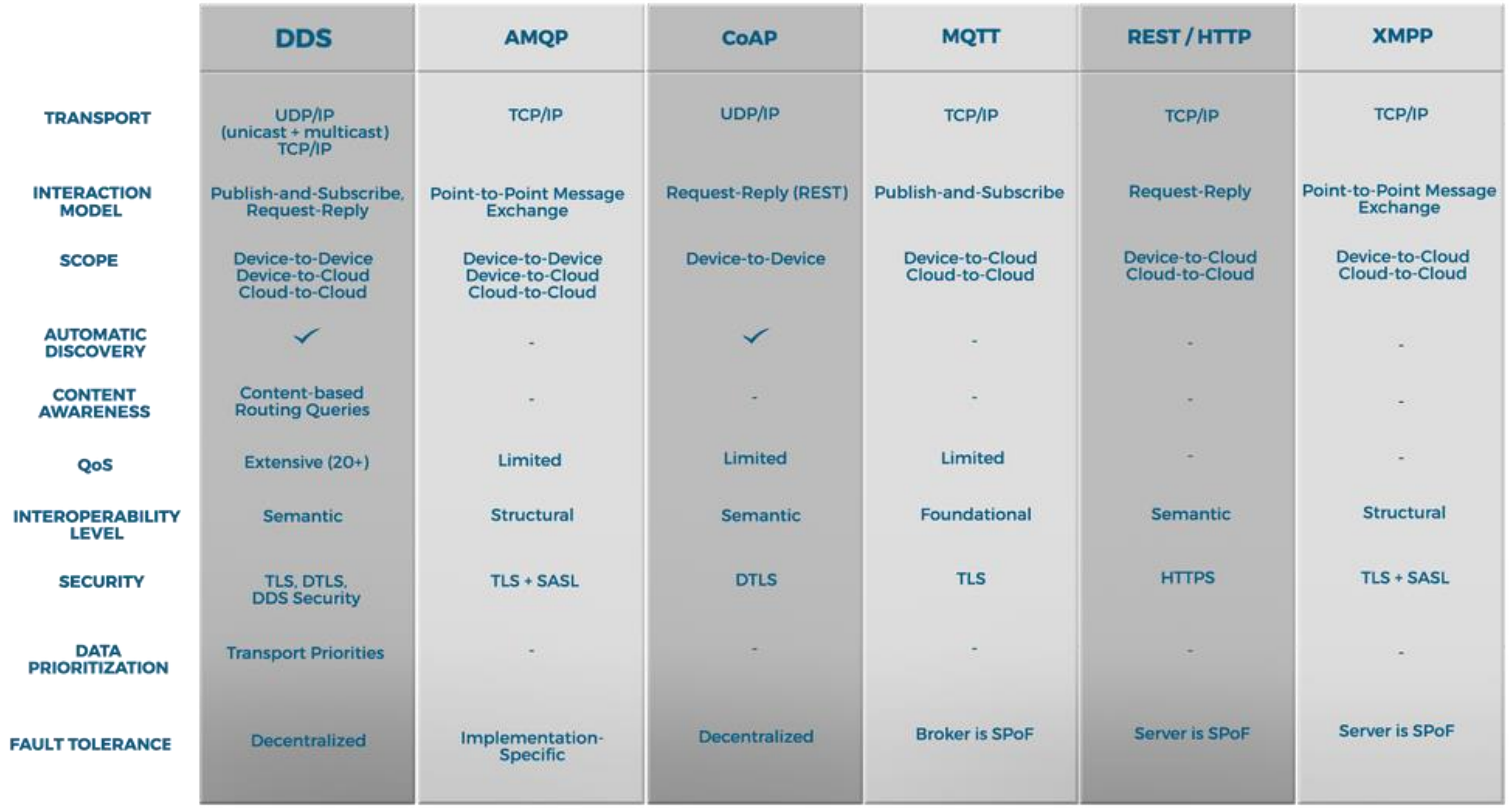
https://www.postscapes.com/internet-of-things-protocols/

IoT


Remember the I in IoT!

https://www.slideshare.net/slideshow/embed_code/45755086

Comparing CoAP with …



	DDS	AMQP	CoAP	MQTT	REST / HTTP	XMPP
TRANSPORT	UDP/IP (unicast + multicast) TCP/IP	TCP/IP	UDP/IP	TCP/IP	TCP/IP	TCP/IP
INTERACTION MODEL	Publish-and-Subscribe, Request-Reply	Point-to-Point Message Exchange	Request-Reply (REST)	Publish-and-Subscribe	Request-Reply	Point-to-Point Message Exchange
SCOPE	Device-to-Device Device-to-Cloud Cloud-to-Cloud	Device-to-Device Device-to-Cloud Cloud-to-Cloud	Device-to-Device	Device-to-Cloud Cloud-to-Cloud	Device-to-Cloud Cloud-to-Cloud	Device-to-Cloud Cloud-to-Cloud
AUTOMATIC DISCOVERY	✓	-	✓	-	-	-
CONTENT AWARENESS	Content-based Routing Queries	-	-	-	-	-
QoS	Extensive (20+)	Limited	Limited	Limited	-	-
INTEROPERABILITY LEVEL	Semantic	Structural	Semantic	Foundational	Semantic	Structural
SECURITY	TLS, DTLS, DDS Security	TLS + SASL	DTLS	TLS	HTTPS	TLS + SASL
DATA PRIORITIZATION	Transport Priorities	-	-	-	-	-
FAULT TOLERANCE	Decentralized	Implementation-Specific	Decentralized	Broker is SPoF	Server is SPoF	Server is SPoF

https://devopedia.org/constrained-application-protocol

Other CoAP Stuff

- Intermediaries and Caching
- Security - DTLS (TLS/SSL for Datagrams)
- Alternative Transport (e.g. SMS)
- Message Header Options

Internet of Things: Protocols and Networks

(CSC2106)

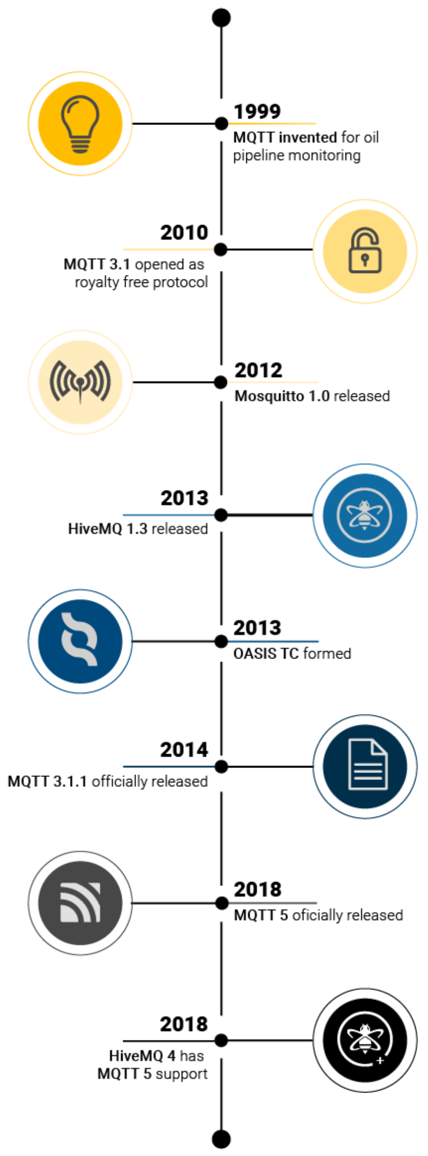
MQ Telemetry Transport (MQTT)

Recent Protocols for IoT

Session	MQTT, SMQTT, CoRE, DDS, AMQP , XMPP, **CoAP, HTTP, REST**, IEC,…	**Security**	**Management**
Network	Encapsulation **6LowPAN**, 6TiSCH, 6Lo, Thread…	IEEE 1888.3, TCG, Oath 2.0, SMACK, SASL, EDSA, ace, DTLS, Dice, …	IEEE 1905, IEEE 1451, IEEE 1377, IEEE P1828, IEEE P1856
	Routing RPL, CORPL, CARP		
Datalink	**Wi-Fi**, 802.11ah, **Bluetooth Low Energy**, **Z-Wave, ZigBee Smart**, DECT/ULE, 3G/LTE, NFC, Weightless, HomePlug GP, 802.15.4e, G.9959, WirelessHART, DASH7, ANT+, LTE-A, **LoRaWAN**, ISA100.11a, DigiMesh, WiMAX, **NB-IoT, SigFox** …		

Message Queuing Telemetry Transport (MQTT)

- MQTT was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) back in 1999, where their use case was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connections. They specified the following goals, which the future protocol should have:
  - Simple to implement
  - Bi-directional asynchronous "push" Communications
  - Provide a QoS Data Delivery
  - Lightweight and Bandwidth Efficient
  - Data Agnostic
  - Continuous Session Awareness


1999
MQTT invented for oil pipeline monitoring
2010
MQTT 3.1 opened as royalty free protocol
2012
Mosquitto 1.0 released
2013
HiveMQ 1.3 released
2013
OASIS TC formed
2014
MQTT 3.1.1 officially released
2018
MQTT 5 oficially released
2018
HiveMQ 4 has MQTT 5 support

https://www.youtube.com/watch?v=QSwR-JMmNOo&feature=emb_logo
https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/

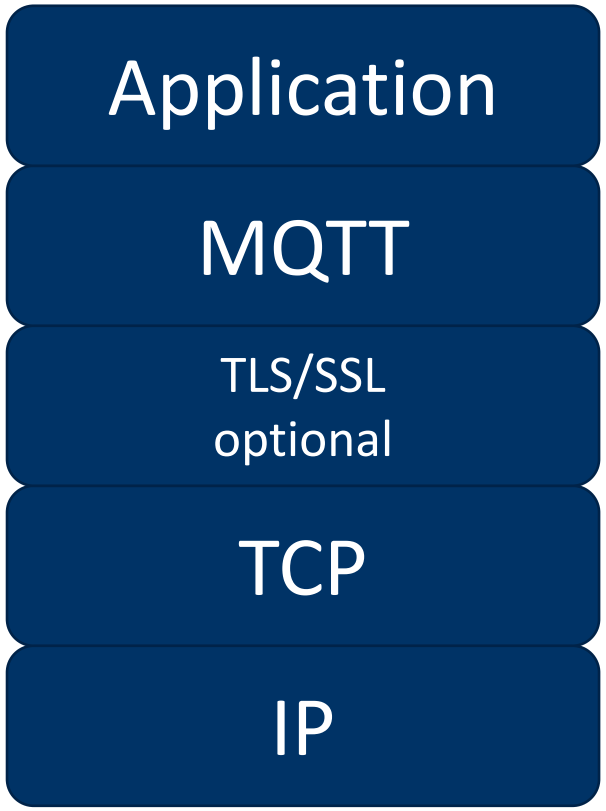
MQTT

- Built for proprietary embedded systems; now shifting to IoT
- You can send anything as a message; up to 256 MB
- Built for unreliable networks (Continuous Session Awareness)
- Enterprise scale implementations down to hobby projects
- Decouples readers and writers
- Message have a topic, quality of service, and retain status associated with them
- MQTT is a binary based protocol where the control elements are **binary bytes** and not **text strings**.

Protocol Stack

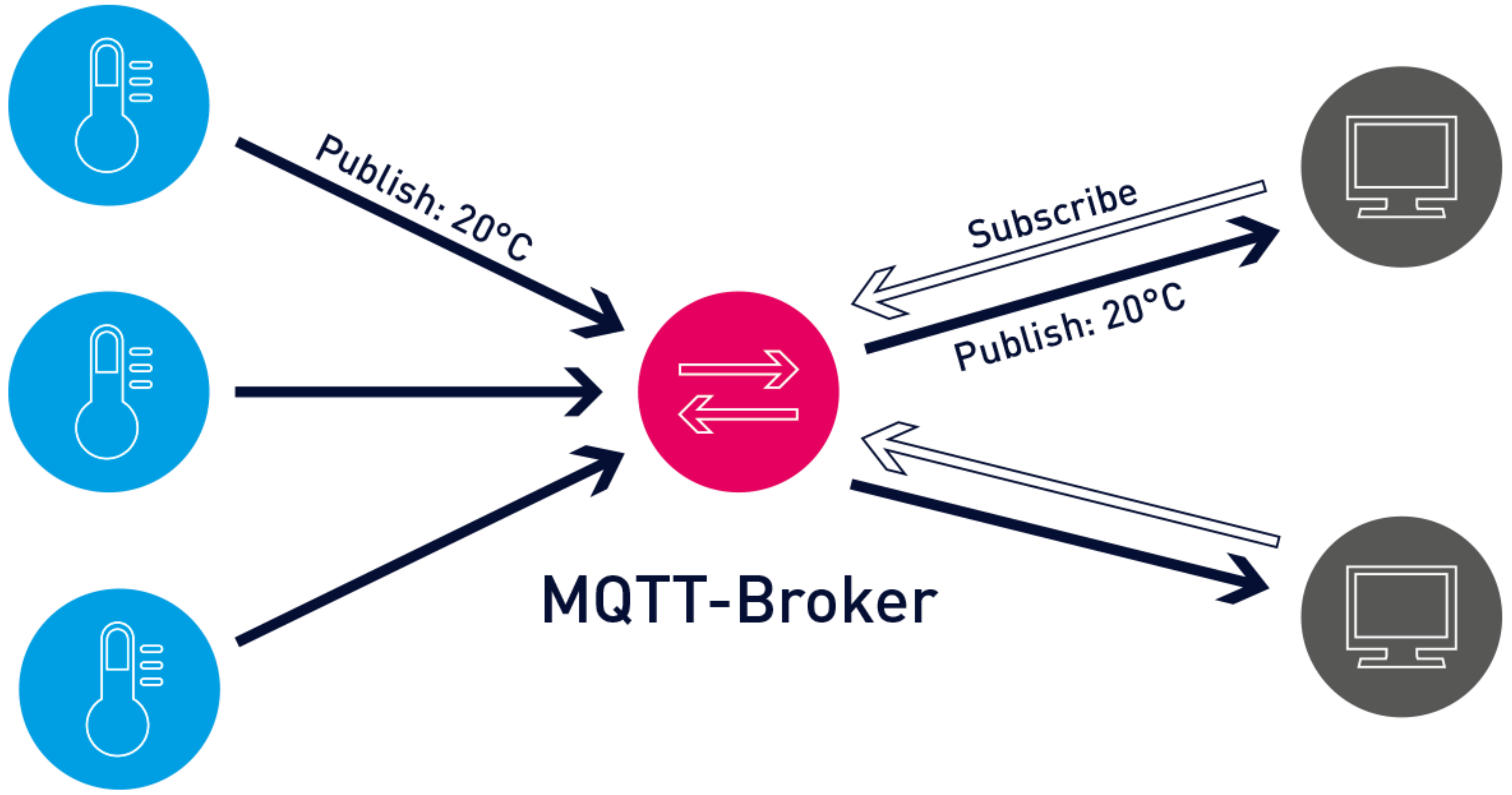
TCP/IP Port: 1883

When running over TLS/SSL, TCP/IP Port 8883
SSL: Secure Socket Layer
TLS: Transport Layer Security*



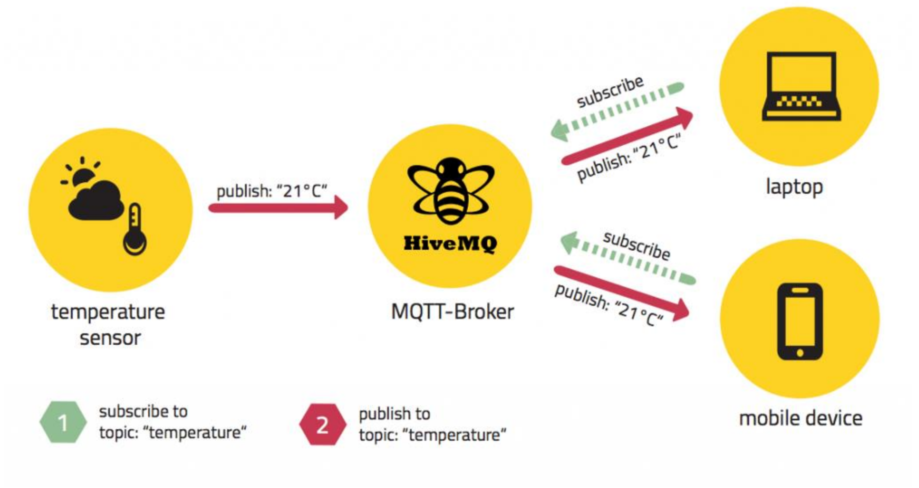
MQTT

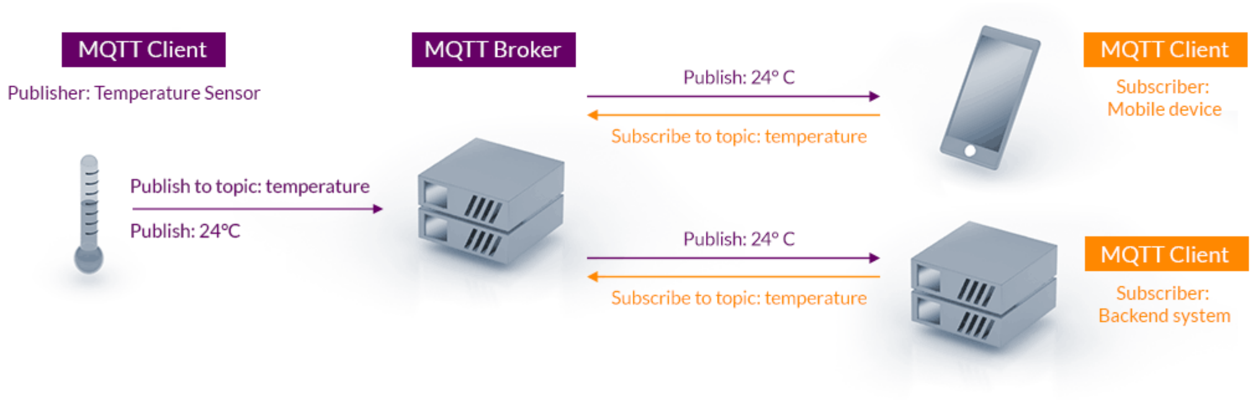
- MQTT consist of three parts:
  - Broker
  - Subscribers
  - Publishers



https://www.paessler.com/it-explained/mqtt

Publish/Subscribe Concept




MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
MQTT Client
Subscriber: Backend system

Decoupled in space and time:

- **Clients** do not need each others IP address
- **Broker**'s IP and port must be known by clients
- **Namespace** hierarchy used for **topic** filtering
- Possible that a published message is never consumed by any subscriber

MQTT: Simple to implement

Connect

Subscribe

Publish

Unsubscribe

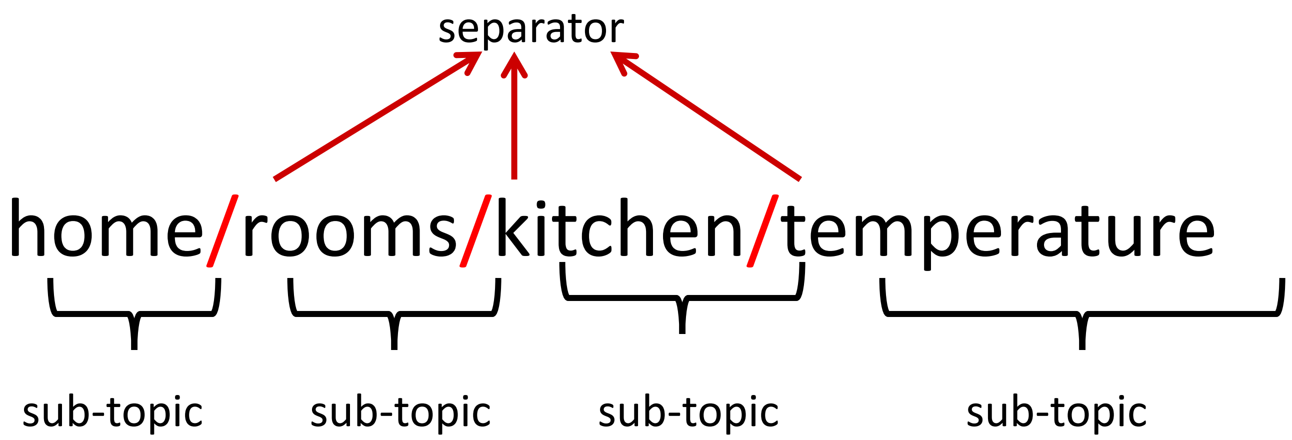
Disconnect

```
client = new Messaging.Client(hostname, port, clientId)
client.onMessageArrived = messageArrived;
client.onConnectionLost = connectionLost;
client.connect({ onSuccess: connectionSuccess });

function connectionSuccess() {
     client.subscribe(“planets/earth");
     var msg = new Messaging.Message("Hello world!");
     msg.destinationName = "planets/earth";
     client.publish(msg);
}

function messageArrived(msg) {
     console.log(msg.payloadString);
     client.unsubscribe("planets/earth");
     client.disconnect();
}
```

Eclipse Paho JavaScript MQTT client

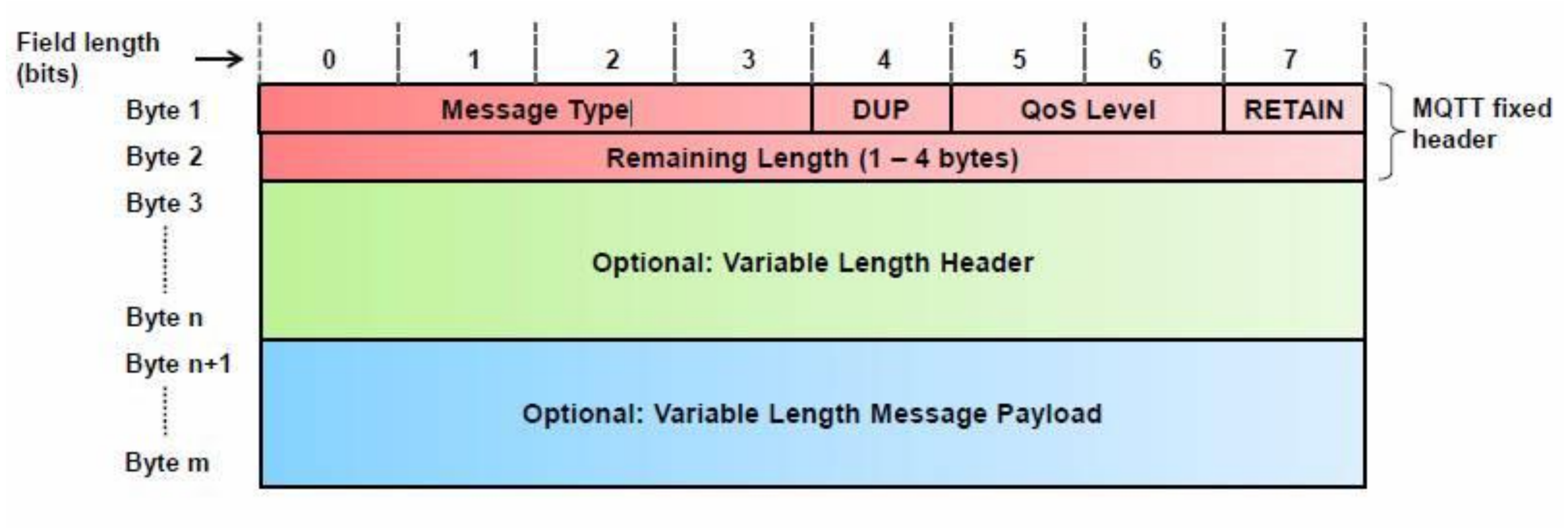
Topics

- Each published data specifies a topic
- Each subscriber subscribed to that topic will receive it
- Topic format:

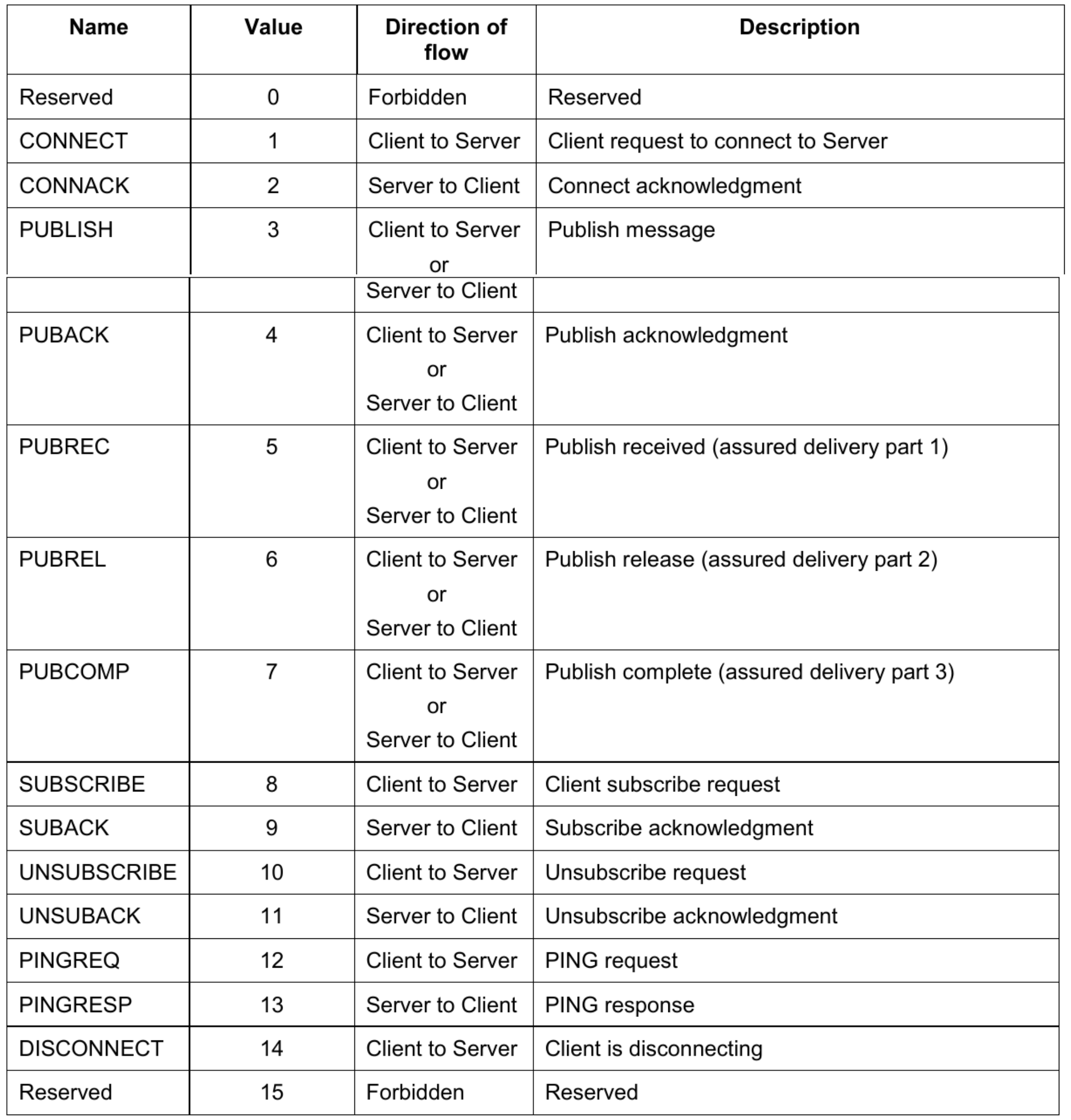


MQTT Message Format

Shortest Message is Two Bytes

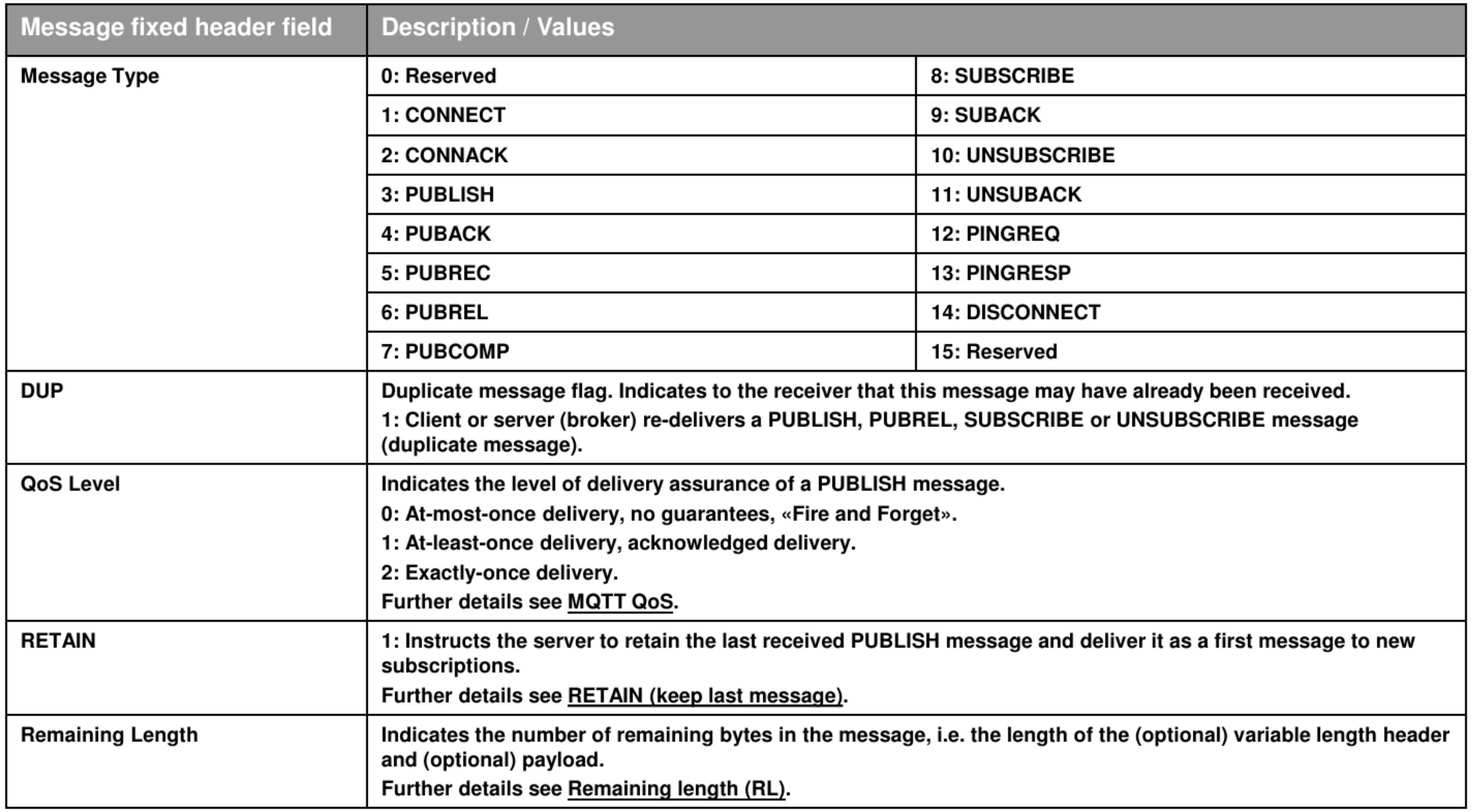


Message Types



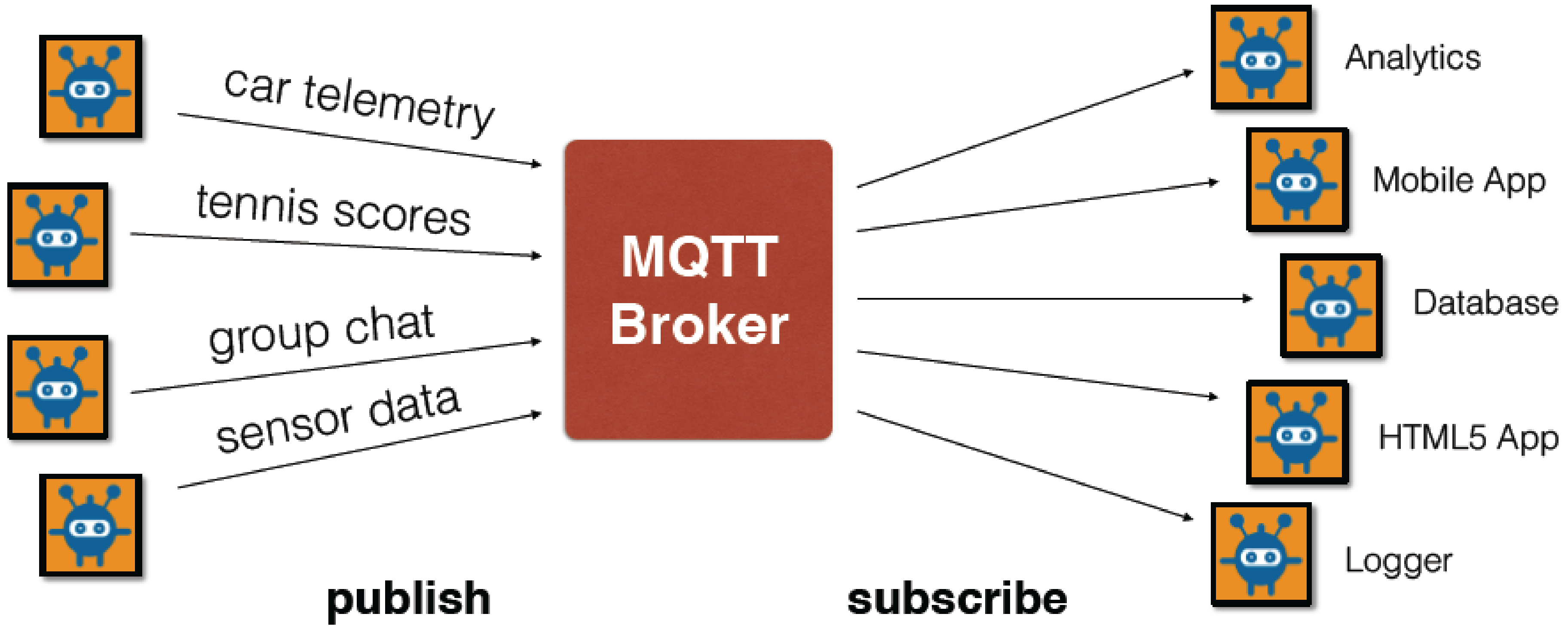
Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Message Types



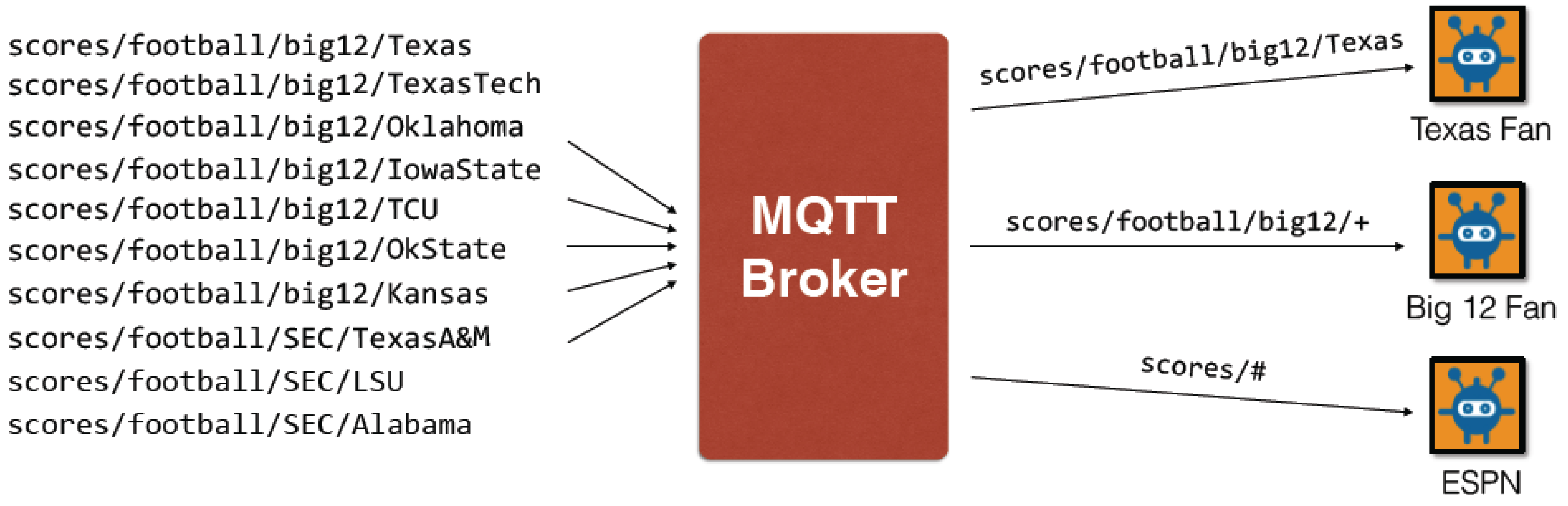
Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	

MQTT: pub/sub <u>decouples</u> sender from receiver



https://www.slideshare.net/BryanBoyd/mqtt-austin-iot-meetup

MQTT: allows wildcard subscription



scores/football/big12/Texas
scores/football/big12/TexasTech
scores/football/big12/Oklahoma
scores/football/big12/IowaState
scores/football/big12/TCU
scores/football/big12/OkState
scores/football/big12/Kansas
scores/football/SEC/TexasA&M
scores/football/SEC/LSU
scores/football/SEC/Alabama
MQTT Broker
scores/football/big12/Texas
Texas Fan
scores/football/big12/+
Big 12 Fan
scores/#
ESPN

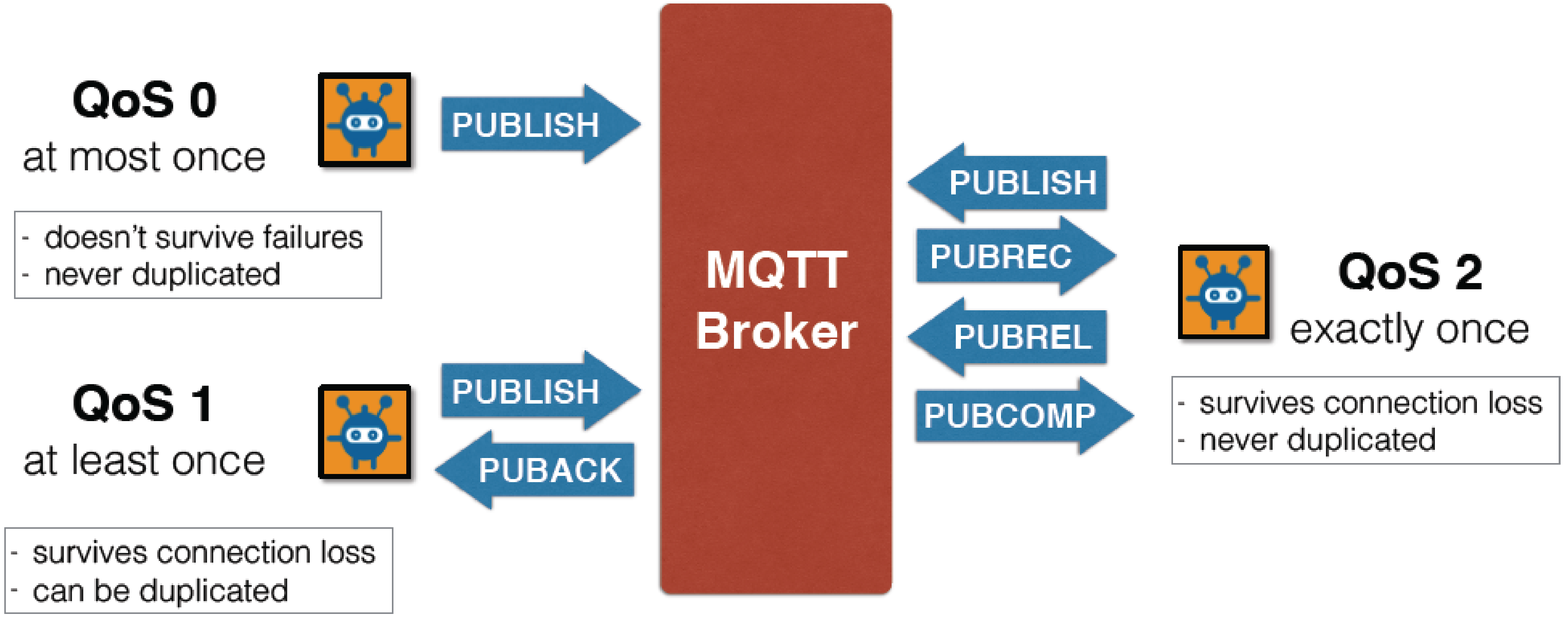
single level wildcard: **+** multi-level wildcard: **#**

get all the bedroom data → **myhome/bedroom/+**
get the temperature data of three rooms → **myhome/+/temperature**
get all the data → **myhome/#** or **#**

Best Practices

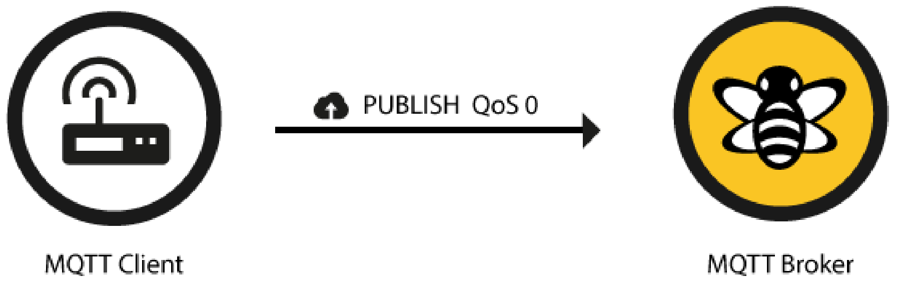
- Do not use topics beginning with $
- Never use a leading forward slash: /myhome/groundfloor/livingroom
- Embed a unique identifier or the Client Id into the topic: client1/status
- Don’t forget extensibility: think about future topic names
- Use specific topics, not general ones:
  - myhome/livingroom/temperature
  - myhome/livingroom/brightness
  - myhome/livingroom/humidity
- Don’t subscribe to #
- Never use spaces in a topic
- Keep the topic short and concise
- Use only ASCII characters; avoid non-printable characters

MQTT: reliable messaging



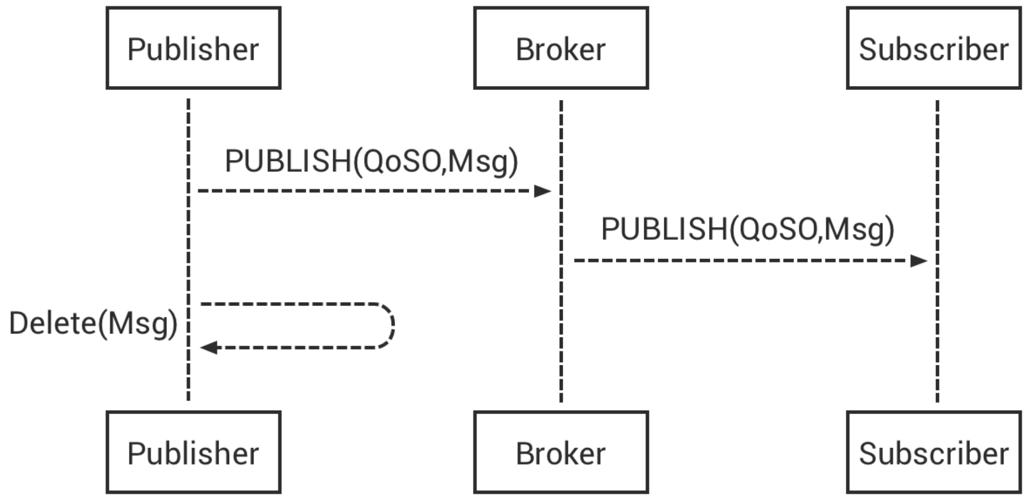
Publishing "QoS" (Reliability)

- 0 – unreliable (aka "**at most once**")
  - OK for continuous streams, least overhead (1 message)
  - "Fire and forget"
  - TCP will still provide reliability



PUBLISH	3	Client to Server or	Publish message

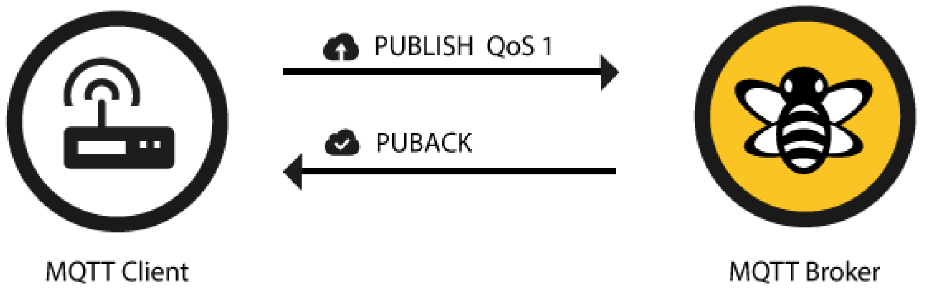
QoS 0:At most once(deliver and forgot)



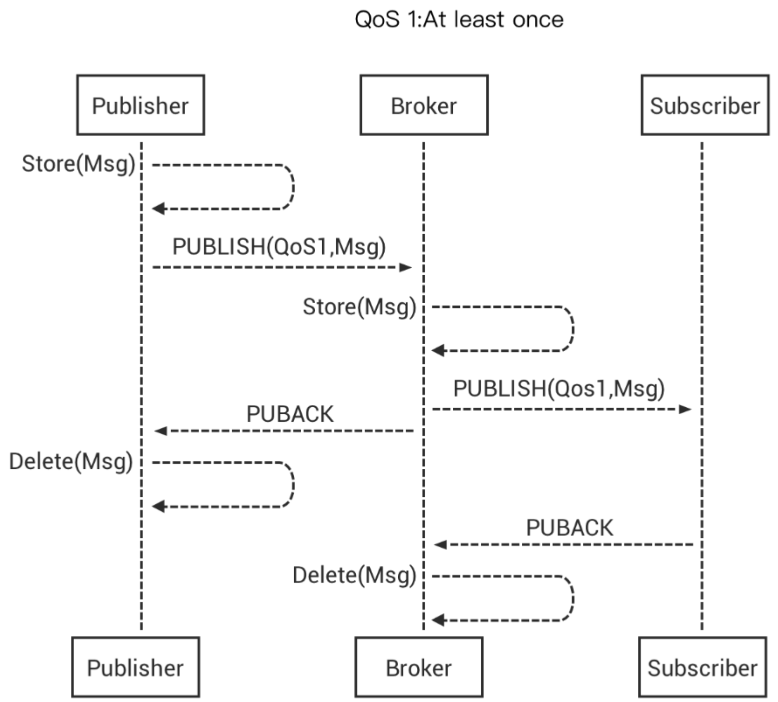
https://www.emqx.com/en/blog/introduction-to-mqtt-qos

Publishing "QoS" (Reliability)

- 1 – delivery "**at least once**" (duplicates possible)
  - Used for alarms – more overhead (2 messages)
  - Contains message ID (to match with **ACK**ed message)

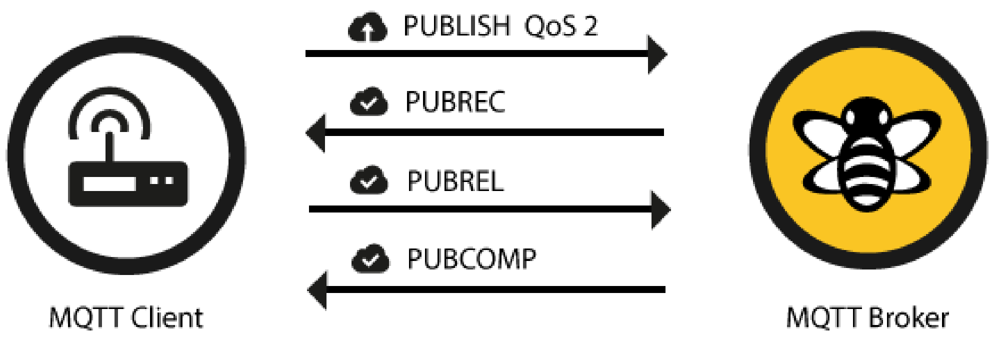


PUBACK	4	Client to Server or Server to Client	Publish acknowledgment



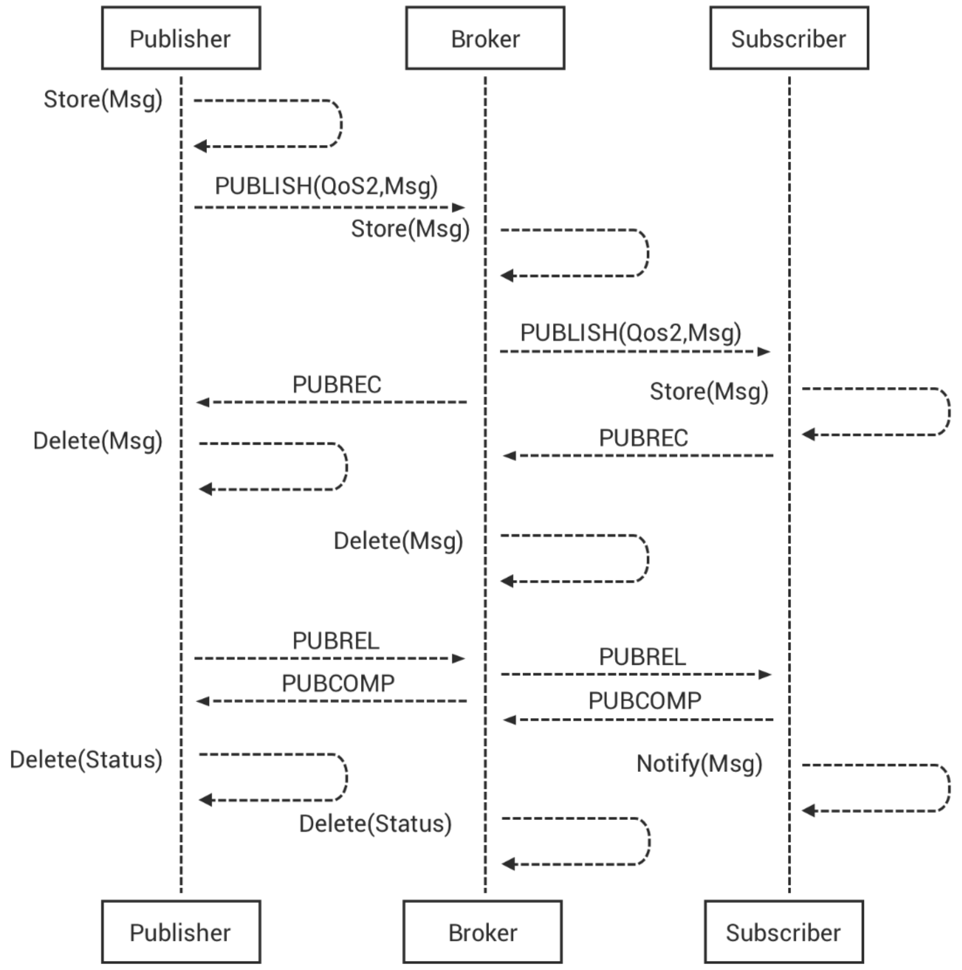
Publishing "QoS" (Reliability)

- 2 – delivery "**exactly once**"
  - Utmost reliability is important – most overhead (4 messages) and slowest

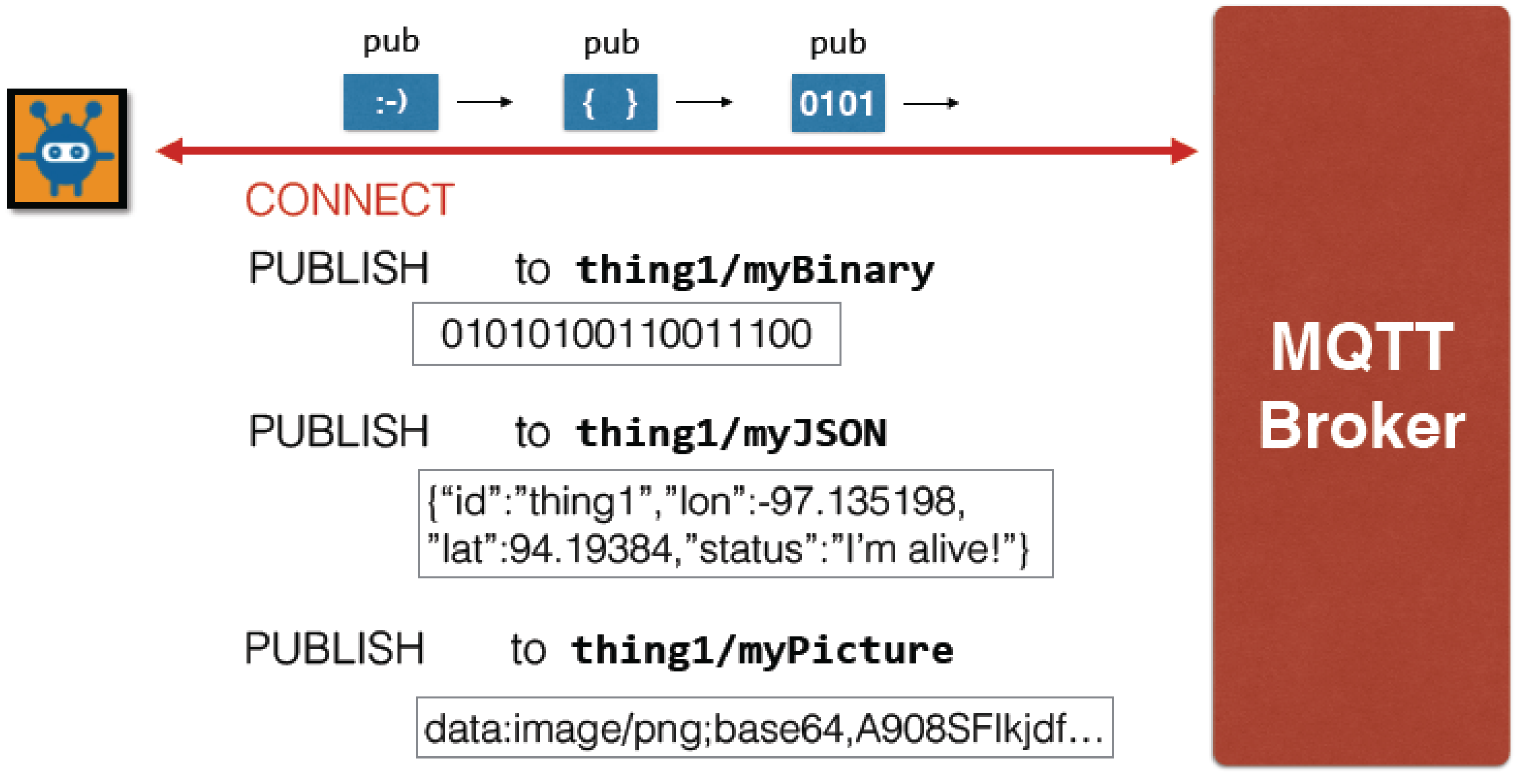


PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)

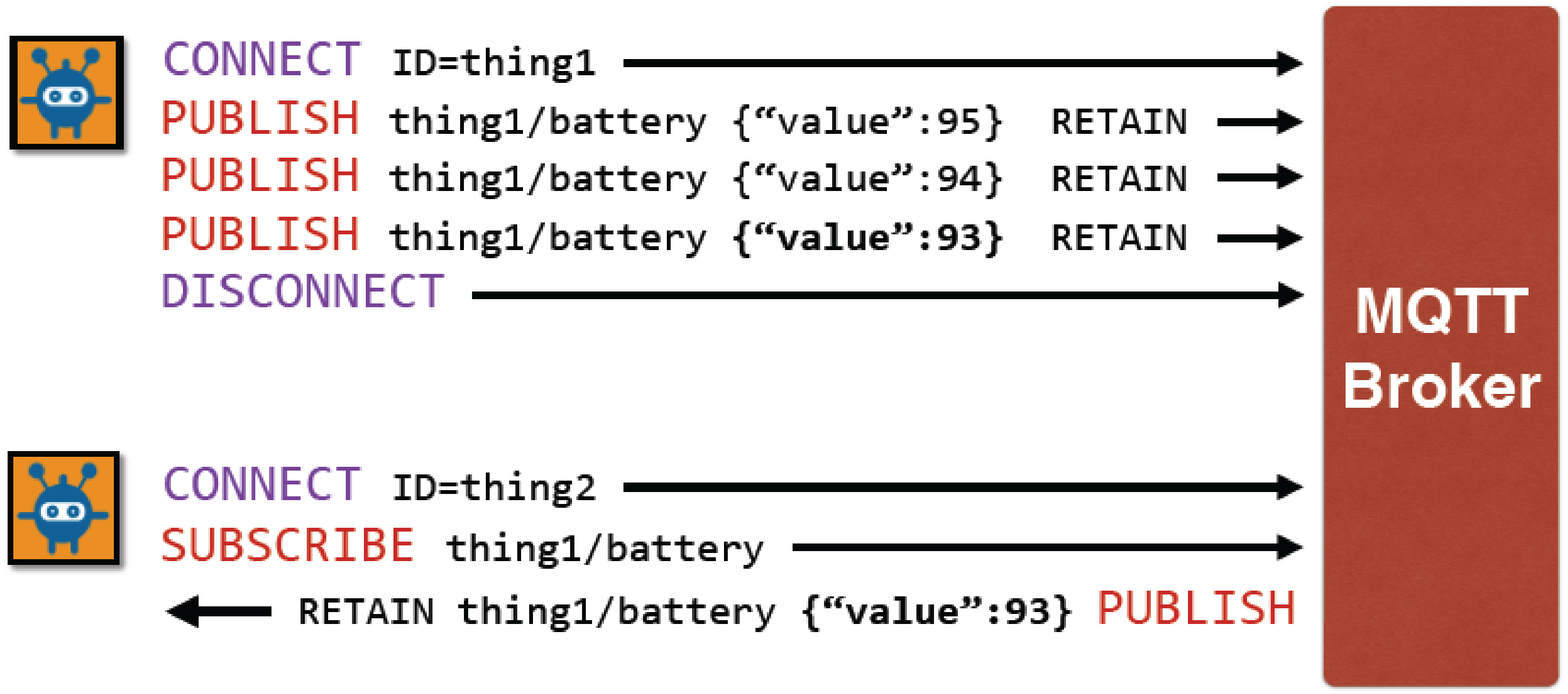
QoS 2:Exactly once



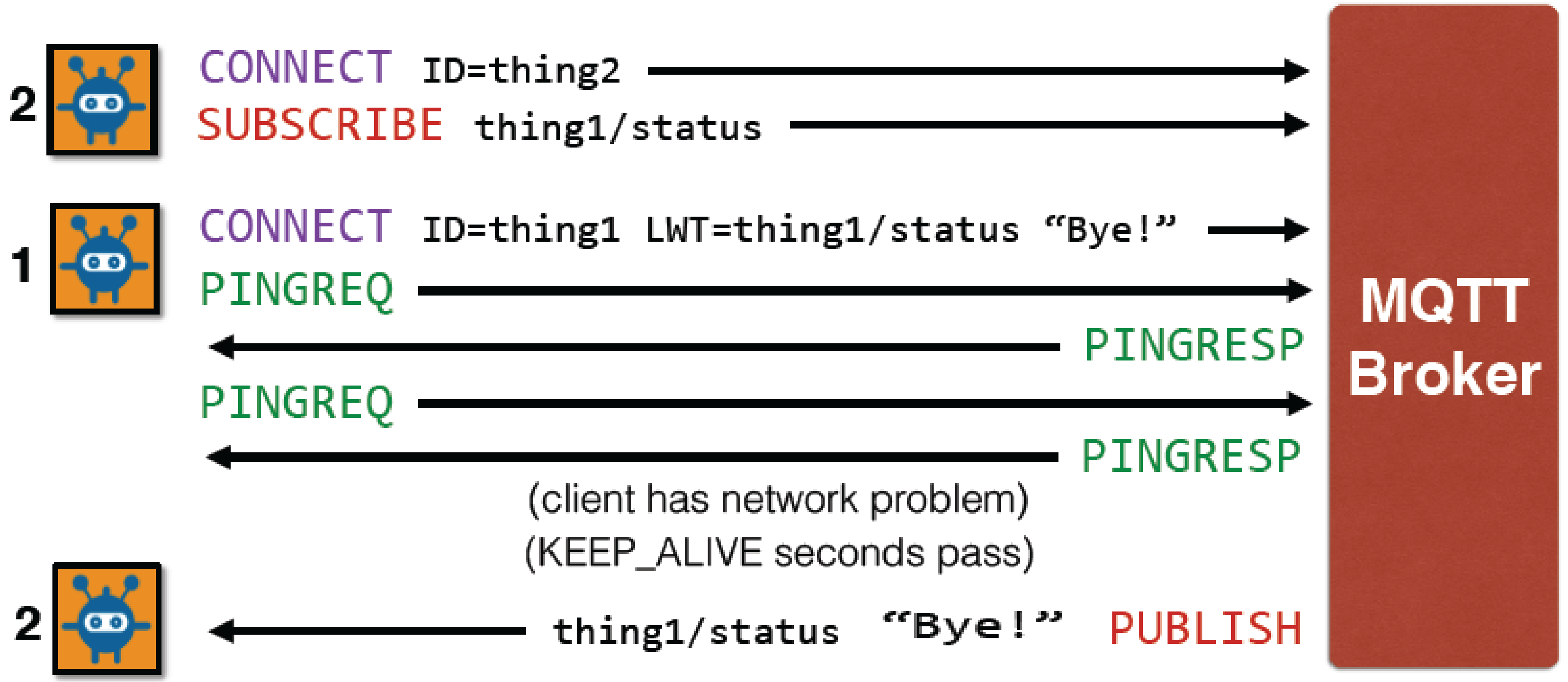
MQTT: Flexible Payload


pub
:-)
pub
{ }
pub
0101
CONNECT
PUBLISH to thing1/myBinary
0101010011001100
PUBLISH to thing1/myJSON
{“id”:”thing1”,”lon”:-97.135198,
”lat”:94.19384,”status”:”I’m alive!”}
PUBLISH to thing1/myPicture
data:image/png;base64,A908SFIkjdf…
MQTT
Broker

MQTT: Retain Message (state retention)


CONNECT ID=thing1
PUBLISH thing1/battery {“value”:95} RETAIN
PUBLISH thing1/battery {“value”:94} RETAIN
PUBLISH thing1/battery {“value”:93} RETAIN
DISCONNECT
MQTT Broker
CONNECT ID=thing2
SUBSCRIBE thing1/battery
RETAIN thing1/battery {“value”:93} PUBLISH

MQTT: Last Will and Testament (LWT)



2
CONNECT ID=thing2
SUBSCRIBE thing1/status
1
CONNECT ID=thing1 LWT=thing1/status “Bye!”
PINGREQ
PINGRESP
PINGREQ
PINGRESP
(client has network problem)
(KEEP_ALIVE seconds pass)
2
thing1/status “Bye!” PUBLISH
MQTT Broker

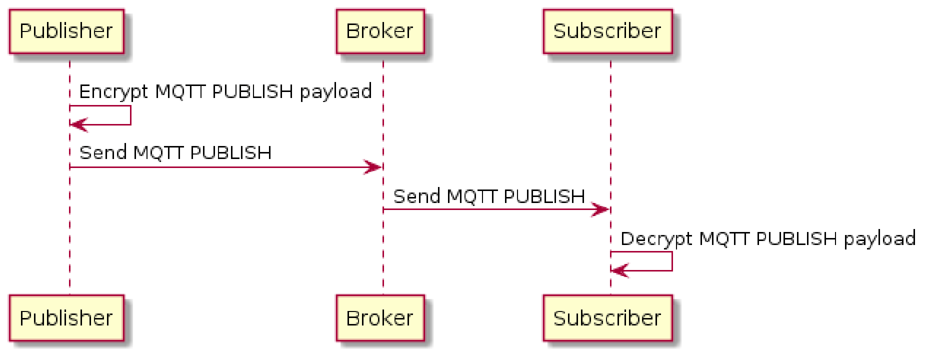
Paho-MQTT (Python-client Example)

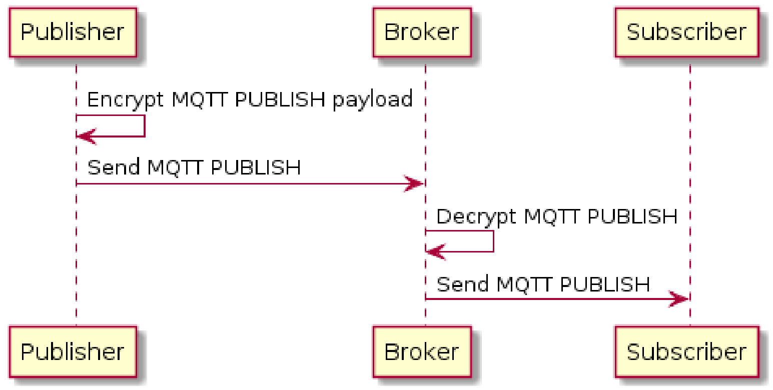
```
#import the client
import paho.mqtt.client as mqtt

#create new instance
client = mqtt.Client("P1")

#connecting to broker (host, port, keepalive, bind_address)
client.connect("iot.eclipse.org“, 1883, 60, “192.168.1.184”)

#Subscribing to topic “house/bulb1” with QoS=1
client.subscribe(“house/bulb1”, 1)
client.subscribe([(“house/bulb3”,2),(“house/bulb4”,1),(“house/bulb5”,0)])

#Publishing message to topic “house/bulb1” with “OFF” payload and QoS of 1 with retain=FALSE
client.publish(“house/bulb1”, “OFF”, 1, FALSE)
```

subscribing to **multiple topics**

Security

- **Transport Layer Security*** (TLS) and **Secure Sockets Layer** (SSL) are cryptographic protocols which use a handshake mechanism to negotiate various parameters to create a secure communication channel between a client and a server.
  - Mitigate "Man-In-The-Middle-Attacks"
  - Ensures that the communication content cannot be read or altered by third parties*
- **Payload Encryption** is the encryption of application-specific data on the application level
  - End-to-end (E2E) encryption
  - Client-to-broker





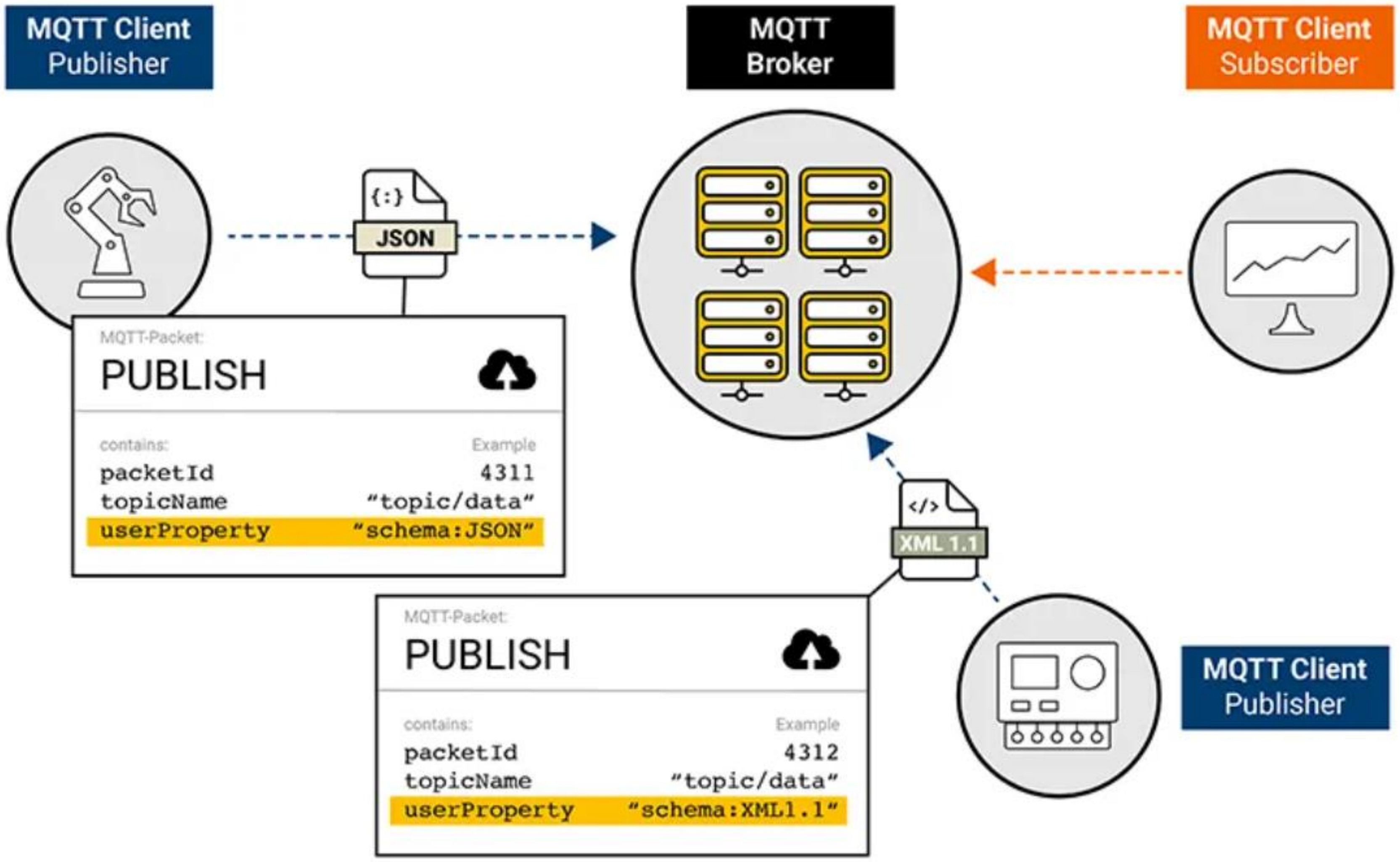
MQTT: Brokers

Appliance	Cloud	Open Source
IBM MessageSight 1m connections 15m QoS 0 / sec policies for security, messaging, connection developer VM	**HiveMQ** **IBM IoT Foundation** **Eurotech EDC** **Litmus Loop** Others	**Mosquitto** (C) **Mosca** (Node.js) **Moquette** (Java) **RSMB** (C) [tiny] Others **Eclipse Sandbox** iot.eclipse.org
Commercial	“Freemium”	Free

MQTT 5 – Whats New

- **User Properties**
- **Shared Subscriptions**
- **Payload Format**
- **Request-Response**
- **Topic Alias**
- **Enhanced Authentication**
- **Flow Control**
- Expiry Interval
  - Message
  - Session
- 128 Reason Codes (19 in MQTT 3.1.1) and Reason String
- Improved Client Feedback – List of supported features
- Prohibition of retransmitting MQTT messages for healthy TCP connections
- Passwords without usernames for authentication

User Properties - What



MQTT-Packet:
PUBLISH

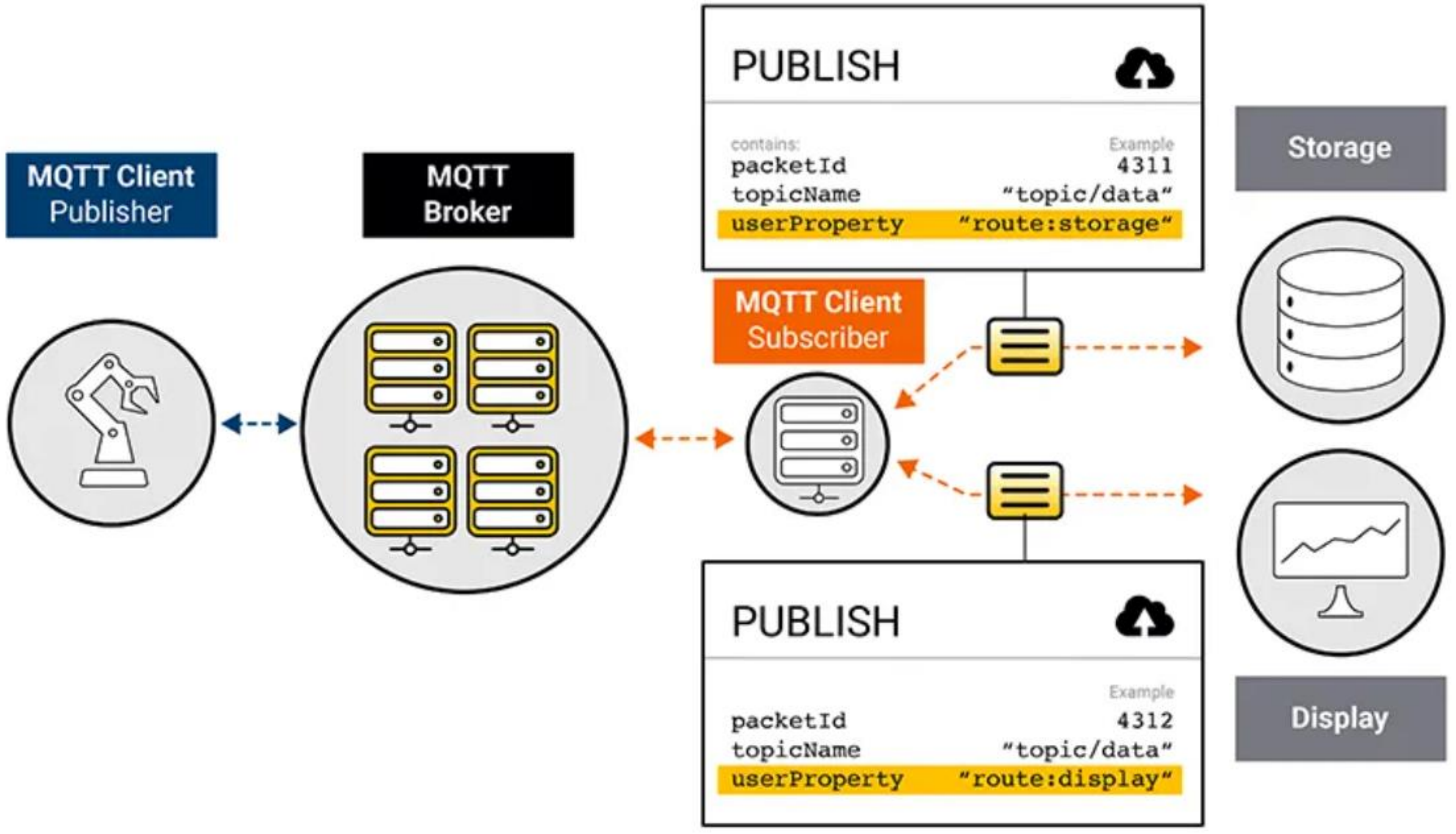
contains:	Example
packetId	4311
topicName	"topic/data"
userProperty	"schema:JSON"

MQTT-Packet:
PUBLISH

contains:	Example
packetId	4312
topicName	"topic/data"
userProperty	"schema:XML1.1"

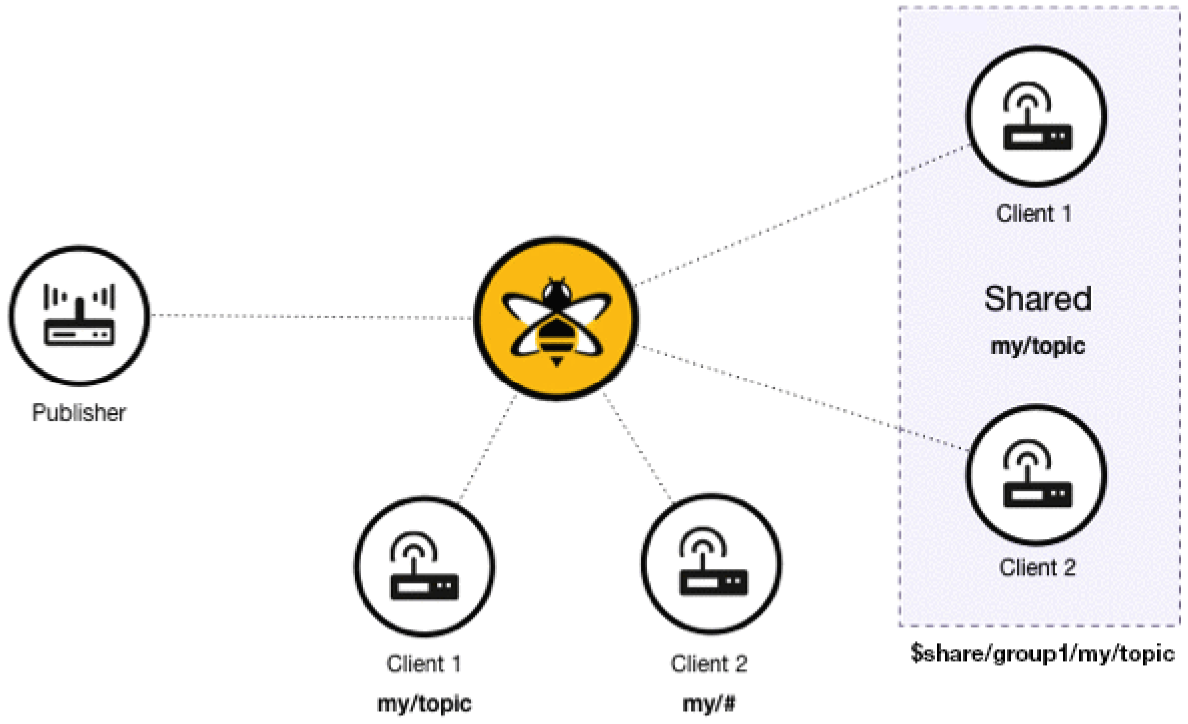
Resource: https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/

User Properties - Why



Resource: https://www.hivemq.com/blog/mqtt5-essentials-part6-user-properties/

Shared Subscriptions



$share/GROUPID/TOPIC

The shared subscription consists of 3 parts:

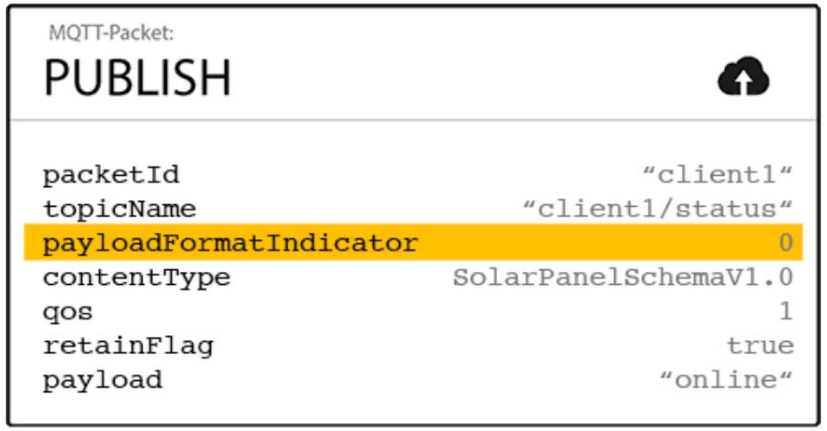
- A static shared subscription identifier ($share)
- A group identifier
- The actual topic subscriptions (may include wildcards)

Example for a subscriber:

$share/my-shared-subscriber-group/myhome/groundfloor/+/temperature

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part7-shared-subscriptions/

Payload Format

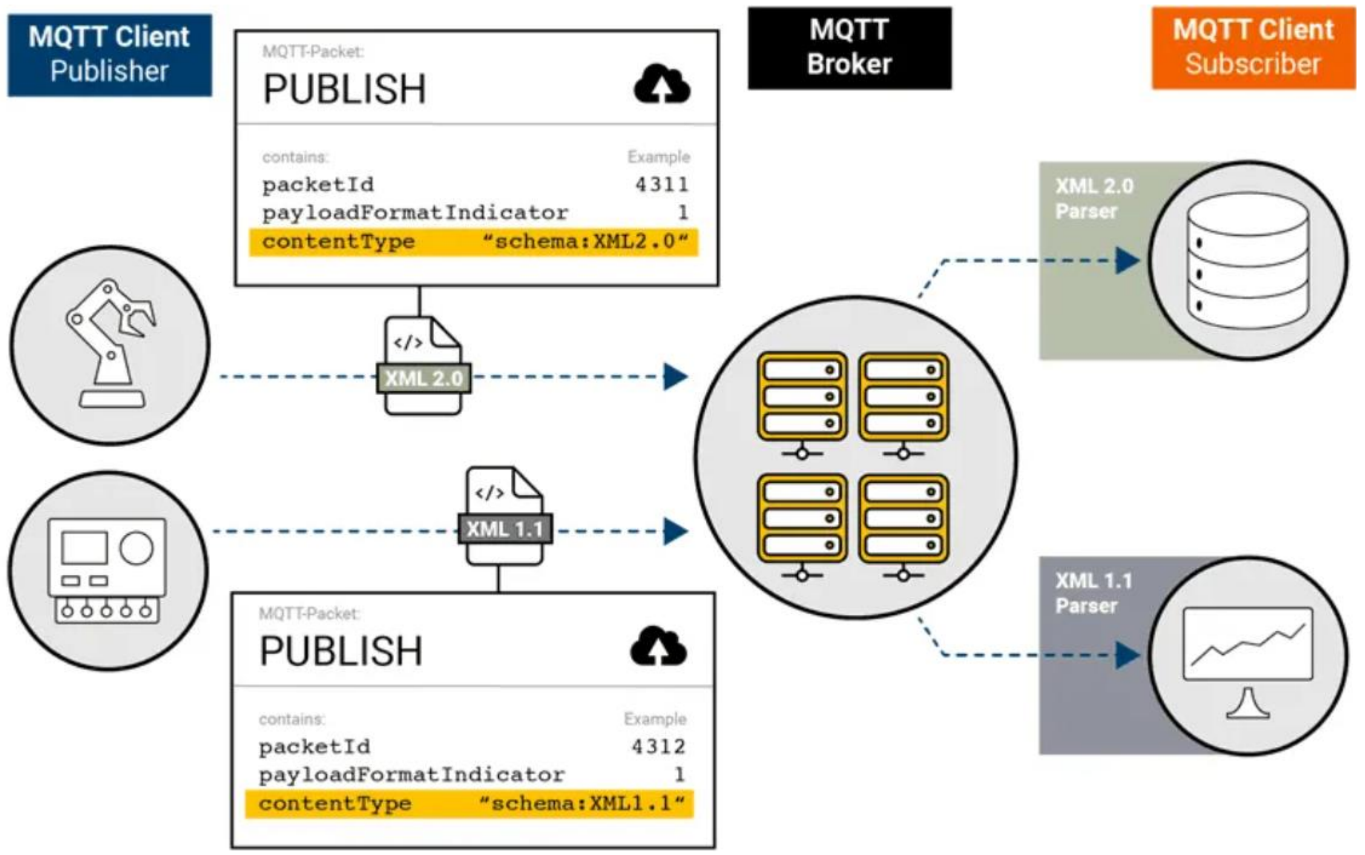


MQTT-Packet:

PUBLISH

packetId	"client1"
topicName	"client1/status"
payloadFormatIndicator	0
contentType	SolarPanelSchemaV1.0
qos	1
retainFlag	true
payload	"online"

- 0 indicates an "unspecified byte stream"
- 1 represents a "UTF-8 encoded payload."
  - MIME-like content type descriptor
- When the Payload Format Indicator isn't provided, it automatically defaults to 0.



MQTT-Packet:

PUBLISH

contains:	Example
packetId	4311
payloadFormatIndicator	1
contentType	"schema:XML2.0"

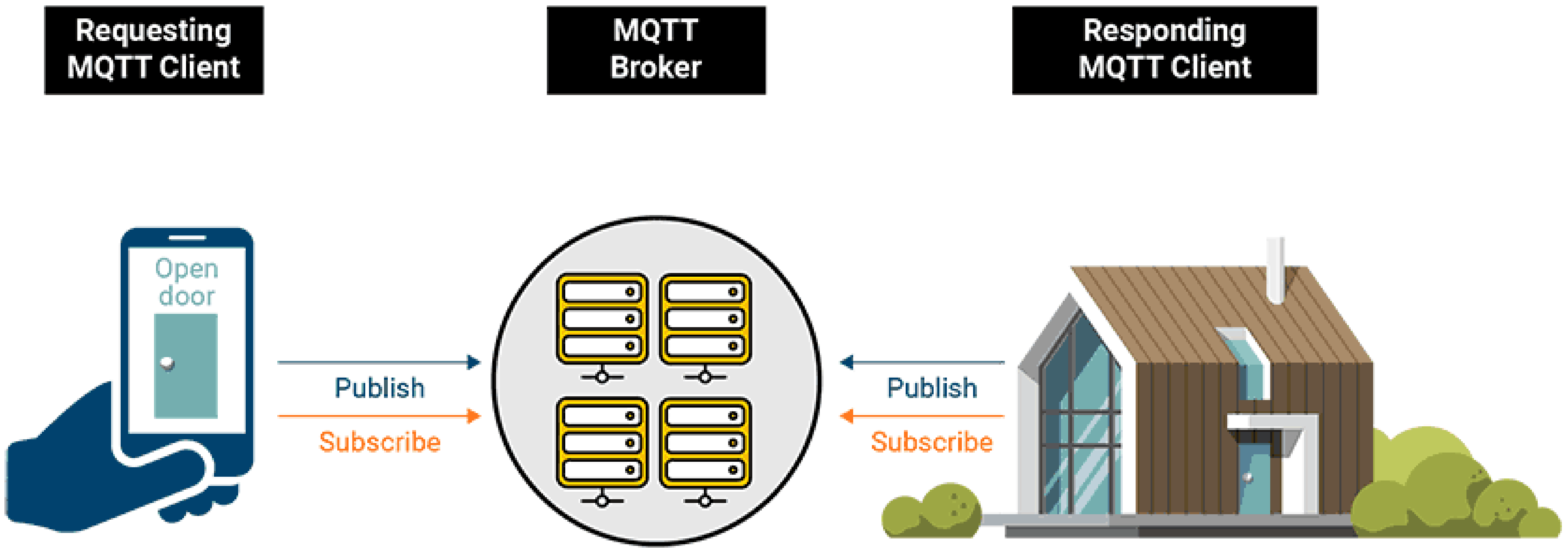
MQTT-Packet:

PUBLISH

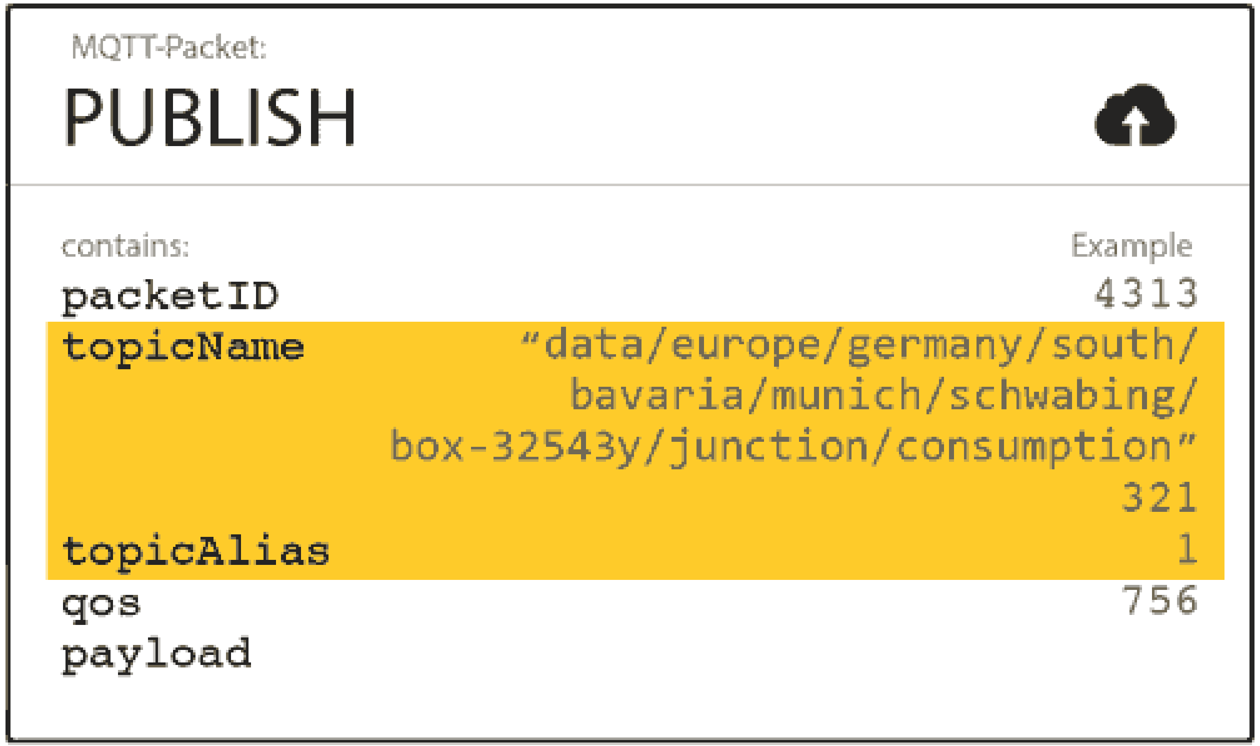
contains:	Example
packetId	4312
payloadFormatIndicator	1
contentType	"schema:XML1.1"

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part8-payload-format-description/

Request-Response



Topic Alias



MQTT-Packet:

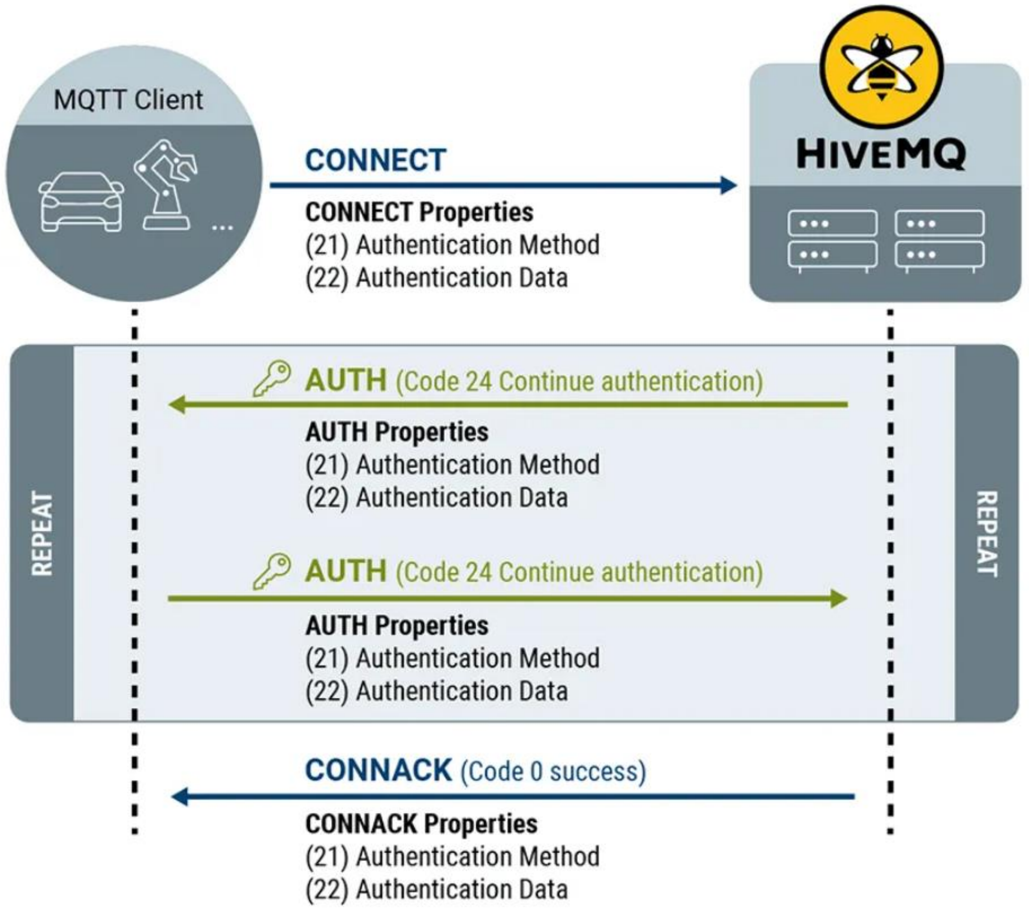
PUBLISH

contains:	Example
packetID	4313
topicName	"data/europe/germany/south/bavaria/munich/schwabing/box-32543y/junction/consumption"
	321
topicAlias	1
qos	756
payload	

- Payload are relatively small
- Topics are relatively long
- Message throughput is relatively high

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part10-topic-alias/

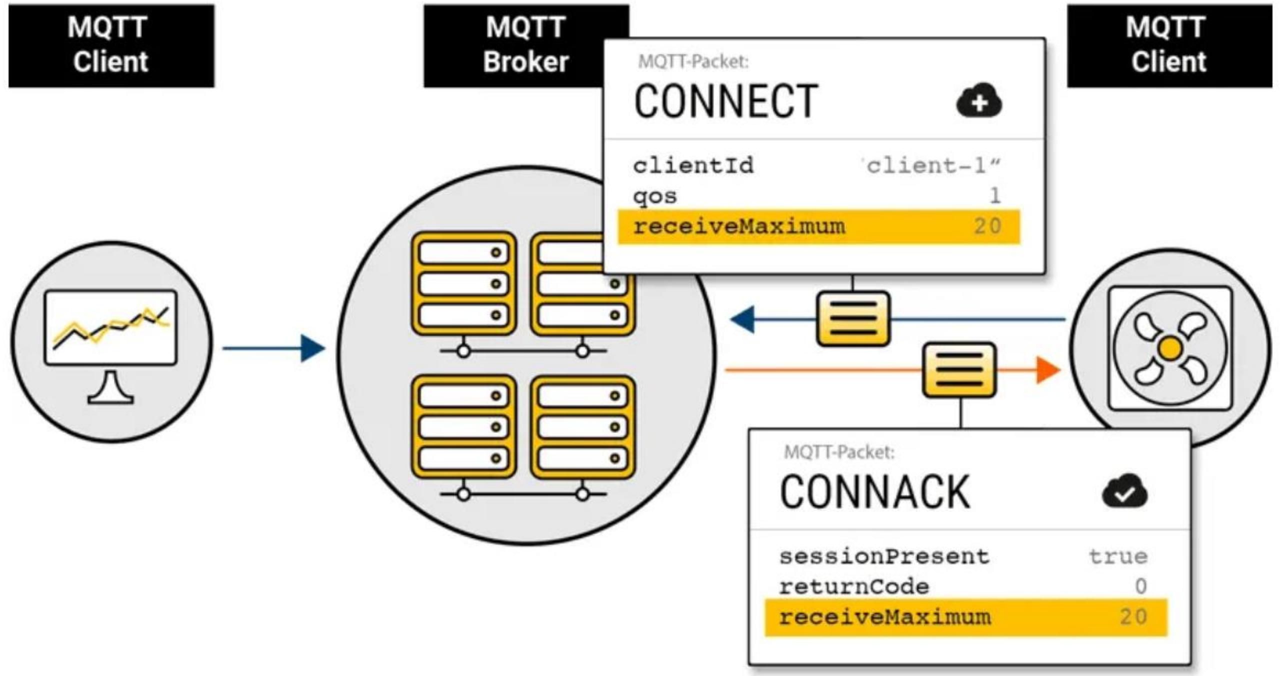
Enhanced Authentication



- Challenge-response
  - vs credential-based
- Authentication methods:
  - Salted Challenge Response Authentication Mechanism (SCRAM)
  - Kerberos
- Enabled for every message of the entire flow

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part11-enhanced-authentication/

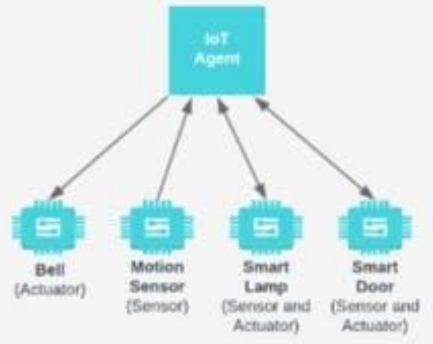
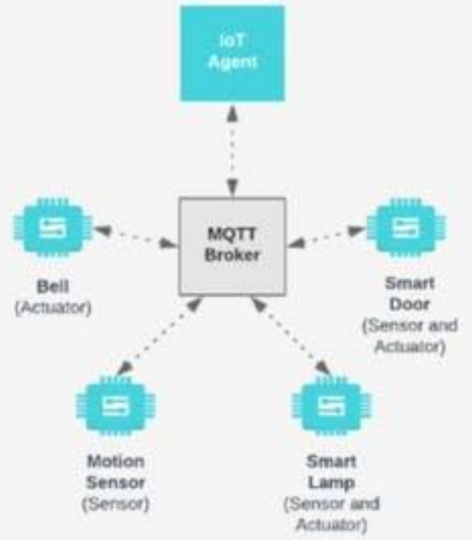
Flow Control



- Flow Control is designed to maintain balanced message processing, preventing the overloading of any participating parties.
- Receive Maximum: Max number of unacknowledged PUBLISH messages the client can accommodate
- "Receive Maximum" value is not specified, the default value of 65,535
- Helps with not overloading low resourced IoT

Resource: https://www.hivemq.com/blog/mqtt5-essentials-part12-flow-control/

MQTT: Summary

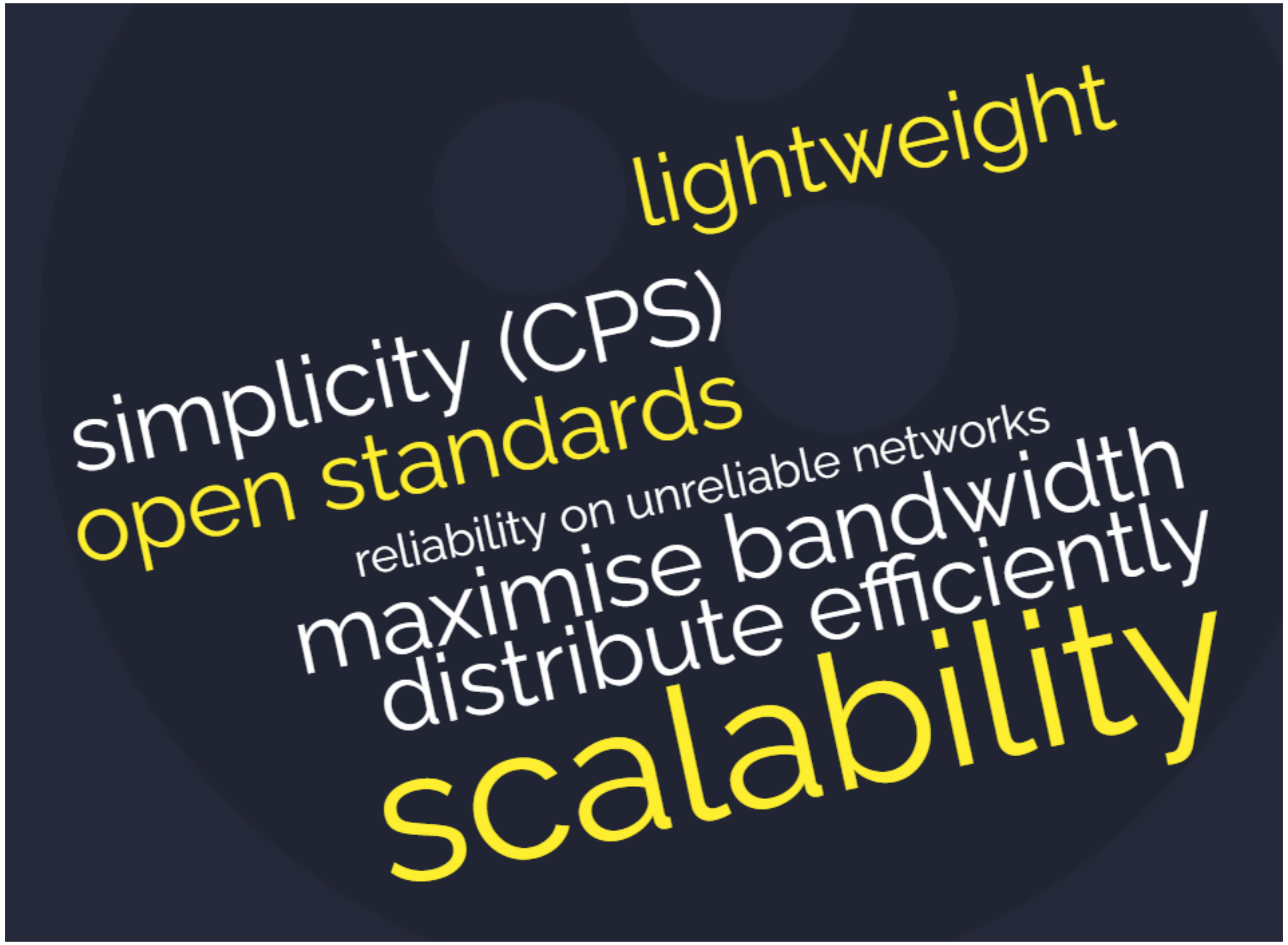
HTTP Transport	MQTT Transport
	
IoT Agent communicates with IoT devices **directly**	IoT Agent communicates with IoT devices **indirectly** via an MQTT Broker
Request-Response Paradigm	**Publish-Subscribe** Paradigm
IoT Devices must always be ready to receive communication	IoT Devices choose when to receive communication
Higher Power Requirement	Low Power Requirement

IoT Agent
Bell (Actuator)
Motion Sensor (Sensor)
Smart Lamp (Sensor and Actuator)
Smart Door (Sensor and Actuator)

IoT Agent
MQTT Broker
Bell (Actuator)
Smart Door (Sensor and Actuator)
Motion Sensor (Sensor)
Smart Lamp (Sensor and Actuator)

https://create.arduino.cc/projecthub/akarsh98/what-is-mqtt-how-to-use-tutorial-mqtt-reyax-broker-d5a6b4

MQTT: Summary


lightweight
simplicity (CPS)
open standards
reliability on unreliable networks
maximise bandwidth
distribute efficiently
scalability