

LABORATORUL 04

Nod IoT complet: de la senzor la panou de comandă

Durată: 4 ore

Suport: Capitolul 11

Platformă: ESP32-C6 + Raspberry Pi 5 ca punte

Carte capitolul de referință

CUPRINS	
1.	Obiectivele lucrării
2.	Arhitectura sistemului
3.	Materiale necesare
4.	De ce MQTT și nu HTTP
5.	Ierarhia de topicuri
6.	Instalarea brokerului
7.	Nodul care publică
8.	Puntea: colectare, stocare, afișare
9.	Calitatea serviciului și costul ei
10.	Ce vede cineva care ascultă
11.	Sarcini de lucru
12.	Provocare
13.	Întrebări de verificare
14.	Resurse

Un senzor care măsoară perfect, dar nu poate trimite datele nimănui, nu folosește la nimic. În această lucrare legăm nodul de la lucrarea a doua la o punte care rulează pe Raspberry Pi, prin protocolul MQTT. Apoi facem ceva ce se sare aproape întotdeauna la cursurile de IoT: ascultăm propriul trafic și vedem, cu ochii noștri, ce anume scapă pe rețea.

1 Obiectivele lucrării

- ☐ Înțelegerea modelului publicare–abonare și a motivului pentru care înlocuiește cererea–răspuns în IoT
- ☐ Proiectarea unei ierarhii de topicuri și stăpânirea metacaracterelor + și #
- ☐ Instalarea și configurarea unui broker MQTT pe Raspberry Pi
- ☐ Publicarea din ESP32 și consumul datelor într-o punte scrisă în Python
- ☐ Folosirea mesajului de adio și a mesajelor reținute pentru a detecta nodurile căzute
- ☐ Alegerea nivelului de calitate a serviciului în funcție de costul energetic
- ☐ Demonstrarea practică a vulnerabilităților unei configurații implicite și securizarea ei

2 Arhitectura sistemului

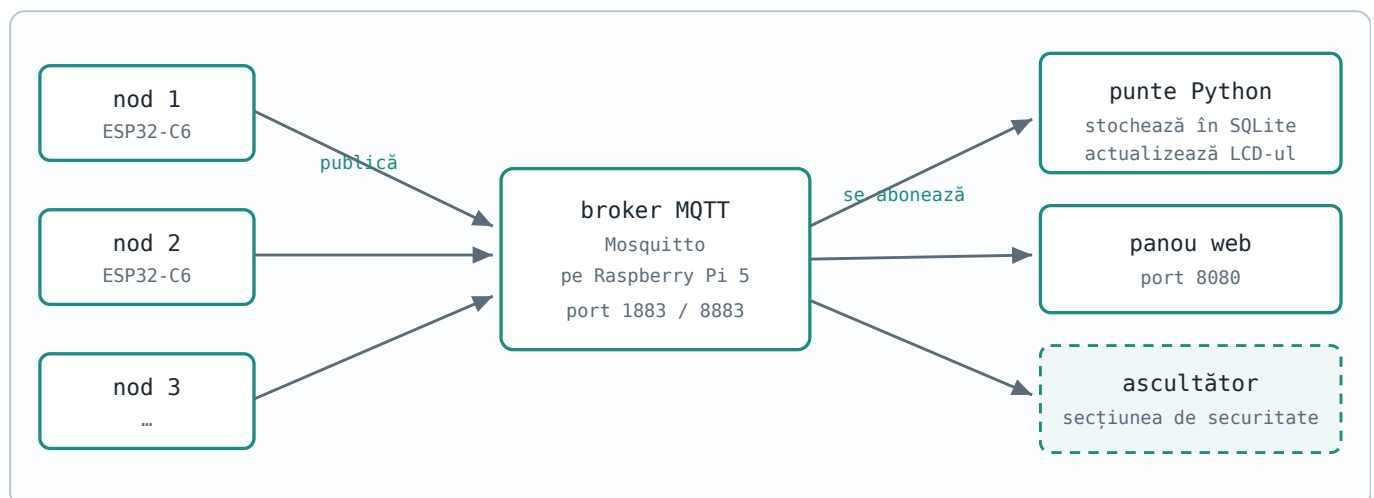


Fig. 1 - Nodurile nu se cunosc între ele și nu cunosc consumatorii. Toată lumea vorbește doar cu brokerul. Se pot adăuga noduri sau aplicații noi fără să se modifice nimic la cele existente - aceasta este proprietatea care face modelul potrivit pentru IoT.

3 Materiale necesare

1

Raspberry Pi 5 (din lucrarea 1)

2

ESP32-C6 cu senzor (din lucrarea 2)

1

LCD 16×2 I²C

1

Rețea Wi-Fi de laborator, la care aveți acces

1

Calculator pentru panoul web

REGULA DE AUR A ACESTEI LUCRĂRI

Secțiunea 10 presupune interceptarea traficului de rețea. Această operație se face **EXCLUSIV** pe rețeaua de laborator, pe traficul generat de propriile dumneavoastră dispozitive, cu brokerul dumneavoastră. Interceptarea traficului altcuiva, chiar și pe aceeași rețea, este ilegală. Scopul este să înțelegeți de ce configurația implicită nu este acceptabilă - nu să exersați pe sisteme străine.

4 De ce MQTT și nu HTTP

La lucrarea a doua, nodul trimitea datele cu o cerere HTTP. A funcționat. De ce am schimba ceva?

	HTTP	MQTT
Model	cerere–răspuns: clientul întreabă, serverul răspunde	publicare–abonare: nodul anunță, cine vrea ascultă
Antet minim	sute de octeți	2 octeți
Conexiune	se deschide și se închide la fiecare mesaj	rămâne deschisă
Server → nod	imposibil fără interogare repetată	imediat, prin abonament
Detectarea nodurilor căzute	trebuie construită separat	inclusă: mesajul de adio
Consumatori multipli	fiecare trebuie contactat separat	brokerul multiplică singur

CEI DOI OCTEȚI CONTEAZĂ CU ADEVĂRAT

Un antet HTTP minimal are, cu tot cu răspuns, câteva sute de octeți. La un nod care transmite o valoare de temperatură de zece octeți, antetul este de treizeci de ori mai mare decât datele. Fiecare octet în plus înseamnă timp de emisie radio, iar timpul de emisie este cea mai scumpă operație din bugetul energetic calculat la lucrarea a doua. MQTT a fost proiectat pentru legături prin satelit ale conductelor de petrol, unde fiecare octet se plătea la propriu.

Rolul brokerului

Nodul nu trimite date *către cineva*. Le publică pe un **topic**, iar brokerul le trimite tuturor celor abonați la acel topic. Nodul nu știe câți consumatori are - poate zero, poate zece. Consumatorii nu știu câte noduri există. Această decuplare este întreaga idee: puteți adăuga un panou web, o alarmă și o bază de date fără să reprogramați niciun senzor.

5 Ierarhia de topicuri

Un topic este un șir cu niveluri separate prin /, ca o cale de fișier. Un abonament poate folosi metacaractere. Aici se nasc cele mai multe neînțelegeri, așa că merită exersat interactiv.

► CE ABONAMENTE PRIMESC UN MESAJ PUBLICAT

TOPIC PUBLICAT

NIVELURILE TOPICULUI

1: lab 2: si 3: nod1 4: temperatura

lab/si/nod1/temperatura **PRIMEȘTE** lab si nod1 temperatura toate cele 4 niveluri se potrivesc.

lab/si/+/temperatura **PRIMEȘTE** lab si + temperatura toate cele 4 niveluri se potrivesc.

lab/si/nod1/# **PRIMEȘTE** lab si nod1 # „#” preia nivelul 4 și tot ce urmează.

lab/si/# **PRIMEȘTE** lab si # „#” preia nivelul 3 și tot ce urmează.

lab/+/nod1/temperatura **PRIMEȘTE** lab + nod1 temperatura toate cele 4 niveluri se potrivesc.

lab/si/+ **nu primește** lab si + topicul are 4 niveluri, filtrul doar 3 - fără „#” nu se potrivesc.

+ / + / + / + **PRIMEȘTE** + + + + toate cele 4 niveluri se potrivesc.

PRIMEȘTE # „#” preia nivelul 1 și tot ce urmează.

+ înlocuiește exact un nivel · # înlocuiește zero sau mai multe niveluri și poate apărea doar la sfârșit · topicurile care încep cu \$ nu sunt prinse de un metacarakter aflat pe primul nivel.

Versiune interactivă pe site.

ÎNCERCAȚI ACESTE TOPICURI, PE RÂND

- lab/si/nod1/temperatura - cazul obișnuit
- lab/si/nod1/umiditate - observați care abonamente cad
- lab/si/nod1 - de ce lab/si/# îl prinde, iar lab/si+/temperatura nu?
- lab/si/nod1/senzor/temperatura - de ce + nu acoperă două niveluri?

Cum se proiectează o ierarhie bună**REGULA: DE LA GENERAL LA PARTICULAR**

lab / si / nod1 / temperatura

↑ ↑ ↑ ↑

locul grupa nodul mărimea

Așezați nivelurile în ordinea în care veți dori vreodată să filtrați. Cu această ierarhie puteți cere „toate temperaturile” (lab/si+/temperatura) sau „tot ce spune nodul 1” (lab/si/nod1/#). Cu ordinea inversă - temperatura/nod1/si/lab - a doua întrebare devine imposibilă.

TREI GREȘELI FRECVENTE**• NIVEL GOL LA ÎNCEPUT:**

/lab/si/nod1 începe cu un nivel vid. Este valid, dar aproape sigur nu ați vrut asta - iar lab/si/nod1 nu se va potrivi cu el.

• DATE ÎN TOPIC:

lab/si/nod1/temperatura/23.5 pare ingenios, dar generează un topic nou la fiecare măsurătoare și umple brokerul. Valoarea merge în conținutul mesajului, nu în topic.

• ABONAMENT LA # ÎN PRODUCȚIE:

primiți absolut tot, inclusiv traficul intern al brokerului. Util la depanare, dezastruos pentru consumul unui nod pe baterie.

6 Instalarea brokerului

1**Instalați Mosquitto pe Raspberry Pi**

```
$ instalare
```

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
sudo systemctl enable mosquitto
```

2**Configurați-l pentru laborator**

Configurația implicită acceptă doar conexiuni locale. Deschidem portul pentru rețea - deocamdată **fără nicio măsură de securitate**, tocmai ca să vedem la secțiunea 10 ce înseamnă asta.

```
$ creare fișier de configurare
```

```
sudo nano /etc/mosquitto/conf.d/lab.conf
```

```
/etc/mosquitto/conf.d/lab.conf
```

```
# ATENTIE: configuratie nesigura, folosita deliberat pentru
# prima parte a lucrarii. Se inlocuieste la sectiunea 10.
listener 1883 0.0.0.0
allow_anonymous true

# Jurnalizare detaliata, ca sa vedem ce se intampla.
log_type all
log_dest file /var/log/mosquitto/mosquitto.log
```

```
$ repornire și verificare
```

```
sudo systemctl restart mosquitto
sudo systemctl status mosquitto --no-pager
```

3

Aflați adresa brokerului

```
$ adresa în rețea
```

```
hostname -I
```

Notați adresa - o veți scrie în programul nodului. Ideal, rezervați-i o adresă fixă în routerul de laborator, altfel se schimbă la fiecare repornire și nodurile rămân orfane.

4

Testați brokerul cu el însuși

Deschideți două ferestre de terminal. În prima:

```
$ terminalul 1 - ascultăm tot
```

```
mosquitto_sub -h localhost -t 'lab/si/#' -v
```

În a doua:

```
$ terminalul 2 - publicăm
```

```
mosquitto_pub -h localhost -t 'lab/si/nod1/temperatura' -m '23.5'
mosquitto_pub -h localhost -t 'lab/si/nod2/umiditate' -m '61'
```

Mesajele trebuie să apară imediat în prima fereastră. Dacă apar, brokerul funcționează și puteți trece la nod. Dacă nu, problema este locală și nu are legătură cu ESP32 - rezolvați-o aici, este mult mai ușor de depanat.

7 Nodul care publică

Instalați biblioteca **PubSubClient** din managerul de biblioteci al mediului Arduino, apoi încărcați programul de mai jos. Este nodul de la lucrarea a doua, cu HTTP înlocuit prin MQTT.

nod_mqtt.ino

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <esp_sleep.h>

#define uS_PER_S      1000000ULL
#define PERIODADA_S    60
#define ID_NOD         "nod1"

const char* SSID       = "reteaua_laboratorului";
const char* PAROLA     = "parola";
const char* BROKER     = "192.168.1.10";
const uint16_t PORT    = 1883;

// Topicurile se construiesc o singura data, din identificatorul nodului.
const char* T_TEMP     = "lab/si/" ID_NOD "/temperatura";
const char* T_UMID     = "lab/si/" ID_NOD "/umiditate";
const char* T_BATERIE  = "lab/si/" ID_NOD "/baterie";
const char* T_STARE    = "lab/si/" ID_NOD "/stare";

RTC_DATA_ATTR uint32_t ciclu = 0;

WiFiClient retea;
PubSubClient mqtt(retea);
Adafruit_BME280 bme;

bool conecteaza_broker() {
    mqtt.setServer(BROKER, PORT);

    // Mesajul de adio: brokerul il publica AUTOMAT daca nodul dispare
    // fara sa se deconecteze politicos. Este singurul mod de a distinge
    // "nodul tace pentru ca nu are ce spune" de "nodul a murit".
    return mqtt.connect(ID_NOD,           // identificator unic in retea
                        NULL, NULL,       // utilizator si parola (deocamdata fara)
                        T_STARE,          // topicul mesajului de adio
                        1,                 // calitatea serviciului
                        true,              // retinut: si cine se aboneaza mai tarziu il vede
                        "offline");       // continutul mesajului de adio
}

void setup() {
    Serial.begin(115200);
    delay(150);
    ciclu++;

    Wire.begin(6, 7);
    if (!bme.begin(0x76)) {
        Serial.println("senzorul nu raspunde");
    }
    bme.setSampling(Adafruit_BME280::MODE_FORCED,
                    Adafruit_BME280::SAMPLING_X1, Adafruit_BME280::SAMPLING_X1,
                    Adafruit_BME280::SAMPLING_X1, Adafruit_BME280::FILTER_OFF);
}
```

```

bme.takeForcedMeasurement();
float temperatura = bme.readTemperature();
float umiditate = bme.readHumidity();

WiFi.mode(WIFI_STA);
WiFi.begin(SSID, PAROLA);
uint32_t t0 = millis();
while (WiFi.status() != WL_CONNECTED && millis() - t0 < 10000) delay(10);

if (WiFi.status() == WL_CONNECTED && conecteaza_broker()) {
    char valoare[16];

    // Mesaj RETINUT: brokerul pastreaza ultima valoare si o trimite imediat
    // oricui se aboneaza. Un panou deschis peste o ora vede numere, nu un
    // ecran gol pana la urmatoarea masuratoare.
    snprintf(valoare, sizeof(valoare), "%.2f", temperatura);
    mqtt.publish(T_TEMP, valoare, true);

    snprintf(valoare, sizeof(valoare), "%.1f", umiditate);
    mqtt.publish(T_UMID, valoare, true);

    snprintf(valoare, sizeof(valoare), "%d", (int) (analogRead(0) / 40.95));
    mqtt.publish(T_BATERIE, valoare, true);

    // Anuntam ca suntem in viata; inlocuieste mesajul de adio retinut.
    mqtt.publish(T_STARE, "online", true);

    mqtt.loop(); // lasam biblioteca sa trimita efectiv
    delay(100);
    mqtt.disconnect(); // deconectare politicoasa: NU declanseaza adio
    Serial.printf("ciclul %lu publicat in %lu ms\n", ciclu, millis());
} else {
    Serial.println("nu am putut publica");
}

WiFi.disconnect(true);
WiFi.mode(WIFI_OFF);
Serial.flush();
esp_sleep_enable_timer_wakeup((uint64_t) PERIOADA_S * uS_PER_S);
esp_deep_sleep_start();
}

void loop() {}

```

Verificați imediat, de pe Raspberry Pi:

```
$ ascultăm nodul
```

```
mosquitto_sub -h localhost -t 'lab/si/#' -v
```

TESTUL MESAJULUI DE ADIO

Cu abonamentul de mai sus pornit,

ÎNTRERUPEȚI ALIMENTAREA NODULUI

în timp ce este conectat. După expirarea intervalului de așteptare (implicit de aproximativ 1,5 ori intervalul de menținere a conexiunii), brokerul publică singur lab/si/nod1/stare → offline. Nimeni nu a scris cod pentru asta. Comparați cu ce ar fi însemnat implementarea aceleiași funcții peste HTTP.

8 Puntea: colectare, stocare, afișare

Puntea rulează pe Raspberry Pi. Se abonează la tot ce publică nodurile, salvează într-o bază de date, actualizează LCD-ul și servește un panou web.

```
$ pregătire
```

```
source ~/si-lab/bin/activate
pip install paho-mqtt
mkdir -p ~/si-lab/lab04 && cd ~/si-lab/lab04
```

```
► punte.py
```

```
#!/usr/bin/env python3
"""Punte MQTT: colecteaza masuratorile, le salveaza si le afiseaza."""

import json
import sqlite3
import threading
import time
from http.server import BaseHTTPRequestHandler, HTTPServer

import paho.mqtt.client as mqtt

BROKER = "localhost"
PORT = 1883
TOPIC = "lab/si/#"
BAZA = "masuratori.db"
PORT_WEB = 8080

# Ultima valoare a fiecarui topic, pentru afisare rapida.
ultimele = {}
blocaj = threading.Lock()

def pregateste_baza():
    con = sqlite3.connect(BAZA)
    con.execute("""
        CREATE TABLE IF NOT EXISTS masuratori (
            moment REAL,
            nod TEXT,
            marime TEXT,
            valoare TEXT
        )
    """)
    con.execute("CREATE INDEX IF NOT EXISTS idx_moment ON masuratori(moment)")
    con.commit()
    return con

def la_conectare(client, date, steaguri, cod, proprietati=None):
    if cod == 0:
        print("conectat la broker")
        client.subscribe(TOPIC, qos=1)
    else:
        print("conectare esuata, cod", cod)
```

```

def la_mesaj(client, date, mesaj):
    text = mesaj.payload.decode("utf-8", errors="replace")
    nivele = mesaj.topic.split("/")

    # Asteptam forma lab/si/<nod>/<marime>; ignoram restul.
    if len(nivele) != 4:
        return
    _, _, nod, marime = nivele

    with blocaj:
        ultimele[nod, marime] = (text, time.time())

    con = date["con"]
    con.execute("INSERT INTO masuratori VALUES (?, ?, ?, ?)",
                (time.time(), nod, marime, text))
    con.commit()

    eticheta = " (retinut)" if mesaj.retain else ""
    print("%-34s %s%s" % (mesaj.topic, text, eticheta))

def actualizeaza_lcd(panou):
    """Roteste pe LCD, la fiecare 3 secunde, cate un nod."""
    while True:
        with blocaj:
            noduri = sorted({n for (n, _) in ultimele})
            instantaneu = dict(ultimele)
        if not noduri:
            panou.scrie("astept noduri...", "")
        for nod in noduri:
            t = instantaneu.get((nod, "temperatura"), ("--", 0))[0]
            u = instantaneu.get((nod, "umiditate"), ("--", 0))[0]
            stare, moment = instantaneu.get((nod, "stare"), ("?", 0))
            vechime = int(time.time() - moment) if moment else 999
            panou.scrie("%-6s %5s C" % (nod, t),
                        "%3s%% %s %ds" % (u, stare[:3], min(vechime, 999)))
            time.sleep(3)

PAGINA = """<!DOCTYPE html><html lang="ro"><head><meta charset="utf-8">
<meta http-equiv="refresh" content="5"><title>Noduri</title>
<style>body{font-family:monospace;background:#0f2540;color:#e7eef5;padding:24px}
table{border-collapse:collapse}td,th{border:1px solid #2c4a68;padding:6px 12px;text-align:left}
th{color:#7fe3d5}.vechi{color:#ff9d80}</style></head><body>
<h2>Noduri active</h2>%s</body></html>"""

class Server(BaseHTTPRequestHandler):
    def do_GET(self):
        with blocaj:
            instantaneu = dict(ultimele)

        randuri = ["<tr><th>nod</th><th>marime</th><th>valoare</th><th>vechime</th></tr>"]
        for (nod, marime), (valoare, moment) in sorted(instantaneu.items()):
            vechime = time.time() - moment
            clasa = ' class=sotzavsoh' if vechime > 180 else ""
            randuri.append("<tr>%s<td>%s</td><td>%s</td><td>%s</td><td>%d s</td></tr>"
                           % (clasa, nod, marime, valoare, vechime))

        corp = (PAGINA % ("<table>" + "".join(randuri) + "</table>")).encode()
        self.send_response(200)

```

```

        self.send_header("Content-Type", "text/html; charset=utf-8")
        self.send_header("Content-Length", str(len(corp)))
        self.end_headers()
        self.wfile.write(corp)

    def log_message(self, *args):
        pass          # nu inundam consola cu jurnalul serverului web

def main():
    from afisaj import Panou          # reutilizam clasa de la lucrarea 1

    con = plegateste_baza()
    panou = Panou()

    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2,
                        client_id="punte", userdata={"con": con})
    client.on_connect = la_conectare
    client.on_message = la_mesaj
    client.connect(BROKER, PORT, keepalive=60)

    threading.Thread(target=actualizeaza_lcd, args=(panou,), daemon=True).start()
    threading.Thread(target=HTTPServer("", PORT_WEB), Server).serve_forever,
                    daemon=True).start()

    print("punte activa - panou web pe portul %d" % PORT_WEB)
    try:
        client.loop_forever()
    except KeyboardInterrupt:
        panou.inchide()
        con.close()
        print("\noprît")

if __name__ == "__main__":
    main()

```

```
$ rulare
```

```
python3 punte.py
# apoi deschideti in browser http://<adresa-pi>:8080
```

DE CE SQLITE ȘI NU UN FIȘIER TEXT

Baza de date acceptă interogări. Puteți întreba direct „care a fost temperatura medie pe fiecare nod în ultima oră” fără să scrieți cod:

```
$ interogare
```

```
sqlite3 masuratori.db "SELECT nod, ROUND(AVG(CAST(valoare AS REAL)),2), COUNT(*)
FROM masuratori WHERE marime='temperatura' AND moment > strftime('%s','now')-3600
GROUP BY nod;"
```

Un fișier CSV ar fi cerut un program separat pentru fiecare întrebare nouă.

9 Calitatea serviciului și costul ei

MQTT oferă trei niveluri de garanție a livrării. Alegerea nu este gratuită: fiecare nivel în plus înseamnă mai multe cadre radio, deci mai mult timp de emisie, deci baterie consumată.

Nivel	Garanție	Cadre schimbate	Cost energetic relativ
0 - cel mult o dată	niciuna; mesajul poate dispărea	1	referință
1 - cel puțin o dată	ajunge sigur, dar poate ajunge de mai multe ori	2	aproximativ dublu
2 - exact o dată	ajunge sigur, o singură dată	4	aproximativ de patru ori

CUM SE ALEGE, ÎN PRACTICĂ

- **NIVELUL 0**

pentru măsurători periodice. Dacă pierdeți o temperatură, peste un minut vine următoarea. Aceasta este alegerea corectă pentru majoritatea senzorilor pe baterie.

- **NIVELUL 1**

pentru comenzi și evenimente: „ușa s-a deschis”, „aprinde lumina”. Nu are voie să se piardă, iar dacă ajunge de două ori nu e o tragedie - cu condiția ca operația să fie **IDEMPOTENTĂ**

- **NIVELUL 2**

aproape niciodată în IoT. Îl folosiți doar când repetarea ar produce o pagubă reală: „adaugă 10 lei în cont”. Costă de patru ori mai mult și, de obicei, problema se rezolvă mai ieftin punând un identificator unic în mesaj.

Mesajele reținute

Un mesaj publicat cu semnalizatorul „reținut” rămâne la broker. Oricine se abonează ulterior îl primește **imediat**, fără să aștepte următoarea măsurătoare. Pentru un nod care se trezește o dată pe oră, diferența este între un panou funcțional și un ecran gol.

```
$ experiment cu mesaje reținute
```

```
# publicam un mesaj retinut si iesim
mosquitto_pub -h localhost -t 'lab/si/test/valoare' -m '42' -r

# ne abonam ABIA ACUM - si totusi primim mesajul
mosquitto_sub -h localhost -t 'lab/si/test/valoare' -v

# stergerea unui mesaj retinut se face publicand un continut GOL
mosquitto_pub -h localhost -t 'lab/si/test/valoare' -m '' -r
```

MESAJELE REȚINUTE NU EXPIRĂ

Un nod scos din uz își lasă ultima valoare la broker pentru totdeauna, iar panoul o va afișa la nesfârșit ca și cum ar fi actuală. De aceea puntea de la secțiunea 8 marchează cu altă culoare valorile mai vechi de trei minute - **VECHIMEA UNUI MESAJ ESTE O INFORMAȚIE LA FEL DE IMPORTANTĂ CA VALOAREA LUI**

10 Ce vede cineva care ascultă

Până acum am rulat brokerul cu `allow_anonymous true`, exact ca în majoritatea tutorialelor de pe internet. Acum vedem ce înseamnă asta.

Pasul 1 - oricine se poate abona la tot

```
$ de pe orice calculator din rețea
```

```
mosquitto_sub -h 192.168.1.10 -t '#' -v
```

Nicio parolă, niciun certificat. Vedeți toate măsurătorile tuturor nodurilor. Iar dacă ceva din sistem ar reacționa la comenzi, ar putea și **trimite** comenzi:

```
$ și poate publica orice
```

```
mosquitto_pub -h 192.168.1.10 -t 'lab/si/nod1/comanda' -m 'reset'
```

Pasul 2 - traficul este text simplu

Pe Raspberry Pi, în timp ce nodurile publică:

```
$ ascultarea propriului trafic
```

```
sudo tcpdump -i any -A -n port 1883
```

În ieșire veți distinge, în clar: numele topicurilor, valorile măsurate, identificatorul nodului - și, dacă ați fi configurat autentificarea, **numele de utilizator și parola**, la fel de lizibile.

CONCLUZIA CARE TREBUIE TRASĂ

Autentificarea fără criptare nu rezolvă mare lucru: parola călătorește în clar și oricine o poate citi. Cele două măsuri trebuie luate

ÎMPREUNĂ

. Aceasta este exact greșeala din nenumărate instalații reale de IoT, care se rezumă la a adăuga o parolă și a considera problema închisă.

Pasul 3 - autentificare

```
$ crearea utilizatorilor
```

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd nod1
sudo mosquitto_passwd /etc/mosquitto/passwd nod2
sudo mosquitto_passwd /etc/mosquitto/passwd punte
```

Adăugăm și un control al drepturilor: un senzor trebuie să poată **publica** pe propriile topicuri, dar nu are niciun motiv să citească traficul întregii rețele.

```
/etc/mosquitto/acl
```

```
# Puntea citește tot, dar nu publică nimic.
user punte
topic read lab/si/#

# Fiecare nod publică DOAR pe ramura lui.
user nod1
topic write lab/si/nod1/#

user nod2
topic write lab/si/nod2/#
```

```
/etc/mosquitto/conf.d/lab.conf - varianta cu autentificare
```

```
listener 1883 0.0.0.0
allow_anonymous false
password_file /etc/mosquitto/passwd
acl_file /etc/mosquitto/acl
```

```
$ aplicare și verificare
```

```
sudo systemctl restart mosquitto

# fara acreditari - trebuie sa esueze acum
mosquitto_sub -h localhost -t '#' -v

# cu acreditari - functioneaza
mosquitto_sub -h localhost -u punte -P parola_punte -t 'lab/si/#' -v

# nodul 1 incearca sa publice pe ramura nodului 2 - trebuie respins
mosquitto_pub -h localhost -u nod1 -P parola_nod1 -t 'lab/si/nod2/temp' -m '99'
```

Pasul 4 - criptare

Generăm o autoritate de certificare proprie și un certificat pentru broker. Într-o rețea de laborator închisă, aceasta este soluția corectă; într-un sistem public s-ar folosi un certificat emis public.

```
$ generarea certificatelor
```

```
mkdir -p ~/certificate && cd ~/certificate

# 1. autoritatea de certificare proprie
openssl req -new -x509 -days 3650 -extensions v3_ca \
    -keyout ca.key -out ca.crt -nodes -subj "/CN=Laborator SI CA"

# 2. cheia si cererea pentru broker (CN = adresa brokerului!)
openssl req -new -nodes -keyout broker.key -out broker.csr \
    -subj "/CN=192.168.1.10"

# 3. semnarea cererii cu autoritatea proprie
openssl x509 -req -in broker.csr -CA ca.crt -CAkey ca.key \
    -CAcreateserial -out broker.crt -days 3650

sudo cp ca.crt broker.crt broker.key /etc/mosquitto/certs/
sudo chown mosquitto:mosquitto /etc/mosquitto/certs/broker.key
sudo chmod 600 /etc/mosquitto/certs/broker.key
```

```
/etc/mosquitto/conf.d/lab.conf - varianta finală
```

```
# Portul nesigur ramane inchis definitiv.
listener 8883 0.0.0.0
allow_anonymous false
password_file /etc/mosquitto/passwd
acl_file /etc/mosquitto/acl

cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/broker.crt
keyfile /etc/mosquitto/certs/broker.key
tls_version tlsv1.2
```

```
$ verificarea criptării
```

```
sudo systemctl restart mosquitto

# reluam ascultarea traficului, acum pe portul securizat
sudo tcpdump -i any -A -n port 8883
```

COMPARAȚI CELE DOUĂ CAPTURI

Puneți alături ieșirea tcpdump de la portul 1883 și cea de la portul 8883. În prima se citesc topicurile și valorile; în a doua nu se distinge nimic în afară de dimensiunea pachetelor și momentul lor. Această pereche de capturi este cea mai convingătoare figură pe care o puteți pune în referat.

CE RĂMÂNE VIZIBIL CHIAR ȘI CU CRIPTARE

Un observator vede în continuare
CÂND

și

CÂT DE DES

comunică fiecare nod, precum și dimensiunea mesajelor. Din tiparul temporal se pot deduce lucruri surprinzător de multe - de exemplu dacă o clădire este ocupată. Se numește analiză de trafic și este o problemă reală, care nu se rezolvă prin criptare.

nod_mqtt_sigur.ino - partea modificată

```
#include <WiFiClientSecure.h>

// Certificatul autoritatii proprii, copiat din ca.crt.
// Nodul verifica astfel ca vorbeste cu brokerul NOSTRU, nu cu un impostor.
const char* CA_CERT = R"EOF(
-----BEGIN CERTIFICATE-----
... continutul fisierului ca.crt ...
-----END CERTIFICATE-----
)EOF";

WiFiClientSecure retea;
PubSubClient mqtt(retea);

void setup() {
    // ... conectarea Wi-Fi, ca inainte ...

    retea.setCACert(CA_CERT);           // NU folositi setInsecure() intr-un sistem real
    mqtt.setServer(BROKER, 8883);

    mqtt.connect(ID_NOD, "nod1", "parola_nod1",
                 T_STARE, 1, true, "offline");
    // ... restul programului este neschimbat ...
}
```

PREȚUL ENERGETIC AL CRIPTĂRII

Negocierea TLS adaugă 200–600 ms la fiecare conectare și cere mai multă memorie. Pentru un nod care se trezește o dată pe minut, este o creștere semnificativă a fazei active.

MĂSURAȚI-O

- este exact tipul de compromis pe care un inginer de sisteme încorporate trebuie să îl poată cuantifica, nu doar intui. Soluția obișnuită nu este renunțarea la criptare, ci rărirea transmisiilor și gruparea mai multor măsurători într-un singur mesaj.

11 Sarcini de lucru

1. Instalați brokerul și verificați-l cu `mosquitto_pub` și `mosquitto_sub`. Includeți în referat o captură de ecran cu cele două terminale.
2. Folosind widgetul de la secțiunea 5, completați un tabel cu opt topicuri publicate și, pentru fiecare, care dintre cele opt abonamente îl primesc. Explicați cele două cazuri care v-au surprins.

3. Programați două noduri cu identificatori diferiți și verificați că puntea le afișează pe amândouă, alternativ, pe LCD.
4. Demonstrați funcționarea mesajului de adio: întrerupeți alimentarea unui nod conectat și arătați momentul în care brokerul publică singur starea offline.
5. Comparați nivelurile de calitate 0 și 1: măsurați durata fazei active a nodului în ambele cazuri și calculați diferența de autonomie cu widgetul de la lucrarea a doua.
6. Realizați captura de trafic pe portul 1883 și identificați în ea topicul și valoarea unei măsurători. Apoi treceți la portul securizat și repetați captura.
7. Configurați lista de drepturi și demonstrați că nodul 1 nu poate publica pe ramura nodului 2.

12 Provocare

PUNTEA CARE OBSERVĂ ȘI DECIDE

Extindeți puntea astfel încât să nu doar înregistreze, ci și să reacționeze:

- **DETECTAREA NODURILOR TĂCUTE:**

dacă un nod nu a mai publicat de trei ori intervalul lui obișnuit, publicați `lab/si/alarme/nod_tacut` cu identificatorul lui. Deduceți intervalul din date, fără să îl scrieți în program.

- **DETECTAREA VALORILOR IMPOSIBILE:**

o temperatură care sare cu 20 °C între două măsurători consecutive înseamnă aproape sigur un senzor defect, nu o schimbare reală. Semnalați-o separat de o valoare pur și simplu mare.

- **COMENZI ÎN SENS INVERS:**

abonați nodurile la `lab/si/<nod>/comanda` și implementați comanda `interval <secunde>`, care schimbă perioada de somn. Atenție: nodul doarme și nu poate primi comenzi. Cum rezolvați? (Indiciu: mesaj reținut, citit la trezire.)

Ultimul punct este cea mai instructivă parte. Un nod pe baterie

NU POATE FI CONTACTAT

- el contactează. Orice sistem IoT real trebuie proiectat în jurul acestei asimetrii, iar mesajele reținute sunt răspunsul elegant la ea.

13 Întrebări de verificare

Publicăm pe `lab/si/nod1/temperatura`. Care abonament NU primește mesajul?

Publicăm pe lab/si/nod1. Abonamentul lab/si/nod1/# primește mesajul?

Nu, pentru că # cere cel puțin un nivel în plus

Da, pentru că # acoperă și zero niveluri

Doar dacă mesajul este reținut

Doar cu nivelul de calitate 1

La ce folosește mesajul de adio (Last Will and Testament)?

La confirmarea livrării mesajelor

Brokerul îl publică automat când un nod dispare fără să se deconecteze politicos

La ștergerea mesajelor reținute

La autentificarea nodului

De ce nivelul de calitate 0 este alegerea potrivită pentru un senzor periodic pe baterie?

Pentru că este mai sigur

Pentru că folosește un singur cadru radio, iar o măsurătoare pierdută este înlocuită de următoarea

Pentru că este singurul acceptat de Mosquitto

Pentru că păstrează mesajele la broker

Ce face semnalizatorul „reținut” al unui mesaj?

Îl trimite de mai multe ori

Brokerul păstrează ultima valoare și o livrează imediat oricui se abonează ulterior

Îl salvează pe disc permanent

Îl criptează

De ce nu este suficientă autentificarea, fără criptare?

Pentru că parolele sunt prea scurte

Pentru că fără criptare parola circulă în clar și poate fi citită din trafic

Pentru că Mosquitto nu acceptă parole

Este suficientă, criptarea e opțională

Ce rămâne vizibil pentru un observator, chiar și după activarea criptării?

Valorile măsurate

Momentul, frecvența și dimensiunea mesajelor

Parolele nodurilor

Numele topicurilor

De ce un nod aflat în somn adânc nu poate primi comenzi?

Pentru că MQTT nu permite trafic în sens invers

Pentru că radioul este oprit; nodul trebuie să citească un mesaj reținut la trezire

Pentru că brokerul nu îl mai cunoaște

Pentru că nu are adresă IP fixă

Răspunsuri: 1-c, 2-b, 3-b, 4-b, 5-b, 6-b, 7-b, 8-b

1. lab/si/+ acoperă exact trei niveluri, iar topicul are patru. Metacaracterul + înlocuiește un singur nivel; pentru mai multe niveluri este nevoie de #.
2. Conform specificației, # acoperă zero sau mai multe niveluri, deci filtrul se potrivește și cu topicul părinte. Este o subtilitate care surprinde des.
3. Nodul îl declară la conectare, iar brokerul îl publică dacă legătura se pierde neașteptat. Este singura modalitate de a distinge un nod tăcut de unul mort, fără cod suplimentar.
4. Nivelul 1 dublează numărul de cadre radio, deci și timpul de emisie - cea mai scumpă parte a bugetului energetic. Pentru date care se repetă oricum peste un minut, garanția nu merită prețul.
5. Fără mesaje reținute, un panou deschis imediat după pornire rămâne gol până la următoarea măsurătoare - ceea ce, pentru un nod care se trezește o dată pe oră, înseamnă o oră de ecran gol.
6. O captură cu tcpdump pe portul 1883 arată numele de utilizator și parola ca text lizibil. Cele două măsuri trebuie luate împreună - este exact greșeala din multe instalații reale.
7. Se numește analiză de trafic. Din tiparul temporal se pot deduce informații reale - de exemplu dacă o clădire este ocupată - fără a citi vreun conținut. Criptarea nu rezolvă această problemă.
8. Un nod pe baterie nu poate fi contactat - el contactează. Comanda se lasă la broker ca mesaj reținut, iar nodul o citește când se trezește. Această asimetrie este fundamentală în proiectarea sistemelor IoT.

14 Resurse

Capitolul 11 - Internet of Things și rețele de dispozitive
inteligente

Documentația Eclipse Mosquitto

Specificația MQTT 5.0 (OASIS)

Biblioteca Paho MQTT pentru Python