



**UNIVERSITÀ DEGLI STUDI DI CATANIA**  
DIPARTIMENTO DI MATEMATICA E INFORMATICA  
CORSO DI LAUREA TRIENNALE IN INFORMATICA

---

*Catalano Sophia Grazia*

**Reti Wireless Basate su Tecnologia LoRa**

Una analisi del funzionamento e dei protocolli di sicurezza con validazione tramite prototipo su rete Meshtastic.

---

RELAZIONE PROGETTO FINALE

---

Relatore: Di Raimondo Mario

---

Anno Accademico 2024 - 2025

# Abstract

La tecnologia LoRa rappresenta attualmente una delle soluzioni più comuni per le comunicazioni wireless su lunghe distanze e a basso consumo, utilizzata in diversi ambiti applicativi, dai sistemi di monitoraggio ambientale alle comunicazioni off-grid.

La presente tesi propone un'analisi comparativa di tre diverse implementazioni basate su LoRa: LoRaWAN, Meshtastic e MeshCore. Attraverso lo studio della letteratura scientifica e della documentazione tecnica disponibile, vengono esaminate le architetture di riferimento e i protocolli di sicurezza adottati. L'analisi teorica è affiancata dalla realizzazione di un piccolo sistema di monitoraggio ambientale basato su nodi Meshtastic e da un'applicazione mobile dedicata alla visualizzazione della telemetria e della posizione dei dispositivi.

L'integrazione tra studio e prototipazione consente di mettere in evidenza differenze, vantaggi e limitazioni delle tecnologie considerate, fornendo indicazioni utili per la scelta della tecnologia più adatta ai diversi scenari applicativi.

# Indice

<b>1</b>	<b>Introduzione</b>	<b>4</b>
<b>2</b>	<b>Tecnologia LoRa</b>	<b>6</b>
2.1	Reti LPWAN . . . . .	6
2.2	Principi fisici e modulazione LoRa . . . . .	6
2.2.1	Parametri di modulazione LoRa . . . . .	7
2.3	Hardware e dispositivi . . . . .	9
2.4	Confronto tra tecnologie LPWAN . . . . .	10
2.4.1	Sigfox . . . . .	11
2.4.2	NB-IoT . . . . .	11
<b>3</b>	<b>LoRaWAN</b>	<b>13</b>
3.1	LoRa vs LoRaWAN . . . . .	13
3.2	Architettura di rete . . . . .	13
3.2.1	Implementazioni pratiche . . . . .	15
3.2.2	Classi di dispositivi . . . . .	15
3.2.3	Gateway LoRaWAN . . . . .	16
3.3	Livello MAC . . . . .	17
3.3.1	Formato dei frame MAC . . . . .	18
3.4	Attivazione dei dispositivi . . . . .	20
3.4.1	Attivazione Over-The-Air (OTAA) . . . . .	20
3.4.2	Attivazione tramite personalizzazione (ABP) . . . . .	22
3.5	Sicurezza . . . . .	22
3.5.1	Advanced Encryption Standard . . . . .	22
3.5.2	Crittografia e Chiavi di Sicurezza . . . . .	23
3.5.3	Generazione delle Chiavi di Sessione . . . . .	24
3.5.4	Proprietà di Sicurezza . . . . .	24
3.5.5	Contatore di frame . . . . .	26

<i>INDICE</i>	3
<b>4 Meshtastic</b>	<b>27</b>
4.1 Reti mesh a lungo raggio . . . . .	27
4.2 Architettura di rete . . . . .	27
4.2.1 Configurazione dei canali . . . . .	28
4.2.2 Classificazione dei nodi . . . . .	29
4.3 Protocollo di comunicazione . . . . .	30
4.3.1 Messaggi diretti . . . . .	33
4.4 Sicurezza . . . . .	34
4.4.1 Crittografia nei canali di comunicazione . . . . .	34
4.4.2 Messaggi diretti e crittografia asimmetrica . . . . .	35
4.4.3 Limitazioni del modello di sicurezza . . . . .	36
4.4.4 Soluzioni suggerite . . . . .	37
<b>5 MeshCore</b>	<b>39</b>
5.1 Panoramica generale . . . . .	39
5.2 Classificazione dei nodi . . . . .	39
5.3 Funzionamento della rete . . . . .	40
5.4 Sicurezza . . . . .	41
5.4.1 Identificazione degli algoritmi crittografici . . . . .	42
5.4.2 Proprietà di sicurezza . . . . .	42
<b>6 Implementazione e test</b>	<b>44</b>
6.1 Introduzione . . . . .	44
6.2 Architettura hardware . . . . .	44
6.3 Configurazione del firmware . . . . .	45
6.4 MeshView . . . . .	47
6.5 Test e risultati . . . . .	50
<b>Conclusione</b>	<b>56</b>
<b>Bibliografia</b>	<b>58</b>

# Capitolo 1

## Introduzione

La crescente dipendenza dalle infrastrutture di telecomunicazione ha evidenziato la necessità di sviluppare sistemi di comunicazione autonomi e resilienti, particolarmente in contesti dove la connettività tradizionale è assente o difficilmente accessibile.

In risposta a questa esigenza, le tecnologie LPWAN (Low Power Wide Area Network) emergono come una soluzione particolarmente adatta. Tra queste, LoRa (Long Range) si distingue come una delle alternative più valide grazie alla sua capacità di offrire comunicazioni a lunga distanza con un ridotto consumo energetico e costi contenuti.

Diverse implementazioni e protocolli sono stati sviluppati basandosi su LoRa, gestendo la comunicazione in modi differenti. Tuttavia, queste soluzioni devono far fronte a diverse sfide significative. In primo luogo, la sicurezza e l'affidabilità delle comunicazioni rappresentano aspetti cruciali, poiché i dispositivi collocati in ambienti aperti possono essere soggetti a manomissioni e guasti, mentre le trasmissioni wireless sono vulnerabili a interferenze e perdite di pacchetti. Inoltre, l'efficienza energetica costituisce una questione fondamentale, in quanto molti nodi sono alimentati a batteria e devono operare per lunghi periodi senza manutenzione.

Il fine di questa tesi è presentare un'analisi comparativa delle principali soluzioni che utilizzano LoRa, mettendo in luce i loro punti di forza e le loro limitazioni, con l'obiettivo di offrire linee guida per la scelta della soluzione più adeguata in base ai requisiti operativi. In particolare, saranno analizzate tre diverse tecnologie:

- **LoRaWAN**, uno standard aperto, in quanto le specifiche riguardanti il protocollo di comunicazione sono pubblicamente disponibili e consultabili senza restrizioni o costi di licenza. È progettato per reti a stella con gestione centralizzata dei nodi tramite gateway e server di rete, ideale

per scenari di larga scala. Sono disponibili anche implementazioni open source del server LoRaWAN, tra cui ChirpStack e The Things Stack.

- **Meshtastic**, progetto open source per reti mesh decentralizzate, in cui i nodi possono instradare i messaggi autonomamente, indicato per comunicazioni off-grid. Non fa riferimento ad alcuno standard ufficiale e utilizza un protocollo definito e mantenuto dalla comunità.
- **MeshCore**, progetto open source anch'esso orientato alla realizzazione di reti mesh decentralizzate. Come Meshtastic, non è legato a uno standard formale e implementa un protocollo indipendente sviluppato dalla comunità.

Oltre all'analisi teorica, la tesi include una parte progettuale dedicata alla realizzazione di un sistema di monitoraggio ambientale e allo sviluppo di un'applicazione mobile che permette di visualizzare in tempo reale la posizione dei nodi e la telemetria raccolta. Questa implementazione pratica consente di valutare il comportamento delle reti mesh LoRa e di confrontare i risultati ottenuti con le caratteristiche descritte nei capitoli precedenti.

Il lavoro si compone di cinque capitoli, oltre all'introduzione. I primi capitoli trattano la tecnologia LoRa e le tre implementazioni considerate, mentre l'ultimo è dedicato al progetto realizzato. Le principali differenze tra le varie implementazioni emergono nei capitoli dedicati e sono oggetto di una sintesi comparativa nelle conclusioni.

# Capitolo 2

## Tecnologia LoRa

### 2.1 Reti LPWAN

Le *Low-Power Wide-Area Networks* (LPWAN) sono un insieme di tecnologie di rete concepite per la trasmissione di pacchetti di dati di piccole dimensioni a basse velocità, coprendo distanze considerevoli e con un consumo energetico nettamente inferiore rispetto a tecnologie convenzionali come Wi-Fi o Bluetooth[1]. Queste tecnologie sono particolarmente adatte per applicazioni IoT (Internet of Things) che richiedono comunicazioni sporadiche e a bassa latenza, come il monitoraggio ambientale, la gestione di asset industriali, l'agricoltura di precisione e le smart city. Generalmente, le LPWAN adottano una topologia a stella, in cui ciascun nodo comunica direttamente con una stazione base.

### 2.2 Principi fisici e modulazione LoRa

Tra le diverse tecnologie LPWAN in rapida evoluzione sul mercato, si distingue **LoRa** (Long Range). LoRa è un'evoluta tecnica di modulazione, che trasforma i dati in segnali radio, basata sulla modulazione *Chirp Spread Spectrum* (CSS). Il CSS è una tecnologia wireless a banda larga che produce segnali denominati "chirp", il cui significato è un segnale la cui frequenza varia linearmente nel tempo, in modo crescente o decrescente [2].

In pratica, il segnale non viene trasmesso su una singola frequenza, ma viene distribuito o espanso su una banda di frequenza più ampia. Questo processo di espansione dell'energia del segnale conferisce alla comunicazione una notevole robustezza contro rumore e interferenze, migliorando significativamente la resilienza del segnale in ambienti difficili.

### 2.2.1 Parametri di modulazione LoRa

La modulazione LoRa si basa su diversi parametri configurabili, i quali influenzano prestazioni, portata e consumo energetico[3]:

**Coding Rate (CR)** Il *livello di correzione d'errore*, implementato attraverso la Forward Error Correction (FEC) può essere impostato su valori compresi tra 4/5 e 4/8. Maggiore è il CR, più alta è la protezione contro gli errori, a discapito del tempo di trasmissione.

**Potenza di trasmissione (PT)** La *potenza di trasmissione* tipicamente regolabile da -4 dBm a +20 dBm, ha un impatto diretto sulla portata e affidabilità del collegamento. Tuttavia, deve sempre rispettare i limiti normativi imposti dalle autorità regionali per l'utilizzo delle bande ISM.

**Frequenza portante (CF)** La *frequenza portante* determina il canale radio utilizzato per la trasmissione e ha un impatto diretto sulla copertura del segnale. LoRa può operare su diverse bande di frequenza, offrendo grande flessibilità in termini di copertura, velocità e consumo energetico. In particolare, utilizza le bande sub-gigahertz senza licenza, come i 868 MHz in Europa, i 915 MHz negli Stati Uniti e i 433 MHz in alcuni paesi asiatici. Può anche essere utilizzata sulla banda a 2.4 GHz per raggiungere velocità di trasmissione dati più elevate rispetto alle bande sub-gigahertz, a scapito della portata. Queste frequenze rientrano nelle bande ISM, riservate a livello internazionale a scopi industriali, scientifici e medici [4].

**Bandwidth (BW)** La *larghezza di banda* rappresenta la porzione di spettro utilizzata dal segnale LoRa intorno alla frequenza portante. I valori tipici sono 125 kHz, 250 kHz e 500 kHz. Una bandwidth maggiore consente velocità di trasmissione più elevate, ma comporta un maggiore consumo energetico e una minore sensibilità del ricevitore.

**Spreading Factors** I *Fattori di Diffusione* (Spreading Factors, SF) sono parametri di fondamentale importanza, in quanto regolano l'ampiezza del segnale di estensione. Essi rappresentano il quantitativo di codice di diffusione applicato sul segnale dati originale. LoRa stabilisce sei Fattori di Diffusione: da SF7 a SF12.

Fattori bassi (es. SF7) garantiscono velocità elevate e tempi di trasmissione minore (Time-on-Air). Prevedono un rapporto Segnale/Rumore (SNR) più alto e sono ideali per comunicazioni a breve distanza. Fattori alti (es. SF12) migliorano il cosiddetto "guadagno di elaborazione", ovvero un miglioramento del SNR percepito dal ricevitore, aumentando la portata del segnale. Ciò comporta però una minore velocità dati e un tempo di trasmissione più elevato, con un conseguente aumento del consumo energetico per trasmissione. Questi fattori sono adatti a sensori situati a grande distanza dal gateway[5].

Un aspetto peculiare è l'*ortogonalità* degli SF: dispositivi che trasmettono contemporaneamente sulla stessa frequenza ma con SF diversi non interferiscono tra loro, rendendo la rete scalabile.

La velocità dati effettiva dipende dalla combinazione di **bandwidth** e **spreading factor** secondo la formula:

$$\text{Bit Rate} = SF \times \left( \frac{BW}{2^{SF}} \right) \times CR \quad (2.1)$$

Per riassumere e confrontare in modo diretto le caratteristiche di diversi Fattori di Diffusione, nella Tabella 2.1 sono sintetizzate le informazioni relative a velocità dati, portata, durata di trasmissione e consumo energetico stimato.

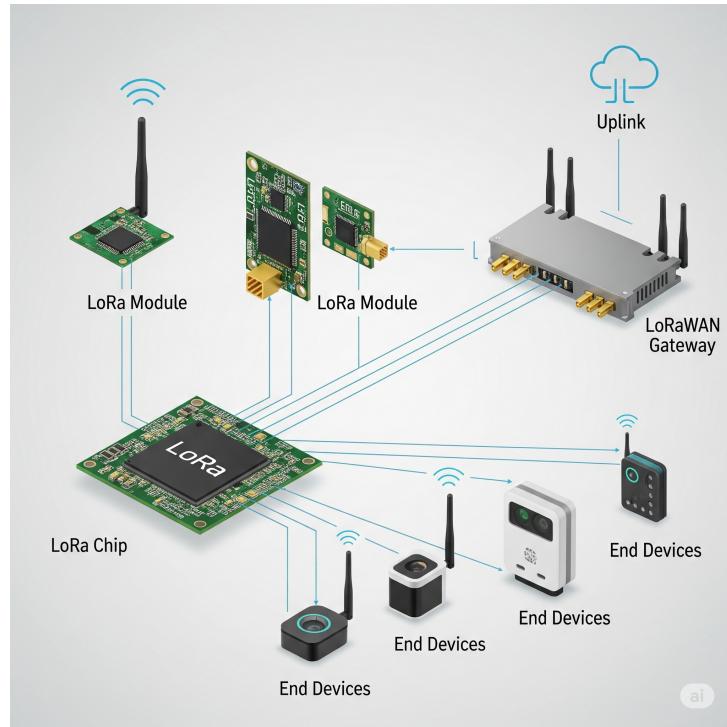
Spreading Factor (UL at 125kHz)	Bit Rate	Range (terrain dependent)	Time on Air (11-byte payload)
SF10	980 bps	8 km	371 ms
SF9	1760 bps	6 km	185 ms
SF8	3125 bps	4 km	103 ms
SF7	5470 bps	2 km	61 ms

**Tabella 2.1:** Parametri LoRa per diversi Spreading Factor

Come si può osservare, aumentando il Fattore di Diffusione si ottiene una maggiore portata, ma si riduce la velocità di trasmissione e aumenta il consumo energetico a causa del tempo di trasmissione più lungo. La scelta del SF ottimale dipende quindi dall'applicazione specifica, trovando un equilibrio tra copertura, durata della batteria e velocità richiesta.

## 2.3 Hardware e dispositivi

La tecnologia LoRa si basa su un ecosistema hardware composto principalmente da *chip radio*, *moduli radio*, *microcontrollori* e *sensori*, che operano all'unisono per abilitare reti IoT distribuite, efficienti e scalabili.



**Figura 2.1:** Componenti chiave di LoRa (*Generata dall'AI*)

Il componente centrale dell'intero sistema è il **chip radio LoRa**, sviluppato da Semtech. Questo chip implementa la modulazione LoRa, permettendo comunicazioni a lunga distanza anche in presenza di rumore, grazie alla sua elevata sensibilità di ricezione e al basso consumo energetico. Queste caratteristiche sono fondamentali per applicazioni IoT che richiedono autonomia pluriennale con batterie di piccole dimensioni.

Spesso, questi chip vengono forniti come *moduli pre-assemblati*, come i modelli RFM95W o RFM96W. Questi moduli integrano l'antenna, i circuiti di supporto e interfacce standard (SPI, UART), semplificando notevolmente l'integrazione nei dispositivi finali e accelerando il time-to-market.

I **dispositivi finali** sono generalmente sensori o attuatori con connettività LoRa, alimentati a batteria e progettati per operare in modo autonomo. Questi dispositivi raccolgono dati da sensori ambientali (temperatura, umidità, movimento, qualità dell'aria, ecc.) e li inviano a intervalli regolari.

Al loro interno, è presente un *microcontrollore a basso consumo*, come ESP32, STM32 o le piattaforme Arduino, che gestisce il sensore, la logica applicativa e la comunicazione radio. Al momento della produzione, ai dispositivi basati su LoRa vengono assegnati diversi identificatori univoci. Questi identificatori servono per attivare e amministrare in modo sicuro il dispositivo, per garantire il trasporto sicuro dei pacchetti su una rete pubblica o privata e per fornire dati cifrati al cloud.[6].

Per quanto riguarda il consumo energetico, i dispositivi finali inviano piccoli pacchetti di dati a intervalli regolari e, durante i periodi di inattività, entrano in modalità *sleep*, consumando solo pochi microampere. Questa modalità operativa consente una durata della batteria che, in molte applicazioni, può raggiungere i 5–10 anni, riducendo notevolmente i costi di manutenzione.

Dal punto di vista della copertura, un dispositivo LoRa può trasmettere e ricevere segnali su distanze superiori a 15 km in aree rurali o poco urbanizzate. In contesti urbani, la portata si riduce tipicamente a 2–5 km, a seconda della densità edilizia, della topografia e delle interferenze elettromagnetiche.

L’architettura hardware di LoRa è progettata per massimizzare l’efficienza energetica e la semplicità di integrazione, rendendo possibile lo sviluppo di reti IoT su ampia scala anche in assenza di infrastrutture tradizionali. Grazie alla disponibilità di moduli pre-assemblati, microcontrollori versatili e gateway configurabili, LoRa si afferma come una delle tecnologie di riferimento per l’Internet of Things, in grado di abilitare soluzioni intelligenti e connesse in ambiti sempre più estesi.

## 2.4 Confronto tra tecnologie LPWAN

Nel mondo delle tecnologie LPWAN, LoRa si confronta con diverse alternative, ognuna con le proprie caratteristiche che le rendono più o meno adatte a specifici contesti applicativi. Le principali tecnologie concorrenti includono **Sigfox** e **NB-IoT**. Per avere una visione più chiara delle tecnologie menzionate, è utile suddividere le soluzioni LPWAN in due grandi categorie:

- **Reti cellulari:** tecnologie LPWAN sviluppate all’interno del consorzio *3GPP*, lo stesso che standardizza le reti mobili 4G e 5G. Operano su bande di frequenza concesse in licenza e si appoggiano all’infrastruttura degli operatori mobili esistenti (es. NB-IoT, LTE-M).
- **Reti non cellulari:** operano tipicamente su bande di frequenza non licenziate (ISM) e non dipendono dagli operatori mobili. Queste reti possono essere pubbliche o private e includono tecnologie come LoRa e Sigfox.

### 2.4.1 Sigfox

Sigfox è stata una delle prime tecnologie LPWAN a essere introdotte nel mercato IoT. Il livello fisico impiega una modulazione a banda ultra-stretta (Ultra Narrow Band, UNB), che permette un utilizzo estremamente efficiente dello spettro radio e offre un'elevata resistenza al rumore. I dettagli relativi ai protocolli di livello superiore sono proprietari e non documentati pubblicamente[7].

Grazie all'approccio UNB, Sigfox riesce a garantire un'elevata sensibilità in ricezione, consumi energetici estremamente ridotti e una progettazione hardware semplificata, il che si traduce in moduli radio e antenne a basso costo. Tuttavia, questi vantaggi sono accompagnati da un throughput massimo piuttosto limitato (circa 100 bps in uplink), rendendo la tecnologia adatta solo a scenari in cui la frequenza e la quantità di dati trasmessi sono molto contenute.

### 2.4.2 NB-IoT

Narrowband IoT (NB-IoT) è una tecnologia LPWAN cellulare sviluppata dal consorzio 3GPP come parte della *Release 13*. Anche se è integrata nello standard LTE, NB-IoT è progettata per essere il più semplice possibile, così da ridurre i costi dei dispositivi e massimizzare l'efficienza energetica [8]. Utilizza bande di frequenza concesse in licenza, le stesse di LTE, e impiega la modulazione QPSK.

Rispetto a LoRa, NB-IoT offre una maggiore affidabilità grazie all'uso di frequenze licenziate e all'infrastruttura degli operatori mobili, che garantiscono una migliore protezione dalle interferenze e un livello superiore di qualità del servizio (QoS). Inoltre, NB-IoT consente una gestione centralizzata attraverso le reti mobili esistenti, rendendo più semplice l'integrazione con servizi cloud e sistemi di provisioning automatizzato.

Tuttavia, NB-IoT presenta anche alcune limitazioni: richiede l'uso di SIM card, la registrazione in rete e spesso un abbonamento a pagamento, aumentando i costi operativi rispetto a tecnologie come LoRa. Inoltre, non consente la creazione di reti private locali, una caratteristica che invece rende LoRa più flessibile per soluzioni decentralizzate. Infine, pur essendo ottimizzato rispetto a LTE, NB-IoT tende a consumare più energia, soprattutto in scenari con trasmissioni frequenti.

In sintesi, NB-IoT è ideale per applicazioni industriali su larga scala che necessitano di un'infrastruttura affidabile e gestita centralmente, mentre LoRa rimane preferibile per implementazioni locali, a basso costo e con alta autonomia dei dispositivi periferici.

La tabella 2.2 riassume le principali caratteristiche tecniche delle tre tecnologie, evidenziando le differenze più rilevanti in termini di copertura, consumo energetico, costi e prestazioni.

Parametro	LoRa	Sigfox	NB-IoT
Copertura rurale (km)	10–15	30–50	1–10
Copertura urbana (km)	2–5	3–10	1–5
Velocità dati (Kbps)	0.3–37.5	0.1	160–250
Bande di frequenza	Non licenziate	Non licenziate	Licenziate
Topologia	Stella a stella	Stella	Cellulare
Dispositivi per BS	15.000	1.000.000	52.000
Durata batteria	5–10 anni	10–20 anni	Fino a 10 anni
Latenza	1 secondo	Diversi secondi	1.6–10 secondi
Costi di connettività	Bassi	Medi	Alti
Mobilità	Limitata	Limitata	Nativa

**Tabella 2.2:** Confronto tra tecnologie LPWAN

# Capitolo 3

## LoRaWAN

### 3.1 LoRa vs LoRaWAN

LoRa e LoRaWAN sono due elementi distinti ma strettamente interconnessi all'interno delle reti LPWAN. Nonostante vengano frequentemente utilizzati come sinonimi, in realtà operano su livelli differenti del modello di rete e svolgono ruoli specifici e complementari.

LoRa è una tecnica di modulazione wireless proprietaria sviluppata da Semtech, basata sulla tecnologia chirp spread spectrum (CSS), che opera a livello **fisico** del modello ISO/OSI e si occupa della trasmissione dei segnali radio tra un nodo e un ricevitore.

**LoRaWAN**, invece, è un protocollo di comunicazione aperto e standardizzato dalla LoRa Alliance che opera a livello **MAC** ed è basato sulla modulazione LoRa. Si tratta di un livello software che definisce come i dispositivi utilizzano l'hardware LoRa per comunicare all'interno della rete[4]. LoRaWAN gestisce aspetti fondamentali come l'autenticazione dei dispositivi, la crittografia e la comunicazione bidirezionale.

### 3.2 Architettura di rete

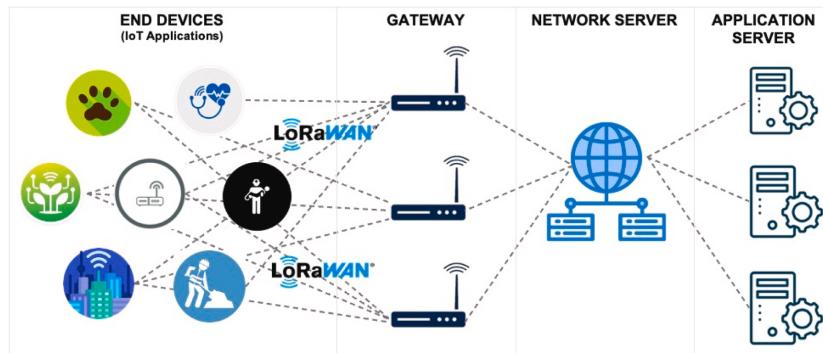
Le reti LoRaWAN sono realizzate seguendo una topologia a **stella di stelle**, in cui i dispositivi finali possono comunicare solamente con i gateway LoRaWAN e non direttamente tra di loro[9]. Diversi gateway sono poi connessi al Network Server.

Questa struttura si articola in quattro componenti principali, ciascuno con un ruolo ben definito nella gestione della rete.

- **End Devices:** dispositivi IoT alimentati a batteria e dotati di sensori o attuatori. Si occupano dell'invio e ricezione di messaggi wireless modulati LoRa.
- **Gateway:** i gateway LoRaWAN operano come ponte fra dispositivi finali e Network Server.
- **Network Server:** gestisce l'autenticazione dei dispositivi, la cifratura dei dati e l'instradamento dei dati al server applicativo appropriato.
- **Application Server:** i server applicativi sono responsabili della gestione dei dati delle applicazioni dei sensori.

Troviamo inoltre il **Join Server**, un software in esecuzione su un server che elabora i messaggi di richiesta di join inviati dai dispositivi finali.

La struttura della rete e le relazioni tra i componenti sopra elencati sono illustrate nella Figura 3.1.



**Figura 3.1:** Tipica architettura di rete LoRaWAN[10]

A differenza di altre tecnologie wireless che adoperano topologie mesh per creare reti multi-hop, LoRaWAN non attua alcun meccanismo di routing tra dispositivi. I gateway non svolgono funzioni di instradamento ma operano come semplici relay, ricevendo i messaggi dai dispositivi finali e inoltrandoli al Network Server tramite connessione IP. Questo approccio semplifica l'infrastruttura, diminuisce il consumo energetico dei nodi e assicura maggiore prevedibilità nella comunicazione, a svantaggio però della flessibilità e della copertura tipica delle reti mesh.

### 3.2.1 Implementazioni pratiche

Ci sono svariate piattaforme che facilitano l'implementazione e la gestione di reti LoRaWAN.

**The Things Network** TTN è una rete globale open-source che fornisce infrastrutture condivise di gateway, network server e application server, consentendo a sviluppatori e comunità di connettere dispositivi LoRaWAN senza la necessità di sviluppare un'infrastruttura propria. Grazie a TTN, è possibile sperimentare e distribuire soluzioni IoT su ampia scala con un approccio collaborativo e decentralizzato.

**ChirpStack** ChirpStack è invece una suite software open-source che fornisce tutti i componenti necessari per implementare una rete LoRaWAN privata. Diversamente da TTN, ChirpStack è pensato per essere installato e amministrato autonomamente, offrendo maggiore controllo e personalizzazione dell'infrastruttura di rete.

### 3.2.2 Classi di dispositivi

I dispositivi finali (o end-device) basati su LoRa possono operare secondo tre distinte modalità, denominate classi, che cambiano per caratteristiche di latenza e consumo energetico. Tutti i dispositivi finali devono supportare il funzionamento di Classe A, mentre le modalità di Classe B e Classe C sono opzionali. Queste classi definiscono il modo in cui i dispositivi interagiscono con la rete[6].

- **Classe A:** il dispositivo finale trascorre la maggior parte del tempo in modalità **sleep** per massimizzare il risparmio energetico. Quando si verifica un cambiamento che richiede una trasmissione, il dispositivo si sveglia e avvia un **uplink** (trasmissione dal dispositivo finale verso la rete). Ultimata la trasmissione, il dispositivo apre due brevi finestre di ricezione sequenziali (*Rx1* e *Rx2*) per attendere eventuali messaggi **downlink** (trasmissione dalla rete verso il dispositivo finale). Se non riceve alcuna risposta entro queste finestre, torna in modalità sleep fino alla trasmissione successiva.
- **Classe B:** i dispositivi di questa classe estendono le funzionalità di Classe A aprendo periodicamente delle finestre di ricezione dette **ping slot** per ricevere messaggi di downlink.

Senza un riferimento temporale condiviso, il server non saprebbe quando il dispositivo è pronto a ricevere, per tale motivo si utilizza un **beacon**, ovvero un messaggio trasmesso da gateway LoRaWAN ad intervalli regolari, che contiene un timestamp e un CRC per il controllo degli errori. Questa possibilità rende la classe particolarmente adatta al monitoraggio di sensori e attuatori.

- **Classe C:** dispositivi che estendono anch'essi le capacità della Classe A mantenendo le finestre di ricezione sempre aperte, tranne quando trasmettono un uplink. Questa modalità consente ai dispositivi di ricevere downlink in qualunque momento con una latenza estremamente bassa, a discapito però di un consumo energetico notevolmente maggiore.

### 3.2.3 Gateway LoRaWAN

I gateway LoRaWAN costituiscono l'elemento cruciale dell'infrastruttura di rete. Al loro interno usano i chip radio LoRa, descritti nel capitolo precedente, per consentire la ricezione simultanea su molteplici frequenze e spreading factor.

È fondamentale notare che non sussiste un'associazione unica tra dispositivi e gateway: un singolo end-device può essere ricevuto da più gateway contemporaneamente, migliorando l'affidabilità e la ridondanza del sistema.

I gateway si suddividono principalmente in due categorie:

- **Single-channel:** versioni economiche, limitate a un solo canale, ideali per test e prototipazione ma non adatte in contesti reali.
- **Multi-channel:** dispositivi professionali capaci di gestire fino a 8 o più canali simultaneamente, essenziali per supportare reti su larga scala con molti nodi.

Inoltre, ci sono versioni *indoor* e *outdoor*, dove le ultime sono dotate di contenitori resistenti alle intemperie e, spesso, di alimentazione solare, perfette per l'installazione in ambienti remoti o privi di accesso alla rete elettrica.

### 3.3 Livello MAC

Dopo avere esaminato l'architettura complessiva delle reti LoRaWAN, è fondamentale esaminare il funzionamento del livello MAC (Medium Access Control), che rappresenta il protocollo di comunicazione principale. Il livello MAC si occupa di stabilire la connessione tra i vari elementi della rete, gestire l'invio e la ricezione dei comandi MAC (identificati in base al tipo di messaggio, come mostrato in figura 3.2) e, infine, di aggiungere un'intestazione MAC (header) all'inizio e un codice di integrità del messaggio (MIC) alla fine del MAC payload.

CID	Command	Transmitted by End-device	Transmitted by Network Server	Brief Description
0x02	<i>LinkCheckReq</i>	x		Used by an end-device to validate its connectivity to a network.
0x02	<i>LinkCheckAns</i>		x	Answers <i>LinkCheckReq</i> . Contains the received signal power estimation, which indicates the quality of reception (link margin) to the end-device.
0x03	<i>LinkADRReq</i>		x	Requests the end-device to change data rate, TX power, redundancy, or channel mask.
0x03	<i>LinkADRAns</i>	x		Acknowledges <i>LinkADRReq</i> .
0x04	<i>DutyCycleReq</i>		x	Sets the maximum aggregated transmit duty cycle of an end-device.
0x04	<i>DutyCycleAns</i>	x		Acknowledges <i>DutyCycleReq</i> .
0x05	<i>RXParamSetupReq</i> <sup>7</sup>		x	Sets the reception slot parameters.
0x05	<i>RXParamSetupAns</i>	x		Acknowledges <i>RXParamSetupReq</i> .
0x06	<i>DevStatusReq</i>		x	Requests the status of the end-device.
0x06	<i>DevStatusAns</i>	x		Returns the status of the end-device, namely its battery level and its radio status.
0x07	<i>NewChannelReq</i>		x	Creates or modifies the definition of a radio channel.
0x07	<i>NewChannelAns</i>	x		Acknowledges <i>NewChannelReq</i> .
0x08	<i>RXTimingSetupReq</i> <sup>7</sup>		x	Sets the timing of the reception slots.
0x08	<i>RXTimingSetupAns</i>	x		Acknowledges <i>RXTimingSetupReq</i> .
0x09	<i>TXParamSetupReq</i> <sup>7</sup>		x	Used by a Network Server to set the maximum allowed dwell time and MaxEIRP of end-device, based on local regulations.
0x09	<i>TXParamSetupAns</i>	x		Acknowledges <i>TXParamSetupReq</i> .
0x0A	<i>DIChannelReq</i> <sup>7</sup>		x	Modifies the definition of a downlink RX1 radio channel by shifting the downlink frequency from the uplink frequencies (i.e. creating an asymmetric channel).
0x0A	<i>DIChannelAns</i>	x		Acknowledges <i>DIChannelReq</i> .
0x0B to 0x0C				RFU
0x0D	<i>DeviceTimeReq</i>	x		Used by an end-device to request the current GPS time.
0x0D	<i>DeviceTimeAns</i>		x	Answers <i>DeviceTimeReq</i> .
0x0E to 0x0F				RFU
0x10 to 0x1F				Class B commands (cf. Sections 12).
0x20 to 0x2F				Reserved for Class C commands.
0x30 to 0x7F				RFU
0x80 to 0xFF	Proprietary	x	x	Reserved for proprietary network command extensions.

Figura 3.2: Comandi MAC LoRaWAN[11]

### 3.3.1 Formato dei frame MAC

Tutti i pacchetti LoRaWAN, sia in uplink che in downlink, trasportano un *PHY payload* [11] strutturato come segue:

$$\text{PHYPayload} = \text{MHDR} \parallel \text{MACPayload} \parallel \text{MIC}$$

Nel caso dei frame di join, la struttura si semplifica in:

$$\text{PHYPayload} = \text{MHDR} \parallel \text{Join-Request/Join-Accept} \parallel \text{MIC}$$

**MAC Header (MHDR)** Il campo MHDR occupa 1 byte ed è suddiviso in tre parti:

Bit	7–5	4–2	1–0
Campo	FType	RFU	Major

**Tabella 3.1:** Formato del campo MHDR

FType indica il tipo di frame, RFU è un campo riservato per impieghi futuri e attualmente inutilizzato, e Major denota la versione principale del protocollo LoRaWAN con cui il frame è stato codificato. Tramite FType LoRaWAN distingue sei diversi tipi di frame MAC:

FType	Descrizione
000	Join-Request
001	Join-Accept
010	Dati uplink non confermati
011	Dati downlink non confermati
100	Dati uplink confermati
101	Dati downlink confermati
110	RFU (Riservato per uso futuro)
111	Proprietario

**Tabella 3.2:** Tipi di frame MAC LoRaWAN

**MAC Payload (MACPayload)** Il campo MACPayload contiene i dati del livello MAC. La sua estensione massima  $M$  dipende dalla regione e dal data rate impiegati. Qualsiasi frame ricevuto, sia da un end-device che dal Network Server, che superi tale lunghezza viene *silenziosamente scartato*, ovvero ignorato senza generare errori.

$$\text{MACPayload} = \text{FHDR} \parallel \text{FPort (opzionale)} \parallel \text{FRMPayload (opzionale)}$$

*FHDR* (Frame Header) contiene informazioni di controllo fondamentali per la gestione della comunicazione tra il dispositivo finale e la rete. *FPort* è un campo opzionale che indica la porta applicativa a cui è destinato il messaggio. *FRMPayload* è anch'esso un campo opzionale che trasporta il contenuto dati effettivo, che può essere un messaggio dell'applicazione o comandi MAC, a seconda del valore di *FPort*.

Campo	Dimensione (byte)	Descrizione
DevAddr	4	Indirizzo del dispositivo finale (short address)
FCtrl	1	Byte di controllo del frame
FCnt	2	Contatore del frame
FOpts	0–15	Campo opzionale per comandi MAC

**Tabella 3.3:** Struttura del campo FHDR

Il contenuto del campo *FCtrl* varia tra frame uplink e downlink e contiene diversi bit di controllo importanti per la gestione della comunicazione:

Bit	Frame Downlink	Descrizione
7	ADR	Controllo data rate adattivo
6	RFU	Riservato per usi futuri
5	ACK	Bit di acknowledgment
4	FPending	Frame in attesa
3–0	FOptsLen	Lunghezza delle opzioni frame

Bit	Frame Uplink	Descrizione
7	ADR	Controllo data rate adattivo
6	ADRACKReq	Richiesta ACK per ADR
5	ACK	Bit di acknowledgment
4	ClassB	Abilitazione Classe B
3–0	FOptsLen	Lunghezza delle opzioni frame

**Tabella 3.4:** Struttura del campo FCtrl nei frame Downlink e Uplink

L'**Adaptive Data Rate** (ADR) è un meccanismo per ottimizzare la velocità di trasmissione, il tempo di comunicazione ed il consumo energetico della rete[4]. Quando il bit ADR è attivato nell'uplink, il Network Server può controllare il numero di ritrasmissioni, il data rate e la potenza di trasmissione del dispositivo finale attraverso adeguati comandi MAC.

Per garantire la robustezza della connessione, il meccanismo ADR include anche un algoritmo di backoff: se un dispositivo invia un certo numero di uplink consecutivi senza ricevere un downlink di conferma, imposta il bit ADRACKReq nel successivo uplink come richiesta di acknowledgment per ADR. Se la rete non risponde entro un certo numero di tentativi, il dispositivo incrementa progressivamente la potenza di trasmissione fino al massimo consentito e, se necessario, diminuisce il data rate in modo da ripristinare la connettività.

**Message Integrity Code (MIC)** Il campo MIC è un codice di verifica dell'integrità del messaggio lungo 4 byte, usato appunto per garantire l'autenticità e l'integrità del frame trasmesso. Il codice è calcolato su tutti i campi del messaggio e poi aggiunto al messaggio stesso. I dettagli del processo di calcolo e della gestione delle chiavi saranno approfonditi nella sezione dedicata alla sicurezza.

## 3.4 Attivazione dei dispositivi

Per prendere parte ad una rete LoRaWAN, ogni dispositivo finale deve essere registrato. Questa procedura è nota come **attivazione**, e può avvenire in due modi.

### 3.4.1 Attivazione Over-The-Air (OTAA)

OTAA è il metodo di attivazione più sicuro e raccomandato per i dispositivi finali [4]. L'attivazione ha inizio con la procedura di join, che viene sempre avviata dal dispositivo finale. Essa richiede lo scambio di due messaggi MAC: la richiesta di adesione (*Join-Request*) inviata dal dispositivo al server di rete, ed il messaggio di accettazione (*Join-Accept*) spedito dal server.

Per iniziare la procedura, ogni dispositivo deve essere in possesso di una chiave segreta a 128 bit, la **AppKey**, cioè una chiave condivisa solamente tra il dispositivo finale ed il Join Server.

Il messaggio di **Join-Request** è composto dai seguenti campi:

Dimensione	Campo
8 byte	AppEUI
8 byte	DevEUI
2 byte	DevNonce

**Tabella 3.5:** Campi del messaggio Join-Request

**AppEUI** è un identificatore di applicazione a 64 bit che identifica, in modo univoco, l'entità in grado di elaborare il frame di richiesta.

**DevEUI** è un identificatore di dispositivo a 64 bit che identifica in modo univoco il dispositivo finale.

Infine, **DevNonce** è un valore univoco, casuale, di 2 byte che viene generato dal dispositivo finale. Il server di rete usa questa variabile per tenere traccia delle svariate richieste di join. Se un dispositivo finale manda una richiesta con un DevNonce già utilizzato (tentativo di replay attack), il server rifiuta la richiesta.

Sebbene il messaggio sia autenticato mediante il codice MIC, calcolato su tutti i campi della Join-Request utilizzando la AppKey, il contenuto del messaggio non è cifrato: tutti i campi vengono trasmessi in chiaro.

Il secondo messaggio della procedura è il messaggio di **Join-Accept**, composto dai seguenti campi:

Dimensione (byte)	Campo
3	AppNonce
3	NetID
4	DevAddr
1	DLSettings
1	RxDelay
16	CFList (facoltativo)

**Tabella 3.6:** Campi del messaggio Join-Accept

**AppNonce** è un valore univoco fornito dal server e impiegato dal dispositivo per generare le chiavi di sessione.

**NetID** è un identificatore univoco assegnato a ogni rete LoRaWAN. I 7 bit più significativi di tale ID rappresentano il *NwkID*, che identifica il tipo o il proprietario della rete.

**DevAddr** è un indirizzo dinamico a 32 bit assegnato al dispositivo dal server di rete, utilizzato per identificarlo univocamente all'interno della rete attuale. È possibile che un server di rete assegni lo stesso DevAddr a più dispositivi. Questo non crea alcun problema per il server di rete, poiché il server identifica il dispositivo utilizzando la combinazione di DevAddr e NwksKey (chiave di sessione), che è univoca per ciascun dispositivo[12].

**DLSettings** è un campo di 1 byte che contiene le impostazioni relative alla comunicazione downlink.

**RxDelay** indica il ritardo, espresso in secondi, tra la trasmissione del dispositivo finale (TX) e l'apertura della finestra di ricezione (RX).

Infine, **CFList** è un elenco opzionale che contiene le frequenze dei canali radio disponibili per la rete a cui il dispositivo finale si sta connettendo.

Il server di rete invia il messaggio di **Join-Accept** che, a differenza della richiesta, viene cifrato usando la **AppKey**. Se la richiesta di join non viene accettata, il server non invia alcuna risposta al dispositivo.

Quando il dispositivo finale decifra il messaggio **Join-Accept** l'attivazione è completata.

### 3.4.2 Attivazione tramite personalizzazione (ABP)

L'attivazione tramite personalizzazione, nota come *ABP* (Activation By Personalization), connette un dispositivo finale direttamente a una rete preselezionata, saltando la procedura di attivazione over-the-air [4].

È un metodo di attivazione meno sicuro e meno flessibile, poiché i dispositivi finali non possono cambiare provider di rete senza aggiornare manualmente le chiavi di sicurezza. Ciò accade perché, in questa modalità di attivazione, l'indirizzo del dispositivo (**DevAddr**) e le due chiavi di sessione sono già memorizzati nel dispositivo finale, anziché essere derivati dinamicamente durante la procedura di join.

Ogni dispositivo finale dispone ovviamente di una coppia di chiavi di sessione univoca: la **NwkSKey**, condivisa con il Network Server, e la **AppSKey**, condivisa con l'Application Server.

## 3.5 Sicurezza

La sicurezza in LoRaWAN è progettata per rispettare i principi fondamentali del protocollo: basso consumo energetico, semplicità di implementazione, basso costo e alta scalabilità[13].

### 3.5.1 Advanced Encryption Standard

I sistemi di sicurezza che osserveremo, in questo e nei capitoli successivi, si fondano sull'algoritmo crittografico *AES*, standardizzato dal NIST nel documento *FIPS PUB 197*[14]. AES è un cifrario a blocchi simmetrico, in cui sia l'input che l'output sono composti da sequenze di 128 bit, noti come **blocchi**. La lunghezza della chiave crittografica può invece essere pari a 128, 192 o 256 bit, indicate rispettivamente come AES-128, AES-192 e AES-256[14].

Come anticipato, AES appartiene alla famiglia dei cifrari a blocchi che, secondo quanto riportato nell’ “*Handbook of Applied Cryptography*” [15], possono essere definiti come funzioni che mappano blocchi di  $n_b$  bit di testo in chiaro in blocchi di  $n_b$  bit di testo cifrato, parametrizzate da una chiave di lunghezza  $n_k$ . In altre parole, un cifrario a blocchi può essere considerato come un insieme di  $2^{n_k}$  permutazioni booleane, chiamate **round transformations**, e ciascuna applicazione di esse costituisce un **round**.

Sebbene l’algoritmo originario Rijndael fosse progettato per gestire blocchi e chiavi di dimensioni variabili, lo standard AES limita questi parametri alle dimensioni sopra indicate. Per poter applicare AES a messaggi di lunghezza variabile, è necessario ricorrere alle **modalità operative**. Una modalità operativa è una tecnica che definisce come il cifrario a blocchi (AES) deve essere impiegato per cifrare e decifrare messaggi più lunghi della dimensione fissa del blocco (128 bit). Queste modalità saranno illustrate nei paragrafi in cui vengono effettivamente utilizzate.

### 3.5.2 Crittografia e Chiavi di Sicurezza

LoRaWAN adotta un’architettura basata su chiavi pre-condivise, una scelta che garantisce un’efficienza energetica ottimale, soprattutto per dispositivi *IoT* a basso consumo.

**Nota:** nella versione originale dello standard LoRaWAN, il campo identificativo dell’entità responsabile della gestione della procedura di join è chiamato AppEUI. Nelle versioni più recenti dello standard, questo campo è stato ridenominato in JoinEUI. Entrambi i termini denotano lo stesso identificatore a 64 bit, che identifica il Join Server. Analogamente, il campo AppNonce, generato dal Join Server, è stato ridenominato in JoinNonce.

Ogni dispositivo LoRaWAN è dotato di:

- una chiave *AES* a 128 bit (AppKey);
- un identificatore univoco globale (DevEUI);
- un identificatore del *Join Server* (JoinEUI).

**Nota:** l’assegnazione degli identificatori EUI-64 avviene tramite l’OUI (Organizationally Unique Identifier) rilasciato dall’IEEE Registration Authority.

Questi elementi vengono utilizzati nella procedura di attivazione del dispositivo per instaurare una comunicazione sicura.

### 3.5.3 Generazione delle Chiavi di Sessione

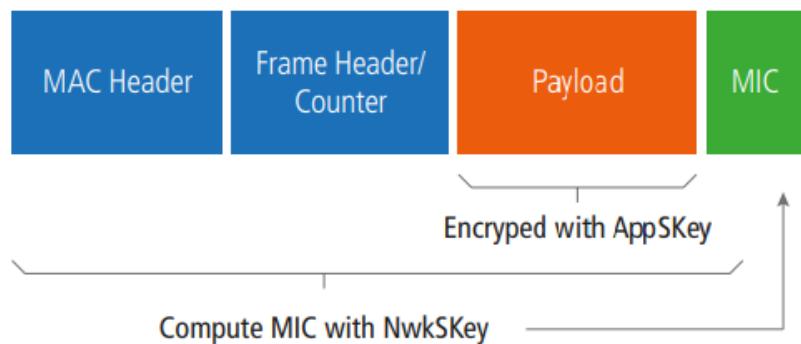
A seguito della procedura di *join*, il server genera le chiavi di sessione nel seguente modo:

$$\begin{aligned} \text{NwkSKey} &= \text{aes128\_encrypt}\left(\text{AppKey}, 0x01 \parallel \text{JoinNonce} \parallel \text{NetID} \parallel \text{DevNonce} \parallel \text{pad16}\right) \\ \text{AppSKey} &= \text{aes128\_encrypt}\left(\text{AppKey}, 0x02 \parallel \text{JoinNonce} \parallel \text{NetID} \parallel \text{DevNonce} \parallel \text{pad16}\right) \end{aligned}$$

La NwkSKey viene distribuita al *network server* di LoRaWAN, mentre la AppSKey viene condivisa con il *server applicativo*.

### 3.5.4 Proprietà di Sicurezza

La Figura 4.1 fornisce una panoramica dei meccanismi di sicurezza che saranno illustrati nei paragrafi successivi, applicati alla struttura di un frame LoRaWAN.

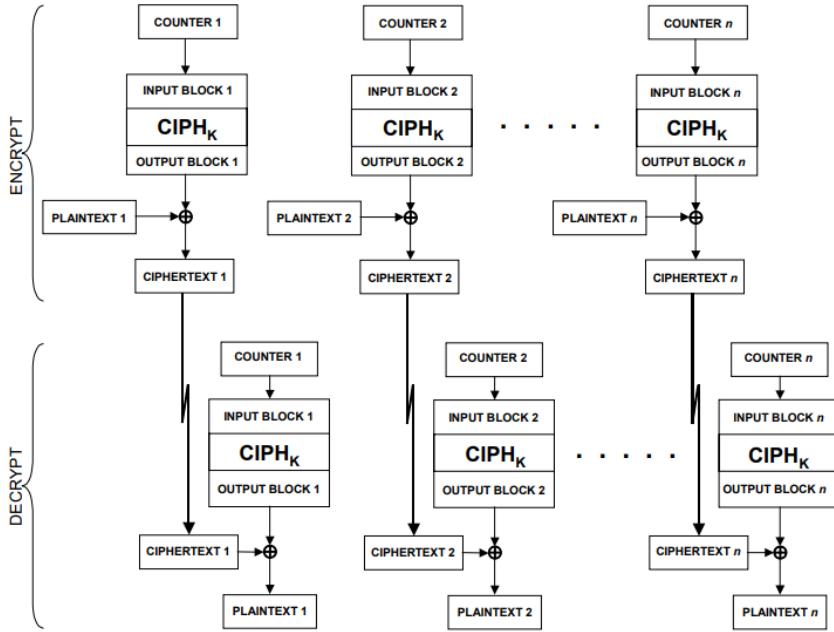


**Figura 3.3:** Applicazione dei meccanismi di sicurezza al frame LoRaWAN[13]

**Riservatezza** Il contenuto dei messaggi, ovvero il *payload*, è cifrato attraverso l'algoritmo *AES-CTR*, impiegando la chiave AppSKey. La modalità *CTR* (Counter Mode Encryption) applica l'algoritmo *AES* su una serie di blocchi noti come contatori, generando uno *stream* di output (bitstream). Ciascun blocco del bitstream viene poi combinato, mediante un'operazione di XOR, con il corrispondente blocco del testo in chiaro, producendo il testo cifrato [16].

È fondamentale che ogni contatore sia diverso dagli altri all'interno della sequenza. In LoRaWAN, la sequenza dei contatori è generata a partire dal contatore di frame FCnt, assicurando così l'unicità necessaria per garantire la sicurezza della cifratura.

La Figura 3.4 mostra il funzionamento della modalità *AES-CTR*.



**Figura 3.4:** Funzionamento di *AES* in modalità *CTR*[16]

**Autenticazione e Integrità** Ogni frame include un codice di verifica dell'integrità, conosciuto come *MIC* (*Message Integrity Code*) o *MAC* (*Message Authentication Code*), che viene calcolato mediante *AES-CMAC*.

Quest'ultimo è uno strumento crittografico basato su *AES* che fornisce garanzie di integrità e autenticità del messaggio, garantendo una protezione molto più robusta rispetto a un semplice checksum o a un codice di rilevamento errori: mentre questi ultimi rilevano solo modifiche accidentali dei dati, *AES-CMAC* è progettato per identificare sia modifiche accidentali sia alterazioni intenzionali e non autorizzate[17].

Per calcolare il *MIC*, *AES-CMAC* non opera direttamente sul singolo messaggio, ma lo combina con un blocco di controllo chiamato *B0*.

Il calcolo del *MIC* può essere sintetizzato come segue:

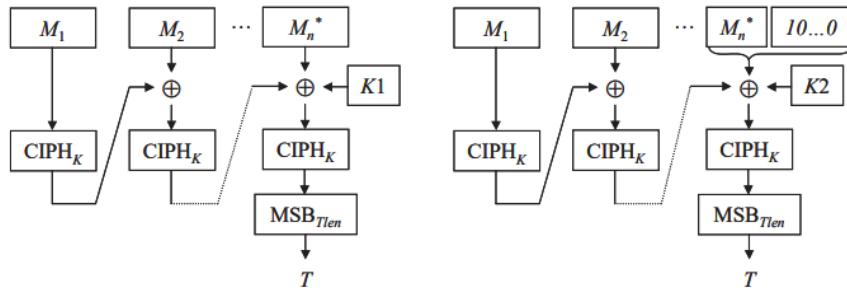
$$\text{MIC} = \text{aes128\_cmac}\left(\text{NwkSKey}, \text{B0} \parallel \text{msg}\right)$$

Dove *B0* è costruito come:

$$\text{B0} = 0x49 \parallel 0x00000000 \parallel \text{Dir} \parallel \text{DevAddr} \parallel \text{FCnt} \parallel 0x00 \parallel \text{len}(\text{msg})$$

Il messaggio concatenato a B0 viene quindi suddiviso in blocchi di dimensione fissa e processato iterativamente dall'*AES-CMAC*, dove ogni blocco viene combinato con l'output del blocco precedente e cifrato con *AES*.

Alla fine di questo processo vengono utilizzate due subkey segrete, K1 e K2, derivate dalla NwkSKey, il cui impiego dipende dalla lunghezza del messaggio. La figura 3.5 illustra proprio come la gestione dei messaggi varia in base alla loro lunghezza: a sinistra è riportato il caso in cui la lunghezza del messaggio è un multiplo esatto del blocco, a destra il caso opposto.



**Figura 3.5:** Funzionamento dell'algoritmo *AES-CMAC*[17]

### 3.5.5 Contatore di frame

Un elemento fondamentale per la sicurezza è il contatore di frame FCnt, che aiuta a garantire sia l'integrità che la riservatezza delle comunicazioni, le quali avvengono in un ambiente radio in cui chiunque potrebbe intercettare i messaggi trasmessi. Sebbene non sia possibile decifrare il contenuto, né alterare i dati senza compromettere il *MIC*, rimane comunque possibile catturare e ritrasmettere messaggi precedenti, configurando un attacco di tipo *replay*.

Per contrastare questa vulnerabilità, LoRaWAN implementa il contatore di frame FCnt, che viene inizializzato a zero durante l'attivazione del dispositivo e incrementato ad ogni trasmissione verso il *network server*. Quest'ultimo tiene traccia dell'ultimo valore di FCnt ricevuto da ciascun dispositivo e rifiuta automaticamente qualsiasi messaggio con un contatore inferiore o uguale a quello precedentemente elaborato.

# Capitolo 4

## Meshtastic

### 4.1 Reti mesh a lungo raggio

Sebbene le LPWAN tradizionali adottino tipicamente topologie a **stella**, alcune implementazioni adoperano le medesime tecnologie radio, come **LoRa**, per creare **reti mesh decentralizzate**.

In queste architetture ogni nodo comunica direttamente con i nodi adiacenti, formando appunto una *struttura a maglia*[18], in cui non è presente alcun gateway centrale di raccolta o smistamento dati. L'architettura mesh garantisce **scalabilità**, agevolando l'aggiunta di nuovi nodi senza compromettere le prestazioni, e **resilienza**, poiché la rete continua a funzionare anche in caso di malfunzionamento di alcuni nodi, grazie ai *percorsi multipli* disponibili che consentono di instradare i dati attraverso nodi alternativi.

Fra le implementazioni più rilevanti di questo approccio figura **Meshtastic**, un progetto open-source che sfrutta la tecnologia LoRa per creare reti di comunicazione mesh a lunga distanza[19].

### 4.2 Architettura di rete

Per comprendere il funzionamento di Meshtastic, è necessario analizzare la sua architettura di rete, strutturata su due livelli: **fisico** e **logico**.

La **rete fisica** Meshtastic è composta da un insieme di nodi che dividono i parametri di modulazione LoRa, descritti nella Sezione 2.2.1, i quali vincolano ciascun nodo a partecipare a una sola rete fisica.

All'interno di questa rete fisica, Meshtastic consente la creazione di sottoreti logiche, dette **“canali”**, che definiscono i gruppi di nodi autorizzati a decifrare messaggi specifici.

Tutti i nodi della rete fisica ricevono i messaggi trasmessi, ma solo quelli appartenenti allo stesso canale sono in grado di decifrarli ed interpretarli.

### 4.2.1 Configurazione dei canali

Ogni canale è identificato da un *nome* e da una *chiave crittografica*. È presente un canale predefinito (**canale 0**), privo di nome e con una chiave crittografica standard (AQ==).

Ogni nodo può aderire a un massimo di **otto canali**, configurati in modo che i canali attivi siano consecutivi, senza interruzioni da canali disabilitati.

Ogni canale ha un ruolo specifico tra questi:

- **PRIMAY** o 1: è il canale creato durante la configurazione iniziale; ne esiste solo uno e non può essere disabilitato. Le trasmissioni periodiche, come quelle relative a posizione e telemetria, vengono spedite tramite questo canale.
- **SECONDARY** o 2: canali definiti dall'utente, dove è possibile cambiare la chiave del canale (PSK).
- **DISABLED** o 0: canali non disponibili all'uso, le cui impostazioni sono riportate ai valori predefiniti.

Dopo aver definito il ruolo di un canale, è possibile personalizzarne le impostazioni:

- **Nome del canale:** un identificatore di massimo 12 byte utilizzato per distinguere i vari canali. Alcuni nomi hanno significati specifici; ad esempio, un canale senza nome indica il canale primario di default, mentre il canale denominato admin è utilizzato sui canali secondari come canale di amministrazione per la gestione remota dei nodi.
- **Chiave crittografica (PSK)**, che può essere:
  - 0 byte, ovvero nessuna chiave;
  - 16 byte, che indica l'utilizzo di AES-128;
  - 32 byte, che indica l'utilizzo di AES-256.
- **Downlink:** se abilitato, i messaggi ricevuti dalla rete Internet pubblica possono essere inoltrati nella rete mesh locale. Di default, questa opzione è disattivata.

- **Uplink:** se abilitato, i messaggi provenienti dalla rete mesh possono essere inoltrati a Internet tramite un nodo configurato come gateway. Anche questa opzione è disattivata di default.

Per semplificare la configurazione e la condivisione delle impostazioni dei canali e dei parametri LoRa tra più nodi, Meshtastic utilizza codici QR o URL di canale attraverso i quali è possibile applicare automaticamente tutte le impostazioni del canale e della configurazione LoRa incluse[19].

#### 4.2.2 Classificazione dei nodi

All'interno di una rete Meshtastic i vari dispositivi possono ricoprire ruoli differenti, che determinano il loro comportamento nella ritrasmissione dei pacchetti e la loro visibilità nella topologia della rete:

- **Client:** dispositivo di messaggistica autonomo o connesso all'app. I client possono presentare tre configurazioni principali:
  1. CLIENT: inoltra i pacchetti se rileva che nessun altro nodo lo ha ancora fatto.
  2. CLIENT\_MUTE: non invia i pacchetti ricevuti da altri dispositivi.
  3. CLIENT\_HIDDEN: trasmette i pacchetti solo quando necessario per motivi di discrezione o per ridurre i consumi energetici.
- **Tracker:** dispositivo specializzato nella trasmissione periodica della posizione GPS, utile per il monitoraggio di beni o persone. Di norma non ritrasmette il traffico altrui.
- **Sensor:** dispositivo dedicato all'invio di pacchetti di telemetria, utile per raccogliere dati ambientali o di altri sensori. Di norma non inoltra il traffico altrui.
- **Router:** nodo infrastrutturale che ritrasmette ogni pacchetto una sola volta, indipendentemente dal comportamento degli altri nodi.
- **Repeater:** analogamente al router, ritrasmette sempre ogni pacchetto una sola volta, ma opera in modo trasparente: non compare nell'elenco dei nodi né nella topologia della rete. Viene tipicamente posizionato in luoghi strategici per estendere la copertura complessiva.

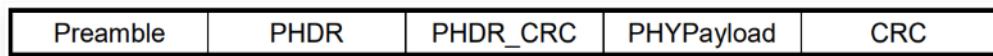
### 4.3 Protocollo di comunicazione

`Meshtastic` impiega un protocollo di routing personalizzato e leggero, spesso definito **Managed Flood Routing**[20]; un protocollo fortemente ispirato all'algoritmo di routing mesh della libreria Radiohead.

Opera su quattro livelli distinti, con approcci diversificati per messaggi broadcast e diretti.

**Layer 0: LoRa Radio** Questo livello gestisce la trasmissione radio, occupandosi della modulazione del segnale, dove i dati vengono convertiti in simboli LoRa per la trasmissione.

Ogni pacchetto trasmesso inizia con un preambolo di 16 simboli (invece del minimo di 8), impiegato per sincronizzare il ricevitore con il trasmettitore[4]. Segue l'intestazione fisica LoRa (PHDR) che contiene informazioni sulla lunghezza del pacchetto e una parola di sincronizzazione per distinguere le reti (per `Meshtastic` è 0x2B). Si tratta di un header aggiunto automaticamente dal chipset LoRa ed è specifico del livello fisico.

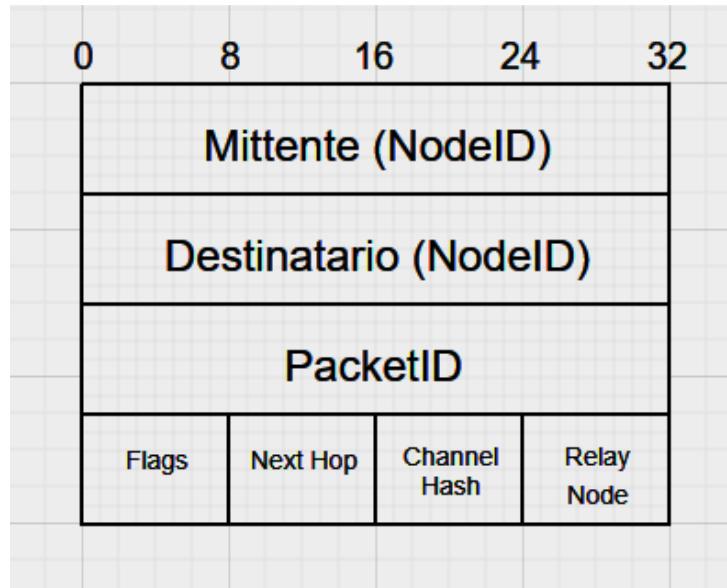


**Figura 4.1:** Formato del pacchetto di livello fisico LoRa[4]

`Meshtastic` riduce l'overhead del pacchetto fisico LoRa, omettendo alcuni campi opzionali come il CRC aggiuntivo per ottimizzare l'efficienza nelle comunicazioni mesh.

**Layer 1: Unreliable Zero Hop Messaging** Questo livello gestisce la trasmissione diretta e non affidabile di pacchetti LoRa tra due nodi adiacenti. Non è previsto alcun meccanismo di conferma o di ritrasmissione.

Un pacchetto generato da un dispositivo Meshtastic include un header applicativo di 16 byte, diverso da quello del layer 0:



**Figura 4.2:** Header Meshtastic

Il **NodeID** è un identificatore univoco del nodo che deriva dagli ultimi 4 byte dell’indirizzo MAC.

Il **Relay node** serve ad identificare il nodo che sta attualmente ritrasmettendo il pacchetto, spesso diverso dal mittente originale.

All’interno dei **flags** sono presenti i seguenti bit di controllo:

- **HopLimit (3 bit)**. Come possiamo vedere dalla dimensione in bit del campo, il massimo di salti è 7.
- **WantAck**, indica se il mittente richiede una conferma di ricezione del pacchetto.
- **ViaMQTT**<sup>1</sup>. Un nodo Meshtastic impostato come **gateway MQTT** funge da ponte verso un broker MQTT. Quando un dispositivo Meshtastic riceve un messaggio, lo decifra usando la chiave adatta e, una volta decifrato, trasmette il payload all’applicazione Meshtastic.
- **HopStart (3 bit)**, indica l’HopLimit iniziale.

---

<sup>1</sup>MQTT (Message Queuing Telemetry Transport) è un protocollo di comunicazione largamente impiegato nell’IoT per collegare reti locali con il mondo esterno.

Tale intestazione viene trasmessa in chiaro come byte grezzi, consentendo un risparmio nel tempo di trasmissione ma soprattutto un filtraggio hardware: le radio riceventi possono leggere l'intestazione e, se necessario, scartare il pacchetto senza attivare inutilmente la CPU del dispositivo.

Dopo l'intestazione troviamo il payload (*SubPacket*) che contiene i dati effettivi trasmessi. A differenza dell'intestazione, il payload viene serializzato con `Protobuf`<sup>2</sup> e poi cifrato. La dimensione massima del payload è di 237 bytes; se un messaggio eccede tale lunghezza, viene troncato.

Prima di trasmettere un pacchetto, `Meshtastic` utilizza una tecnica di accesso al canale chiamata **CSMA/CA** (Carrier-Sense Multiple Access with Collision Avoidance).

Il nodo effettua un rilevamento dell'attività del canale (Channel Activity Detection, **CAD**) e, se il canale risulta occupato, attende che si liberi prima di procedere.

Per ridurre la probabilità di collisioni quando più nodi tentano di trasmettere, ogni nodo attende un intervallo di tempo casuale (*slot time*) scelto all'interno di una **finestra di contesa**. Questa finestra si adatta dinamicamente alle condizioni del canale e al livello di traffico: ad esempio, in caso di SNR basso, la finestra si riduce per favorire i nodi più distanti nella ritrasmissione, mentre in caso di traffico elevato si allarga per ridurre le collisioni.

**Router** e **Repeater** hanno priorità nella ritrasmissione, potendo trasmettere anche se altri nodi sono già attivi.

**Layer 2: Reliable Zero Hop Messaging** Mentre il livello 1 gestisce una trasmissione non affidabile, questo livello estende la messaggistica per assicurare che i pacchetti pervengano in modo affidabile al nodo successivo del percorso. A tal fine, `Meshtastic` usa il flag `WantAck`. Quando questo è attivo, il mittente richiede una conferma di ricezione e, in assenza di risposta entro un certo lasso di tempo, il mittente ritrasmette il pacchetto tramite il livello 1, per un massimo di 3 tentativi. Se l'assenza di risposta persiste, il nodo genera un `NAK` internamente.

Per i messaggi broadcast, il flag `WantAck` è gestito diversamente per evitare di intasare il canale: il mittente rimane in ascolto per verificare se almeno un nodo sta ritrasmettendo; se rileva tale attività, il modulo `FloodingRouter.cpp` genera un *ACK* implicito e lo invia al mittente originale.

---

<sup>2</sup>Protocol Buffers, un metodo per serializzare dati strutturati in formato binario.

**Layer 3: Multi-Hop Messaging** L’ultimo livello gestisce la logica dei messaggi broadcast e multi-hop, meccanismo che consente ai pacchetti di propagarsi attraverso più nodi intermedi, estendendo la portata della rete oltre il singolo salto fra nodi adiacenti. Il protocollo impiegato è noto come **flooding gestito** (Managed Flood Routing), una variante del flooding tradizionale. A differenza del flooding ”puro”, in cui un pacchetto può continuare propagarsi senza fine, saturando il canale, il flooding gestito implementa dei controlli volti a limitare duplicazioni e congestioni.

Come abbiamo osservato, ciascun pacchetto nell’intestazione contiene un campo chiamato HopLimit, che viene decrementato ad ogni ritrasmissione, e un identificatore che consente ai nodi di riconoscere i messaggi già ricevuti e di ignorarli. Quando il valore di HopLimit raggiunge zero, il pacchetto non viene più ritrasmesso e viene scartato.

Per limitare ulteriormente le ridondanze, un nodo interrompe la ritrasmissione se rileva che un vicino ha già confermato di aver ricevuto il pacchetto. L’inoltro dei pacchetti è comunque coordinato dai meccanismi di accesso al canale esposti nel Layer 1, che regolano la ritrasmissione dei nodi, riducendo le collisioni.

### 4.3.1 Messaggi diretti

Nelle reti mesh, oltre alle comunicazioni broadcast, sorge spesso l’esigenza di stabilire comunicazioni punto-a-punto. Meshtastic risponde a questa necessità attraverso i **messaggi diretti** (DM).

Anche in questa circostanza, per raggiungere la destinazione viene inizialmente impiegato l’approccio di *flooding*. Tuttavia, il sistema tiene traccia dei nodi che inoltrano il pacchetto. Se la consegna va a buon fine e il mittente riceve un *ACK* di risposta, e se il nodo che ha inoltrato la risposta coincide con quello che aveva inoltrato il messaggio originale, allora questo nodo viene designato come *next-hop* per le comunicazioni future verso quella destinazione. In questo modo, il mittente può evitare di utilizzare il flooding per i messaggi successivi, instradandoli direttamente verso il nodo identificato come *next-hop*. Ciò consente di ridurre il traffico di rete, ottimizzando la comunicazione.

Tuttavia, in presenza di collegamenti asimmetrici (ad esempio, quando il nodo A può ricevere B ma non viceversa), oppure se il nodo intermedio non supporta questa funzionalità avanzata, il sistema torna automaticamente al flooding gestito.

Considerando la natura dinamica delle reti mesh LoRa, in cui i nodi possono spostarsi o disattivarsi (ad esempio per esaurimento della batteria), un *next-hop* precedentemente valido potrebbe non essere più disponibile.

In questi casi, se un nodo tenta di inoltrare un pacchetto al *next-hop* senza ricevere un *ACK* dopo l'ultimo tentativo di ritrasmissione, il sistema ritorna al flooding gestito, assicurando così la resilienza della rete.

## 4.4 Sicurezza

Fin dalle prime versioni, Meshtastic ha cercato di conciliare le esigenze di sicurezza con le limitazioni hardware tipiche delle piattaforme a basso consumo.

Il sistema di sicurezza originale si basava esclusivamente su **chiavi pre-condivise (PSK)**, identiche per tutti i membri di un canale. Sebbene tale approccio garantisse una protezione di base, non era in grado di assicurare la **riservatezza dei messaggi diretti**.

Con il rilascio della versione firmware 2.5, Meshtastic ha introdotto una svolta significativa: l'integrazione della **crittografia a chiave pubblica (PKC)** per i messaggi diretti e per l'amministrazione remota.

### 4.4.1 Crittografia nei canali di comunicazione

Per avviare una comunicazione su un canale privato, Meshtastic offre due opzioni: modificare la chiave di default del canale, rendendosi invisibili ai nodi che utilizzano il canale principale, oppure istituire un nuovo canale con una chiave dedicata, condivisa unicamente tra i nodi coinvolti.

Meshtastic protegge il *payload* dei pacchetti tramite *AES-256*, utilizzando una chiave a 256 bit che offre un elevato livello di sicurezza. La cifratura avviene tramite la **chiave di canale**, condivisa tra tutti i nodi appartenenti allo stesso canale.

Meshtastic adotta la modalità operativa **Counter Mode (AES-CTR)**[21], descritta nel capitolo precedente. L'algoritmo non elabora direttamente il messaggio, ma un IV (*Initialization Vector*) che varia per ogni blocco, generando un *bitstream* che viene combinato con il testo in chiaro tramite *XOR*[16].

In Meshtastic, l'IV è costruito come segue:

- **NONCE** (96 bit), costituito da NodeID + PacketID.
- **CONTATORE** (32 bit), inizia da 0 per ciascun pacchetto e si incrementa di 1 ad ogni blocco di 16 byte.

A differenza di LoRaWAN, dove il contatore deriva dal *FCnt*, in Meshtastic il contatore interno è utilizzato solo per differenziare i blocchi all'interno dello stesso pacchetto, mentre l'unicità dell'IV è garantita dal *nonce*.

#### 4.4.2 Messaggi diretti e crittografia asimmetrica

Attualmente i messaggi diretti in Meshtastic sono instradati tramite un sistema ottimizzato, ma prima della versione 2.5 del firmware venivano cifrati attraverso la stessa **chiave pre-condivisa (PSK)** del canale. Difatti, i messaggi diretti venivano implementati come messaggi di canale standard, utilizzando il campo `to` definito nel *protobuf*. Questo implicava che tutti i membri del canale potevano potenzialmente leggere i messaggi diretti, rendendo la privacy affidata esclusivamente a un modello di **fiducia implicita**.

Per risolvere questa vulnerabilità Meshtastic ha introdotto un sistema di **crittografia a chiave pubblica** basato su **curve ellittiche** [22].

Ogni nodo, al primo avvio, genera una **coppia di chiavi univoca** (pubblica e privata) tramite l'algoritmo X25519. La chiave pubblica viene successivamente distribuita alla mesh attraverso gli annunci regolari del nodo.

Quando un nodo avvia una comunicazione attraverso un messaggio diretto, completando così lo **scambio di chiavi**, combina la propria chiave privata con quella pubblica del destinatario. Da questo processo si ottiene un **segreto condiviso**, dal quale viene derivata una **chiave di sessione** simmetrica. Tale chiave viene utilizzata per cifrare il messaggio utilizzando *AES* in modalità **CCM** (Counter with CBC-MAC), che combina:

- Cifratura in modalità CTR, che garantisce la riservatezza dei dati. Il processo di cifratura prevede l'utilizzo di un `nonce` di 4 byte, descritto nel precedente paragrafo, il quale funge da `IV` e impedisce attacchi di tipo *replay*.
- Autenticazione tramite CBC-MAC, che genera un *Message Authentication Code* (MAC) per garantire l'integrità e l'autenticità del messaggio. Questo meccanismo è concettualmente simile ad AES-CMAC, descritto nel capitolo precedente riguardante LoRaWAN.

In questo modo, i messaggi diretti sono cifrati in modalità *end-to-end*: solo il destinatario è in grado di decifrarli e può verificarne l'autenticità attraverso la firma implicita fornita dal meccanismo di MAC.

Questa versione ha apportato modifiche significative anche riguardo all'amministrazione remota. Ogni nodo amministrabile crea una **chiave di sessione temporanea** di 8 byte, valida per 300 secondi, che deve essere inclusa in ogni comando amministrativo. Questo meccanismo impedisce l'esecuzione di comandi da parte di nodi non autorizzati e protegge da *replay attack*.

#### 4.4.3 Limitazioni del modello di sicurezza

Meshtastic, benché rappresenti una tecnologia promettente per le reti mesh decentralizzate, mostra dei limiti nel suo **modello di sicurezza** quando confrontato con protocolli crittografici consolidati quali WPA3 e TLS 1.3.

Nonostante tutti e tre utilizzino algoritmi robusti come *AES-256*, Meshtastic deve sottostare a diversi **compromessi**, dovuti alla sua natura: **hardware a basso costo, banda limitata e requisiti di efficienza energetica**.

Di conseguenza, Meshtastic non può garantire:

**Perfect Forward Secrecy (PFS)** La garanzia perfetta in avanti garantisce che la compromissione di una chiave di canale non consenta di decifrare le comunicazioni precedenti. Meshtastic non applica questa modalità di sicurezza, rendendosi vulnerabile agli attacchi “*Harvest now, Decrypt later*”, nei quali un attaccante cattura passivamente le comunicazioni cifrate per decifrarle successivamente, qualora la chiave del canale venga compromessa.

Questa vulnerabilità deriva dall’architettura basata su chiavi pre-condivise, che sono riutilizzate per tutte le sessioni di comunicazione all’interno dello stesso canale.

Benché *AES-256* sia attualmente considerato resistente ad attacchi quantistici, la compromissione della chiave può avvenire mediante svariati vettori di attacco: gestione impropria della chiave di canale da parte degli utenti, compromissione fisica dei dispositivi con conseguente estrazione della chiave, oppure vulnerabilità nel processo di distribuzione e condivisione della chiave tra i nodi di una sottorete logica.

**Integrità dei messaggi** Meshtastic non garantisce l’integrità dei messaggi sui canali basati su chiavi pre-condivise, incluso il canale amministrativo, poiché non implementa meccanismi di **autenticazione del mittente** né controlli di **integrità**. Questa lacuna è stata documentata nell’Issue GitHub 4030 [21], e rappresenta una vulnerabilità significativa.

La criticità deriva dall’utilizzo della modalità operativa **AES-CTR**, che fornisce esclusivamente garanzie di **riservatezza**. Un attaccante che intercetti un messaggio cifrato può sfruttare questa limitazione attraverso attacchi di *bit-flipping*: se riesce a dedurre anche solo una piccola porzione del testo in chiaro (sfruttando il contesto o pattern ricorrenti), può derivare la parte corrispondente del *keystream* generato da *AES-CTR*. Con questa informazione, l’attaccante può manipolare specifici bit del messaggio cifrato, alterandone il contenuto senza che il destinatario possa rilevarne la manomissione.

**Autenticazione del mittente** Meshtastic non implementa meccanismi di **autenticazione** all'interno dei canali, rendendo impossibile verificare che il Node ID presente nell'*header* del pacchetto corrisponda effettivamente al nodo che ha trasmesso il messaggio.

La problematica deriva dall'architettura di identificazione dei nodi, basata sugli **indirizzi MAC fisici** codificati dal produttore. Sebbene gli indirizzi MAC siano teoricamente univoci a livello globale, sono facilmente **alterabili**.

#### 4.4.4 Soluzioni suggerite

I problemi di integrità e autenticazione del mittente precedentemente descritti non si manifestano nei messaggi diretti, i quali impiegano crittografia asimmetrica e firme digitali. Questa architettura garantisce le proprietà di sicurezza attualmente assenti nei canali a chiave pre-condivisa, suggerendo teoricamente l'utilizzo esclusivo di messaggi diretti come possibile soluzione alle vulnerabilità individuate.

Tuttavia, tale approccio non risulta attuabile in tutti i contesti comunicativi. I messaggi diretti, infatti, sono limitati a comunicazioni *punto-a-punto* e non supportano la modalità di trasmissione *broadcast* o *multicast*. Inoltre, l'uso della crittografia asimmetrica comporta un notevole *overhead* computazionale, con conseguente aumento del **consumo energetico** e dei **tempi di elaborazione**, aspetti in contrasto con i vincoli tipici delle reti LPWAN.

Un'alternativa praticabile, suggerita dalla community Meshtastic, consiste nell'utilizzo di modalità operative **AEAD** (*Authenticated Encryption with Associated Data*), come ad esempio **AES-GCM**.

*GCM* combina due processi principali: la cifratura, che garantisce la riservatezza dei dati utilizzando una variazione della modalità CTR, e l'autenticazione. Durante il processo di cifratura, viene calcolato un **valore di autenticazione** utilizzando una funzione di *hash universale* chiamata GHASH, che combina i dati cifrati e, optionalmente, dei dati non cifrati, utilizzando un valore derivato dalla **chiave di cifratura**. I bit più significativi di questo risultato costituiscono un *tag* di autenticazione di 16 byte, allegato al messaggio cifrato, che consente al destinatario di verificare simultaneamente l'**integrità del contenuto** e l'**autenticità del mittente**[23].

Il principale svantaggio di *AEAD* è rappresentato dall'*overhead* di 16 byte per il *tag* di autenticazione, che può rivelarsi problematico tenendo conto dei vincoli di banda delle reti LoRa, specialmente per pacchetti di grandi dimensioni o comunicazioni dove l'integrità non è imprescindibile. Per attenuare questo impatto, è stata proposta un'implementazione **facoltativa per canale**, consentendo di attivare *AEAD* selettivamente per canali critici (come quello amministrativo), mantenendo la modalità CTR standard per canali pubblici dove l'*overhead* non è giustificato.

# Capitolo 5

## MeshCore

### 5.1 Panoramica generale

MeshCore rappresenta un'altra soluzione open-source che permette di creare reti mesh *off-grid* autonome, utilizzando la tecnologia LoRa e altri protocolli packet radio[24]. Il suo punto di forza è la sua capacità di bilanciare **semplicità d'uso e scalabilità operativa**.

La semplicità si manifesta nel fatto che l'avvio della rete non richiede configurazioni manuali complicate o ambienti di sviluppo specifici. MeshCore, infatti, supporta una vasta gamma di dispositivi LoRa e offre **binari pre-compilati**<sup>1</sup> che permettono un'installazione immediata. L'installazione del software è altrettanto semplice, il processo di *flashing*<sup>2</sup> è infatti agevolato da strumenti dedicati come Adafruit ESPTool, e una volta completato il dispositivo può subito iniziare a comunicare attraverso la rete mesh.

Per quanto riguarda invece la scalabilità, questa è garantita dalla capacità del sistema di gestire reti con un numero crescente di nodi senza compromettere le prestazioni complessive della rete mesh. MeshCore può supportare configurazioni che vanno da piccole reti domestiche a implementazioni su vasta scala, adattandosi dinamicamente alle esigenze operative.

### 5.2 Classificazione dei nodi

Un elemento distintivo di MeshCore è la stabilità complessiva del sistema, favorita da un'architettura modulare che prevede firmware specifici per ciascun ruolo all'interno della rete, distinguendo tre funzioni principali:

---

<sup>1</sup>I binari precompilati sono file eseguibili pronti all'uso che non necessitano di compilazione del codice sorgente.

<sup>2</sup>Processo di installazione del software su un dispositivo LoRa.

- **Companion**, dispositivi terminali utilizzati per inviare e ricevere messaggi, non sono coinvolti nel routing;
- **Repeater**, nodi dedicati esclusivamente all'*instradamento* dei pacchetti sulla rete;
- **Room server**, nodi che combinano capacità di memorizzazione e funzioni di instradamento.

Il ruolo di *room server* è particolarmente versatile, in quanto può essere configurato per operare simultaneamente come ripetitore, assumendo in tal caso il nome di **roompeater**. In questa modalità, svolge due compiti distinti: inoltra automaticamente i messaggi diretti tra i *companion* nel suo raggio d'azione e gestisce la memorizzazione dei messaggi destinati alle *stanze* per il recupero asincrono.

I firmware destinati ai *repeater* sono estremamente leggeri, poiché si concentrano esclusivamente sulla funzione di *forwarding* dei pacchetti, evitando l'inclusione di funzionalità superflue che potrebbero danneggiare la stabilità del sistema.

La definizione dei ruoli nella rete avviene principalmente durante la fase di *flashing*, in cui si seleziona il firmware corrispondente al ruolo desiderato. Tuttavia, MeshCore mantiene una certa flessibilità anche dopo l'installazione: alcuni ruoli possono essere modificati dinamicamente tramite un'interfaccia web accessibile via USB, comandi da console o persino da remoto via LoRa, attraverso l'applicazione mobile.

### 5.3 Funzionamento della rete

MeshCore, come Meshtastic, supporta l'**instradamento multi-hop**, il che consente ai messaggi di viaggiare attraverso più nodi intermedi, ampliando la copertura ben oltre la portata di una singola radio. Come ogni rete mesh ben progettata, entrambe le soluzioni sono completamente **de-centralizzate e auto-riparanti**, continuando a funzionare anche in caso di malfunzionamento o disconnessione di alcuni nodi.

Tuttavia, a differenza di Meshtastic, che si basa sul flooding e sull'organizzazione dei nodi in canali per gestire gruppi di comunicazione, MeshCore adotta un sistema di **routing manuale** senza l'uso di canali, consentendo di definire percorsi di instradamento precisi. Questa scelta garantisce un maggiore **controllo sul traffico** e una più elevata **affidabilità**, posizionando MeshCore in un ambito più professionale e strutturato.

A differenza dei canali di Meshtastic, pensati per la comunicazione in tempo reale, MeshCore adotta un approccio più asincrono, introducendo il concetto di **stanze**. Le stanze sono spazi virtuali in cui i messaggi vengono archiviati in modo persistente dai *room server*, rendendoli accessibili anche quando i destinatari non sono connessi. Questa architettura, basata su un meccanismo di *store-and-forward*, trasforma le stanze in vere e proprie “caselle postali” distribuite, utili per gestire comunicazioni intermittenti.

Definire percorsi precisi per ciascun messaggio garantisce che i pacchetti transitino solo attraverso i nodi selezionati, aumentando la prevedibilità e la stabilità della comunicazione e consentendo di sperimentare diverse configurazioni topologiche al fine di ottimizzare le prestazioni della rete.

Inoltre, l’architettura è stata progettata per **ridurre il consumo energetico**, consentendo ai nodi di operare autonomamente per periodi prolungati, caratteristica fondamentale in scenari *off-grid* dove l’alimentazione può essere limitata. In particolare, i nodi MeshCore non trasmettono beacon o dati di telemetria, riducendo significativamente il traffico di rete non necessario. Anche la scelta di utilizzare pacchetti di dimensioni ridotte[25] contribuisce a rendere le comunicazioni più rapide ed efficienti, migliorando ulteriormente l’autonomia complessiva del sistema.

## 5.4 Sicurezza

**Nota metodologica:** A causa della limitata disponibilità di documentazione ufficiale dettagliata sui meccanismi di sicurezza implementati in MeshCore, l’analisi seguente si basa principalmente su un’investigazione diretta del codice sorgente del progetto. Le conclusioni presentate derivano dall’interpretazione delle costanti definite e dell’organizzazione modulare del codice.

Per garantire una comunicazione affidabile, la sicurezza in MeshCore rappresenta un elemento centrale, soprattutto in scenari dove l’assenza di infrastrutture centralizzate costituisce una sfida significativa. A tal fine, il sistema impiega un’architettura stratificata con protezione **end-to-end**<sup>3</sup>, combinando algoritmi di crittografia asimmetrica per lo scambio delle chiavi e crittografia simmetrica per la protezione del payload.

---

<sup>3</sup>La crittografia end-to-end (E2EE) è un meccanismo in cui i dati sono cifrati dal mittente e decifrati solo dal destinatario finale, senza alcun accesso di terze parti [26].

### 5.4.1 Identificazione degli algoritmi crittografici

L’analisi del codice sorgente mostra strutture crittografiche che hanno le dimensioni indicate di seguito:

- **Chiavi pubbliche:** 32 byte
- **Chiavi private:** 64 byte
- **Seme per la generazione delle chiavi:** 32 byte
- **Firme digitali:** 64 byte

Questi valori sono coerenti con l’utilizzo del protocollo **Diffie-Hellman (DH)**, nella variante X25519, che è un’implementazione specifica di Elliptic Curve Diffie-Hellman (ECDH) basata sulla curva Curve25519. X25519, standardizzata nella RFC 7748, consente a due parti di derivare una chiave segreta condivisa utilizzando la propria chiave privata e la chiave pubblica dell’altra parte, evitando che il segreto venga mai trasmesso direttamente [27].

### 5.4.2 Proprietà di sicurezza

La valutazione della sicurezza di un sistema di comunicazione richiede l’analisi delle tre proprietà fondamentali che costituiscono la triade CIA (Confidentiality, Integrity, Availability).

**Riservatezza** Per la protezione del payload dei messaggi, MeshCore impiega l’algoritmo di crittografia simmetrica AES con chiave a 128 bit, in contrasto con i 256 bit adottati da Meshtastic. Attualmente il sistema utilizza la modalità operativa **ECB** (Electronic Codebook)[28] che, sebbene semplice da implementare, presenta vulnerabilità note legate all’analisi dei pattern, mettendo potenzialmente a rischio la riservatezza dei dati in presenza di contenuti ripetuti[16]. All’interno della community sono state avanzate proposte per la migrazione verso modalità più sicure, come GCM, che possono offrire una protezione più robusta contro tali problematiche.

**Integrità e autenticazione** Per garantire l’integrità dei messaggi e l’autenticità del mittente, MeshCore adotta un approccio a più livelli. In primo luogo, utilizza un **MAC** (Message Authentication Code) di 2 byte che, nonostante non fornisca un’elevata resistenza ad attacchi mirati di collisione, risulta efficace per rilevare alterazioni accidentali o tentativi di manipolazione di base.

A questo si affianca l'algoritmo di hash SHA-512 (Secure Hash Algorithm), in grado di generare un **digest** di 512 bit che rappresenta l'“impronta digitale” del messaggio, rendendo immediatamente rilevabile qualsiasi modifica, anche la più piccola.

Infine, viene utilizzato un algoritmo di **firma digitale** EdDSA sulla curva Ed25519, compatibile con l'impiego di X25519 per lo scambio di chiavi, che consente di verificare l'autenticità dell'origine del messaggio.

La combinazione di questi tre meccanismi fornisce una forte protezione contro tentativi di falsificazione e manipolazione dei messaggi.

# Capitolo 6

## Implementazione e test

### 6.1 Introduzione

In questo capitolo viene descritta la realizzazione pratica di un sistema di monitoraggio basato su Meshtastic, fornendo una validazione concreta dei concetti teorici affrontati in precedenza.

L'obiettivo principale è la realizzazione di nodi sensoriali in grado di rilevare dati ambientali e di trasmetterli all'interno di una rete mesh gestita dal firmware Meshtastic. A completamento del sistema, è stata realizzata un'applicazione mobile denominata *Mesh View* che, connettendosi via Bluetooth a uno dei nodi, consente la visualizzazione in tempo reale dei dati telemetrici provenienti da tutta la rete.

Il sistema realizzato permette di validare concretamente i vantaggi di Meshtastic: decentralizzazione, indipendenza da infrastrutture esterne, semplicità di deployment e capacità di operare in ambienti dove soluzioni tradizionali sarebbero impraticabili.

### 6.2 Architettura hardware

L'architettura hardware del sistema si basa sul modulo Heltec Wireless Tracker v1.2, che integra un microcontrollore ESP32, il chip radio SX1276 operante nella banda ISM a 868 MHz e un modulo GPS per la rilevazione delle coordinate geografiche. Questo dispositivo rappresenta l'unità fondamentale della rete realizzata: più moduli possono cooperare tra loro tramite Meshtastic, dando vita a una rete mesh.

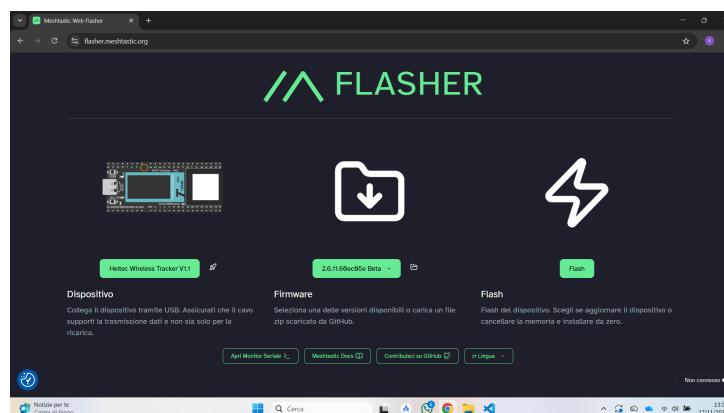
A ciascun dispositivo è stato collegato un sensore ambientale BME280, interfacciato al microcontrollore tramite bus I<sup>2</sup>C. Tale sensore consente la misura di temperatura, pressione atmosferica e umidità, fornendo dati utili per la validazione funzionale del sistema di comunicazione.

Il collegamento tra microcontrollore e modulo radio LoRa avviene tramite interfaccia SPI, gestita dal firmware Meshtastic. Tale firmware sfrutta le risorse dell'ESP32 per controllare sia la logica di rete sia la gestione dei dati sensoriali, integrando la comunicazione LoRa con l'interfaccia Bluetooth Low Energy. Quest'ultima viene impiegata per connettere il nodo principale all'applicazione mobile *Mesh View*.

Poiché il modulo non dispone di un sistema di alimentazione autonomo, ciascun nodo è stato equipaggiato con una batteria ai polimeri di litio (Li-Po) collegata al connettore di alimentazione dedicato della scheda. La batteria può essere ricaricata direttamente tramite la porta USB del modulo. Questa soluzione garantisce un'autonomia sufficiente per le sessioni di test e consente di valutare l'efficienza energetica del sistema in condizioni operative reali, aspetto fondamentale per le applicazioni off-grid.

### 6.3 Configurazione del firmware

L'installazione del firmware è stata eseguita mediante il *Meshtastic Web Flasher*, uno strumento ufficiale accessibile da browser che consente di caricare il firmware sui dispositivi in modo semplice e affidabile. Una volta collegato ciascun Heltec Wireless Tracker al computer tramite porta USB, il modello esatto dell'hardware è stato selezionato manualmente all'interno dell'interfaccia del flasher. In base a questa scelta, il tool ha proposto automaticamente la versione più recente del firmware ritenuta stabile e compatibile.



**Figura 6.1:** Meshtastic web flasher

Dopo l'avvio della procedura, l'immagine è stata trasferita sul microcontrollore ESP32-S3 e il flasher ha verificato la corretta installazione del sistema, segnalando eventuali errori o anomalie.

Terminata l'installazione, ogni nodo è stato configurato attraverso l'app ufficiale Meshtastic, che consente di gestire tutte le impostazioni operative del dispositivo mediante collegamento Bluetooth Low Energy.



**Figura 6.2:** Schermata delle impostazioni dell'app mobile Meshtastic

La prima fase della configurazione ha riguardato l'impostazione dei parametri LoRa. È stata innanzitutto selezionata la regione operativa, impostata su EU868, necessaria per abilitare i profili conformi alla normativa europea e garantire l'utilizzo corretto delle frequenze e dei limiti di potenza previsti per la banda a 868 MHz. Successivamente, i due nodi sono stati configurati in modo uniforme, selezionando uno *Spreading Factor* pari a 10, una larghezza di banda di 125 kHz e un tasso di codifica pari a 4/5.

La scelta di tali parametri è motivata da quanto discusso nel paragrafo 2.2.1, dove viene illustrato come l'aumento dello *Spreading Factor* migliori la portata del segnale a scapito della velocità di trasmissione.

La configurazione è proseguita con l'impostazione dell'identità dei nodi. A entrambi i dispositivi sono stati assegnati nomi univoci per facilitarne il riconoscimento all'interno della rete mesh. Al nodo denominato *Node1* è stato attribuito il ruolo di *tracker*, mentre il secondo dispositivo (*Node2*) è stato configurato come *sensore*, mantenendo attive le funzionalità di acquisizione ambientale previste dal firmware.

Successivamente è stata configurata la sezione dedicata ai canali, assegnando entrambi i dispositivi al medesimo canale primario denominato *TestNet*. A questo canale è stata associata una chiave precondivisa (PSK) a 128 bit, necessaria per la cifratura e la decodifica dei pacchetti scambiati nella rete. Tale configurazione ha garantito che i due nodi appartenessero alla stessa rete logica e potessero instradare correttamente i pacchetti sulla rete mesh.

Infine, sono state configurate le funzionalità legate alla posizione e alla telemetria ambientale, componenti centrali del progetto. Il modulo GPS integrato è stato attivato tramite l'app, impostando un intervallo di campionamento di 15 minuti per la trasmissione delle coordinate del dispositivo. L'applicazione Meshtastic include inoltre una sezione dedicata all'attivazione e al controllo dei sensori supportati dal firmware: attraverso questa interfaccia è stata abilitata la lettura del BME280, definendo la frequenza con cui anche temperatura, umidità e pressione vengono campionate e inviate sulla rete.

## 6.4 MeshView

L'applicazione *MeshView* costituisce il principale strumento di interfaccia tra l'utente e la rete Meshtastic. Essa svolge un duplice ruolo: da un lato gestisce la connessione Bluetooth Low Energy con i dispositivi Heltec, dall'altro si occupa della decodifica dei pacchetti ricevuti, basati sul formato Protobuf definito dal protocollo Meshtastic, per poi renderli disponibili in una forma interpretabile dall'interfaccia grafica.



**Figura 6.3:** MeshView - homepage

La home dell'app è organizzata in due sezioni principali, denominate *Network* e *Log*. La sezione *Network* rappresenta il punto di ingresso al sistema, che consente l'individuazione e la connessione dei dispositivi BLE.

Una volta stabilita la connessione con un nodo, l'app apre una schermata composta dalle viste *Map* e *Heat Map*. La prima mostra la posizione dei nodi sulla mappa e permette, cliccando sul tag associato a ciascun dispositivo, di visualizzare in tempo reale i dati ambientali provenienti dal sensore BME280. La vista *Heat Map* fornisce invece una rappresentazione spaziale delle misure ambientali, permettendo di cogliere variazioni locali di temperatura.

All'interno della sezione *Network* è inoltre presente una modalità dimostrativa, utilizzata per il test dell'interfaccia utente, che simula la presenza di otto sensori con dati generati artificialmente. Tale modalità risulta utile nelle fasi di sviluppo in assenza di nodi reali.

Oltre alla visualizzazione in tempo reale, l'applicazione integra la sezione *Log*, che si occupa della registrazione periodica delle misurazioni ricevute, con un intervallo di 30 secondi. Tale funzionalità consente di analizzare l'andamento temporale dei dati, valutarne la coerenza nel tempo e verificare la stabilità della rete durante la fase di test.

Dal punto di vista implementativo, la ricezione dei dati dai nodi avviene tramite una caratteristica BLE con capacità di notifica. Ogni volta che il dispositivo invia un pacchetto, l'app riceve una sequenza di byte che viene immediatamente convertita nella struttura `FromRadio` mediante le classi Protobuf generate, come mostrato nel Listato 6.1.

**Codice 6.1:** Ricezione dei byte dalla caratteristica BLE e decodifica iniziale

```

1 _fromRadioCharacteristic!.setNotifyValue(true);
2 _fromRadioCharacteristic!.lastValueStream.listen((data) {
3   if (data.isNotEmpty) _parseData(data);
4 });
5
6 void _parseData(List<int> data) {
7   if (data.isEmpty) return;
8   try {
9     final fromRadio = FromRadio.fromBuffer(data);
10    if (fromRadio.hasPacket()) {
11      _handleMeshPacket(fromRadio.packet);
12    }
13  } catch (e) {
14    print('Errore parsing dati: $e');
15  }
16 }
```

Il messaggio viene quindi analizzato per determinare il tipo di informazione ricevuta e, nel caso si tratti di dati telemetrici, decodificato nella struttura `Telemetry` come illustrato nel Listato 6.2.

**Codice 6.2:** Parsing dei pacchetti telemetrici mediante Protobuf

```

1 void _handleMeshPacket(MeshPacket packet) {
2   int nodeId = packet.from;
3   int? rssi = packet.hasRxRssi() ? packet.rxRssi : null;
4   int? snr = packet.hasRxSnr() ? packet.rxSnr.toInt() : null;
5
6   if (packet.hasDecoded()) {
7     Data decoded = packet.decoded;
8
9     if (decoded.portnum == PortNum.TELEMETRY_APP) {
10       _handleTelemetry(nodeId, decoded.payload, rssi, snr);
11     } else if (decoded.portnum == PortNum.POSITION_APP) {
12       _handlePosition(nodeId, decoded.payload);
13     }
14   }
15 }
16 }
```

```

17 void _handleTelemetry(int nodeId, List<int> payload, int? rssi,
18   int? snr) {
19   try {
20     Telemetry telemetry = Telemetry.fromBuffer(payload);
21
22     double? t, h, p;
23     if (telemetry.hasEnvironmentMetrics()) {
24       var env = telemetry.environmentMetrics;
25       t = env.temperature.toDouble();
26       h = env.relativeHumidity.toDouble();
27       p = env.barometricPressure.toDouble();
28     }
29
30     _updateNodeData(
31       nodeId: nodeId,
32       temperatura: t,
33       umidita: h,
34       pressione: p,
35       rssi: rssi,
36       snr: snr,
37     );
38   } catch (e) {
39     print('Errore telemetria: $e');
40   }
41 }
```

Il pacchetto telemetrico così decodificato viene infine trasformato in un modello dati interno e utilizzato sia per aggiornare la visualizzazione nelle *Map*, sia per la registrazione nella sezione *Log*. Questa pipeline consente di passare dal messaggio Protobuf trasmesso sulla rete LoRa ad una rappresentazione coerente e immediatamente fruibile all'interno dell'applicazione.

## 6.5 Test e risultati

Per valutare il corretto funzionamento della rete Meshtastic realizzata e verificare la capacità dei nodi di trasmettere dati ambientali e informazioni di posizione, è stata condotta una serie di test in condizioni operative reali. Gli obiettivi principali della fase sperimentale sono stati la validazione della comunicazione LoRa, l'analisi della telemetria generata dai nodi e la verifica del corretto funzionamento dell'applicazione *MeshView* nelle condizioni previste dal progetto.

Sebbene la rete possa supportarne un numero più elevato, per semplicità la fase sperimentale è stata condotta utilizzando due nodi, numero sufficiente per verificare il corretto instradamento dei pacchetti, la trasmissione dei dati telemetrici e l'interazione con l'applicazione.

La prima fase dei test è stata dedicata alla verifica dell'operatività della rete mesh e della stabilità della comunicazione LoRa tra i due nodi configurati nel canale TestNet. Un aspetto particolarmente significativo è che l'applicazione *MeshView* instaura una connessione BLE con un solo dispositivo alla volta; tuttavia, una volta stabilita questa connessione, l'app continua a ricevere anche i pacchetti provenienti dall'altro nodo della rete. Questo comportamento conferma che la comunicazione LoRa è attiva e che i pacchetti generati dal nodo non collegato via BLE vengono correttamente instradati attraverso il nodo connesso, raggiungendo l'applicazione in modalità trasparente. Tale meccanismo è coerente con il comportamento previsto dal protocollo Meshtastic, basato su un instradamento semplificato (flooding controllato) dei pacchetti nella rete mesh.

Per approfondire l'analisi del traffico scambiato nella rete mesh, è stato esaminato il comportamento dei nodi durante la trasmissione della telemetria ambientale. Per farlo, oltre ai test condotti tramite l'applicazione *MeshView*, è stata effettuata una verifica diretta dei pacchetti utilizzando il comando `meshtastic --listen`, eseguito sul terminale del computer a cui è stato collegato via USB uno dei nodi. Questo test, indipendente dall'app mobile, consente di osservare i messaggi Protobuf ricevuti via LoRa e di convertirli successivamente in JSON per un'analisi più approfondita.

All'avvio del comando, vengono mostrati i dati identificativi dei nodi presenti nella rete, come mostrato di seguito.

**Codice 6.3:** Output iniziale del comando `meshtastic --listen`

```

1 Connected to radio
2 Received nodeinfo: {
3   num: 3680007980
4   user.long_name: "Node1"
5   role: TRACKER
6 }
7 Received nodeinfo: {
8   num: 2666025568
9   user.long_name: "Node2"
10  role: SENSOR
11 }
```

Successivamente il sistema ha iniziato a mostrare i pacchetti telemetrici trasmessi dal nodo Node1, ossia il dispositivo collegato via USB al computer. Ciò ha permesso di osservare in modo diretto il contenuto dei messaggi ambientali inviati sulla rete mesh. Un esempio reale di pacchetto telemetrico ricevuto è riportato nel Listato 6.4.

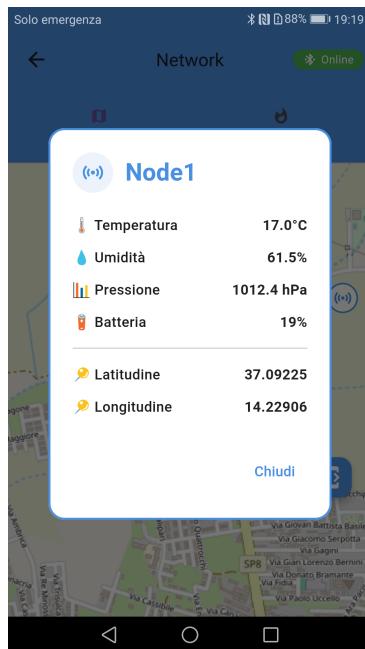
**Codice 6.4:** Estratto reale di pacchetto telemetrico ricevuto dal nodo Node1

```

1 packet {
2   from: 3680007980
3   to: 4294967295
4   decoded {
5     portnum: TELEMETRY_APP
6     payload: "\rK\000\000\000\032\017\r\303\365tA\025\000q\200B
7       \035\351\010}D"
8     telemetry {
9       time: 75
10      environment_metrics {
11        temperature: 15.31
12        relative_humidity: 64.2207
13        barometric_pressure: 1012.1392
14      }
15    }
16    id: 3385330013
17    hop_limit: 3
18    priority: BACKGROUND
19  }

```

Oltre ai dati ambientali, è stata verificata la trasmissione delle coordinate GPS. Il modulo GNSS integrato nel dispositivo è stato attivato tramite l'app Meshtastic, con un intervallo di invio impostato a 15 minuti. L'app *MeshView* è stata in grado di visualizzare correttamente la posizione del nodo, riportandola sia in forma numerica che come marker sulla mappa, come mostrato in Figura 6.4. La qualità del fix satellitare è risultata conforme alle aspettative: stabile in condizioni outdoor e più incerta in ambienti interni, a causa dell'attenuazione naturale del segnale GNSS.



**Figura 6.4:** Visualizzazione dei dati del nodo nella vista *Map* dell'app

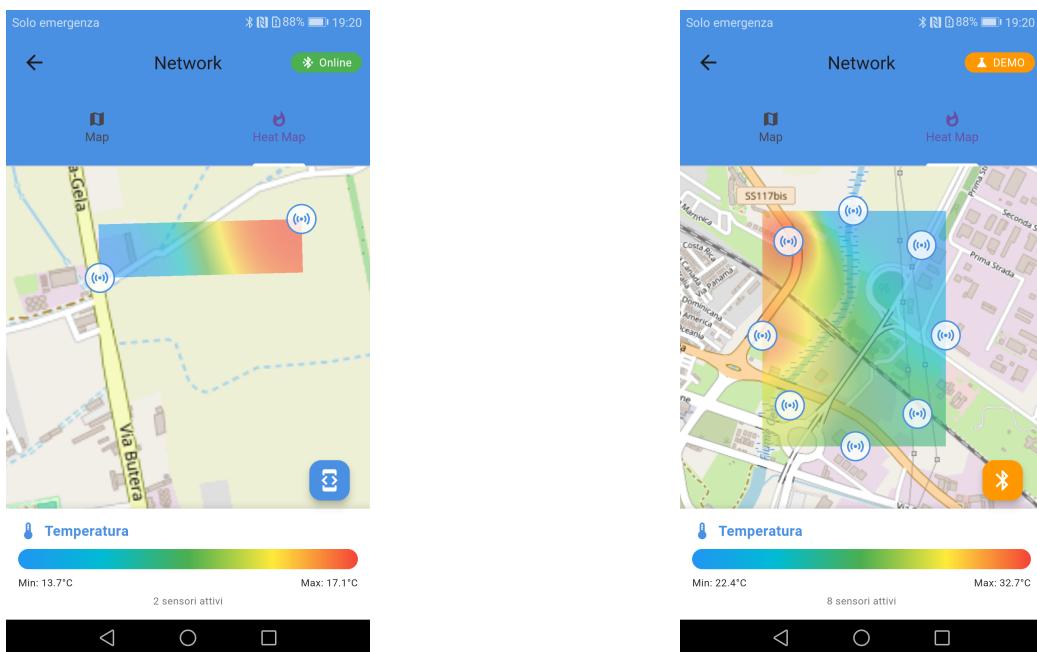
Un esempio reale di pacchetto di posizione, osservato all'interno dello stesso flusso di ricezione, è riportato nel Listato 6.5.

**Codice 6.5:** Esempio di pacchetto di posizione GPS ricevuto dal nodo Node2

```

1 packet {
2   from: 2666025568
3   decoded {
4     portnum: POSITION_APP
5     position {
6       latitude_i: 370933760
7       longitude_i: 142344192
8       altitude: 26
9       sats_in_view: 7
10      pdop: 500
11      timestamp: 1763746891
12    }
13  }
14  rx_rssi: -22
15  rx_snr: 8.25
16  id: 872037587
17 }
```

La vista *Heat Map* ha permesso di rappresentare graficamente la distribuzione spaziale delle misurazioni ambientali. Come mostrato in Figura 6.5, la mappa generata durante il test reale riflette un numero limitato di punti di misura, corrispondenti ai due nodi disponibili. Per evidenziare il comportamento completo dell'algoritmo di interpolazione, è stata inoltre utilizzata la modalità demo dell'applicazione, che consente di osservare in modo più chiaro la distribuzione cromatica.



**Figura 6.5:** Confronto tra la Heat Map reale e quella ottenuta in modalità demo

La sezione *Log*, mostrata in Figura 6.6, ha registrato automaticamente un campione ogni 30 secondi, consentendo di analizzare l'evoluzione temporale dei valori ambientali e di verificare la continuità della trasmissione dei pacchetti quando il nodo risulta connesso. L'interfaccia dell'applicazione consente inoltre di filtrare i log per mese, rendendo più agevole la consultazione dei dati raccolti in periodi di test estesi e permettendo di isolare rapidamente le misurazioni relative a specifiche sessioni sperimentali.



**Figura 6.6:** Schermata della sezione *Log* dell'applicazione MeshView

L’analisi dei parametri radio LoRa è stata condotta a partire dagli stessi pacchetti raccolti durante il test. Ogni pacchetto ricevuto include i campi `rx_rssi` e `rx_snr`, illustrati nell’estratto riportato nel Listato 6.5. Durante l’intera sessione di test, l’RSSI medio è risultato pari a  $-26$  dBm, mentre lo SNR si è attestato intorno a 8 dB. L’RSSI evidenzia un segnale molto forte, coerente con la breve distanza tra i dispositivi, mentre lo SNR denota un buon rapporto segnale-rumore. Tali valori si collocano perfettamente all’interno delle considerazioni teoriche discusse nel capitolo 2.2.1, secondo cui spreading factor elevati permettono di ricevere correttamente pacchetti anche in presenza di un livello di rumore significativo.

Nel complesso, i test hanno confermato la stabilità della comunicazione LoRa, il corretto funzionamento della pipeline di parsing dell’applicazione e la coerenza delle misure telemetriche ottenute. Pur con un numero limitato di dispositivi, la rete ha mostrato una buona affidabilità operativa, validando l’integrazione tra i livelli hardware, firmware e software del sistema.

# Conclusione

Esaminando i tre casi analizzati, emerge chiaramente il contrasto tra l'architettura centralizzata di LoRaWAN e le soluzioni mesh decentralizzate, rappresentate da Meshtastic e MeshCore.

L'architettura centralizzata di LoRaWAN offre vantaggi significativi in termini di gestione delle risorse, controllo della qualità del servizio e scalabilità prevedibile, rendendola particolarmente indicata a scenari IoT su larga scala, dove sono richiesti elevati standard di affidabilità e prestazioni. Tuttavia, la centralizzazione introduce anche criticità, legate alla dipendenza dai gateway e dal Network Server, esponendola al rischio di SPOF (Single Point Of Failure), a costi infrastrutturali più elevati e a una minore flessibilità nella gestione della rete.

Al contrario, le reti mesh decentralizzate, come Meshtastic e MeshCore, privilegiano resilienza e autonomia operativa. La loro abilità di adattarsi in modo dinamico ai cambiamenti topologici garantisce continuità del servizio anche in caso di malfunzionamenti di alcuni nodi, rendendole ideali in contesti remoti o off-grid. Tuttavia, la natura decentralizzata implica una maggiore complessità nel routing, con prestazioni meno prevedibili e una qualità del servizio più variabile. Inoltre, la gestione di reti di grandi dimensioni può richiedere strategie avanzate per prevenire congestioni e inefficienze.

In definitiva, le tre soluzioni analizzate evidenziano come non esista un'unica tecnologia ottimale, ma solo la più adeguata al contesto applicativo.

La parte progettuale ha fornito una validazione concreta delle caratteristiche operative della rete Meshtastic. I nodi hanno infatti mostrato una buona capacità di mantenere la connettività mesh in condizioni off-grid, con trasmissione regolare dei pacchetti. L'applicazione MeshView ha permesso di monitorare in tempo reale posizione e misure ambientali, evidenziando il corretto funzionamento dell'intera pipeline, dalla generazione del dato alla sua visualizzazione. Sono emerse tuttavia alcune criticità tipiche delle reti LoRa mesh, tra cui latenze variabili, perdita occasionale dei pacchetti e aggiornamenti non costanti dei dati di posizione.

Proprio a partire da questi risultati, sono emerse diverse possibilità di

evoluzione dell'app MeshView. Un primo sviluppo riguarda l'introduzione della visualizzazione storica della telemetria attraverso grafici interattivi, il salvataggio in cloud dei pacchetti ricevuti e strumenti per analizzare l'affidabilità del collegamento tra nodi, come la percentuale di pacchetti persi, il tempo medio di consegna o l'andamento del segnale nel tempo. Un secondo possibile miglioramento riguarda l'interfaccia utente, che potrebbe essere ottimizzata introducendo funzionalità come la ricerca dei nodi sulla mappa o la visualizzazione dei percorsi multi-hop utilizzati per raggiungere un determinato nodo.

Una considerazione finale riguarda la tecnologia LoRa nel suo insieme. Essa si conferma come una delle soluzioni LPWAN più versatili, in grado di bilanciare consumi energetici ridotti, costi contenuti e copertura a lunga distanza. Tuttavia, questo equilibrio porta con sé alcuni compromessi, come la velocità limitata, restrizioni sulle dimensioni dei pacchetti e la vulnerabilità alle interferenze radio, oltre alle normative che regolano le bande ISM. Pur con questi vincoli, LoRa risulta perfettamente adeguata ai contesti per cui è stata concepita. Guardando al futuro, l'ottimizzazione dei parametri di modulazione potrà ridurre alcune delle limitazioni attuali, migliorando l'equilibrio tra velocità e portata. Allo stesso tempo, una maggiore standardizzazione delle implementazioni, ancora carente nelle soluzioni mesh esaminate, potrebbe agevolare l'interoperabilità, aumentando la compatibilità tra reti diverse e ampliando le possibilità applicative della tecnologia.

# Bibliografia

- [1] Low-power wide-area networks 101. <https://docs.arduino.cc/learn/communication/low-power-wide-area-networks-101/>.
- [2] What is chirp spread spectrum (css). <https://tektelic.com/what-it-is/chirp-spread-spectrum/>.
- [3] Secure key management in lorawan: A review. <https://ieeexplore.ieee.org/document/8808835>.
- [4] What is lorawan? <https://www.thethingsnetwork.org/docs/lorawan>.
- [5] Sensor design conversion process white paper. [https://info.semtech.com/hubfs/Sensor%20Design%20Conversion%20Process%20WhitePaper\\_Final.pdf?hsLang=en-it](https://info.semtech.com/hubfs/Sensor%20Design%20Conversion%20Process%20WhitePaper_Final.pdf?hsLang=en-it).
- [6] An1200.86 – lora® and lorawan® version 1.0. <https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf>.
- [7] Low power wide area networks: An overview. <https://arxiv.org/pdf/1606.07360.pdf>.
- [8] A survey on lpwa technology: Lora and nb-iot. <https://www.sciencedirect.com/science/article/pii/S2405959517300061>.
- [9] Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios. <https://www.mdpi.com/1424-8220/18/11/3995>.
- [10] Lorawan technology for smart environments: A comprehensive review. <https://www.sciencedirect.com/science/article/pii/S2542660523003761>.

- [11] Lorawan® link-layer specification, version 1.0.4. <https://lora-alliance.org/wp-content/uploads/2021/11/LoRaWAN-Link-Layer-Specification-v1.0.4.pdf>.
- [12] Lorawan® 1.1 specification — activation by personalization (abp). <https://learn.semtech.com/mod/book/view.php?id=171&chapterid=89>.
- [13] Lorawan® security whitepaper. <http://resources.lora-alliance.org/security/lorawan-security-whitepaper>.
- [14] Advanced encryption standard (aes). <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>.
- [15] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [16] Recommendation for block cipher modes of operation: Methods and techniques. <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38a.pdf>.
- [17] Recommendation for block cipher modes of operation: The cmac mode for authentication. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38b.pdf>.
- [18] Wireless mesh network — wikipedia, the free encyclopedia. [https://it.wikipedia.org/wiki/Wireless\\_mesh\\_network](https://it.wikipedia.org/wiki/Wireless_mesh_network).
- [19] Meshtastic documentation. <https://meshtastic.org/docs/introduction>.
- [20] Meshtastic: A decentralized wireless mesh network based on lora technology. <https://www.mdpi.com/2079-9292/13/6/1055>.
- [21] Bug, missing integrity checks. <https://github.com/meshtastic/firmware/issues/4030>.
- [22] Meshtastic encryption: Evolving from simple messaging to a versatile solution. [https://meshtastic.org/blog/introducing-new-public-key-cryptography-in-v2\\_5/](https://meshtastic.org/blog/introducing-new-public-key-cryptography-in-v2_5/).
- [23] Recommendation for block cipher modes of operation: Galois/counter mode (gcm) and gmac. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>.

- [24] Meshcore: A lightweight network communication framework. <https://github.com/meshcore-dev/MeshCore>.
- [25] Mesh 26 routing – so why try meshcore now? [https://www.reddit.com/r/meshtastic/comments/liz0qwq/mesh\\_26\\_routing\\_so\\_why\\_try\\_meshcore\\_now/](https://www.reddit.com/r/meshtastic/comments/liz0qwq/mesh_26_routing_so_why_try_meshcore_now/).
- [26] End-to-end encryption. [https://ssd.eff.org/glossary/end-to-end-encryption?](https://ssd.eff.org/glossary/end-to-end-encryption)
- [27] Elliptic curves for security. <https://www.rfc-editor.org/rfc/rfc7748>.
- [28] Security issues in encryption. <https://github.com/meshcore-dev/MeshCore/issues/259>.