Communication Protocols in IoT

Inernet of Things

UTEC UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS5055 – 2025I
PROF.: CWILLIAMS@UTEC.EDU.PE
SRC: selected journals

Outline

Introduction

Communication protocols: HTTP and websocket

CoAP and AMQP

MQTT

Conclusion

Recall: Communication models

- Client – Server
  - Sends all processing tasks to server according to IoT devices
  - Computational capacity is asymmetric in IoT
- Publish/subscribe
  - Stablishes a central entity called broker, which receives messages from publishers and forwards them to its subscribers

Outline

Introduction

Communication protocols: HTTP and websocket

CoAP and AMQP

MQTT

Conclusion

Communication Protocols

- There is no such protocol!
  - No data protocols is standard for all IoT applications. It can change according to design parameters.
    - HTTP
    - WebSocket
    - AMQP
    - CoAP
    - MQTT
    - XMPP, STOMP, SSI, etc.
  - Some are client/server and some are publish/subscribe

Communication Protocols

- Client/Server category
  - HTTP/HTTPS: Requisition and response. APIs work as an abstraction layer (REST)
  - CoAP: Diverges from HTTP because of its smaller header and the device can be client or server. We can use this to our advantage: Internet is dynamic and CoAP supports dynamics.

Communication Protocols

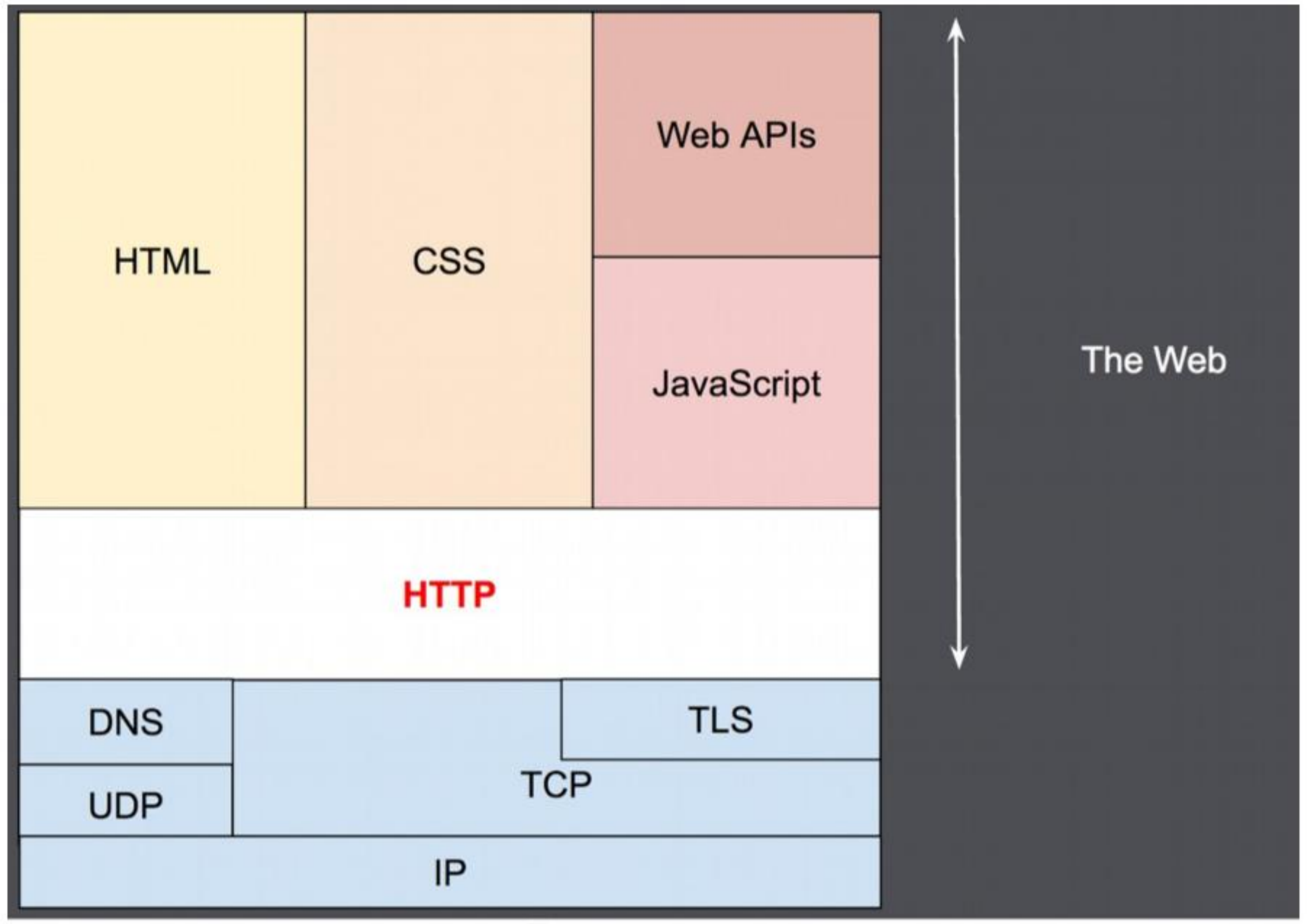
- Publish/Subscribe category
  - Remember last week:

The central entity called Broker and how it forwards Subscribers.

  - Examples: WebSocket, AMQP and MQTT

HTTP

- HTTP is a communication protocol used for hypermedia information systems, distributed and collaborative.
  - It is the base protocol for the world wide web
- HTTP works as a request-response protocol on a client-server paradigm
- The client sends a new HTTP request to the server. It responds with data, such as: HTML data and related content. It can return a response message to the client.



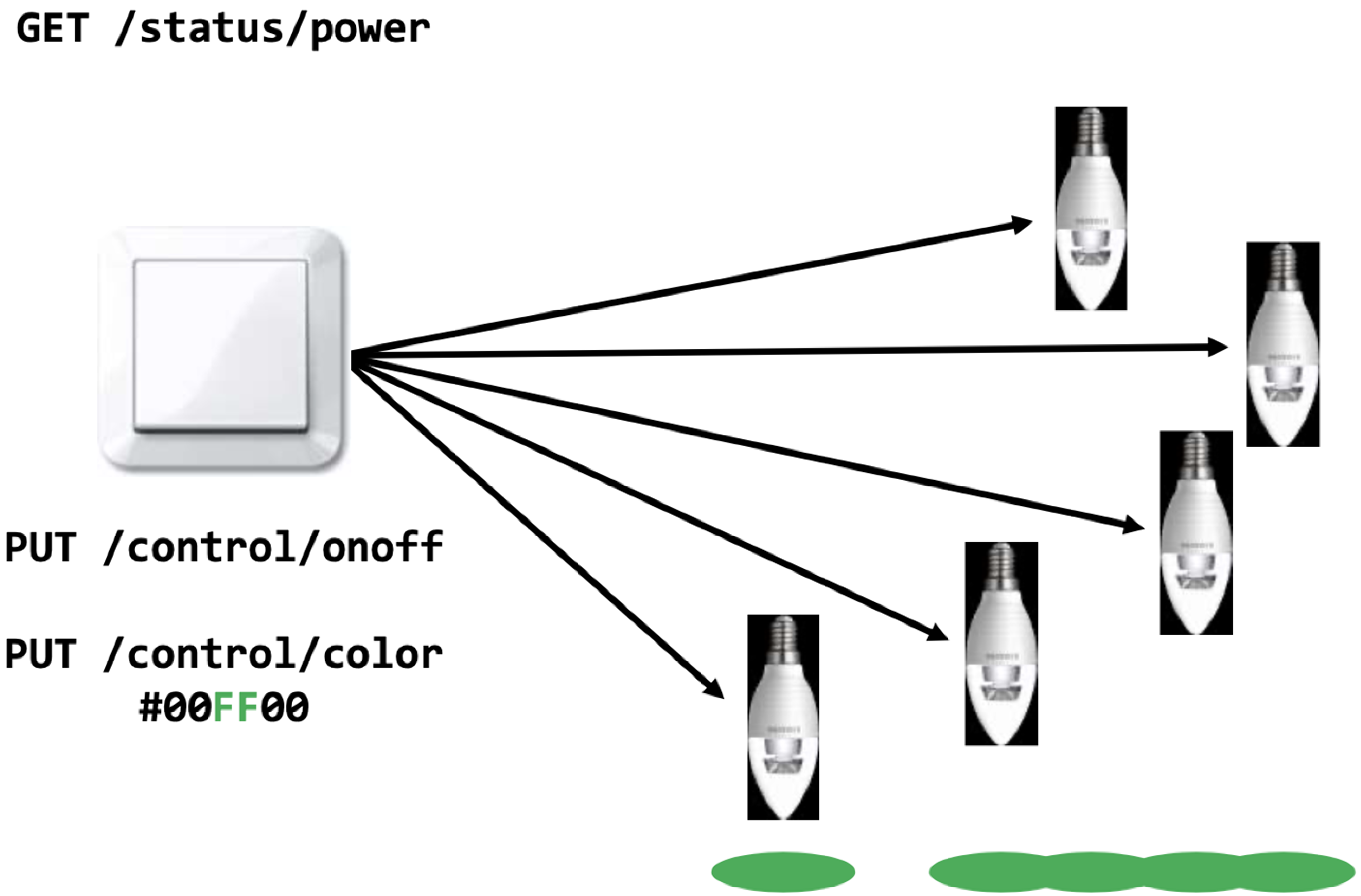
HTTP

- Represent the actions to be performed on resources

- HTTP GET
- HTTP POST - creates a resource
- HTTP PUT - updates a resource
- HTTP DELETE

HTTP: example

- REST for devices control communication



GET /status/power

PUT /control/onoff

PUT /control/color #00FF00

REST

- Widely used:
  - Google Maps - https://developers.google.com/maps/web-services/
    - Try http://maps.googleapis.com/maps/api/geocode/json?address=bangkok
  - Twitter - https://dev.twitter.com/rest/public
  - Facebook - https://developers.facebook.com/docs/atlas-apis
  - Amazon offers several REST services, e.w., for their S2 storage solution

  http://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html
  - The Google Glass API, known as "Mirror API", is a pure REST API.
  - Here is (https://youtu.be/JpWmGX55a40) a video talk about this API. (The actual API discussion starts after 16 minutes or so.)

REST

- Widely used:
  - Look up https://cloud.google.com/apigee

WebSocket

- WeSocket is a protocol which allows the creation of a client-server bidirectional communication channel.
  - HTTP is unidirectional (communication only occurs on a one way basis) where client sends a request and a server answers.
  - WebSocket is a protocol which runs on top of TCP sockets
- When can we use WebSocket?
  - When you need an open connection for a long time in order to trade date constantly
  - A WebSocket connection, after handshake between client and server, stays open until one of them finalizes it.
    - Benefits when compared to HTTP: Low latency, persistent connection and full-duplex (bidirectional transmission)
    - Examples: Social Network feeds, chats, collaborative editing tools, multi-player games

Outline

Introduction

Communication protocols: HTTP and websocket

CoAP and AMQP

MQTT

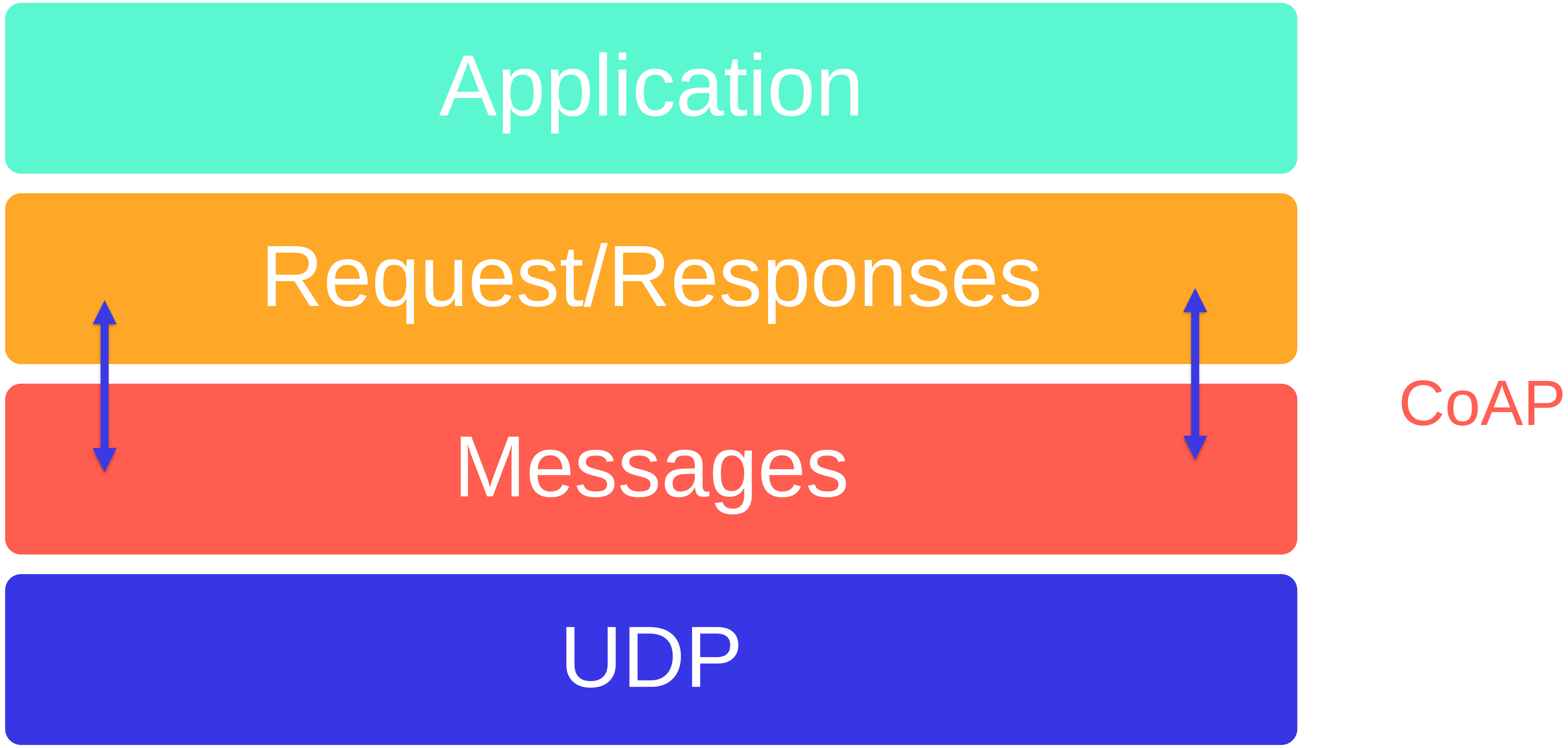
Conclusion

CoAP

- It can be used in devices with important restrictions:
  - Low computational power
  - A system which cannot use Computer Networks-compatible protocols
  - CoAP – Constrained Application Protocol was developed to accommodate these restrictions
- CoAP is based on a REST architecture (Representational State Transfer)
- CoAP uses UDP as a transport protocol and exchanges small messages with small overhead, which is the best for low-memory devices and limited computational power.
- CoAP is used in industrial applications, residential automation, Machine-to-Machine (M2M) managing and satellite communications (small bandwidth).

CoAP

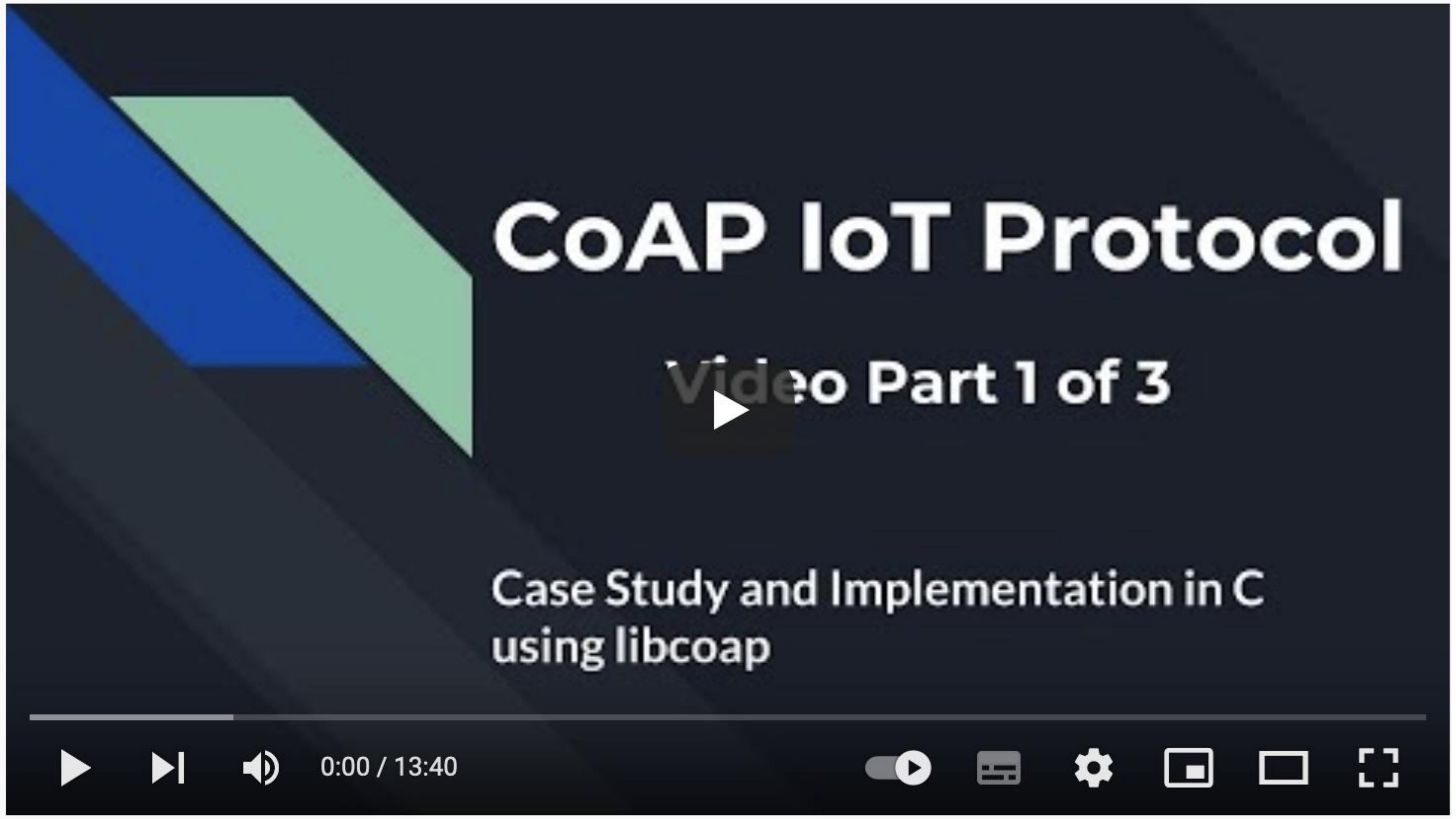
- Interaction between devices is like the client-server model; however, both end devices can be client or server at the same time.



CoAP - Video

https://www.youtube.com/watch?v=Bd3BRv4hO-4

Three-part video



CoAP IoT Protocol

Video Part 1 of 3

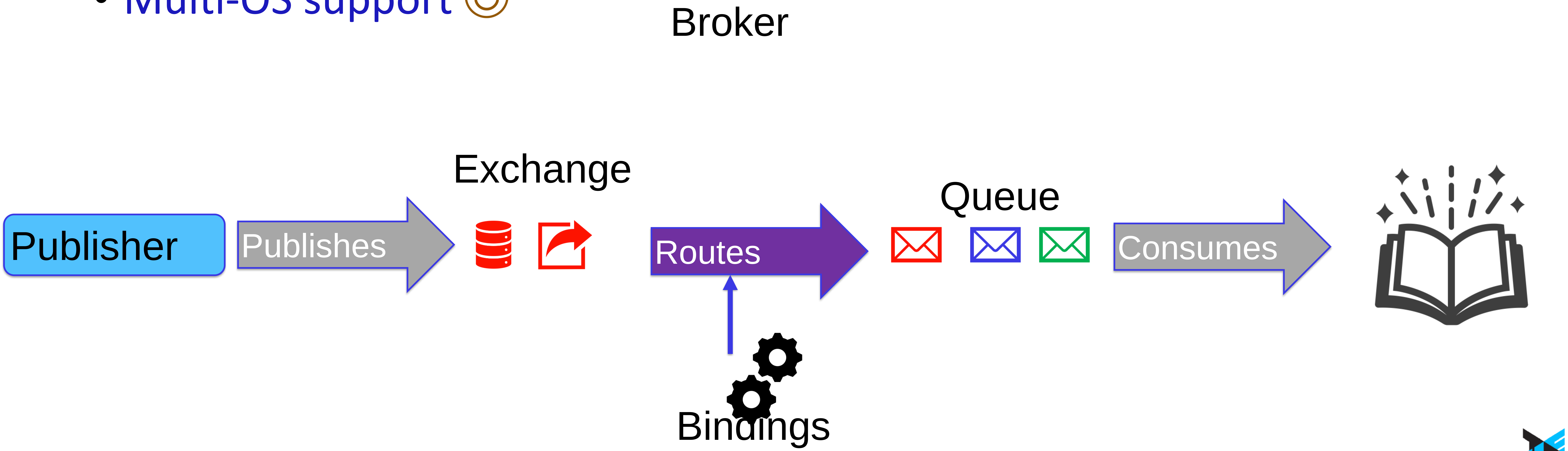
Case Study and Implementation in C using libcoap

AMQP

- AMQP (Advanced Message Queuing Protocol) is a protocol which uses TCP as transport layer.
- Allows asynchronous message sending and receiving.
- Can be understood as the asynchronous equivalent of HTTP, where a client is able to contact a broker “midway”.

AMQP

- AMQP was originally developed by the finance community
  - Its goal is to allow interoperability between devices
  - RabbitMQ is broker-like application for AMQP. Rabbit MQ allow asynchronous communication.
  - Multi-OS support ☺



Outline

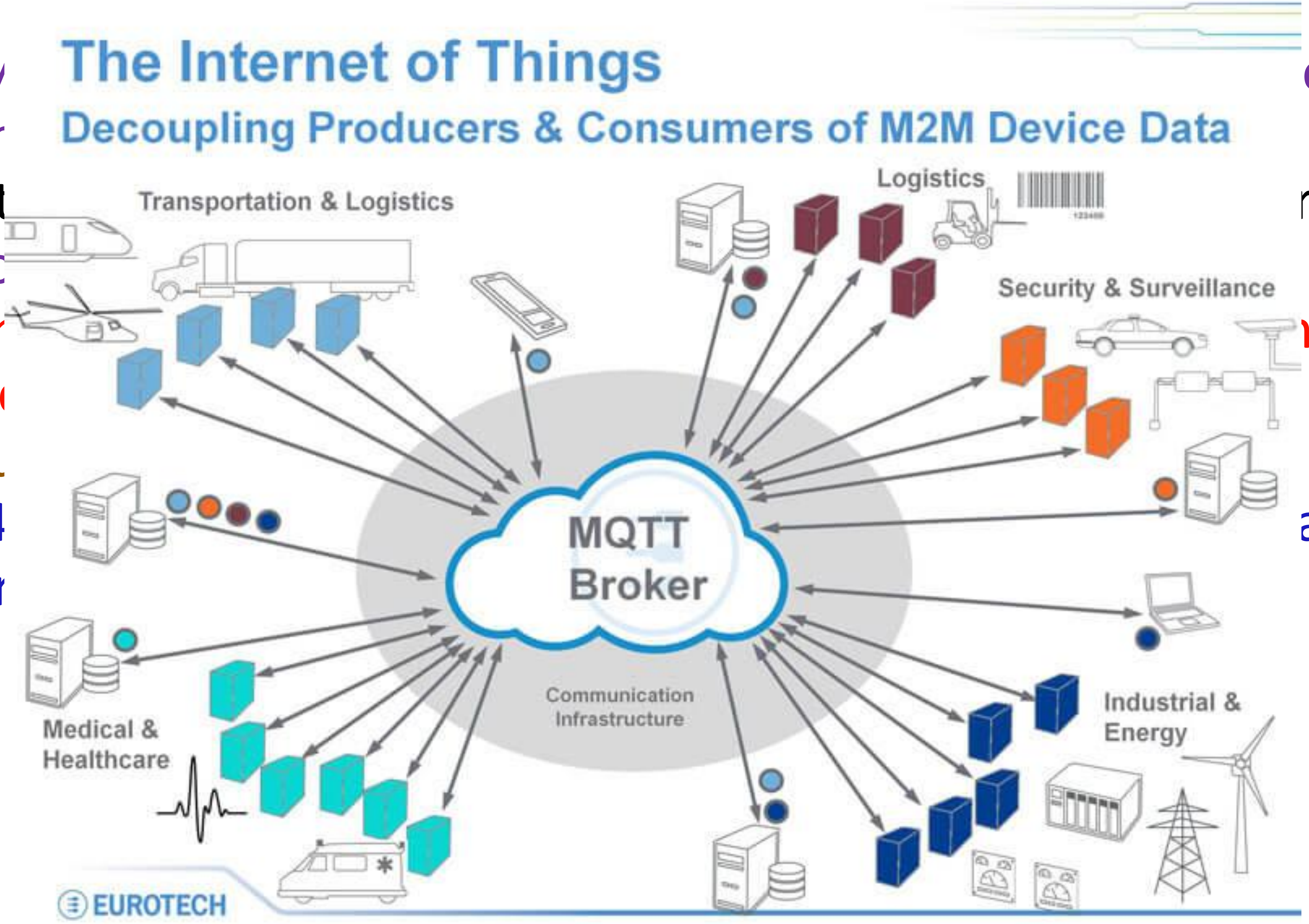
- Introduction
- Communication protocols: HTTP and websocket
- CoAP and AMQP
- MQTT
- Conclusion

MQTT

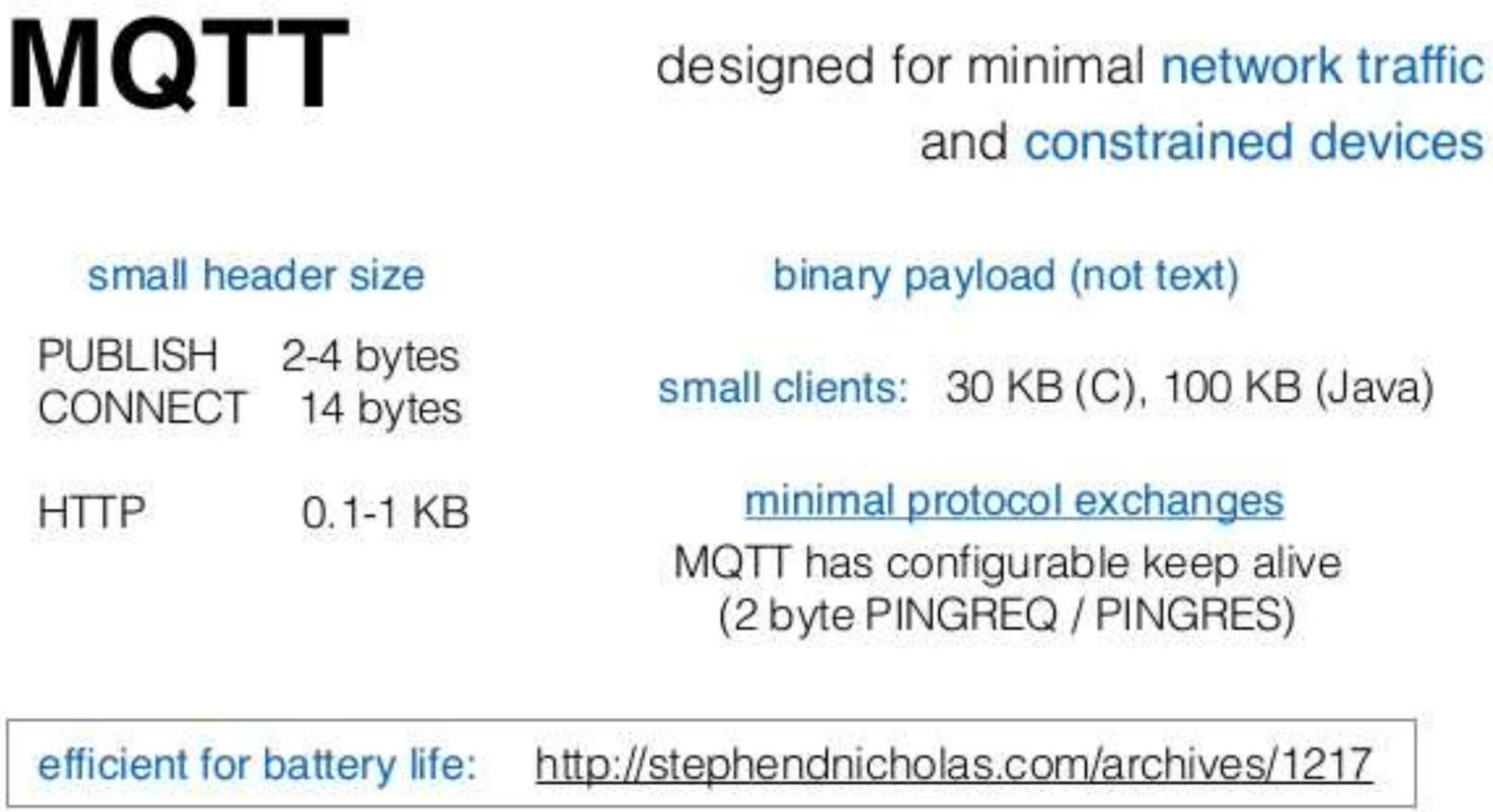
- MQTT (Message Queuing Telemetry Transport Protocol) was developed by IBM in 99.
  - Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link)
- Asynchronous communication protocol
  - Time and space de-coupling of emitter/transmitter making the network architecture scalable.
  - Uses publish-subscribe model
  - In 2014 was declared open by OASIS. To this day there are many free software implementations.

MQTT

- MQTT (M eveloped by IBM ir
  - Andy St rus Link)
- Asynchro
  - Time ar e network archite
  - Uses pu
  - In 2014 any free softwar



MQTT



MQTT

designed for minimal network traffic
and constrained devices

small header size

PUBLISH	2-4 bytes
CONNECT	14 bytes
HTTP	0.1-1 KB

binary payload (not text)

small clients: 30 KB (C), 100 KB (Java)

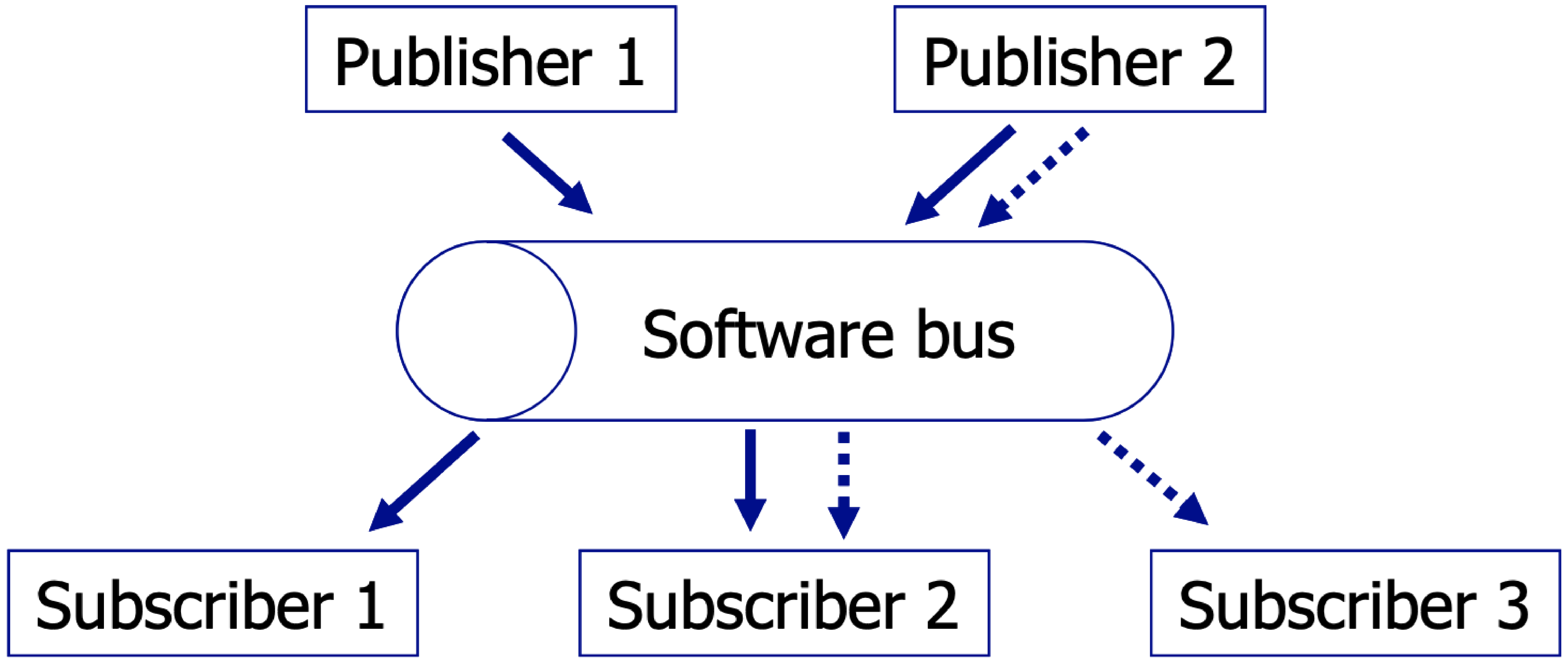
minimal protocol exchanges

MQTT has configurable keep alive
(2 byte PINGREQ / PINGRES)

efficient for battery life: http://stephendnicholas.com/archives/1217

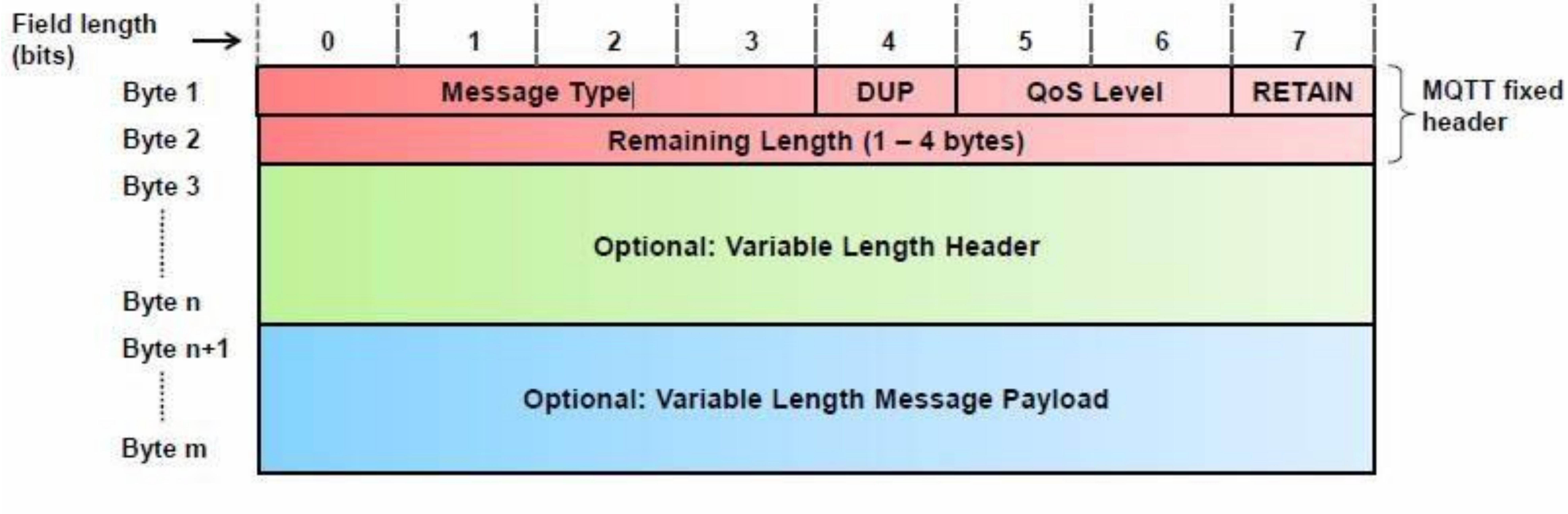
MQTT

- MQTT is light and allow network communications with low transmission capacity hardware.
- Supports many scenarios and applications for mobile and IoT services.
- MQTT defines two types of entities on a network: a broker and a client.
  - Broker: Receives messages from clients and forwards them to clients (routing)
  - Client: Any “thing” which interacts with a broker and can receive messages. It can be a sensor, or an application on a data center which processes IoT data



MQTT

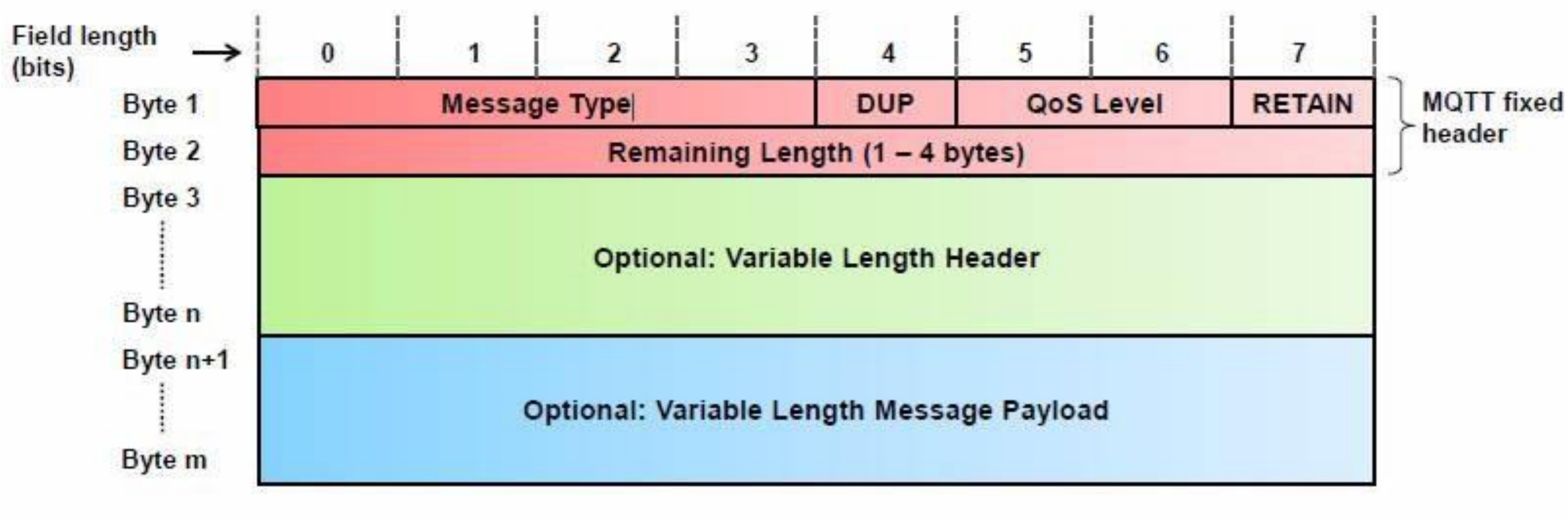
- MQTT is data agnostic!
- MQTT uses TCP How to use TCP?
  - It is not necessary to know about TCP implementation, although it can be helpful
  - Understanding header-payload paradigm is enough
  - MQTT Header:



Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 ⋮ Byte n	Optional: Variable Length Header								
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload								

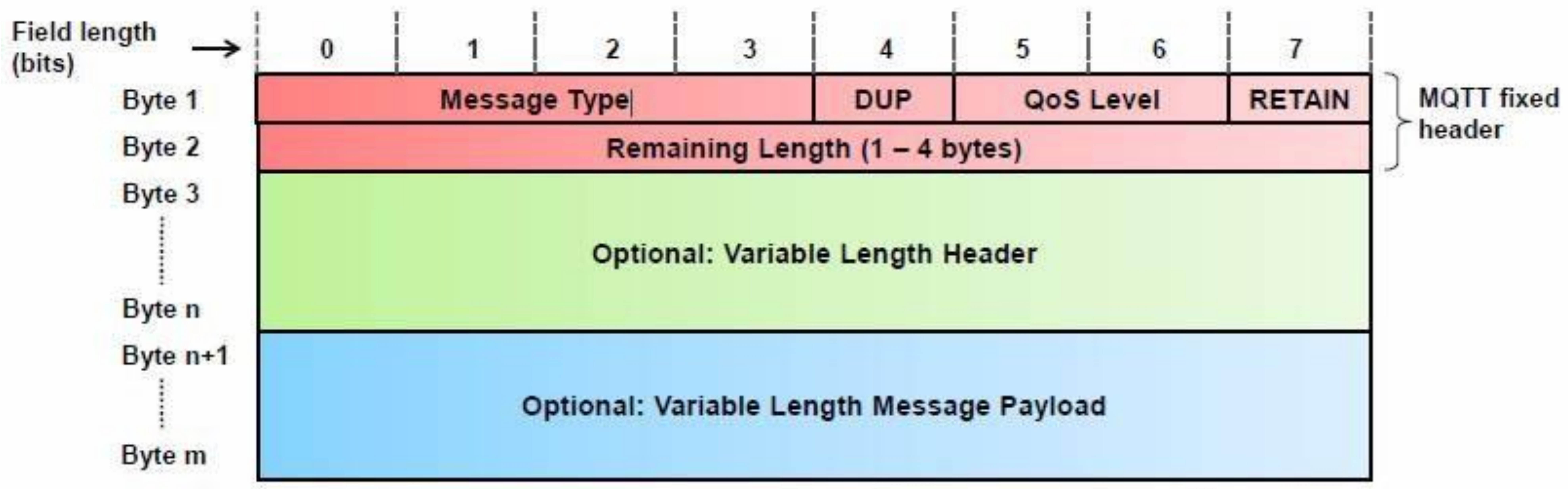
MQTT

- MQTT uses TCP How to use TCP?
  - It is not necessary to know about TCP implementation, although it can be helpful
  - Understanding header-payload paradigm is enough
  - MQTT Header:



MQTT

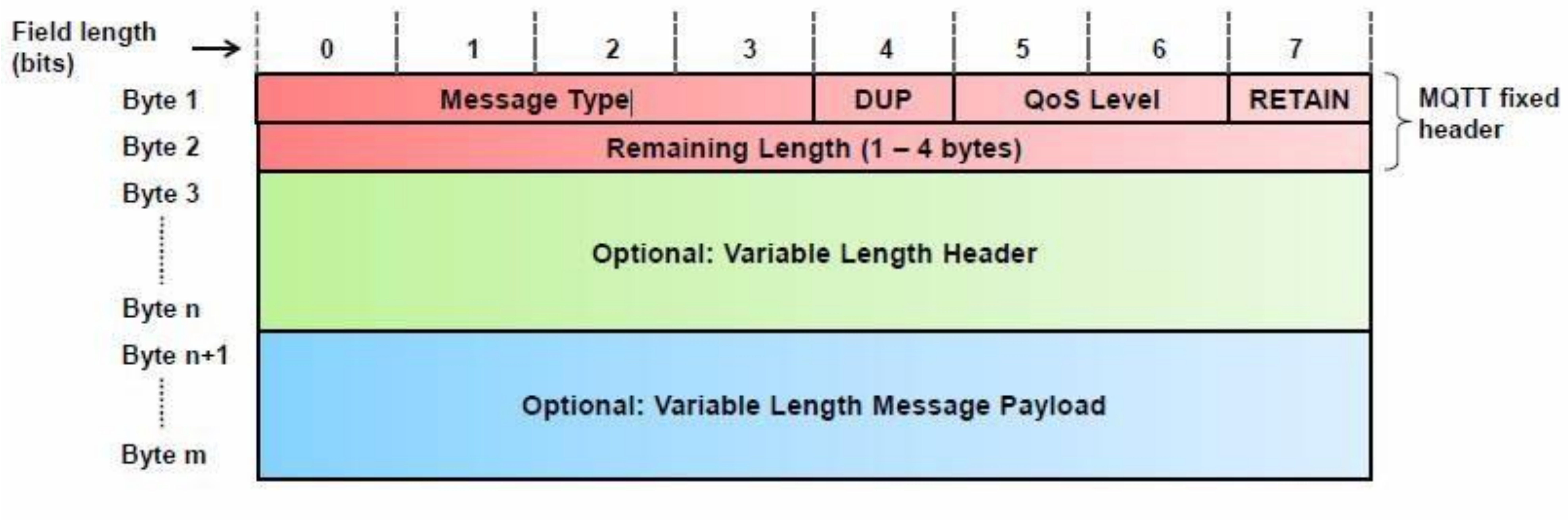
- MQTT header:
  - 2 to 5 bytes long
  - First byte is mandatory: the first 4 bits indicate the type of message, the next bit is the duplicated message flag.



Field length (bits) →	0	1	2	3	4	5	6	7	
Byte 1	Message Type				DUP	QoS Level		RETAIN	MQTT fixed header
Byte 2	Remaining Length (1 – 4 bytes)								
Byte 3 ⋮ Byte n	Optional: Variable Length Header								
Byte n+1 ⋮ Byte m	Optional: Variable Length Message Payload								

MQTT

- MQTT header:
  - Two more bits to indicate QoS of the package and a bit to indicate retention – when somebody wants to connect and receive the last message sent.
  - The next 4 bytes define the size of the rest of the package.
  - Lastly, data which is not standardized.



MQTT

- 3 Types of messages
  - CONNECT: Creates a connection with the broker and waits for the stablished connection
  - DISCONNECT: Waits for the client to finish an action and for the TCP/IP connection to be finalized – it stops listening
  - PUBLISH: Returns the data which was sent by the MQTT client
- Quality of Service (QoS) for interchange of messages
  - There are 3 QoS and all connections to broker can specify which one is to be used.
    - <1 QoS 0
    - >1 QoS 1
    - =1 QoS 2

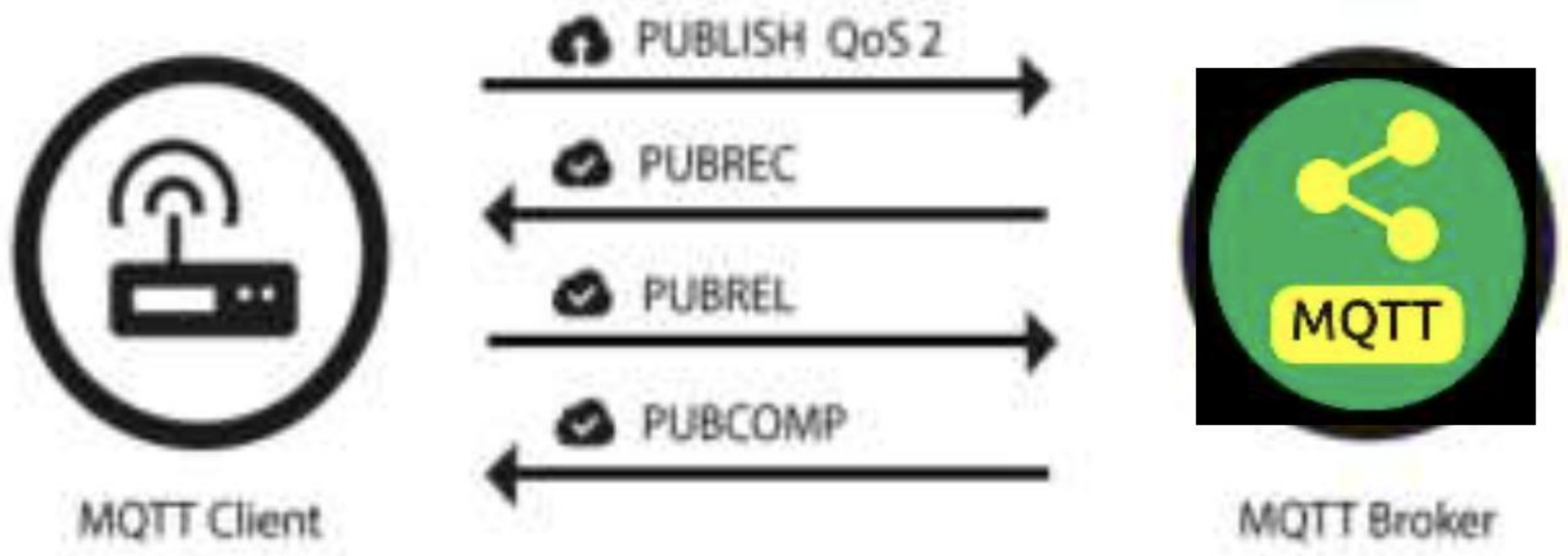
MQTT

- QoS 0: Fire/forgot –r “at most once”
  - The message is sent one time and then forgotten
  - The message is not stored and it does not have feedback to know if it arrived to its destination.
  - It´s the fastest transfer mode
  - If the transmission fails or if the client disconnects, the message is lost completely.

- QoS 1: “at least once”
  - Message delivered at least 1 time with feedback waiting (PUBBACK – acknowledgement)
  - If PUBBACK is not received, it sends the message continuously until feedback PUBBACK changes.

MQTT

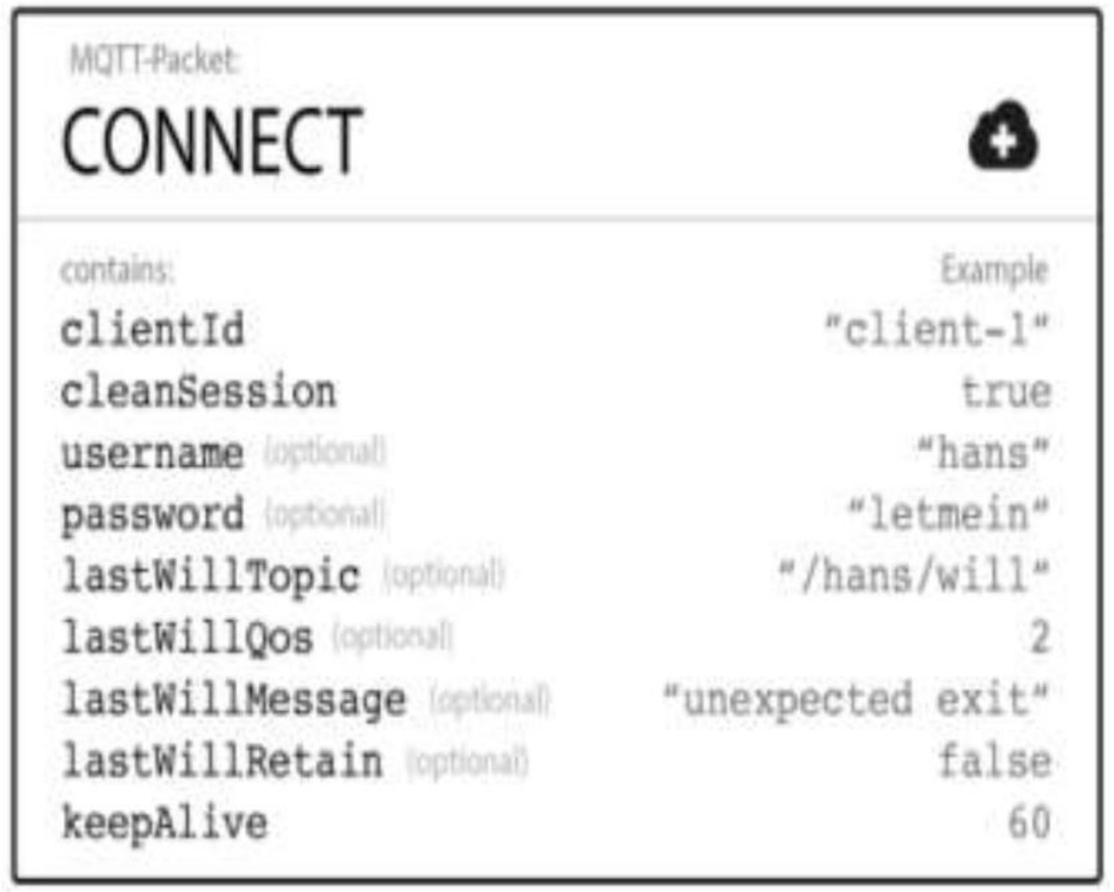
- QoS 2: “exactly once”
  - Message is delivered exactly one time.
  - Needs local storage on the emitter and receiver until it is processed.
  - To guarantee this scenario we need a four-part handshake, which are two sets of request-response communications.
    - Message is sent (PUBLISH)
    - Response to receive (PUBREL) and confirmation that the process was concluded and data can be deleted (PUBCOMP)
    - After receiving PUBREL, the receiver can delete the message and when the emitter receives PUBCOMP it can delete de message as well.



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

MQTT

- The Will message
  - When clients connect, they can specify an optional “will” message, to be delivered if they are unexpectedly disconnected from the network.
    - (In the absence of other activity, a 2-byte ping message is sent to clients at a configurable interval.)
  - This “last will and testament” can be used to notify other parts of the system that a node has gone down.



MQTT-Packet

CONNECT

contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

MQTT

- Software available
  - Brokers (https://github.com/mqtt/mqtt.github.io/wiki/servers):
    http://mosquitto.org/
    http://www.hivemq.com/
    https://www.rabbitmq.com/mqtt.html
    http://activemq.apache.org/mqtt.html
    https://github.com/mcollina/mosca

- Clients
  http://www.eclipse.org/paho/

- Source: https://github.com/eclipse/paho.mqtt.python
  http://www.hivemq.com/demos/websocket-client/

- Tools
  https://github.com/mqtt/mqtt.github.io/wiki/tools

Outline

Introduction

Communication protocols: HTTP and websocket

CoAP and AMQP

MQTT

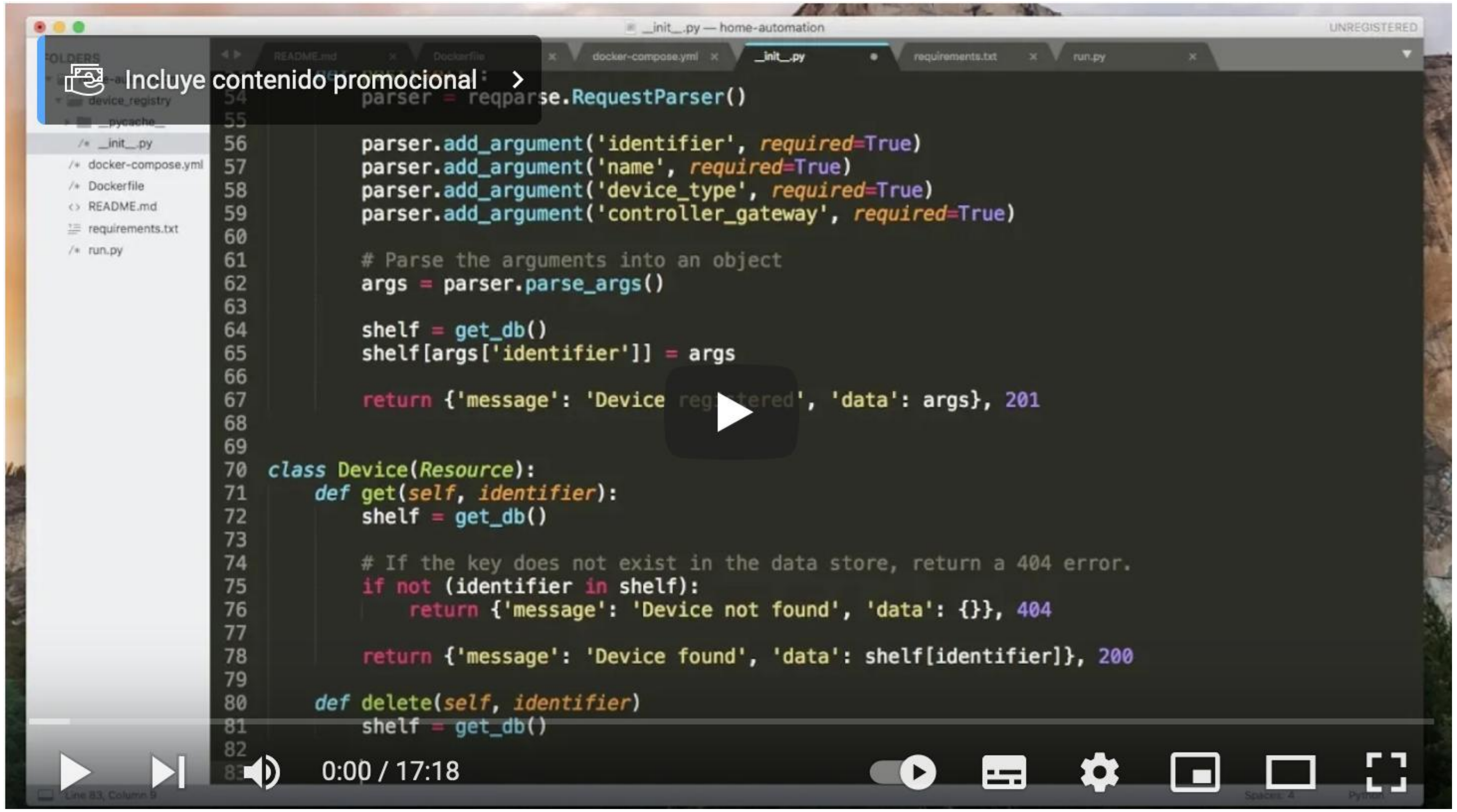
Conclusion

Conclusion

- There are many ways to communicate our IoT sensors/actuators
- Protocols are categorized in two groups
- Computational power of the “things” is an important feature for choosing communication protocols
- MQTT is important because:
  - Low memory consumption
  - Low processing requirements
  - Low bandwidth consumption

Activity

- REST API in Python
- https://www.youtube.com/watch?v=4T5Gnrmzjak



```python
        parser = reqparse.RequestParser()

        parser.add_argument('identifier', required=True)
        parser.add_argument('name', required=True)
        parser.add_argument('device_type', required=True)
        parser.add_argument('controller_gateway', required=True)

        # Parse the arguments into an object
        args = parser.parse_args()

        shelf = get_db()
        shelf[args['identifier']] = args

        return {'message': 'Device registered', 'data': args}, 201


class Device(Resource):
    def get(self, identifier):
        shelf = get_db()

        # If the key does not exist in the data store, return a 404 error.
        if not (identifier in shelf):
            return {'message': 'Device not found', 'data': {}}, 404

        return {'message': 'Device found', 'data': shelf[identifier]}, 200

    def delete(self, identifier)
        shelf = get_db()
```