POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Ingegneria del Cinema e dei Mezzi di Comunicazione

Tesi di Laurea Magistrale

Progetto e realizzazione di un sistema IoT per la raccolta e l'elaborazione di dati legati ad una filiera produttiva

Relatori
prof. Giovanni Malnati
prof. Fabio Forno
firma dei relatori

. . . . . .
. . . . . .

Candidato
Sofia Caterini

firma del candidato

. . . . . .

Anno Accademico 2020-2021

Non sono le cose che non sai
a metterti nei guai.
È quello che dai per certo
che invece non lo è.

[Mark Twain]

Abstract

Il monitoraggio dei dati legati ad una filiera produttiva sta assumendo sempre più un ruolo fondamentale per le aziende che vogliono dimostrare la propria attendibilità sul mercato e che necessitano di analizzare in profondità le informazioni provenienti dai diversi partner che operano al proprio interno. Ciò è reso possibile dall'introduzione dello standard EPCIS (Electronic Product Code Information Services), definito dall'ente no profit GS1, che consente ai partner commerciali di condividere informazioni sul movimento fisico e sullo stato dei prodotti mentre viaggiano lungo tutta la catena di approvvigionamento, da un'azienda all'altra e infine ai consumatori. L'acquisizione di tali dati EPCIS interessa non solo l'azienda stessa e i propri partner commerciali che vogliono esaminare i prodotti, ma anche il consumatore nel momento in cui deve effettuare l'acquisto.
La cattura, quindi, di informazioni relative allo stato e all'ambiente in cui i prodotti si formano ha un'importanza significativa per molteplici stakeholders.

Questa tesi si pone l'obiettivo di rilevare dati significativi relativi ai contesti produttivi ed ai prodotti ed integrarli, attraverso l'utilizzo di un sistema IoT (Internet of Things), con la piattaforma commerciale iChain, ambiente software basato sullo standard EPCIS, che permette la collaborazione tra aziende diverse attraverso l'interscambio di dati logistici e produttivi. Il sistema sperimentale progettato e realizzato è costituito da un insieme di dispositivi fisici capaci di essere connessi fra di loro e con Internet in maniera da effettuare alcune rilevazioni e azioni nell'ambiente circostante.
Questo sistema prevede l'implementazione di sensori, integrati all'interno di schede di sviluppo, che rilevano le informazioni ambientali circostanti e che creano un'ampia rete comunicando fra di loro e trasmettendo i dati registrati al di fuori di essa. Tali informazioni vengono inserite in iChain, attraverso un apposito endpoint.

Dopo un'osservazione dello stato dell'arte e un'attenta analisi dei protocolli di comunicazione per la trasmissione dati fra i device, si è formata l'idea dell'architettura generale di tale sistema, con particolare attenzione alla sua facilità di configurazione e messa in campo in ambienti complessi come quelli in cui avviene la produzione industriale e la logistica.

A questo scopo è stata implementata una Mesh Bluethooth Low Energy in grado di scambiare flussi di dati con un Gateway. I dispositivi all'interno della Mesh ricevono comandi dal Gateway, come ad esempio il settaggio dei parametri relativi alla frequenza di pubblicazione dei rilevamenti e trasmettono al Gateway i valori che i sensori hanno acquisito. Il Gateway è collegato ad un computer, in modo da trasmettere e ricevere informazioni grazie ad un'app Seriale. Quest'applicazione permette sia di inviare i comandi ai nodi sia di inviare tramite MQTT i valori riguardanti le misure dei sensori e lo stato dei nodi verso iChain. In particolare questi messaggi sono ricevuti da un'applicazione Consumer la quale cattura le informazioni dei sensori, le integra con quelle relative alla tracciabilità del prodotto, le codifica in modo standard EPCIS e le inserisce in iChain.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Definizione dell'obiettivo

L'obiettivo finale di tale lavoro di tesi è la realizzazione di un sistema IoT in grado di raccogliere e successivamente elaborare alcuni dati relativi ad una filiera produttiva.
Nel percorso di un prodotto lungo la propria catena di approvvigionamento questo si interfaccia con informazioni riguardanti il proprio stato e l'ambiente che lo circonda. Tali dati sono importanti non solo per l'azienda stessa che lo produce e i suoi partner commerciali ma anche per il consumatore che lo dovrà acquistare.

Un sistema IoT[1] è un insieme di dispositivi in grado sia di comunicare tra di loro che con l'ambiente che li include tramite Internet. Tali device, che nel nostro caso devono acquisire dati, prevedono l'utilizzo di alcuni sensori che rilevano i valori e di schede di sviluppo che sono in grado di poter elaborare quanto appreso.

Per la progettazione di tale sistema si è resa necessaria un'analisi preliminare approfondita. Sono state quindi varate le principali tecniche di trasmissione, le schede di sviluppo e le toolchain che permettono la realizzazione del progetto nel modo più semplice ed ottimale.

1.2 Analisi dello scenario

Le motivazioni che hanno portato alla necessità di elaborare tale progetto sono riconducibili all'introduzione, negli ultimi anni, di uno standard commerciale che

[1]Internet of Things o Internet delle Cose è un termine introdotto circa 20 anni fa da Kevin Ashton ed indica il processo di connessione a Internet di oggetti fisici.

permette di condividere informazioni sul movimento fisico e sullo stato dei prodotti in tempo reale mentre viaggiano lungo tutta la catena produttiva: EPCIS (Electronic Product Code Information Services). Tale standard è stato introdotto da GS1, un'organizzazione che sviluppa e mantiene standard globali per la comunicazione tra imprese. In seguito verrà dedicata un'intera sezione del Capitolo 2 per descrivere in dettaglio la struttura e le specifiche di EPCIS.

La creazione del sistema descritto in tale documento, formato da dispositivi dotati di sensori, ha rilevanza nel momento in cui quanto appreso viene successivamente inserito in tempo reale all'interno di iChain in modo standard mediante eventi standardizzati EPCIS. iChain è una piattaforma per le aziende che raccoglie e mostra tutte le informazioni relative ad un prodotto lungo la Supply Chain. In iChain vengono quindi inseriti i dati rilevati dal sistema IoT mediante lo standard EPCIS. Tale piattaforma viene descritta dettagliatamente all'interno del Capitolo 2.

1.3 Struttura del documento

Tale documento è articolato nei seguenti capitoli:

- Capitolo 2 - Stato dell'arte: in questo capitolo viene esaminato lo scenario attuale, vengono quindi descritti EPCIS e iChain;
- Capitolo 3 - Implementazione obiettivo: descrive le analisi preliminari riguardanti i protocolli di comunicazione effettuate per determinare la scelta dell'architettura generale;
- Capitolo 4 - Specifiche Bluetooth: analizza il protocollo Bluetooth e in particolare la mesh BLE, architettura che è risultata essere ottimale per il nostro caso specifico;
- Capitolo 5 - Strumenti e Software: in questo capitolo sono elencati gli strumenti Hardware e Software che hanno permesso di progettare l'intera applicazione;
- Capitolo 6 - Progettazione e realizzazione: descrive gli step più importanti per il raggiungimento del lavoro finale ovvero l'implementazione della rete in cui comunicano tra di loro i device e la progettazione del gateway nella sua interezza;
- Capitolo 7 - Conclusioni: analizza il prodotto risultante nei suoi punti di forza e punti deboli. Descrive il livello raggiunto dal sistema progettato ed elenca

quali migliorie potrebbero essere effettuate su tale lavoro per incrementarne le prestazioni;

- Capitolo 8 - Appendice: in tale sezione sono presenti i datasheet delle board utilizzate, per comprenderne al meglio le loro caratteristiche e capacità.

Capitolo 2

Stato dell'arte

Prima di iniziare a descrivere in che modo è stato progettato e realizzato tale lavoro di tesi, occorre descrivere le motivazioni per le quali c'è stata la necessità di affrontare tale argomento. In tale capitolo verrà quindi fatta un'accurata analisi della situazione attuale.
L'introduzione infatti dello standard EPCIS da parte di GS1 ha portato alla necessità di tener traccia delle informazioni legate ai prodotti e acquisite da essi lungo la filiera produttiva. Il fine ultimo di questo progetto è infatti la cattura dei dati rilevati dal sistema IoT per essere inseriti sotto forma di eventi EPCIS.
Occorre inoltre dedicare una sezione ad IChain, una piattaforma per le aziende nella quale vengono raccolte tali informazioni standardizzate infatti EPCIS.

2.1 EPCIS

In tale sezione viene analizzato il dettaglio lo standard EPCIS, da chi è stato implementato, quali sono le sue caratteristiche ed i suoi benefici nel business globale.

2.1.1 GS1

GS1 Italy è un'associazione che riunisce oltre 35 mila imprese di beni di largo consumo e da più di 40 anni si occupa di sviluppare e mantenere gli standard più usati al mondo per la comunicazione tra aziende, ossia i sistemi standard GS1. Tra di essi il più conosciuto è il codice a barre, che è ritrovabile in qualsiasi prodotto da acquistare.

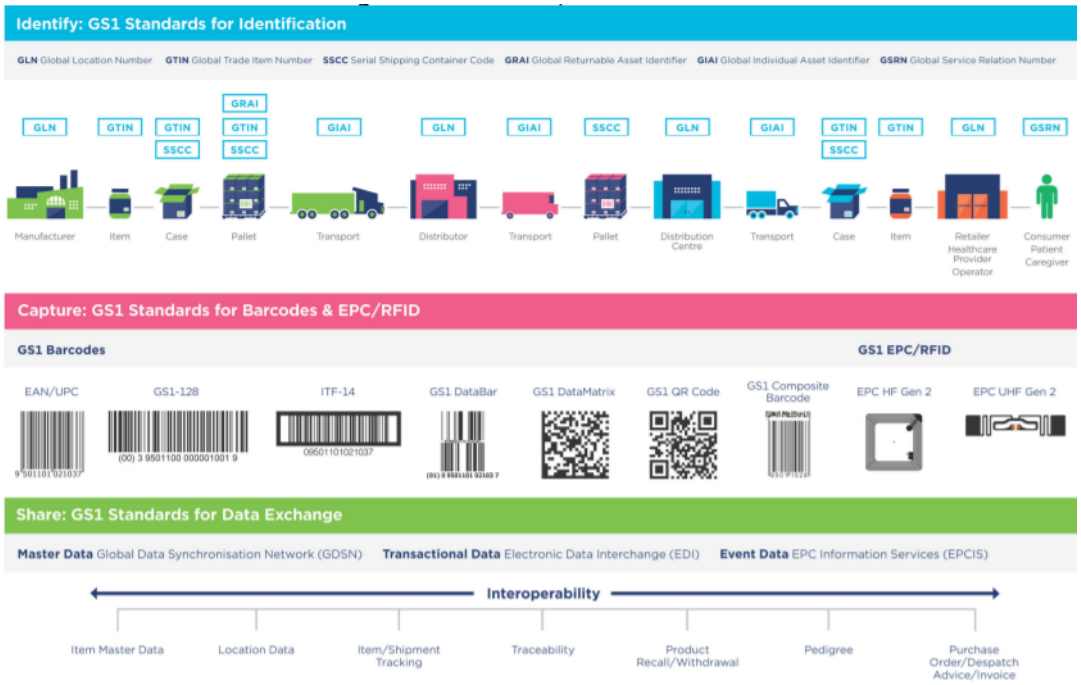


Figura 2.1: Sistema degli standard GS1

2.1.2 Sistema di codifica e EPC

Il sistema degli standard GS1 è descritto in figura 2.1. La sua architettura si articola in tre strati: identificazione, cattura e condivisione delle informazioni.
Lo strato d'identificazione, che nell'immagine è evidenziato in azzurro, rappresenta gli oggetti e i soggetti di una generica filiera produttiva. La catena di approvvigionamento va dal manufacturer al consumatore e ogni attore all'interno di essa viene identificato con delle chiavi di identificazione.
Il secondo strato invece annovera le simbologie di codici a barre e RFID (tag in radiofrequenza), utilizzati per trasportare in modo automatico le informazioni del primo strato.
L'ultimo strato, in verde nella figura, rappresenta gli standard legati alla condivisione delle informazioni e in esso rientra lo standard EPCIS.
Il sistema GS1 si basa sul sistema delle chiavi di identificazione. Queste devono essere uniche, non significative, internazionali, sicure e multisettoriali. Servono per identificare tutti gli oggetti e i soggetti della catena, come ad esempio il prodotto (GTIN Global Trade Item Number), l'unità logistica (SSCC Serial Shipping Container Code) e i luoghi (GLN Global Location Number). Questi codici servono

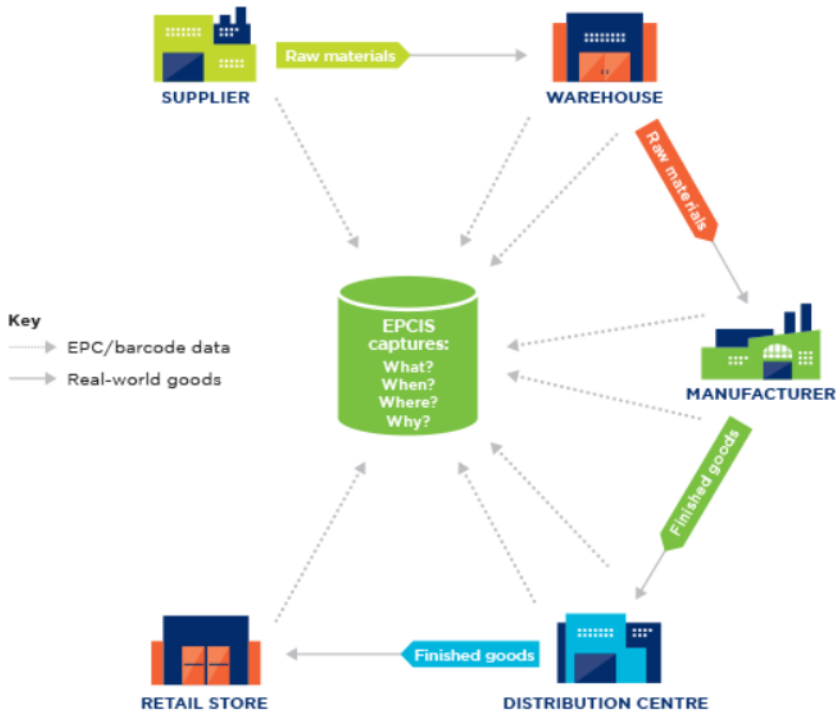


Figura 2.2: Standard EPCIS

per accedere ai sistemi informativi in cui sono contenute tutte le informazioni registrate.

L'EPC (Electronic Product Number) è un codice presente nel secondo strato dell'architettura degli standard GS1. Questo è l'ID number che identifica un oggetto ed è globalmente univoco, ovvero definisce in maniera unica un singolo pezzo a livello mondiale, è indipendente dal supporto, quindi esiste a prescindere dallo strumento che lo veicola e rappresenta una chiave di accesso per informazioni supplementari presenti su database. L'EPC identifica quindi in modo univoco i singoli prodotti, rileva le informazioni importanti delle merci lungo la supply chain e le rende disponibili grazie alla radiofrequenza (RFID).
Il codice EPC è creato dal GTIN che è già relativo al prodotto, con le regole dello standard, e viene poi aggiunto di un numero seriale.
Quindi da GTIN più numero seriale, viene ricavato il codice EPC da inserire in un tag RFID. L'EPC può essere ricreato anche sulla base di altre chiavi di identificazione GS1 come ad esempio SSCC o GRAI.

2.1.3 Standard EPCIS

L'Electronic Product Code Information Services è uno standard GS1 che definisce le interfacce per la cattura e la richiesta delle informazioni relative ad un prodotto che si sposta lungo la filiera produttiva e delinea un modello di dati per gli eventi di visibilità tramite file JSON. L'EPCIS è uno standard open ed è stato approvato come ISO/IEC 19987 a riprova del ruolo fondamentale riconosciutogli. Esso abilita soluzioni e servizi di tracciabilità e può acquisire informazioni da diversi data-carrier. Osservando l'immagine 2.2 è evidente come l'EPCIS acquisisce informazioni lungo la filiera produttiva da ogni attore, dal supplier fino al retail store. Tale codice è quindi una struttura dati che consente di catturare in modo standard tutte le informazioni relative agli eventi. EPCIS può acquisire informazioni mediante tag EPC/RFID o barcode.
La cattura degli eventi lungo la filiera produttiva viene descritta dall'immagine 2.3. In essa è evidente che per ogni evento vengono acquisite 4 informazioni, what, where, when e why, che vengono catturate nel database dell'attore di riferimento. L'architettura EPC Global fornisce delle regole per riuscire ad interfacciare tutti questi database contenenti informazioni per EPCIS in modo tale da ricostruire la filiera e fornire attraverso delle applicazioni specializzate le informazioni relative alla tracciabilità e alla rintracciabilità al consumatore finale.
Le quattro informazioni che vengono registrate ad ogni evento EPCIS sono:

- What: Chi è il protagonista dell'evento. Può essere un GTIN (prodotto) o un SSCC (unità logistica) o un GRAI (asset utilizzabile);
- Where: Attore presso il quale l'evento ha avuto luogo;
- When: Quando è avvenuto l'evento, descritto da un time stamp;
- Why: Fase di business alla quale l'oggetto era sottoposto.

2.1.4 Benefici di impiego

L'utilizzo di EPCIS ha numerosi benefici. Primo fra tutti è il miglioramento della visibilità e trasparenza lungo la filiera produttiva, che è fondamentale per l'azienda stessa in quanto rende rintracciabile una qualsiasi falla all'interno della catena di approvvigionamento e fondamentale per migliorare l'attendibilità nei confronti del consumatore e delle aziende partner.
Un altro vantaggio nell'impiego di questo standard è il monitoraggio del prodotto, sia a livello di pezzo che di lotto.
L'ultimo importante beneficio è l'incremento dell'accuratezza dei processi che a sua volta reca migliorie alla gestione dell'inventario, alla riduzione di errori di

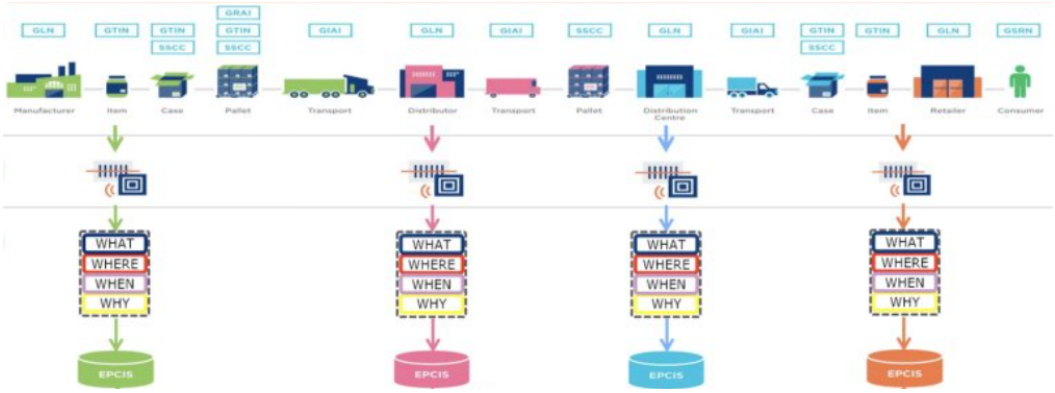


Figura 2.3: EPCIS: Cattura eventi lungo la filiera

fatturazione, al miglioramento dell'ordine e all'incremento dell'efficienza operativa attraverso i vari processi di business.

2.2 iChain

iChain è una piattaforma digitale creata dalla Xeliontech che fornisce servizi e strumenti digitali per le aziende ed è raggiungibile al sito `https://xeliontech.com/it/XTAP`. Essa permette alle aziende di mettersi in comunicazione e di scambiare fra di loro dati in maniera efficiente e veloce. Tali informazioni sono relative a prodotti ed eventi e grazie a queste rilevazioni permette di monitorare i processi produttivi delle aziende di una filiera. Al suo interno utilizza lo standard EPCIS. È un'infrastruttura digitale Cloud IoT per la Supply Chain Transparency che utilizza tecnologie avanzate come Big Data e IoT, seguendo il modello di Industry 4.0.
I servizi di iChain sono fruibili da tutte le filiere produttive e distributive che vogliono raggiungere obiettivi di eccellenza, collaborando con i partners di filiera, traendo vantaggio competitivo dalla condivisione dei dati, dalle relazioni interaziendali e dall'ottimizzazione dei processi di filiera.
Come descritto anche nella propria Homepage i vantaggi di tale piattaforma sono:

- Tracciabilità e rintracciabilità delle merci, utile non solo per l'azienda stessa ma anche per i propri partner;
- Trasparenza e gestione delle relazioni all'interno della filiera;
- Visione d'insieme e al dettaglio della produzione;
- Interoperabilità grazie allo Standard GS1;

- Analisi dei dati a scopo di indagine e predittivo grazie ad alcuni indicaori KPI;
- Sicurezza e immutabilità delle informazioni;
- Grafica intuitiva ed avanzata.

Nel contesto di questo lavoro di tesi iChain fa ingestion dei dati raw che gli vengono trasmessi dal sistema IoT, li analizza, li rielabora e li rende disponibili fino all'utente finale, utilizzando lo standard EPCIS.

2.3 Problematiche legate alla raccolta dei dati digitali

Sebbene molti aspetti del processo produttivo siano già controllati digitalmente, spesso tali controlli avvengono grazie a piattaforme proprietarie, integrate con gli impianti stessi, da cui si estraggono con molta complessità le informazioni (perché i produttori non sono interessati a dischiudere le informazioni raccolte e hanno interesse a creare un lock in dell'azienda). Inoltre, spesso una singola azienda dispone di impianti produttivi differenti, forniti da aziende diverse, che raccolgono dati non necessariamente omogenei tra loro.

In questo quadro, diventa importante poter integrare le informazioni disponibili con altre, prelevate dal campo, e debitamente correlate ai processi produttivi in corso. Poiché i costi di impianto possono risultare elevati, è interessante disporre di soluzioni wireless, che tuttavia soffrono di problemi di affidabilità e complessità di configurazione. Da qui l'interesse per lo studio delle reti ad hoc.

Capitolo 3

Implementazione obiettivo

Per poter progettare ed implementare una rete ad hoc è necessaria un'analisi approfondita delle tecnologie in grado di creare il sistema nel modo più efficace ed efficiente. Il sistema IoT da realizzare deve essere sicuro, conveniente ed efficiente in termini di consumi.

In tale Capitolo vengono quindi analizzati i principali protocolli di comunicazione. Le caratteristiche sottoposte a un'attenta e accurata valutazione sono state:

- Area di copertura;
- Velocità di trasmissione;
- Prezzo dei device;
- Sicurezza;
- Numero di nodi supportati;
- Gestione e configurazione dei nodi;
- Consumo batteria.

3.1 Confronto protocolli di comunicazione

Le tecnologie analizzate sono state suddivise in LPWAN (Low Power Wide Area), Short Range, WiFi e Cellular. Tra le LPWAN vi sono LoRa, Sigfox e Narrow Band IoT. Tra le Short Range sono state esaminate ZigBee, WiFi, Bluetooth e Bluetooth Low Energy. Le tecnologie di trasmissione cellulari non sono poi state analizzate in dettaglio in quanto non sono state ritenute adatte a tale scenario a causa dell'utilizzo di una sim e dell'elevato costo di implementazione.

3.1.1 LPWAN

LPWAN è l'acronimo di Low Power Wide Area Network e comprende le tecnologie capaci di creare reti con un'ampia area di copertura e con un basso consumo energetico. Tra le principali soluzioni di questo tipo vi sono LoRa (Long Range), Sigfox e Narrow Band IoT, una nuova tecnologia cellulare.
Nella tabella 3.1 vengono confrontati i principali protocolli di trasmissione LPWAN con focus su range, velocità, sicurezza e autenticazione e consumo di batteria.
In seguito sono analizzate in dettaglio tutte le tecnologie prese in esame.

	LoRa	**Sigfox**	**NB-IoT**
Cellulare	No	No	Si
Spettro	Unlicensed	Unlicensed	Licensed
Range, km	Urbano: 2-5 Rurale: 15	Urbano: 3-10 Rurale: 30-50	Urbano: 1-5 Rurale: 10-15
Velocità, uplink /downlink	50Kbitps/ 50Kbitps	50Kbitps/ -	250Kbitps/ 250Kbitps
Consumo di batteria	***	*	*
Sicurezza	**	**	***
Immunità interferenze	***	***	*
Autenticazione e cifratura	Si	Si	No
Localizzazione	Si	Si	No
Num max messaggi giornalieri	Illimitati	140	Illimitati
Prezzo device e connessioni	**	*	**

Tabella 3.1: Confronto protocolli di trasmissione LPWAN

3.1.1.1 LoRa

Specifiche e area di copertura. LoRa, Long Range, è una tecnica di modulazione ideata da Semtech che garantisce una comunicazione wireless molto più competitiva rispetto alle più comuni reti cellulari. Grazie a trasmissione sicura dei dati, portata di circa 15 chilometri, capacità di oltre 1 milione di nodi e basso consumo di energia, seppur a discapito della ridotta velocità di trasmissione che è al massimo 50 kbps, questa tecnologia sta guadagnando notevole diffusione nelle

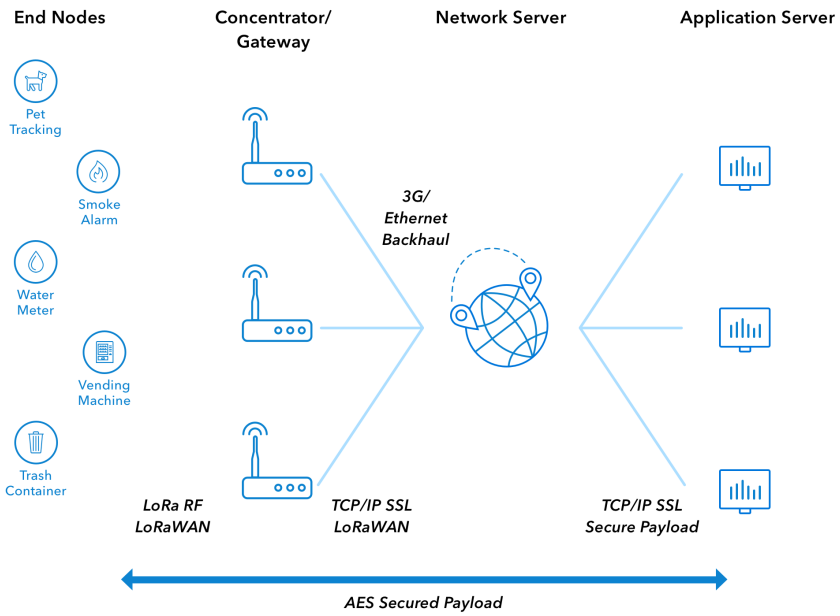


Figura 3.1: Architettura di rete LoRaWAN

reti Internet of Things.
La modulazione LoRa lavora su bande libere ISM[1] di 433 MHz, 868 MHz (Europa) o 915 MHz (Nord America) a seconda della regione in cui viene sviluppata; tali frequenze sono meno utilizzate rispetto a quella di 2.4 GHz e ciò comporta minori interferenze e minor attenuazione del segnale.

Architettura di rete. LoRa descrive il livello fisico che può quindi essere implementato su differenti topologie di reti e protocolli come ad esempio Mesh, Star o 6lowPAN.
LoRaWAN ne definisce il livello MAC (Media Access Control). È stato standardizzato da LoRa Alliance e prevede specifiche su comunicazione e architettura di rete.
La rete LoRaWAN, come si può vedere dalla figura 3.1 ha una topologia a stella ed è costituita da quattro elementi:

[1]Industrial Scientific and Medical, si tratta di specifiche bande di frequenze regolarmente assegnate dal piano di ripartizione nazionale ed internazionale non per uso commerciale, ma industriale, scientifico e medico.

- Gli End Nodes che raccolgono i dati dei sensori, li trasmettono a monte, ricevono le comunicazioni dal server funzionando da transceiver. Utilizzano la comunicazione wireless single-hop con uno o più Gateway;

- Il Gateway che funziona da ponte trasparente e trasporta i dati bidirezionalmente tra gli End Device e il Server;

- Il Server di rete che si connette a più Gateway tramite una connessione TCP/IP protetta, cablata o wireless;

- L'Application Server che raccoglie e analizza i dati dai nodi e determina le azioni.

Sicurezza. Un'ulteriore caratteristica implementata da LoRaWAN è lo stato di sicurezza della rete. Il protocollo utilizza infatti la crittografia AES[2] a 128 bit e dispone di molteplici layer indipendenti di protezione.
Ha infatti più chiavi:

- una chiave di sessione di rete, la Network Session Key, usata dal Network Server e dal dispositivo per creare il MIC (Message Integrity Code) per controllare l'integrità del messaggio e per criptarne e decriptarne il contenuto;

- una chiave di sessione di applicazione, l'Application Session Key, che è univoca per ogni device di rete e viene usata per criptare e decriptare il payload dei messaggi di applicazione ed è utilizzata dal nodo e dall'Application Server.

Ogni device ha inoltre due indirizzi:

- il DevAddr (Device Address) che identifica il nodo in modo univoco all'interno della rete e che è formato da 32 bit, di cui i sette più significativi identificano la rete mentre i rimanenti sono assegnati dal Network Server in modo casuale;

- l'AppEUI (Application Identifier) che identifica il proprietario dell'applicazione ed è salvato all'interno del dispositivo. Questo è composto da 64 bit.

Gestione dei device. Affinchè un nodo possa essere configurato all'interno di una rete LoRaWAN vi sono due metodologie:

[2]Advanced Encryption Standard è un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America.

- L' Activation by Personalization (ABP) nel quale Network Session Key, Application Session Key, Device Address e AppEUI sono già inseriti e memorizzati all'interno di ogni device di rete.

- L'Over the Air Activation (OTA) è invece una procedura in cui tutte le informazioni di rete vengono assegnate e non sono possedute già dai device. In tale metodo ogni nodo invia un Join Request Message, contenente il DevEUI e l'AppEUI e un Device Token Random di 2 byte, al Network Server che a sua volta lo manderà ad un Join Server. Quest'ultimo memorizza i Device Token random dai precedenti Join Requests Message da ciascun End Node. Solo se il Join Server è in grado di autenticare la combinazione degli identificativi contenuti nel messaggio, genera un Device Address, un Network ID ed Application token univoci. Questi parametri vengono mandati dal Network Server all'End Node in un comando Join Response. Solo adesso l'end node può considerarsi all'interno della rete e generare le proprie chiavi di sicurezza.

Queste due metodoglogie hanno un differente grado di sicurezza, la seconda prevede infatti un invio delle chiavi e questo potrebbe rendere la rete più soggetta ad attacchi.

3.1.1.2 Sigfox

Specifiche e area di copertura. Anche Sigfox, come LoRa, appartiene alle tecnologie LPWAN. È una tecnologia proprietaria francese che offre copertura capillare su quasi 50 paesi in tutto il mondo e opera alla frequenza di 868 MHz in Europa e di 902 MHz in America. Permette la trasmissione di 140 messaggi al giorno con un payload di 12 byte, e la ricezione di 4 messaggi al giorno con un payload di 8 byte prevedendo il pagamento di una quota di abbonamento ed è pensata per lo sviluppo di applicazioni pervasive in scenari IoT. Oltre alla tecnologia di trasmissione offre anche l'uso di piattaforme Cloud proprietarie per la gestione dei dati. Nel caso migliore può raggiungere oltre 100 km di distanza tra trasmettitore e ricevitore (stazione base) ma in scenari reali giunge fino a pochi km (città) o decine di km (campagna) a seconda della topologia e dell'ambiente.
In Italia la rete SigFox è resa disponibile esclusivamente da Nettrotter, un'azienda di telecomunicazioni italiana, e ancora non copre tutto il territorio italiano. Inoltre i costi del servizio possono essere copiosi se si considera che ogni singolo device può costare all'anno 14 euro. Inoltre la tariffa per ottenere la certificazione "Sigfox Ready", che è necessaria per creare un nuovo sistema locale da connettere alla rete, è alta: dai 2.500 euro per dispositivi che utilizzano un modulo già certificato, fino ai 10 mila euro.

Architettura di rete. Come si può vedere dall'immagine 3.2 i principali componenti di una rete SigFox sono: dispositivi, stazioni base, Sigfox Cloud e Customer IoT. Lo scambio di informazioni attraverso la rete avviene in questo modo:

1. Il dispositivo invia un messaggio con una richiesta bidirezionale;
2. Le stazioni base acquisiscono il messaggio;
3. Sigfox Cloud autentica il messaggio e raggruppa i duplicati;
4. Il Cloud invia il messaggio al customer IT;
5. Il customer IT manda un messaggio di risposta;
6. Sigfox CLOUD cerca la stazione base più vicina per trasmettere la risposta;
7. La stazione base manda il messaggio;
8. Il dispositivo riceve il messaggio di risposta.

La rete Sigfox si può quindi pensare come una rete a stella in cui tutti i device comunicano col Cloud.

Sicurezza. Ogni messaggio è firmato con una chiave unica associata al dispositivo. I messaggi possono essere crittografati o codificati e nessuna chiave viene scambiata in rete. Con ogni messaggio, viene calcolato un hash[3] e inviato utilizzando:

- ID del dispositivo;

[3]Funzione crittografica che mappa sequenze di bit o stringhe in altre sequenze di bit o stringhe correlate a quelle in input.

Figura 3.2: Architettura rete Sigfox

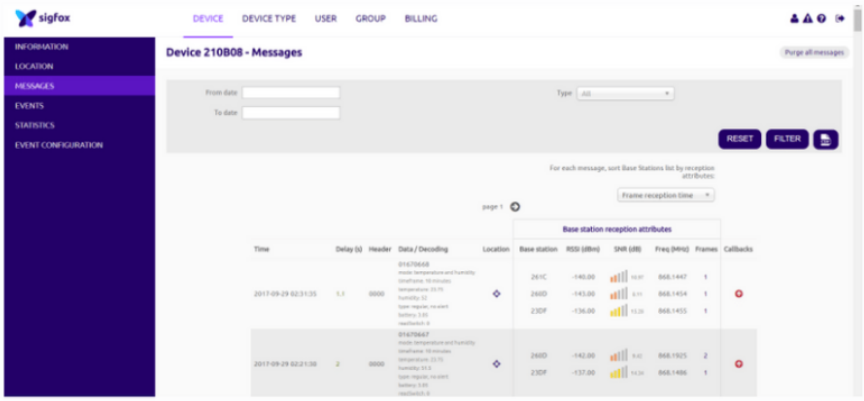


Figura 3.3: Gestione device Sigfox dalla pagina Sigfox Backend

- Chiave segreta unica per il dispositivo;
- Payload;
- Incremento interno.

Viene utilizzato l'algoritmo di crittografia Advanced Encryption Standard (AES) 128.

Gestione dei device. Ogni dispositivo ha un identificatore univoco chiamato Device ID. È anche correlato a un altro numero univoco chiamato PAC (Porting Authorization Code). Il PAC dimostra la proprietà di un dispositivo e solo l'attuale proprietario del dispositivo lo conosce. La coppia ID-PAC è obbligatoria per la registrazione o il trasferimento del dispositivo. Non appena il dispositivo viene registrato o trasferito, il PAC cambierà.
Per la gestione e configurazione dei dispositivi è necessario registrarsi alla pagina Sigfox Back-End[4]. In tale pagina è possibile configurare i device della rete, osservare quelli attivi e i messaggi ricevuti ede effettuare molteplici altre operazioni, come descritto nella pagina `https://support.databoom.com/hc/it/articles/115005208265-Sigfox`. Ciò rende la rete Sigfox facilmente manutenibile.
In figura 3.3 è mostrata la pagina Sigfox Backend per la gestione dei device in cui vengono mostrati tutti i nodi della rete e le loro caratteristiche.

[4] `https://backend.sigfox.com/auth/login`

3.1.1.3 Narrow Band-IoT

Specifiche e area di copertura. NB-IoT (Narrow Band-IoT) appartiene alle tecnologie LTE (LTE Cat. NB1) per soluzioni IoT ed è considerata una tecnologia LPWAN. È pensata per avere bassi consumi e quindi dispositivi alimentati a batteria. Opera sulla frequenza degli 800 MHz, banda licenziata che non richiede autorizzazioni per essere sfruttata e integra le caratteristiche principali delle classiche reti cellulari LTE 2G, 3G e 4G. NB-IoT ha il vantaggio di poter coprire aree molto vaste e di essere estremamente efficiente dal punto di vista energetico: i dispositivi che si appoggiano a essa possono operare per 10 anni senza bisogno di dover sostituire le batterie. Una caratteristica molto apprezzata in ambito industriale è la densità di dispositivi gestibili da una singola cella.

Architettura di rete. La rete è una rete cellulare, quindi una rete mobile che permette la comunicazione grazie alla presenza di stazioni radio base che dividono l'area di copertura in celle. La presenza di segnale non dipende da chi crea la rete di dispositivi a livello locale e quindi ciò può interferire negativamente se non vi è copertura.

Sicurezza. La possibilità di sfruttare le frequenze licenziate migliora l'affidabilità e la sicurezza delle comunicazioni e assicura la corretta allocazione delle risorse utili per gestire al meglio la Quality of Service (QoS). Il bit rate a bordo cella è nell'ordine di 1 kbps, con la possibilità di arrivare fino a 250 kbps in zone a buona copertura.

3.1.2 Short Range

In tale sezione vengono analizzati i protocolli che raggiungono range di copertura meno estesi. Una maggior estensione della portata può essere comunque raggiunta dal collegamento hop to hop di più device. Nella tabella 3.2 vengono messe a confronto le principali tecniche di trasmissione short range. Successivamente vengono elencati i punti salienti di ciascun protocollo di questo tipo ovvero ZigBee, Wifi, Bluetooth e Bluetooth Low Energy.

3.1.2.1 ZigBee

Specifiche e area di copertura. ZigBee è un protocollo di comunicazione basato sullo standard IEEE 802.15.4 per Wireless Personal Area Network (WPAN) e opera nelle frequenze ISM di 868MHz in Europa, 915MHz negli Stati Uniti e 2.4GHz nella maggior parte del resto del mondo. Ha una portata che va dai 10 ai 100 metri circa e riesce a raggiungere al massimo una velocità di trasmissione

	ZigBee	**WiFi**	**Bluetooth**	**BluetoothLE**
Standard	IEEE 802.15.4	IEEE 802.11	IEEE 802.15.1	IEEE 802.15.1
Frequenza di trasmissione	868 MHz, 915 MHz, 2,4 GHz	2,4 e 5 GHz	2,4 Ghz	2,4 Ghz
Range, m	10-100	100	10	100
Data rate	20, 40 e 250 Kbitps	11 e 54 Mbitps	1-3 Mbitps	1 Mbitps
Sicurezza	128-bit AES	SSID Autenticazione	64-128 bit	128 bit AES
Topologia rete	Mesh, Star e Cluster Tree	Star e point to point	Piconet	Piconet, Broadcast e Mesh
Costo	*	***	***	*
Numero di nodi	65000	32	8	8
Vita della batteria	Da mesi ad anni	Ore	Giorni	Da mesi ad anni

Tabella 3.2: Confronto protocolli di trasmissione short range

di 250Kbs a 2.4Ghz e inferiore alle altre frequenze. La ridotta velocità di trasmissione gli consente però di ottenere consumi molto limitati. Tale specifica prevede l'implementazione di 3 tipologie di nodi:

- ZigBee Coordinator (ZC) – Sono i nodi che hanno la maggior capacità di contenere e trasmettere informazioni. Sono i dispositivi di capacità più elevata, formano la radice in una rete ad albero e possono interfacciarsi con altre reti. Essi memorizzano le chiavi di sicurezza, assegnano gli indirizzi agli altri nodi, definiscono l'identificativo della rete e selezionano il canale da utilizzare nella trasmissione. Hanno anche un ruolo fondamentale nei confronti di tutti gli altri nodi in quanto assegnano loro gli indirizzi, permettono loro di unirsi o di abbandonare la rete e memorizzano all'interno di una lista tutti i nodi vicini. All'interno di una rete ZigBee può esserci un unico Coordinator;

- ZigBee Router (ZR) - Sono i nodi che hanno la capacità di inoltrare i dati

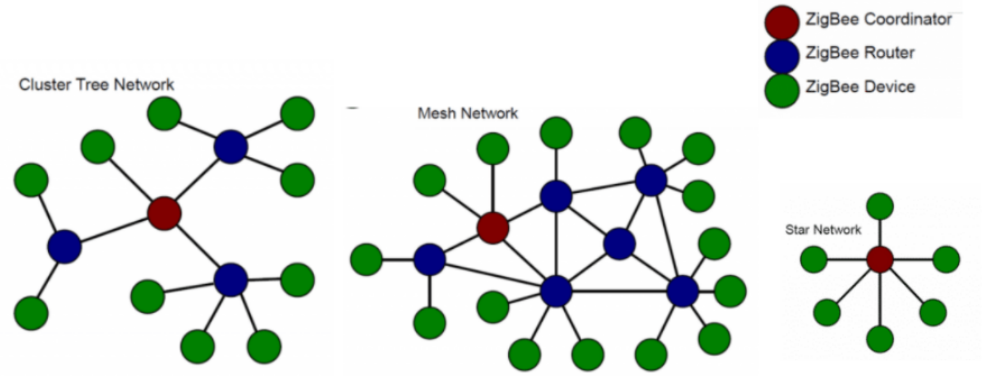


Figura 3.4: Architettura rete ZigBee

agli altri nodi fungendo appunto da router immediato. Questi possono essere più di uno all'interno della rete e permettono a quest'ultima di incrementare la propria area di trasmissione;;

- ZigBee End Device (ZED) – Come dice anche il nome, questi sono i nodi foglia della rete. Hanno poca quantità di memoria e quindi sono meno costosi. Hanno la possibilità di trasmettere informazioni solo con i nodi parenti (Coordinator o Router) e non possono quindi dialogare con dispositivi dello stesso tipo. All'interno di una rete ce ne possono essere molti.

Architettura di rete. Come si può vedere nella figura 3.4 la rete ZigBee può assumere tre diverse tipologie: Cluster Tree, Mesh e Star. La rete a stella, data la portata al massimo di 100 m tra device, non permette di raggiungere distanze molto elevate. All'interno di essa possiamo notare la sola presenza di un Coordinator e di molteplici End Device. Nelle reti del tipo Cluster Tree e Mesh invece sono presenti anche i device Router, che infatti permettono di ampliare il range di copertura. La differenza tra queste due topologie è che nella rete mesh i Router sono messi in comunicazione fra loro.

Sicurezza. La sicurezza di reti Short Range è maggiore rispetto a quella delle reti LPWAN. Per quanto riguarda lo standard Zigbee è garantita dallo scambio di più chiavi crittografate all'interno della rete. Vi sono infatti due chiavi, la Link Key e la Network Key. La Link Key è la chiave che viene scambiata tra due dispositivi, è una chiave AES a 128 bit ed è utilizzata per la comunicazione unicast. La Network Key, come dice il nome, è la chiave di rete ovvero la chiave che viene utilizzata per le comunicazioni broadcast e quelle a livello di rete. Viene condivisa

tra tutti i device della rete. Anch'essa è una AES a 128 bit.
La sicurezza dell'intera rete dipende infatti da queste due chiavi che devono essere inizializzate e installate in maniera sicura. Inoltre essa è incrementata dalla presenza del Trust Center, un dispositivo che distribuisce tutte le chiavi e stabilisce e gestisce la politica di sicurezza della rete. Nello standard ZigBee 3.0, inoltre, la sicurezza della rete è accresciuta dal fatto che ogni device possiede una chiave di sicurezza diversa per criptare i messaggi.

Gestione dei device. Una rete Zigbee è comunemente implementata con moduli Xbee. Tali moduli appartengono alla famiglia di moduli radio Digi International implementanti i protocolli IEEE 802.15.4. A seconda del tipo di modulo è possibile implementare architetture di rete peer-to-peer, a stella o mesh e operano alle frequenze di 2.4 GHz e 900 MHz. Questi, per essere inseriti all'interno di una rete, devono prima essere configurati. Tale configurazione può essere effettuata mediante un software dal nome XCTU. Questo rende la rete poco manutenibile. I parametri più importanti sono:

- ID : deve essere lo stesso per tutti i moduli;
- BD (Baud Rate): 9600;
- SH e SL: il serial number alto e basso del modulo;
- DH e DL: il serial number alto e basso del modulo destinatario della connessione.

Il settaggio di tali parametri permette quindi la configurazione dei dispositivi all'interno della rete.

3.1.2.2 WiFi

Specifiche e area di copertura. Il WiFi è la tecnologia di trasmissione più utilizzata per la realizzazione di reti WLAN (Wireless Local Area Network). È definito dallo standard 802.11 ed ha un raggio di trasmissione inferiore ai 100m. Per quanto riguarda la velocità di trasmissione e la frequenza esse dipendono dalla classe dei dispositivi. Infatti possiamo elencare le 5 distinte classi WiFi più conosciute:

- classe a a 54 Mb/s (5 GHz);
- classe b a 11 Mb/s (2,4 GHz);
- classe g a 54 Mb/s (2,4 GHz);
- classe n a 450 Mb/s (2,4 GHz e 5 GHz);
- classe ac a 3 Gb/s (5 GHz).

Architettura di rete. La rete Wi-Fi può essere considerata come una rete cellulare a scala locale in cui gli access point prendono il posto delle stazioni radio base. Essa quindi in termini di topologia può essere riconduciobile ad una rete a stella.

Sicurezza. La sicurezza di una rete WiFi può essere incrementata in vari modi. Criptando i dati sulla rete: la crittografia dei dati wireless impedisce a chiunque possa accedere alla rete di visualizzarli. Sono disponibili diversi protocolli di crittografia per fornire questa protezione. Wi-Fi Protected Access (WPA), WPA2 e WPA3, che utilizzano lo standard AES, crittografano le informazioni trasmesse tra router e dispositivi. WPA3 è attualmente la crittografia più potente. WPA e WPA2 sono ancora disponibili, ma meno efficienti.
Proteggendo il Service Set Identifier (SSID), disabilitandone la trasmissione. Per impedire agli estranei di accedere facilmente alla rete, si può evitare di pubblicizzare l'SSID. Tutti i router Wi-Fi consentono agli utenti di proteggere l'SSID del proprio dispositivo, il che rende più difficile per gli aggressori trovare una rete.
Modificando le password predefinite. La maggior parte dei dispositivi di rete, inclusi i punti di accesso wireless, sono preconfigurati con password di amministratore predefinite per semplificare la configurazione. Queste password predefinite sono facilmente reperibili online e quindi forniscono solo una protezione marginale. La modifica delle password predefinite rende più difficile per gli aggressori accedere a un dispositivo.

3.1.2.3 Bluetooth

Specifiche e area di copertura. Il Bluetooth è uno standard di trasmissione dati per reti WPAN ideato dall'azienda Ericcson nel 1994 e gestito ad oggi dal Bluetooth SIG (Special Interest Group). Fornisce da sempre un metodo economico e sicuro per scambiare informazioni tra dispositivi diversi. Il Bluetooth è uno standard IEEE 802.15.1 che opera nelle bande delle microonde ovvero a 2.45GHz. Il range di copertura del singolo dispositivo è dell'ordine delle decine di metri e dipende dalla classe del dispositivo.
I device infatti possono appartenere a diverse tipologie di classi:

- Classe 1: può raggiungere una distanza in linea d'aria massima di 100 m e una potenza di 100 mW;
- Classe 2: può raggiungere una distanza in linea d'aria massima di 10 m e una potenza di 2.5 mW;
- Classe 3: può raggiungere una distanza in linea d'aria massima di 1 m e una potenza di 1 mW;

Figura 3.5: Architettura rete Bluetooth

- Classe 4: può raggiungere una distanza in linea d'aria massima di 0.5 m e una potenza di 0.5 mW.

Ad ora le versioni di Bluetooth disponibili vanno dalla 1.0 (quella meno evoluta) alla 5.0 (la più recente).

Architettura di rete. L'architettura che presenta include una connessione master-slave. Il master, è il dispositivo che all'interno della rete si occupa della sincronizzazione del clock degli altri dispositivi (slave) e della sequenza dei salti di frequenza. Lo slave, è sincronizzato al master sia in termini di clock che di frequenza. All'interno di una rete ci può essere un solo master e più slave.
L'architettura tipica del Bluetooth classico è la punto-punto, in cui tutti gli slave si connettono all'unico master, come si può osservare nella figura 3.5.

Sicurezza e Gestione dei device Tale protocollo di comunicazione è uno fra i più sicuri sia in termini di cifratura che per quanto riguarda la gestione sicura dei dispositivi che si vogliono connettere alla rete.
Il Bluetooth utilizza l'algoritmo SAFER+ (Secure And Fast Encryption Routine) per autenticare i dispositivi e per generare la chiave utilizzata per cifrare i dati.
Chi si occupa della gestione dei nuovi device è il link manager che ne controlla la connessione e la sicurezza. L'autenticazione avviene con uno schema "challenge-response". Il dispositivo che si vuole far autenticare è il richiedente ("claimant"). Il dispositivo che verifica l'autenticità si chiama verificatore ("verifier"). L'autenticazione ha successo se il verificatore e il richiedente condividono la stessa link key, che è stata generata appositamente per il collegamento tra quei due dispositivi e viene verificata tutte le volte che i due si collegano (finché non viene cambiata). Il Pairing (fase in cui il master scansiona i device attivi nelle vicinanze e invia una richiesta di connessione allo slave) è la procedura di inizializzazione del link, durante il quale viene generata la chiave di inizializzazione, dalla quale è ottenuta

Figura 3.6: Architettura rete BluetoothLE

la chiave segreta del link, La chiave di inizializzazione si ottiene a partire da un PIN, dall'indirizzo bluetooth dei dispositivi, e da un numero casuale. Il PIN deve essere inserito dall'utente con una procedura che dipende dal particolare dispositivo Bluetooth. Una volta che la chiave di inizializzazione è generata, la chiave segreta del link viene generata con procedura mutua di autenticazione all'inizio del collegamento, a partire da un numero casuale che il master spedisce allo slave.

3.1.2.4 Bluetooth Low Energy

Specifiche e area di copertura. Il Bluetooth Low Energy è supportato dalle versioni Bluetooth 4.0 e successive, è stato quindi introdotto all'incirca nel 2010. Rispetto al Bluetooth "classico", il Bluetooth Low Energy ha lo scopo di fornire un consumo energetico e un costo notevolmente ridotto, mantenendo un intervallo di comunicazione simile.
Il BLE ha una velocità di trasmissione ridotta, sui 300 Kbps e con questo standard i dati vengono inviati in pacchetti di dimensioni ridotte (20 byte) e il raggio di copertura si assesta tra i 10 e i 100 metri. Il vero vantaggio della versione Low Energy, come è evidente anche dal nome, è infatti il piccolissimo consumo energetico che, rispetto al Bluetooth "classico" può arrivare ad essere inferiore fino a 100 volte. I dispositivi BLE alimentati a batteria, possono addirittura funzionare per 1/2 anni.

Architettura di rete. Un'altra importante differenza rispetto al Bluetooth classico è la possibilità di implementare altre topologie di rete rispetto alla point to point. Come si può vedere in figura 3.6 infatti possono essere create rei broadcast e mesh. La rete mesh, in particolare, consente alla tecnologia Bluetooth di supportare la creazione di reti di dispositivi affidabili e su larga scala.

3.2 Scelta architettura generale

Attraverso l'analisi dei principali protocolli di comunicazione, esaminati nelle precedenti pagine, è nata l'idea dell'architettura generale del Sistema IoT da implementare in tale progetto. Essa dipende in particolar modo dallo scenario applicativo. Ci sono infatti due principali contesti: un contesto locale e indoor, come ad esempio i capannoni di un'azienda, in cui i device devono comunicare a brevi distanze e un contesto urbano e outdoor in cui la distanza tra i nodi aumenta in maniera significativa. Per questo abbiamo preso in considerazione due tipologie di rete e abbiamo costituito elementi che potessero funzionare indipendentemente dal protocollo.

Nel caso di ambienti ampi i protocolli più idonei sono quelli che costituiscono LPWAN ed fra questi LoRaWAN ha le migliori qualità in termini di sicurezza ed efficienza energetica.
Una rete LoRaWAN si presenta come quella in figura 3.7. E costituita da un Gateway che è connesso ad un PC sul quale è implementato il Network Server e comunica con una topologia a stella con tutti gli altri nodi della rete.

Nel caso di scenari locali e indoor una rete LPWAN è da considerarsi eccessiva in termini di distanze raggiunte e troppo latente per efficienza energetica, per questo la scelta è ricaduta su una Mesh BLE, come si può vedere in figura 3.8.

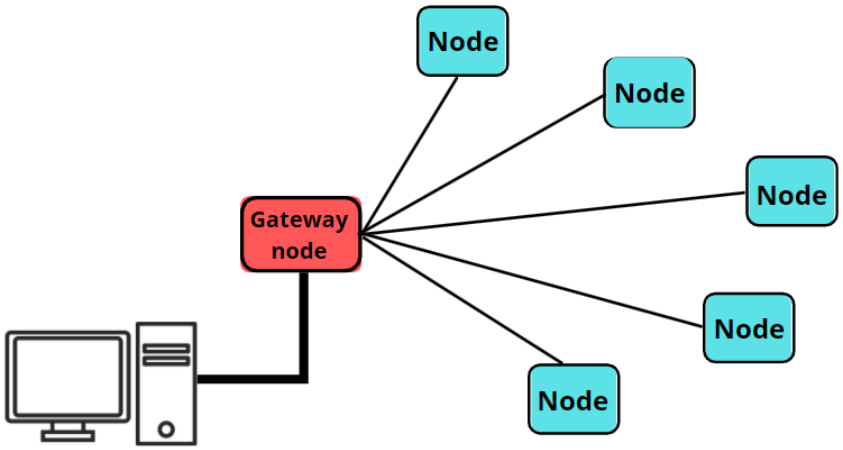


Figura 3.7: Architettura generale scenario urbano outdoor: rete LoRaWAN

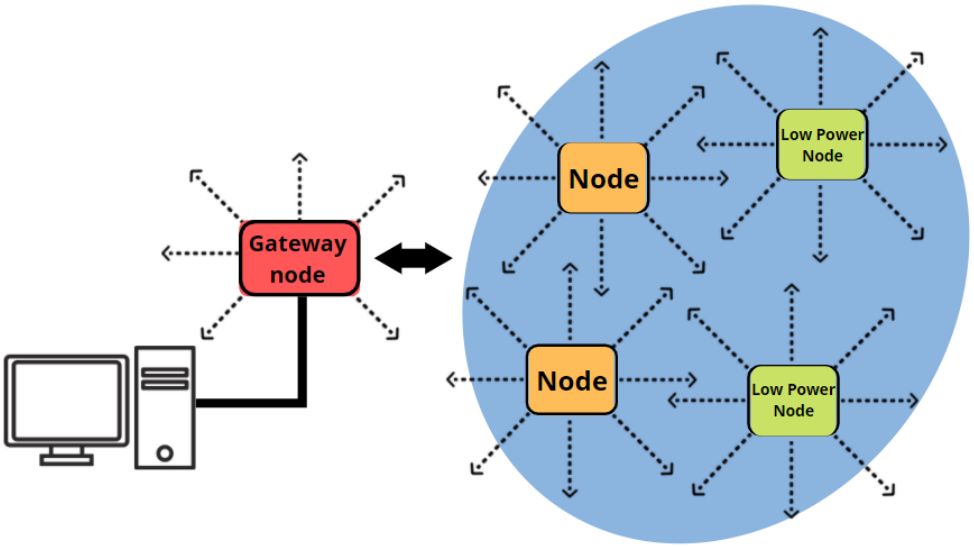


Figura 3.8: Architettura generale scenario locale indoor: Mesh BLE

Il Bluetooth, ed in particolare il Bluetooth Low Energy è uno standard altamente conosciuto e sicuro. La topologia Mesh, tipica del BLE, inoltre, permette di raggiungere distanze più elevate rispetto alle tecnologie short range, data la connessione hop to hop tra i dispositivi e di avere un notevole risparmio energetico grazie alla presenza di nodi LowPower. Nella figura 3.8 è descritta l'architettura generale con BLE nel caso di scenari applicativi che devono coprire distanze più ristrette. In entrambi i casi è possibile progettare e realizzare un Gateway con le stesse caratteristiche, che comunichi in modo seriale e che invii i dati via MQTT[5] in modo tale da renderli codificabili come eventi EPCIS ed inserirli in iChain.

Tale progetto è stato inizialmente commissionato da un'azienda che aveva bisogno di controllare lo stato dei propri prodotti all'interno di capannoni, quindi uno scenario indoor con distanze ristrette da raggiungere. Per questo fra le due precedenti topologie la rete che è stata implementata è la Mesh BLE.
Come si può vedere dalla figura 3.8, la rete Mesh comprende più nodi che comunicano tra di loro attraverso BLE. Tra i device sono inclusi nodi low power sui quali sono inseriti i sensori e che sono alimentati a batteria e nodi semplici che sono connessi all'alimentazione e che incrementano le distanze raggiungibili dalla

[5]MQTT o Message Queue Telemetry Transport è un protocollo di messaggistica per l'IoT di tipo publish/subscribe.

rete. Quest'ultimi sono connessi ad un Gateway, un nodo BLE che comunica con la Mesh sempre attraverso BLE e che è connesso in modo seriale al computer in modo da essere gestito dall'utente direttamente da un'applicazione con interfaccia grafica. Questo è implicato in due flussi di comunicazione:

- Richieste / risposte (PC <-> Nodo): tra il computer e i device, in cui vengono inviati comandi e ricevute le relative risposte;
- Eventi asincroni (Nodo -> PC): in cui il computer riceve i valori rilevati dai sensori attraverso eventi.

Nel Capitolo 6 verrà analizzata in dettaglio la progettazione della Mesh BLE e del Gateway e le relative caratteristiche.

Capitolo 4

Specifiche Bluetooth

In tale Capitolo vengono inizialmente descritte le specifiche del protocollo Bluetooth Low Energy. Successivamente viene dedicata una sezione allo standard BLE Mesh che è la specifica utilizzata per implementare il sistema IoT descritto in tale tesi. Vengono esaminate in dettaglio tutte le caratteristiche utili per comprendere come è stato possibile progettare ed implementare il sistema IoT in esame.

4.1 Bluetooth Low Energy

Il Bluetooth Low Energy (BLE) è un protocollo sviluppato dalla Bluetooth SIG, Special Interest Group, nel 2010 per affrontare la rapida crescita dei casi d'uso nel campo dell'Internet of Things (IoT).

4.1.1 Protocol Stack

Il Bluetooth Low Energy Protocol Stack, come è possibile analizzare dalla figura 4.1 è composto da tre principali componenti:

- Application. È responsabile della logica dell'applicazione, dell'interfaccia utente e della gestione dei dati. La sua architettura dipende fortemente da ogni particolare implementazione.
- Host. È diviso in molti layer:
  - Generic Access Profile (GAP);
  - Generic Attribute Profile (GATT);
  - Logical Link Control and Adaptation Protocol (L2CAP);
  - Attribute Protocol (ATT);

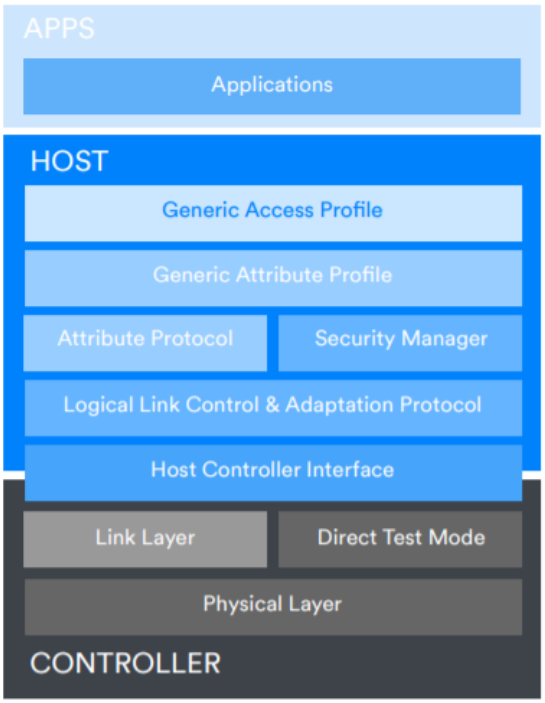


Figura 4.1: Stack Bluetooth Low Energy

  - Security Manager (SM);
  - Host Controller Interface (HCI) - parte Host.

- Controller. È suddiviso nei principali livelli:
  - Host Controller Interface (HCI) - parte Controller;
  - Link Layer (LL);
  - Physical Layer (PHY);

In seguito analizziamo cosa comportano i livelli dello stack più rilevanti, procedendo dal basso verso l'alto.

4.1.2 Controller

PHY. Corrisponde al Physical Layer dello stack ISO/OSI. Descrive in dettaglio come avviene la trasmissione dei dati, quale è la specifica banda e il tipo di modulazione. Definisce la radiofrequenza operativa (RF) tra 2,4000 GHz e 2,4835 GHz inclusi, la larghezza di banda del canale di 2 MHz, la banda operativa suddivisa in

40 canali (di cui 37 per i dati e 3 per gli advertising), utilizza FHSS (Frequency-Hopping Spread Spectrum) per ridurre al minimo l'effetto delle interferenze su ogni singolo canale e implementa lo schema di modulazione GFSK (Gaussian Frequency Shift-Keying).

LL. Il LL esegue compiti simili al livello MAC (Medium Access Control) del modello OSI. Si interfaccia direttamente con il BLE PHY e gestisce lo stato del collegamento della radio per definire il ruolo di un dispositivo come master, slave, advertiser o scanner.

HCI Lato Controller. Definisce un insieme di comandi ed eventi per la trasmissione e la ricezione dei pacchetti di dati. Quando si ricevono pacchetti dal controller, l'HCI estrae i dati grezzi dal controller per inviarli all'host.

4.1.3 Host

HCI Lato Host. Le sue funzionalità sono descritte nel paragrafo precedente in quanto tale livello si interfaccia tra Host e Controller.

SM. Applica algoritmi di sicurezza per crittografare e decrittografare i pacchetti di dati.

ATT. Trasferisce i dati degli attributi tra client e server nei profili basati su GATT e definisce i ruoli dell'architettura client-server. I ruoli corrispondono tipicamente al master e allo slave come definiti nel livello di collegamento. In generale, un dispositivo potrebbe essere un client, un server o entrambi, indipendentemente dal fatto che sia un master o uno slave. L'ATT esegue anche l'organizzazione dei dati in attributi suddividendo il pacchetto dati in byte da assegnare alle caratteristiche di ogni attributo ovvero il tipo, il valore, le autorizzazioni e l'handle.

L2CAP. Incapsula i dati dai livelli superiori BLE (ATT e SM) nel formato del pacchetto BLE standard per la trasmissione o viceversa estrae i dati dal pacchetto BLE standard ai dati dei livelli ATT e SM.

GATT. Descrive in dettaglio come vengono trasferiti gli attributi (dati) una volta che i dispositivi hanno una connessione. GATT si concentra in particolare su come i dati vengono formattati, impacchettati e inviati secondo le regole descritte. GATT utilizza ATT per descrivere come vengono scambiati i dati da due dispositivi collegati. Per i dispositivi che implementano il profilo GATT, il client è il

dispositivo che avvia comandi e richieste verso il server, può ricevere risposte, indicazioni e notifiche. Il server è il dispositivo che accetta i comandi e le richieste in entrata dal client. Il server invia risposte, indicazioni e notifiche al client. Il GATT utilizza un'architettura client-server. I ruoli non sono fissi e vengono determinati quando un dispositivo avvia una procedura definita. I ruoli vengono rilasciati al termine della procedura.

GAP. Definisce la topologia generale dello stack di rete BLE. Specifica ruoli, modalità e procedure di un dispositivo. Gestisce anche l'instaurazione della connessione e la sicurezza. Il GAP si interfaccia direttamente con il livello Application.

4.1.4 Application

È l'interfaccia utente diretta che definisce i profili che consentono l'interoperabilità tra le varie applicazioni.

4.2 BLE Mesh

Lo standard BLE Mesh si basa su BLE e ne utilizza molti concetti. Ci sono due stati principali in cui opera un dispositivo BLE: in advertising (o scanning) o in connessione. La mesh Bluetooth utilizza solo gli stati di advertising/scanning dei dispositivi BLE. Ciò significa che i dispositivi che fanno parte di una rete mesh Bluetooth non si connettono tra loro come fanno i dispositivi BLE tradizionali, ma piuttosto si trasmettono informazioni l'un l'altro tramite pacchetti di advertising che vengono ricevuti dagli altri dispositivi tramite scanning (c'è però un'eccezione, che coinvolge quello che viene chiamato un nodo proxy, che verrà descritto in tale paragrafo). La mesh Bluetooth supporta tutte le versioni di BLE (dalla versione Bluetooth 4.0, che è la prima versione delle specifiche Bluetooth che includeva BLE). Il numero massimo di nodi all'interno di una rete mesh Bluetooth (definito nella versione 1.0 della specifica) è 32.767 nodi. Gli hop massimi che un messaggio può viaggiare all'interno di una rete mesh Bluetooth (definita nella versione 1.0 della specifica) è 127 hop.

La specifica mesh Bluetooth è stata rilasciata a luglio 2017 con l'obiettivo di aumentare la gamma di reti Bluetooth e aggiungere il supporto per più applicazioni industriali che utilizzano BLE. Le versioni precedenti di Bluetooth supportavano due diverse topologie:
- One-to-one : quando sono collegati due dispositivi BLE.
- Uno a molti : quando i dispositivi BLE operano esclusivamente nello stato Broadcast, come i Beacon.

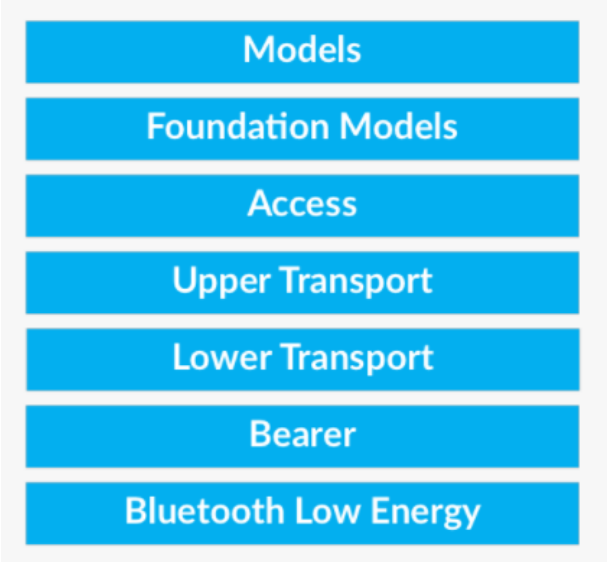


Figura 4.2: Architettura della libreria mesh BLE

Con Bluetooth mesh viene introdotta una nuova topologia per BLE. I dispositivi possono ora operare in una topologia molti-a-molti, o cosiddetta mesh, in cui i dispositivi possono configurare connessioni con più altri dispositivi all'interno della rete.

4.2.1 Libreria

La libreria BLE Mesh è descritta dalla figura 4.2. La sua architettura è articolata in 7 layer, che, esaminati dal basso verso l'alto, sono:

- Bluetooth Low Energy

  Come accennato in precedenza, tale mesh si basa su BLE, quindi richiede uno stack BLE completo. Utilizza gli stati di advertising e scanning per inviare e ricevere messaggi tra dispositivi all'interno della rete mesh. Supporta anche connected state e GATT per dispositivi speciali chiamati nodi proxy.

- Bearer

  Definisce come vengono gestiti i diversi pacchetti mesh (Protocol Data Units o PDU). Esistono due tipi di bearer di rete Bluetooth:

  – Advertising Bearer: Sono usati dai nodi di tipo Low Power, Relay e Friend. Utilizza gli stati di advertising e scanning dei dispositivi BLE GAP per inviare e ricevere messaggi.

  - GATT Bearer: Viene utilizzato dai nodi Proxy o da uno Smartphone. Utilizza lo stato di connessione dei dispositivi BLE GATT. Consente ai dispositivi non di supporto mesh di interagire con la rete mesh tramite operazioni GATT point to point. Il Bearer GATT utilizza un lo standard GATT e fornisce le caratteristiche per la gestione delle notifiche di valore e per scrivere i comandi.

- Lower Transport

  Gestisce la segmentazione dei pacchetti dal livello superiore (upper transport) e il riassemblaggio dei pacchetti dal livello sottostante (livello Bearer).

- Upper Transport

  È responsabile delle funzionalità di crittografia, decrittografia,autenticazione e trasporto dei messaggi di controllo.

- Access

  Tale livello definisce come l'applicazione utilizza l'Upper Transport layer. Gestisce il formato dei dati dell'applicazione, la crittografia, la decrittografia e la verifica dei dati.

- Foundation Models

  Questo livello riguarda la configurazione della rete e i modelli di gestione della rete. È il modello che implementano tutti i dispositivi di una mesh BLE.

- Models

  Questo livello si occupa dell'implementazione di tutti gli altri Models, che sono analizzati in una sezione a parte.

4.2.2 Topologia e caratteristiche

Una mesh è una rete diramata all'interno della quale i nodi sono messi in comunicazione con il maggior numero possibile di nodi attivi in modo da trasmettere multi-hop i dati più efficientemente possibile. Un esempio di tale topologia può essere osservato nella figura 4.3 in cui si nota la struttura.
Una delle caratteristiche principali della rete mesh BLE è il protocollo con il quale vengono trasmessi i messaggi tra i device, il flooding protocol. Il termine flooding, che in italiano può essere tradotto come allagamento, rende l'idea di come un nodo trasmetta un qualunque messaggio a tutti gli altri device che siano attivi e in ascolto sulla rete. Questa tecnica multi-hop estende in maniera rilevante il raggio di comunicazione. In particolare la mesh BLE sfrutta il managed flooding: i messaggi contengono un numero in sequenza, in modo che il nodo possa capire se

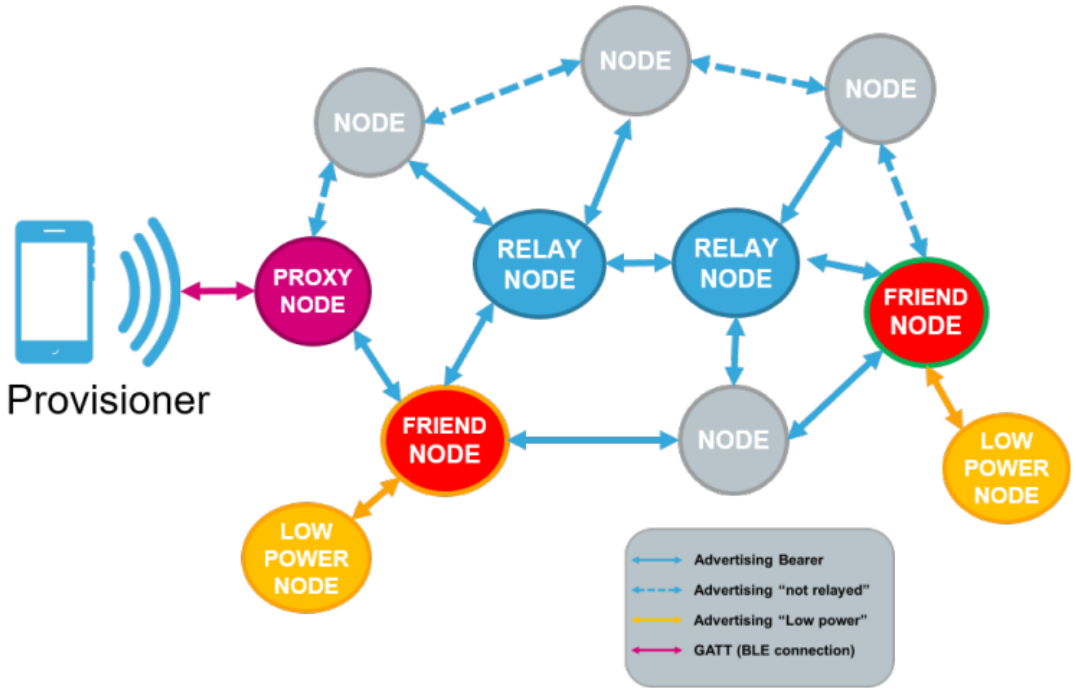


Figura 4.3: Topologia di una mesh BLE

il messaggio ricevuto è nuovo o meno. I vecchi messaggi non sono quindi inoltrati per ottimizzare l'utilizzo della rete.
Un altro aspetto fondamentale per la gestione della rete è il self-healing dinamico. Ad esempio, se uno nodo fallisce la ritrasmissione di un messaggio, la comunicazione può essere ripristinata utilizzando altri nodi vicini.

4.2.3 Nodi

Innanzi tutto può essere considerato nodo solo un device che ha fatto il provisioning alla rete mesh. Questo processo è necessario per la configurazione dei dispositivi e verrà descritto in dettaglio nelle prossime sezioni. Vi sono quattro tipologie di nodi che implementano differenti caratteristiche:

- Relay - Hanno la capacità di ritrasmettere messaggi da altri nodi. Supportano quindi la funzione di inoltro e hanno quindi la capacità di estendere la rete oltre la portata del singolo nodo originale;
- Proxy - Consentono la comunicazione con dispositivi BLE non supportati da mesh, come ad esempio uno smartphone. In tale caso il protocollo di comunicazione è proxy ed è basato sul profilo GATT;

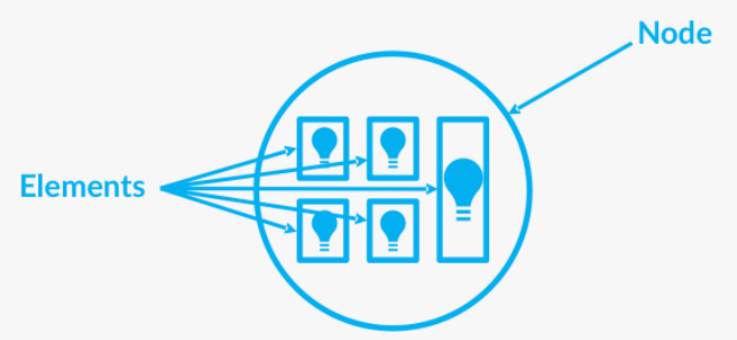


Figura 4.4: Elementi di un nodo

- Friend - Sono nodi che richiedono relazioni di amicizia con altri nodi all'interno della Mesh. Sono strettamente correlati ad i nodi Low Power. Infatti un nodo a basso consumo può essere considerato tale grazie all'amicizia con un nodo Friend, il quale salva nella cache i messaggi da inoltrare al nodo Low Power, il quale sotto al controllo dell'amico si riattiva e riceve i messaggi memorizzati;

- Low Power - Hanno un basso consumo energetico in quanto rimangono attivi solo per un certo periodo. Il nodo amico permette loro di rimanere inattivi più a lungo. Essi generalmente sono alimentati a batteria, in quanto consumano poco.

4.2.4 Elementi

Un elemento corrisponde alla parte di un nodo che può essere controllata e gestita in modo indipendente. Come si può osservare dalla figura 4.4 un nodo può contenere molteplici elementi.

4.2.5 Stati

Gli elementi possono trovarsi in varie condizioni, rappresentate da valori di stato. Ad esempio, acceso e spento sono gli stati di una lampadina all'interno di un apparecchio di illuminazione. Un passaggio da uno stato a un altro è chiamato transizione di stato. Questo può essere istantaneo o può verificarsi nel tempo, dopo quello che viene chiamato un periodo di transizione . Quando si verifica un cambiamento di stato, è probabile che causi un cambiamento nel comportamento di un elemento.

4.2.6 Proprietà

Le proprietà contengono, come gli stati, valori relativi a un elemento, tuttavia, a differenza degli stati, le proprietà forniscono il contesto per interpretare gli stati e comprendono sia un identificativo che un valore. Si consideri, ad esempio, un dispositivo che desidera inviare un valore di stato della temperatura. Lo stato della temperatura può essere "Temperatura ambiente interna attuale" o "Temperatura ambiente esterna attuale". In questo caso, verrebbe utilizzata una proprietà per fornire il contesto per il valore dello stato di temperatura. Le proprietà possono essere proprietà del manufacturer, che sono di sola lettura o proprietà di Admin, che consentono l'accesso in lettura e scrittura.

4.2.7 Indirizzi

I messaggi in una rete mesh Bluetooth devono essere inviati da e verso un indirizzo. Esistono tre tipi di indirizzi:

- Indirizzo unicast: un indirizzo che identifica in modo univoco un singolo nodo assegnato durante il processo di provisioning, il processo di configurazione di un nodo all'interno di una rete che sarà descritto in seguito;

- Indirizzo di gruppo: un indirizzo utilizzato per identificare un gruppo di nodi. Un indirizzo di gruppo di solito riflette un raggruppamento fisico di nodi come tutti i nodi all'interno di una stanza specifica. Gli indirizzi di gruppo possono essere fissi o dinamici. Gli indirizzi di gruppo fisso definiti dalla SIG sono: Tutti i proxy, Tutti i friend, Tutti i relay e Tutti i nodi. L'indirizzo di gruppo dinamico può essere definito dall'utente tramite un'applicazione di configurazione;

- Indirizzo virtuale: un indirizzo che può essere assegnato ad uno o più elementi, che si estende su uno o più nodi. Questo funge da etichetta e assume la forma di un UUID a 128 bit a cui può essere associato qualsiasi elemento. È probabile che gli indirizzi virtuali siano preconfigurati al momento della produzione.

4.2.8 Provisioning

Affinchè un dispositivo sia considerato un nodo all'interno della rete questo deve essere stato prima provisionato, grazie ad un processo chiamato provisioning. Questi passaggi avvengono tra l'unprovisioned device e il provisioner, ovvero chi si occupa di configurare i nodi all'interno della rete. Il provisioner può essere sia un nodo della stessa rete che un dispositivo esterno di una tipologia diversa come

uno Smartphone.
Tale processo si articola in 5 fasi:

1. Beaconing: In tale passaggio l'unprovisioned device produce un avviso, un unprovisioned device Beacon, che viene visto dal provisioner nel momento della scansione della rete. Questo nuovo tipo di messaggio è introdotto nello standard mesh Bluetooth.

2. Invito: Il provisioner esorta l'unprovisioned device a inviare le informazioni sulle sue capacità di provisioning. Questo utilizza un nuovo tipo di PDU (Protocol Data Unit) introdotto nella rete Bluetooth chiamato provisioning invite PDU. Il dispositivo di cui non è stato eseguito il provisioning risponde quindi con un PDU che include:

   - Il numero di elementi supportati dal dispositivo.
   - Il set di algoritmi di sicurezza supportati.
   - La disponibilità della sua chiave pubblica utilizzando una tecnologia Out-of-Band (OOB).
   - La capacità di questo dispositivo di restituire un valore all'utente.
   - La capacità di questo dispositivo di consentire l'immissione di un valore da parte dell'utente.

3. Scambio delle chiavi pubbliche: Lo scambio delle chiavi pubbliche tra provisioner e unprovisioned device avviene in modo sicuro. Vengono infatti scambiate una combinazione di chiavi simmetriche e asimmetriche (ECDH Elliptic-curve Diffie–Hellman). Questo scambio può avvenire o lungo il layer BLE o usando il canale OOB.

4. Autenticazione: Tale fase può avvenire in diversi modi. In un caso, chiamato output OOB, il dispositivo unprovisionato emette un numero casuale a una o più cifre all'utente in qualche forma, come il lampeggio di un LED un numero di volte. Tale numero viene quindi immesso nel dispositivo di provisioning tramite un metodo di input. Altri casi includono un ingresso OOB, in cui il numero è generato dal provider e immesso nel dispositivo non predisposto, un OOB statico o nessun OOB. Indipendentemente dal metodo di autenticazione utilizzato, l'autenticazione include anche una fase di generazione del valore di conferma e una fase di verifica della conferma.

5. Scambio delle chiavi del provisioning: Da ciascuno dei due dispositivi viene ricavata la Session Key. Questa viene quindi utilizzata per proteggere i dati successivi necessari per completare il processo di provisioning. Una volta che il provisioning è completato, il dispositivo provisionato possiede la NetKey e un indirizzo unicast, allocato dal provisioner.

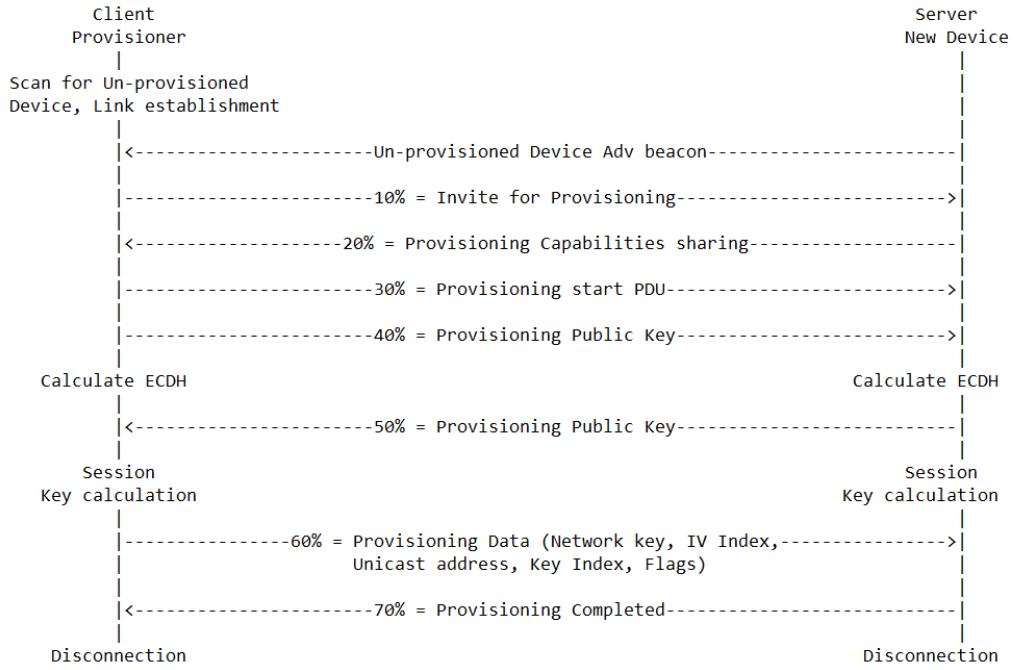


Figura 4.5: Grafico a traliccio del provisioning

Nella figura 4.5 è riportato il grafico a traliccio di come avviene il processo di provisioning.

4.2.9 Messaggi e Publish/Subscribe

Nella mesh Bluetooth, tutte le comunicazioni all'interno della rete sono message-oriented e i nodi inviano messaggi per controllare o scambiarsi informazioni. I messaggi sono il meccanismo mediante il quale vengono invocate le operazioni sui nodi. Se un nodo ha bisogno di segnalare il suo stato, lo invia anche tramite un messaggio. Un determinato tipo di messaggio rappresenta un'operazione su uno stato o una raccolta di più valori di stato. Esistono tre tipi di messaggi , ognuno dei quali è definito da un codice operativo univoco (codice operativo):

- Un messaggio GET: un messaggio per richiedere lo stato da uno o più nodi;
- Un messaggio SET: un messaggio per modificare il valore di un dato stato;
- Un messaggio di STATO : un messaggio di stato viene utilizzato in diversi scenari:
  - In risposta a un messaggio GET, contenente il valore dello stato;

  - In risposta a un messaggio SET riconosciuto;
  - Inviato indipendentemente da qualsiasi messaggio per segnalare lo stato dell'elemento. Un esempio è un messaggio che viene attivato da un timer in esecuzione sull'elemento che invia questo messaggio.

Alcuni messaggi richiedono l'invio di un messaggio di conferma da parte del destinatario del messaggio originale. Un messaggio di conferma ha due scopi: conferma di ricezione del messaggio o restituzione dei dati relativi al messaggio ricevuto. Nel caso in cui il mittente non riceva una risposta al messaggio o riceva una risposta imprevista, il mittente può inviare nuovamente il messaggio. Più messaggi confermati ricevuti da un nodo non influiscono sul comportamento (è come se il messaggio fosse stato ricevuto una volta).

Il modo in cui i messaggi vengono scambiati in una rete mesh Bluetooth è tramite il modello publish-subscribe.
Un nodo publisher invia messaggi e solo i nodi che hanno fatto subscribe al nodo publisher riceveranno questi messaggi. Ciò garantisce che diversi tipi di prodotti possano coesistere in una rete senza essere disturbati dai messaggi provenienti da dispositivi che non devono ascoltare.
In genere, i messaggi sono indirizzati a indirizzi di gruppo o virtuali. I nodi possono fare subscribe a più indirizzi. Inoltre, più nodi possono pubblicare allo stesso indirizzo.
Il vantaggio dell'utilizzo di indirizzi di gruppo o virtuali è che l'aggiunta o la rimozione di nodi non richiede la riconfigurazione dei nodi.

4.2.10 Modelli

I modelli vengono utilizzati per definire la funzionalità di un nodo. Esistono tre tipi di modelli mesh:

- Modello Server: un modello Server può avere uno o più stati che si estendono su uno o più elementi. Il modello server definisce i messaggi che un modello può trasmettere e ricevere. Definisce anche i comportamenti dell'Elemento in base a questi messaggi. Detto in altro modo, i Modelli Server espongono gli stati dell'Elemento che possono essere letti o controllati da un Client;

- Modello Client: un modello client definisce l'insieme di messaggi per richiedere e modificare lo stato di un server;

- Modello di Controllo o Setup: un'applicazione finale può utilizzare modelli Server o modelli Client o entrambi insieme alla logica di controllo. Qualsiasi combinazione di modelli Server e Client si traduce in un modello di controllo.

Figura 4.6: Bluetooth Mesh Models

Le tipologie di modello sono descritte dall'immagine 4.6, ogni modello si differenzia da un altro per le funzionalità. I modelli possono estendere le funzionalità di altri modelli. Cioè, i modelli possono essere gerarchici. Ad esempio, il modello "Light Lightness" estende il modello "Generic OnOff Server" e il modello "Generic Level Server". Ciò significa che se si implementa il modello Light Lightness in un'applicazione, si ottengono tutte le funzionalità Light Lightness più tutte le funzionalità Generic Level e Generic OnOff. I modelli che non estendono altri modelli sono noti come "modelli radice".

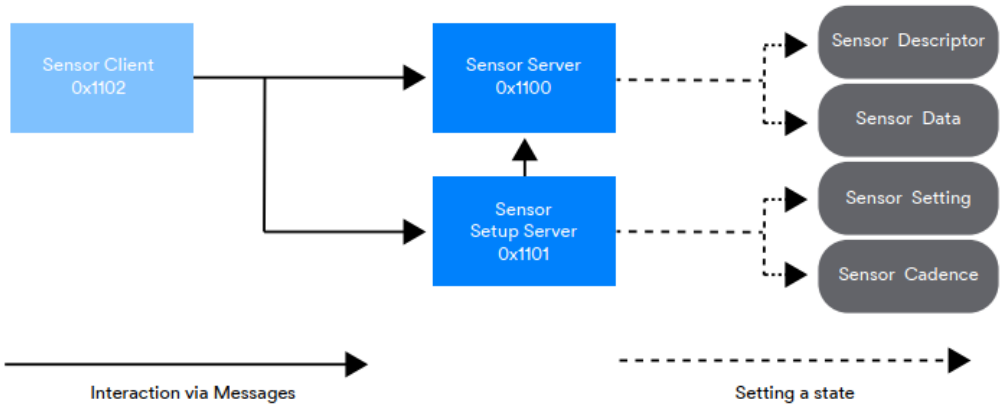


Figura 4.7: Sensor Model

Le tipologie di modello principali, che sono state integrate all'interno dei nodi della rete implementata in questa tesi sono Config Model, Sensor Model e Generic OnOff. Con i soli primi due modelli è possibile realizzare una rete di sensori funzionante. Il Generic OnOff è stato integrato per ausilio nel riconoscere un nodo all'interno della rete, come verrà descritto nel Capitolo 6.

4.2.10.1 Config Model

Tutti i dispositivi devono essere configurabili. L'implementazione del modello del Config Server è quindi obbligatorio e fornisce al dispositivo la possibilità di essere configurato, in genere utilizzando un'applicazione su uno smartphone che implementerà il modello Config Client.

Il modello Config Server contiene un numero significativo di stati che corrispondono ai vari aspetti di dispositivo da configurare. La composizione complessiva del dispositivo è contenuta in uno stato chiamato Composition Data. Questo contiene l'indirizzo di destinazione da utilizzare per la pubblicazione di messaggi e altri parametri relativi alla pubblicazione periodica di messaggi, gli indirizzi a cui ci si è iscritti e gli indirizzi dei nodi friend, low power o proxy con i quali si interfaccia.

4.2.10.2 Sensor Model

I sensori svolgono un ruolo fondamentale in molte applicazioni. Rilevano e segnalano eventi e misurano gli attributi dell'ambiente, condividendo questi dati con altri dispositivi. Una mesh Bluetooth LE rende possibile questo scenario attraverso l'uso del Sensor Model.

Figura 4.8: Stati del Sensor Model

Questo modello consente a qualsiasi tipo di sensore di comunicare le proprie letture ad altri nodi della rete ed è articolato in Sensor Client, Sensor Server e Sensor Setup Server, dove il Setup Server consente di configurare il sensore e il formato dei suoi dati. La relazione tra questi tre modelli è descritta della figura 4.7: Il modello Setup Server estende il modello Server, mentre il modello Client non è correlato a nessun altro modello.
Il Sensor Client interagisce tramite messaggi coi Sensor Server e Sensor Server Setup, i quali definiscono gli stati di tale modello che sono Descriprot, Data, Setting e Cadence.

I modelli di sensori fanno un uso ampio delle proprietà con un numero relativamente piccolo di stati. Le proprietà differiscono dagli stati in quanto contengono sia un identificatore che un valore. L'identificatore ci dice che tipo di dati contiene la proprietà in modo che sia autodescrittivo. Gli Stati, d'altra parte, hanno nessun identificatore di tipo esplicito, ed è il modello o il messaggio in cui è contenuto lo stato che ci dice il stato dei dati. Lo sfruttamento delle proprietà ha consentito ai tre modelli di sensori di adattarsi a qualsiasi tipo di sensore e dati del sensore, piuttosto che richiedere modelli, messaggi e stati diversi per ogni tipo immaginabile di sensore che potrebbe essere parte di una rete.

4.2.10.3 Generic OnOff Model

Tale modello non ha un ruolo fondamentale per la costruzione del sistema IoT. È comunque utile per riuscire a distinguere con quale nodo fra tutti quelli nella rete si sta comunicando, poichè permette l'accesione di un led sul device.

I modelli Generic OnOff rendono possibile l'accensione o lo spegnimento di un elemento generico. Il modello Server contiene un solo stato, lo stato onoff generico che è un semplice stato booleano che indica se un elemento è attualmente acceso o spento. Il valore di 0x00 significa che è spento e 0x01 significa che è attivo. Il Generic Client onoff può inviare un Get o un Set.

4.2.11 Sicurezza e privacy

La sicurezza e la privacy sono sempre una preoccupazione nei dispositivi connessi in modalità wireless. BLE Mesh incorpora diverse tecniche per alleviare questo problema.

- **Crittograzia e autenticazione**: Tutti i messaggi Mesh Bluetooth sono crittografati e autenticati.
- **Aggiunte sicura dei nodi nella rete**: BLE Mesh fornisce un metodo sicuro per aggiungere un dispositivo alla rete, il provisioning di un device infatti ha bisogno della conoscenda della chiave di applicazione.
- **Separazione della sicurezza tra i livelli**: A causa della separazione della sicurezza tra i livelli di rete, di applicazione e di dispositivo, esistono tre tipi di chiavi di sicurezza (ognuna che affronta un problema specifico):
  - Chiave di rete (NetKey). Il possesso di questa chiave condivisa rende il dispositivo parte della rete (nota anche come nodo). Ci sono due chiavi derivate dalla NetKey: la chiave di crittografia di rete e la chiave di privacy. Il possesso della NetKey consente a un nodo di decrittografare e

autenticare fino al livello di rete, consentendo l'inoltro dei messaggi, ma nessuna decrittografia dei dati dell'applicazione.

  - Chiave dell'applicazione (AppKey). Questa è una chiave condivisa tra un sottoinsieme di nodi all'interno di una rete mesh, normalmente quelli che partecipano a un'applicazione comune. Ad esempio, un sistema di illuminazione AppKey sarebbe condiviso tra interruttori della luce e lampadine, ma non con un termostato o un sensore di movimento. Una AppKey viene utilizzata per decrittografare e autenticare i messaggi a livello di applicazione, ma è valida solo all'interno di una rete mesh, non su più reti.
  - Chiave dispositivo (DevKey). Si tratta di una chiave specifica del dispositivo utilizzata durante il processo di provisioning per proteggere la comunicazione tra il dispositivo di cui non è stato eseguito il provisioning e il provisioner.

- **Isolamento dell'area**: Una rete Bluetooth Mesh può essere suddivisa in sottoreti, ciascuna crittograficamente distinta e sicura dalle altre. Ad esempio, ogni stanza di un hotel può essere una sottorete mentre l'hotel è la rete completa. Le sottoreti consentono agli ospiti di una stanza di non interferire con le altre stanze.

- **Aggiornamento delle chiavi, protezione dagli attacchi del cestino e rimozione dei nodi**: Le chiavi di sicurezza possono essere modificate durante la vita della rete mesh Bluetooth tramite una procedura Key Refresh. Quando un nodo viene rimosso dalla rete, la rete Mesh esegue la procedura di aggiornamento delle chiavi per modificare le chiavi su tutti gli altri nodi. Una volta aggiornate le chiavi, le (vecchie) chiavi memorizzate sul nodo che sono state rimosse non hanno alcun valore. Questa funzione risolve i problemi di sicurezza se qualcuno ottiene l'accesso a un dispositivo che era presente in precedenza in una rete BLE: questo tipo di collegamento è noto come attacco cestino. .

- **Protezione contro gli attacchi di ripetizione**: La sicurezza mesh Bluetooth protegge la rete dagli attacchi di riproduzione. Gli attacchi di replay sono uno degli attacchi più comuni in cui un intercettatore ascolta un messaggio e poi lo riproduce con cattive intenzioni. Immagina che qualcuno ascolti un messaggio di sblocco della porta e poi lo riproduca nel cuore della notte per irrompere in una casa. BLE Mesh fornisce un meccanismo per evitare attacchi di replay aggiungendo un numero di sequenza in ogni messaggio. I messaggi con un numero di sequenza uguale o inferiore a quello del messaggio precedente vengono automaticamente ignorati.

- **Offuscamento del messaggio**: L'offuscamento dei messaggi rende difficile il tracciamento dei messaggi inviati all'interno della rete e, in quanto tale, fornisce un meccanismo di privacy per rendere difficile il tracciamento dei nodi.

Capitolo 5

Strumenti e Software

In questo capitolo vengono analizzati i componenti hardware utilizzati per la realizzazione dei device del sistema IoT e gli ambienti di sviluppo che ne hanno permesso la programmazione.

5.1 Componenti Hardware

I nodi all'interno della mesh BLE sono stati relizzati utilizzando le schede di sviluppo della STMicroelectronics, un'azienda leader della produzione di componenti elettroniche. Sono stati acquistate le board NUCLEO-WB55RG per implementare i device con tecnologia BLE, e alcune sono state integrate con le shield di sensori X-NUCLEO-IKS01A3, in grado di rilevare temperatura, pressione e umidità o con P-NUCLEO-IKA02A1, una scheda di espansione per il controllo della qualità dell'aria.

5.1.1 STMicroelectronics

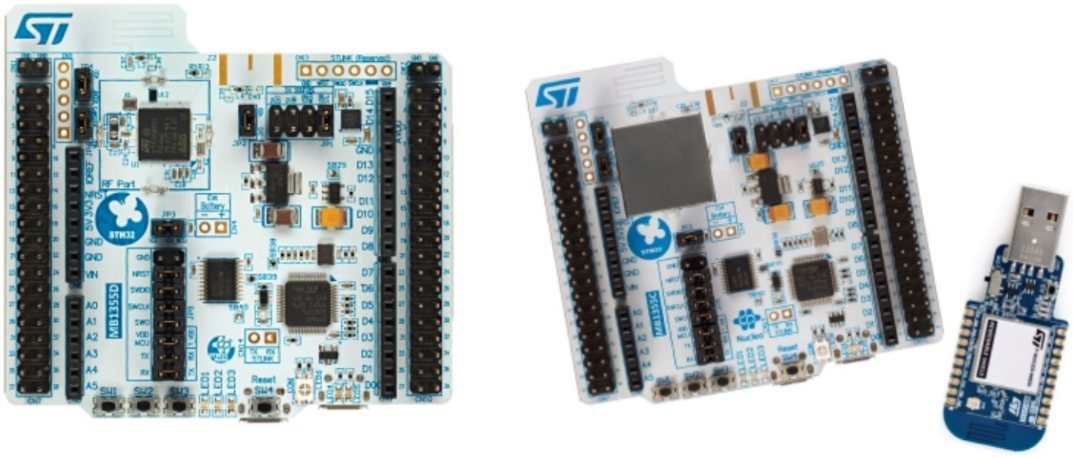
La STMicroelectronics è una multinazionale italo-francese produttrice di componenti elettroniche e semiconduttori. Ha sede in Svizzera, a Plan-les-Ouates vicino a Ginevra, ed è stata fondata nel 1987 come fusione di due società di semiconduttori, l'azienda francese "Thomson Semiconducteurs" e qualla italiana "SGS Microelettronica".
Viene comunemente chiamata "ST" ed è il più grande produttore di chip a semiconduttore in Europa in base alle entrate.

I settori operativi in cui si inserisce sono il settore dell'automotive, il settore industriale, applicazioni IoT e produzioni di apparecchiature di comunicazione, computer e periferiche.

Ha creato attorno a sé numerosi strumenti proprietari come applicazioni mobili, strumenti di sviluppo, toolchain di programmazione e strumenti di valutazione. In tale progetto è stata utilizzata la toolchain proprietaria ST, le cui componenti verranno descritte in dettaglio in seguito.

5.1.1.1 Nucleo Board e USB Dongle

Per la realizzazione dei nodi sono state esaminate numerose schede di sviluppo. Una scheda di sviluppo è una scheda a circuito stampato contenente un microprocessore e la logica di supporto minima necessaria per la programmazione. La STMicroelectronics ha un'ampia scelta in termini di developement board, se ne possono osservare le principali caratteristiche nell'immagine 5.2. Il requisito fondamentale che ha permesso la selezione è stato la presenza del modulo radio Bluetooth Low Energy. Come si può vedere dalla figura solo tre schede montano tale modulo, la NUCLEO-WL55JC, la NUCLEO-WB15CC e la NUCLEO-WB55RG e ciò che le differenzia è la dimenzione della memoria flash, maggiore per la WB55RG. Per questo motivo per l'implementazione dei nodi sono state utilizzate le board NUCLEO-WB55RG (di tipo Nucleo-64) o il pacchetto P-NUCLEO-WB55 contenente una board WB55RG ma di tipo Nucleo-68 più la USB dongle, una "chiavetta", di minori dimensioni rispetto alla scheda Nucleo, che monta lo stesso microcontrollore, lo stesso modulo bluetooth e che ha la possibilità di connettersi in modo seriale ad un pc in modo più semplice, pulito (senza l'uso di cavetti) ed intuitivo.
Nell'immagine 5.1 sono mostrate le schede di sviluppo utilizzate ed in seguito ne verranno descritte in maniera dettagliata le specifiche.



(a) NUCLEO-WB55RG (b) P-NUCLEO-WB55

Figura 5.1: Schede di sviluppo utilizzate

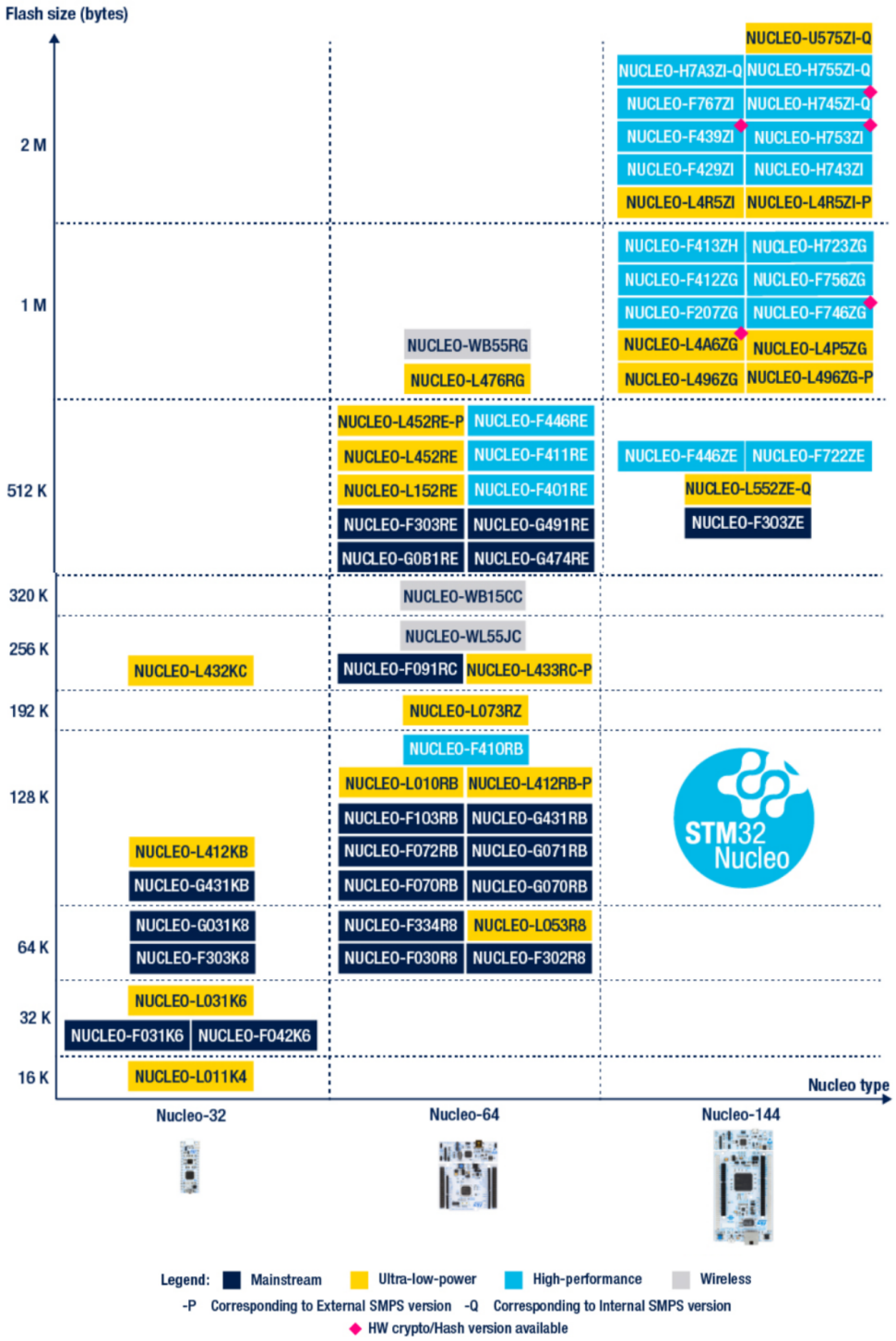


Figura 5.2: Board STMicroelectronis

NUCLEO-WB55RG. Le schede NUCLEO-WB55RG STM32WB Nucleo-64 sono dispositivi wireless Bluetooth Low Energy (BLE) e a bassissima potenza che incorporano un modulo radio conforme al BLE SIG specifica v5.2. NUCLEO-WB55RG offre anche una radio conforme allo standard IEEE 802.15.4-2011. Il supporto per la connettività ARDUINO Uno V3 e per gli header ST morpho forniscono un mezzo semplice per espandere le funzionalità della scheda di sviluppo STM32WB Nucleo con un'ampia scelta di shield specializzati.
Le caratteristiche principali di tale board sono:

- Processore: dual-core a 32 bit con Arm Cortex -M4 e CPU M0+ dedicata per livello radio in tempo reale;
- Memoria: 1-Mbyte Flash memory e 256-Kbyte SRAM;
- Radio: Ricetrasmettitore RF che supporta Bluetooth 5.2, IEEE 802.15.4-2011 PHY e MAC, Thread e Zigbee 3.0;
- Componenti utente: tre led, un bottone di reset e tre bottoni utente;
- Connettori della scheda: connettore di espansione ARDUINO Uno e pin di estensione ST morpho per l'accesso completo a tutti gli I/O STM32WB;
- Opzioni di alimentazione flessibili: ST-LINK, USB o fonti esterne;
- Debugger/programmatore ST-LINK/V2-1 integrato con capacità di rienumerazione USB: archiviazione di massa, porta COM virtuale e porta di debug;
- Supporto di un'ampia scelta di ambienti di sviluppo integrati (IDE) inclusi STM32CubeIDE e Mbed Studio;
- Librerie software gratuite complete ed esempi disponibili con il pacchetto MCU STM32CubeWB.

Tutte le caratteristiche della scheda sono mostrate nell'Appendice a pag 94, nella sezione Datasheets.

STM32WB NUCLEO-68 Tale scheda di sviluppo è molto simile a quella descritta in precedenza. Alcune importanti differenze sono che questa supporta solo connettività Bluetooth e che ha un solo core invece che due.
Le caratteristiche che implementa sono:

- Processore: CPU Arm® Cortex® M0+ a 32 bit dedicata per il livello radio in tempo reale;
- Microcontrollore STM32WB;

- Radio: Ricetrasmettitore RF a 2,4 GHz che supporta le specifiche Bluetooth v5.0 e IEEE 802.15.4-2011 PHY e MAC;
- Componenti utente: tre led, un bottone di reset e tre bottoni utente;
- Connettori della scheda: connettore di espansione ARDUINO Uno e pin di estensione ST morpho per l'accesso completo a tutti gli I/O STM32WB;
- Opzioni di alimentazione flessibili: ST-LINK, USB o fonti esterne;
- Debugger/programmatore ST-LINK/V2-1 integrato con capacità di rienumerazione USB: archiviazione di massa, porta COM virtuale e porta di debug;
- Supporto di un'ampia scelta di ambienti di sviluppo integrati (IDE) inclusi STM32CubeIDE e Mbed Studio;
- Librerie software gratuite complete ed esempi disponibili con il pacchetto MCU STM32CubeWB.

La lista completa delle specifiche è nell'Appendice.

USB DONGLE. La chiavetta presenta le seguenti caratteristiche principali:

- Processore: CPU Arm® Cortex® M0+ a 32 bit dedicata per il livello radio in tempo reale;
- Microcontrollore STM32WB;
- Radio: Ricetrasmettitore RF a 2,4 GHz che supporta le specifiche Bluetooth v5.0 e IEEE 802.15.4-2011 PHY e MAC;
- Componenti utente: tre led, un bottone utente;
- Switch per la gestione dell'avvio.

5.1.1.2 Shield e componenti aggiuntivi

Per espandere le funzionalità delle board per i nodi sensori sono state acquistate due schede: la X-NUCLEO-IKS01A3 per la misurazione di temperatura, pressione e umidità e la P-NUCLEO-IKA02A1 per il controllo della qualità dell'aria in quanto riesce a misurare la concentrazione di CO.

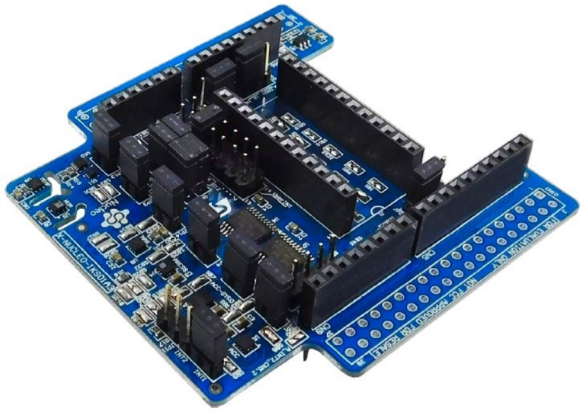


Figura 5.3: X-NUCLEO-IKS01A3

X-NUCLEO-IKS01A3 Tale shield è un MEMS (Micro-electromechanical systems) ovvero un sistema che integra componenti elettriche e meccaniche e che, in questo caso, valuta movimento e condizioni ambientali. Esso si interfaccia con il microcontrollore STM32 tramite I^2C, ed è possibile cambiare la porta I^2C predefinita, come sarà descritto nel capitolo di progettazione.
I sensori presenti in tale scheda di espansione sono:

- LSM6DSO: Accelerometro 3D MEMS (±2/±4/±8/±16 g) + giroscopio 3D (±125/±250/±500/±1000/±2000 dps);
- LIS2MDL: magnetometro 3D MEMS (±50 gauss);
- LIS2DW12: Accelerometro 3D MEMS (±2/±4/±8/±16 g);
- LPS22HH: sensore di pressione MEMS, barometro con uscita digitale assoluta 260-1260 hPa;
- HTS221: umidità relativa e temperatura digitali capacitive;
- STTS751: Sensore di temperatura (da –40 °C a +125 °C).

Si può osservare come è strutturata tale scheda dalla figura 5.3.

P-NUCLEO-IKA02A1 Il P-NUCLEO-IKA02A1 è un pacchetto contenente un microcontrollore NUCLEO-L053R8 e una scheda di espansione per il controllo dei gas presenti nell'aria. Sono stati acquistati entrambi in quanto la shield non era acquistabile singolarmente, ma è stata comunque pensata per l'utilizzo con le board NUCLEO elencate in precedenza.

Figura 5.4: P-NUCLEO-IKA02A1

La scheda di espansione per il controllo dei gas interfaccia i sensori elettrochimici con l'MCU sulla scheda di sviluppo STM32 Nucleo. Due amplificatori operazionali TSU111 forniscono il condizionamento del segnale; sono ideali per il rilevamento elettrochimico grazie alla loro elevata precisione e al basso consumo energetico. La scheda di espansione include un sensore di temperatura analogico di precisione a bassissima corrente STLM20 utilizzato per la compensazione delle letture del gas. Essa include anche un sensore che rileva il monossido di carbonio (CO), il Figaro TGS5141.
In figura 5.4 è mostrata la shield contenuta nel pacchetto che verrà poi montata sulle board Nucleo-WB.

5.2 Software e toolchain

Per la programmazione delle schede sono state esaminate tre tipologie di software: Zephyr OS, Mbed OS e la Toolchain di ST, STM32Cube. Zephyr e Mbed sono due sistemi operativi per l'IoT ovvero sistemi operativi specializzati e progettati per funzionare entro i rigidi vincoli dei piccoli dispositivi IoT. Si tratta di sistemi operativi embedded che consentono ai dispositivi IoT di comunicare con servizi cloud e altri dispositivi IoT su una rete globale e possono farlo entro i parametri ristretti di quantità limitate di memoria e potenza di elaborazione.

	ZephyrOS	**MBedOS**	**STM32Cube**
Provider	Linux Foundation	ARM	STMicroelectronics
Architetture HW supportate	ARM, ARC, NIOS2...	ARM	STM32
Cloud integrato	No	Sì	No
Facilità utilizzo	*	***	**
Presenza di esempi attendibili	**	*	***

Tabella 5.1: Confronto ambienti di sviluppo software

5.2.1 Zephyr, Mbed e STM32Cube

Inizialmente sono stati utilizzati tutti e tre gli ambienti per capire quale di questi facilitasse la programmazione dei device. In seguito è riportata la descrizione esaminandone vantaggi e svantaggi e l'esperienza nel loro utilizzo. La tabella 5.1 riassume tali caratteristiche.

Zephyr OS Per poter utilizzare Zephyr OS c'è la necessità di compiere operazioni complesse a seconda del proprio sistema operativo. Esiste infatti una guida che aiuta l'utente a scaricare sul proprio PC Zephyr OS, a configurarlo e a flashare i programmi all'interno della propria board. Le istruzioni sono raggiungibili alla pagina `https://docs.zephyrproject.org/latest/getting_started/index.html` e prevedono l'istallazione di alcune dipendenze (Cmake[1] e Device-Tree[2]), l'istallazione del codice sorgente di Zephyr in ambiente virtuale, l'esportazione del pacchetto Zephyr CMake (ciò consente a CMake di caricare automaticamente il codice standard richiesto per la creazione di applicazioni Zephyr) e l'istallazione di una Toolchain (un compilatore, un assemblatore, un linker e altri

[1]CMake è una famiglia di strumenti open source e multipiattaforma progettati per creare, testare e creare pacchetti di software. CMake viene utilizzato per controllare il processo di compilazione del software utilizzando semplici file di configurazione indipendenti dalla piattaforma e dal compilatore e generare makefile nativi e spazi di lavoro che possono essere utilizzati nell'ambiente del compilatore.

[2]Devicetree è una struttura dati per descrivere l'hardware. Piuttosto che codificare ogni dettaglio di un dispositivo in un sistema operativo, molti aspetti dell'hardware possono essere descritti in una struttura di dati che viene passata al sistema operativo al momento dell'avvio.

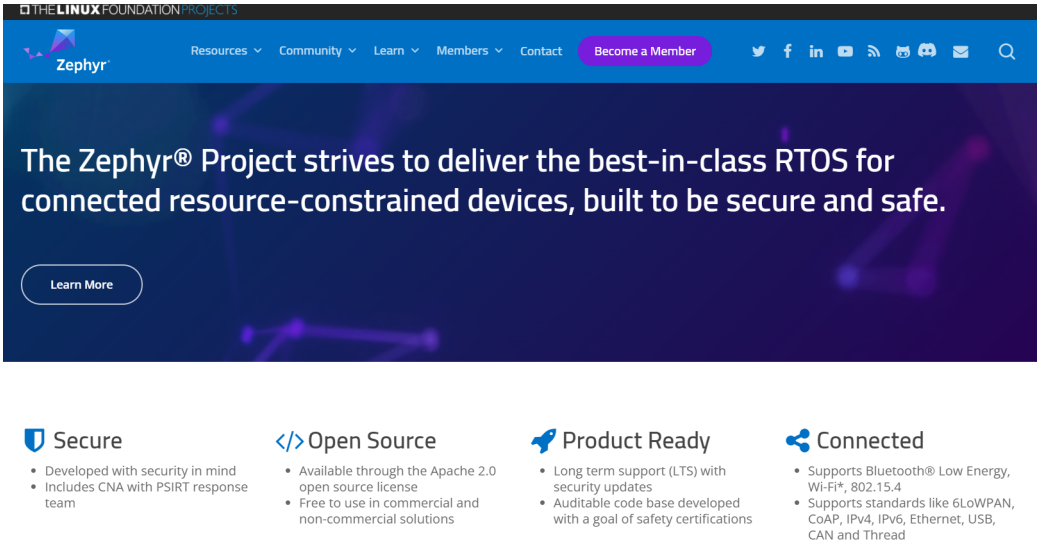


Figura 5.5: Mainpage Zephyr

programmi necessari per creare applicazioni Zephyr). Lo Zephyr Software Development Kit (SDK) contiene toolchain per ciascuna delle architetture supportate da Zephyr, ma non è disponibile su Windows. Zephyr è un progetto della Linux Foundation e permette di programmare ogni tipo di architettura hardware, non solo ARM. Questo è un vantaggio qualora ci fosse la necessità di cambiare tipologia di scheda. Un altro vantaggio è la presenza di molti codici demo per la programmazione.

L'esperienza nel suo utilizzo è stata inizialmente traumatica: la configurazione dell'ambiente sul proprio computer è complicata. Gli esempi sono molti ma non permettevano la facile creazione di una rete mesh BLE, per questo Zephyr non è stato scelto per programmare i device.

Mbed OS Mbed è un sistema operativo IoT progettato dalla ARM che permette la programmazione delle sole architetture ARM. Questo è un dettaglio molto limitante della piattaforma. Ha però molti vantaggi che ne rendono l'utilizzo molto più facile rispetto a Zephyr. Mbed infatti non prevede la necessità di scaricare nulla sul proprio PC, è totalmente online e ha un Cloud integrato. Ha la possibilità di visualizzare ed utilizzare molti codici o librerie scritti dagli altri utenti della piattaforma.

Seppur di facile utilizzo e di comprensione immediata, Mbed non offre la possibilità

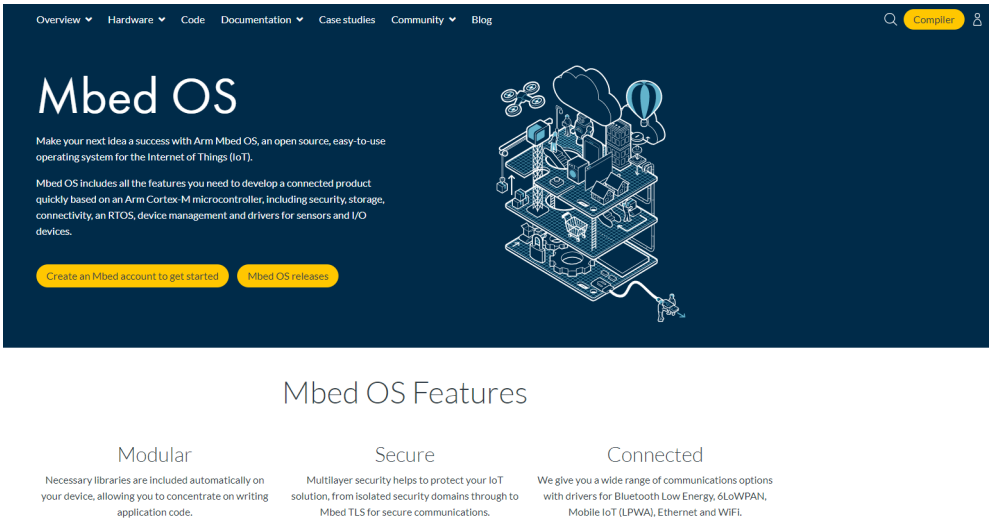


Figura 5.6: Mainpage Mbed

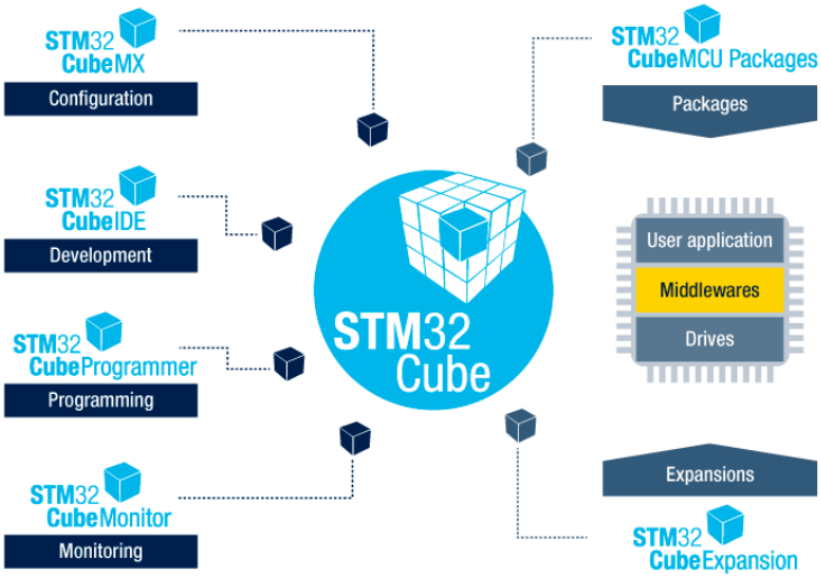


Figura 5.7: Ecosistema STM32Cube

di creare in modo semplice una mesh BLE, per questo si è scelto di abbandonare il suo utilizzo.

ST Toolchain L'ambiente ST per device STM32 prevede molti tool specifici per scopi diversi, come è possibile osservare dall'immagine 5.7. STM32Cube è una soluzione software completa per microcontrollori e microprocessori STM32. È destinato sia agli utenti che cercano un ambiente di sviluppo completo e gratuito per STM32, sia agli utenti che dispongono già di un IDE, tra cui Keil o iAR, in cui possono integrare facilmente i vari componenti. È una combinazione di strumenti e librerie software integrate che soddisfano tutte le esigenze di un ciclo di sviluppo del progetto completo, infatti ogni tool si occupa di una fase: STM32CubeMX, per la configurazione, STM32CubeIDE per lo sviluppo, STM32CubeProgramming per la programmazione e STM32CubeMonitor per il monitoraggio.

STM32CubeMX è uno strumento di configurazione per qualsiasi dispositivo STM32. Questa interfaccia utente grafica di facile utilizzo genera il codice C di inizializzazione per i core Cortex-M e genera l'origine dell'albero dei dispositivi Linux per i core Cortex-A.

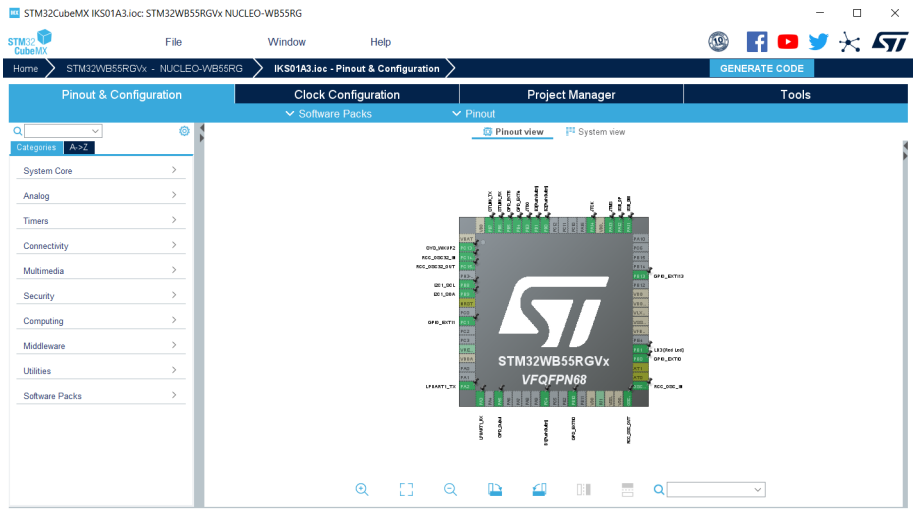


Figura 5.8: Interfaccia STM32CubeMX

STM32CubeIDE è un ambiente di sviluppo integrato. Basato su soluzioni open source come Eclipse o la toolchain GNU C/C++, questo IDE include funzionalità di report di compilazione e funzionalità di debug avanzate. Integra inoltre funzionalità aggiuntive presenti in altri strumenti dell'ecosistema, come l'inizializzazione HW e SW e la generazione di codice da STM32CubeMX.

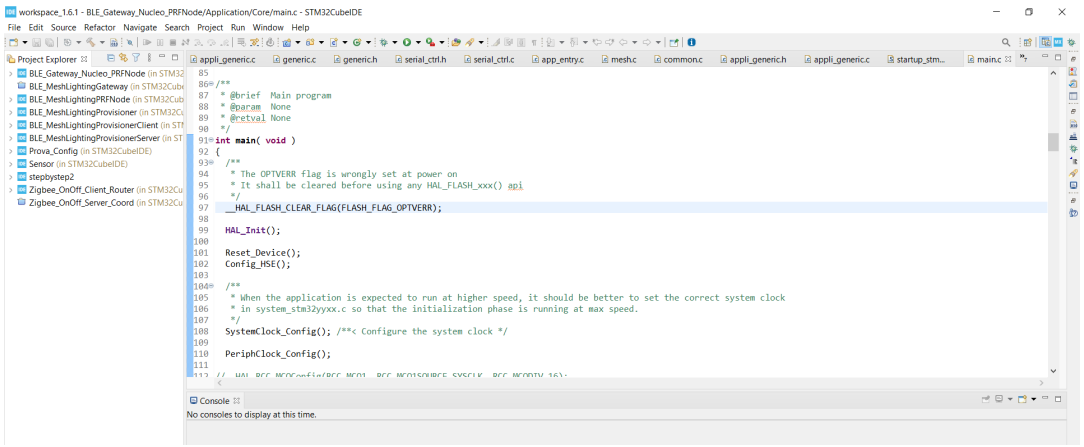


Figura 5.9: Interfaccia STM32CubeIDE

STM32CubeProgrammer è uno strumento di programmazione. Fornisce un ambiente semplice ed efficiente per leggere, scrivere e verificare dispositivi e memorie esterne tramite un'ampia varietà di mezzi di comunicazione disponibili (UART, USB DFU, I2C, SPI, ecc.).

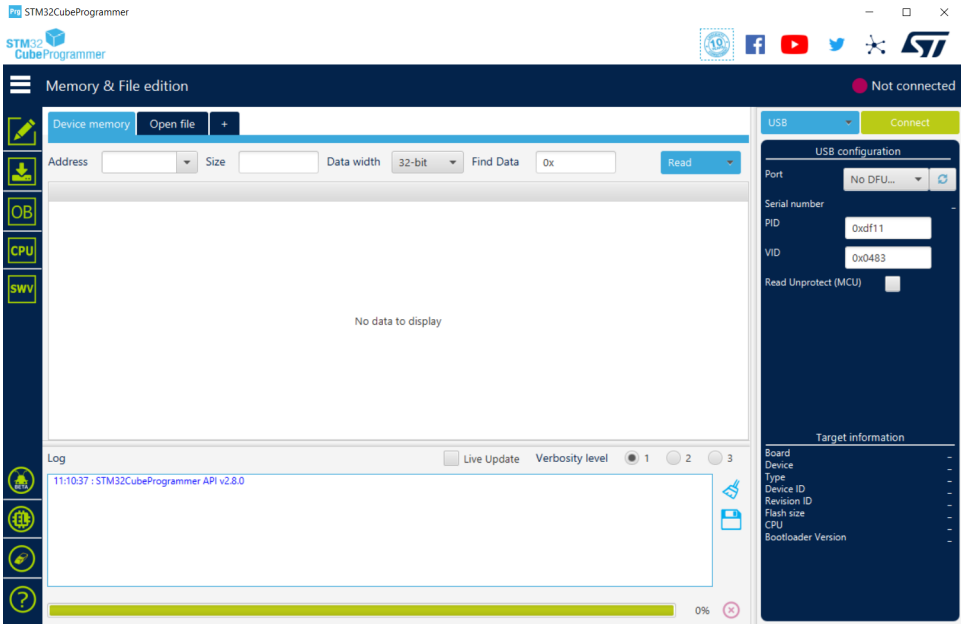


Figura 5.10: Interfaccia STM32CubeProgrammer

STM32CubeMonitor sono potenti strumenti di monitoraggio che aiutano gli

sviluppatori a mettere a punto il comportamento e le prestazioni delle loro applicazioni in tempo reale.

L'esperienza nell'utilizzo di questo ecosistema non è stata inizialmente buona. L'interfaccia utente, specialmente in STM32CubeIDE, non facilita la comprensione del funzionamento di tale piattaforma. Ma dopo aver seguito alcuni tutorial e grazie alla presenza di copiosi codici demo, tale ambiente di sviluppo si è dimostrato essere ottimale per l'implementazione del sistema IoT.
In particolare sono stati largamente utilizzati STM32CubeMX, in quanto la shield di sensori ha richiesto il cambiamento di alcuni pin, STM32CubeIDE, che ha permesso la modifica del codice e il caricamento di questo all'interno della board e STM32CubeProgrammer, che ha permesso l'aggiornamento del FUS[3] e dei firmware.

[3]Firmware Upgrade Services, ovvero un servizio che permette l'aggiornamento del firmware nella scheda.

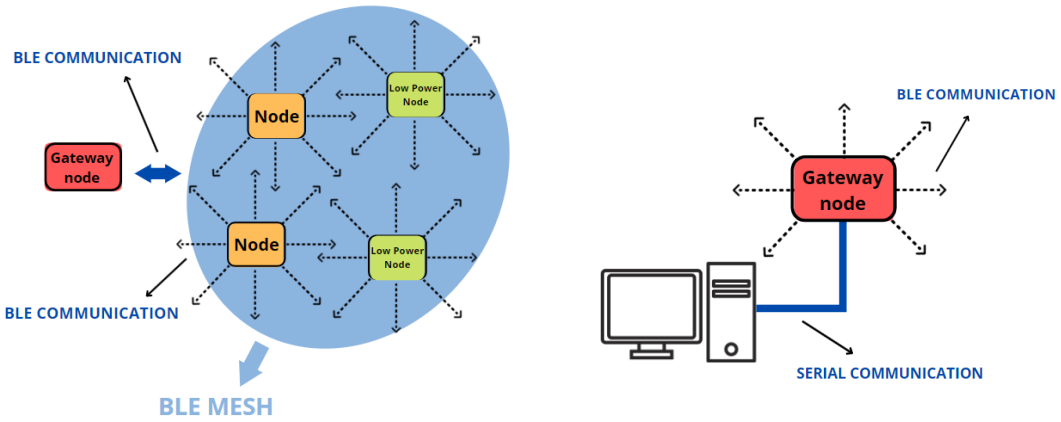
Capitolo 6

Progettazione e realizzazione

Questo capitolo è il fulcro di tale tesi. Nell'immagine 6.1 si ricorda l'architettura generale: le due parti principali progettate ed implementate sono la Mesh BLE e il Gateway. Vengono in seguito descritti in modo dettagliato i passi necessari all'implementazione di tali elementi.

6.1 Implementazione BLE Mesh

Per l'implementazione della Mesh BLE sono stati eseguiti alcuni passaggi principali:



(a) Mesh BLE (b) Gateway

Figura 6.1: Architettura generale del progetto

- Aggiornamento FUS e Wireless Stack - Per il corretto funzionamento del modulo BLE;

- Configurazione pin della shiel X-Nucleo-IKS01A3 - Per connettere i sensori sulla shield in modalità I2C in modo corretto;

- Firmware e comunicazione tra nodi - Descrive la struttura ed il funzionamento del del codice creato mostrando le modalità di trasmissione dati tra i device;

- Provisioning - Mostra le modalità che hanno permesso la configurazione dei nodi all'interno della mesh.

6.1.1 Aggiornamento FUS e Wireless Stack

Affinchè il modulo radio per la comunicazione BLE funzioni correttamente, è necessario aggiornare lo stack Wireless BLE. Questo processo avviene in due fasi: prima è necessario aggiornare il firmware sottostante sul coprocessore (FUS) e poi eseguire il flashing dello stack wireless, ovvero caricare i file binari nella scheda.
Tali file binari vengono scaricati da `https://github.com/STMicroelectronics/STM32CubeWB` e si trovano all'inteno di *STM32CubeWB/Projects/STM32WB_Copro_Wireless_Binaries/STM32WB5x*. All'interno di tale cartella vi è anche il file *Release Notes.html* che apre una pagina internet che spiega i passaggi necessari per aggiornare tali file all'interno della board e che fornisce gli indirizzi di memoria che devono essere usati per ogni binario, questi sono elencati in una tabella chiamata *Firmware Upgrade Services Binary Table* per l'aggiornamento del FUS e in *Wireless Coprocessor Binary Table* per i vari stack Wireless. Da queste tabelle si apprende che per la nostra board STM32WB55RG, il firmware FUS appartiene alla posizione 0x080EC000 e stm32wb5x_BLE_Stack_full_fw dovrebbe essere flashato all'indirizzo 0x080CA000.
La board può essere collegata al PC in due modi: usando ST-Link e usando la modalutà DFU (Device Firmware Update), che prevede delle operazioni fisiche sulla scheda come lo spostamento di uno switch e il collegamento di alcuni piedini. Operazioni descritte in figura 6.2. Il file *Release Notes.html* descrive come possono essere effettuati gli aggiornamenti in entrambe le modalità. L'aggiornamento del FUS e del Wireless Stack è stato effettuato con STM32CubeMX e nella modalità ST-Link, maniera più facile ed intuitiva che non prevede collegamenti fisici diversi sulla board.
Aprendo il software STM32CubeMX con la board collegata via USB al pc, la schermata che compare è quella in figura 6.3.

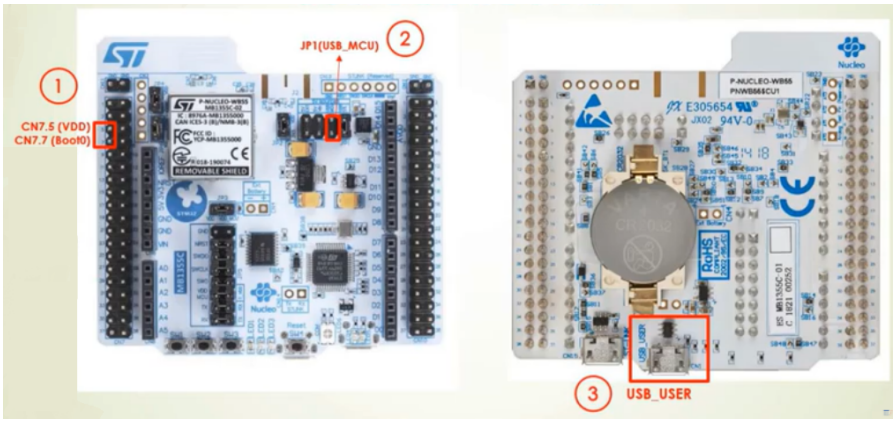


Figura 6.2: Modalità DFU

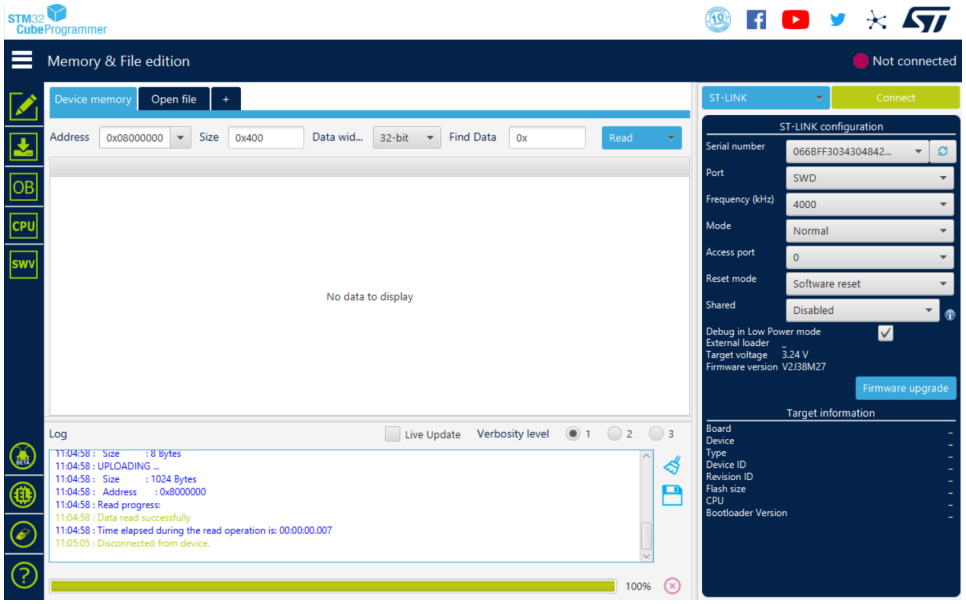


Figura 6.3: Schermata iniziale CubeMX

Sulla parte destra della figura vengono effettuate alcune scelte per la connessione della board. Vanno selezionate: ST-Link, porta SWD, modalità Normal e modalità Software reset.
Dopo aver cliccato su Connect si raggiunge la pagina Firmware Update Services dall'icona rappresentante le connessioni wireless del menù a tendina sulla sinistra (cerchiata in rosso), come è possibile vedere dall'immagine 6.4.

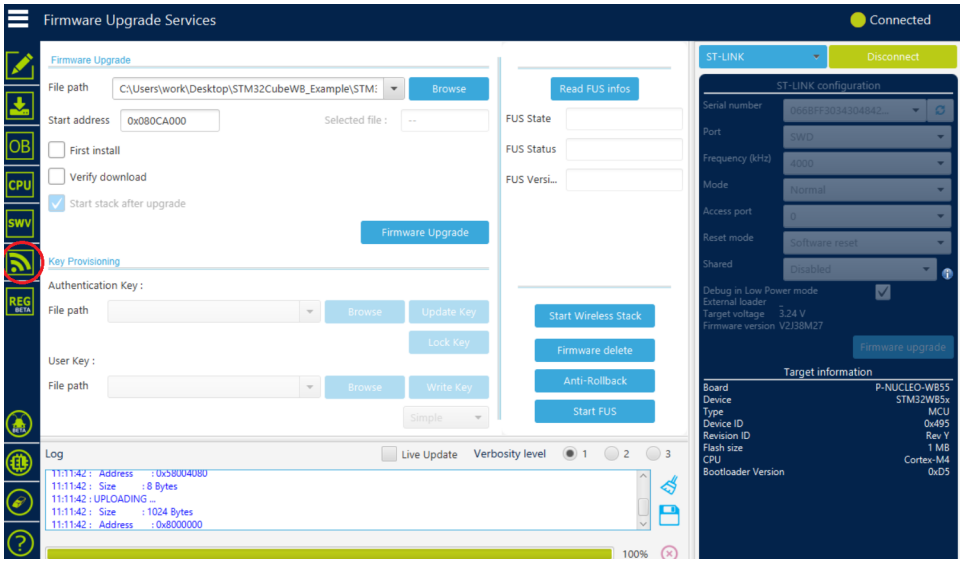


Figura 6.4: Firmware Update Services - CubeMX

In questa schermata le operazioni da compiere sono:

1. Cliccare su *Start FUS* per far partire tale servizio. In questo momento verranno lette le informazioni (State, Status, Version) sul FUS e verranno visualizzate in altro a destra della schermata 6.4;

2. Cliccare su *Firmware delete* per eliminare sulla scheda la versione vecchia del FUS e poterla flashare correttamente;

3. In Firmware Upgrade selezionare il file FUS_fw_0_5_3.bin a indirizzo 0x080EC000 e spuntare verify download, come è possibile vedere dall'immagine 6.5;

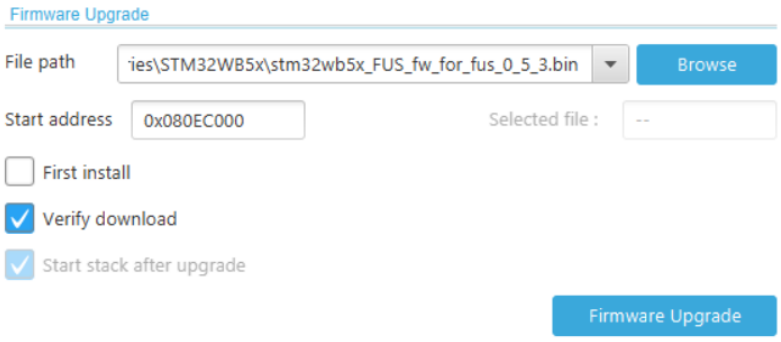


Figura 6.5: Update FUS parte 1 - CubeMX

4. In Firmware Upgrade selezionare il file FUS_fw.bin (contenente la versione 1.2.0 del FUS) a indirizzo 0x080EC000 e spuntare verify download, come è possibile vedere dall'immagine 6.6. È necessario compiere l'operazione 2 prima della 3 in quanto le versioni del FUS hanno una particolare compatibilità per essere aggiornate, prima infatti va aggiornato alla FUS version V0.5.3 per poter passare alla V1.2.0;

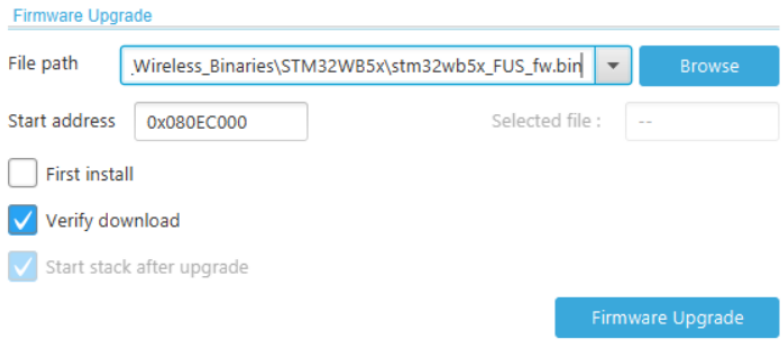


Figura 6.6: Update FUS parte 2 - CubeMX

5. In Firmware Upgrade selezionare file BLE_Stack_full_fw.bin a indirizzo 0x080CA000 e spuntare first install e verify download, come è possibile vedere dall'immagine 6.7.

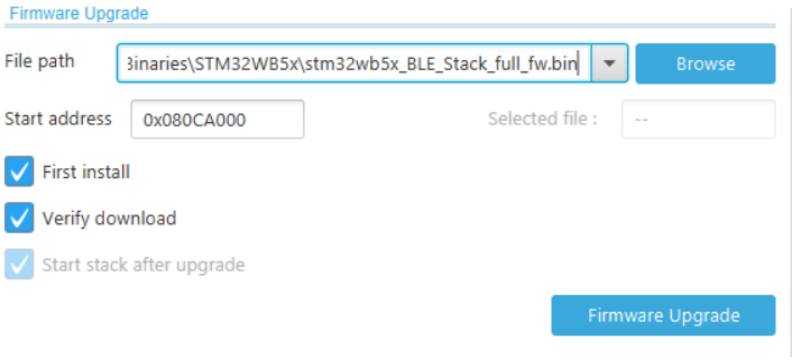


Figura 6.7: Update BLE Wireless Stack - CubeMX

Alla fine di questi passi risulterà correttamente aggiornato lo stack BLE, operazione necessaria per il corretto funzionamento del modulo radio.

6.1.2 Configurazione pin della shield X-Nucleo-IKS01A3

La shield X-Nucleo-IKS01A3 viene assemblata sopra la board Nucleo64 come un "panino", come è possibile vedere dall'immagine 6.8. Per il corretto funzionamento però ne vanno configurati i pin come è lungamente descritto nel documento [28].

Figura 6.8: X-Nucleo-IKS01A3 connessa alla board Nucleo-WB55RG

La configurazione viene effettuata tramite il software STM32CubeMX.
Una volta aperto, si crea un nuovo progetto selezionando la board, cliccando come in figura 6.9, si apre il selettore delle board, quello in figura 6.10 e si sceglie la NUCLEO WB55RG.

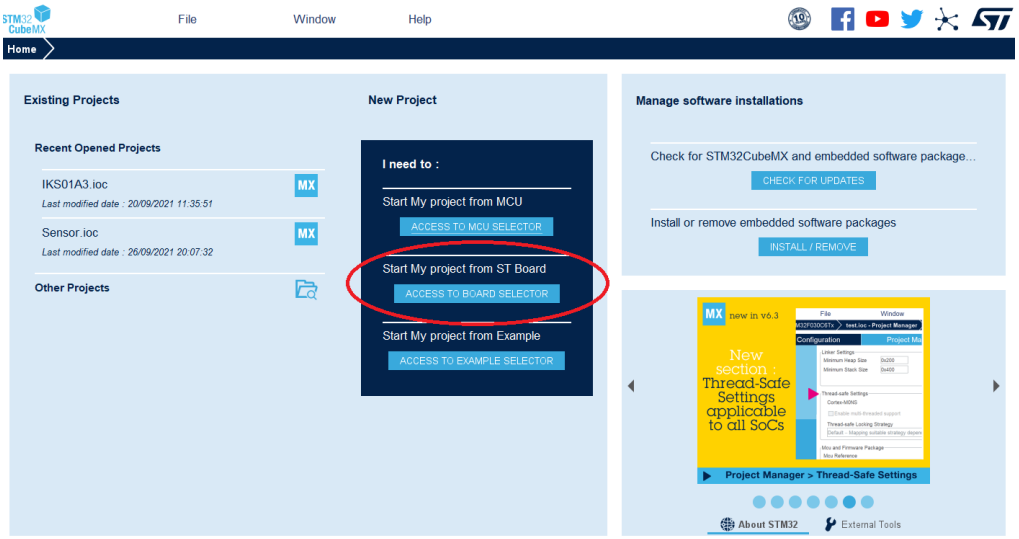


Figura 6.9: Schermata STM32CubeMX dove creare un nuovo progetto

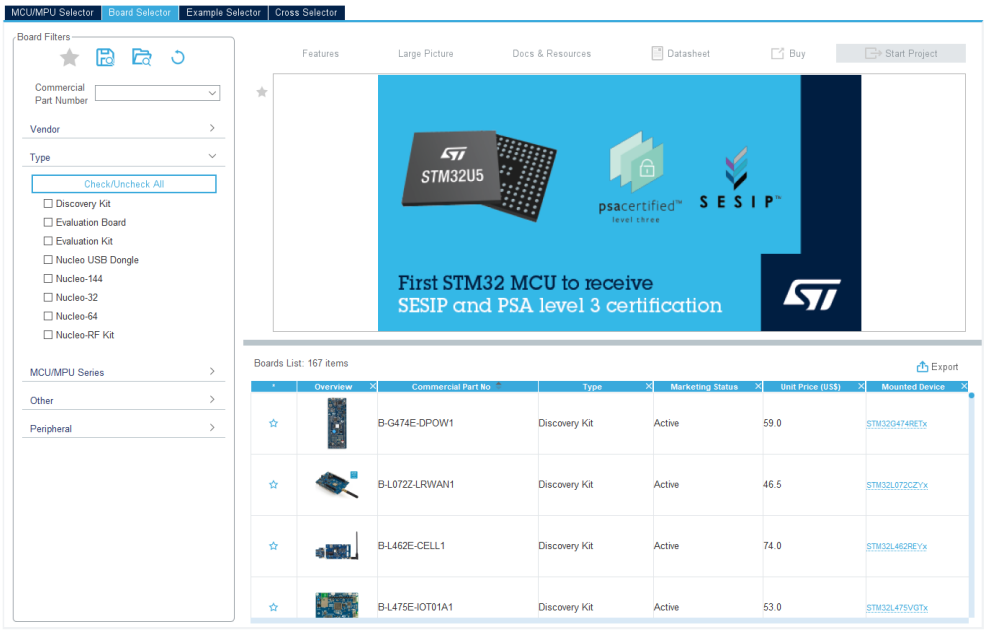


Figura 6.10: Schermata STM32CubeMX Per selezionare la board

A questo punto la finestra che si apre è quella in figura 6.11, con la configurazione dei pin di default.

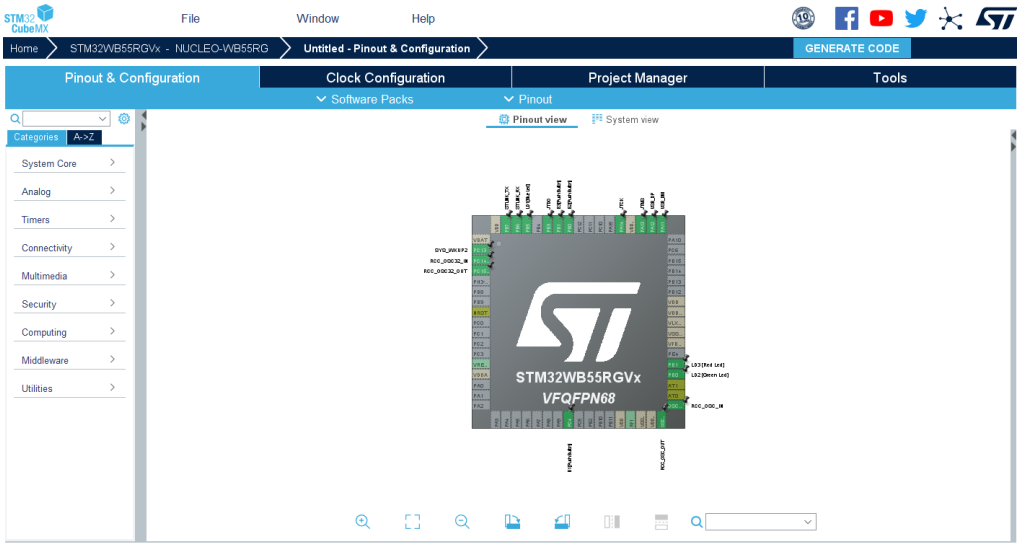


Figura 6.11: Schermata STM32CubeMX Pinout & Configuration

Per la corretta configurazione vi sono alcune operazioni da compiere, come è possibile leggere dal documento citato sopra.
Dalla tab Pinout & Configuration:

- dallo schema Pinout, cliccare su PB8 e settare come I2C1_SCL;
- dallo schema Pinout, cliccare su PB9 e settare come I2C1_SDA;
- dalla categoria "Connectivity", abilitare I2C1 cone I2C;
- dalla categoria "Connectivity" abilitare la LPUSART1 in Asynchronous mode;
- dallo schema Pinout settare i pin come in taballa 6.1.

A questo punto è necessario installare l'estensione per la shield, cliccando il alto su "Software packs" e poi "Select components" è possibile scaricare il pacchetto "Board Extension IKS01A3" come si vede in figura 6.12.

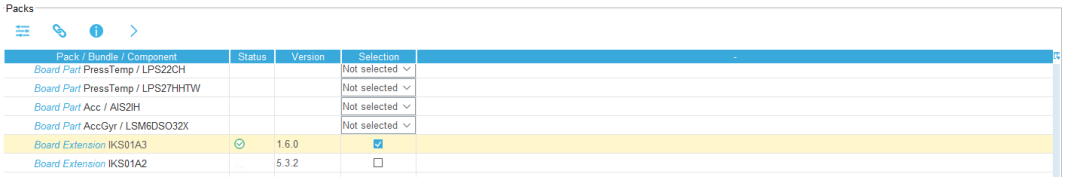


Figura 6.12: Schermata STM32CubeMX di selezione del pacchetto di estensione per la shield IKS01A3

PIN	Modalità	Etichetta
PB5	GPIO_EXTI5	
PB10	GPIO_EXTI10	
PB0	GPIO_EXTI0	
PB13	GPIO_EXTI13	B1 [Blue PushButton]
PC1	GPIO_EXTI1	
PA5	GPIO_Output	LD2 [Green Led]
PA2	LPUSART1_TX	LPUSART_TX
PA3	LPUSART1_RX	LPUSART_RX

Tabella 6.1: Schema Pinout IKS01A3 con board Nucleo64

Adesso nella categoria di sinistra "Software packs" si seleziona STMicroelectronics-X-Cube-MEMS e spuntando "Board Extension IKS01A3" si sceglie in IKS01A3 BUS IO driver la connessione I2C:I2C, come descritto in figura 6.13. Da notare che sarebbe possibile scaricare il pacchetto "Device MEMS1 Application" contenente alcuni esempi già funzionanti. Ma in tale progetto l'utilizzo della shield è stato integrato in una mesh BLE quindi il codice è stato creato ex novo.

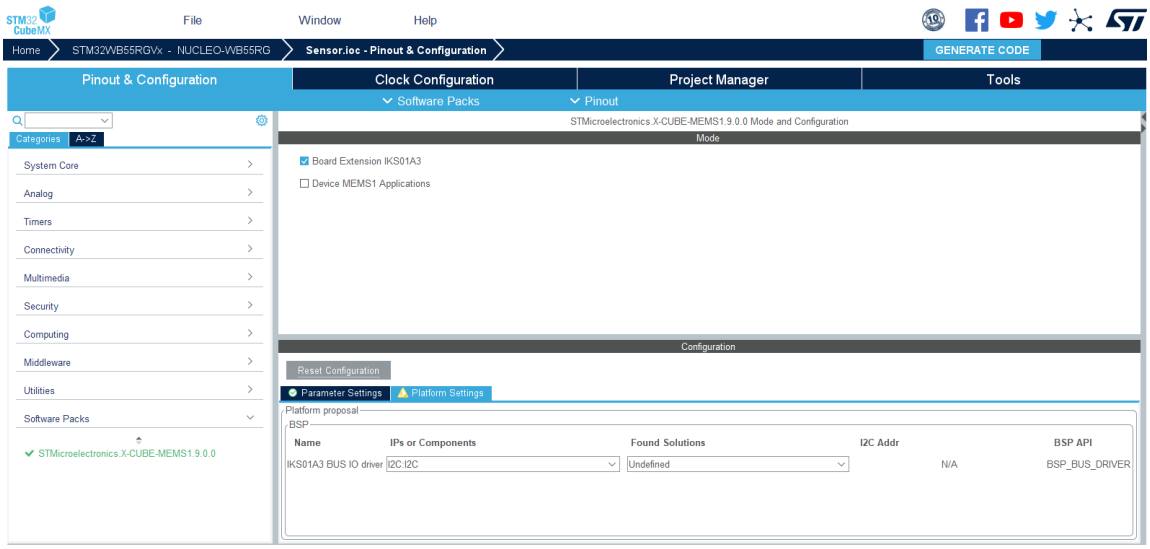


Figura 6.13: Schermata STM32CubeMX Software packs

La configurazione finale corretta è visibile nell'immagine 6.14, una volta salvata il software genera un'applicazione per la STM32CubeIDE e un .ioc file, file che visivamente fa vedere il Pinout.

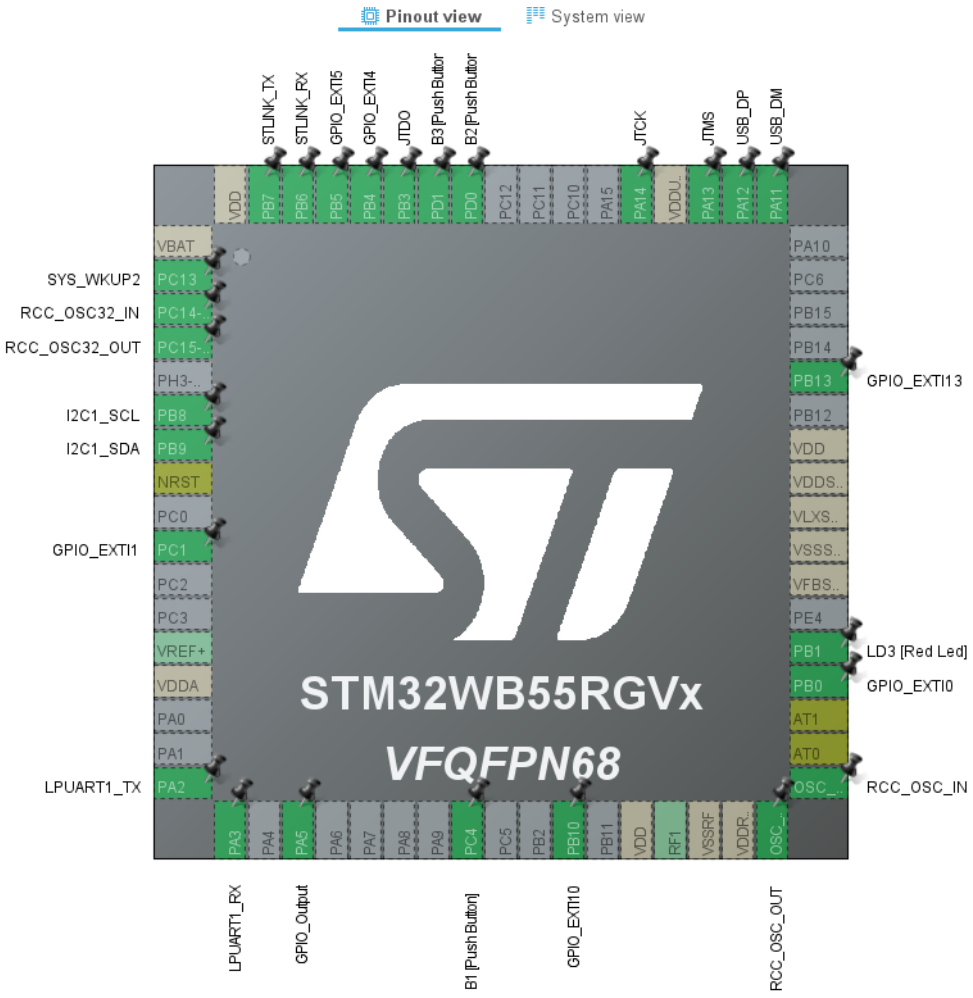


Figura 6.14: Pinout finale

6.1.3 Firmware e comunicazione tra i nodi

Sono stati creati due codici differenti per Gateway e Sensore. Questi sono accessibili dal GitHub `https://github.com/SofiaCaterini/Gateway_Nucleo`, per il Gateway e da `https://github.com/SofiaCaterini/Sensor_Nucleo` per il Sensore.
In entrambi la struttura generale è quella in figura 6.15. Un progetto così definito rende possibile l'implementazione della BLE Mesh ed è lo scheletro delle due applicazioni da creare. I dettagli di ogni file sono descritti nel documento [9], da cui è stata ripresa tale immagine.
Si procede col descrivere le parti principali del progetto.

I file contornati in giallo rappresentano i file middleware della mesh BLE e includono i servizi BLE e l'implementazione dei vari modelli.
In dettaglio possiamo evidenziare:

- *svc_ctl.c* Inizializza lo stack BLE e gestisce gli eventi del servizio GATT dell'applicazione;
- *mesh.c* Inizializza la libreria mesh BLE in cui vengono create e gestite le caratteristiche della mesh. Include l'aggiornamento delle caratteristiche, la possibilità di ricevere le notifiche o scrivere comandi e di creare il collegamento tra lo stack BLE e la parte applicativa. In questa parte del codice vengono analizzate le caratteristiche del device, se è provisionato o meno, gli elementi che integra e i modelli che sono supportati e vengono inizializzati nel modo corretto;
- *common.c* Include le funzioni utilizzate negli altri file Middlewere dei modelli;
- *config_client.c* Modello di configurazione utilizzato dal provisioner per provisionare e configurare un nodo;
- *generic.c* Gestisce tutti i modelli Generic Server, ne definisce una serie di stati e messaggi. I modelli Generic sono esplicitamente definiti come funzionalmente non specifici;
- *generic_client.c* Gestisce tutti i modelli Generic Client, ne definisce una serie di stati e messaggi;
- *light.c* Gestisce tutti i modelli Light Server, ne definisce una serie di stati e messaggi. Esso definisce una serie di funzionalità per il controllo della luce;
- *light_client.c* Gestisce tutti i modelli Light Client, ne definisce una serie di stati e messaggi;
- *light_lc.c* Gestisce tutti i modelli Light Control Server, ne definisce una serie di stati e messaggi. Questo modello di controllo gestisce comportamenti specifici;
- *sensors.c* Gestisce tutti i modelli Sensor Server, ne definisce una serie di stati e messaggi. Definisce una modalità standard di interfacciamento con i sensori;
- *vendor.c* Gestisce tutti i modelli Vendor Server, ne definisce una serie di stati e messaggi;
- *time_scene.c* Gestisce tutti i modelli Time e Scene Server, ne definisce una serie di stati e messaggi;

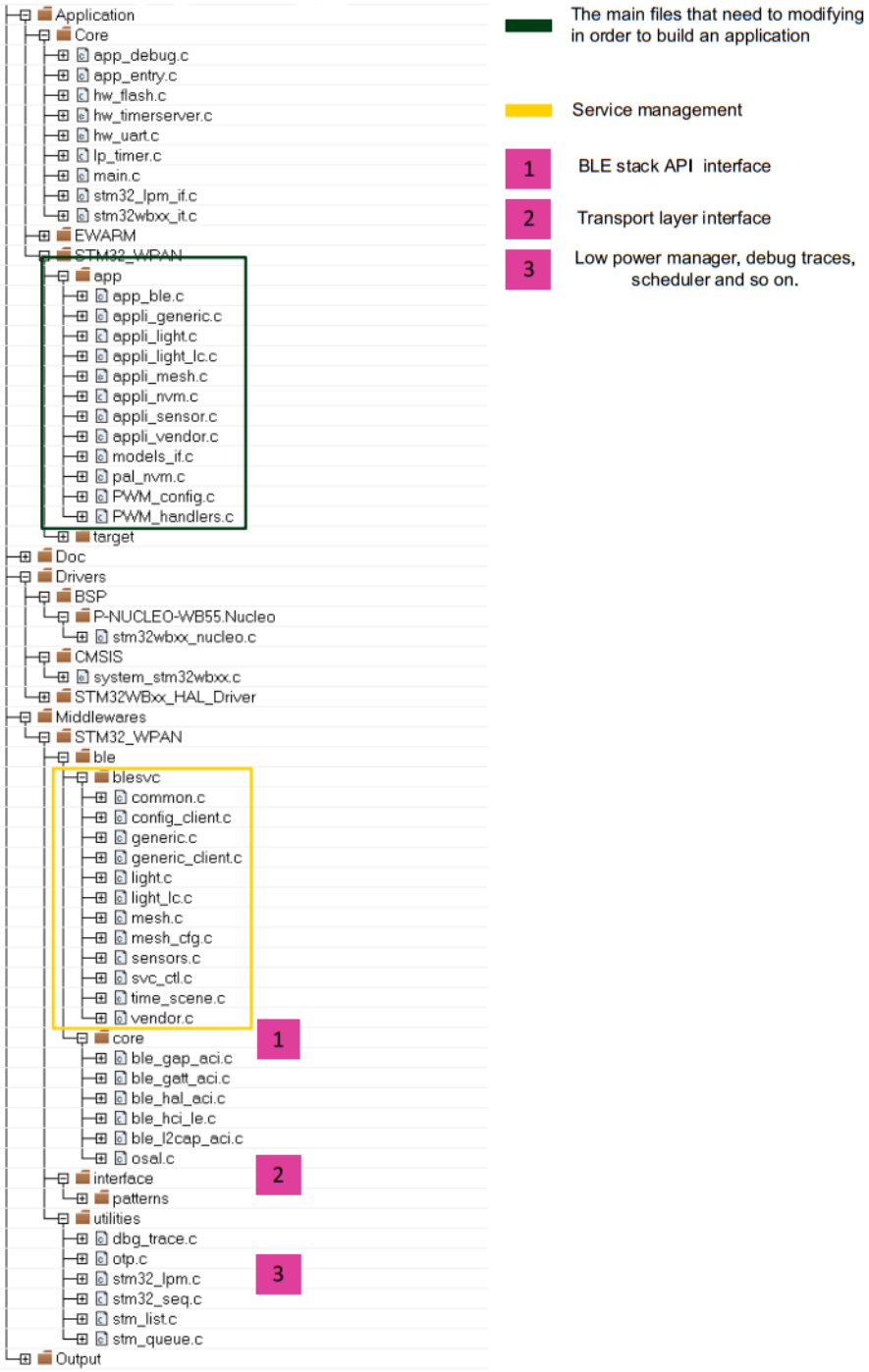


Figura 6.15: Struttura progetto

- *mesh_cfg.c* Inizializza tutte le strutture dati associate ai modelli in uso.

All'interno della sottocartella *Core* i file più importanti sono:

- *app_entry.c* Per inizializzare il livello di trasporto BLE e il BSP (Board Suppor Package) per la gestione ad esempio di LED, bottoni e così via;
- *main.c* Inizializza e fa avviare la piattaforma STM32WB;
- *stm32wbxx_it.c* Contiene le principali routine del servizio di interrupt. Questo file fornisce il modello per tutti i gestori di eccezioni e le routine dei servizi degli interrupt delle periferiche;

All'interno della sottocartella *STM32_WPAN\App* sono presenti i file da modificare per cambiare le funzionalità dell'applicazione. Infatti nei codici finali sono state apportate modifiche ad alcuni di questi, quelli che implementavano le funzionalità dei modelli Sensor, Config e Generic. I file più rilevanti sono:

- *app_ble.c* Inizializza l'applicazione BLE, gestisce il GAP e connessioni tra device come l'advertising, la scansione ecc...;
- *appli_config_client.c* Interfaccia per l'applicazione del Configuration Client Model;
- *appli_generic.c* Applicazioni dei Generic Models: come l'accensione e lo spegnimento di un nodo;
- *appli_light.c* Applicazioni del modello Light Server: come luci regolabili e che cambiano colore;
- *appli_light_client.c* Applicazioni del modello Light Client: come l'invio di comandi a luci regolabili;
- *appli_light_lc.c* Applicazioni del modello Light Control per produrre comportamenti specifici attivati da sensori;
- *appli_mesh.c* Gestisce tutte le funzioni di callback della libreria mesh e le azioni associate al pulsante utente;
- *appli_sensor.c* Applicazione del modello Sensor;
- *appli_vendor.c* Applicazione modello Vendor;
- *models_if.c* Gestisce l'interfaccia con i modelli e pianifica tutti i processi associati.

In questa cartella sono presenti i file header per la configurazione di ogni nodo, infatti in *mesh_cfg_usr.h* è possibile definire il tipo di nodo (Proxy, Friend, Relay o Low Power) e quali tipi di modelli implementare sul nodo attraverso delle define (es. #define ENABLE_FRIEND_FEATURE).
Per i dettagli sulle funzioni e le callback implementate nell'applicazione si rimanda al file [9].

Firmware Sensore Il Sensore supporta la MEMS shield e quindi ha bisogno della configurazione dei pin descritta nella sezione *Configurazione pin della shield X-Nucleo-IKS01A3* ed inoltre deve pubblicare i valori letti in modo tale che il Gateway possa riceverli.

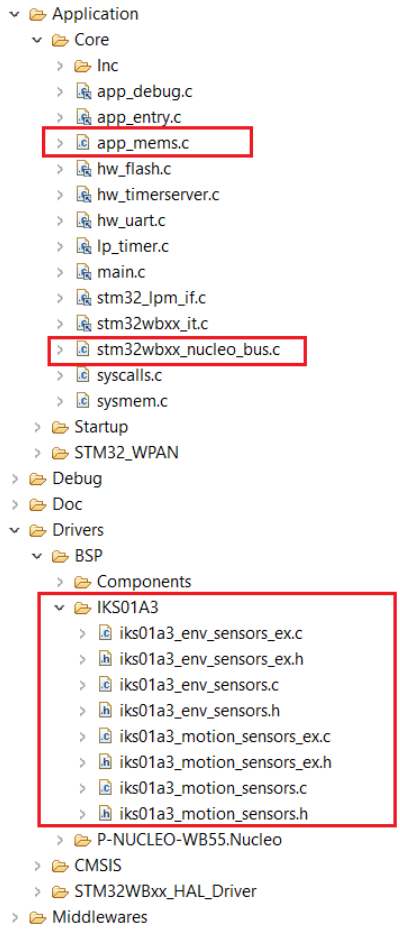


Figura 6.16: Struttura progetto Sensore

Questa comunicazione tra il Sensore e il Gateway è resa possibile dal Sensor

Model Server in cui vengono settati i dati da inviare e dal Sensor Model Server Setup in cui viene deciso l'intervallo di pubblicazione. Quindi nel codice visto in precedenza, scheletro della mesh BLE, sono stati aggiunti alcuni file, come è possibile vedere nella figura 6.16. *app_mems.c* rende possibile l'inizializzazione e l'avvio della shield IKS01A3 e legge i valori dei sensori. Ciò è possibile grazie ai driver, che sono stati integrati al progetto nella cartella *Drivers\BSP\IKS01A3* e quelli dentro *Drivers\BSP\Components*, che sono relativi ai singoli componenti della shield come ad esempio il sensore lps22hh. Il file *stm32wbxx_nucle_bus.c* rende possibile la giusta configurazione dei pin assieme al file *stm32wb_nucleo.c* dentro *Drivers\BSP\P-NUCLEO-WB55.Nucleo*, questi "traducono" il file .ioc prodotto da STM32CubeMX per il Pinout.
Inoltre nel codice del Sensore è stato modificato il file *appli_generic.c* in modo tale che ogni volta che il nodo riceve un comando di accensione del led esso genera una callback che fa stampare una risposta sul nodo Gateway. Questo servirà per identificare un nodo all'interno della rete per poterlo rendere tracciabile. Un altro file cambiato è stato *appli_sensor.c* all'interno di *STM32_WPAN\App* per la corretta pubblicazione dei valori, presi dai sensori.

Con il codice così implementato il nodo Sensore è definito con le funzionalità di Proxy, Relay e Friend e implementa i modelli di GenericOnOff Client, Sensor Server e Sensor Setuo Server, è in grado di pubblicare la temperatura rilevata dal sensore sulla shield IKS01A3 e di pubblicare una risposta ad un comando di accensione del led generato dal Gateway.

Firmware Gateway Il codice per la creazione del Gateway è lo stesso della struttura della figura 6.15. In esso però è stato modificato il file *mesh_cfg_usr.h* in modo tale da attivarne il controllo seriale. Sono state infatti effettuate le modifiche dell'immagine 6.17. Ciò rende possibile inviare comandi seriali ad altri nodi

```
/* Enables the serial interface using Uart */
#define ENABLE_SERIAL_INTERFACE 1
#define ENABLE_UT 0
#define ENABLE_SERIAL_CONTROL 1
#define ENABLE_APPLI_TEST 0
```

Figura 6.17: Inizializzazione Gateway seriale

connessi alla rete. Infatti con "atcl 0003 8202 01 00" è possibile inviare il comando di Generic On/Off Set Ack al nodo 0003, in modo tale che il led si accenda (01) (00 per spegnerlo) e con la richiesta di risposta (00). Nel file [9] sono definiti tutti i comandi possibili, come quello riportato in figura 6.18.

Con il codice così implementato il nodo Gateway è definito con le funzionalità di

Identifier	Destination address(range)	Op Code	Data	Description
ATCL	0x0001-0xFFFF	0x8202	0x01 00	Generic ON/OFF Set Ack

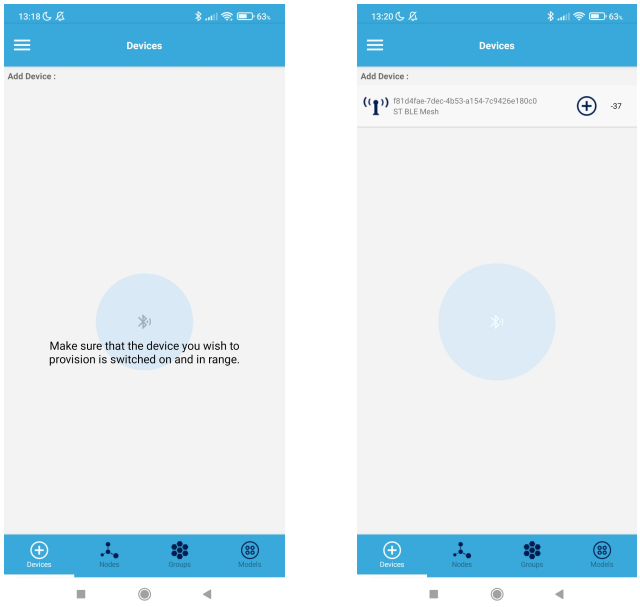
Figura 6.18: Codifica comendi Gateway seriale

Proxy, Relay e Friend e implementa i modelli di GenericOnOff Server e Sensor Client, è in grado di ricevere la temperatura rilevata dal Sensore con una stringa del tipo "EVT = Sensor_Value ; srcAddr = 03 ; type = temp ; value = 21" , di pubblicare un comando di accensione del led sul Sensore e di riceverne la risposta con la stringa del tipo "ANS = Generic_OnOff ; srcAddr = 03 ; status = 01".
Queste stringhe sono adesso rese ben riconoscibili da un'app seriale che ne permette la gestione: i valori rilevati verranno inviati via MQTT ad un'applicazione consumer che sarà collegata a iChain, inoltre da un'interfaccia web sarà possibile visualizzare la lista dei nodi attivi ed inviare loro dei comandi di accensione del led, con lo scopo di individuarli e salvarne le coordinate.

6.1.4 Provisioning

Il provisioning dei nodi all'intero della rete è stato effettuato mediante l'app ST BLE Mesh, disponibile sia per Android, scaricabile al link `https://play.google.com/store/apps/details?id=com.st.bluenrgmesh&hl=it&gl=US` che iOS `https://apps.apple.com/us/app/st-ble-mesh/id1348645067`.
Da quest'app è possibile, nella sezione "Device" ricercare gli unprovisioned device, come si può vedere dalla figura 6.19 e una volta cliccato sull'icona del "+" è possibile procedere col provisioning e con la configurazione.
I passaggi di questi processi sono mostrati in figura 6.20. Subito dopo aver cliccato sul "+" appare la schermata di figura 6.20(a), si clicca su "Fast Provisioning" e in questo modo il provisioning ha inizio in modo automatico fino al suo completamento (figura 6.20(b)). Qui si può procedere alla configurazione in modo "Quick Configuration" ovvero per ogni modello viene settata la stessa chiave di applicazione e gruppo di default dei device su cui fare subscribe e da usare come target di pubblicazione, altrimenti si può selezionare modello per modello la chiave e i device a cui fare publish/subscribe, cliccando nella schermata di figura 6.20(c) su "Configure". In questo caso è stata scelta la configurazione veloce, in cui c'è solo bisogno di scegliere la chiave di applicazione (che può essere creata ex-novo), come si vede in figura 6.20(d). Qui è stata scelta "AppKey0". Dopo la selezione si è aperta la schermata 6.20(e) in cui è possibile scegliere i device publish/subscribe (sono state lasciate le spunte di default) e cliccando su "Add Configuration" la configurazione inizia in modo automatico fino al suo completamento visibile in figura 6.20(f).

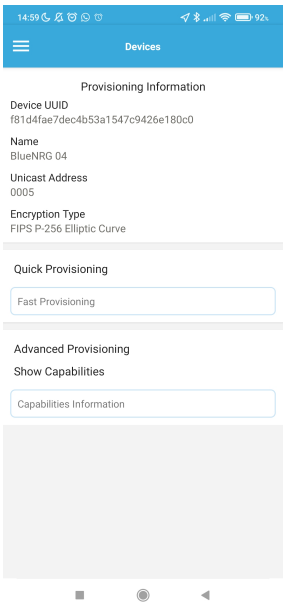
Una volta che il Sensore è stato correttamente configurato deve essere riavviato affinchè riceva il parametro di intervallo di pubblicazione nel modo corretto.
Nel caso in cui si scelga la configurazione manuale i parametri da scegliere sono elencati nelle tabelle 6.2 e 6.2. Nel caso della configurazione automatica bisogna assicurarsi che i modelli Sensor Client del Gateway e Generic On Off del Sensore rispettino i parametri della tabella.
Una volta seguiti questi passi i nodi sono correttamente aggiunti alla rete e configurati. All'interno dell'app possono essere visualizzati tutti i nodi della rete, con le proprie caratteristiche (P sta per Proxy, R per Relay e F per Friend) e il proprio uuid, come in figura 6.21(a).
Cliccando sull'icona delle impostazioni di ogni device se ne osservano le impostazioni, come in figura 6.21(b), in tale schermata si vede l'indirizzo univoco del nodo nella rete, se ne può modificare il nome e si può rimuovere dalla mesh.
Cliccando invece sull'icona delle impostazioni dell'elemento dentro ad un nodo si apre la schermata di figura 6.22, in cui si possono modificare le configurazioni dei modelli.



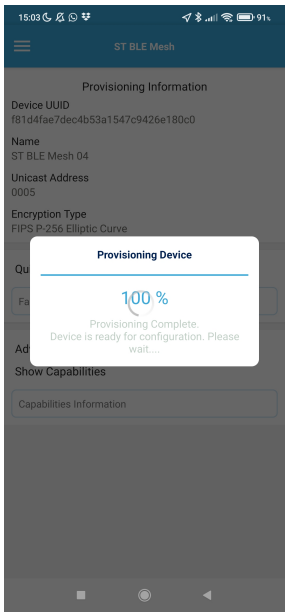
(a) Nessuno nelle vicinanze

(b) Un device unprovisioned nelle vicinanze

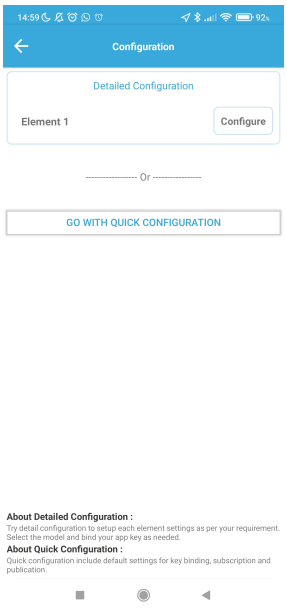
Figura 6.19: Ricerca degli unprovisioned device



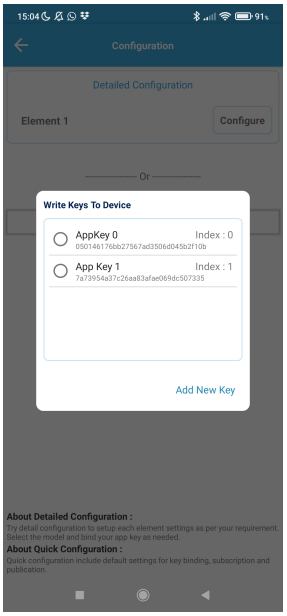
(a) Informazioni del provisioning



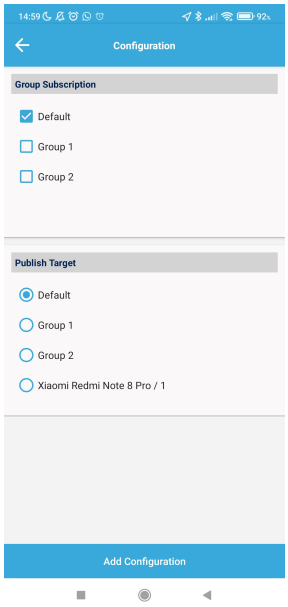
(b) Completamento del provisioning



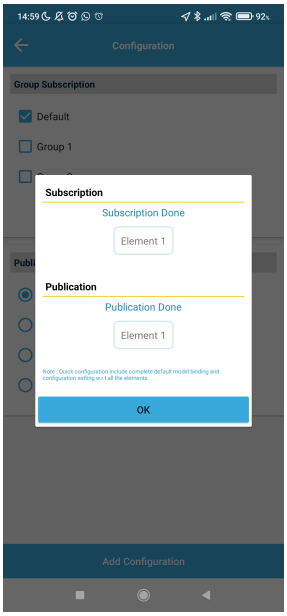
(c) Quick Configuration o configurazione manuale



(d) Scelta della chiave di applicazione



(e) Scelta device publish/subscribe



(f) Completamento della configurazione

Figura 6.20: Provisioning e configurazione

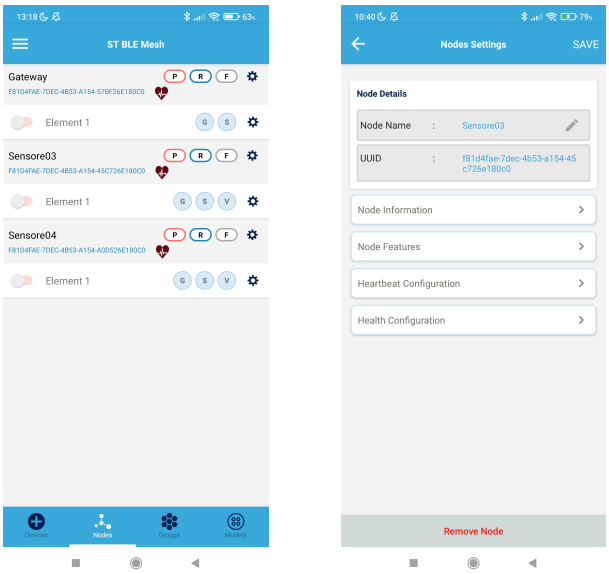
Modello	AppKey	Subscribe List	Publish Address
Generic On Off Server	AppKey0	c000[1]	c000
Generic Level Server	AppKey0	c000	c000
Generic Power On Off Server	AppKey0	c000	c000
Generic Power On Off Server Setup	AppKey0	c000	c000
Sensor Client	AppKey0	c000	None

Tabella 6.2: Parametri Modelli nella Configurazione del Gateway

Modello	AppKey	Subscribe List	Publish Address
Configuration Client	None	None	None
Generic On Off Server	AppKey0	None	c000
Generic Level Server	AppKey0	c000	c000
Generic Power On Off Server	AppKey0	c000	c000
Generic Power On Off Server Setup	AppKey0	None	c000
Sensor Server	AppKey0	c000	c000
ST Vendor Server	None	c000	None

Tabella 6.3: Parametri Modelli nella Configurazione del Sensore

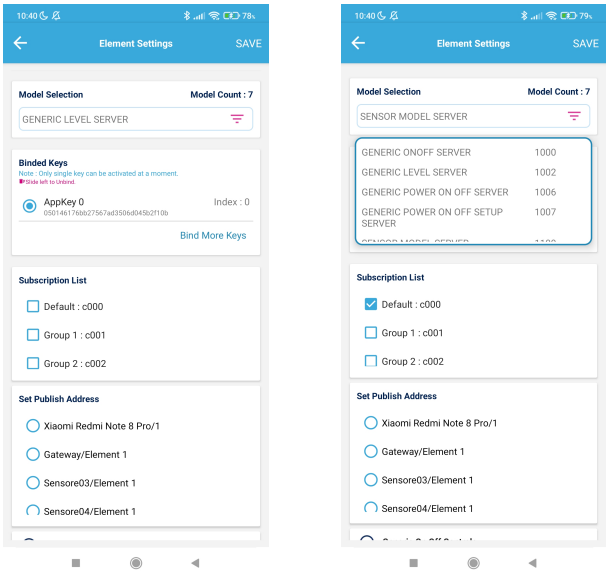
[1]Gruppo di default che indica tutti i device della rete.



(a) Device della rete

(b) Informazioni sui device

Figura 6.21: Device



(a) Element settings

(b) Lista modelli per ogni elemento

Figura 6.22: Elementi

6.2 Progettazione Gateway

Il progetto e l'implementazione del Gateway è una parte fondamentale di questo lavoro. Esso è stato pensato in modo da mantenere il progetto persistente ai cambiamenti di scenario applicativo che porterebbero a cambiamenti della tipologia di rete.

6.2.1 Ruolo

Il Gateway ha un ruolo principale in tale tesi. Esso infatti collega l'implementazione a basso livello del Sistema IoT con la piattaforma iChain. Ciò è possibile grazie a due principali applicazioni:

- L'app Seriale, che permette di configurare i device di rete, mandare loro dei comandi e agire sui dati rilevati da questi, inviandoli tramite MQTT[2];
- L'app Consumer, che cattura i dati da MQTT e li codifica e li invia ad iChain.

L'utilizzo di MQTT è vantaggioso per due motivi significativi. Esso è un protocollo di comunicazione svincolato da qualsiasi tipologia di rete e può essere quindi utilizzato nel caso di reti BLE o LoRa o qualsivoglia altra tecnologia. Nel caso quindi di un cambiamento di scenario che rendesse necessario il cambio di protocollo di comunicazione (ad esempio a causa di dati che devono essere inviati a distanze maggiori), la struttura dell'app siffatta resterebbe uguale. Il vantaggio principale è quello di rendere significativi i dati per EPCIS. I dati finora catturati dai device del sistema IoT non sono completi per diventare eventi EPCIS in quanto mancano le informazioni relative al WHERE. Tali informazioni di tracciabilità sono aggiunte in un secondo momento grazie proprio all'utilizzo di MQTT e possono derivare da più fonti, nel nostro caso dalle coordinate geografiche che inserisce l'utente nel momento di configurazione dei device, ma potrebbero anche provenire da rilevazioni GPS sui decise di rete. Ad ora è stata implementata solo l'app Seriale e verrà quindi descritta dettagliatamente nei prossimi paragrafi.

6.2.2 Struttura App Seriale

Il progetto di tale app è presente su GitHub all'indirizzo `https://github.com/SofiaCaterini/app_seriale`. La sua struttura è descritta nella figura 6.23.
L'applicazione è infatti formata da una parte di backend e una di frontend. I file di backend sono:

[2]MQTT o Message Queue Telemetry Transport è un protocollo di messaggistica per l'IoT di tipo publish/subscribe.

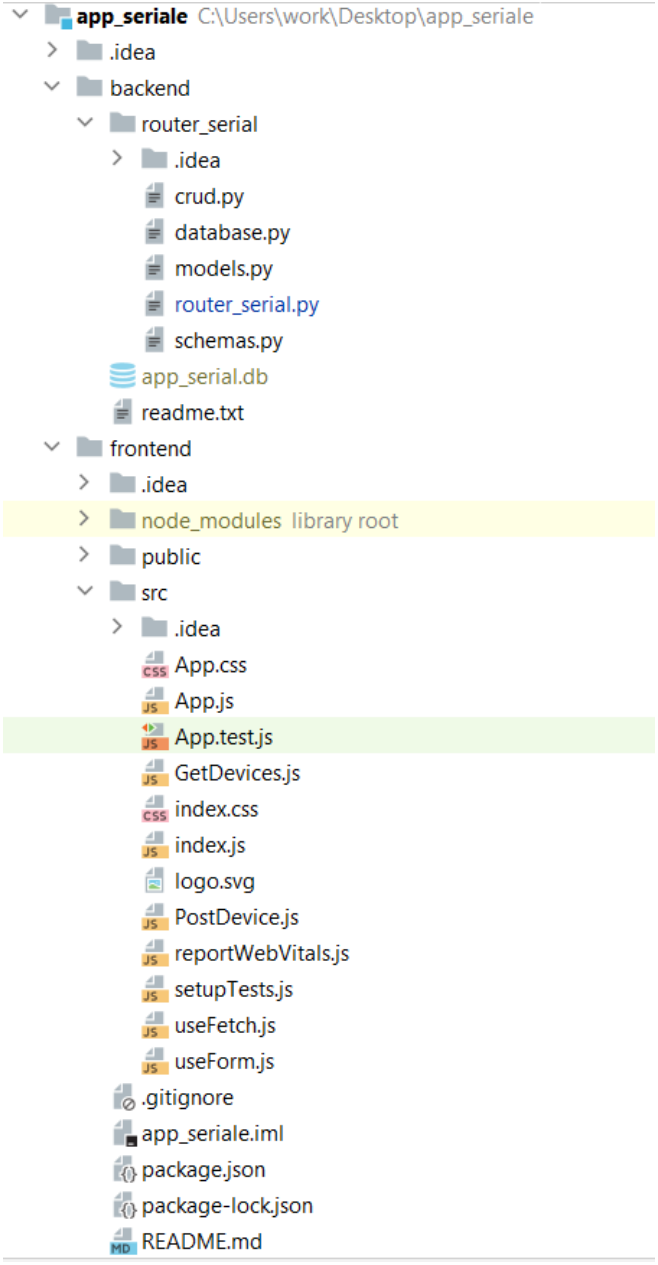


Figura 6.23: Struttura App Seriale

```python
class Device(Base):
    __tablename__ = "Devices"

    id = Column(String, primary_key=True, index=True)
    status = Column(String, index=True)
    time_last_measurement = Column(String, index=True)
    sensor_type = Column(String(100))
    measurement = Column(String(100))
    region = Column(String(255))
    x_location = Column(String(255))
    y_location = Column(String(255))
    z_location = Column(String(255))
    characteristics = Column(String(255))
```

Figura 6.24: Classe Device

- *crud.py* dove avviene l'accesso, la modifica e la raccolta dei dati. CRUD è infatti l'acronimo di Create Read Update Delete e in tale file sono implementati i metodi per la gestione dei device;
- *database.py* è il punto di accesso al database SQLite *app_serial.db*, che viene creato ed inserito sempre all'interno dei file di backend;
- *models.py* dove è implementata la classe Device, classe base per il salvataggio dei dati nel DB. Come si può vedere dalla figura 6.24, la classe Device ha attributi come l'ID, chiave univoca e primaria, lo Status ("ON" o "OFF"), il Tempo relativo all'ultima misurazione, il SensorType, l'ultima Misurazione, la Regione in cui è inserito, le Coordinate x,y,z in cui è posizionato all'interno di tale regione e alcune Caratteristiche. Fra quesi dati la maggior parte sono decodificabili dagli eventi che arrivano dai device, mentre la Regione e le Coordinate verranno inserite dall'utente dall'interfaccia Web;
- *schemas.py* dove vengono descritte le classi derivanti dalla classe base Device;
- *router_serial.py* è il fulcro dell'intero programma. Esso infatti:
  - Si connette alla seriale;
  - Riceve gli eventi dai device tramite stringhe che spacchetta e ne ricava i dati da inserire in ogni istanza della classe Device da aggiungere al database;

  - Crea un'app come istanza di FastAPI in cui è possibile eseguire delle operazioni (metodi http: Get, Post, Put, Delete) da vari percorsi;
  - È in grado di rilevare lo status dei device tramite un task asincrono che controlla il tempo passato dall'ultima misurazione;
  - Esporta tutti i comandi necessari del gateway via http: quando riceve la richiesta, la gira sul Gateway seriale codificandola in modo opportuno, aspetta la risposta, la codifica e la invia come risposta alla richiesta http;
  - pubblica tutte le misure ricevute su MQTT in topic dal nome /nomegw/*/misura;
  - pubblica tutte le informazioni di stato su MQTT in topic dal nome /nomegw/*/admin.

I file principali che implementano la parte di frontend sono:

- *GetDevices.js* Si collega all'url base dell'app FastAPI e fa la fetch. Fa il render di come devono essere mostrati i device all'interno dell'interfaccia e permette, premendo il bottone BLINK, di collegarsi all'url /Device/id e di scrivere su seriale il comando di accensione del led.
- *PostDevices.js* Permette di inserire delle informazioni del device sul db scrivendole da interfaccia Web, è utile nell'inserimento di Regione e Coordinate;
- *useFetch.js* e *useForm.js* Sono funzioni che utilizzano gli Hooks di React;
- *App.js* Implementa il render finale della schermata importando gli altri file.

L'interfaccia finora creata è stata inserita in figura 6.25.

6.2.3 Configurazione device

Attraverso tale app possono essere comunicati dei dati relativi alla configurazione dei device. L'idea è quella di poter settare l'intervallo di pubblicazione dei sensori attraverso il proprio id o gruppo di appartenenza Questa funzionalità non è però ancora stata implementata. La possibilità di inserire un nodo in un determinato gruppo avviene dall'appSTM. In questo momento è possibile configurare i nodi inserendo per ciascuno di essi le coordinate geografiche da interfaccia Web. Questo rende la rete più facilmente gestibile e manutenibile.

6.2.4 MQTT

MQTT è un protocollo di messaggistica publish/subscribe che permette di pubblicare messaggi su un broker. Chiunque ha fatto subscribe al topic di quel determinato broker riceve in automatico le informazioni. I topic MQTT sono una

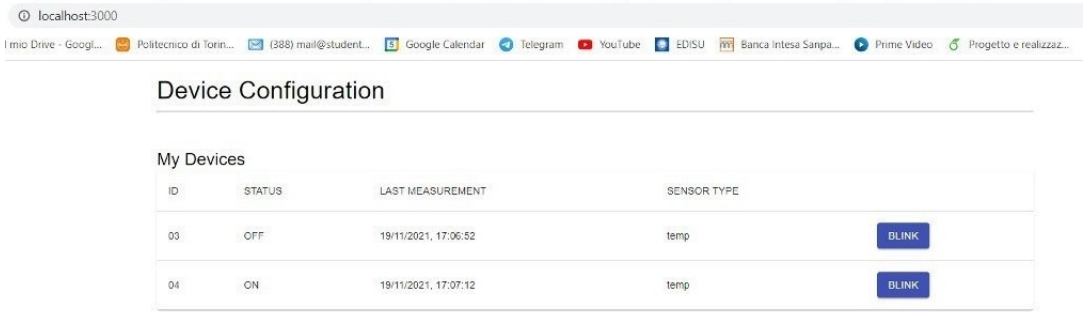


Figura 6.25: Interfaccia Web

forma di indirizzamento che consente ai client MQTT di condividere informazioni. I topic su cui sono pubblicate le misurazioni sono:

- *nomegw/nodoId/misure*,
- *nomegw/groupId/misure*,
- *nomegw/all/misure*

e quelli su cui vengono inviate le informazioni di stato sono:

- *nomegw/nodoId/admin*,
- *nomegw/groupId/admin*,
- *nomegw/all/admin*.

Grazie alla struttura gerarchica dei topic è possibile iscrivere un client a tutte le informazioni regative al Gateway con *nomegw/#*, a quelle relative a solo un nodo con *nomegw/nodoId/#*, a quelle relative ad un gruppo con *nomegw/groupId/#* e tutte le info riguardanti le misure con *nomegw/+/misure* e le info di stato con *nomegw/+/admin*.
Il broker che è stato utilizzato per tale progetto è Mosquitto. Esso viene avviato da linea di comando da Linux con
mosquitto -v

```
broker = 'localhost'
port = 1883
username = 'sofia'
password = 'mqtt'
```

Figura 6.26: Credenziali MQTT

e di default parte sulla porta 1883. Per rendere Mosquitto più sicuro è stato aggiunto un determinato utente con una specifica password digitando il comando *sudo mosquitto_passwd -c /etc/mosquitto/passwd sofia*
e successivamente inserendo due volte la password.
Con le credenziali di figura 6.26 è adesso possibile inviare e ricevere in modo sicuro le informazioni sul e dal broker, grazie alla libreria paho.mqtt.

6.2.5 Comunicazione con iChain

La comunicazione con iChain avviene grazie ad un'app Consumer. Tale applicazione è necessaria in quanto i dati finora ricevuti non costituiscono degli eventi EPCIS e quindi devono essere prima codificati in modo corretto includendo le informazioni riguardo a what, where, when e why.
Questa applicazione non è stata ancora implementata, ma è stata ideata in modo tale da avere tali caratteristiche:

- permette la cattura di tutti i dati dei sensori da MQTT;
- permette di configurarsi alla piattaforma iChain nel modo corretto: utilizzando conf con indirizzo e credenziali;
- permette di inviare due tipologie di dato su iChain:
  - dati raw dei singoli valori con delle POST http;
  - eventi EPCIS con alcuni dati aggregati relativi al prodotto.

In sostanza quindi vengono inviati ogni singolo valore al server su un endpoint specifico http e dati riassuntivi legati alla produzione su un secondo endpoint.

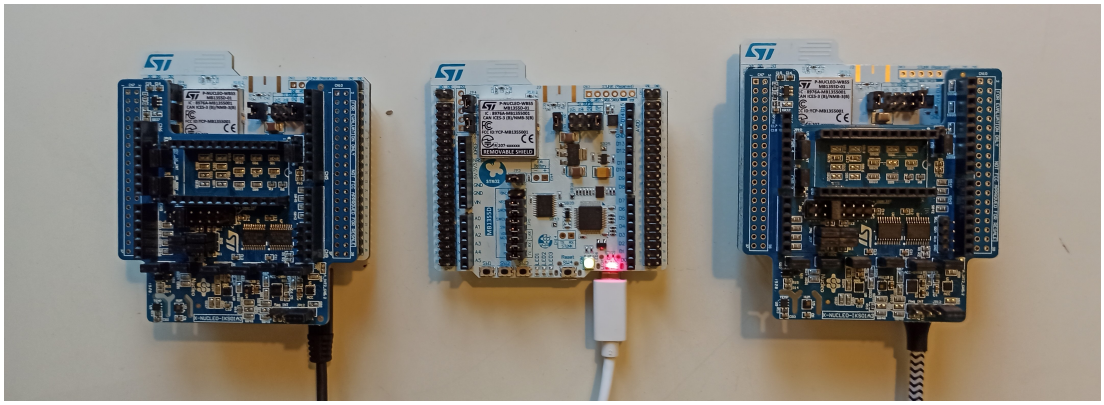
Capitolo 7

Conclusioni

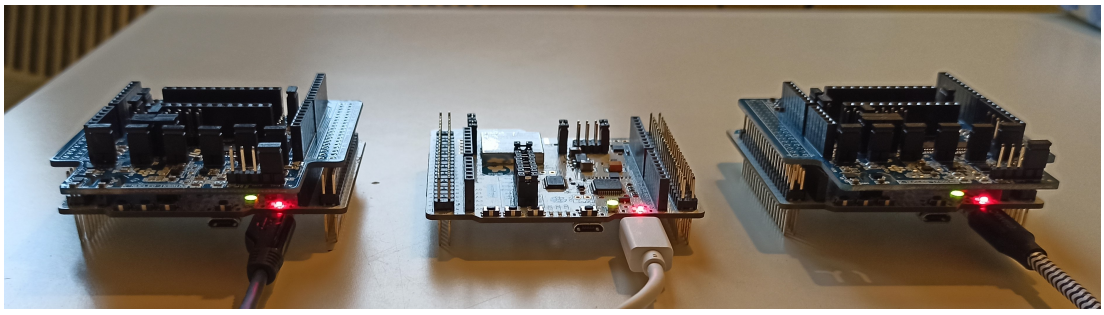
7.1 Risultati ed analisi

In figura 7.1 sono state inserite le foto della rete implementata. Poichè erano state acquistate tre Nucleo board della STM e due shield IKS01A3 la rete implementata finora è formata solamente da tre nodi.
Il device al centro della figura, quello con cavo bianco, è il Gateway. Esso non ha bisogno della MEMS shield in quanto ha la funzione di ricezione dei messaggi da parte degli altri nodi della rete che fungono da sensori. Esso è correttamente collegato in modo seriale ad un PC e tramite l'app seriale implementata ha la possibilità di inviare i dati ricevuti via MQTT ad un'app consumer che permette di salvarli localmente e li codifica in modo corretto e li inserisce su iChain.
Gli altri due dispositivi disposti ai lati del Gateway sono i nodi Sensori. Essi montano la shield che è facilmente integrata "a panino" con la Nucleo Board. Essi sono ad ora collegati via USB per essere alimentati. In futuro dovranno però essere alimentati a batteria, se si tratta dei nodi Low Power (nodi foglia della rete) o collegati a presa elettrica (nel caso abbiano funzione di nodi dorsale, nodi Friend).
La rete potrebbe essere facilmente ampliata inserendo altri device Sensori con apposita shield. Su di essi andrebbe caricato il codice relativo ai Sensori, descritto nel capitolo precedente in modo da far loro pubblicare i dati rilevati. Il Gateway, che effettua la subscription a tutti i nodi presenti nella rete, riuscirebbe a ricevere tali dati senza ulteriori interventi a livello di codice. Su di essi è possibile inviare un comando di accensione di un led che accende il led cerchiato in blu in figura 7.2.
La rete così implementata è facilmente gestibile e manutenibile.
Dall'interfaccia web di configurazione dei device è possibile visualizzare la lista dei device all'interno della rete, il proprio ID, il proprio status (ON o OFF) e l'ora e la data dell'ultima misurazione. Ogni device della lista è fisicamente inserito

all'interno di una posizione specifica del luogo d'interesse e essa può essere salvata per rendere la rete più facilmente manutenibile. Infatti un device erroneamente con status OFF può essere controllato andando ad agire fisicamente alle coordinate della sua posizione. Per evitare che vengano inserite posizioni su device sbagliati è possibile distingurere il device su cui si sta inserendo la posizione grazie all'accensione di un led.



(a) Vista dall'alto



(b) Vista laterale

Figura 7.1: Mesh implementata

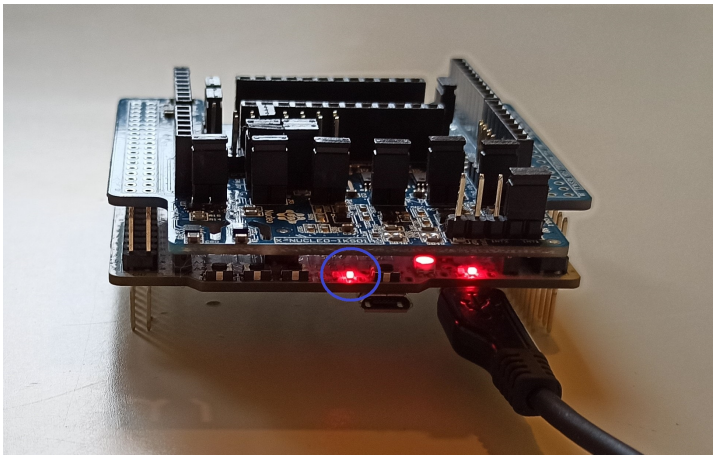


Figura 7.2: Sensore con led acceso

7.2 Lavori futuri

Il sistema IoT implementato è correttamente funzionante. Le funzionalità e le caratteristiche sono state ampliamente descritte nei capitoli precedenti.
In tale capitolo vengono descritte le migliorie e le aggiunte che potrebbero essere effettuate in futuro.
Un'importante caratteristica della Mesh BLE è la possibilità di introdurre nodi Low Power a ridotto consumo energetico. All'interno della rete quindi i nodi verranno suddivisi in due tipologie, nodi Low Power e nodi Friend. I nodi Sensori finora implementati dovranno avere le caratteristiche di nodi Low Power, quindi essere tenuti spenti per la maggior parte del ciclo di vita ed essere accesi solo nel momento di pubblicazione del dato. Questo permetterebbe a tali device di essere alimentati a batteria e in tal modo di avere ciclo di vita di mesi. Chi si occupa di accenderli sarebbero i nodi Friend, nodi inseriti nella rete anche con la funzionalità di estendere il range di comunicazione. Tali nodi dovranno stare sempre accesi e quindi verranno alimentati con la corrente elettrica. Nell'interfaccia Web i nodi visualizzati saranno solamente i Sensori.
Finora i Sensori utilizzano la shield IKS01A3 per pubblicare solamente il dato relativo alla temperatura. La pubblicazione verrà estesa anche a informazioni riguardo l'umidità dell'aria e la pressione atmosferica, acquisizioni che la shield può già effettuare. Inoltre alcuni Sensori verranno integrati con la shield IKA02A1 per il controllo della qualità dell'aria che rileverà la concentrazione di CO e pubblicherà tale dato nella mesh. Dall'interfaccia sarà possibile suddividere i nodi in base al tipo di dato che rilevano, "temp", "hum", "press" e "co".

I device verranno suddivisi in gruppi, così che il Gateway sarà in grado di effettuare la subscription ad uno o più gruppi. Un gruppo potrebbe essere formato dai device che rilevano la stessa tipologia di dato o i device all'interno della stessa zona geografica. Dall'interfaccia sarà possibile selezionare solo quelli con le caratteristiche desiderate e vederne le informazioni.
La chiave di rete sarà una chiave esterna generata ex novo in modo tale da rendere la rete più sicura.
L'interfaccia verrà migliorata e adattata alle nuove funzionalità implementate dal sistema.
Verrà creata una pagina Admin che permetterà di rilevare in modo più efficiente il Gateway connesso in modo seriale. Non ci saranno quindi più problemi in caso di errata disconnessione di esso.
Sarà inoltre implementata un'app Consumer in modo da catturare tutti i dati dei sensori da MQTT, quelli relativi alle informazioni ambientali e allo status dei device. Sarà configurata per inviare i dati verso iChain: l'invio consiste nell'invio di dati raw (con delle POST http) e nell'invio di eventi EPCIS con alcuni dati aggregati relativi al prodotto.

Il sistema IoT implementato è frutto di un'accurata ricerca delle tecniche e degli strumenti hardware e software più congeniali. Il risultato è una rete funzionante, le cui informazioni rilevate sono esaminabili e configurabili mediante una semplice interfaccia Web. Tutte le funzionalità descritte in tale capitolo sono da considerarsi delle migliorie effettuabili su un sistema già valido.

Capitolo 8

Appendice

Nell'Appendice sono stati inseriti i Datasheet delle componenti elettroniche utilizzate. Essi possono essere consultati anche dal documento [16].

life.augmented

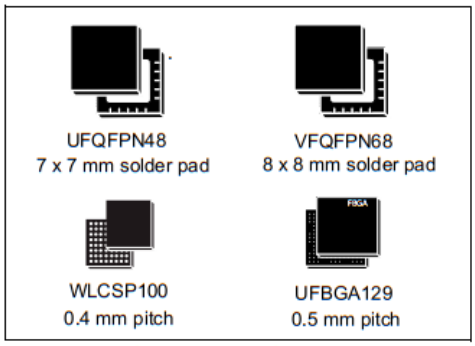
STM32WB55xx
STM32WB35xx

Multiprotocol wireless 32-bit MCU Arm®-based Cortex®-M4 with FPU, Bluetooth® 5.2 and 802.15.4 radio solution

Datasheet - production data

Features

- Include ST state-of-the-art patented technology
- Radio
  - 2.4 GHz
  - RF transceiver supporting Bluetooth® 5.2 specification, IEEE 802.15.4-2011 PHY and MAC, supporting Thread and Zigbee® 3.0
  - RX sensitivity: -96 dBm (Bluetooth® Low Energy at 1 Mbps), -100 dBm (802.15.4)
  - Programmable output power up to +6 dBm with 1 dB steps
  - Integrated balun to reduce BOM
  - Support for 2 Mbps
  - Dedicated Arm® 32-bit Cortex® M0+ CPU for real-time Radio layer
  - Accurate RSSI to enable power control
  - Suitable for systems requiring compliance with radio frequency regulations ETSI EN 300 328, EN 300 440, FCC CFR47 Part 15 and ARIB STD-T66
  - Support for external PA
  - Available integrated passive device (IPD) companion chip for optimized matching solution (MLPF-WB-01E3 or MLPF-WB-02E3)
- Ultra-low-power platform
  - 1.71 to 3.6 V power supply
  - – 40 °C to 85 / 105 °C temperature ranges
  - 13 nA shutdown mode
  - 600 nA Standby mode + RTC + 32 KB RAM
  - 2.1 µA Stop mode + RTC + 256 KB RAM
  - Active-mode MCU: < 53 µA / MHz when RF and SMPS on
  - Radio: Rx 4.5 mA / Tx at 0 dBm 5.2 mA

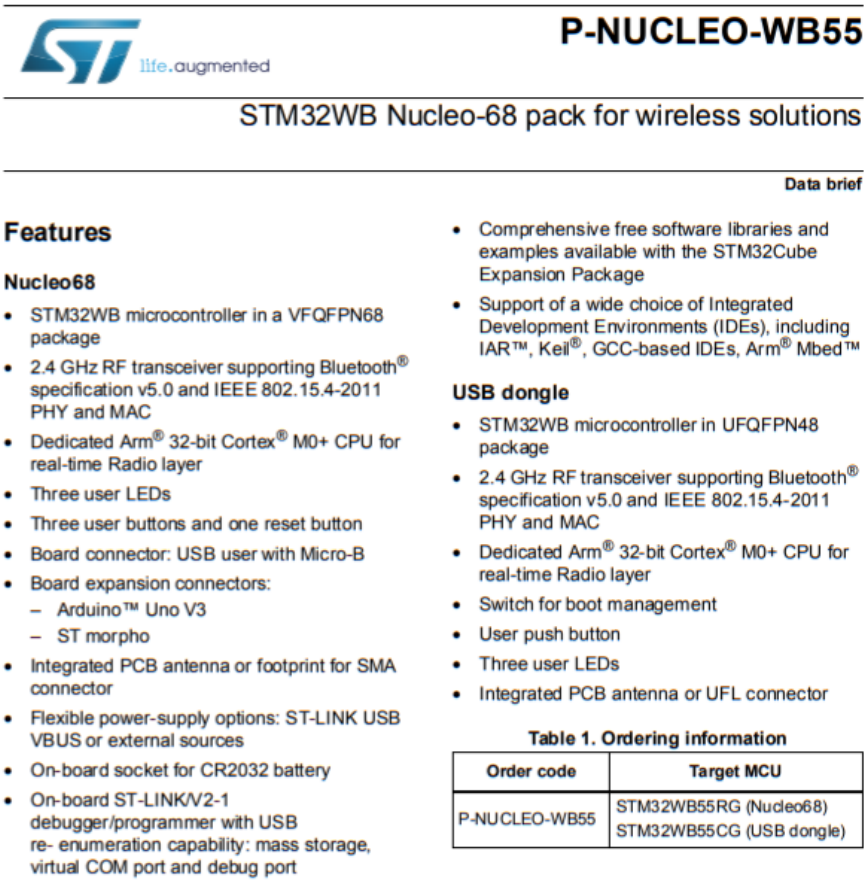


- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, adaptive real-time accelerator (ART Accelerator) allowing 0-wait-state execution from Flash memory, frequency up to 64 MHz, MPU, 80 DMIPS and DSP instructions
- Performance benchmark
  - 1.25 DMIPS/MHz (Drystone 2.1)
  - 219.48 CoreMark® (3.43 CoreMark/MHz at 64 MHz)
- Energy benckmark
  - 303 ULPMark™ CP score
- Supply and reset management
  - High efficiency embedded SMPS step-down converter with intelligent bypass mode
  - Ultra-safe, low-power BOR (brownout reset) with five selectable thresholds
  - Ultra-low-power POR/PDR
  - Programmable voltage detector (PVD)
  - V_{BAT} mode with RTC and backup registers
- Clock sources
  - 32 MHz crystal oscillator with integrated trimming capacitors (Radio and CPU clock)
  - 32 kHz crystal oscillator for RTC (LSE)
  - Internal low-power 32 kHz (±5%) RC (LSI1)
  - Internal low-power 32 kHz (stability ±500 ppm) RC (LSI2)

Figura 8.1: Datasheet NUCLEO-WB55RG

  - Internal multispeed 100 kHz to 48 MHz oscillator, auto-trimmed by LSE (better than ±0.25% accuracy)
  - High speed internal 16 MHz factory trimmed RC (±1%)
  - 2x PLL for system clock, USB, SAI and ADC
- Memories
  - Up to 1 MB Flash memory with sector protection (PCROP) against R/W operations, enabling radio stack and application
  - Up to 256 KB SRAM, including 64 KB with hardware parity check
  - 20x32-bit backup register
  - Boot loader supporting USART, SPI, I2C and USB interfaces
  - OTA (over the air) Bluetooth® Low Energy and 802.15.4 update
  - Quad SPI memory interface with XIP
  - 1 Kbyte (128 double words) OTP
- Rich analog peripherals (down to 1.62 V)
  - 12-bit ADC 4.26 Msps, up to 16-bit with hardware oversampling, 200 µA/Msps
  - 2x ultra-low-power comparator
  - Accurate 2.5 V or 2.048 V reference voltage buffered output
- System peripherals
  - Inter processor communication controller (IPCC) for communication with Bluetooth® Low Energy and 802.15.4
  - HW semaphores for resources sharing between CPUs
  - 2x DMA controllers (7x channels each) supporting ADC, SPI, I2C, USART, QSPI, SAI, AES, timers
  - 1x USART (ISO 7816, IrDA, SPI Master, Modbus and Smartcard mode)
  - 1x LPUART (low power)
  - 2x SPI 32 Mbit/s
  - 2x I2C (SMBus/PMBus)
  - 1x SAI (dual channel high quality audio)
  - 1x USB 2.0 FS device, crystal-less, BCD and LPM
  - Touch sensing controller, up to 18 sensors
  - LCD 8x40 with step-up converter
  - 1x 16-bit, four channels advanced timer
  - 2x 16-bit, two channels timer
  - 1x 32-bit, four channels timer
  - 2x 16-bit ultra-low-power timer
  - 1x independent Systick
  - 1x independent watchdog
  - 1x window watchdog
- Security and ID
  - Secure firmware installation (SFI) for Bluetooth® Low Energy and 802.15.4 SW stack
  - 3x hardware encryption AES maximum 256-bit for the application, the Bluetooth® Low Energy and IEEE802.15.4
  - Customer key storage / key manager services
  - HW public key authority (PKA)
  - Cryptographic algorithms: RSA, Diffie-Helman, ECC over GF(p)
  - True random number generator (RNG)
  - Sector protection against R/W operation (PCROP)
  - CRC calculation unit
  - Die information: 96-bit unique ID
  - IEEE 64-bit unique ID. Possibility to derive 802.15.4 64-bit and Bluetooth® Low Energy 48-bit EUI
- Up to 72 fast I/Os, 70 of them 5 V-tolerant
- Development support
  - Serial wire debug (SWD), JTAG for the application processor
  - Application cross trigger with input / output
  - Embedded Trace Macrocell™ for application
- All packages are ECOPACK2 compliant

Figura 8.2: Datasheet NUCLEO-WB55RG



life.augmented

P-NUCLEO-WB55

STM32WB Nucleo-68 pack for wireless solutions

Data brief

Features

Nucleo68

- STM32WB microcontroller in a VFQFPN68 package
- 2.4 GHz RF transceiver supporting Bluetooth® specification v5.0 and IEEE 802.15.4-2011 PHY and MAC
- Dedicated Arm® 32-bit Cortex® M0+ CPU for real-time Radio layer
- Three user LEDs
- Three user buttons and one reset button
- Board connector: USB user with Micro-B
- Board expansion connectors:
  - Arduino™ Uno V3
  - ST morpho
- Integrated PCB antenna or footprint for SMA connector
- Flexible power-supply options: ST-LINK USB VBUS or external sources
- On-board socket for CR2032 battery
- On-board ST-LINK/V2-1 debugger/programmer with USB re- enumeration capability: mass storage, virtual COM port and debug port
- Comprehensive free software libraries and examples available with the STM32Cube Expansion Package
- Support of a wide choice of Integrated Development Environments (IDEs), including IAR™, Keil®, GCC-based IDEs, Arm® Mbed™

USB dongle

- STM32WB microcontroller in UFQFPN48 package
- 2.4 GHz RF transceiver supporting Bluetooth® specification v5.0 and IEEE 802.15.4-2011 PHY and MAC
- Dedicated Arm® 32-bit Cortex® M0+ CPU for real-time Radio layer
- Switch for boot management
- User push button
- Three user LEDs
- Integrated PCB antenna or UFL connector

Table 1. Ordering information

Order code	Target MCU
P-NUCLEO-WB55	STM32WB55RG (Nucleo68) STM32WB55CG (USB dongle)

Figura 8.3: Datasheet P-NUCLEO-WB55

Bibliografia

[1] https://www.gs1.org/sites/default/files/docs/epc/EPCIS-Standard-1.2-r-2016-09-29.pdf

[2] https://www.youtube.com/playlist?list=PL3EThPF3sbJr6TyUOlnHch2lOGyiF2R_9

[3] https://xeliontech.com/it/

[4] https://www.ictpower.it/tecnologia/lora-nozioni-di-base-e-approfondimenti.htm

[5] https://www.sigfox.com/en

[6] https://edalab.it/zigbee-comunicazione-wireless/

[7] https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[8] https://www.novelbits.io/bluetooth-mesh-tutorial-part-1/

[9] STMicroelectronics. (2020). How to build a Bluetooth Low Energy mesh application for STM32WBx5 line microcontrollers.

[10] https://www.cypress.com/html_doc/AppNotes/AN227069/index.html#ble-mesh-operation-and-key-concepts

[11] https://punchthrough.com/how-gap-and-gatt-work/

[12] https://it.mathworks.com/help/comm/ug/bluetooth-protocol-stack.html

[13] https://www.bluetooth.com/blog/provisioning-a-bluetooth-mesh-network-part-1/

[14] Martin Woolley. (2019). Bluetooth Mesh Models. Technical Overview.

[15] https://www.st.com/en/evaluation-tools/nucleo-wb55rg.html

[16] STMicroelectronics. (2017). STM32WB55xx.

[17] STMicroelectronics. (2019). X-NUCLEO-IKS01A3.

[18] https://www.st.com/en/evaluation-tools/p-nucleo-wb55.html

[19] https://www.st.com/en/ecosystems/x-nucleo-iks01a3.html

[20] STMicroelectronics. (2017). Getting started with the P-NUCLEO-IKA02A1 STM32 Nucleo pack for electrochemical toxic gas sensor expansion board with CO sensor.

[21] https://www.st.com/en/ecosystems/x-nucleo-iks01a3.html

[22] https://www.st.com/en/ecosystems/p-nucleo-ika02a1.html

[23] https://docs.zephyrproject.org/latest/getting_started/index.html

[24] https://www.st.com/content/st_com/en/ecosystems/stm32cube-ecosystem.html

[25] https://os.mbed.com/mbed-os/

[26] https://blog.davidbramsay.com/getting-started-with-stm32wb-and-ble-communications/

[27] https://www.youtube.com/watch?v=tRn08tdRPUU

[28] STMicroelectronics. (2019). ST firmware upgrade services for STM32WB Series.

[29] STMicroelectronics. (2018). Getting started with the STMicroelectronics X-CUBE-MEMS1 software package for STM32CubeMX.

[30] `http://www.steves-internet-guide.com/mosquitto-broker/`

[31] `https://mqtt.org/`

[32] `http://www.steves-internet-guide.com/understanding-mqtt-topics/`

[33] `https://fastapi.tiangolo.com/tutorial/first-steps/`