Inatel
Instituto Nacional de Telecomunicações

CONTROLE AUTOMÁTICO DE TEMPERATURA DE UM LABORATÓRIO NO INATEL

PROJETO

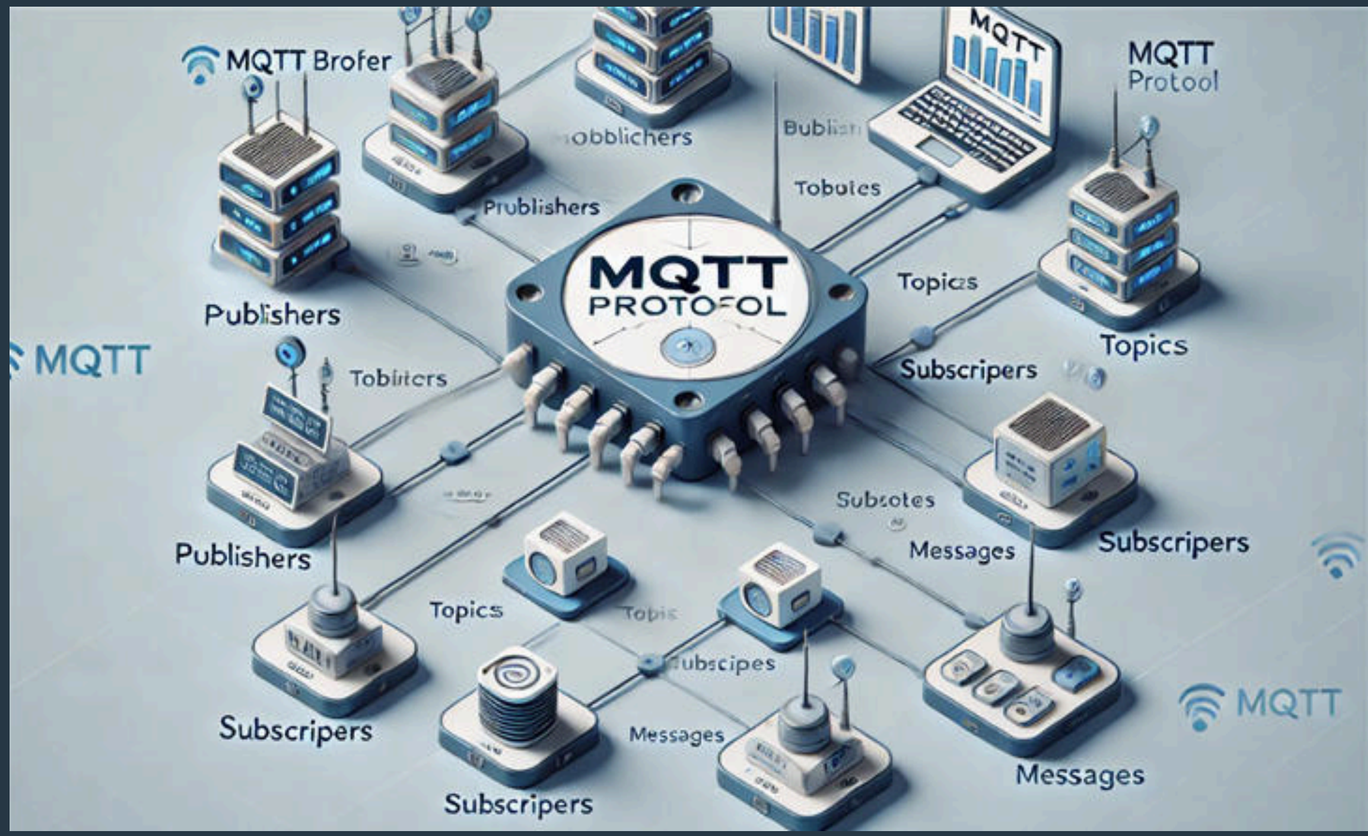
TP-546 IoT

Agiana da Silva e Rafaella Dias

1. O QUE É MQTT ?

O MQTT, (MESSAGE QUEUING TELEMETRY TRANSPORT), É UM PROTOCOLO LEVE DE COMUNICAÇÃO, USADO PRINCIPALMENTE PARA A TROCA DE MENSAGENS ENTRE DISPOSITIVOS EM REDES, ESPECIALMENTE EM IOT (INTERNET DAS COISAS).

ELE É BEM EFICIENTE EM TERMOS DE USO DE BANDA E É IDEAL PARA AMBIENTES COM POUCA LARGURA DE BANDA OU COM CONEXÕES INSTÁVEIS.



2. OBJETIVO PRINCIPAL

O MQTT serve para transmitir mensagens pequenas entre dispositivos conectados em rede, como:

1- Sensores de temperatura, umidade, GPS, etc.

2- Atuadores (como relés, LEDs, motores).

3- Gateways IoT e servidores na nuvem.

4- Tudo isso de forma leve, simples e com baixo consumo de energia e banda.

3. VANTAGENS

1

BAIXO CONSUMO DE ENERGIA

Os dispositivos podem dormir e acordar só para enviar dados.

2

Assíncrono

os dispositivos não precisam estar online ao mesmo tempo.

3

Escalável

Pode conectar milhares de sensores simultaneamente.

O MQTT usa o modelo publicador / corretor / assinante:

Papel	Função
Publicador (Publisher)	Envia mensagens para um **tópico**. Exemplo: “sensor/temperatura”.
Corretor (Broker)	É o **servidor central** que recebe todas as mensagens dos publicadores e as repassa aos assinantes.
Assinante (Subscriber)	Recebe mensagens de um ou mais tópicos de interesse.

4. O QUE É PAHO?

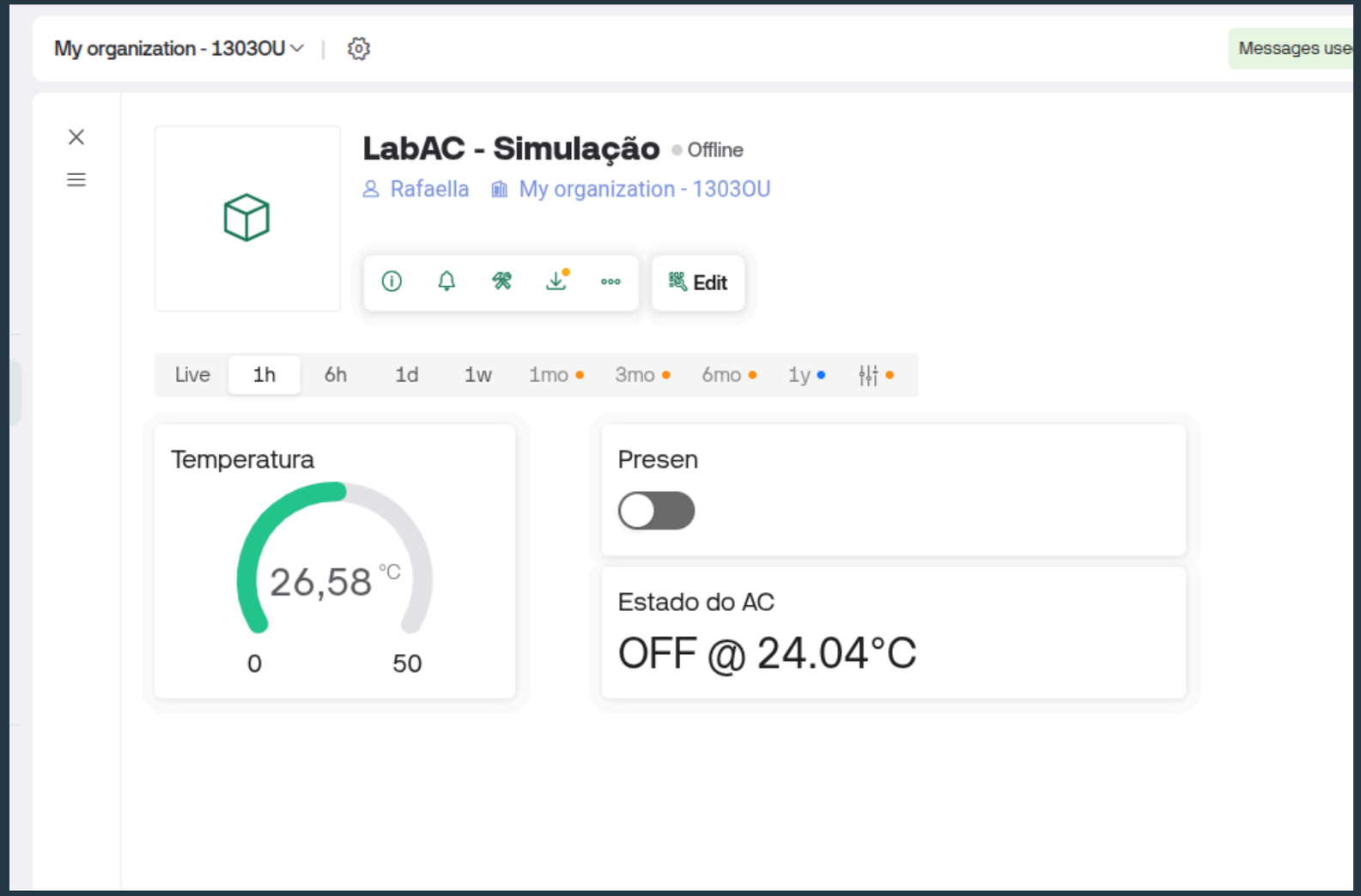
Paho é uma implementação de cliente MQTT desenvolvida pela Eclipse Foundation. É uma biblioteca que permite usar o protocolo MQTT facilmente em diversas linguagens, como:

- C/C++J
- avaJavaScript (para web)
- Python.

Termo	Função
MQTT	É o **protocolo de comunicação** — define como as mensagens devem ser trocadas.
Paho	É uma **biblioteca cliente** que **implementa** o protocolo MQTT em várias linguagens, facilitando o uso.

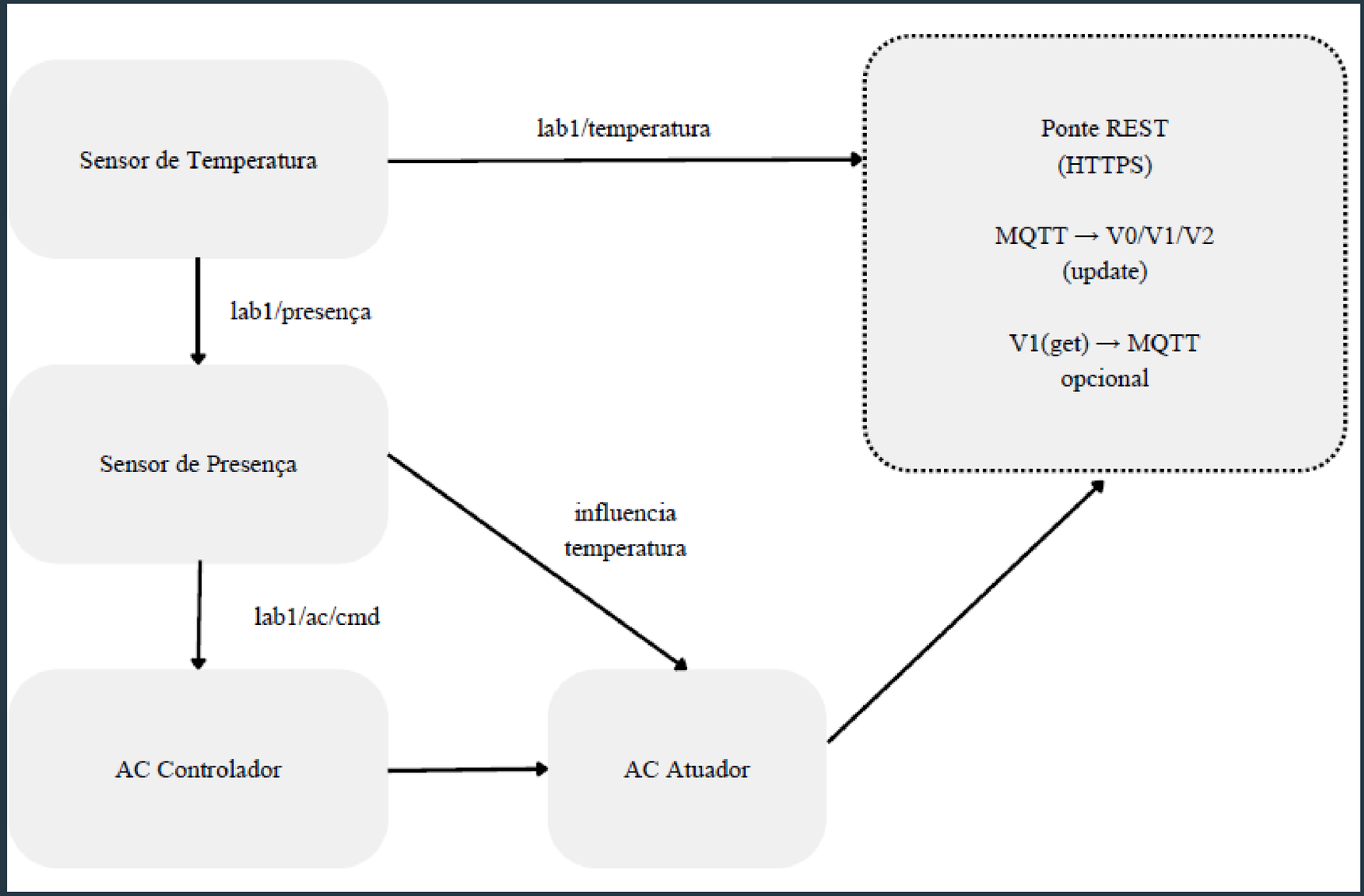
6. SIMULAÇÃO DE CONTROLE DE TEMPERATURA COM MQTT + BLYNK (V0–V2) + PAHO

- **V0 – Gauge “Temperatura”: número em °C mudando devagar.**
- **V1 – Presença: (switch) alterna sozinho conforme o simulador (ou você clica).**
- **V2 – Label “Estado do AC”: texto muda entre OFF, COOL e SLEEP (ex.: COOL @ 29.2°C).**



- **OBSERVAÇÃO: o cabeçalho do Blynk pode mostrar “Offline”, mas os widgets atualizam em tempo real (a ponte usa HTTPS e não socket).**

Diagrama de blocos (resumo)



7. O QUE CADA TÓPICO CARREGA

- **lab 1/presence → {"present": true}**
- **lab 1/ac/cmd →{"power":"on|off","mode":"cool|sleep|off","setpoint":24.0}**
- **lab 1/ac/state → {"state":"COOL|SLEEP|OFF","temp":24.4,"present":true}**
- **lab 1/temperature → {"celsius": 24.37}**

7.1 Mapeamento Blynk (apenas V0–V2)

Vpin	Widget	Origem/Destino	O que mostra/recebe
V0	Gauge “Temperatura”	MQTT → Blynk	Valor de lab1/temperature (°C)
V1	Switch “Presença”	MQTT ⇄ Blynk	Espelha/publica lab1/presence
V2	Label “Estado do AC”	MQTT → Blynk	Texto OFF/COOL/SLEEP @ X°C de lab1/ac/state

7.2 Como os dados fluem (passo a passo)

“Temperatura esfria com o AC”

[1] sensor_temperature → lab1/temperature (ex.: 30.00 °C)

[2] ac_controller assina temperat
ura + presença e decide: - se presença=true e temp≥28 → publica lab1/ac/cmd={"on","cool",24}

[3] ac_actuator recebe cmd e “esfria” a temp (cool_rate).

[4] ac_actuator publica nova temp → lab1/temperature (feedback).

[5] ac_controller publica estado → lab1/ac/state (ex.: "COOL @ 29.5°C").[6] bridge REST envia:
V0 ← temperatura V2 ← "COOL @ 29.5°C"

```python
import json, time
from datetime import datetime
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC_TEMP = "lab1/temperature"
TOPIC_PRES = "lab1/presence"
TOPIC_CMD  = "lab1/ac/cmd"
TOPIC_STATE= "lab1/ac/state"

SETPOINT = 24.0
ON_AT    = 28.0
HYST     = 0.5
MIN_OFF  = 120  # segundos mínimo OFF

state = "OFF"          # OFF | COOL | SLEEP
last_temp = None
present = False
last_off_time = datetime.min

def publish_cmd(c, power, mode, setpoint):
    c.publish(TOPIC_CMD, json.dumps({"power": power, "mode": mode, "setpoint": setpoint}),
    qos=1, retain=True)

def publish_state(c, msg):
    c.publish(TOPIC_STATE, json.dumps(msg), qos=1, retain=True)

def on_message(c, userdata, msg):
    global last_temp, present
    try:
        data = json.loads(msg.payload.decode())
        if msg.topic == TOPIC_TEMP:
            last_temp = float(data["celsius"])
        elif msg.topic == TOPIC_PRES:
            present = bool(data["present"])
    except Exception:
        pass
```

```python
def main():
    global state, last_off_time
    c = mqtt.Client(client_id="ac_controller_lab1")
    c.on_message = on_message
    c.connect(BROKER, 1883, 60)
    c.subscribe([(TOPIC_TEMP,1),(TOPIC_PRES,1)])
    c.loop_start()

    publish_state(c, {"state": state, "reason":"boot"})
    while True:
        if last_temp is None:
            time.sleep(0.5); continue

        now = datetime.now()
        can_turn_on = (now - last_off_time).total_seconds() >= MIN_OFF

        # lógica de decisão
        desired, mode = "OFF", "off"
        if not present:
            desired, mode = "OFF", "off"
        else:
            if last_temp >= ON_AT and can_turn_on:
                desired, mode = "COOL", "cool"
            elif SETPOINT - HYST <= last_temp <= SETPOINT + HYST:
                desired, mode = "SLEEP", "sleep"
            elif last_temp > SETPOINT + HYST and can_turn_on:
                desired, mode = "COOL", "cool"
            else:
                desired, mode = "OFF", "off"

        if desired != state:
            state = desired
            if state == "OFF":
                publish_cmd(c, "off", "off", SETPOINT)
                last_off_time = now
            elif state == "COOL":
                publish_cmd(c, "on", "cool", SETPOINT)
            elif state == "SLEEP":
                publish_cmd(c, "on", "sleep", SETPOINT)

            publish_state(c, {"state": state, "temp": round(last_temp,2), "present": present, "reason":"rule"})
```

7.3 “Presença alterna e o AC desliga”

[1] sensor_presence publica lab1/presence={"present": false}

[2] ac_controller vê presence=false → estado=OFF → lab1/ac/cmd={"off","off",24}

[3] ac_actuator para de resfriar; a temp volta lentamente p/ “outside”.

[4] ac_controller publica lab1/ac/state={"state":"OFF",...}

[5] bridge REST atualiza V1 (switch) e V2 (label).

```python
import json, time
import paho.mqtt.client as mqtt

BROKER = "localhost"
TOPIC_CMD  = "lab1/ac/cmd"
TOPIC_TEMP = "lab1/temperature"

outside = 31.0
temp = 30.0
cool_rate = 0.25
sleep_rate = 0.08
leak_rate = 0.02

mode = "off"
setpoint = 24.0
power = "off"

def on_message(c, userdata, msg):
    global mode, setpoint, power
    try:
        data = json.loads(msg.payload.decode())
        power = data.get("power","off")
        mode = data.get("mode","off")
        setpoint = float(data.get("setpoint",24.0))
    except Exception:
        pass
```

```python
def main():
    global temp
    c = mqtt.Client(client_id="ac_actuator_sim_lab1")
    c.on_message = on_message
    c.connect(BROKER, 1883, 60)
    c.subscribe([(TOPIC_CMD,1)])
    c.loop_start()

    while True:
        if power == "on" and mode == "cool":
            if temp > setpoint:
                temp -= cool_rate
        elif power == "on" and mode == "sleep":
            if temp > setpoint: temp -= sleep_rate
            elif temp < setpoint: temp += sleep_rate/2.0
        else:
            if temp < outside: temp += leak_rate
            elif temp > outside: temp -= leak_rate

        c.publish(TOPIC_TEMP, json.dumps({"celsius": round(temp,2)}), qos=1, retain=True)
        time.sleep(1.0)

if __name__ == "__main__":
    main()
```

8. POR QUE ESSA ARQUITETURA É FUNCIONAL PARA IOT?

MQTT (paho-mqtt): protocolo leve, publish/subscribe, perfeito para sensores/atuadores.

Histerese + tempo mínimo OFF: práticas reais para conforto e durabilidade.

Blynk via REST: painel web simples, funciona por HTTPS (porta 443), sem dor de rede.

Scripts separados: cada peça é independente — como se fossem dispositivos reais.

OBRIGADO PELA ATENÇÃO!