



UNIVERSITÀ DI FIRENZE

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

**CREAZIONE DI UNA RETE DI SENSORI PER
IL MONITORAGGIO DELLE SERRE
DELL'ORTO BOTANICO DI FIRENZE**

Relatore:

Boni Enrico

Candidato:

Carletti Simone

Abstract

In un'epoca caratterizzata dall'Internet delle Cose (IoT) e dalla crescente automazione, l'uso di soluzioni intelligenti e connesse sta guadagnando terreno in vari contesti. Questa tesi si concentra sull'applicazione dei principi dell'IoT alle serre dell'*Orto Botanico "Giardino dei Semplici"* di Firenze, attraverso la creazione di una rete di sensori intelligenti.

Attualmente, il monitoraggio delle serre presenta sfide significative a causa delle dimensioni e dell'altezza di questi ambienti. Tentativi precedenti con sistemi commerciali sono stati costosi e dipendenti da server proprietari esterni, rendendo il processo di acquisizione dati laborioso e vulnerabile a cambiamenti nell'azienda terza.

Per superare queste sfide, la presente tesi propone la creazione di una rete di sensori basata su Zigbee, un protocollo di comunicazione senza fili a basso consumo energetico creato *ad-hoc* per l'IoT. La presente esplora dettagliatamente le specifiche di quest'ultimo, inclusi i livelli fisici e MAC, i componenti della rete e le topologie. Inoltre, si discute l'utilizzo di MQTT, un altro protocollo chiave nel mondo dell'IoT.

Il sistema realizzato utilizza un Raspberry Pi come macchina centrale per la gestione della rete, oltre una serie di software open-source per la raccolta, persistenza e visualizzazione dei dati (Zigbee2MQTT, Mosquitto, Node-RED, InfluxDB e Grafana). Infine, al termine della trattazione, si vede anche come realizzare un semplice dispositivo *smart* che comunica tramite MQTT.

In conclusione, questa tesi offre un'analisi dettagliata dell'implementazione di una rete di sensori intelligente per il monitoraggio delle serre dell'Orto Botanico di Firenze, contribuendo a superare le sfide esistenti e fornendo un sistema flessibile e aperto per il monitoraggio ambientale.

*A tutti i miei colleghi e amici
con cui ho condiviso questo percorso*

Indice

1	Introduzione	3
2	Reti di sensori	5
2.1	Scelta del protocollo di comunicazione	5
2.2	Zigbee Alliance	6
2.3	IEEE 802.15.4	6
2.3.1	Livello fisico	7
2.3.2	Livello MAC	8
2.3.3	Componenti della rete	8
2.3.4	Topologie di rete	9
2.3.5	Basso consumo energetico	11
2.3.6	Panoramica funzionale	11
2.4	Zigbee	14
2.4.1	Componenti della rete	15
2.4.2	Topologia di rete	15
2.4.3	Livello rete (NWK)	16
2.4.4	Livello applicazione (APL)	17
2.4.5	Application Profiles e Clusters	19
2.5	MQTT	20
3	Il sistema realizzato	22
3.1	Ricezione dati	22
3.1.1	Zigbee Coordinator	22
3.1.2	Inizializzazione RPi	22
3.1.3	Zigbee2MQTT	23
3.2	Persistenza dati e visualizzazione	25
3.2.1	InfluxDB	25
3.2.2	Node-RED	27
3.2.3	Grafana	31
3.3	Schema riassuntivo	34
3.4	Landing page	34
3.5	Display MQTT	35
4	Conclusioni	43

Capitolo 1

Introduzione

Nell'era dell'Internet delle Cose (dall'inglese IoT, **Internet of Things**) e della crescente automatizzazione, l'applicazione di soluzioni intelligenti e connesse per problemi di vario genere si sta diffondendo sempre di più.

Al giorno d'oggi sempre più edifici sono attrezzati di sensori intelligenti, i quali ci permettono di ottenere un'informazione più completa degli ambienti, così da poter fare (o far fare ad un algoritmo) scelte più consapevoli, ad esempio dal punto di vista ecologico.

La presente tesi si concentra sull'applicazione dei concetti dell'IoT agli ambienti delle serre dell'*Orto Botanico "Giardino dei Semplici"* attraverso la creazione di una **rete di sensori intelligente**.

Il problema attuale che hanno i dipendenti dell'orto è che non hanno modo di monitorare correttamente gli ambienti in questione. Questo perchè, soprattutto le 2 serre principali, sono ambienti molto ampi e alti (circa 9m in altezza) e i pochi termometri (non connessi, le informazioni si leggono da un display) che hanno disposto servono solo per avere un'informazione molto generale, oltre a mancare di una cosa essenziale: lo storico.

Oltre a questo, per avere un'immagine completa degli ambienti, sarebbe necessario disporre i sensori in posizioni diverse, anche dove non è possibile accedervi facilmente per recuperare i dati (ad esempio ad un'altezza elevata, in modo da poter osservare le differenze di temperatura).

Prima di questa proposta sono stati sperimentati alcuni sistemi commerciali. Il problema incontrato, oltre al grosso costo delle apparecchiature, è stato che non sono riusciti ad individuare nessun sistema che disponesse di sensori wireless, interconnessi e "open". Con quest'ultimo termine intendo che gli unici sistemi individuati utilizzavano server proprietari (remoti) per l'immagazzinamento dei dati, per cui solo scaricare queste informazioni era un processo laborioso, oltre al fatto che il sistema era dipendente dalla piattaforma di un'azienda terza (e quindi, se un giorno

tale ditta dovesse scomparire, e con quella i loro server, il sistema diventerebbe pressochè inutile).

In alternativa erano stati provati anche dei sensori non interconnessi, dai quali però era necessario scaricare i dati manualmente, andando fisicamente a prendere il sensore (e quindi non pratici).

Proprio per questi motivi si è deciso di realizzare la rete di cui questa tesi è oggetto.

Capitolo 2

Reti di sensori

Le *Wireless Sensor Network*, sono reti costituite da dispositivi (solitamente alimentati a batteria) che comunicano tra di loro tramite segnali radio.

Negli ultimi tempi questi tipi di reti stanno suscitando sempre più interesse, questo è dato soprattutto dalla necessità di creare reti che non richiedano né infrastrutture costose né un controllo centralizzato, rendendole quindi semplici ed economiche da implementare. Solitamente queste reti operano in modalità ad-hoc, consentendo ai nodi di comunicare tra loro e determinare autonomamente le migliori modalità di cooperazione per trasmettere di dati.

2.1 Scelta del protocollo di comunicazione

I protocolli standard *de-facto* per le reti di sensori sono sicuramente **WiFi** e **Zigbee**, i quali ormai dominano il mercato. Questo perché sono *open* ed economici da implementare, rendendoli perfetti anche per uso commerciale.

Il vantaggio principale di Zigbee rispetto a WiFi, è che il primo è stato ideato proprio per reti di sensori: è quindi stato pensato per essere leggero, economico, a basso consumo energetico ed altamente versatile.^[1] Questo lo rende perfetto per comunicazioni fra sensori alimentati a batteria. Inoltre, grazie alla diffusione di questo protocollo a livello globale, sul mercato sono disponibili tantissimi sensori e dispositivi, economici e facilmente reperibili.

Detto questo, WiFi si riserva comunque il suo spazio nel mondo dell'IoT, soprattutto per dispositivi dove la durata della batteria non è così importante (anche se grazie a vari accorgimenti è possibile creare sensori WiFi a batteria, seppur con una durata inferiore a quella di Zigbee). Ad esempio nel mondo del *DIY*¹, WiFi rimane sicuramente il protocollo più popolare, vista la sua semplicità di implementazione (ad esempio su schede Arduino o esp8266).

Questo protocollo rimane di larga diffusione anche nel mondo della domotica visto

¹*Do It Yourself, Fai da te*

che non necessita di hardware aggiuntivo: è possibile acquistare e connettere un sensore wifi direttamente al router di casa, cosa che non è possibile con altri protocolli (come zigbee, che richiede un gateway che lo supporti, vedi [2.4](#)).

2.2 Zigbee Alliance

La necessità di creare una rete mesh a basso consumo energetico nacque verso la fine degli anni '90, quando molti ingegneri cominciarono a capire che i protocolli preesistenti (come WiFi e Bluetooth) non erano in grado di soddisfare le esigenze di molte nuove applicazioni. Proprio per affrontare questo problema, nel 2002 nacque la *Zigbee Alliance* (oggi nota come *Connectivity Standards Alliance*). Composta da più di 500 aziende fra cui Amazon, Apple, Samsung, Philips, Motorola, Texas Instruments, etc., [\[2\]](#) è quest'ultima che mantiene e gestisce lo standard Zigbee.

La prima versione di Zigbee fu definita nel dicembre 2004, anno in cui fu definito lo standard IEEE 802.15.4, su cui il primo si basa. Zigbee utilizza il livello fisico (PHY) e MAC messi a disposizione dallo standard IEEE, per poi andare a definire sopra essi il livello rete (NWK) e la struttura per il livello applicativo (APL).

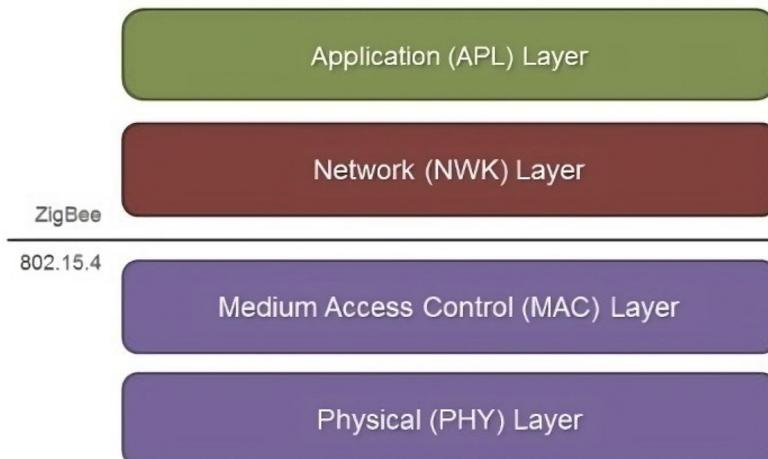
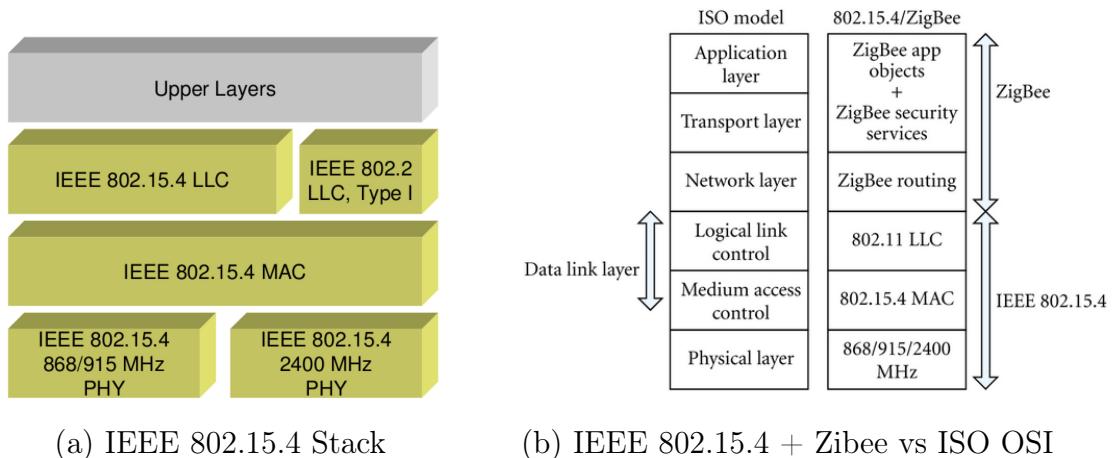


Figura 2.1: Stack Zigbee [\[3\]](#)

2.3 IEEE 802.15.4

Lo standard IEEE 802.15.4, [\[4\]](#) definisce il livello fisico ed il livello MAC per la creazione di reti wireless in area personale che lavorano con basse velocità di trasferimento dati (LR-WPAN, Low-Rate Wireless Personal Area Networks).



2.3.1 Livello fisico

IEEE 802.15.4 definisce due livelli fisici: uno basato sulla banda 868/915 MHz e un altro basato sulla banda 2400 MHz (rientrano tutte in quella che viene chiamata ISM, *Industrial Scentific Medical*, band). Questo perchè a seconda della locazione geografica, cambiano le frequenze che possono essere utilizzate liberamente senza licenza: [5]

- Il primo opera su due bande diverse: 868 MHz per l'Europa e 915 MHz per l'America e l'Australia. Quella europea mette a disposizione un solo canale con un data rate massimo di 20kbps, mentre la seconda mette a disposizione 10 canali con un data rate massimo di 40kbps.
- Il secondo opera sulla banda ISM mondiale (2.4 GHz, la stessa utilizzata da WiFi). Supporta un rate di massimo 240 kbps e la banda è divisa in 16 canali.

I principali vantaggi dei due livelli sono:

- PHY 868/915 MHz: è meno utilizzato e quindi ci sono meno interferenze
- PHY 2400 MHz: maggiore velocità e compatibilità al livello mondiale

Nonostante la condivisione della frequenza con altri servizi (come WiFi), è il secondo livello che ha visto il maggior utilizzo. [6] Questo per una serie di motivi:

- la compatibilità globale è comoda dal punto di vista commerciale, visto che è possibile creare una singola versione del dispositivo distribuibile ovunque.
- grazie alla velocità più elevata la trasmissione/ricezione sono attive per un periodo più breve e questo porta ad un consumo energetico più basso.

Per concludere, i compiti principali del livello fisico sono: [4] [7]

- Attivazione e disattivazione del trasmettitore radio

- Selezione del canale
- Indicazione del livello di qualità (*Link Quality, LQI*) per i pacchetti ricevuti
- Verifica del canale libero con il protocollo CSMA-CA (*Carrier Sense Multiple Access - Collision Avoidance*)
- Rilevazione del livello di potenza del segnale ricevuto (*RSSI, Received Signal Strength Indicator*)
- Trasmissione e ricezione dei dati

2.3.2 Livello MAC

Il sottolivello MAC è colui che gestisce tutti gli accessi al canale radio fisico, oltre ad essere responsabile delle seguenti attività: [4] [6]

- Fornitura di servizi per associare/disassociare i dispositivi alla rete
- Fornire il controllo dell'accesso ai canali condivisi
- Utilizzo del meccanismo CSMA-CA per l'accesso al canale
- Generazione di beacon (se applicabile)
- Gestione dei *guaranteed time slots* (GTS) (se applicabile)

Inoltre, offre i seguenti servizi al livello immediatamente superiore:

- **MAC Data Service (MCPS)**: fornisce un meccanismo per il passaggio dei dati da e verso il livello superiore.
- **MAC Management Services (MLME)**: fornisce meccanismi per controllare le impostazioni di comunicazione e della rete, dal livello superiore.

2.3.3 Componenti della rete

Nelle reti basate sullo standard IEEE 802.15.4, esistono le seguenti tipologie di dispositivi: [4]

- **Device**: Una qualsiasi entità contenente un'implementazione del livello PHY e MAC definito dallo standard. Essi si suddividono in: RFD (*reduced-function device*) e FFD (*full-function device*).
- **RFD** (*reduced-function device*): Un dispositivo che funziona con un'implementazione minimale del protocollo IEEE 802.15.4.
- **FFD** (*full-function device*): Un dispositivo in grado di funzionare come coordinatore o *device* generico; implementa il set completo del protocollo.

- **Coordinator:** Un *FFD* configurato per fornire servizi di sincronizzazione attraverso l'invio di segnali chiamati beacon. Se un coordinatore è il controllore principale di una rete personale (PAN), viene chiamato **PAN Coordinator**.

Per costituire una WPAN (*Wireless Personal Area Network*), è necessario che ci siano almeno due *device* che comunicano sullo stesso canale fisico, localizzati all'interno di uno stesso *POS*². È inoltre necessaria la presenza di almeno 1 *FFD* che opera come *coordinatore PAN*.

Ogni dispositivo che opera sulla rete deve avere un indirizzo univoco globale a 64 bit (con $2^{64} \approx 1.8 \times 10^{19}$, è sicuramente possibile associare un indirizzo diverso ad ogni dispositivo sulla terra).

L'indirizzo a 64 bit può essere utilizzato per la comunicazione diretta all'interno della PAN, oppure può essere scambiato con un indirizzo breve a 16 bit assegnato dal *PAN Coordinator* durante la procedura di associazione. Per quest'ultimo motivo una singola rete è capace di gestire fino a 65535 nodi.

2.3.4 Topologie di rete

A seconda dei requisiti dell'applicazione, la rete può funzionare in due topologie diverse: **a stella** o **peer-to-peer**.^[4] Entrambe sono mostrate nella figura 2.3:

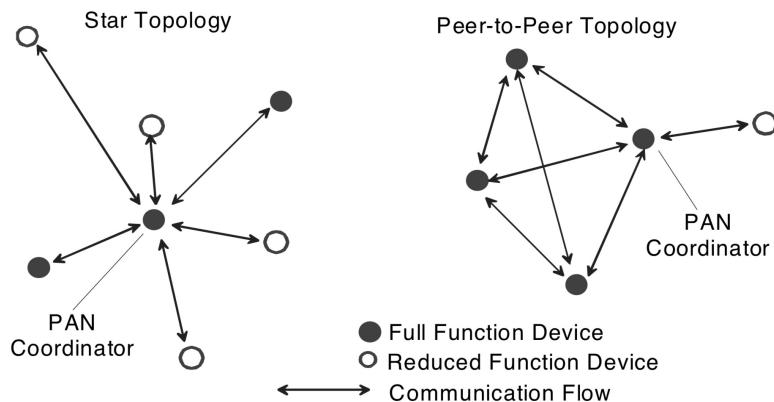


Figura 2.3: Topologie di reti IEEE 802.15.4^[4]

- Nella topologia a **stella** i nodi comunicano direttamente con il *PAN coordinator*. Quest'ultimo è colui che inizializza la rete e permette agli altri nodi (sia RFD che FFD) di connettersi.
- Anche nella topologia **peer-to-peer** è prevista la presenza di un coordinatore PAN, tuttavia, differisce dalla topologia a stella in quanto qualsiasi dispositivo può comunicare con qualsiasi altro dispositivo, purché si trovino nel raggio

²**POS** (*Personal Operating Space*): Lo spazio attorno a una persona o a un oggetto, tipicamente di un raggio di circa 10m

d'azione l'uno dell'altro.

La principale differenza è che la topologia peer-to-peer consente di implementare formazioni di rete più complesse, come la topologia di **rete mesh**. I vantaggi principali di una rete del genere sono che essa si può auto-organizzare³ e auto-riparare. Può anche consentire un routing *multi-hop* da qualsiasi dispositivo a qualsiasi altro dispositivo sulla rete. Tali funzioni possono essere aggiunte a livello di rete (NWK), ma non fanno parte di questo standard (vedremo più avanti nella sezione 2.4 che sarà Zigbee ad implementarle).

Topologia Cluster-Tree

Questa topologia è un caso speciale di rete **peer-to-peer**, in cui la maggior parte dei dispositivi sono FFD; i nodi RFD possono unirsi alla rete solo come foglie alla fine di un ramo.

Qualsiasi FFD può fungere da coordinatore e fornire servizi di sincronizzazione ad altri dispositivi o altri coordinatori. Solo uno di questi coordinatori però può essere il *PAN coordinator*, il quale generalmente disporrà di maggiori risorse computazionali rispetto a qualsiasi altro dispositivo della rete.

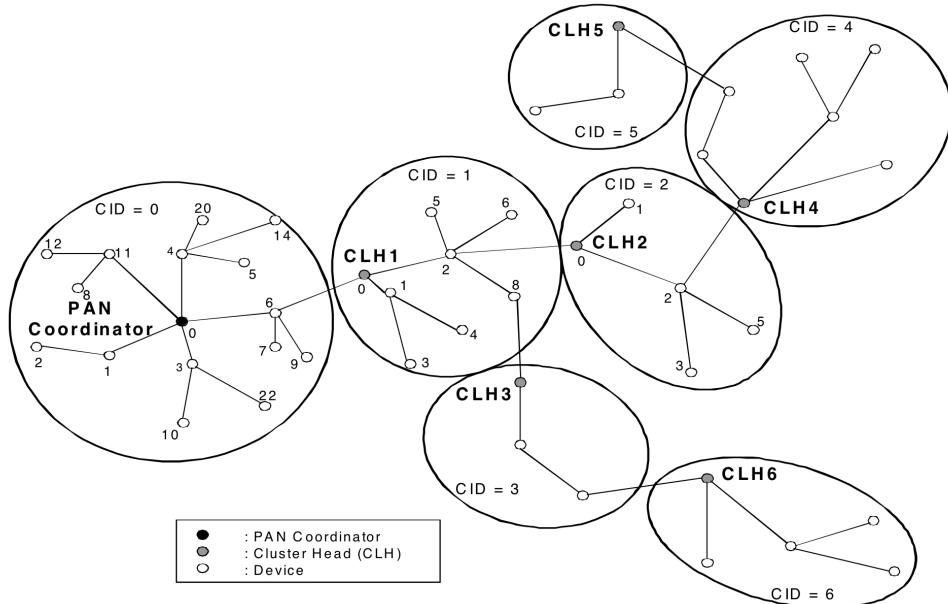


Figura 2.4: Topologie Cluster-Tree^[4]

³Il principale vantaggio di una rete mesh è la sua abilità di auto-organizzarsi e si auto-configurarsi *dinamicamente*, riducendo così i costi di installazione. La capacità di autoconfigurazione, inoltre, consente una distribuzione dinamica del carico di lavoro, specialmente in caso di malfunzionamenti di alcuni nodi. Ciò a sua volta contribuisce alla tolleranza ai guasti e alla riduzione dei costi di manutenzione.

Il coordinatore PAN forma il primo cluster stabilendosi come *cluster head* (CLH), ponendo il CID (*Cluster Identifier*) pari a zero. I nodi adiacenti, se permesso, possono poi unirsi alla rete. La rete risultante può essere successivamente estesa attraverso l'elezione di nuovi CLH che formano nuovi cluster adiacenti al primo.^[4]^[7]

La forma più semplice è una rete a cluster singolo, ma sono possibili reti più grandi formando una maglia di più cluster vicini (come si può vedere in figura 2.4). Il vantaggio di una struttura *multicluster* è l'aumento dell'area di copertura, mentre lo svantaggio è l'aumento della latenza dei messaggi.

2.3.5 Basso consumo energetico

Un tipico dispositivo di rete alimentato a batteria presenta notevoli sfide tecniche. Questi dispositivi sono generalmente di piccole dimensioni, utilizzano batterie a bassa capacità, ed è spesso richiesta una manutenzione poco frequente (implicando lunghi periodi fra sostituzioni di batterie, che quindi devono durare a lungo).

L'utilizzo della batteria deve essere quindi gestito con attenzione per sfruttare al meglio risorse energetiche molto limitate per un lungo periodo di tempo.

Per questo motivo il protocollo utilizza bassi *duty cycles*: la maggior parte del consumo energetico di un dispositivo wireless corrisponde ai momenti in cui il dispositivo trasmette.

Si definisce *duty cycle* (*ciclo di lavoro*) il rapporto:

$$D = \frac{\text{durata trasmissione}}{\text{intervallo fra trasmissioni}}$$

Nelle dispositivi che implementano IEEE 802.15.4, l'uso della batteria viene ottimizzato utilizzando cicli di lavoro estremamente brevi, in modo che il dispositivo trasmetta solo per una piccolissima frazione di tempo. Oltre a questo, quando non trasmette, il dispositivo dovrebbe passare a una modalità di *deep sleep* a basso consumo energetico per minimizzare il consumo di batteria.

Da notare che non tutti i dispositivi di una rete IEEE 802.15.4 sono adatti ad essere alimentati a batteria: ad esempio ci sono nodi che devono essere sempre accesi (e non possono entrare in *deep sleep*), come i *Coordinators*.

2.3.6 Panoramica funzionale

In questa sezione vediamo, in breve, come funziona la comunicazione fra dispositivi basati sullo standard.

In primis, il protocollo MAC IEEE 802.15.4 definisce due differenti modalità di funzionamento: beacon enabled e non-beacon enabled mode.^[6]

Beacon enabled mode

In questa modalità il *PAN coordinator* invia periodicamente un segnale di beacon ai nodi della rete. I beacon contengono informazioni che consentono ai nodi di tenere la rete sincronizzata.

Normalmente, due beacon successivi segnano l'inizio e la fine di un *superframe*. Un superframe contiene 16 finestre temporali che possono essere utilizzate dai nodi per comunicare sulla rete (potrebbe esserci anche un periodo morto alla fine del superframe). L'intervallo di tempo totale di questi time slot è chiamato *Contention Access Period (CAP)*, durante il quale i nodi possono tentare di comunicare utilizzando lo *slotted CSMA/CA*. Questo è mostrato in figura 2.5.

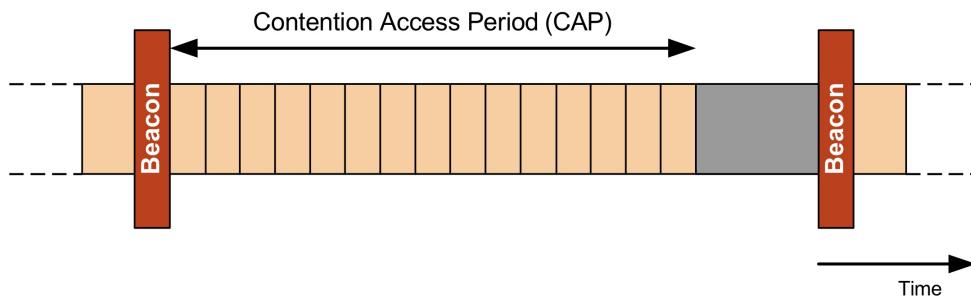


Figura 2.5: Superframe^[6]

Un nodo può richiedere che gli vengano assegnati particolari timeslots tra i 16 disponibili. Si tratta di timeslots consecutivi chiamati *Guaranteed Timeslots (GTSs)*: ogni GTS può essere costituito da più timeslots. Quest'ultimi si trovano subito dopo il CAP. L'intervallo di tempo totale di tutti i GTS (per tutti i nodi) è chiamato *Contention Free Period (CFP)*. La comunicazione nella CFP non richiede l'uso di CSMA/CA (da questo il nome). Ciò è illustrato nella figura 2.6.

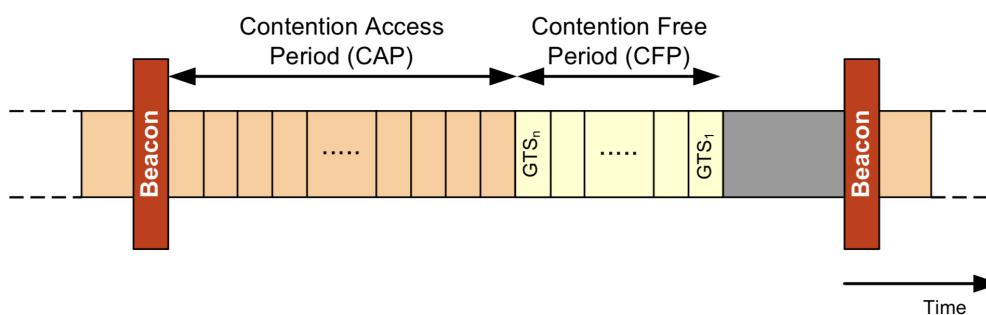


Figura 2.6: Superframe con GTSs^[6]

Non-beacon enabled mode

In questa modalità, invece, i beacon non vengono trasmessi periodicamente (possono però, ad esempio, essere richiesti per associare un nuovo dispositivo al coordinatore). Invece, le comunicazioni sono asincrone: un dispositivo comunica con il Coordinatore solo quando necessario, il che può accadere relativamente di rado. Ciò consente di risparmiare energia. Per determinare se ci sono dati in sospeso per un nodo, il nodo deve interrogare il Coordinatore (in una rete beacon-enabled, invece, la disponibilità dei dati in sospeso è indicata nei beacon). Rispetto al precedente (beacon-enabled) questa modalità è leggermente più lenta e non è adatta a situazioni che richiedono bassa latenza.

Trasferimento dati

Lo standard definisce tre differenti modalità di trasferimento dati, definiti in modo tale che queste possano essere controllate dai nodi della rete piuttosto che dal coordinatore: [7] [4]

- **Trasferimento dati verso il coordinatore** (trasmissione dati diretta): viene eseguito usando un algoritmo CSMA-CA slottizzato oppure non slottizzato, a seconda della modalità di funzionamento. L'uso degli ACK (acknowledgement) è opzionale.
- **Trasferimento dati da un coordinatore ad un nodo** (trasmissione dati indiretta): viene controllato dal nodo. Qua la modalità differisce a seconda se siamo in beacon-enabled o in non-beacon-enabled mode: nella prima il nodo si accorge di dati disponibili grazie alle informazioni presenti nei beacon, nella seconda deve interrogare il coordinatore per vedere se sono disponibili nuovi dati. In questa modalità sono obbligatori gli ACK.
- **Trasferimento dati peer-to-peer**: In una PAN peer-to-peer, ogni dispositivo può comunicare con ogni altro dispositivo nella raggio di copertura. Per fare ciò in modo efficace, i dispositivi che desiderano comunicare dovranno o ricevere costantemente o sincronizzarsi tra loro.

Tipi di Frame

Le strutture dei frame sono state progettate per mantenere la complessità al minimo e allo stesso tempo renderle sufficientemente robuste per la trasmissione su un canale rumoroso. Ogni livello successivo del protocollo aggiunge alla struttura un header e un footer specifico. Sono state definite quattro strutture per un frame:

- Frame beacon, utilizzato da un coordinatore per trasmettere beacon
- Frame di dati, utilizzato per tutti i trasferimenti di dati
- Frame ACK, utilizzato per confermare l'avvenuta ricezione di frames

- Frame di comando MAC, utilizzato per gestire tutti i trasferimenti fra livelli MAC di peers

2.4 Zigbee

Come anticipato nella sezione 2.2, Zigbee è stato pensato come standard di comunicazione wireless a bassissimo costo e a bassissimo consumo energetico, mirato al mercato dell'IoT.

Inoltre, come già accennato, Zigbee va a costruire sull'esistente stack definito da IEEE 802.15.4 (vedi 2.3), aggiungendo il **livello rete (NWK)** e **livello applicazione (APL)**, permettendo così a questo protocollo di ereditarne tutti i vantaggi (come basso consumo energetico, topologia rete mesh, etc.).

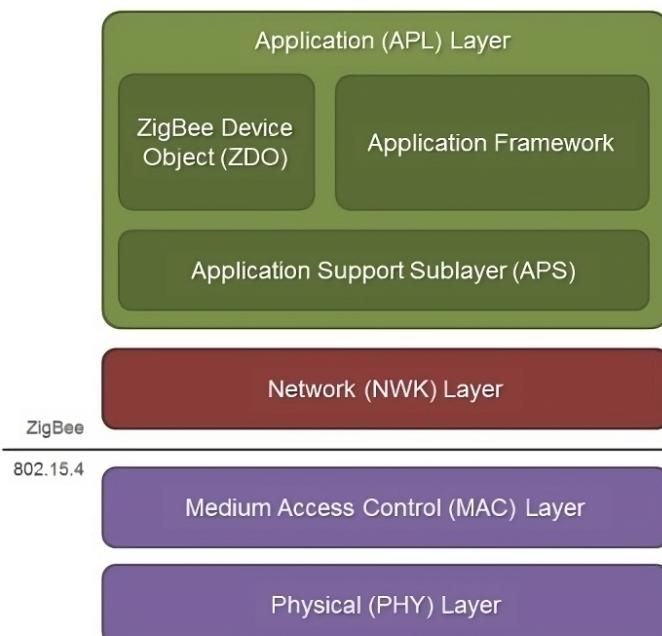


Figura 2.7: Stack Zigbee^[3]

Essendo Zigbee uno standard mirato al mondo dell'IoT, le sue principali caratteristiche sono:

- Basso costo
- Sicuro, affidabile e auto-riparante
- Flessibile ed estendibile
- Bassissimo consumo energetico
- Facile ed economico da implementare

2.4.1 Componenti della rete

Zigbee estende le tipologie di devices viste per IEEE 802.15.4 ([2.3.3](#)), definendo 3 tipi di dispositivi: [\[8\]](#)

- **Coordinator:** coincide con il *PAN Coordinator* già visto per il protocollo IEEE 802.15.4. È responsabile per l'inizializzazione e il mantenimento della rete ZigBee, permettendo anche la connessione di nuovi nodi.
- **Router:** coincide con un *FFD* di IEEE 802.15.4 che può però agire anche come un *IEEE 802.15.4 Coordinator* (che è diverso dal coordinator di cui al punto precedente). Questo gli permette di avere capacità di routing, di conseguenza può estendere la copertura della rete, oltre a fornire percorsi alternativi in caso di congestione o guasti. Può connettersi al coordinatore o ad un altro router. Ad esso vi si possono connettere altri routers o end-devices.
- **End devices:** Questi possono essere FFD o RFD, possono però solo trasmettere e ricevere messaggi, senza l'abilità di eseguire alcuna operazione di routing. Possono collegarsi direttamente al coordinatore o ad un router, ma ad essi non vi si possono collegare altri dispositivi.

2.4.2 Topologia di rete

In accordo con IEEE 802.15.4, Zigbee supporta le topologie stella, cluster-tree e mesh. [\[8\]](#) [\[7\]](#) Sono illustrate nella figura [2.8](#).

- Nella topologia a **stella**, la rete è controllata dal coordinatore, e tutti gli altri dispositivi dialogano direttamente con lui.
- Nelle topologie a **maglia** o **cluster-tree**, invece, la rete può essere estesa grazie all'utilizzo dei routers. La differenza fra le due la possiamo vedere in [2.3.4](#)⁴. Una differenza aggiuntiva fra le ultime due è che l'ultima (*cluster-tree*) può utilizzare la modalità *beacon-enabled*, mentre la prima no.

⁴Per quanto riguarda il cluster-tree abbiamo una differenza di nomenclatura: quelli che erano i *Cluster Heads* ora prendono il nome di *Routers* (o *Coordinator* per il CLH 0), mentre quelli che erano *Devices* prendono il nome di *End-Devices*. La topologia a mesh, invece, coincide con quella che era stata chiamata peer-to-peer.

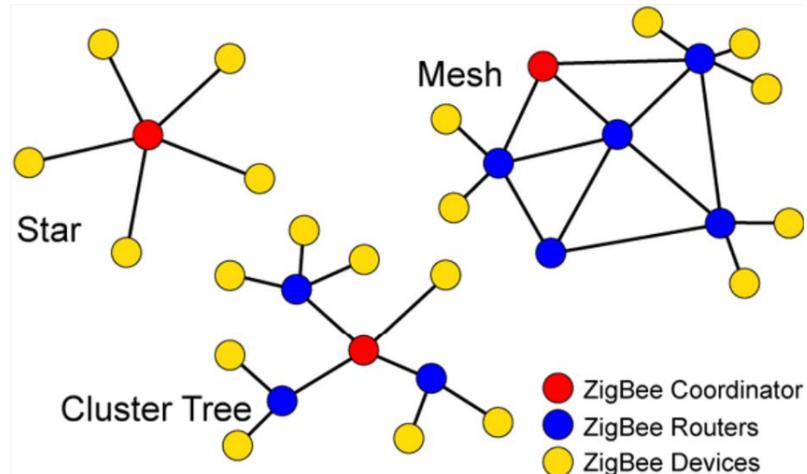


Figura 2.8: Topologia di reti Zigbee^[9]

2.4.3 Livello rete (NWK)

Le funzioni principali del livello rete consistono nel garantire l'uso corretto del sottolivello MAC e nel fornire un'interfaccia adatta per l'utilizzo da parte del livello APL. I compiti di questo livello includono: l'inizializzazione della rete, l'assegnazione degli indirizzi, l'aggiunta e la rimozione di dispositivi di rete e l'instradamento (routing) dei messaggi. È inoltre questo livello a garantire la sicurezza della rete.

Per quanto riguarda i singoli dispositivi:

- **Coordinator:** è suo il compito di inizializzare la rete. Sceglie il canale sul quale avverranno le comunicazioni e imposta il *PAN ID*, quindi avvia la rete.
- **Router:** essi si comportano come i routers di una comune rete, permettendo quindi di instradare i messaggi fra i vari dispositivi ad esso collegati. In particolare, il livello NWK di Zigbee permette anche l'instradamento *multi-hop*.
- **End-device:** gli end-device non hanno alcuna responsabilità di mantenimento dell'infrastruttura di rete, e quindi sono liberi di spegnersi e accendersi in qualsiasi momento. Proprio per questo motivo sono gli end-device che solitamente sono i più adatti ad essere alimentati a batteria.

L'algoritmo di routing scelto è "Ad hoc On-Demand Distance Vector" o *AODV*. In figura 2.9 si può vedere una tabella riassuntiva di confronto tra i ruoli delle tre tipologie di dispositivi.

ZigBee Network Layer Function	Coordinator	Router	End Device
Establish a ZigBee network	.		
Permit other devices to join or leave the network	.	.	
Assign 16-bit network addresses	.	.	
Discover and record paths for efficient message delivery	.	.	
Discover and record list of one-hop neighbors	.	.	
Route network packets	.	.	
Receive or send network packets	.	.	.
Join or leave the network	.	.	.
Enter sleep mode			.

Figura 2.9: Confronto di dispositivi Zigbee al livello rete [10]

2.4.4 Livello applicazione (APL)

Il livello applicazione è l'ultimo livello definito da Zigbee, e costituisce l'effettiva interfaccia verso gli utenti finali.

Questo livello si suddivide a sua volta in altri sottolivelli: *Application Framework*, *Zigbee Device Object (ZDO)* e *Application Support Sublayer (APS)*, come mostrato in figura 2.10. Questi sottolivelli comunicano fra di loro grazie ai *SAP* (o *Service Access Points*), rappresentati come ovali nell'immagine.

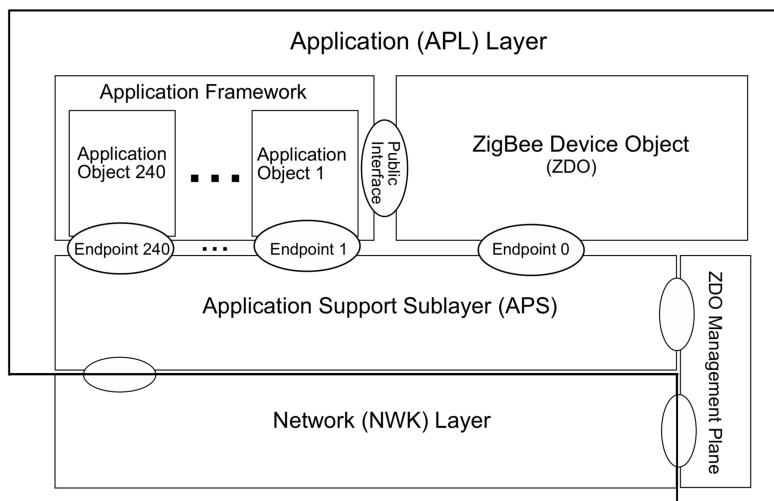


Figura 2.10: Livello Applicazione [10]

Application Support Sublayer (APS)

Il sottolivello APS è responsabile di:

- **Multiplexing/demultiplexing:** inoltra i messaggi del livello rete ai corretti *Application Objects*⁵ in base al loro *endpoint ID*.
- **Binding:** mantiene la tabella di associazione locale, ovvero registra i nodi e gli endpoints remoti che si sono registrati per ricevere messaggi dall'endpoint locale (questo permette di creare una connessione logica end-to-end fra le relative app dei dispositivi).
- Mappa gli indirizzi estesi a 64-bit sugli indirizzi brevi a 16-bit
- Gestisce gli ACK end-to-end

Application Framework

È il blocco logico che contiene gli *Application Objects*. Ciascun application-object implementa un'applicazione definita dal costruttore del dispositivo, che verrà poi eseguita dall'application framework.

Ogni applicazione è indirizzata dal corrispettivo *endpoint*⁶, rappresentato da un numero decimale a 8 bit: per questo motivo ci sono al massimo 255 endpoints. Fra i 255 endpoints solo il range 1-240 è utilizzabile dalle applicazioni (limitando quindi il numero di app a 240), i restanti sono riservati: l'endpoint 0 comunica con lo ZDO (vedi prossima sezione) e l'endpoint 255 è l'indirizzo di broadcast (i.e. un messaggio inviato qui viene mandato a tutti gli endpoint). I restanti endpoint (da 241 a 254) sono riservati per usi futuri.

Zigbee Device Object (ZDO)

È una particolare applicazione che gira sull'endpoint 0, creata per gestire lo stato del nodo zigbee. In particolare è responsabile di:

- Inizializzare il sottolivello APS e il livello NWK
- Definire la modalità operativa del dispositivo (i.e. se il dispositivo è un coordinator, router, o end device)
- Rilevamento del dispositivo e determinazione dei servizi applicativi forniti da quest'ultimo
- Avvio e/o risposta per richieste di binding
- Gestione della sicurezza

⁵Anche chiamati *Manufacturer-Defined Application Objects*, si tratta di componenti definiti dai produttori che effettivamente implementano le varie applicazioni.

⁶Un endpoint può essere pensato come l'analogo di una porta di TCP/IP

2.4.5 Application Profiles e Clusters

Un *Application Profile*, descrive un insieme di dispositivi utilizzati per un'applicazione specifica e, implicitamente, lo schema di messaggistica tra tali dispositivi (i.e. definisce quali messaggi e quali attributi certi dispositivi utilizzeranno).

Ad esempio, ci sono *application profiles* definiti per *Home Automation (HA)* e *Smart Energy*. Quest'ultimo profilo, per esempio, definisce varie tipologie di dispositivi come load-controllers, termostati, displays, etc., e per ciascun dispositivo definisce le funzionalità richieste (e.g. il load-controller deve rispondere a un comando di on/off per accendere o spegnere un carico).

Un *Profile ID* viene assegnato a ciascuna applicazione per identificarla in modo univoco.

Esistono due tipi di *application profiles*:

- **Public Application Profiles:** software interoperabili sviluppati direttamente dalla Zigbee Alliance per svolgere compiti specifici.
- **Private Application Profiles:** software proprietario scritto dal produttore per il suo particolare dispositivo

I dispositivi all'interno di un *application profile* comunicano tra loro tramite **clusters**, i quali possono essere relativi a *inputs* o *outputs* dal dispositivo.

Un cluster non è altro che un messaggio, il quale formato viene definito dall'*application profile*. I seguenti sono esempi di clusters definiti nell'*application-profile HA*:

- On/Off: permette di accendere e spegnere dispositivi
- Level Control: utilizzato per controllare i dispositivi che possono essere impostati su un livello compreso tra acceso e spento (e.g. luminosità di una lampadina)
- Color Control: controlla il colore dei dispositivi (e.g. il colore di una lampada rgb)

Ogni cluster, nell'ambito di un particolare *application-profile*, è identificato in maniera univoca da un *cluster ID*.

In particolare tutti i dispositivi che operano in un *application-profile* (pubblico o privato) devono rispondere correttamente a tutti i cluster richiesti. Ad esempio, un interruttore per un punto luce, che opera nel profilo pubblico HA, deve implementare correttamente il cluster On/Off.

La Zigbee Alliance ha definito una libreria di cluster, chiamata **Zigbee Cluster Library (ZCL)**, che contiene vari cluster di base utili a molte applicazioni.



An example endpoint implementation:

Endpoint # - Profile Name: Device Type

0 - ZigBee Device Profile (ZDP): ZDO

1 - HA: Thermostat

2 - HA: On/Off Output

3 - SE: In-Home Display

4 - MSP: Proprietary vendor extensions

Figura 2.11: Esempio di dispositivo Zigbee

2.5 MQTT

Concludiamo la lista dei protocolli con uno che riguarda dispositivi che comunicano via **WiFi** (IEEE 802.11) o, in generale, che utilizzano lo stack **TCP/IP**.

Il protocollo oramai diventato il *de-facto* standard è proprio MQTT: ideato per applicazioni IoT, risponde perfettamente alle necessità di quest'ultime.

MQTT^[11]^[12] (ISO/IEC 20922) è un protocollo di messaggistica client-server di tipo publish-subscribe, studiato per essere estremamente leggero da tutti i punti di vista: consuma poca banda, poca energia ed occupa poco spazio (e quindi perfetto anche sistemi embedded dove la memoria è limitata). Tutte queste specifiche lo rendono perfetto per applicazioni IoT.

Come già accennato MQTT si basa sul modello *publish/subscribe*. In questo modello ci sono due attori principali: i *publisher* (pubblicatori) e i *subscriber* (sottoscrittori). I publisher inviano messaggi a specifici canali di comunicazione chiamati **topic**, mentre i subscriber si iscrivono a questi topic per ricevere i messaggi di interesse.

Al centro del sistema MQTT c'è il **broker**. Quest'ultimo funziona come un intermediario che gestisce la distribuzione dei messaggi: quando un publisher invia un messaggio su un determinato topic, il broker lo riceve e lo inoltra a tutti i subscriber che sono iscritti a quel topic. In figura 2.12 è illustrata l'architettura.

MQTT Topic

Il modello di messaggistica di MQTT si basa quindi sui *topics*. Questi sono dei canali di comunicazione virtuali a cui i dispositivi/applicazioni si possono iscrivere o su cui possono pubblicare messaggi. I topic sono organizzati in una struttura gerarchica a livelli. Dal punto di vista pratico, un topic è rappresentato da una stringa, con i vari livelli separati da "/" (vedi figura 2.13).

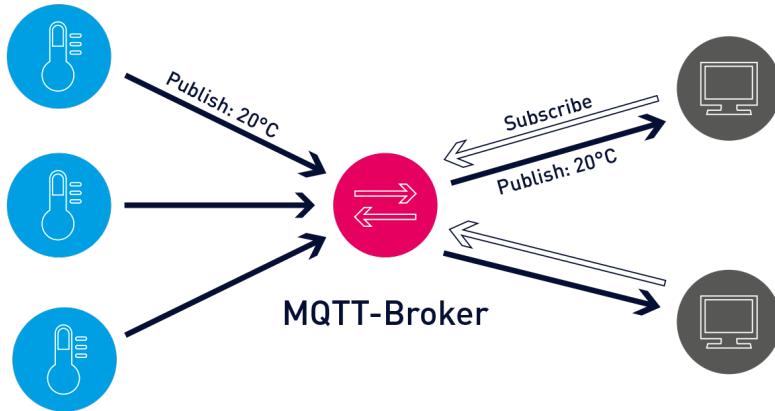


Figura 2.12: Architettura MQTT [13]

Inoltre esistono dei caratteri speciali per sottoscriversi a più topic in maniera semplice:

- **Wildcard livello singolo:** con il simbolo + è possibile indicare un qualsiasi livello. Ad esempio, iscrivendosi a `home/+/temperature` si potranno ricevere messaggi da `home/livingroom/temperature`, `home/bedroom/temperature`, etc.
- **Wildcard multilivello:** con il simbolo #, invece è possibile estendere il concetto del '+' a molteplici livelli. Da notare che è permesso solo come ultimo carattere in un topic. Esempio: iscrivendosi a `home/#` potremmo ottenere messaggi da `home/livingroom/temperature`, `home/kitchen/humidity`, etc.



Figura 2.13: Struttura di un topic [12]

Esempio

Un dispositivo pubblica un messaggio sul topic `home/livingroom/temperature`. Se qualche altro dispositivo è interessato alla temperatura del salotto può "iscriversi" al topic indicato, e ogni tal volta che il publisher invierà un messaggio su quel topic, lo riceverà.

Capitolo 3

Il sistema realizzato

Per quanto esposto nel capitolo 1, si vuole realizzare un sistema completamente locale che sia in grado di interconnettere più sensori, raccoglierne i dati ed elaborarli, mantenendo sempre il controllo completo su quanto acquisito.

3.1 Ricezione dati

Come primo passo, è necessario definire come ricevere i dati dai sensori in una macchina centrale, per poi poterli elaborare e visualizzare.

Visto che utilizzeremo il protocollo Zigbee (sezione 2.4), sappiamo che la rete necessita di un nodo centrale chiamato coordinatore. Sarà quest'ultimo il punto di raccolta dati (sink).

3.1.1 Zigbee Coordinator

Per creare un coordinatore Zigbee è necessario disporre di un elaboratore e di un dispositivo che ci permetta di ricevere il segnale wireless del protocollo.

In particolare è stato scelto come computer un Raspberry Pi 4 (un Single Board Computer relativamente economico) e un *SONOFF Zigbee 3.0 USB Dongle Plus-P*¹, che ci permette di interfacciare il RPi con la rete Zigbee.

3.1.2 Inizializzazione RPi

Per iniziare è necessario installare un sistema operativo sul nostro elaboratore. La scelta è stata per *Raspberry Pi OS Lite 64-bit*, so basato su Debian (distro di Linux) studiato proprio per il SBC da noi acquistato. Abbiamo scelto la versione lite poichè questa è più leggera, non disponendo di un desktop grafico (che a noi non servirà).

¹da qui in poi semplicemente dongle Zigbee

L'installazione è abbastanza semplice:

1. Collegare ad un PC un supporto di installazione. In questo caso abbiamo scelto una ssd esterna².
2. Installare sul PC "Raspberry Pi Imager", scaricabile dal sito ufficiale:
<https://www.raspberrypi.com/software/>.
3. Selezionare dalla lista l'os scelto e il supporto di memoria precedentemente inserito.
4. Nelle impostazioni avanzate è possibile inserire una serie di dati comodi all'avvio del rpi: credenziali per la connessione ad una rete wifi locale, nome dell'host, username e password.
Nel nostro caso si è impostato `username = admin`, `password = ProgettoSerra2.0` e `hostname = server-serra`. Grazie a quest'ultima impostazione, dalla rete locale, sarà possibile accedere al rpi tramite l'indirizzo `server-serra.local`.
5. Per concludere basta cliccare su "Write".

Una volta completata la procedura di scrittura su disco, basterà inserire l'ssd in una delle porte usb del raspberry. Ora accendiamo il rpi, dopo un paio di minuti dovrebbe essere pronto.

3.1.3 Zigbee2MQTT

Connettiamoci al raspberry via ssh (utilizzando le credenziali impostate nel processo di installazione). Una volta entrati è possibile procedere all'installazione dei software necessari.

Per la parte che riguarda zigbee la scelta è stata per *Zigbee2MQTT*: questo software ci permette automaticamente di creare un coordinatore in maniera molto facile. [14] In particolare ci permette di "convertire" i messaggi Zigbee in messaggi MQTT, facilmente integrabili in molti software di automazione.

Per l'installazione ci basta seguire la guida presente sul sito ufficiale:

https://www.zigbee2mqtt.io/guide/installation/01_linux.html.

Alcuni accorgimenti per l'installazione: bisogna seguire anche lo step opzionale "Running as a daemon with systemctl", così facendo zigbee2mqtt potrà girare in background e partire con l'avvio del sistema. Nel file di configurazione (*zigbee2mqtt.service*), modifichiamo solo due righe rispetto a quello di esempio:
`StandardOutput = null` e `User = admin`.

²Il motivo di questa scelta contro la normale micro-sd comune su un Rpi è che quest'ultima è molto più prona ad usurarsi nel tempo, mentre una ssd è molto più durevole.

Mosquitto

Visto che Zigbee2MQTT "converte" in messaggi MQTT sarà necessario installare un broker come richiesto dalla specifica dello standard (vedi 2.5). Per fare questo installiamo Mosquitto, un broker mqtt open-source molto popolare.

Dopo l'installazione c'è bisogno di cambiare solo qualche impostazione:

- Entriamo nella cartella del programma: `cd /etc/mosquitto`.
- Modifichiamo il file di configurazione con `vim mosquitto.conf`. Aggiungiamo `listener 1883` in modo da ammettere connessioni esterne a localhost.
- Sempre nello stesso file di configurazione aggiungiamo: `allow_anonymous false` e `password_file /etc/mosquitto/passwd` per abilitare l'autenticazione (in due righe separate).
- Ora non ci resta che aggiungere un file con le credenziali: lanciamo il comando `mosquitto_passwd -c passwd admin`, ci chiederà di inserire una password (nel nostro caso abbiamo di nuovo usato ProgettoSerra2.0). Così facendo abbiamo creato un utente con username admin e password ProgettoSerra2.0.

Possibile errore

Nel nostro caso, all'avvio di Zigbee2MQTT è stato incontrato il seguente errore: *network commissioning timed out - most likely network with the same panId or extendedPanId already exists nearby*. Questo è stato risolto collegando il dongle Zigbee tramite una prolunga usb (probabilmente, ponendo il dongle Zigbee troppo vicino al RPi, si andava a creare interferenza con il chip WiFi di quest'ultimo).

Collegare i sensori

Una volta avviato, il software è molto triviale: basta porre un qualsiasi dispositivo Zigbee in modalità *pairing* e quest'ultimo si conserverà automaticamente al coordinatore da noi creato. Un esempio si può vedere in figura 3.1: abbiamo connesso 2 sensori, uno di temperatura e umidità e un altro per la qualità dell'aria.

Inoltre, per ogni dispositivo collegato, abbiamo anche attivato l'`availability` nelle impostazioni dello stesso (così potremmo avere informazioni sullo stato dei sensori).

Messaggi MQTT

Ad ogni messaggio Zigbee ricevuto dai sensori, Zigbee2MQTT pubblicherà i relativi dati ricevuti nel topic `zigbee2mqtt/FRIENDLY_NAME`, dove `FRIENDLY_NAME` è l'indirizzo IEEE del dispositivo (quello a 64-bit), o se definito, il nome impostato da noi nella dashboard. Le informazioni sull'`availability` saranno disponibili sul topic

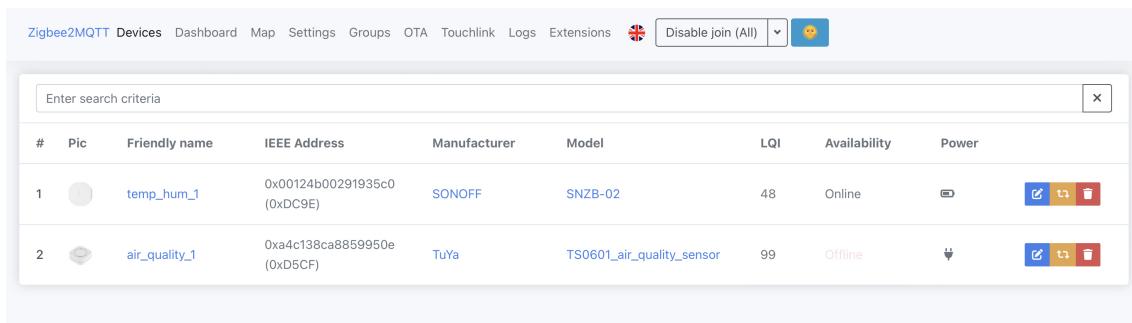


Figura 3.1: Esempio di schermata principale con 2 sensori connessi

`zigbee2mqtt/FRIENDLY_NAME/availability`. Di seguito un esempio di **payload** (ovvero il messaggio MQTT) di un sensore di temperatura:

```
{
  "temperature": 27.23 ,
  "humidity": 53.26
}
```

Per tutte le tipologie di messaggi MQTT inviati si può consultare la guida ufficiale (www.zigbee2mqtt.io/guide/usage/mqtt_topic_and_messages.html)

3.2 Persistenza dati e visualizzazione

A questo punto abbiamo a disposizione i dati dei sensori tramite il protocollo MQTT. Dobbiamo trovare il modo di salvarli per poi visualizzarli in modo appropriato.

3.2.1 InfluxDB

Per quanto riguarda la persistenza dei dati abbiamo scelto InfluxDB: un **time-series database** open-source.^[15]

La differenza principale tra database di questo tipo e comuni db relazionali (come MySQL) è che i primi sono ottimizzati per salvare coppie timestamp-valore, e per questo motivo prendono il nome di **Time-Series Databases**, o TSDB.

Uno dei vantaggi rispetto a tradizionali database relazionali, ad esempio, è la possibilità di comprimere i dati a seconda della distanza temporale (e.g. si acquisisce un dato da un sensore ogni 10 minuti. Decidiamo che, per dati più vecchi di un anno, ci basta conservare una media giornaliera dei valori. I database come InfluxDB includono strumenti nativi per fare proprio questo). Altri vantaggi includono:^[16]

- Meno spazio occupato: grazie all'uniformità dei dati su un TSDB, comuni algoritmi di compressione riescono a lavorare molto meglio.

- Schemaless: in generale i TSDB tendono ad essere schemaless, il che li rende molto più versatili, ad esempio, per aggiunte/rimozioni di campi.
- Oltre a comprimere vecchi dati è possibile anche impostare un ts-dbms per cancellare, ad esempio, tutti i dati più vecchi di un anno.

Il linguaggio di querying di InfluxDB è **Flux**, nativo proprio a questo database. Per fortuna, una volta installato, il database dispone di un'interfaccia web che, fra tante cose, ci permette anche di generare query in questo linguaggio tramite un'interfaccia grafica.

Installazione

Per l'installazione abbiamo seguito la guida ufficiale, disponibile all'indirizzo: <https://portal.influxdata.com/downloads/>. Una volta installato, analogamente al software precedente, abbiamo lanciato: `sudo systemctl unmask influxdb` e `sudo systemctl enable influxdb` in modo da farlo girare come servizio e farlo partire al boot.

Una volta avviato, entriamo nell'interfaccia web per inizializzarlo (ip del rpi, o `server-serra.local`, porta 8086). Nel nostro caso abbiamo impostato username = `admin`, password = `ProgettoSerra2.0`, organization name = `serra` e bucket name = `greenhouse`. Copiamo e salviamo da qualche parte il token che ci verrà fornito (ci servirà in seguito per connetterci al db).

Definizioni

InfluxDB utilizza dei termini diversi rispetto ai dbms relazionali, nonostante il significato sia più o meno il solito:

- **Bucket**: quello che su InfluxDB viene chiamato "bucket" non è altro che un database. Sarà quindi questo a "contenere" tutti i nostri dati. È possibile creare buckets dall'interfaccia web: noi abbiamo creato 1 bucket chiamato `greenhouse`.
- **Field**: questi sono i campi che conterranno i nostri valori. Le tipologie di dati ammessi sono: string, float, integer, o boolean. Ogni value è sempre associato ad un timestamp (ogni field è a sua volta scomposto in field-key e field-value: la key è il nome del campo, value il suo valore).
- **Measurement**: questi sono campi stringa, utili per separare logicamente i dati. Possiamo pensarli come alle tabelle di un normale db relazionale.
- **Tags**: Sono simili ai fields, però hanno un obiettivo diverso: servono per salvare metadati relativi ai valori salvati. In particolare i tags sono indicizzati dal dbms, mentre i fields no.

<u>measurement</u>	<u>field</u>	<u>value</u>	<u>time</u>	<u>topic</u>
group	group	no group	no group	group
string	string	double	dateTime:RFC3339	string
air_quality	temperature	28.6	2023-09-11T19:30:00.000Z	air_quality_1
temperature_humidity	temperature	28.73	2023-09-10T15:30:00.000Z	temp_hum_1
temperature_humidity	temperature	28.76	2023-09-10T16:00:00.000Z	temp_hum_1
temperature_humidity	temperature	28.79	2023-09-10T16:30:00.000Z	temp_hum_1
temperature_humidity	temperature	28.83	2023-09-10T17:00:00.000Z	temp_hum_1

Figura 3.2: Esempio di bucket (topic è un tag)

3.2.2 Node-RED

Dobbiamo ora connettere Zigbee2MQTT con il nostro database. Per fare questo utilizziamo Node-RED, che ci permette di semplificare questo processo, oltre a darcì moltissimi strumenti per future automazioni.

Node-RED è un software di programmazione open-source *low-code*, che ci permette di connettere assieme dispositivi hardware, API e molti servizi. Il paradigma utilizzato da Node-RED è quello della programmazione *flow-based*.

Installazione

Per l'installazione abbiamo seguito la guida disponibile sul sito ufficiale:

<https://nodered.org/docs/getting-started/raspberrypi>. Una volta installato abbiamo abilitato il servizio per farlo partire al boot:

`sudo systemctl enable nodered.service`. Node-RED è ora disponibile sulla porta 1880.

Utilizzo

I **nodi** sono gli elementi di base di node-red, che, interconnessi fra loro, creano i cosiddetti **flussi**. Ogni nodo si può attivare alla ricezione di un messaggio dal nodo precedente o all'avvenire un evento esterno (come una richiesta HTTP, un timer, un interrupt, ecc.). Una volta ricevuto il messaggio (ed elaborato), possono inviarne a loro volta un altro ai nodi successivi. Ogni nodo può avere molteplici outputs, ma un solo input.

Node-RED include di default molti nodi già pronti, ma è comunque possibile svilupparsi i propri. Ad esempio esiste il blocco "function" che permette di scrivere direttamente al suo interno del codice javascript, andandone a definire il comportamento.

Flusso realizzato

Per quanto riguarda il nostro caso, è stato realizzato un *flusso* che permette di prendere i dati da Zigbee2MQTT, elaborarli e salvarli sul database.

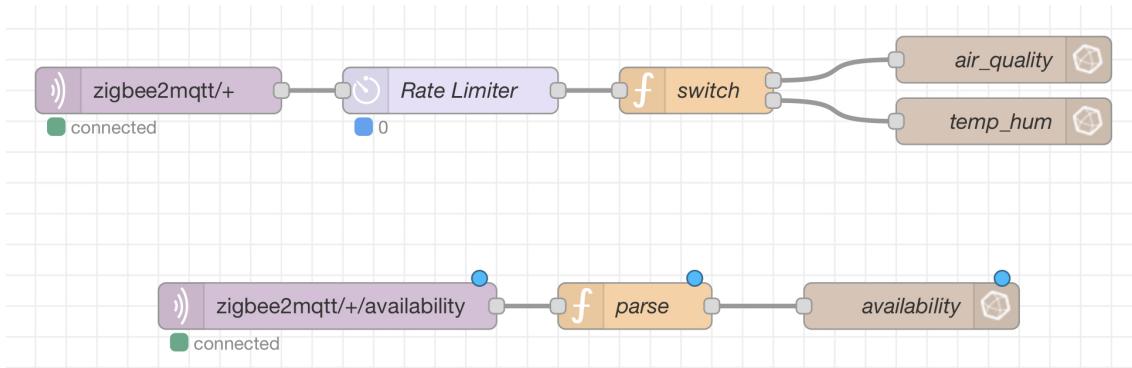


Figura 3.3: Flusso realizzato

Facendo riferimento alla figura 3.3, da in alto a sinistra, i nodi sono:

- **MQTT**: Il primo nodo ci permette di iscriverci ad un topic e ricevere i relativi messaggi. In questo caso ci siamo iscritti al topic `zigbee2mqtt/+`, in modo da ricevere tutti i messaggi inviati da Zigbee2MQTT.
- **Rate Limiter**: Questo nodo ci permette di limitare i messaggi che passano ai nodi successivi, utile ad esempio per evitare spam di dati. Ad esempio, se un sensore invia un dato ogni 5 secondi ma a noi basta un dato ogni 5 minuti ci basta impostare questo nodo al rate "1 msg per 5 minutes" (bisogna impostare anche "Drop intermediate messages" e "Send all topics").
- **Switch**: Questo è un nodo *function*, in cui è stato scritto del codice javascript custom. Ci permette di reindirizzare i messaggi in due diverse tabelle (o meglio *measurements*) in base agli attributi.
- **InfluxDB**: gli ultimi due nodi (`air_quality` e `temp_hum`) ci permettono di scrivere i dati sul database.

Cosa analoga è stato fatto per il secondo flusso in basso: quest'ultimo salva sul measurement `availability` i messaggi ricevuti (e propriamente elaborati) da zigbee2mqtt per quanto riguarda lo stato dei sensori (i.e. se essi sono online o offline).

Nodo MQTT

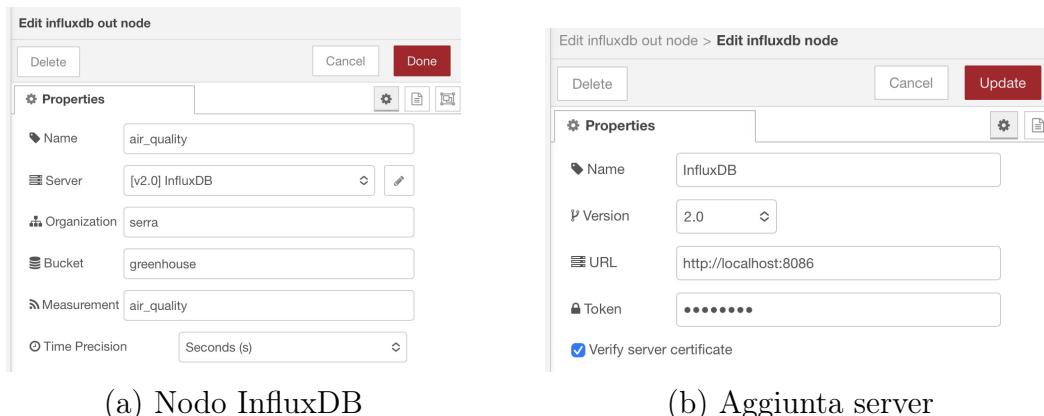
Per quanto riguarda il nodo MQTT, prima di poterlo usare è necessario aggiungere un broker. Per fare questo, bisogna entrare nel nodo (fig. 3.4a) e cliccare sull'icona della penna accanto a "Server": questo aprirà una nuova finestra dove potremo inserire tutti i dati necessari (fig. 3.4b e 3.4c). Nel nostro caso dovremo connetterci al broker disponibile su `localhost:1883` con le credenziali impostate in precedenza (vedi 3.1.3).



Nodo InfluxDB

Questo nodo non è incluso direttamente con l'installazione base di Node-RED e pertanto andrà installato. Ci basta andare nel menu di node-red (dall'interfaccia web in alto a destra), poi cliccare su "manage palette", click su tab "install", cercare influxdb e infine premere su "install". Una volta installato ci resta solo da aggiungere il nostro database, in modo analogo per quanto fatto con il nodo MQTT: dobbiamo entrare nel nodo (fig. 3.5a) e cliccare sull'icona della penna accanto a "Server". Nella finestra che si aprirà (fig. 3.5b) dovremmo inserire i dati relativi al nostro database: <http://localhost:8086> e il token fornитoci in fase di installazione.

L'utilizzo del nodo in se è molto semplice: ci basta inserire organization (**serra**), bucket (**greenhouse**) e measurement.



Nodo switch

Per quanto riguarda il primo nodo function, chiamato *switch*, il codice (sul tab *On Message*³) è quello che segue :

```
const currKeys = Object.keys(msg.payload);
```

³così verrà eseguito ad ogni messaggio ricevuto.

```

const air_quality_keys = ['co2', 'formaldehyd', 'humidity', ,
    'temperature', 'voc'];
const temp_hum_keys = ['humidity', 'temperature'];

// Remove 'elapsed' key so that it won't be saved in the db
if ('elapsed' in currKeys) delete msg.payload.elapsed;

const topic = msg.topic.replace('zigbee2mqtt/' , '');
msg.payload = [msg.payload, { topic }];

// If payload has all air quality fields save to air_quality
// bucket
if (air_quality_keys.every(e => currKeys.includes(e)))
    return [msg, null];
// If payload has not all air quality fields, but has
// temperature and humidity save to temperature_humidity
// bucket
else if (temp_hum_keys.every(e => currKeys.includes(e)))
    return [null, msg];
// If no match, don't save anything
else
    return [null, null];

```

Quello che viene fatto, in pratica, è dividere i messaggi in due categorie: se un messaggio contiene informazioni sulla qualità dell'aria (ovvero contiene le chiavi *co2*, *voc*, etc.) verrà salvato nel measurement *air_quality*, altrimenti nel measurement *temp_hum*: questo è fatto abilitando 2 outputs nel nodo e facendo il return di un array. Inoltre il codice aggiunge ai dati del sensore (che corrispondono a *msg.payload*), un tag in modo da riconoscere da quale sensore provengono: questo è fatto aggiungendo `{ topic }` nel payload (il topic è preso direttamente dal topic mqtt).

Nodo parse

Analogo al nodo switch, però molto più semplificato (non avendo da smistare nessun tipo di informazione). Questo nodo semplicemente aggiunge il tag necessario per individuare il sensore di provenienza:

```

// Extract sensor name from topic
const r = msg.topic.match(/zigbee2mqtt\/(.*\/availability)/);
const name = r?.[1];

if (!name) return null;

msg.payload = [msg.payload, { sensor: name }];
return msg;

```

3.2.3 Grafana

Ci rimane solo da installare un software che ci permetta di visualizzare i dati salvati in modo comodo ed efficiente. La scelta è stata per Grafana, software open-source per la visualizzazione e l'analisi interattiva di dati.

Anche per questo programma seguiamo la guida di installazione del sito ufficiale: <https://grafana.com/tutorials/install-grafana-on-raspberry-pi/>. Una volta avviato sarà disponibile sulla porta 3000 (al primo avvio sarà richiesto un cambio di password, noi abbiamo di nuovo inserito ProgettoSerra2.0).

Andiamo ora a creare delle dashboards, che ci permetteranno di visualizzare i dati dei nostri sensori.

Aggiunta della connessione al database

Prima di poter creare dashboards è necessario definire la connessione al database. Per fare questo dal menu in alto a sinistra clicchiamo su "Connections" e poi cerchiamo "InfluxDB": a questo punto clicchiamo su *Create an InfluxDB data source* e inseriamo i dati necessari.

Creazione di una dashboard

Creiamo una nuova dashboard: clicchiamo sul pulsante "+" (in alto a destra) e poi su "New Dashboard". A questo punto possiamo aggiungere una *Visualization* (e.g. un grafico, tabella, ecc.) indicando la connessione che utilizzeremo per i dati (InfluxDB nel nostro caso).

Adesso possiamo creare un grafico a nostro piacimento: una volta definiti i parametri del grafico (tipo, nome, ecc.) non ci resta che scrivere la query. Per fortuna, come accennato in sezione 3.2.1, InfluxDB ci fornisce un tool grafico per auto-generarla.

Generazione query

Andiamo su InfluxDB (porta 8086) e selezioniamo *Buckets* dal menu laterale. Una volta selezionato il nostro bucket (*greenhouse*) ci ritroveremo una finestra come mostrato in figura 3.6: da qui possiamo selezionare i dati che vogliamo includere nella query. Noi, per esempio, abbiamo selezionato quelli mostrati in figura: ci verrà restituita la query con i dati dell'umidità per il sensore chiamato *temp_hum_1*. Cliccando ora su *Script Editor* sarà possibile vedere la query generata (figura 3.7): copiamola ed incolliamola su Grafana.

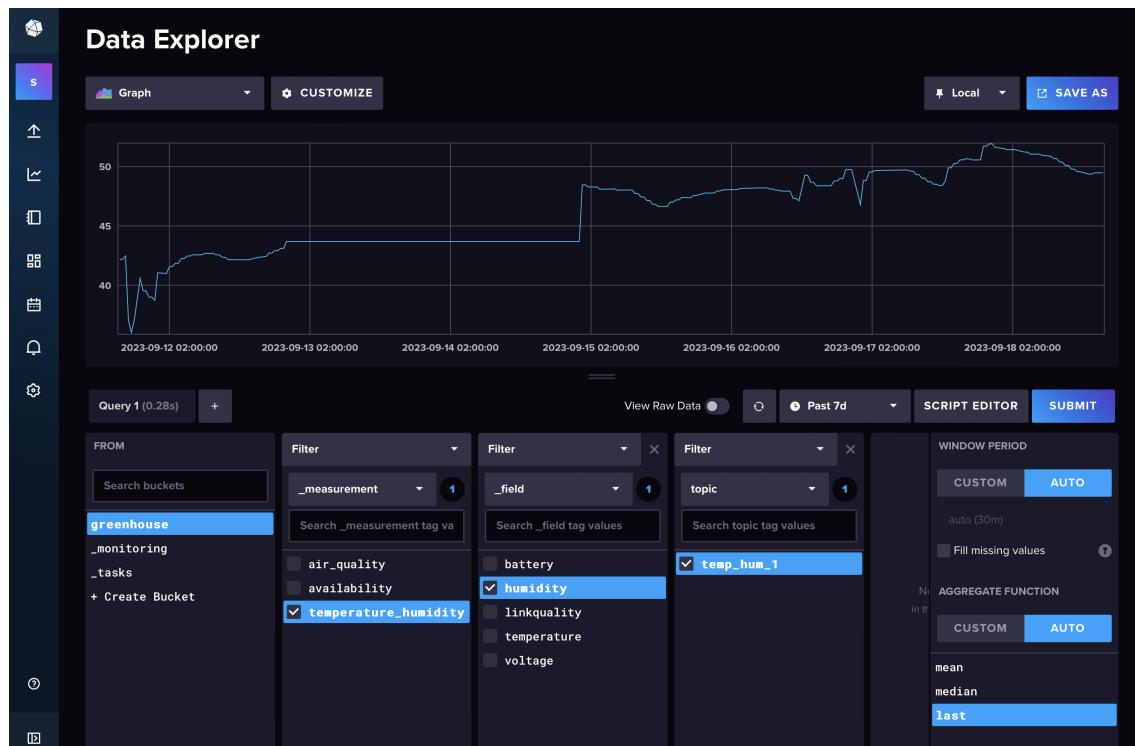


Figura 3.6: Strumento per creazione di query

The screenshot shows the Grafana Query Builder interface. At the top, there's a search bar, a 'View Raw Data' button, a time range selector ('Past 7d'), and 'QUERY BUILDER' and 'SUBMIT' buttons.

The main area displays the generated query code:

```

1 from(bucket: "greenhouse")
2 |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3 |> filter(fn: (r) => r["_measurement"] == "temperature_humidity")
4 |> filter(fn: (r) => r["_field"] == "humidity")
5 |> filter(fn: (r) => r["topic"] == "temp_hum_1")
6 |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
7 |> yield(name: "last")

```

To the right of the code editor is a sidebar with tabs for 'Transformations' (selected), 'Functions', and 'Variables'. The 'Transformations' tab lists functions like 'aggregate.rate', 'chandeMomentumOscillator', 'columns', 'cov', 'covariance', 'cumulativeSum', 'date.hour', and 'date.microsecond'. The 'Functions' tab lists 'Filter Functions...', 'Variables', and 'Functions' (underlined). The 'Variables' tab is currently empty.

Figura 3.7: Esempio di query generata

Per poter creare dashboard riutilizzabili, possiamo utilizzare una funzionalità di Grafana che ci permette di definire delle variabili valide sull'intera dashboard: andiamo nelle impostazioni di quest'ultima (icona dell'ingranaggio) e clicchiamo su *Variables*, da qui definiamo una nuova costante chiamata "topic" e impostiamone il valore (per esempio) a `temp_hum_1`.

Torniamo ora sulla query del grafico precedentemente realizzato e sostituiamo "`temp_hum_1`" con `${topic}`, in modo da utilizzare la variabile appena creata. Ora, se vogliamo creare la stessa dashboard per più sensori di temperatura ci basterà clonarla e cambiare il nome del sensore fra le variabili.

Dashboard realizzate

In figura 3.8 e 3.9 sono mostrate le dashboard realizzate.

I *gauges* mostrano l'ultimo valore disponibile dal sensore (ovvero il valore attuale), mentre i restanti grafici ne mostrano l'andamento storico. È presente anche una *text box* con lo stato (online/offline) del sensore.



Figura 3.8: Esempio di dashboard per un sensore nel measurement `temp_hum`



Figura 3.9: Esempio di dashboard per un sensore nel measurement *air_quality*

3.3 Schema riassuntivo

In figura 3.10 si può vedere uno schema riassuntivo dell’architettura.

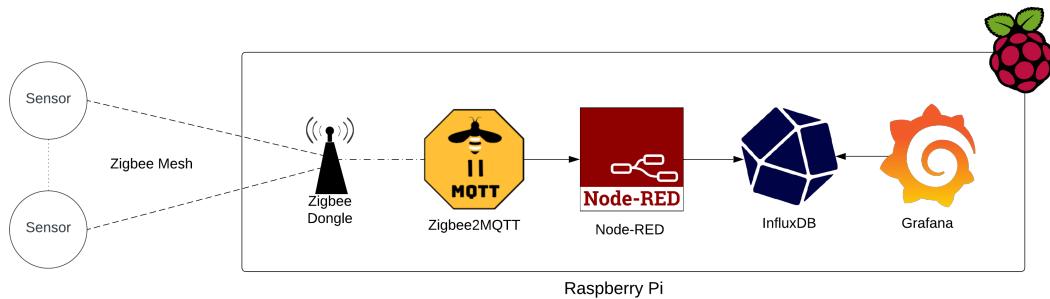


Figura 3.10: Diagramma riassuntivo dell’architettura

3.4 Landing page

Per semplificare l’utilizzo dei vari servizi, è stata creata anche una *landing page* per il server (ovvero il Raspberry Pi), contenente tutti i link ai vari applicativi.

Prima di poter scrivere la pagina dobbiamo installare un *web server*, come Apache:

1. Installazione Apache 2: da un terminale lanciare `sudo apt install apache2 -y`
2. Creazione pagina: `sudo vim /var/www/html/index.html`

In figura 3.11 è illustrata la pagina realizzata.

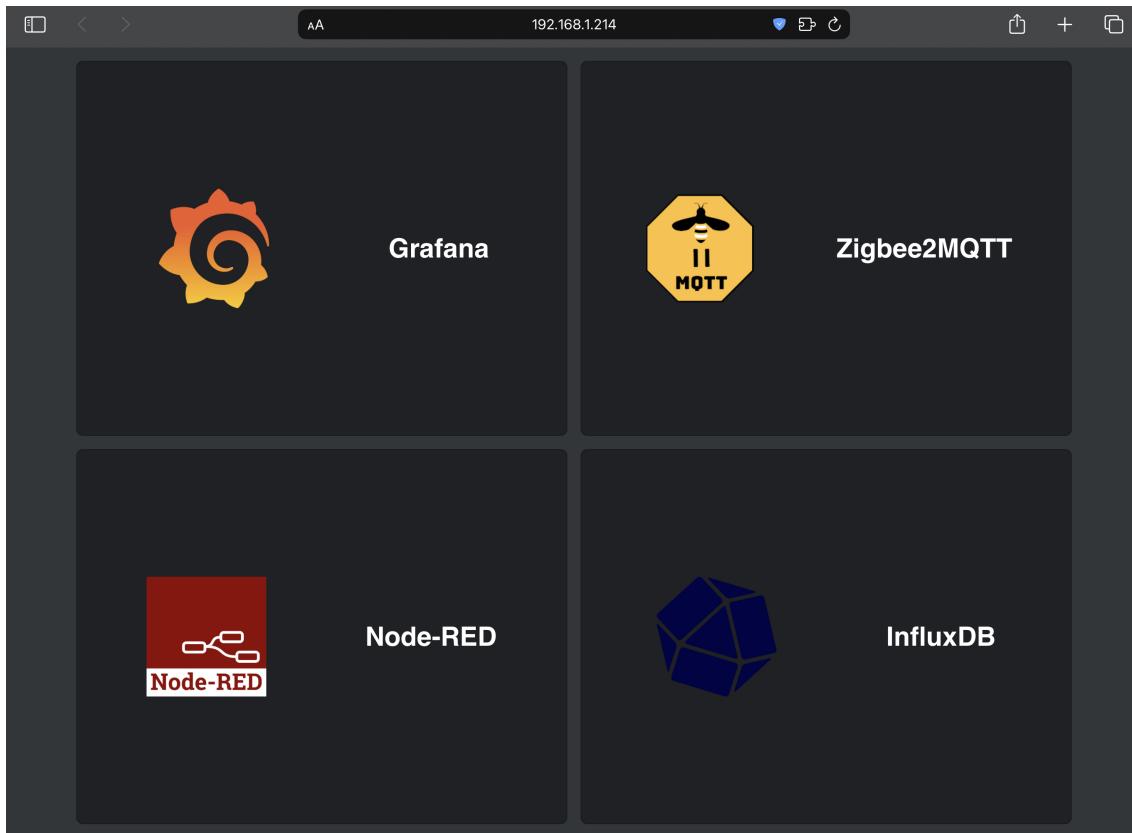


Figura 3.11: Landing page del server

3.5 Display MQTT

Per concludere, si vuole realizzare anche un display *smart* per sostituire gli schermi dei termometri attuali.

Vista la natura della nostra rete, dobbiamo costruire un dispositivo che sia in grado di interagire con il sistema creato: per questo motivo, si è deciso di utilizzare la tecnologia Wi-Fi con il protocollo MQTT. È stata utilizzata questa tecnologia al posto di Zigbee perché per il nostro caso d'uso è stato molto più semplice implementarla.

L'idea è quindi quella di creare un display connesso nel quale sia possibile scrivere qualsiasi tipo di informazione tramite un messaggio MQTT. Per inviare i messaggi utilizzeremo Node-RED.

Componenti e circuito

Come accennato vogliamo utilizzare la tecnologia Wi-Fi, per fare questo possiamo affidarci ad una board basata sul chip **ESP8266**⁴, nel nostro caso una *Wemos D1*

⁴Chip *Arduino-compatible* con capacità Wi-Fi

Mini. Per quanto riguarda il display è stato scelto un semplice LCD 16x2 con connessione I₂C⁵.

In figura 3.12 è illustrato il diagramma del circuito realizzato.

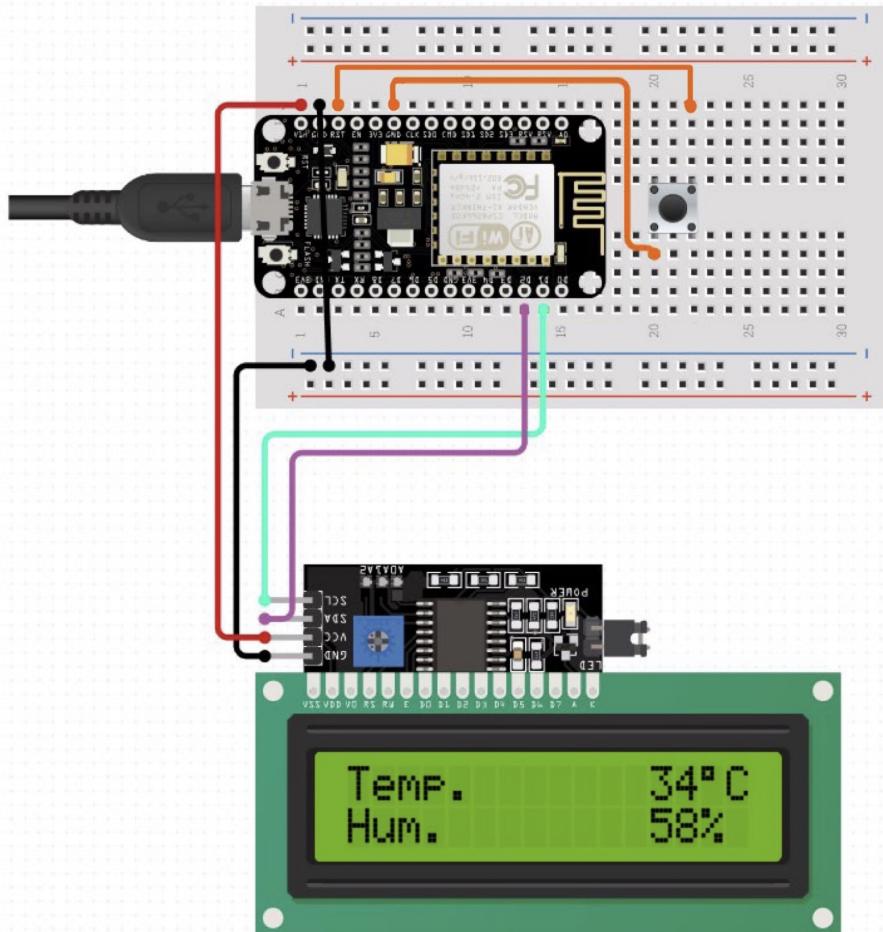


Figura 3.12: Diagramma del circuito

È stato anche collegato un pulsante per poter svegliare il dispositivo dallo stato di *deep sleep* (vedi sezioni successive).

Codice

Il progetto è stato strutturato in 3 parti:

1. Gestione dell'LCD
2. Connessione Wi-Fi e MQTT
3. Definizione del comportamento

⁵Bus di comunicazione seriale sincrono, multi-master/multi-slave ampiamente utilizzato per collegare circuiti integrati

Cominciamo dal primo, di seguito possiamo vedere il codice in questione:

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F, 16, 2); // set the LCD address to
// 0x27 for a 16 chars and 2 line display

void initLCD() {
    lcd.init();
    lcd.backlight();
}

char* replace_char(char* str, char find, char replace) {
    char *current_pos = strchr(str, find);
    while (current_pos) {
        *current_pos = replace;
        current_pos = strchr(current_pos, find);
    }
    return str;
}

void removeChar(char *str, char garbage) {
    char *src, *dst;
    for (src = dst = str; *src != '\0'; src++) {
        *dst = *src;
        if (*dst != garbage) dst++;
    }
    *dst = '\0';
}

/**
 * Print the given string to the LCD display
 * @param str The string to print
 * @param lcdLines The number of lines of the display to use (
 * default 2)
 */
void printToLCD(char* str, int lcdLines = 2) {
    char* res[2];
    byte i = 0;

    // Split string by \n or \r\n
    char* ptr = strtok(str, "\r\n");
    while (ptr) {
        res[i++] = ptr;
        ptr = strtok(NULL, "\r\n");
        if (i == lcdLines) break; // max 2 lines on display
    }

    // Replace the degree symbol with the right code for the
    // display (c interprets the degree symbol as two values:
    -62,-80)
```

```

removeChar(str, -62);
replace_char(str, -80, 223);

// Print to display
lcd.clear();
lcd.setCursor(0,0);
for (int j = 0; j < i; j++) {
    lcd.print(res[j]);
    lcd.setCursor(0,j+1);
}
}
}

```

Questo file contiene una funzione comoda per la stampa su display: ricevuta una stringa, stampa ogni riga sullo schermo (limitando il numero di righe a quelle del display).

È stato necessario anche fare degli accorgimenti per la stampa del simbolo del grado (°), visto che il metodo nativo `print()` dell'LCD non riesce a riconoscere il rispettivo `char`.

Passiamo ora al secondo punto, ovvero la gestione della connessione Wi-Fi e MQTT: di seguito i file relativo (esso richiama anche un secondo file chiamato `secrets.h` che contiene le credenziali per il broker mqtt e per la rete Wi-Fi).

```

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include "secrets.h"
#include "lcd.h"

// MQTT broker address and port
const char *MQTT_HOST = "server-serra.local";
const int MQTT_PORT = 1883;

const char *MQTT_CLIENT_ID = "Display 1";
const char *TOPIC = "displays/1";

// -----
WiFiClient client;
PubSubClient mqttClient(client);

/**
 * Initializes the MQTT client, connects to the broker and
 * subscribes to the topic.
 * When a message is received, the callback function will be
 * called.
 */
void initMQTT(MQTT_CALLBACK_SIGNATURE) {
    // Connect to WiFi (and wait until connected)
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

```

```

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    lcd.print(".");
}
lcd.clear();

mqttClient.setServer(MQTT_HOST, MQTT_PORT);
mqttClient.setCallback(callback);

// Wait for MQTT to connect
while (!client.connected()) {
    if (mqttClient.connect(MQTT_CLIENT_ID,
                           MQTT_USER,
                           MQTT_PASSWORD)) {
        lcd.clear();
    }
    else {
        delay(500);
        lcd.print(".");
    }
}

mqttClient.subscribe(TOPIC);
}

void MQTTLoop() {
    mqttClient.loop();
}

```

Il metodo `initMQTT()` (oltre a connettersi alla rete WiFi indicata) si iscrive al topic impostato nella costante `TOPIC`, e per ogni messaggio ricevuto, eseguirà la funzione di callback passata come parametro.

Le librerie usate per entrambe le sezioni sono quelle incluse fra parentesi angolate all'inizio di ogni file. È possibile installarle graficamente tramite l'IDE di Arduino (con il quale compileremo anche il codice). Tratteremo questo argomento più avanti.

Concludiamo con il file principale, che unisce i due precedenti: definisce la funzione di callback e il comportamento per il *deep sleep* (il file incluso fra virgolette è l'ultimo visto):

```

#include <Arduino.h>
#include "mqtt.h"

void callback(char *topic, byte *payload, unsigned int length)
{
    payload[length] = '\0';
    printToLCD((char *)payload);
}

```

```

}

const int SLEEP_DELAY = 5000;
const int LCD_GND_PIN = 2;

void setup() {
    // Init display
    pinMode(LCD_GND_PIN, OUTPUT);
    digitalWrite(LCD_GND_PIN, LOW);

    initLCD();
    initMQTT(callback);
}

void loop() {
    MQTTLoop();

    // Deep sleeps after SLEEP_DELAY milliseconds
    if (millis() > SLEEP_DELAY) {
        digitalWrite(LCD_GND_PIN, HIGH); // Turn off display
        ESP.deepSleep(0);
    }
}

```

Come si può vedere dal codice la funzione di callback stampa la stringa ricevuta da MQTT sul display LCD. Dopo un delay, il microprocessore viene posto in uno stato di *deep sleep* in modo da consumare pochissima energia (circa $20\mu\text{A}$), così che possa essere alimentato anche a batteria. Per riaccenderlo basterà premere il pulsante: al click viene collegato il pin RST (reset) a GND e quando questo avviene il microcontrollore si resetta, ripartendo dalla funzione di setup.

Librerie e compilazione

Per la compilazione abbiamo usato l'IDE di Arduino⁶ che è compatibile con il chip usato, dobbiamo però aggiungere la specifica dell'esp8266 nelle impostazioni dell'editor: aprire *File > Preferences > Additional Boards Manager* e incollare https://arduino.esp8266.com/stable/package_esp8266com_index.json. Una volta fatto questo ci resta solo da installare la scheda: cerca e installa *esp8266* da *Tools > Boards > Boards Manager*. Per concludere dobbiamo installare le librerie necessarie: da *Sketch > Include Library > Manage Libraries* cercare ed installare PubSubClient e LiquidCrystal_I2C.

Per quanto riguarda la compilazione, abbiamo utilizzate le impostazioni illustrate in figura 3.13 (sottomenu *Tools*).

⁶<https://www.arduino.cc/en/software>

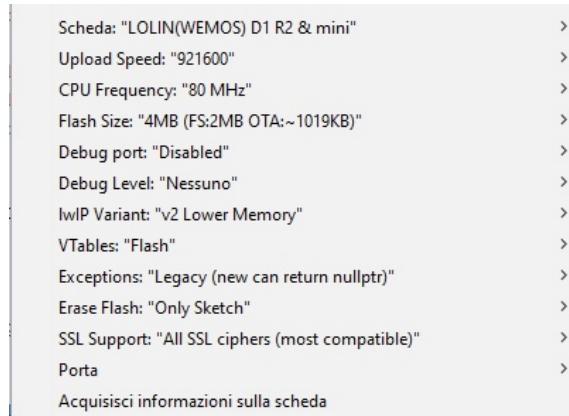


Figura 3.13: Impostazioni di compilazione

Flusso Node-RED

Come già accennato utilizziamo Node-RED per inviare i messaggi MQTT. Il flusso da realizzare quindi deve semplicemente inviare (publish) dei messaggi stringa al topic a cui si è iscritto il nostro esp8266.

In figura 3.14 è illustrato il flusso realizzato.

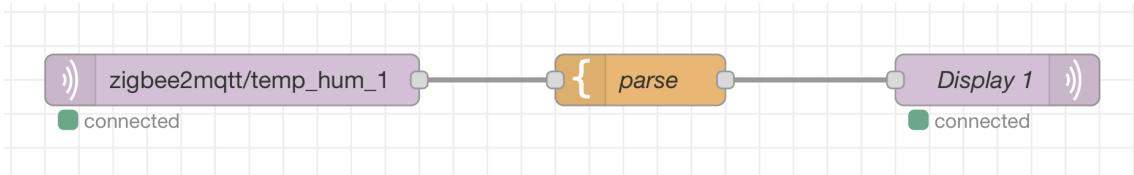


Figura 3.14: Flusso Node-RED

Da sinistra (fig. 3.14), in ordine:

- **Nodo MQTT:** riceve i messaggi da un topic, in questo caso dal sensore di temperatura *temp_hum_1*
- **Parse:** Crea la stringa da inviare (nel nostro esempio quella mostrata nel display in figura 3.12). In particolare questo nodo ci permette di interpolare l'input dello stesso in una stringa. Il contenuto è il seguente:

```

Temp:      {{payload.temperature}}°C
Hum:       {{payload.humidity}}%
  
```

- Infine abbiamo un altro **nodo MQTT**, questo invia il messaggio ricevuto in input sul topic impostato (che sarà lo stesso a cui è iscritto l'esp8266). In particolare abbiamo impostato *Retain* a **true**: così facendo questo topic mqtt

"conserva" l'ultimo messaggio (e quindi appena qualcuno si iscrive riceverà subito l'ultimo valore del sensore).

Concludendo, basterà accendere il dispositivo: appena connesso ad una rete Wi-Fi e al broker MQTT mostrerà sul display i dati del sensore di temperatura indicato. Ovviamente, il flusso creato è solo un esempio e l'oggetto creato è molto versatile: basterà cambiare flusso per cambiare la tipologia di messaggio mostrato.

Ad esempio, per future estensioni, è possibile aggiungere un secondo pulsante sull'esp8266 che invia un messaggio MQTT al broker, e tramite Node-RED utilizzare questa informazione per "scorrere" fra tutti i sensori disponibili.

Capitolo 4

Conclusioni

Questa tesi ha affrontato il problema del monitoraggio delle serre dell'*Orto Botanico "Giardino dei Semplici"* di Firenze attraverso la creazione di una rete di sensori intelligenti basata su tecnologie come Zigbee e MQTT.

I risultati ottenuti durante il corso di questa ricerca dimostrano chiaramente l'efficacia di tale approccio, apportando una serie di benefici:

1. **Monitoraggio dettagliato:** la rete di sensori riesce a fornire dati dettagliati su temperatura, umidità e altri parametri ambientali in punti chiave delle serre, consentendo una comprensione più approfondita delle condizioni ambientali.
2. **Storico dati:** a differenza dei sistemi precedenti, la soluzione proposta permette di archiviare e accedere ad uno storico dati: questo è essenziale per analizzare le tendenze nel tempo e prendere decisioni basate su dati passati.
3. **Facilità di accesso ai dati:** i dati dei sensori sono stati resi facilmente accessibili e visualizzabili tramite un'interfaccia utente intuitiva, migliorando notevolmente la fruibilità delle informazioni raccolte.
4. **Scalabilità e flessibilità:** il sistema è stato progettato in modo scalabile, consentendo l'aggiunta futura di ulteriori sensori o funzionalità senza dover ridisegnare l'intera infrastruttura.

Inoltre, la scelta di utilizzare standard e protocolli aperti, oltre a software open-source, ha permesso di evitare la dipendenza da fornitori o piattaforme proprietarie, garantendo la durata e la flessibilità del sistema nel tempo.

In conclusione, questa tesi ha dimostrato che l'applicazione di concetti IoT alle serre dell'Orto Botanico, attraverso la creazione di una WSN, è una soluzione efficace per il monitoraggio ambientale. La creazione di un sistema scalabile, basato su standard aperti, offre opportunità significative per migliorare la gestione e la comprensione degli ambienti delle serre.

Bibliografia

- [1] Connectivity Standards Alliance, “Zigbee faqs.” [Online]. Available: <https://csa-iot.org/all-solutions/zigbee/zigbee-faq/>
- [2] “CSA Memebers.” [Online]. Available: <https://csa-iot.org/members/>
- [3] [Online]. Available: https://www.digi.com/resources/documentation/Digidocs/90002002/Content/Reference/r_zb_stack.htm
- [4] “802.15.4 - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).” [Online]. Available: <https://people.iith.ac.in/tbr/teaching/docs/802.15.4-2003.pdf>
- [5] “Radio Regulations 2020, CHAPTER II – Frequencies, ARTICLE 5 Frequency allocations, Section IV – Table of Frequency Allocations.” [Online]. Available: <https://search.itu.int/history/HistoryDigitalCollectionDocLibrary/1.44.48.en.101.pdf>
- [6] “IEEE 802.15.4 Stack User Guide.” [Online]. Available: <https://www.nxp.com/docs/en/user-guide/JN-UG-3024.pdf>
- [7] A. Realis-Luc, “Localizzazione in reti di sensori zigbee,” 2009. [Online]. Available: <https://www.alus.it/pubs/LocationService/Tesi.pdf>
- [8] ZigBee Alliance, “Zigbee specification,” 2017. [Online]. Available: <https://csa-iot.org/wp-content/uploads/2022/01/docs-05-3474-22-0csg-zigbee-specification-1.pdf>
- [9] [Online]. Available: https://www.researchgate.net/figure/ZigBee-network-topologies-51_fig2_23307553
- [10] D. I. Inc., “An introduction to zigbee,” 2008. [Online]. Available: <https://ftp1.digi.com/support/documentation/zigbeeintro.pdf>
- [11] “Mqtt official website.” [Online]. Available: <https://mqtt.org/>
- [12] “Mqtt essentials.” [Online]. Available: <https://www.hivemq.com/mqtt-essentials/>

- [13] [Online]. Available: <https://hlassets.paessler.com/common/files/infographics/mqtt-architecture.png>
- [14] [Online]. Available: <https://www.zigbee2mqtt.io>
- [15] [Online]. Available: <https://github.com/influxdata/influxdb>
- [16] [Online]. Available: <https://www.influxdata.com/blog/relational-databases-vs-time-series-databases/>