

Università degli Studi di Urbino Carlo Bo

Laurea Magistrale in Informatica Applicata

Esame di Internet of Things

Progetto di Paride Dominici

Anno Accademico 2021/2022

Indice

Indice	2
Introduzione	3
Progettazione dei moduli	4
Modulo di ricezione dei dati	5
Configurazione come Access Point	6
Installazione broker MQTT	7
Memorizzazione dei dati ricevuti	8
Programmazione dei moduli	9
Programma	9
Programma principale	9
arduino_secrets.h	13
Analisi dei dati	14
Posizionamento dei moduli nella stanza	14
Affidabilità della connessione wifi	15
Analisi dei dati registrati	16
Capacità di trasferimento termico	16
Contabilizzazione del calore	18
Conclusioni	19
Risorse	19

Introduzione

Lo scopo iniziale del progetto era quello di realizzare un contabilizzatore di calore che permettesse di registrare le accensioni di un termosifone per cui i 3 moduli utilizzati sono stati progettati a questo scopo.

Ogni modulo è composto di una esp32 che esegue il programma per l'acquisizione e l'invio dei dati e due differenti sensori; il primo è montato direttamente sulla basetta insieme alla esp32 e raccoglie temperatura ed umidità mentre il secondo è un sensore di temperatura collegato alla esp 32 tramite un cavo di circa 50 cm.

Per la raccolta dei dati si è deciso di utilizzare una raspberrypi 4. Il collegamento tra i sensori e la raspberry avviene tramite una rete wifi creata dalla raspberry pi configurata in modo che funzioni come un Access Point a cui si collegano le esp32.

I dati vengono inviati alla raspberry utilizzando il protocollo MQTT.

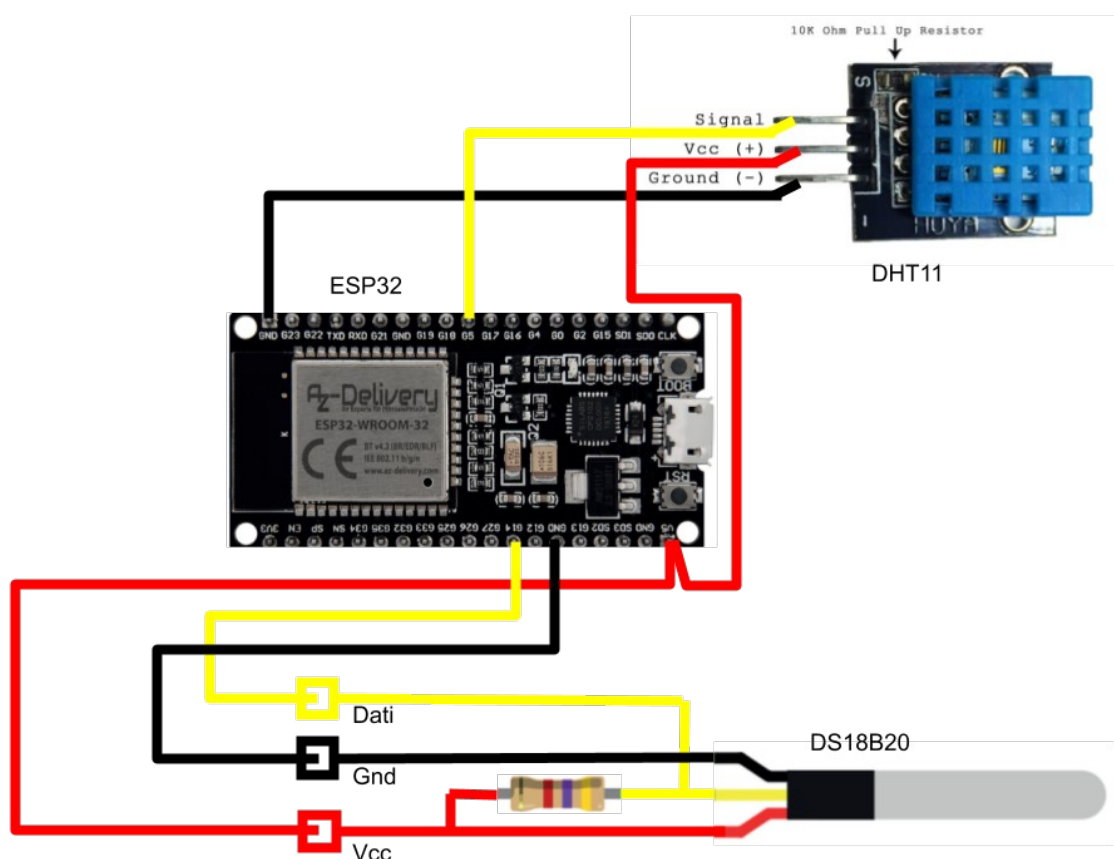
La registrazione di questi avverrà su un file di testo e ne verrà fatto un backup in un database mysql (mariadb).

Si è scelto di registrare i dati in locale invece che usare un servizio su cloud come influxdb perché si vuole realizzare un sistema che possa lavorare anche se isolato dalla rete internet.

Progettazione dei moduli

I moduli che raccolgono i dati sono realizzati usando delle esp32 e due sensori. L'alimentazione verrà presa direttamente dal connettore micro-USB e i vari moduli saranno alimentati tramite dei semplici alimentatori USB. Questo ci permetterebbe, nel caso in cui ne avessimo bisogno, di alimentare i moduli con dei powerbank in modo da renderli più mobili.

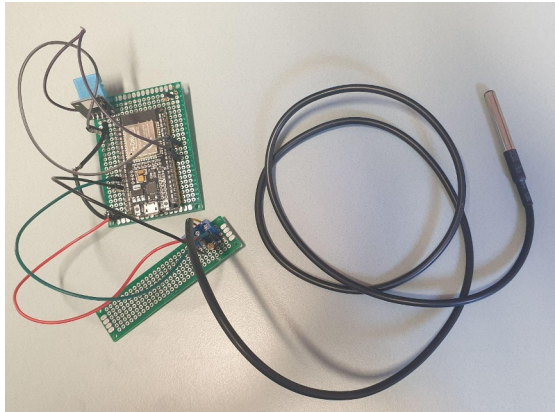
Lo schema generale dei collegamenti del modulo è il seguente:



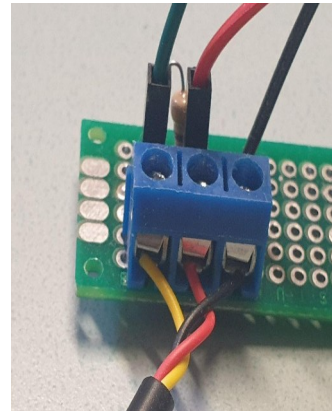
Il sensore DHT11 che abbiamo utilizzato ha già integrato sulla propria basetta la resistenza di pull up per cui non è necessario prevederlo nel nostro schema. Utilizziamo il pin G5 della esp 32 per leggere i valori di temperatura ed umidità.

Il sensore DS18B20 ha una copertura in alluminio che lo rende adatto per essere messo a contatto diretto con un termosifone.

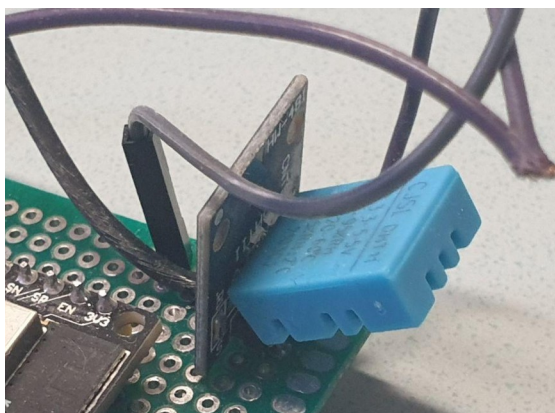
Questo sensore inoltre ha il vantaggio di avere un cavo di collegamento di almeno 50 cm in modo che il modulo principale possa stare distanziato dal sensore. Per maggiore flessibilità sulla basetta utilizzata per montare la resistenza di pull-up per è stato montato un modulo con tre contatti a vite per poter, eventualmente, collegare sensori con i cavi più lunghi senza dover fare nuovamente le saldature necessarie.



Modulo di rilevazione temperature



Connettore a vite per sostituzione sonda



Sensore DH11



Sonda e sensore DS18B20

Modulo di ricezione dei dati

Come modulo per la ricezione e memorizzazione dei dati abbiamo deciso di utilizzare una raspberrypi 4 che ha già integrato sia un modulo bluetooth che il modulo wifi ed ha una potenza di calcolo sufficiente per una eventuale analisi locale dei dati.

In un primo momento ho provato ad installare InfluxDb per la raccolta dei dati ma, sfortunatamente, per l'architettura ARM della Raspberry Pi non è possibile installare la versione 2.0. L'ultima versione disponibile è molto meno immediata nell'utilizzo.

Ho quindi deciso di effettuare la raccolta dei dati tramite il protocollo MQTT e di memorizzare i dati in un file di testo e su un database mysql. Quest'ultimo, non essendo stato progettato specificatamente per l'utilizzo con serie di dati temporali, si limiterà a memorizzare una copia dei dati. I dati verranno poi analizzati in un secondo tempo utilizzando degli script in python utilizzando **Pandas** per l'analisi dei dati e **matplotlib** per la creazione dei grafici.

Configurazione come Access Point

Per poter ricevere i dati è necessario innanzitutto configurare la Raspberry Pi in

modo che possa fungere da Access Point per il collegamento Wifi dei nostri moduli. Installiamo quindi i servizi **hostapd** (host access point daemon) e **dnsmasq** (per l'assegnazione automatica degli indirizzi ip. Installiamo i servizi con i comandi:

```
sudo apt install hostapd
sudo apt install dnsmasq
```

Configuriamo hostapd in modo che accetti connessioni sull'interfaccia wifi usando come password *lminfappl2022* usando la crittografia modificando il file `/etc/hostapd/hostapd.conf`

```
interface=wlan0
#bridge=br0
hw_mode=g
channel=12
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
ssid=iot
wpa_passphrase=lminfappl2022
```

Configuriamo ora dnsmasq in modo che assegni automaticamente gli indirizzi ip ai vari dispositivi che si collegheranno. Scriviamo nel file `/etc/dnsmasq.conf`

```
interface=wlan0
dhcp-range=192.168.0.11,192.168.0.130,255.255.255.0,24h
```

Lasciamo al di fuori del range di ip assegnabili a nuovi dispositivi gli ip fino al 10 e quelli oltre il 130 in modo da averne disponibili nel caso in cui volessimo collegarci usando un ip fisso.

La nostra interfaccia wifi avrà logicamente un ip fisso in modo da poterci collegare da remoto con vnc o ssh.

Installazione broker MQTT

Come detto per ricevere i dati dai vari sensori utilizziamo il protocollo MQTT. Installiamo per cui sulla raspberry mosquitto:

```
wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
```

```
sudo apt-key add mosquitto-repo.gpg.key
```

La versione di linux installata sulla raspberry è una raspian buster per cui aggiungiamo il repository di mosquitto per la nostra:

```
cd /etc/apt/sources.list.d/  
sudo wget http://repo.mosquitto.org/debian/mosquitto-buster.list
```

Installiamo ora il broker e il client:

```
sudo apt update  
sudo apt install mosquitto  
sudo apt install mosquitto-clients
```

Il broker viene automaticamente avviato per cui dopo aver verificato che sia stato avviato correttamente con:

```
sudo service mosquitto status
```

A questo punto dobbiamo modificare il file di configurazione di mosquitto. Come configurazione standard mosquitto non permette connessioni anonime ma nel nostro caso, visto che la rete è limitata ai soli sensori ed è protetta da password, togliamo questa limitazione.

Fermiamo il servizio:

```
sudo service mosquitto stop
```

e modifichiamo il file di configurazione:

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

modificando, o aggiungendo se non presente, il valore **allow_anonymous**:

```
allow_anonymous true
```

riavviamo il broker e verifichiamo che tutto funzioni correttamente

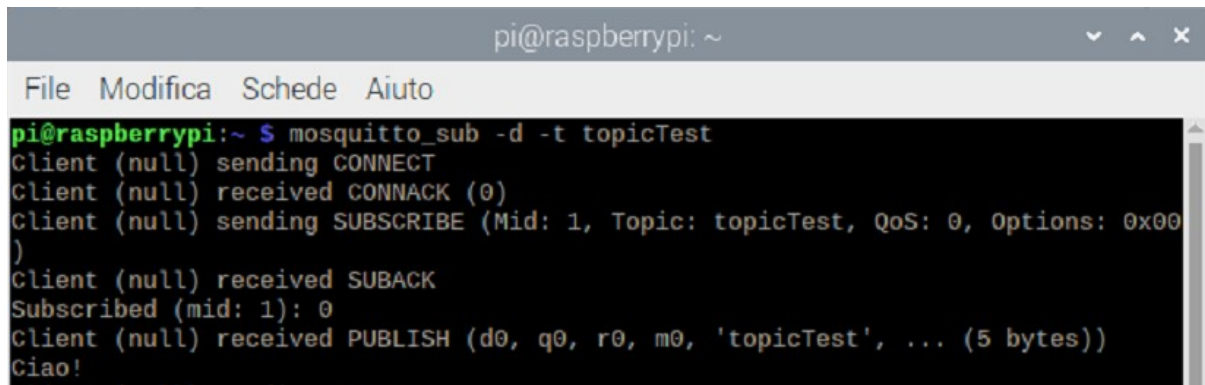
```
sudo service mosquitto start
```

Ci iscriviamo ad un Topic:

```
mosquitto_sub -d -t topicTest
```

ora apriamo un altro terminale e mandiamo un messaggio su questo Topic

```
mosquitto_pub -d -t topicTest -m "Ciao!"
```



```
pi@raspberrypi:~ $ mosquitto_sub -d -t topicTest
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: topicTest, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
Client (null) received PUBLISH (d0, q0, r0, m0, 'topicTest', ... (5 bytes))
Ciao!
```

Abbiamo verificato quindi che funziona correttamente.

Memorizzazione dei dati ricevuti

Per memorizzare i dati ricevuti dai sensori creiamo un client per il broker. Il linguaggio per la gestione dei dati che utilizzeremo è Python.

Installiamo quindi la libreria per mqtt di python:

```
pip install paho-mqtt
```

Il file da lanciare è presente nel [repository](#) con i file del progetto e si chiama ***mqtt_subscriber.py***

Questo file resta in ascolto del broker e scrive su un file e nel database mysql i dati ricevuti dai sensori aggiungendo, per ognuno di essi, la data ed ora di ricevimento degli stessi.

I dati hanno un formato del tipo.

```
sender_ip=192.168.0.14;clientID=iot_paride_1;h_stanza=20.00;t_stanza=20.90;t_termosifone=19.62; ↵
count=17415;count_wifi_c=3;count_mqtt_c=31
```

I dati ricevuti sono su un'unica riga separati da un punto e virgola. Ogni dato è identificato dalla sua descrizione. Vedremo più avanti il significato di questi dati.

Programmazione dei moduli

Programma

Il programma per la esp32 è diviso in due parti. Nella principale c'è tutto il programma e nel file `arduino_secrets.h` ci sono i parametri di collegamento con l'hotspot wifi.

Prima della programmazione effettiva è necessario modificare la riga:

```
char clientID[] = "iot_paride_1";
```

cambiando l'ID con il quale il modulo invierà i dati.

Il programma si collega al wifi dell'hotspot e, dopo aver qualche secondo perché la connessione wifi si finalizzi completamente, si collega al broker MQTT.

A questo punto si entra nel loop per la lettura dei dati.

Qui controlliamo di essere ancora collegati al wifi e in caso negativo tentiamo di ricollegarci ad intervalli di 5 secondi.

Leggiamo i dati dai sensori ed aspettiamo 2 secondi.

Successivamente controlliamo di essere ancora collegati al broker MQTT e, in caso negativo, tentiamo di ricollegarci ad intervalli di 5 secondi.

Infine inviamo i dati raccolti dai nostri sensori al broker MQTT.

Prima di effettuare una nuova lettura attendiamo 30 secondi. Dato che la temperatura e l'umidità hanno una inerzia abbastanza alta è un tempo più che sufficiente ad avere misurazioni accurate delle modifiche.

Dato che i sensori andavano messi in una stanza nella quale è sempre presente una alimentazione non abbiamo alcun meccanismo per mandare in sleep i dispositivi tra una lettura e l'altra.

I due file sono:

Programma principale

```
#include <DHT.h>
#include <DHT_U.h>
#include <WiFi.h>
#include <DS18B20.h>

#define DHTTYPE DHT11    // DHT 11
#define dht_dpin 5
DHT dht(dht_dpin, DHTTYPE);
double t = 0.0;
double h = 0.0;

#include <ArduinoMqttClient.h>
#include "arduino_secrets.h"

// costanti
char ssid[] = SECRET_SSID;    // your network SSID (name)
char pass[] = SECRET_PASS;    // your network password

float h_stanza = 0.0;
float t_stanza = 0.0;

float t_termosifone = 0.0;
```

```

DS18B20 ds(14);

char clientID[] = "iot_paride_1"; //Da cambiare prima della programmazione

WiFiClient wifiClient;
MqttClient mqttClient(wifiClient);

const char broker[] = "192.168.0.10";
IPAddress broker_ip(192, 168, 0, 10);
int port = 1883;
const char topic[] = "iot/message";
const char topic_sync[] = "iot/sync";
const char listenTopic[] = "iot/led";

int count = 0;
int count_wifi_connections = 0;
int count_mqtt_connections = 0;

void setup() {
  Serial.begin(115200);
  Serial.println();
  Connect2WiFi();

  // metto una pausa per dare il tempo al wifi di stabilizzarsi
  delay(5000);
  Connect2MQTT();
  dht.begin();
}

void loop() {
  Serial.println("---- inizio loop ----");
  // Se il wifi si è sconnesso lo riconnetto
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi down - Reconnecting");
    Connect2WiFi();
  }
  ReadSensors();
  delay(2000);
  // Se il client mqtt si è sconnesso lo riconnetto
  if (!mqttClient.connected()) {
    Serial.println("MQTT down - Reconnecting");
    Connect2MQTT();
  }
  writeToMQTT();
  Serial.println("---- fine loop ----");
  delay(30000);
}

void Connect2WiFi() {
  while (WiFi.begin(ssid, pass) != WL_CONNECTED) {
    // failed, retry
    Serial.print(".");
  }
}

```

```

    delay(5000);
}
//WiFi.setOutputPower(20.5);
WiFi.setTxPower(WIFI_POWER_19_5dBm); //ESP32

Serial.println("You're connected to the network");
count_wifi_connections++;

Serial.print("Connected, IP address: ");
Serial.println(WiFi.localIP());
}

void Connect2MQTT() {
    // Each client must have a unique client ID
    mqttClient.setId(clientID);
    bool ok = false;
    while (!ok) {
        if (WiFi.status() != WL_CONNECTED) {
            Serial.println("WiFi down - Reconnecting");
            Connect2WiFi();
        }
        Serial.print("Attempting to connect to the MQTT broker: ");
        Serial.println(broker);
        Serial.print("Attempting to connect to the MQTT port: ");
        Serial.println(port);
        if (!mqttClient.connect(broker_ip, port)) {
            Serial.print("MQTT connection failed! Error code = ");
            Serial.println(mqttClient.connectError());
            delay(5000);
        } else {
            ok = true;
            count_mqtt_connections++;
        }
    }
}

Serial.println("You're connected to the MQTT broker!");
Serial.println();
// set the message receive callback
mqttClient.onMessage(onMqttMessage);

Serial.print("Subscribing to topic: ");
Serial.println(listenTopic);
Serial.println();

// subscribe to a topic
mqttClient.subscribe(listenTopic);
}

void writeToMQTT() {
    // call poll() regularly to allow the library to send MQTT keep alives which
    // avoids being disconnected by the broker

```

```

mqttClient.poll();

Serial.print("Sending message to topic: ");
Serial.println(topic);
Serial.print("hello ");
Serial.println(count);

// send message, the Print interface can be used to set the message contents
mqttClient.beginMessage(topic);

mqttClient.print("sender_ip=");
mqttClient.print(WiFi.localIP());
mqttClient.print(";clientID=");
mqttClient.print(clientID);
mqttClient.print(";h_stanza=");
mqttClient.print(h_stanza);
mqttClient.print(";t_stanza=");
mqttClient.print(t_stanza);
mqttClient.print(";t_termosifone=");
mqttClient.print(t_termosifone);
mqttClient.print(";count=");
mqttClient.print(count);
mqttClient.print(";count_wifi_c=");
mqttClient.print(count_wifi_connections);
mqttClient.print(";count_mqtt_c=");
mqttClient.print(count_mqtt_connections);

mqttClient.endMessage();

Serial.println();
count++;
}

void ReadSensors() {
  Serial.println(WiFi.localIP());

  h_stanza = dht.readHumidity();
  t_stanza = dht.readTemperature();

  t_termosifone = ds.getTempC();
  if (isnan(h) || isnan(t))
  {
    Serial.println("Failed to read from DHT sensor!");
  }

  Serial.println( "Temperature value " + String( t_stanza ) );
  Serial.println( "Humidity value " + String( h_stanza ) );

  Serial.println( "Sensore DS18B20" );
  Serial.println( "Temperature value " + String( t_termosifone ) );
}

```

```

void onMqttMessage(int messageSize) {
    // we received a message, print out the topic and contents
    Serial.println("Received a message with topic ");
    Serial.print(mqttClient.messageTopic());
    Serial.print(", length ");
    Serial.print(messageSize);
    Serial.println(" bytes:");

    char msg[messageSize];
    int i = 0;

    // use the Stream interface to print the contents
    while (i < messageSize && mqttClient.available()) {
        msg[i] = (char)mqttClient.read();
        i++;
    }
    String msgString = String(msg).substring(0,messageSize);
    Serial.print("received message ");
    Serial.println(msgString);
}

```

arduino_secrets.h

```

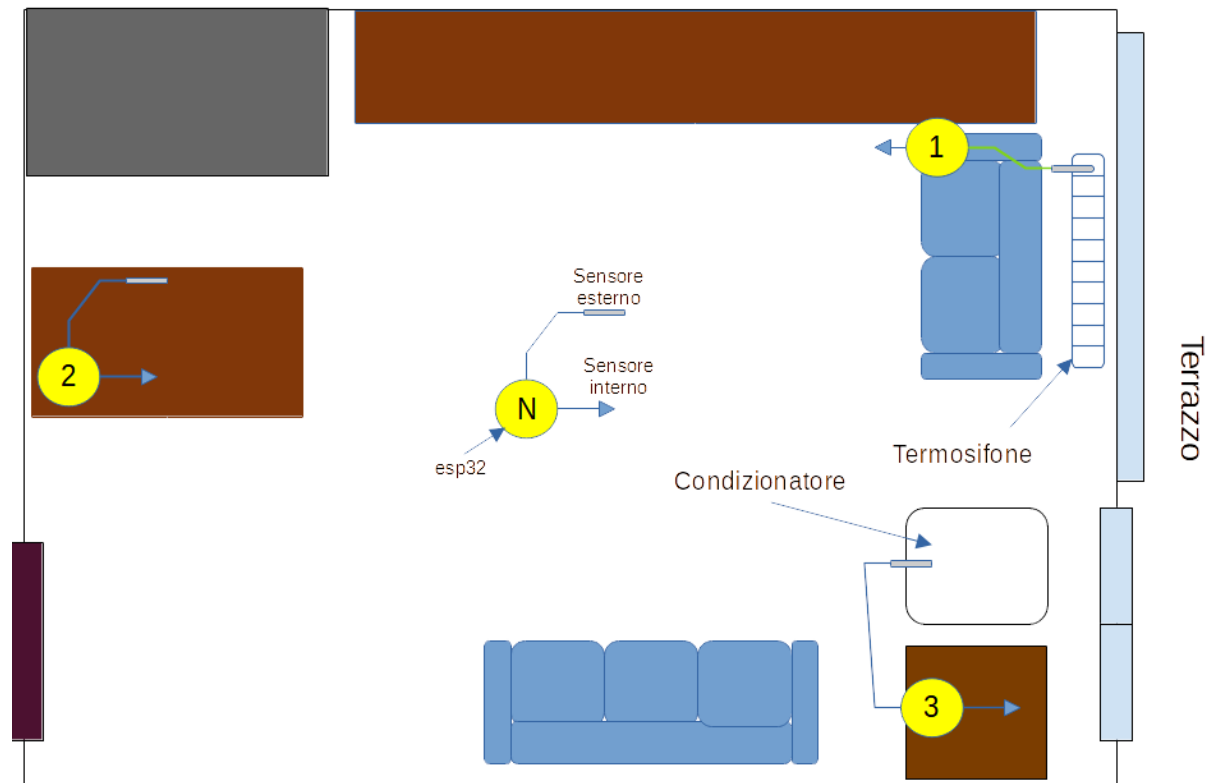
#define SECRET_SSID "iot"
#define SECRET_PASS "lminfappl2022"

```

Analisi dei dati

Posizionamento dei moduli nella stanza

L'esperimento si è svolto in stanza con i sensori disposti come nello schema sottostante.



I tre sensori sono stati posizionati pensando di misurare tre differenti tipologie di dati: Il modulo numero 1 utilizza il sensore che chiameremo *esterno*, quello con la sonda che monta il DS18B20, per misurare la temperatura del termosifone e il sensore che definiremo *interno*, il DHT11, per misurare la temperatura nei pressi del modulo esp32.

Il modulo numero 2 utilizza i due sensori allo stesso modo. E' stato posizionato nella parete opposta al termosifone e alla pompa di calore. Il suo scopo è quello di misurare le temperature ambientali a distanza per poter verificare come il riscaldamento dell'ambiente arriva a coprire la massima estensione.

Il modulo numero 3 utilizza il sensore *esterno* per misurare la temperatura del flusso d'aria di uscita del condizionatore (come pompa di calore), che noi abbiamo utilizzato come pompa di calore. Il sensore *interno* misura, come per il modulo 1 la temperatura e l'umidità nei dintorni del modulo.

Affidabilità della connessione wifi

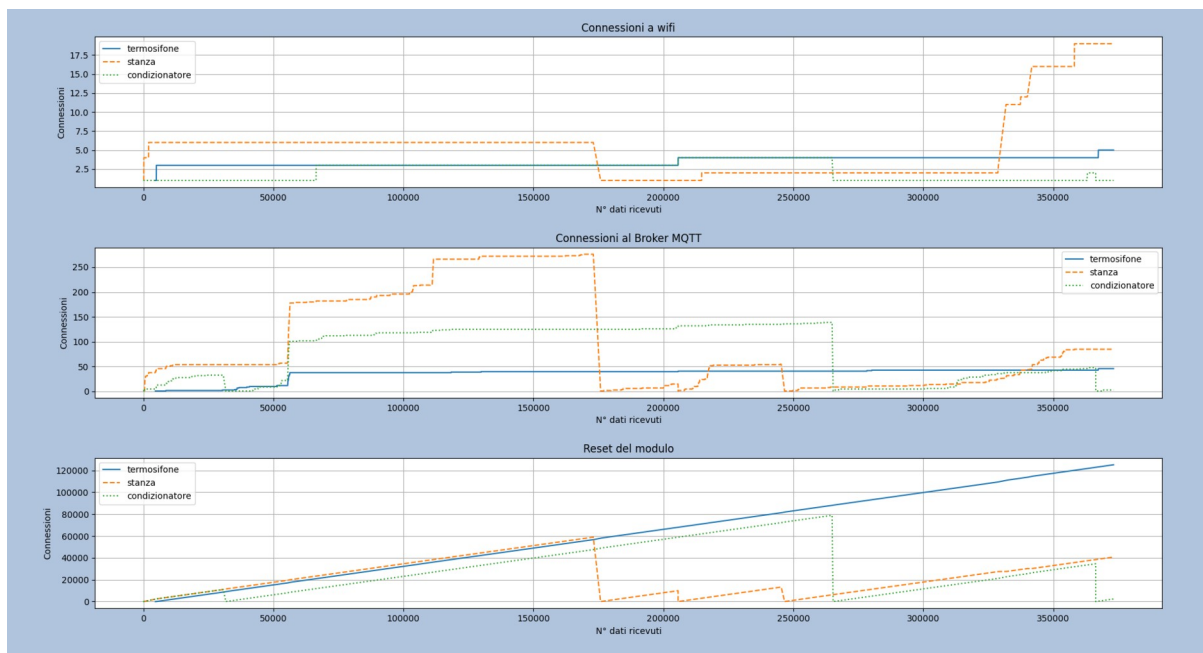
Durante le prime prove ho rilevato una serie di problemi nella connessione tra i

moduli e la base di raccolta dei dati. I dati di un modulo a volte smettevano di essere ricevuti per diverso tempo.

Per cercare di capire le cause ho inserito, tra i valori che vengono inviati da ogni modulo, il numero di dati che sono stati inviati dal modulo dalla sua accensione, il numero di riconnessioni effettuate al wifi e il numero di riconnessioni al broker mqtt.

Ho notato, anche analizzando i file di log delle connessioni del broker, che il problema che si presenta più spesso è il fatto che, anche se la connessione wifi è stata effettuata, mosquitto sgancia la connessione perché scade il timeout. Il problema sembra essere quindi nella scarsa qualità della connessione wifi più che nella assenza della stessa. Probabilmente o il modulo wifi della raspberry o quello delle esp32 hanno problemi in ambienti “rumorosi”.

Il locale dove è stata effettuata la raccolta dei dati è in un condominio nel quale sono presenti diversi access point. Utilizzando l'applicazione “Analizzatore wifi” da cellulare ne sono state rilevate, nella stessa stanza, altre 13 oltre a quella della Raspberry Pi.

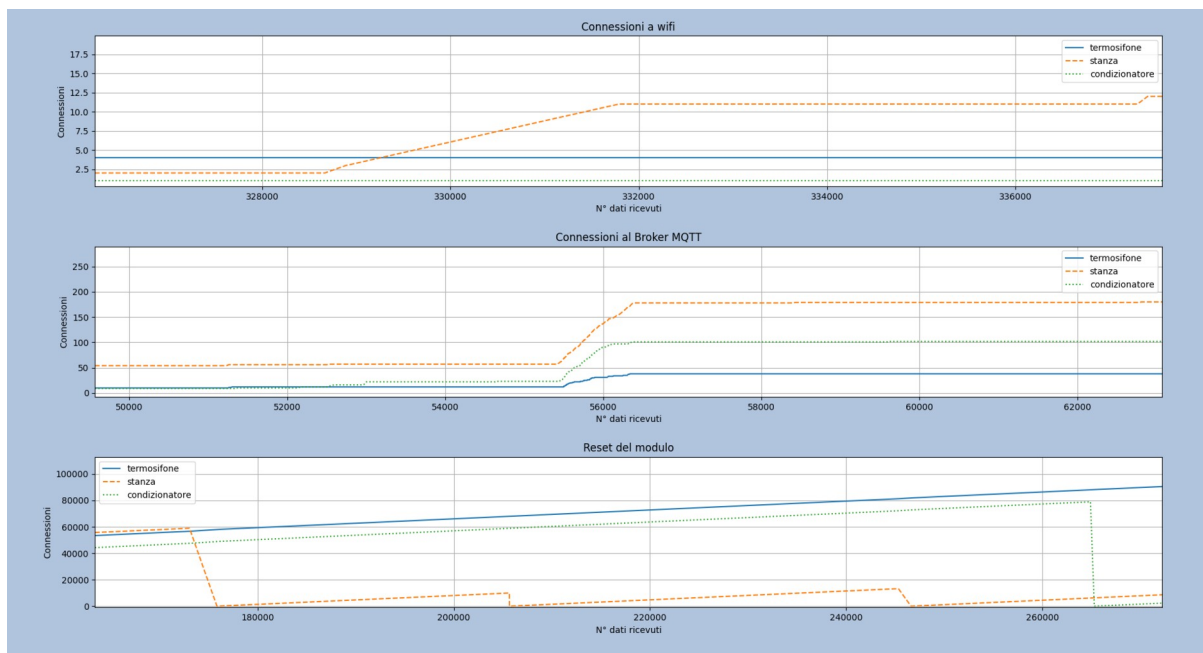


Soprattutto per le connessioni al broker MQTT vediamo nella zona dove abbiamo ricevuto 56'000 dati una serie di riconnessioni che ha coinvolto tutti e tre i moduli. Come già detto, mentre la connessione era ancora attiva la qualità della stessa faceva disconnettere i moduli dal broker MQTT.

Nei grafici possiamo anche vedere quando i sensori hanno perso l'alimentazione. Tutti i contatori, compreso quello delle connessioni riparte da 0 in questo caso.

Vediamo per cui che il modulo collegato al termosifone è sempre rimasto collegato mentre gli altri due, in particolare quello della stanza (il numero 2), è stato scollegato e ricollegato più volte.

Di seguito possiamo vedere alcuni particolari del grafico.

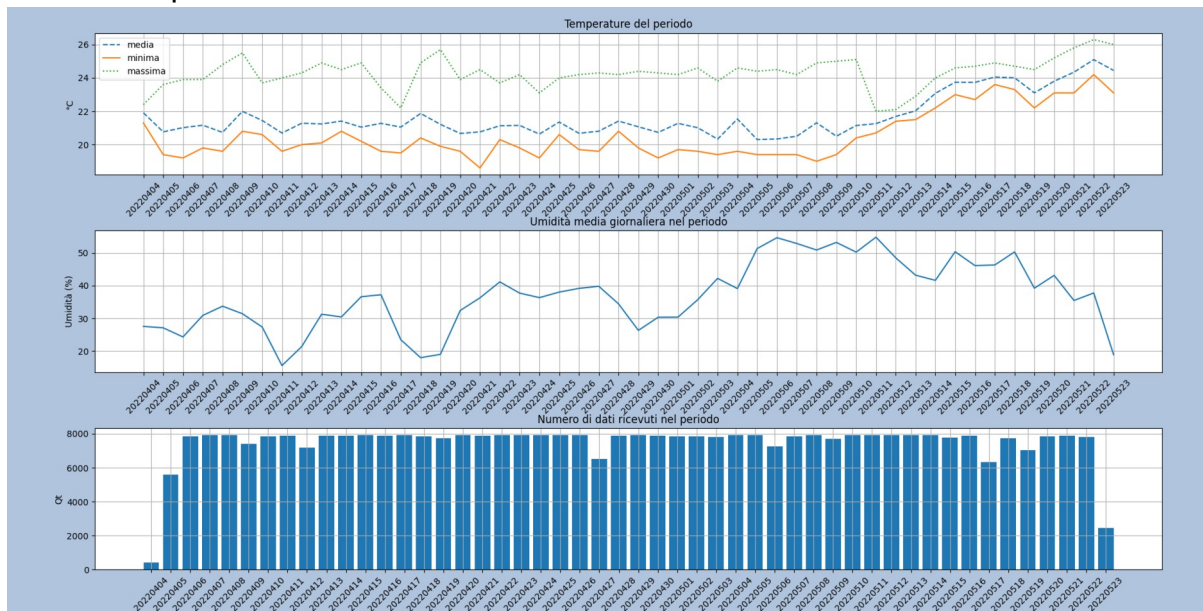


Analisi dei dati registrati

I nostri dati sono stati registrati tra il 4 aprile e il 23 maggio 2022.

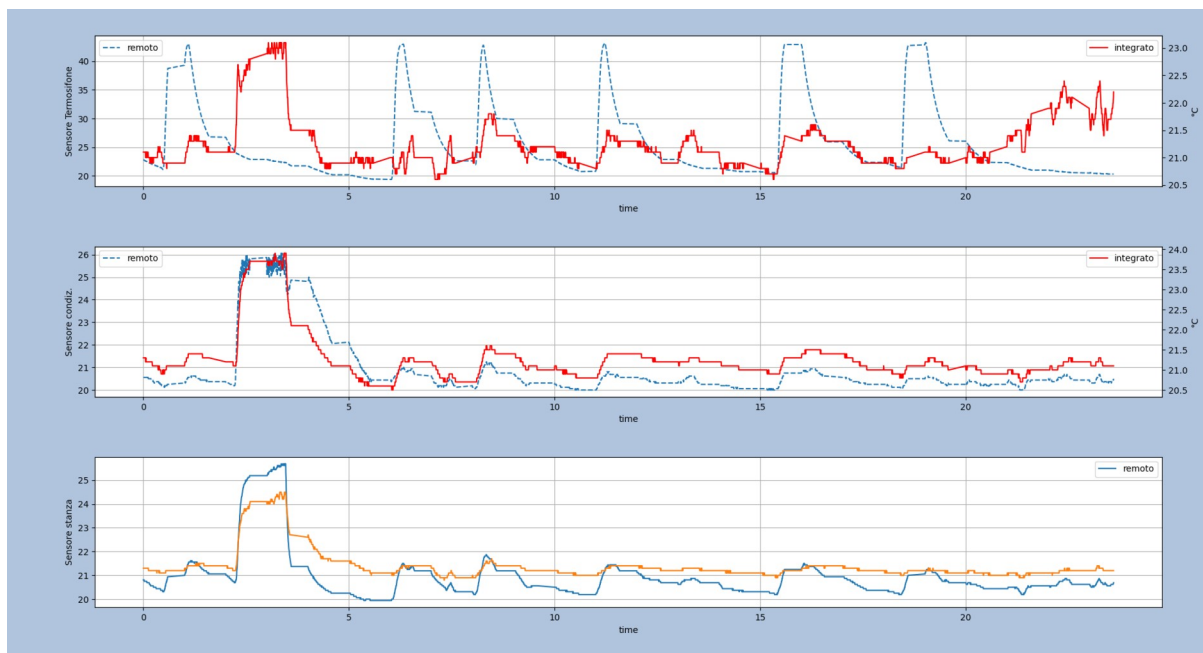
I dati che riguardano l'intero periodo sono sintetizzati nel grafico sotto.

Il conteggio dei dati ricevuti indica la somma dei dati ricevuti da tutti i moduli, mentre i dati di temperatura e umidità sono relativi al sensore interno del modulo 2.



Capacità di trasferimento termico

Un altro tipo di analisi che possiamo fare è quello di confrontare, dato che li abbiamo nella stessa stanza, la capacità di riscaldamento del termosifone con quella della pompa di calore. Esaminando i grafici dei dati su base giornaliera possiamo notare che c'è una discreta differenza tra l'efficienza del termosifone e quella della pompa di calore.



In particolare possiamo notare che, mentre nel caso in cui si accendono i termosifoni la curva di riscaldamento di questi ultimi è molto ripida e arriva velocemente oltre i 40 gradi, la temperatura della stanza segue questo andamento salendo soltanto di qualche decimo di grado e con qualche minuto di ritardo dovuto alla diffusione del calore dal termosifone al resto della stanza.

Quando invece viene accesa la pompa di calore notiamo che la temperatura della stanza, compresa quella rilevata dai sensori del modulo più lontano dalla bocchetta di emissione della pompa (il modulo 2), inizia immediatamente a salire e segue l'andamento della temperatura rilevata dal sensore posto davanti alla bocca di uscita dell'aria.

Notiamo inoltre che la temperatura dell'aria di uscita è di appena 26 gradi, il che vuol dire che la quantità di energia necessaria per riscaldarla è sicuramente minore di quella per riscaldare l'acqua oltre 40 gradi e farla circolare nel circuito dei termosifoni.

Non ci è possibile misurare la quantità di energia che viene utilizzata dalla caldaia per fare un confronto, ma abbiamo misurato i consumi elettrici della pompa di calore e quest'ultima, a pieno regime, consuma circa 1400 watt. La temperatura impostata sulla pompa di calore inoltre era di soli 19 gradi e a volte era necessario spegnerla perché la temperatura nella stanza diventava eccessiva.

Contabilizzazione del calore

All'inizio della realizzazione del progetto avevamo fatto alcune ricerche su come sarebbe stato possibile utilizzare i moduli come contabilizzatori di calore.

La contabilizzazione del calore può essere effettuata nel modo seguente:

1. Si fissa una temperatura minima per considerare che il termosifone funzioni
2. Si fissa la differenza di tempo per far avanzare il conteggio.
3. Quando la temperatura rilevata supera la temperatura minima vuol dire che il termosifone è acceso e quindi iniziamo a contare.
4. Finché la temperatura rimane al di sopra della temperatura minima ogni delta di tempo facciamo avanzare il numero di elementi contati.

Nell'immagine sotto un estratto dalle caratteristiche di un contabilizzatore di calore commerciale.

Conteggio

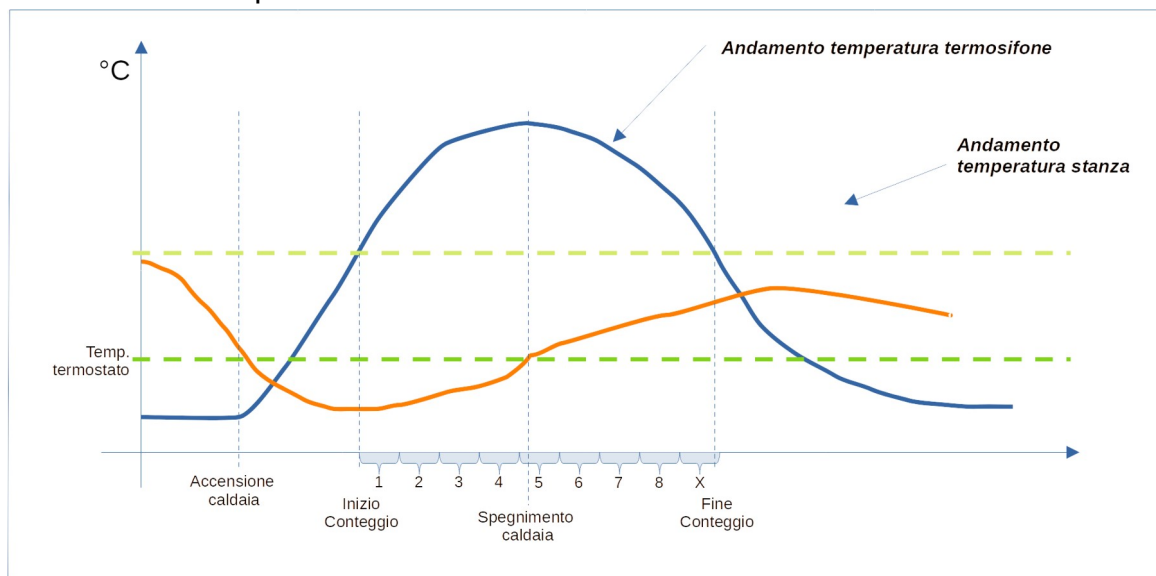
Funzionamento a due sensori e commutazione ad un sensore in presenza di accumulo interno di calore.

ΔT di commutazione: 4,5 K

Temperatura (media di piastra) di inizio conteggio a un sensore: 28°C

Ciclo di conteggio: 2 minuti

Facciamo un esempio di funzionamento:



In questo esempio possiamo vedere che, nel momento in cui scende la temperatura nella stanza dove si trova il termostato questo accende la caldaia. Da quel momento in poi la temperatura del termosifone sale fino a superare la soglia impostata sul contabilizzatore di calore per iniziare il conteggio (28° nello schema più sopra).

Superata la soglia ogni delta di tempo (2 minuti) viene conteggiata una unità.

Nel nostro schema vengono conteggiati 8 delta di tempo prima che la temperatura del termosifone torni al di sotto della soglia. Il 9° intervallo non viene conteggiato (X) perché non è passato l'intero delta di tempo necessario.

Conclusioni

Nonostante il progetto fosse partito con l'idea di realizzare un contabilizzatore di calore, si è rivelato più interessante l'idea di fare un confronto tra due diverse modalità di riscaldamento.

Abbiamo riscontrato la differenza di comportamento dalla sola osservazione dei grafici ma si sarebbe anche potuta fare una analisi di tipo quantitativo più precisa per vedere quanto, ogni ciclo di riscaldamento dei termosifoni, riesca ad influire sulla temperatura nella stanza.

Per questo tipo di misurazioni sarebbe però necessario che la stanza non fosse parte di una casa abitata nella quale le porte vengono aperte e chiuse permettendo una circolazione dell'aria al di fuori della stanza. Tra l'altro il termostato che controlla l'accensione dei termosifoni si trova nel corridoio attiguo alla stanza e i suoi cicli non sono direttamente dipendenti da quelli del termosifone su cui stavamo misurando i dati.

Nell'immediato comunque è più conveniente utilizzare una pompa di calore per riscaldare rapidamente una stanza fredda.

Risorse

E' possibile scaricare tutti i file del progetto dal repository

https://github.com/pdomi2001/IOT_project

I file presenti sono:

- - mqtt_subscriber.py: Questo script resta in ascolto sul server MQTT e scrive le rilevazioni
- - converti_dati.py: Converte i dati ricevuti in un formato che sia leggibile da Pandas
- - analisi_dati_conessioni.py: Analizza i dati relativi alle connessioni effettuate dai moduli e crea una serie di grafici che indicano le riconessioni nell'intero periodo.
- - analisi_dati_periodo_intero.py: Analizza i dati relativi alle temperature medie giornaliere e l'umidità media giornaliera.
- - analisi_dati_termico.py: Analizza i dati relativi ad un dato giorno (nel nostro caso il 14 aprile 2022) e mostra gli andamenti, durante la giornata delle temperature registrate dai sensori interni e dalle sonde dei moduli.
- - progetto_esp.ino: Progetto per Arduino da caricare sulla ESP32
- - arduino_secrets.h: File che contiene le informazioni per far connettere alla rete WIFI la ESP32

Più i file accessori dei grafici generati e degli schemi presenti in questo documento.