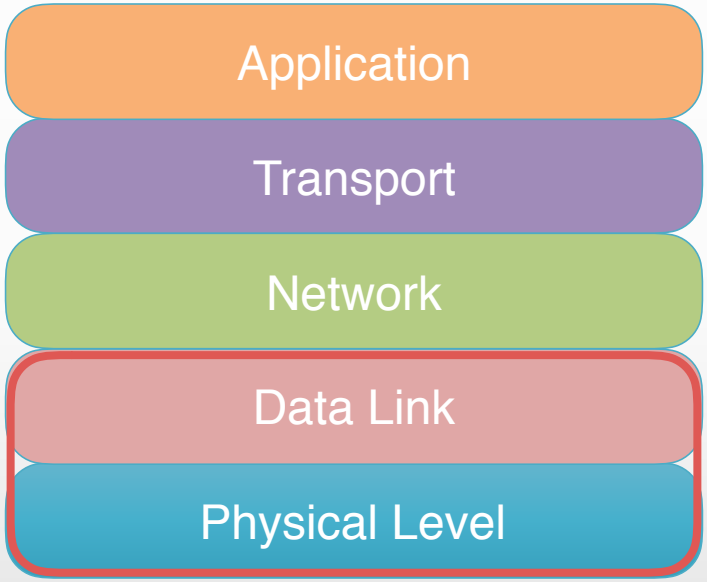
Design and development of embedded systems for the Internet of Things (IoT)

Fabio Angeletti
Fabrizio Gattuso

SAPIENZA
UNIVERSITÀ DI ROMA

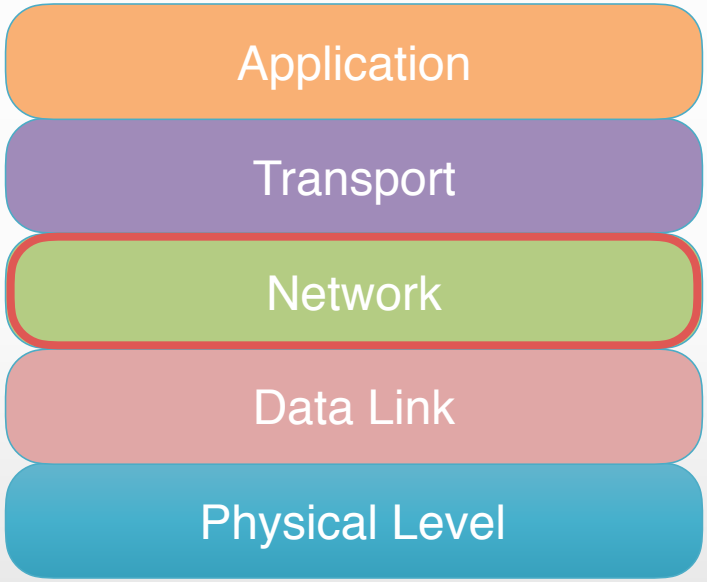
W • SENSE
INTEGRATED CABLELESS SOLUTIONS

Network stack



802.15.4
Bluetooth
Lora
Sigfox

Network level



6LoWPAN

6LowPAN

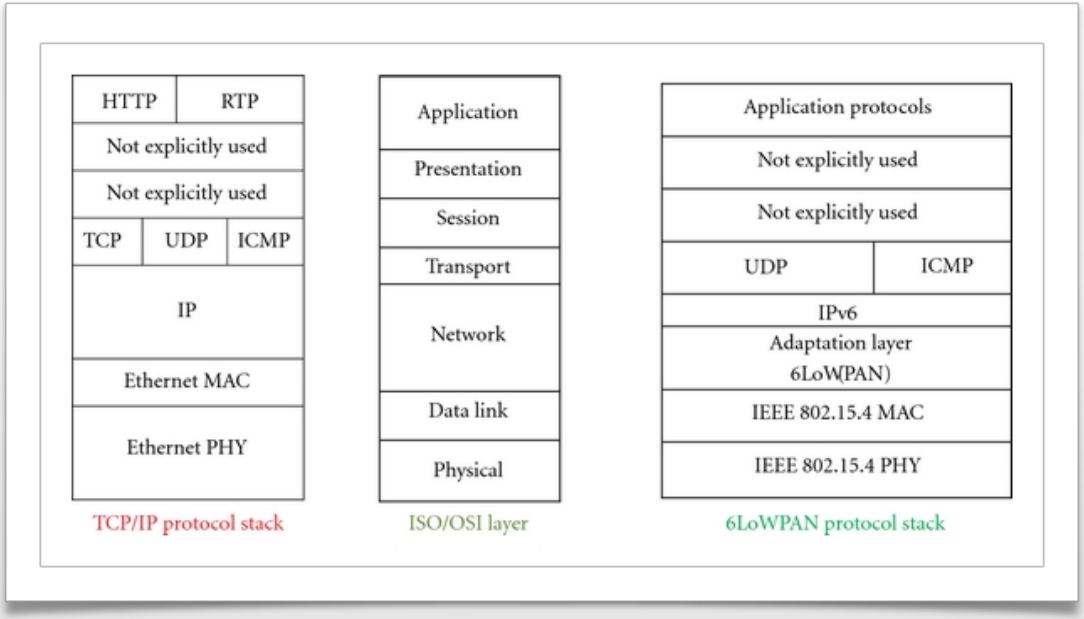
IPv6 over Low power WPAN (6LoWPAN) is an adaptation layer that allows to route Internet traffic over WSNs.

The idea is to move the internet to the embedded systems adapting the existing protocols (IP) to the new requirements.

The adaptation layer is builded on top of the 802.15.4 standard (not ZigBee).

6LoWPAN - Adaptation layer

The adaptation layer sits between the Data-Link and the Network Layer.



We want to address every nodes with an IP

6LowPAN - Adaptation layer (2)

The main problem is the size of the packet.

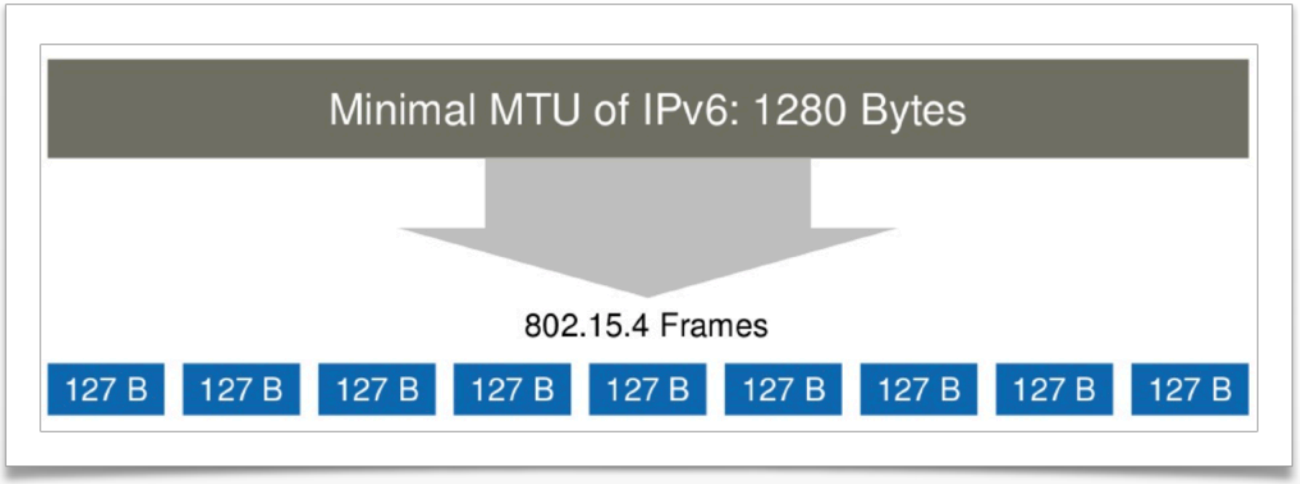
802.15.4 limits the dimension of the payload to **126 bytes** and the minimum packet size of **IPV6 is 1280 bytes**.

The adaptation layer use two technique to solve the problem.

- **Header compression:**

  Redundant information of IPV6 header is removed
- **Fragmentation**

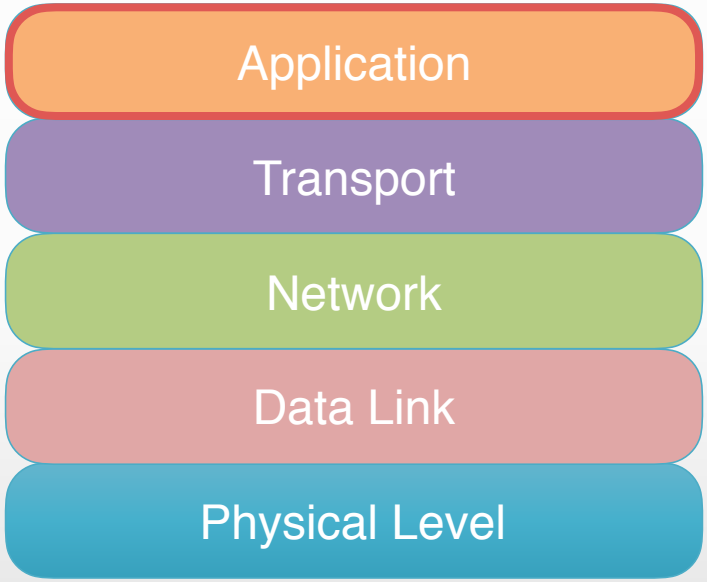
6LowPAN - Fragmentation



In order to handle the fragmentation there are special informations:

- **Datagram size:** size of the ip fragment
- **Datagram tag:** id of the ip fragment
- **Datagram offset:** the offset from the beginning of the ip fragment

Application level



CoAP
MQTT
REST

MQTT

MQTT is a Client Server **publish/subscribe** messaging protocol.

It is designed to be:

- **Light weight**
- **Open**
- **Simple and easy to implement**

It is the ideal protocol to communicate between Machine to Machine (**M2M**) and for the Internet of Things (**IoT**).

MQTT (2)

MQTT (MQ Telemetry Transport but not used anymore) was designed by two computer scientist from **IBM** and **Arcom** back in the **1999**.

They wanted to creare a protocol for **minimal battery loss** and **minimal bandwidth** connecting oil pipelines **over satellite connection**.

Now the focus is changed to the embedded systems and the Internet of Things (IoT) but as you can imagine the requirements are similar to the original focus.

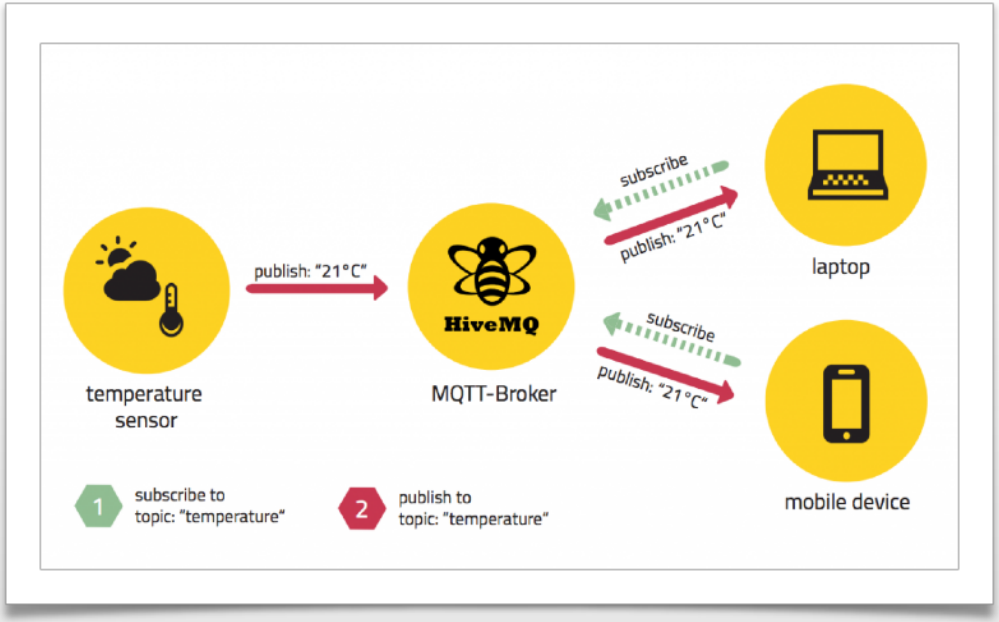
MQTT - Publish and subscribe

The **publish/subscribe pattern** (pub/sub) is an alternative to the traditional client-server model, where a client communicates directly with an endpoint.

There are multiple clients sending a particular message (called **publisher**) and other clients (sometimes the same ones) receiving messages (called **subscriber**).

This means that the **publisher and subscriber don't know about the existence of one other**.

MQTT - Pub and sub (2)



It provides a **greater scalability** than the traditional client-server approach.

This is because **operations on the broker can be highly parallelized and processed event-driven.**

MQTT - Quality of service (QoS)

QoS 0 – at most once

The minimal level is zero and it guarantees a best effort delivery. **A message won't be acknowledged** by the receiver or stored and redelivered by the sender. This is often called "fire and forget" and **provides the same guarantee as the underlying TCP protocol**.

MQTT - QoS (2)

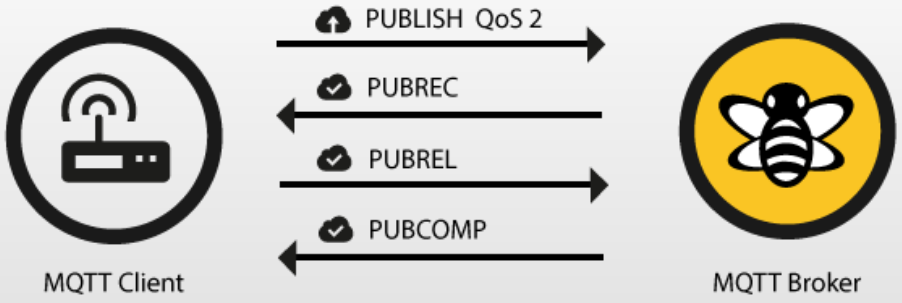
QoS 1 – at least once

When using QoS level 1, **it is guaranteed that a message will be delivered at least once to the receiver.** But t**he message can also be delivered more than once**.

MQTT - QoS (3)

QoS 2

The highest QoS is 2, **it guarantees that each message is received only once by the counterpart**. It is **the safest** and also **the slowest quality of service level**. The guarantee is provided by two flows there and back between sender and receiver.



MQTT - QoS (4)

Use QoS 0 when:

- You have **a complete or almost stable (wired) connection** between sender and receiver.
- You **don't care if one or more messages are lost once a while**. That is sometimes the case if the data is not that important or will be send at short intervals.

Use QoS 1 when:

- **You need to get every message and your use case can handle duplicates**. The most often used QoS is level 1, because it guarantees the message arrives at least once.

Use QoS 2 when:

- **It is critical to your application to receive all messages exactly once.** This is often the case if a duplicate delivery would do harm to application users or subscribing clients. **You should be aware of the overhead and that it takes a bit longer to complete the QoS 2 flow.**

MQTT - Other informations

Retained messages

A retained message is **a normal MQTT message with the retained flag set to true**. **The broker will store the last retained message and the corresponding QoS for that topic each client that subscribes to a topic pattern will receive the message immediately after subscribing**. For each topic only one retained message will be stored by the broker.

Keep Alive

The keep alive functionality assures that the connection is still open and both broker and client are connected to one another. Therefore the client specifies a time interval in seconds and communicates it to the broker during the establishment of the connection. **The interval is the longest possible period of time, which broker and client can endure without sending a message.**

MQTT - Other informations (2)

MQTT is going to be a standard of IoT communications and it also used by **Facebook Messenger**:

"One of the problems we experienced was **long latency** when sending a message. The method we were using to send was reliable but slow, and there were limitations on how much we could improve it. With just a few weeks until launch, we ended up building a new mechanism that maintains a persistent connection to our servers. To do this without killing battery life, we used a protocol called MQTT that we had experimented with in Beluga. MQTT is specifically designed for applications like sending telemetry data to and from space probes, so it is designed to use bandwidth and batteries sparingly. **By maintaining an MQTT connection and routing messages through our chat pipeline, we were able to often achieve phone-to-phone delivery in the hundreds of milliseconds, rather than multiple seconds**."

https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920

MQTT - Other informations (2)

We will se in the next lessons how to use MQTT to connect an IoT system to **Thingsboard**.

ThingsBoard

REST

REST or **RESTful API** (Representational State Transfer) is designed to take advantage of existing protocols by a phd student in the 2000. REST can be used over nearly any protocol, **it usually takes advantage of HTTP when used for Web APIs**.

This means that **developers do not need to install libraries or additional software** in order to take advantage of a REST.

REST is not constrained to XML as SOAP, but instead **can return XML, JSON, YAML or any other format** depending on what the client needs.

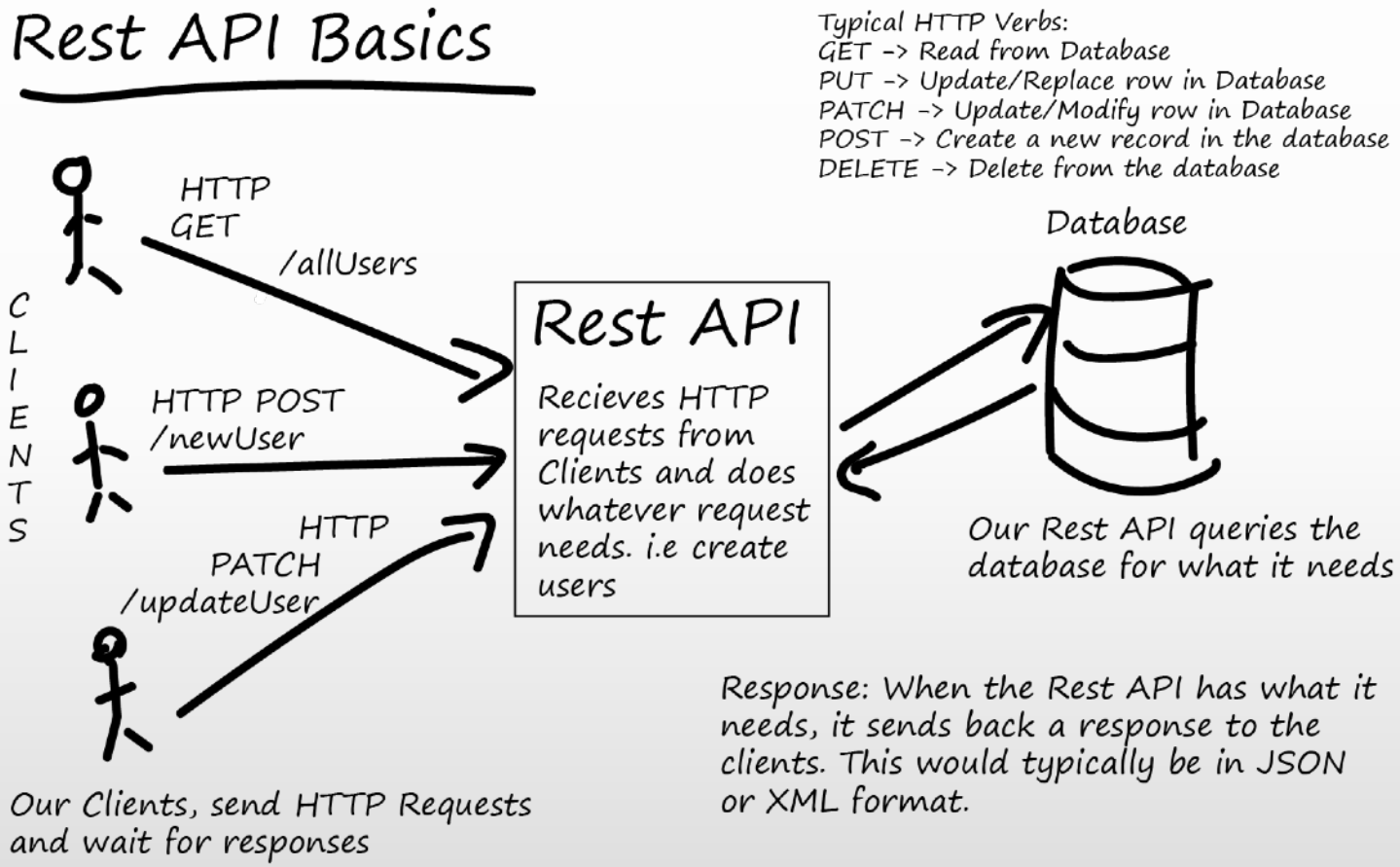
REST (2)

Rest is designed to meet some requirements:

- Client-Server
- Stateless
- Cacheable
- Uniform interface
- Layered system
- Code on demand (RPC)

Also MQTT support RPC.

REST (3)


Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
HTTP GET
/allUsers
CLIENTS
HTTP POST
/newUser
HTTP PATCH
/updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Database
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses

REST (4)

It is designed for the web and work very well on top of HTTP.

Can be used also for the IoT solutions in communication with a web-tool such as **Grafana** that we will see on the next lessons.

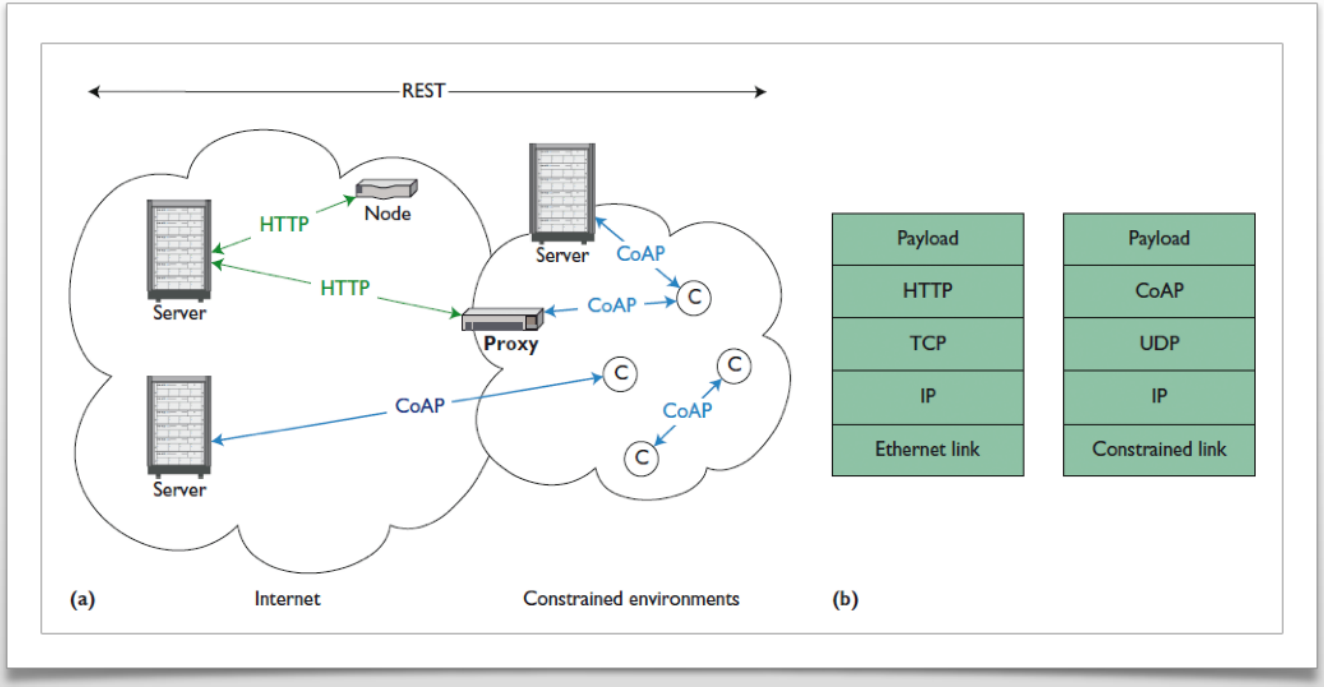
CoAP

Constraint Application Protocol (CoAP) is a specialized web transfer protocol for use with **constrained nodes** and **constrained networks** of Internet of Things and Machine to Machine (M2M) applications.

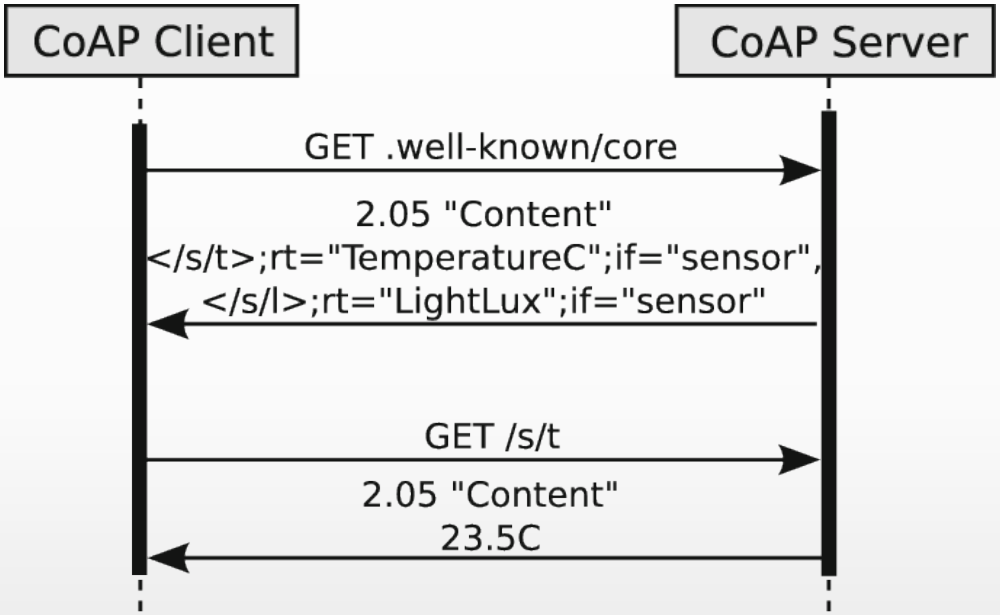
- **UDP** based: reliable support, unicast and multicast requests
- **Low header overhead** and parsing complexity
- URI and Content-Type support
- Support of proxy and caching capabilities
- Stateless
- **Security delegated to DTLS**

CoAP (2)

CoAP is able to handle network nodes as a REST API system. All the external requests are managed by the CoAP proxy.

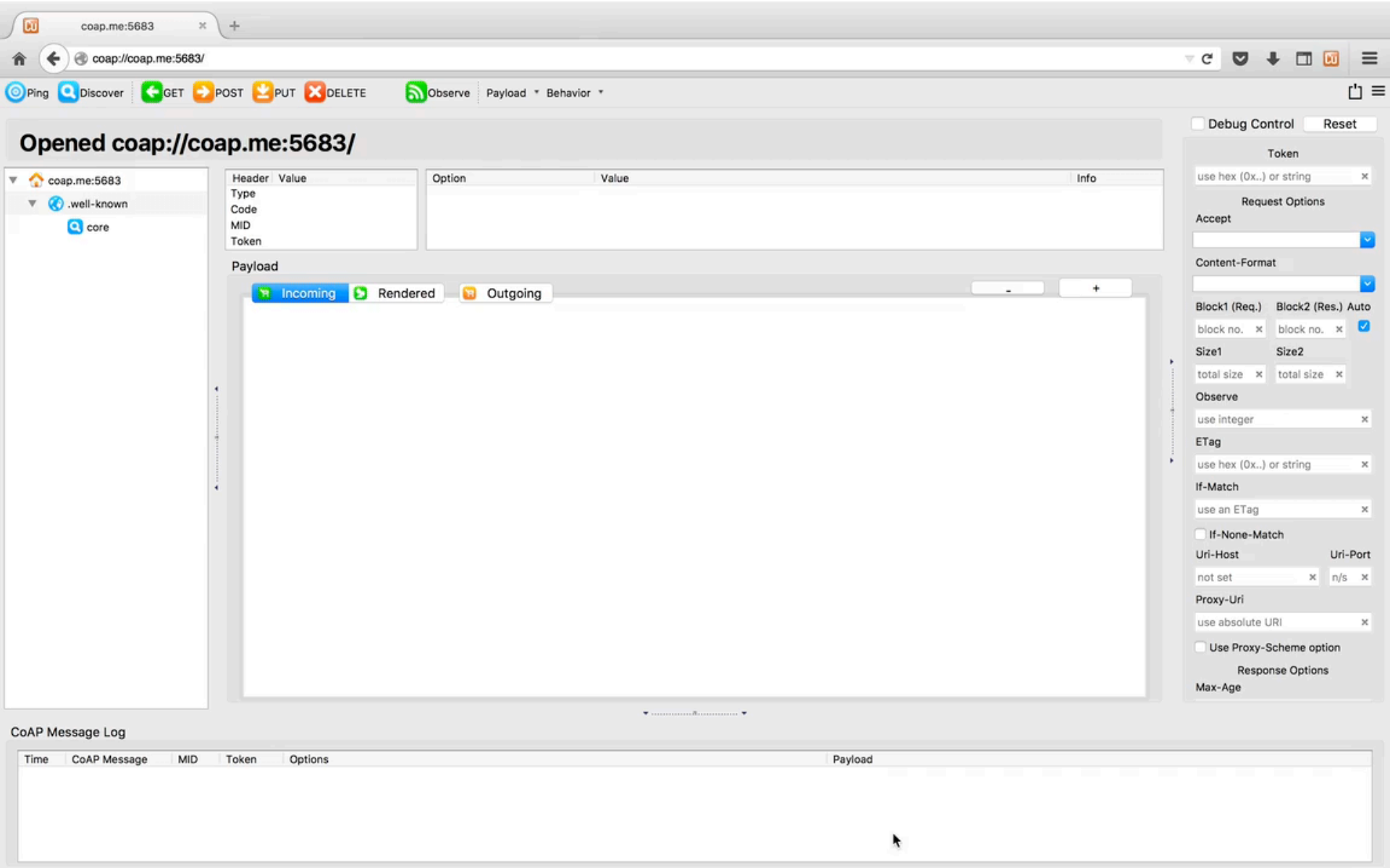


CoAP (3)



The communication is like REST!

CoAP is HTTP compliant.


coap.me:5683
coap://coap.me:5683/
Ping
Discover
GET
POST
PUT
DELETE
Observe
Payload
Behavior
Opened coap://coap.me:5683/
coap.me:5683
.well-known
core
Header
Value
Type
Code
MID
Token
Option
Value
Info
Payload
Incoming
Rendered
Outgoing
Debug Control
Reset
Token
use hex (0x..) or string
Request Options
Accept
Content-Format
Block1 (Req.)
Block2 (Res.)
Auto
block no.
Size1
Size2
total size
Observe
use integer
ETag
use hex (0x..) or string
If-Match
use an ETag
If-None-Match
Uri-Host
Uri-Port
not set
n/s
Proxy-Uri
use absolute URI
Use Proxy-Scheme option
Response Options
Max-Age
CoAP Message Log
Time
CoAP Message
MID
Token
Options
Payload

FINAL PROJECTS

GROUP	TOPIC
ANDREA MONTERUBBIANO	ACCELEROMETER, GYROSCOPE AND OTHER SENSORS TO HANDLE LOCALIZATION
VERONICA BIRINDELLI	A QRC READER (?)
EMILIANO LUCI ARTEM AGEEV	WIRELESS KETTLE AND HTCPCP-TEA PROTOCOL
ROBERTO BRUZZESE	SOMETHING ABOUT AGRICOLTURE
GIORGIO MARIANI MICHELE LAURENTI	WIRELESS DRUM
DOMENICO SILVESTRI ANDREA COLETTA	AN IOT MICROCLIMATE SOLUTION
MATTEO URISELLI	ACCELEROMETER, GYROSCOPE AND MAGNETOMETER EVALUATION

Next lesson

In the next lesson we will talk about:

- **IoT Security** (Gabriele Saturni)
- **Low power**