SORBONNE UNIVERSITÉ

CRÉATEURS DE FUTURS
DEPUIS 1257

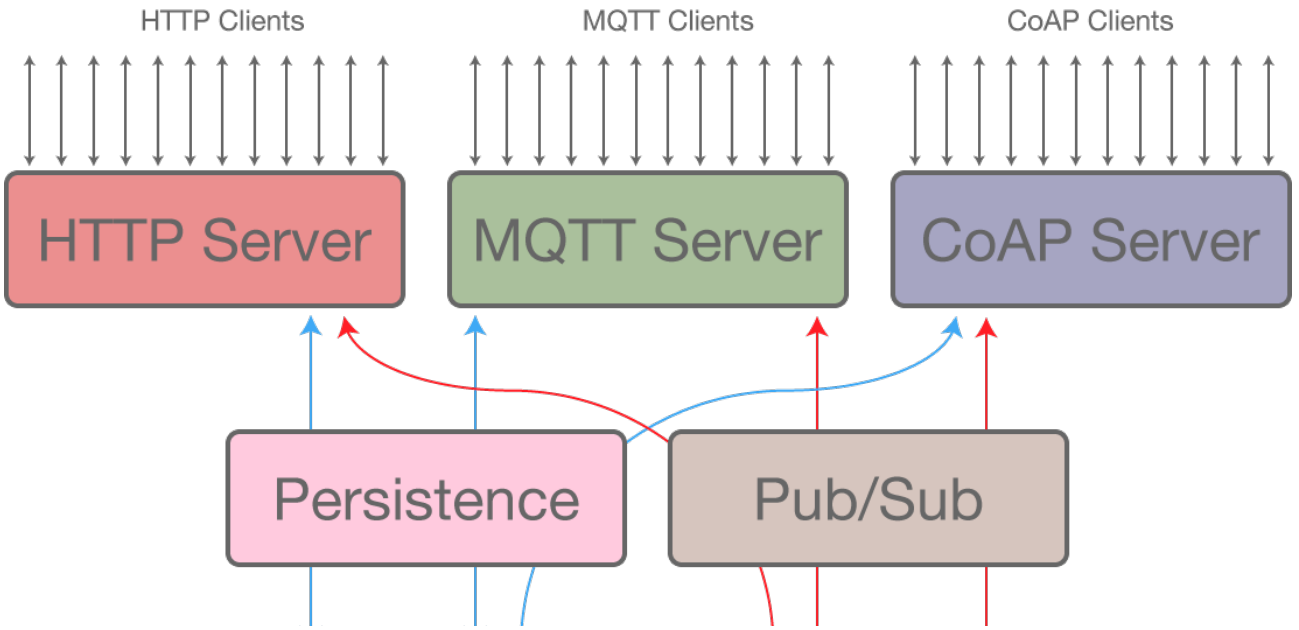
Yann DOUZE
Twitter : @yann_douze
Linkedin :
https://www.linkedin.com/in/yanndouze/

The Internet Of Things
C4 : TCP/IP Protocols HTTP, MQTT, CoAP

Introduction to what is the Internet of Things, why does it change the world where we live, what are the technologies behind the scene ?
How des it apply to your domain ?

TCP/IP, Wifi : Connect objects to the Cloud

When your device is connected with Wi-Fi. Which protocol to choose ? HTTP REST, MQTT or CoAP



Data format?

- **JSON (JavaScript Object Notation)**

```
{
"name":"John",
"age":30,
"car": "Ford",
}
```

- **XML (Extensible Markup Language)**

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes" ?>
<repertoire>
  <personne>Robert</personne>
  <personne>John</personne>
</repertoire>
```

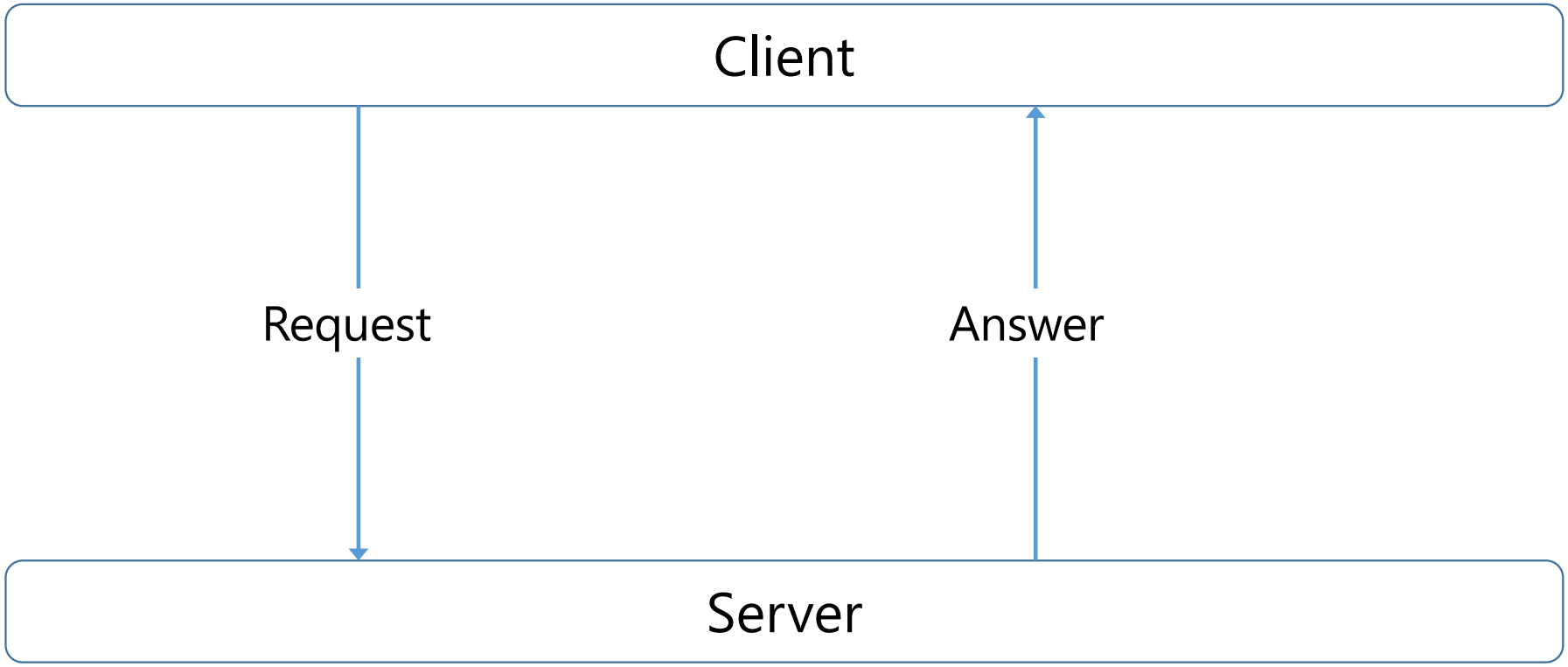
Voir le rapport de stage de Bruce Rosier

- Client / Serveur
- Publish / Subscribe

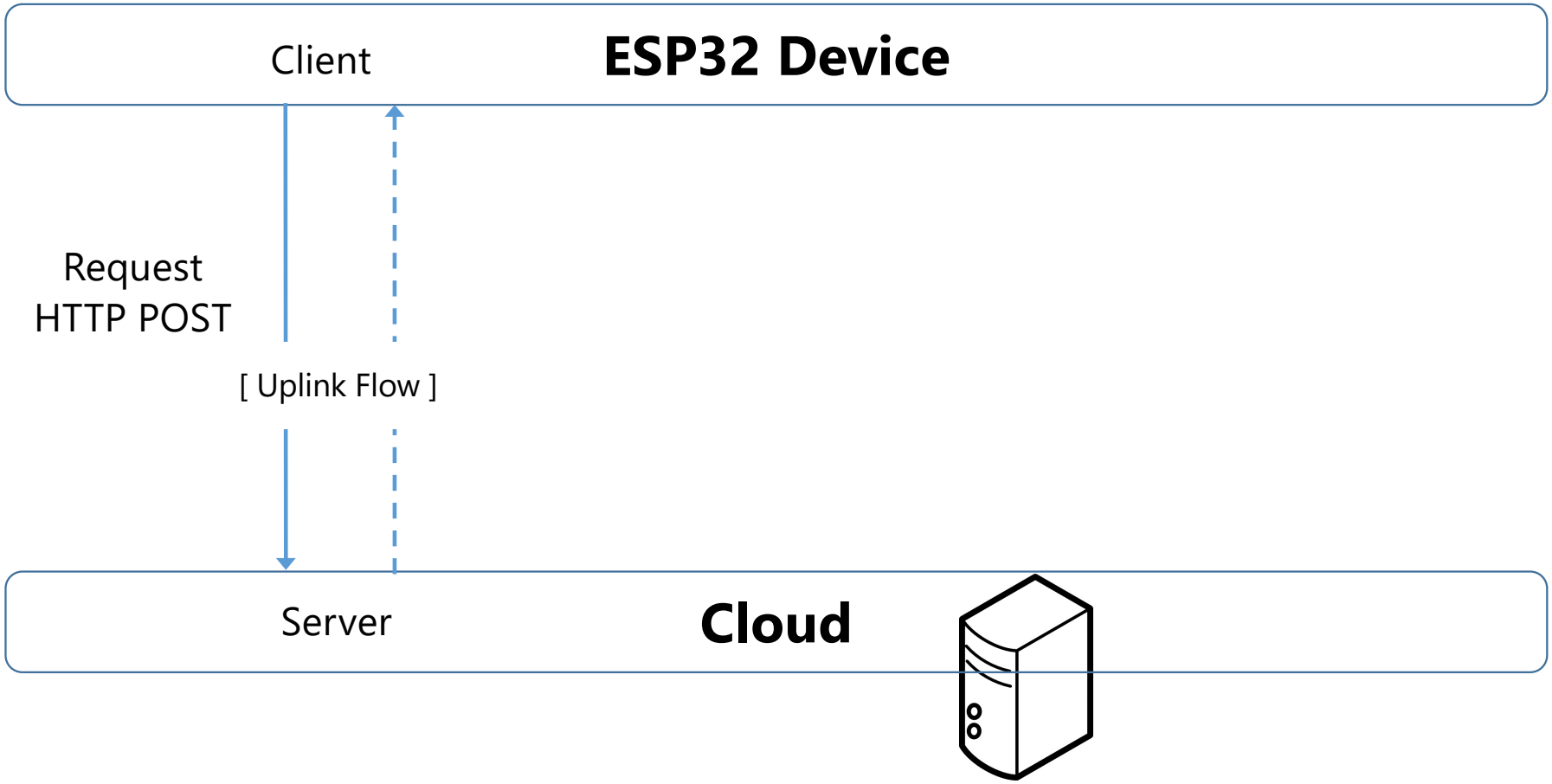
HTTP RESTful API

- REST = Representational State Transfer
- Architecture style for distributed hypermedia systems.
- Created by Roy Fielding in 2000
- Based on the HTTP protocol, uses methods such as GET, PUT, POST, DELETE
- Disadvantage: very verbose so very bandwith consumer.

HTTP : The Client - Server principle



HTTP: Client - Server with POST requests (uplink)



REST example with Microsoft Azure

Connect in TCP with the Server:

"iotarduino.azure-mobile.net",80

Send the data with the JSON :

POST /tables/ss_capteur_temperature HTTP/1.1\r\n

Host: iotarduino.azure-mobile.net\r\n

X-ZUMO-APPLICATION: fmgZCKkivPMnfNJYcHsGyCQufOgOua71\r\n

Content-Type: application/json\r\n

Content-Length: 63\r\n

{"multiprise_key":"90a2daf03576","prise_key":"p4","value": 35}\n

Réponses aux requêtes REST

Success

2.xx, indicates that the request has been correctly received, understood and accepted.

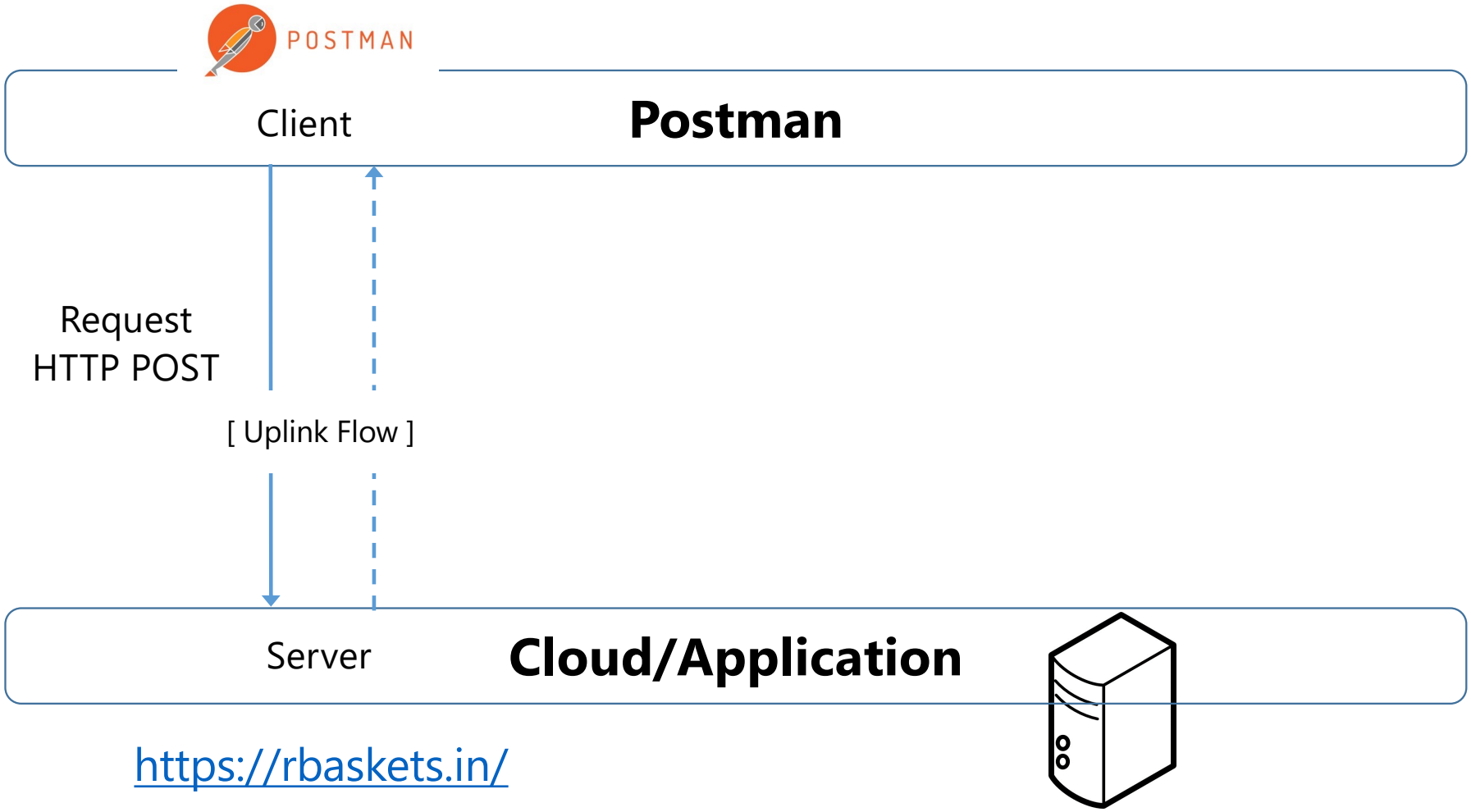
Client Error

4.xx indicates that the client has encountered an error.

Internal Server Error

5.xx indicates that the server is unable to process the request.

HTTP: Client - Server with POST requests
Démonstration with Postman and rbasket.in

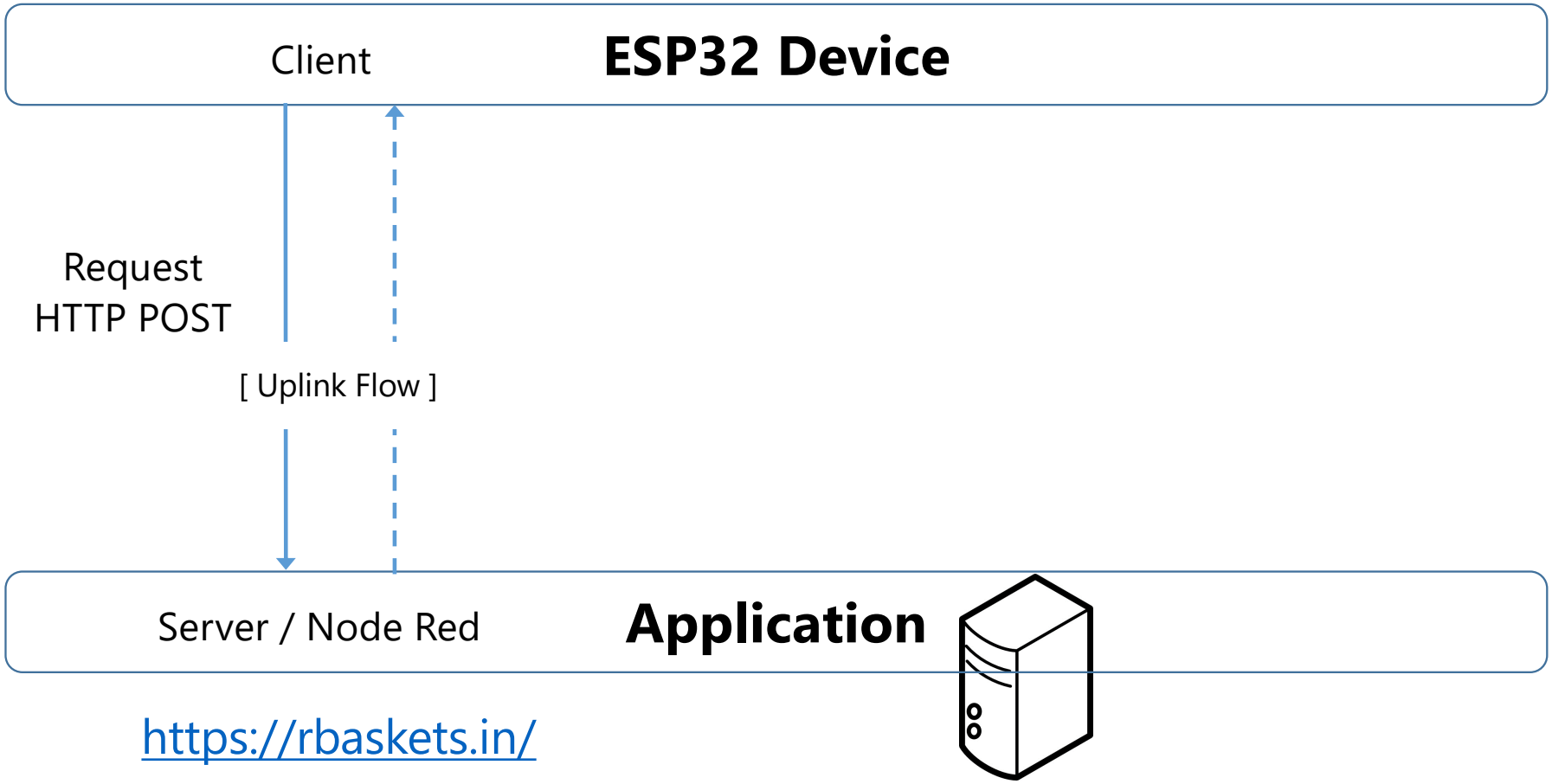


https://rbaskets.in/

HTTP: Client - Server with GET requests

https://rbaskets.in/

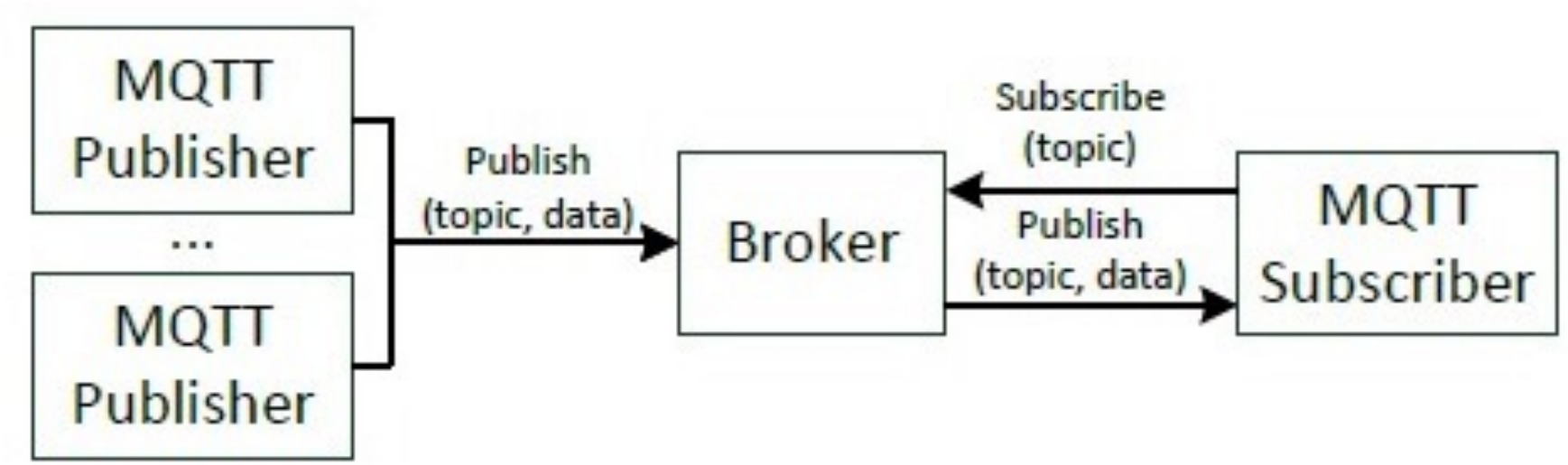
HTTP: Client - Server with POST requests



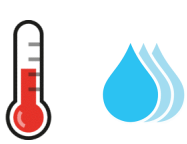
MQTT : introduction

- MQTT (Message Queue Telemetry Transport)
  - ✓ Open : created by IBM & Eurotech and donated to Eclipse “Paho” M2M project (OASIS standard in 2014)
  - ✓ Lightweight : smallest packet size 2 bytes (header), reduced clients footprint (C# M2Mqtt library 30 Ko)
  - ✓ Reliable : three Quality of Service (QoS) and patterns to avoid packet loss on client disconnection.
  - ✓ Simple :
    - TCP based
    - Asynchronous
    - Publish/Subscribe architecture
    - Few verbs
    - Payload agnostic

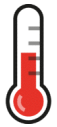
MQTT Communication



MQTT : The Publisher - Broker - Subscriber principle



Client
Publisher 1



Client
Publisher 2

Topic
Temperature

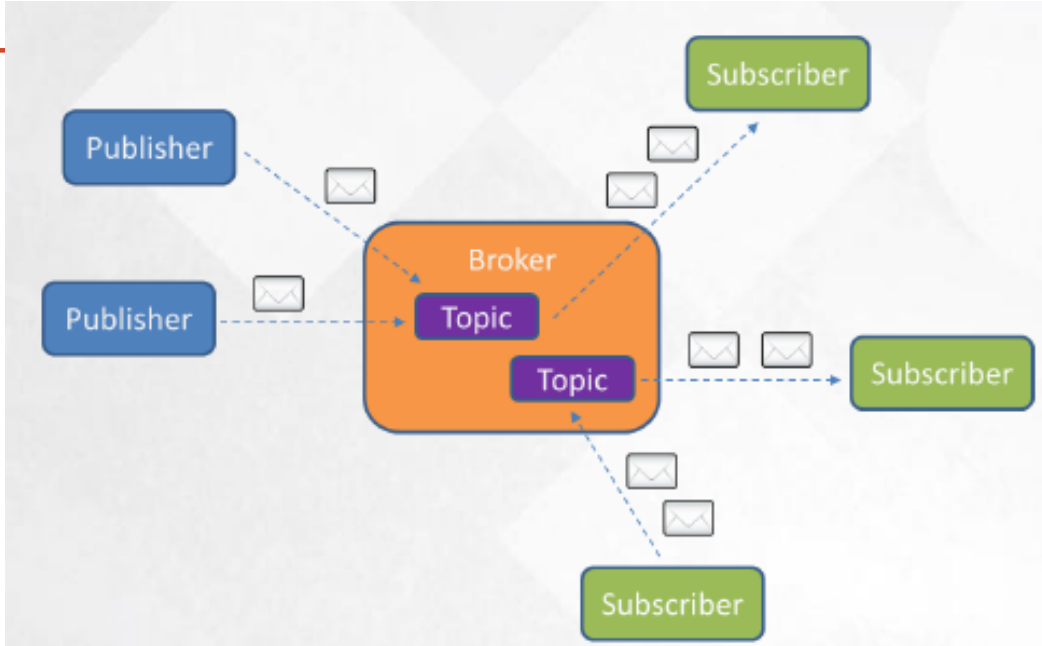
Broker

Topic
Humidity

Client
Subscriber 1

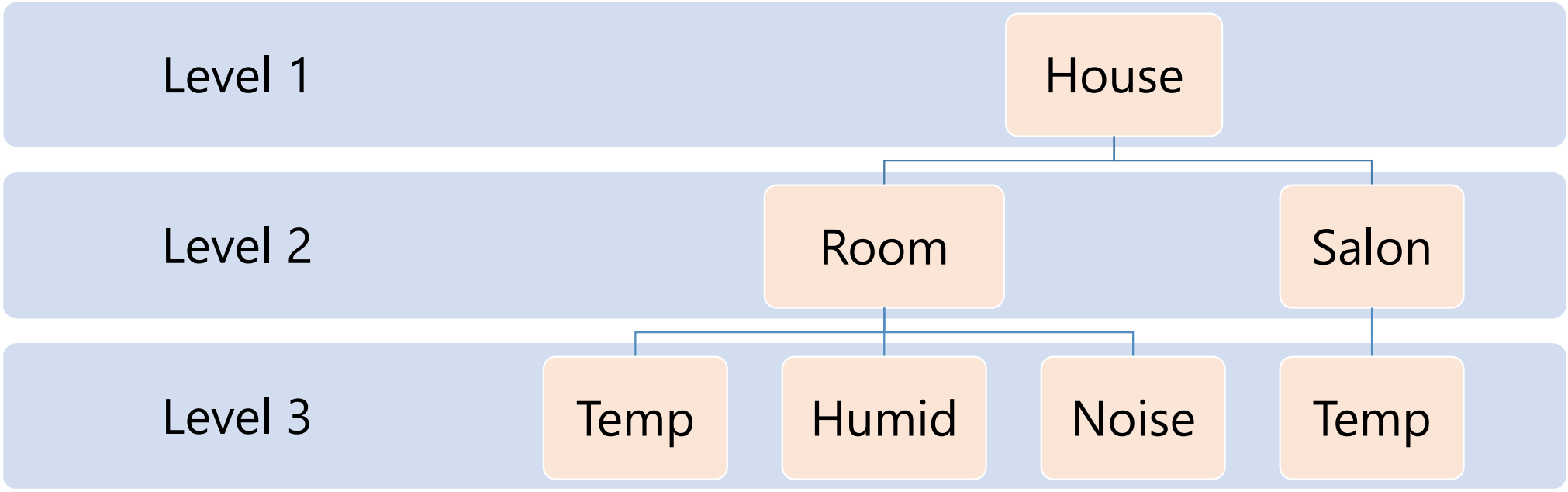
Client
Subscriber 2

MQTT Publish/Subscribe



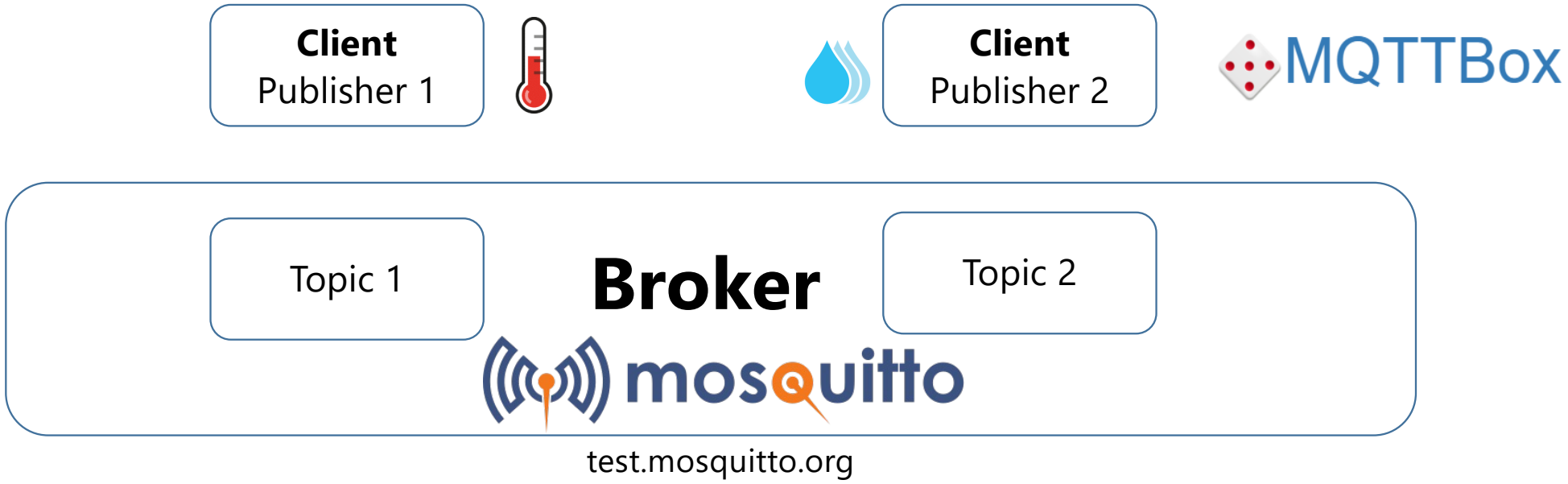
- Broker and connected Clients
  - ✓ Broker receives subscription from clients on topics
  - ✓ Broker receives messages and forward them
  - ✓ Clients subscribe/publishes on topics
- Topics for publish and subscribe (like queue)

MQTT : The Topics



Topic detail	Topic name
The **temperature** of the **room** in the **house**	house/room/temp
The **temperature of** the **Home Salon**	house/salon/temp
The **temperature**, **humidity** and **noise of** the whole **house**	house/#
Temperatures in **all rooms of** the **house**	House/+/temp

MQTT : Demonstration with Mosquitto



MQTT: Quality of Service

QoS 0
''At most once''

PUBLISH

Message deleted after sending

QoS 1
''At least once''

PUBLISH

PUBLISH Ack

Recorded message
Surviving a loss of connection
Possible duplications

Broker

QoS 2
''Exactly once''

PUBLISH

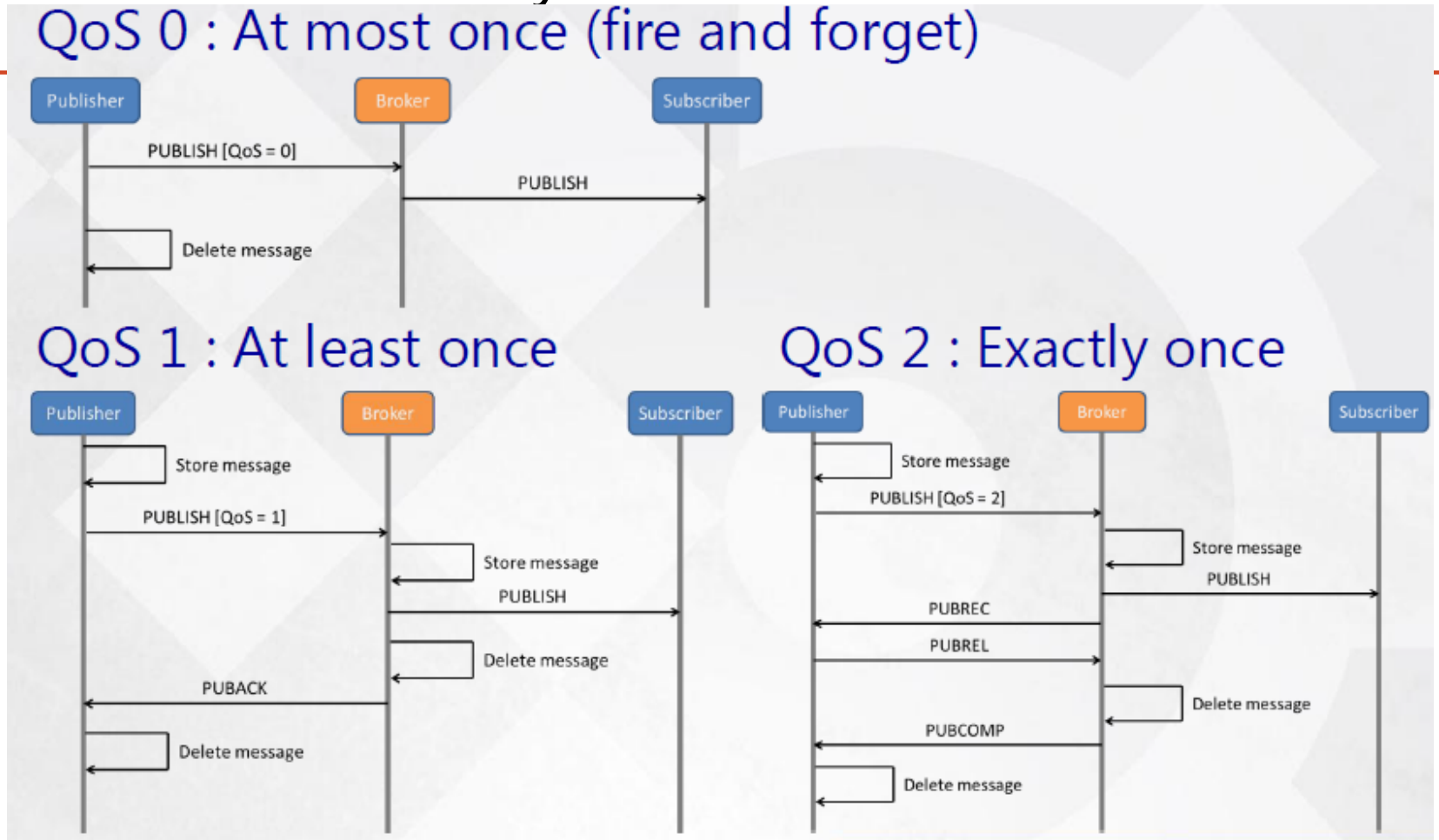
PUBLISH Receive

PUBLISH Release

PUBLISH Complete

Recorded message
Survival of connection loss
No duplication

MQTT : Quality of Service



- MQTT security : over TCP, use SSL/TLS

MQTT Header

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Byte 1

Contains the Message Type and Flags (DUP, QoS level, and RETAIN) fields.

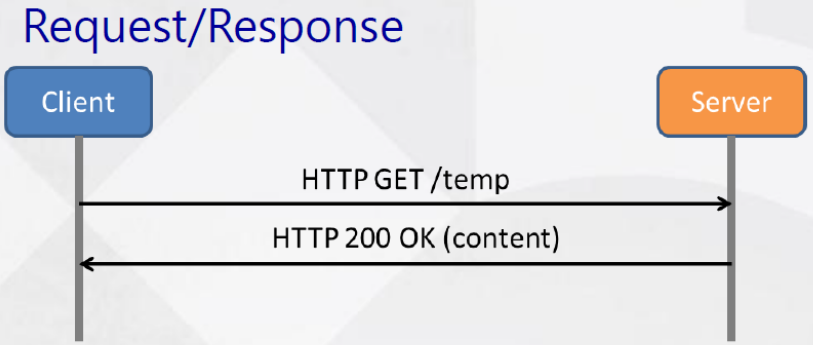
Byte 2 :contains the Remaining Length field.

Détail du protocole :
http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

HTTP vs MQTT

	MQTT	HTTP
Design orientation	Data centric	Document centric
Pattern	Publish/subscribe	Request/response
Complexity	Simple	More complex
Message size	Small, with a compact binary header just two bytes in size	Larger, partly because status detail is text-based
Service levels	Three quality of service settings	All messages get the same level of service
Extra libraries	Libraries for C (30 KB) and Java (100 KB)	Depends on the application (JSON, XML), but typically not small
Data distribution	Supports 1 to zero, 1 to 1, and 1 to n	1 to 1 only

HTTP vs MQTT



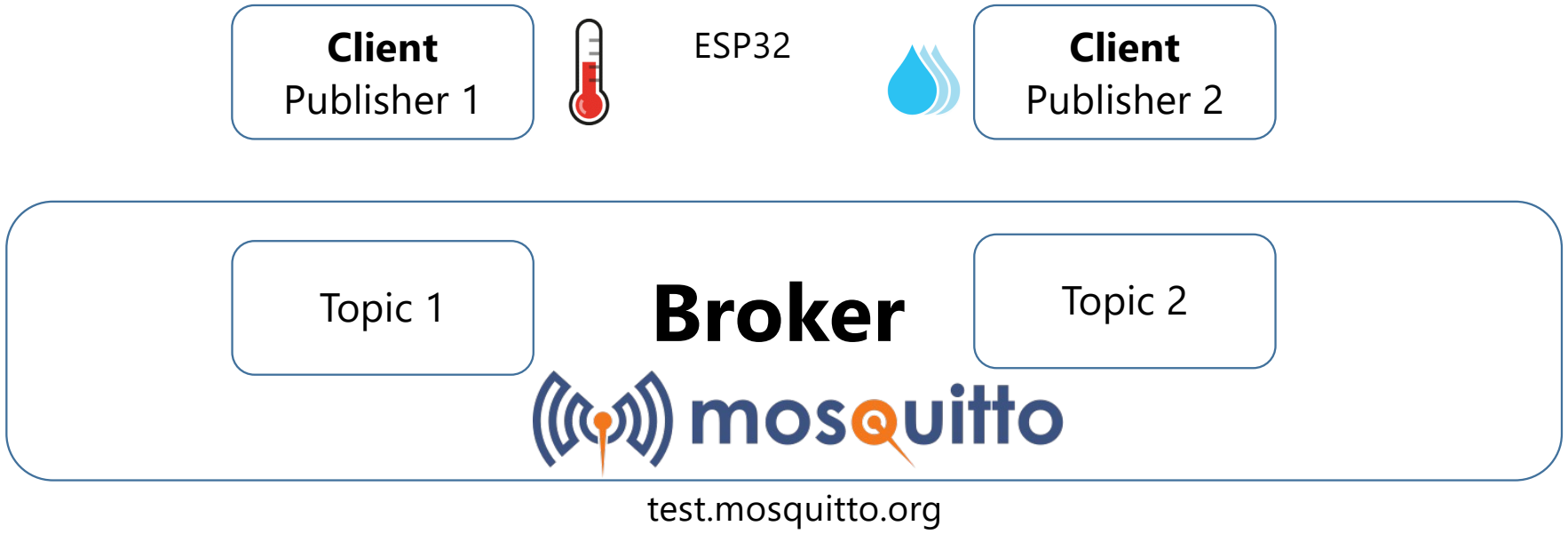
Sending (1024 msg – 1 byte)

	3G		Wifi	
	HTTPS	MQTT	HTTPS	MQTT
% Battery / Hour	18.43%	**16.13%**	**3.45%**	4.23%
Messages / Hour	1708	**160278**	3628	**263314**
% Battery / Message *	0.01709	**0.00010**	0.00095	**0.00002**
Messages Received	240 / 1024	**1024 / 1024**	524 / 1024	**1024 / 1024**

Receiving (1024 msg – 1 byte)

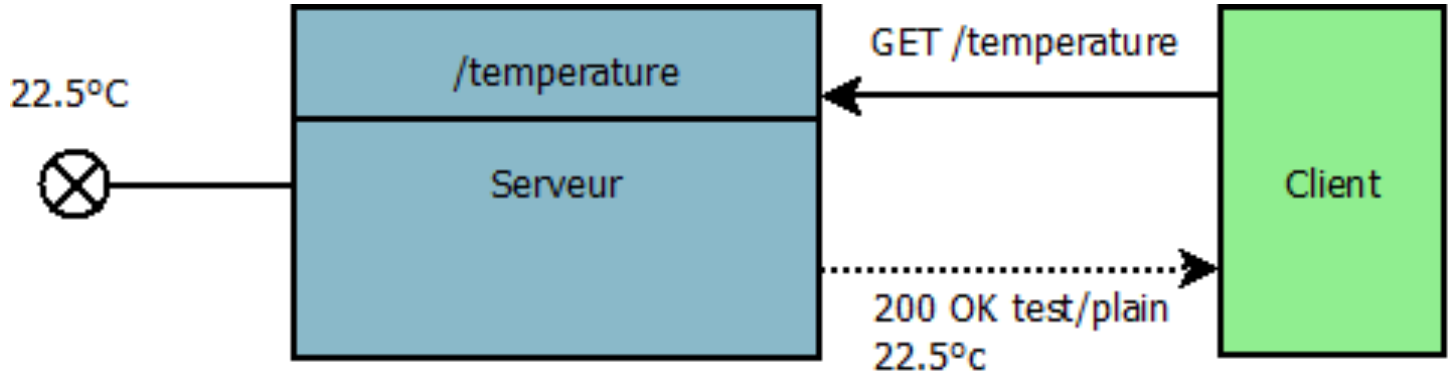
	3G		Wifi	
	HTTPS	MQTT	HTTPS	MQTT
% Battery / Hour	18.79%	**17.80%**	5.44%	**3.66%**
Messages / Hour	1926	**21685**	5229	**23184**
% Battery / Message *	0.00975	**0.00082**	0.00104	**0.00016**

MQTT : Demonstration with Mosquitto and ESP32



CoAP : Constrained Application Protocol

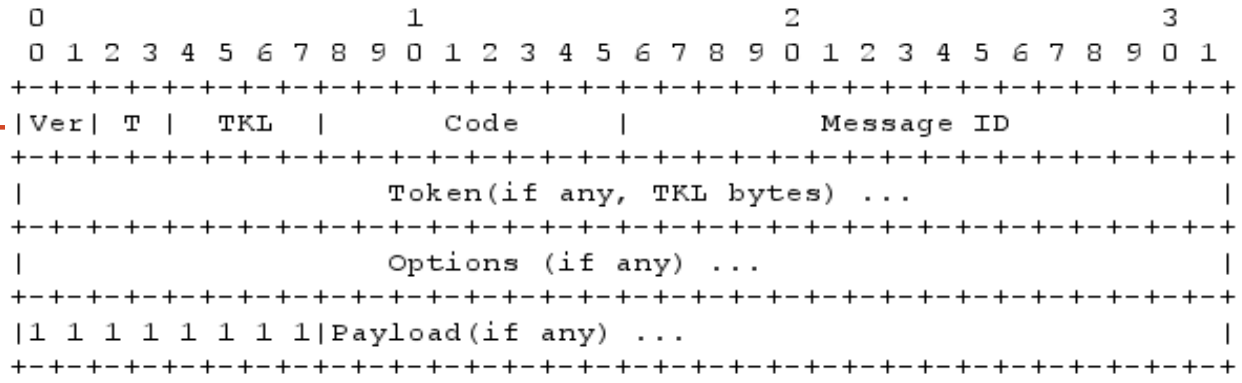
- CoAP protocol designed by the Constrained RESTful Environments (CoRE)
- Based on the optimized REST architectural style for constrained devices and networks used in wireless sensor (IoT) networks.
- Example of a customer querying a sensor to get the ambient temperature:



CoAP vs HTTP

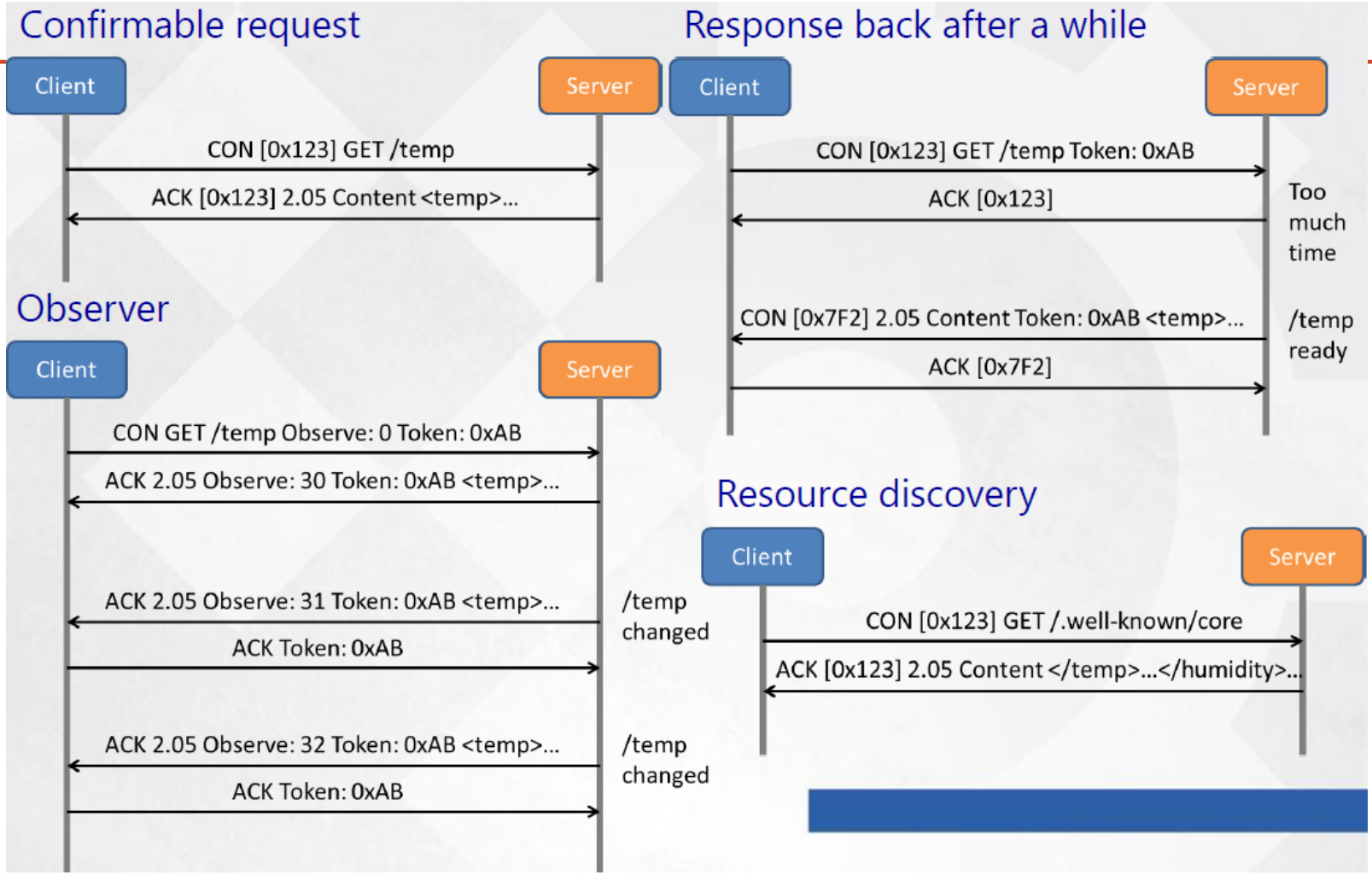
- HTTP-like but based on UDP (no TCP)
- Client/Server (HTTP verbs, status codes HTTP-like)
- "Options" (like HTTP headers) are binary
  - ✓ Client more simple (than HTTP)
- Quality of Service with "confirmable" messages
- Security with DTLS (Datagram TLS)
- Resource discovery

Format du message CoAP



Champ	Description
Ver (Version)	Le champ *Ver* possède 2 bits, indiquant la version de CoAP utilisée.
T (Type)	•***Confirmable*** **(0) :** Le message requiert une réponse •***Non-confirmable*** **(1) :** Le message ne requiert aucune réponse ou acquittement; •***Acknowledgment*** **(2) :** Le message confirme la réception d'un message *Confirmable*. •***Reset*** **(3) :** Dans le cas où le message n'a pu être traité.
TKL (Token Length)	est composé de 4 bits, indiquant la longueur du champ *Token*.
Code	est composé de 8 bits, dont les 3 bits les plus significatifs (c) indiquent la classe et les 5 bits les moins significatifs les détails (dd). Le code au format « c.dd », permet d'indiquer le type de message, « 0 » pour une requête, « 2 » pour une réponse OK , « 4 » pour réponse en erreur client, « 5 » pour une erreur serveur.
Message ID	est composé de 16 bit, utilisés pour détecter la duplication de messages et faire correspondre les messages *acknowledgment/reset* aux messages de type *Confirmable/Non-Confirmable*

CoAP


Confirmable request
Client
Server
CON [0x123] GET /temp
ACK [0x123] 2.05 Content <temp>...
Response back after a while
Client
Server
CON [0x123] GET /temp Token: 0xAB
ACK [0x123]
Too much time
CON [0x7F2] 2.05 Content Token: 0xAB <temp>...
/temp ready
ACK [0x7F2]
Observer
Client
Server
CON GET /temp Observe: 0 Token: 0xAB
ACK 2.05 Observe: 30 Token: 0xAB <temp>...
ACK 2.05 Observe: 31 Token: 0xAB <temp>...
/temp changed
ACK Token: 0xAB
ACK 2.05 Observe: 32 Token: 0xAB <temp>...
/temp changed
ACK Token: 0xAB
Resource discovery
Client
Server
CON [0x123] GET /.well-known/core
ACK [0x123] 2.05 Content </temp>...</humidity>...

CoAP vs MQTT

	CoAP	MQTT
Communications Model	Request-Response, or Publish-Subscribe	Publish-Subscribe
RESTful	Yes	No
Transport Layer Protocol	UDP (TCP can be used)	TCP (UDP can be used; MQTT-SN)
Header	4 bytes	2 bytes
Number of Message Types	4	16
Messaging	Asynchronous & Synchronous	Asynchronous
Scalability	Complex	Simple
Security	DTLS	SSL/TLS
QoS options	Yes (Confirmable/Non confirmable messages)	Yes (3 levels)
Encoding	Binary	Binary
Dynamic discovery	Yes	No