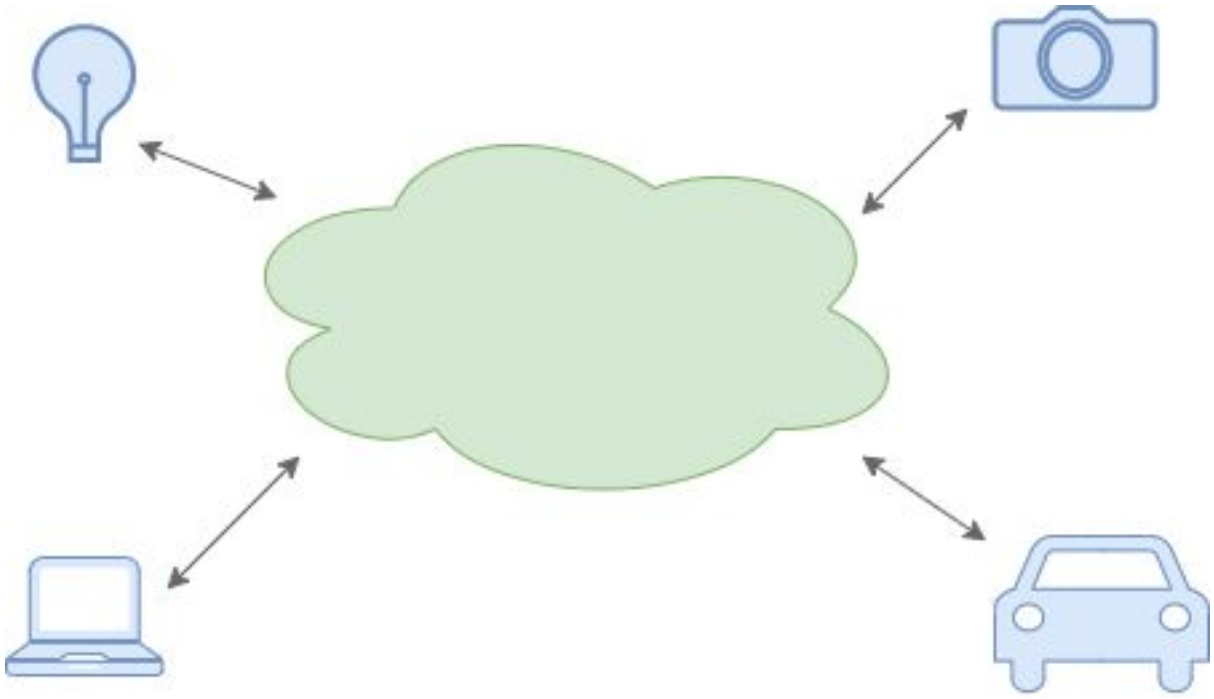
SAPIENZA
UNIVERSITÀ DI ROMA

Internet of Underwater Things

Christian Cardia, Claudio Starnoni
Dept. of Computer Science

What is Internet of Things?



IoUT
Internet of Underwater Things

Why is it important?

- 71% of our planet is covered with water;
- Ocean temperatures determine the climate and winds that affect life on earth;
- The main goal of IoUT is to extend the Internet of Things in the Underwater world;
- Thanks to this, it is possible to constantly monitor the oceans, pollutions (and not only that...)...

Real applications

- Ocean sampling and monitoring
- Environmental monitoring
- Exploration
- Prevention of environmental disasters
- Navigation assistance
- Distributed surveillance
- Detention of mines
- Etc.

IoUT: wireless communication

Radio waves: They propagate over long distances only at extremely low frequencies (30 - 300 Hz) -> require huge, very expensive antennas that involve a large power consumption.

Optical waves: Very fast transmission (up to 10Mbps) but can be used for short-range communications (e.g., 10 meters) and require high accuracy between source and destination (not easy in underwater).

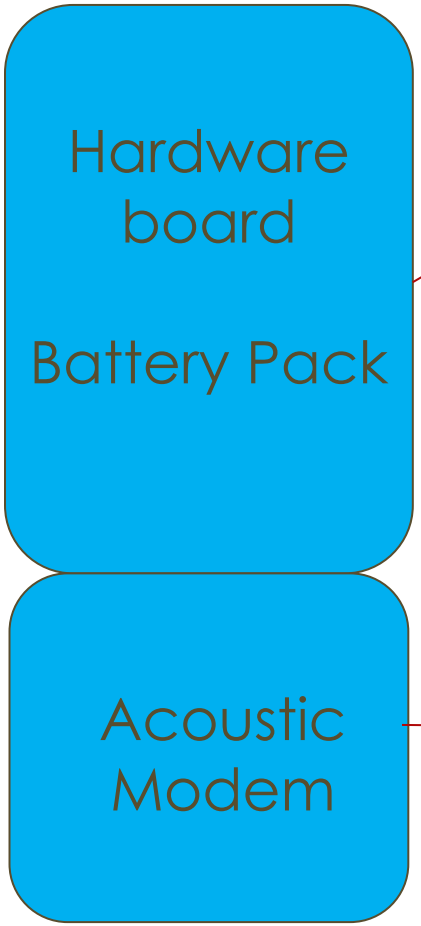
IoUT: wireless communication

Acoustic waves: This technology enables long-range communication, up to several kilometers. Despite this, it still presents difficult issues to address:

- Low Data Rate: 80bps - 64kbps
- Variable and very long propagation delays
- Very low bandwidth
- High BER (Bit Error Rate)

An underwater acoustic network consists of a set of nodes that can act as a source node, destination node or relay node.

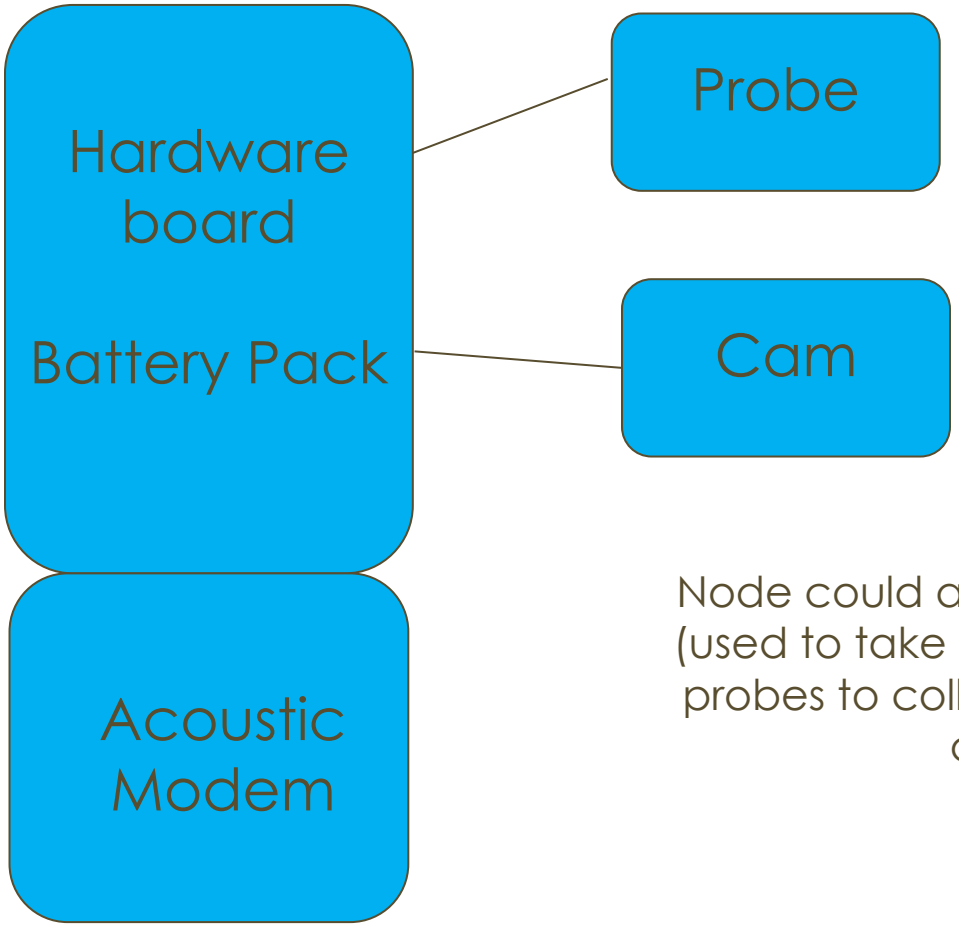
What do we mean by node? (1/2)



Inside this "box" is a hardware (which for example could be an Arduino board) with installed the protocols to be used (applications, routing protocols, mac etc.) and a battery.

The "box" described above is connected to an acoustic modem used to receive or send data via acoustic waves.

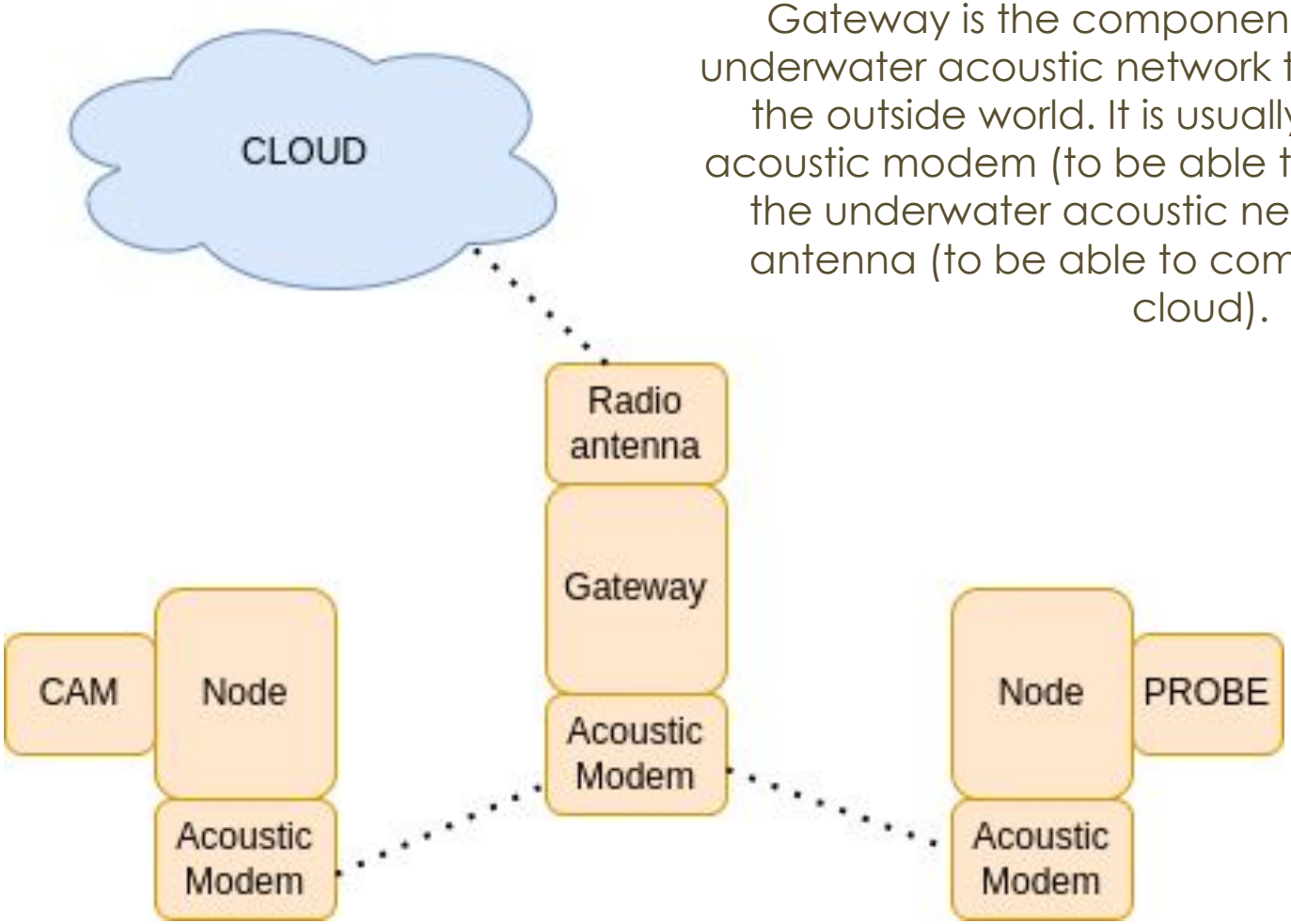
What do we mean by node? (2/2)



Node could also have connected a camera (used to take photos or transmit live video) or probes to collect environmental parameters and transmit them

What is the Gateway?

Gateway is the component that enables the underwater acoustic network to communicate with the outside world. It is usually connected to an acoustic modem (to be able to communicate with the underwater acoustic network) and a radio antenna (to be able to communicate with the cloud).



Gateway- Cloud: communication protocols

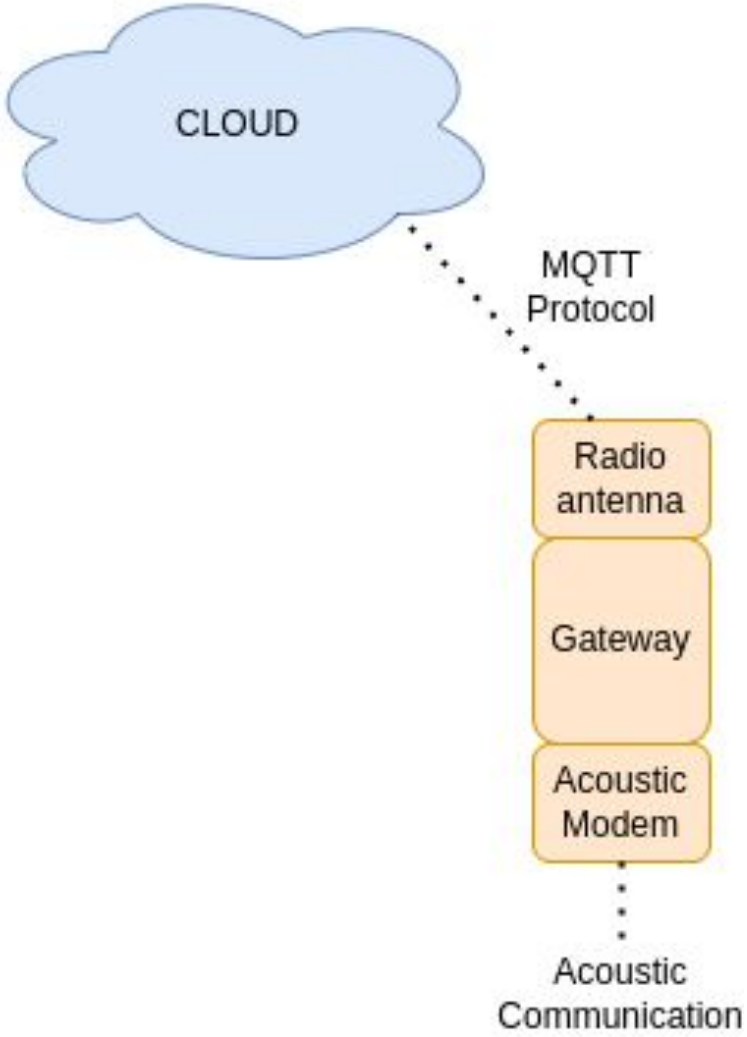
There are many protocols to be able to communicate, which may vary depending on the layer of the stack in which they are found.
In fact, an IoT system might use multiple protocols that bridge the different layers.
An example would be Bluetooth and wireless that support communication at the lower layers of the stack, while DDS (Data Distribution Service) and MQTT work at the application layer.

Among the most well-known protocols:

- AMQP (*Advanced Message Queuing Protocol*)
- Bluetooth and BLE
- Cellular
- CoAP (Data Distribution Service)
- LoRa and LoRaWAN
- LWM2M
- Etc..

In this lesson we are going to analyze the **MQTT (Message Queuing Telemetry Transport)** protocol.

MQTT protocol



Why MQTT?

- packet agnostic

- can be used for low bandwidth applications

- lower battery power consumption

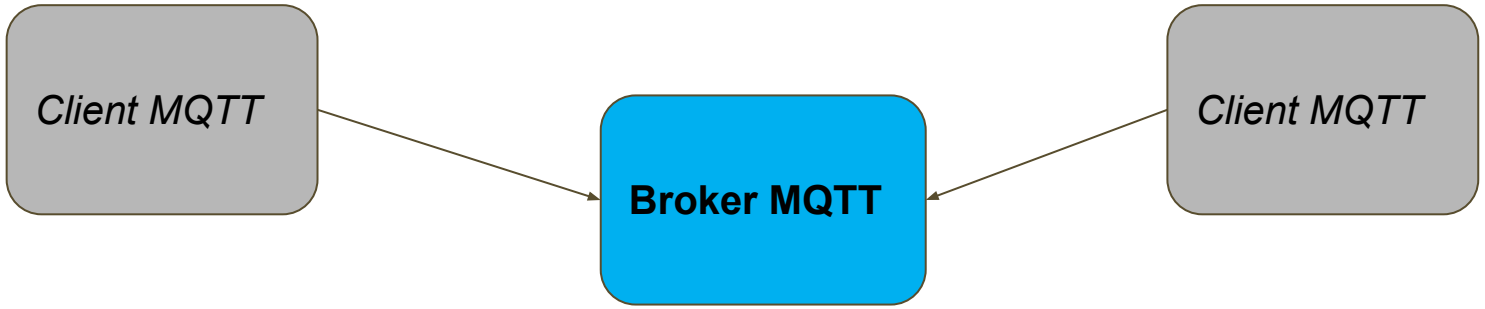
- QoS options

More info:
https://mqtt.org/

MQTT: How does it work?

The MQTT broker is the main component of the architecture. Its responsibility is to coordinate all the messages that are sent by the different clients. In particular, it must filter and forward the received messages to the clients that are subscribed and authorized to receive the specific message.

Communication between the different clients and the broker is through an MQTT connection. Clients never connect with each other and therefore do not communicate directly. They can only and exclusively connect with the broker to which all messages must be sent.



MQTT: publish/subscribe

Each client can subscribe to one or more topics and thus will receive all messages sent on the topics to which it is subscribed.

So a message sent by the client consists of a topic and a payload. **The topic** indicates the recipient clients, while **the payload** carries the content of the message, which can be in any format: JSON, XML, base64, binary, etc.

Topics have a hierarchical organization, such as a directory of files and folders. We can imagine a client sending temperature, oxygen and salinity measurements and then could send them to the three respective topics:

telemetry/temperature *telemetry/oxygen* *telemetry/salinity*

We can also imagine two clients sending the temperature and located in two different places, who publish their messages on the two respective topics:

telemetry/italy/temperature *telemetry/norway/temperature*

MQTT: subscribe with wildcards

SAPIENZA UNIVERSITÀ DI ROMA

The wildcards + (plus sign)

to match a single level of hierarchy - replaces one topic level

Example:
subscribe to: **telemetry/+/temperature**

In this case the client receives all temperature-related messages, from any location.

- ✚ telemetry/italy/temperature
- ✚ telemetry/norway/temperature
- ✖ telemetry/italy/oxygen
- ✖ telemetry/norway/salinity

MQTT: subscribe with wildcards

The wildcards

This symbol, in order to be interpreted correctly by the broker, must be placed as the last level of the topic.
Client receives all messages with the topic beginning with the pattern preceding the wildcard regardless of the length of the topic itself.

Example:
subscribe to: **telemetry/italy/#**

In this case the client receives all messages from italy.

- ✚ telemetry/italy/temperature
- ✖ telemetry/norway/temperature
- ✚ telemetry/italy/oxygen
- ✖ telemetry/norway/salinity

MQTT: QoS

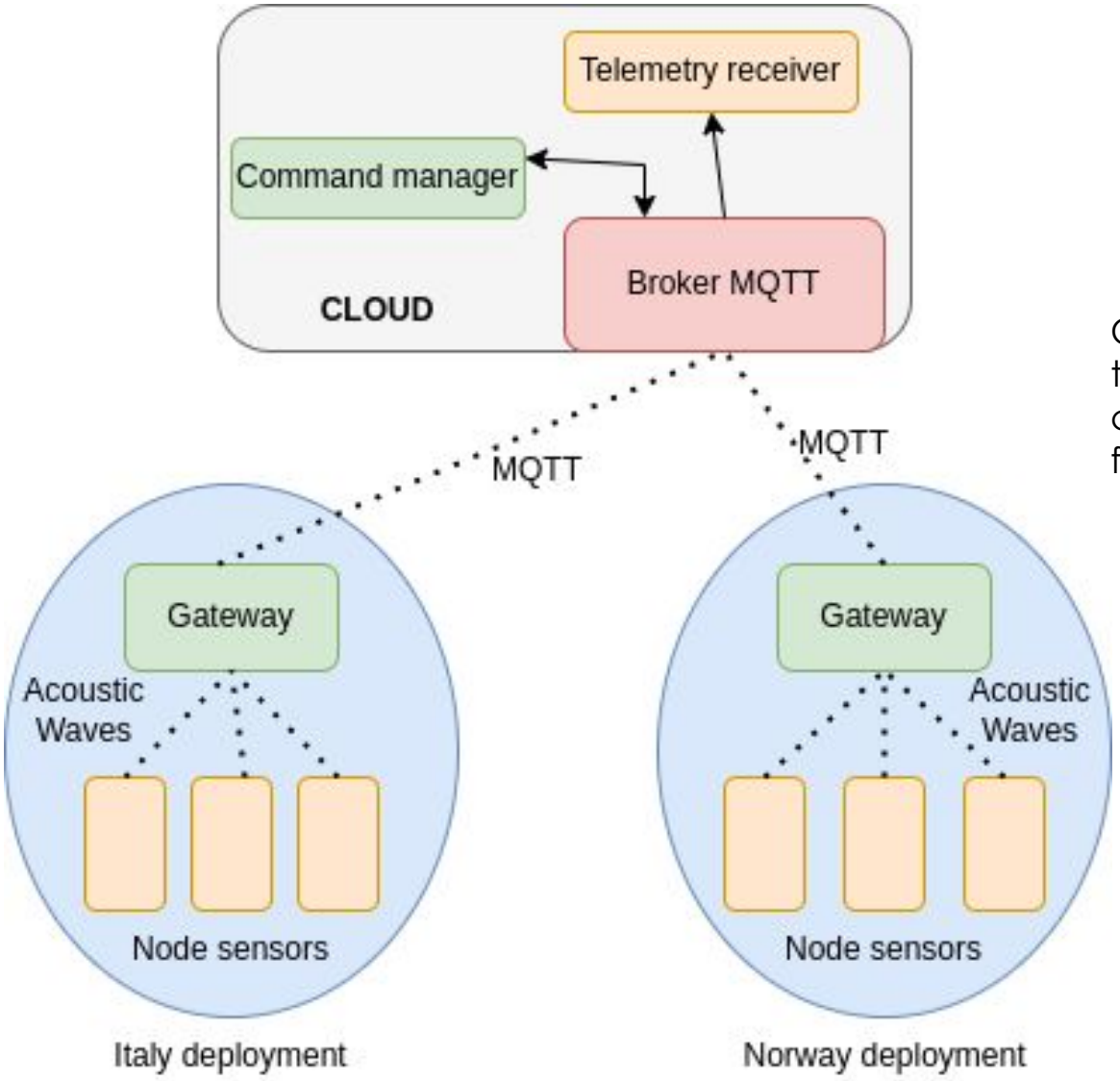
QoS is a fundamental feature of the protocol. Clients can choose which level of service to use, depending on the reliability of the network and the type of application.

QoS 0: This service provides no guarantee of message delivery. The recipient sends no confirmation and the message is neither stored nor retransmitted.

QoS 1: This level ensures that a message is delivered at least once. The message is retained until the client receives an ACK from the recipient. There is a possibility that the message will be sent or undelivered multiple times.

QoS 2: This is the highest level. It ensures that the message is received only once by the recipients. This service is the most secure but also the slowest.

A real example



Communication can be two-way. This allows to send commands to the sensor from the cloud.

MQTT: a real python client

step 1 (create a new python virtual environment with *venv* and install requirements)

- create a new environment, in a specific folder run the command:

python3 -m venv exercise

- now activate the new virtual environment:

source exercise/bin/activate

- now install the paho-mqtt library:

pip install paho-mqtt

- Finally, install a local mqtt broker (we use mosquitto):

sudo apt update -y && sudo apt install mosquitto mosquitto-clients -y

With this last command we have installed a local broker and a client MQTT (for local tests)

MQTT: a real python client

step 2 verify the local broker mosquitto is active

- To verify the mosquitto broker status, run:

sudo systemctl status mosquitto

```
(exercise) christian@christian-pc:~/Desktop/IoT_exercise$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
     Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2023-05-02 14:40:34 CEST; 40s ago
       Docs: man:mosquitto.conf(5)
             man:mosquitto(8)
    Process: 29091 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 29092 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
    Process: 29093 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
    Process: 29094 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
   Main PID: 29095 (mosquitto)
      Tasks: 1 (limit: 18775)
     Memory: 1.4M
        CPU: 24ms
     CGroup: /system.slice/mosquitto.service
             └─29095 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

mag 02 14:40:34 christian-pc systemd[1]: Starting Mosquitto MQTT Broker...
mag 02 14:40:34 christian-pc systemd[1]: Started Mosquitto MQTT Broker.
```

- If it is not active, run:

sudo systemctl start mosquitto

MQTT: publish a message

Now you can also use the mosquitto client in the following ways.
Subscribe to a topic:
mosquitto_sub -h localhost -t topic
Publish:
mosquitto_pub -h localhost -t topic -m "message"

EXERCISE: subscribe to the topic telemetry/# and run in the virtual environment the following python script.

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

BROKER_ADDRESS = "localhost"
TOPIC_TO_PUBLISH = "telemetry/italy/temperature"
PAYLOAD_TO_PUBLISH = "22"

client = mqtt.Client("client_id_11") # create new client (client id must be unique)
client.connect(BROKER_ADDRESS) # connect to broker

client.publish(TOPIC_TO_PUBLISH, PAYLOAD_TO_PUBLISH) # publish message

print(f"Message correctly sent!")
```

MQTT: subscribe to a topic

EXERCISE: run in the virtual environment the following script.

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

BROKER_ADDRESS = "localhost"
TOPIC_TO_SUBSCRIBE = "telemetry/#"

client = mqtt.Client("client_id_11") # create new client (client id must be unique)
client.connect(BROKER_ADDRESS) # connect to broker

def on_message(client, userdata, message):
    """
    callback function called when a new message is received
    """
    print(f"Received new message: {message.payload.decode('utf-8')}")
    print(f"Topic of the message: {message.topic}")

client.subscribe(TOPIC_TO_SUBSCRIBE) # subscribe client to a topic
# attach our callback function to our client object as follows
client.on_message=on_message
# run a loop otherwise we won't see the callbacks
client.loop_start() # start the loop

# loop to keep script running
try:
    while True:
        time.sleep(1)

except KeyboardInterrupt:
    print("Script terminated!")
    client.disconnect()
    client.loop_stop()
```

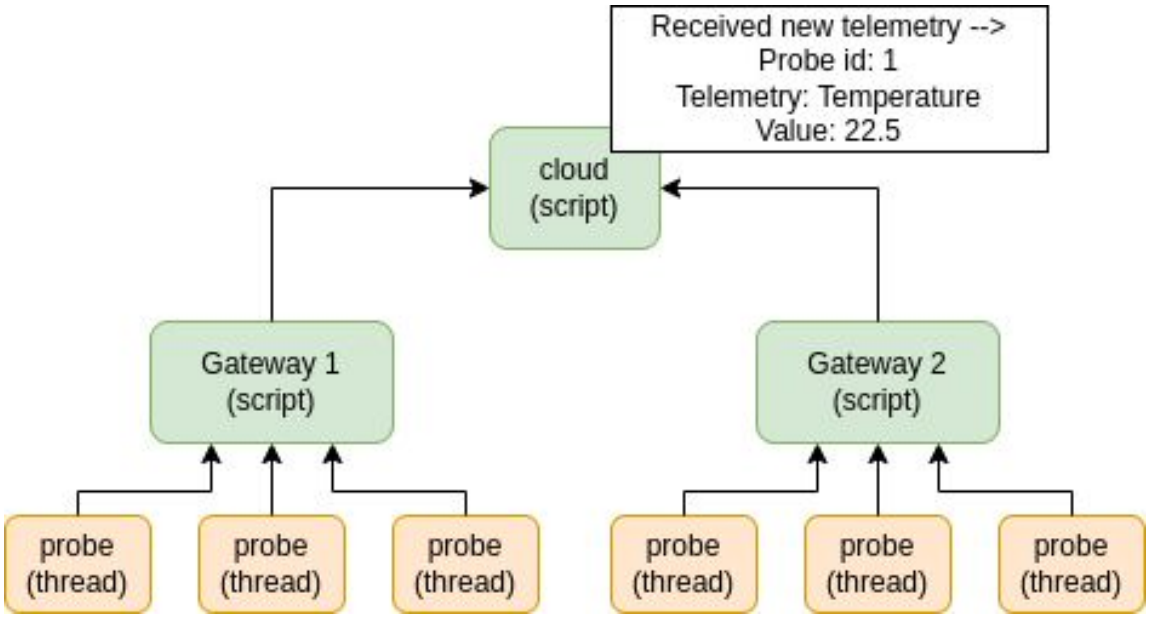
Now publish messages with the mosquitto client command

Exercise (step 1)

The exercise consists to implement, each one simulated by a python script, the following components:

- **2 different gateways** and each receives telemetry from three sensors (simulated by threads). The telemetries from the three sensors are temperature, salinity and oxygen, respectively and are random values. The three sensors can send telemetries every X seconds, where X can be chosen as desired. Each sensor is identified by the name of the telemetry it sends.

- **1 script that simulates the cloud** (which must then receive all telemetry from both gateways) and prints them to the terminal.

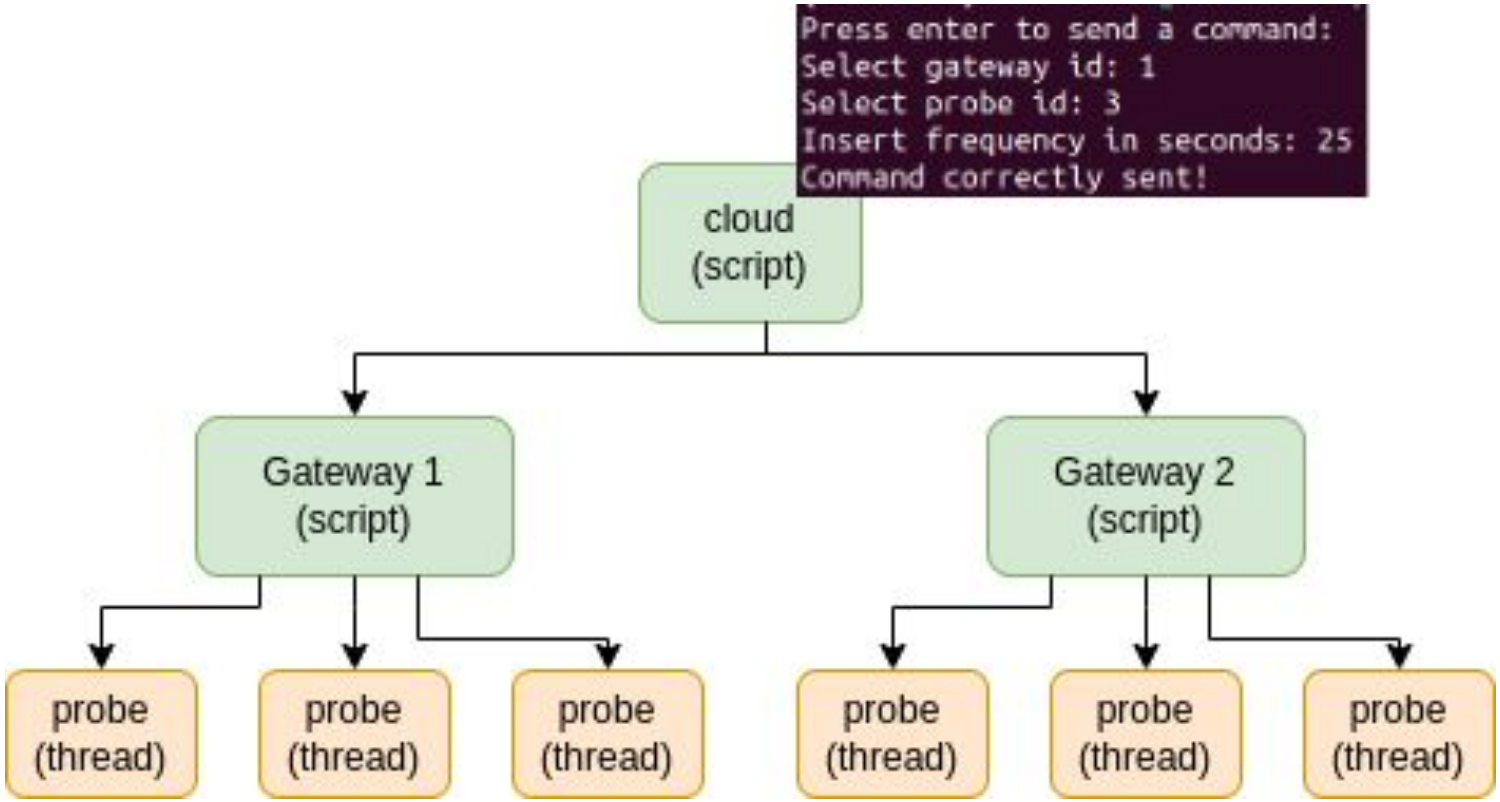


Suggestion: project topics so that telemetry can be differentiated from gateways and sensors.

Each telemetry is composed by a value and a timestamp (use a json message)

Exercise (step 2)

Once the previous exercise is complete, add the ability on the script that simulates the cloud (via a text interface on the terminal) to send commands (through the gateway) to individual sensors. The command should be able to set how often a sensor should send telemetry. Suggestion: use topics in a way that a command for a gateway is not received from the other one.



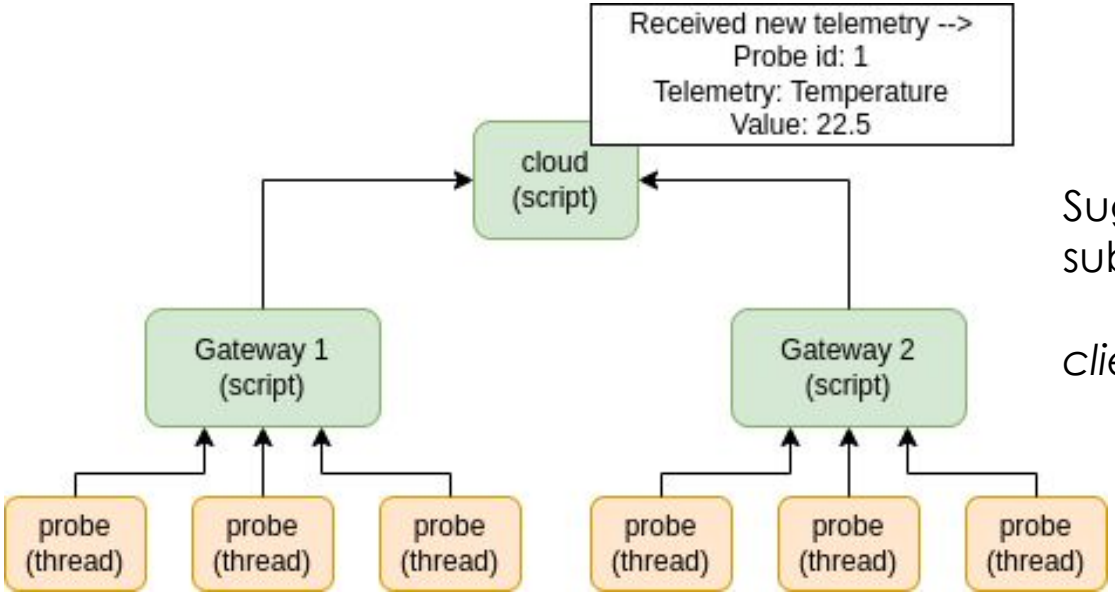
Exercise (step 3)

Through a text interface on the terminal, allow the user on the cloud to choose what telemetry receive. The filter can be for:

- gateway id

- telemetry name

- gateway id and telemetry name



Suggestion: to change topic subscription, use the function

client.unsubscribe(topic_name)

Possible solutions