Cenni storici su meshtastic in Italia e mio coinvolgimento

La prima apparizione, con tanto di gruppo Telegram, risale al Dicembre 2020 dove mi aggrego ai 14 iscritti al gruppo di allora (oggi sono oltre 1000). Il fondatore del gruppo era (ed è) Andrea IU1IVA di Torino e tra i primi iscritti risultavano anche Manuel IZ1KGA e IK1JNS tutti torinesi. A quel tempo già conoscevo i controller ESP32 su cui erano basati i devices Lora in uso meshtastic e dato che avevo già allestito un server su AWS per il controllo remoto di ESP32 che inviavano dati telemetrici di temperatura pressione e umidità via internet, la prima cosa che mi venne in mente fu quella di mettere in piedi un sistema di controllo dei messaggi di protocollo e testuali ricevuti basato su collegamento seriale e applicazione python su PC.

Nacque allora https://github.com/vinloren/meshtastic_broadcast che, dopo la serie di aggiornamenti apportati d'allora, oggi prevede oltre la risposta automatica a messaggi contenenti qsl? e l'invio a discrezione di messaggi periodici con cadenza ogni 10 minuti, vede aggiunta la funzione di invio dati di protocollo a un server globale centrale (https://vinmqtt.hopto.org) cui farò cenno di seguito.

meshtastic_broadcast

Questa applicazione prevede già di per sé il mapping su mappa OpenStreet dei nodi visti dal device collegato sulla seriale (sia in ambiente Windows che Linux) e un embrione di tracking ma ovviamente ha limite di non essere accessibile che dal PC da cui è stata lanciata. E' anche vero che loraitalia ha avuto sviluppi notevoli nel 2023-2024 per cui il servizio di mapping e tracking (oltre ad altre chicche sulle statistiche e sulla telemetria) sono disponibili per chiunque abbia accesso alla rete, ma essendo orientato al controllo di una rete propria, ho deciso di optare per la messa in linea di un server dedicato al supporto di funzioni similari ma totalmente sotto il controllo mio e degli iscritti al gruppo Whatsapp che fa riferimento alla nostra rete circoscritta.

Server globale

Nel Marzo 2025 nasce così https://vinmqtt.hopto.org che è un python flask https server (non tragga in inganno mqtt che sta nel nome che ho lasciato com'era al tempo della gestione ESP32 per telemetria che appunto lavorava in mqtt) che riceve in input i dati trasmessi dai meshtastic_broadcast che colleghi della nostra rete hanno installato in proprio. La cosa ha funzionato così fino a Luglio 2025 quando mi sono reso conto che sarebbe stato necessario un aggiornamento all'approccio dell'input dati meshtastic via applicazione suddetta. L'installazione della stessa, segnatamente all'ambiente Linux, risultava lunga e laboriosa disincentivando così la sua diffusione, inoltre la stessa, in rapporto alla semplice funzione di raccogliere e inviare dati a server globale era inutilmente ridondante. Nasce così l'idea che ha portato a https://github.com/vinloren/meshtastic_meshmonitor.

meshtastic_meshmonitor

E' costituito da due componenti che accedono in comune ad un unico Data Base Sqlite3:

1. mesh_controller.py applicaione python d'interfaccia col device Tlora collegato in seriale
2. Flask server locale per accedere al Db e prelevare i dati atti alla rappresentazione in mappa dei nodi visti dal Tlora d'interfaccia oltre che al tracciamento dei percorsi dei nodi mobili.

mesh_controller.py

Attraverso lo stesso codice emendato usato da broadcast mesh_controller.py si prende cura di rilevare e gestire tutti i messaggi di:

1. NODEINFO
2. POSITION
3. TELEMETRY
4. TEXT_APP per risposta automatica messaggio con dentro qsl? e/o accumulo messaggi della giornata in tabella 'messaggi'

per registrarne il contenuto in tabella meshnodes di DB Sqlite3:

```
CREATE TABLE "meshnodes" (
	"data"	VARCHAR(8) NOT NULL,
	"ora"	VARCHAR(8) NOT NULL,
	"nodenum"	INTEGER NOT NULL,
	"node_id"	VARCHAR(9) NOT NULL,
	"longname"	VARCHAR(28) NOT NULL,
	"alt"	INTEGER,
	"lat"	FLOAT,
	"lon"	FLOAT,
	"batt"	INTEGER,
	"snr"	FLOAT,
	"pressione"	FLOAT,
	"temperat"	FLOAT,
	"umidita"	FLOAT,
	PRIMARY KEY("nodenum")
)
```

onde caratterizzare ciascun nodo rilevato in rete e poterne fissare la posizione in mappa con data e ora della sua ultima presenza. Per riempire i vari campi di questa tabella vengono considerati i messaggi di NODEINFO, POSITION e TELEMETRY via via ricevuti. Poi per poter realizzare il tracciamento del percorso di nodi mobili abbiamo la tabella tracking:

```
CREATE TABLE "tracking" (
	"node_id"	VARCHAR(9) NOT NULL,
	"longname"	VARCHAR(28) NOT NULL,
	"data"	VARCHAR(8) NOT NULL,
	"ora"	VARCHAR(8) NOT NULL,
	"lon"	FLOAT,
	"lat"	FLOAT,
	"alt"	INTEGER,
	"batt"	INTEGER,
	"temperat"	FLOAT,
	"pressione"	FLOAT,
	"umidita"	FLOAT,
	"_id"	INTEGER NOT NULL,
	PRIMARY KEY("_id")
)
```

L'aggiornamento della tabella 'tracking' ha luogo a cura di mesh_controller.py quando un nuovo messaggio di POSITION per un determinato node_id trova l'ultima posizione registrata più lontana di

75mt rispetto l'attuale. In questo caso viene aggiunto un record in tabella tracking con la nuova posizione e contemporaneamente si aggiorna 'meshnodes' con data, ora, nuova posizione. Se la nuova posizione è meno di 75mt lontana dalla precedente viene solo aggiornata data e ora in 'meshnodes'.

Abbiamo poi la gestione dei messaggi testuali ricevuti aggiornando la tabella 'messaggi' da parte di mesh_controller.py::

```
CREATE TABLE messaggi (
        _id INTEGER NOT NULL,
        data VARCHAR(8) NOT NULL,
        ora VARCHAR(8) NOT NULL,
        msg VARCHAR(200) NOT NULL,
        PRIMARY KEY (_id)
)
```

I messaggi salvati in questa tabella saranno visualizzabili dal server flask locale come vedremo più avanti.

Autorizzazione a comparire in mappa

La nostra rete "Lombardia e Canton Ticino" lavora in modalità medium_fast e come tale vede l'intera rete loraitalia che consta di circa 250 nodi. Questa gran massa di nodi presenti creerebbe una visione caotica e priva di reale rilevanza per chi è interessato ai soli nodi iscritti alla nostra rete, quindi per superare questo problema ho previsto sul lato server locale una procedura atta a specificare uno per uno i nodi che vogliamo possano essere mostrati in mappa o in tracking. A questo proposito esiste la tabella 'modes':

```
CREATE TABLE modes (
        node_id VARCHAR(9) NOT NULL,
        nome VARCHAR(28) NOT NULL,
        freq INTEGER NOT NULL,
        mode VARCHAR(14) NOT NULL,
        PRIMARY KEY (node_id)
)
```

Tramite questa tabella il server locale sarà in grado anche di marcare con colori diversi i marker dei nodi in mappa: in rosso i nodi a 868Mhz di iscritti al gruppo Whatsapp "Lombardia e Canton Ticino", in blu i nodi scelti di essere mostrati appartenenti a non iscritti, in verde i nodi a 433Mhz.

La rappresentazione dei percorsi, curata dal Flask server locale, avrà sempre il marker di partenza in rosso, i marker intermedi lungo il percorso in verde, il marker finale in blu. Sul marker finale sarà anche riportata la misura in Km del percorso effettuato.

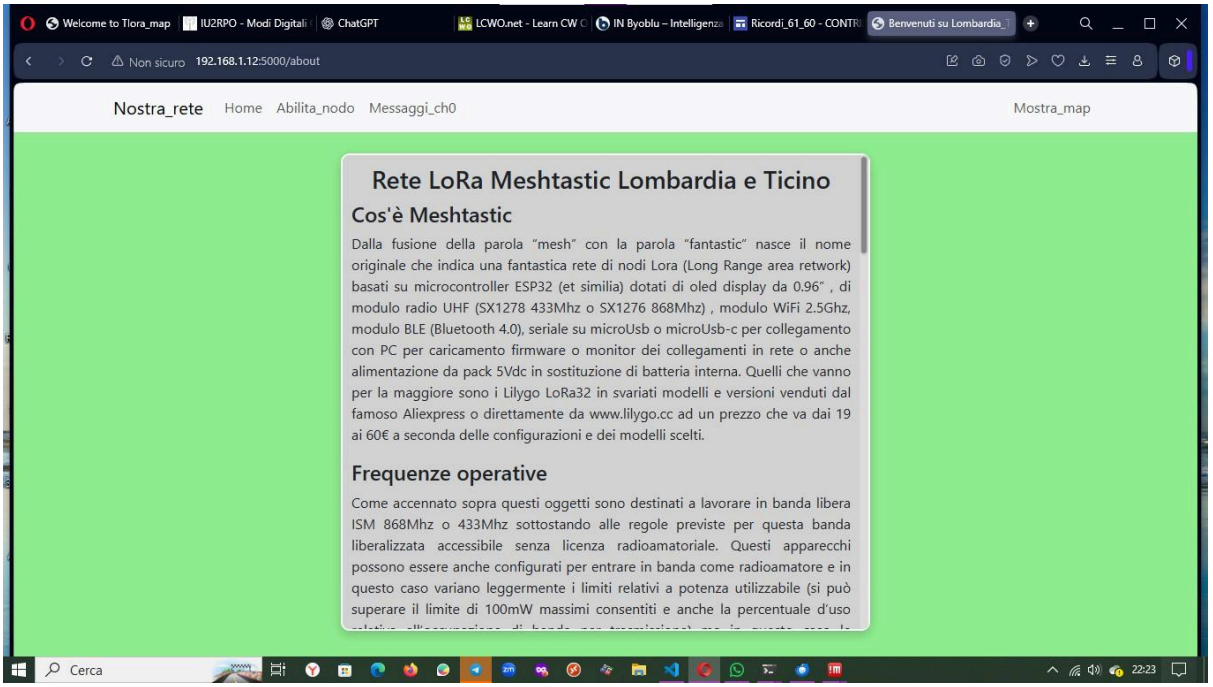
Pulizia del Db

Vengono mantenuti attivi i dati in Db per un massimo di 15gg per non ingrandire le dimensioni del Db inutilmente ritenendo che una storia dei soli ultimi 15 giorni sia rispondente alle nostre esigenze. A questo proposito ogni giorno intorno alle ore 11:10 e 18:00 mesh_controller.py provvede ad eventualmente cancellare i vecchi record. Ho previsto due orari di controllo perché un utente magari accende solo il pomeriggio e non la mattina.

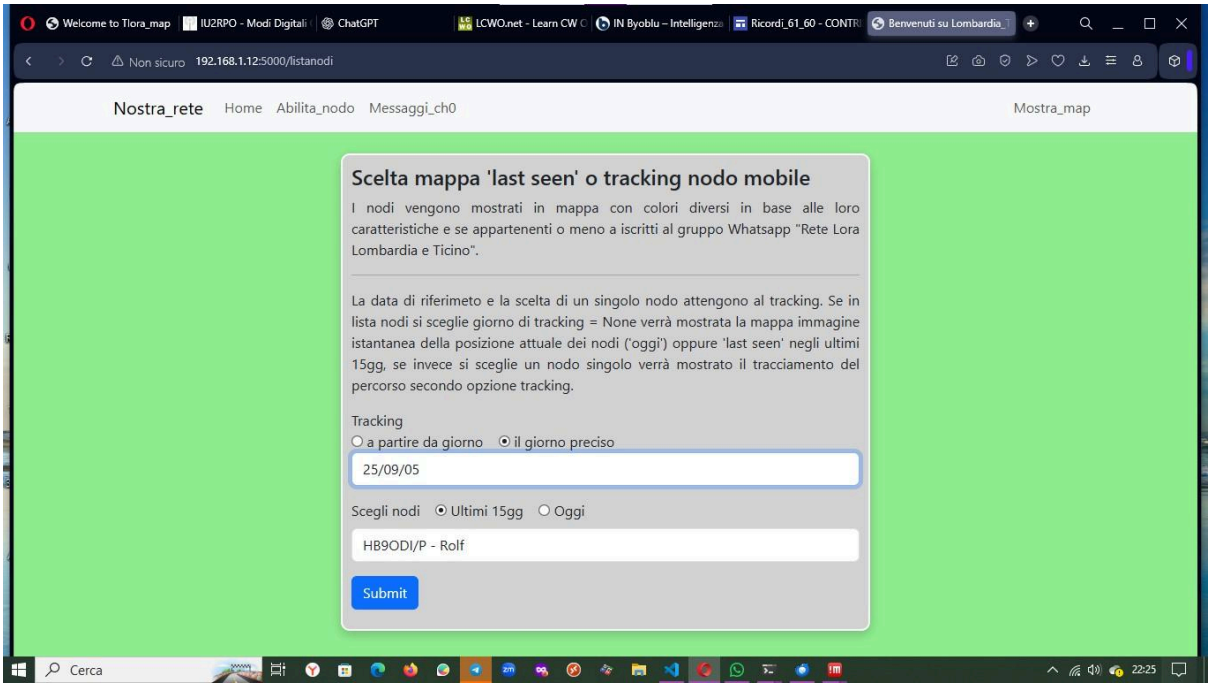
Il server locale

Come già accennato si tratta di un Python Flask server che lavora in modalità ‘sviluppo’ (non ‘produzione’) e questo perché essendo non esposto ad accessi esterni oltre quello locale non necessita di configurazioni più sofisticate che si appoggiano a Gunicorn e Nginx che sono appunto gli ambienti di ‘produzione’. In ambiente ‘sviluppo’ abbiamo poi un vantaggio accessorio che è quello di lavorare in modalità ‘debug’ che ci solleva dai problemi di gestione Flask cache / Nginx cache che altrimenti dovremmo affrontare.

Presentiamo le varie pagine di accesso al Server:

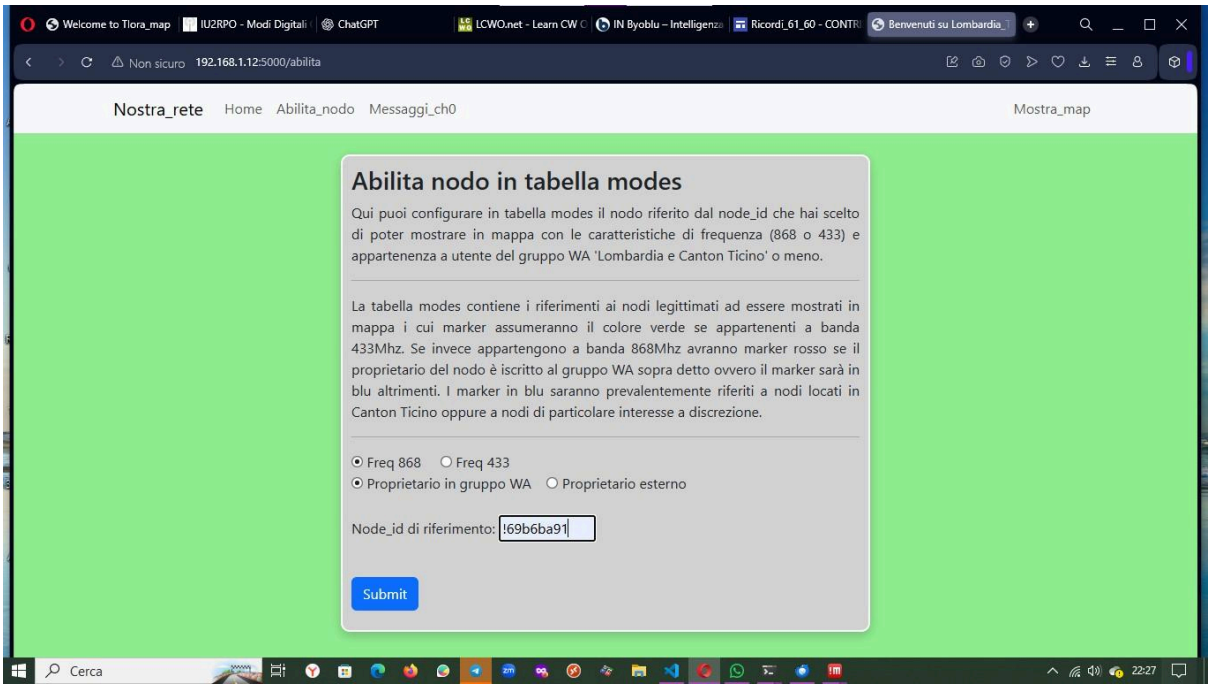


E’ la pagina d’ingresso del server dove viene fatta un po’ di storia e data qualche spiegazione.

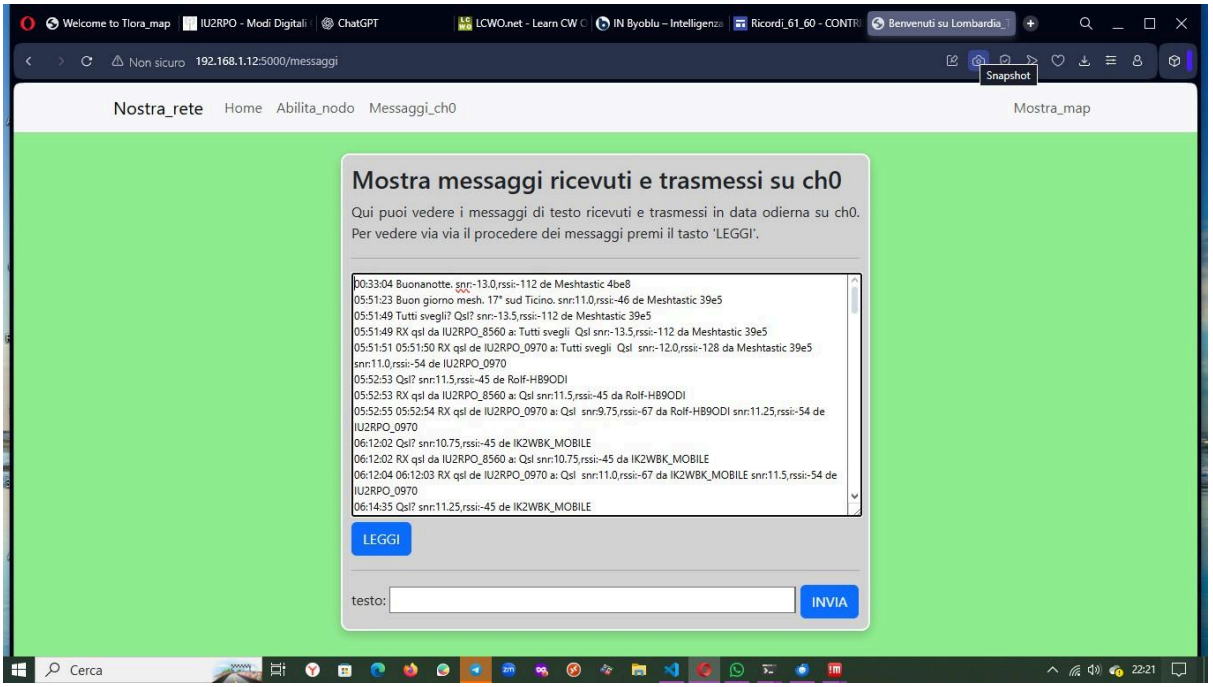


E’ la pagina Home dove possiamo scegliere di mostrare in mappa o il tracciamento di un nodo scelto nella lista ‘Scelta nodi’ dove si può riferire di tracciare un giorno preciso o tutti i tracciamenti negli ultimi 15gg, oppure si sceglie di mostrare la mappa dell’attuale collocazione dei nodi nell’arco degli ultimi 15gg oppure nella giornata di oggi. Se nel campo tracking si mette data = None verrà automaticamente considerato di mostrare lo stato della mappa attuale nel giorno di oggi o lo storico degli ultimi 15gg.

Circa il tracking osserviamo che la scelta ‘Ultimi 15gg’ serve solo a capire poi quale giorno davvero ci interessa tracciare per il nodo scelto in lista non sapendo a priori quando si è mosso in questo arco temporale.



Pagina ‘abilita_nodo’ dove impostare le caratteristiche del nodo prescelto. Si spiega da sé.



Pagina ‘Messaggi_ch0’. Premendo ‘LEGGI’ vengono mostrati i messaggi testuali ricevuti su ch0. Ad ogni depressione di ‘LEGGI’ viene mostrata l’intera storia della giornata da ore 00:00 al momento attuale. La tabella ‘messaggi’ contiene solo quelli ricevuti nella giornata in corso.

Invio messaggi

Premendo il tasto ‘INVIA’ è possibile inviare messaggi testuali sul ch0 avendo riempito il testo. Il server, ricevuto il testo, lo inserisce in tabella ‘messaggi’ preponendovi il carattere ^ che fa capire a mesh_controller.py, al termine di ciascun ciclo di ricezione (dopo esaminato il messaggio di protocollo o di testo ricevuto via seriale), che c’è un messaggio da inviare su ch0. Se ^ è presente nell’ultimo messaggio presente in tabella ‘messaggi’, questo viene inviato in rete su ch0 privato del carattere ^.

Per vedere quando il nostro messaggio è davvero partito occorre premere ‘LEGGI’ dopo premuto ‘INVIA’. L’acquisizione in tabella ‘messaggi’ del messaggio inviato ci è notificata dalla scomparsa del testo, a questo punto premendo ‘LEGGI’ vediamo in nostro messaggio con preposto ^ nell’ultima riga. Per vedere l’effettivo invio in rete dobbiamo attendere che un nuovo messaggio di protocollo faccia il suo giro in mesh_controller.py il che può prendere anche un minuto in funzione del traffico di rete. In ogni caso quando nell’ultima riga, premuto ‘LEGGI’ troviamo il nostro messaggio privato del carattere ^, sappiamo che esso è stato davvero inviato.

Ripensandoci ora, un metodo ulteriore per avere echo di ritorno a messaggio inviato da un nodo qualunque recepito dal Tlora di mesh_controller è quello di anteporre ^ al messaggio senza bisogno di qsl?. Ho visto che davvero funziona! Questo può anche essere un metodo per capire chi in rete sta usando l’ambiente Meshtastic_meshmonitor.

Innovazioni Ver. 1.1 (15 Sett. 2025)

1. Con l’obiettivo di allargare la visuale sulla rete meshtastic vista da più punti diversi è stato introdotto accesso al server MQTT pubblico broker.emqx.io da parte del sistema meshtastic_meshmonitor. Sarà così possibile mostrare la mappa aggiuntiva mqtt_map.
2. con l’obiettivo di permettere all’utente di avere contezza degli eventi registrati nel log ./logs/Server.log, ora questi saranno visibili a richiesta attraverso un nuova pagina html dedicata sul server. L’utilità principale di visualizzare il log degli eventi sta nel vedere quanti e quali nodi sono visibili al nostro nodo mesh_controller oltre a quelli autorizzati ad apparire in mappa e quindi poter decidere di autorizzare visibilità aggiuntive a discrezione.

Realizzazione obiettivo mqtt_map

1. viene creata una nuova tabella ‘loranodes’ in Db analoga alla ‘meshnodes’ già esistente:

```
CREATE TABLE "loranodes" (
        "data"      VARCHAR(8) NOT NULL,
        "ora"       VARCHAR(8) NOT NULL,
        "node_id" VARCHAR(9) NOT NULL UNIQUE,
        "longname" VARCHAR(28) NOT NULL,
        "alt"       INTEGER,
        "lat"       FLOAT,
        "lon"       FLOAT,
        "batt"      INTEGER,
        "snr"       FLOAT,
        "pressione" FLOAT,
```

```
        "temperat" FLOAT,
        "umidita"  FLOAT,
        PRIMARY KEY("node_id")
)
```

Si è preferito inserire una tabella aggiuntiva per distinguere la mappa di base della nostra rete vista dal nostro punto di vista rispetto la mappa della nostra o altre reti in riferimento a ciò che ci perviene dal server.

2. Viene introdotta applicazione mqtt_subscribe.py per connessione al server mqtt broker.emqx.io dal quale perverranno i dati inviati da altri utenti sul topic 'meshtastic/vinloren' (default modificabile) che aggiorneranno la tabella 'loranodes'.
3. Viene introdotta applicazione mqtt_send_all.py per accedere al server broker.emqx.io sul topic 'meshtastic/vinlore (default modificabile) se vogliamo che la nostra porzione visibile di rete vada ad integrare quella vista da altri utenti.

mqtt_subscribe.py

Applicazione residente in directory ./source insieme con mesh_controller.py può essere lanciata se si ha interesse a vedere la rete da un punto di vista diverso dal nostro. Il lancio può essere eseguito indifferentemente prima o dopo quello di mesh_controller.py. I record dati ricevuti hanno la particolarità di avere, oltre i nomi dei campi che andranno in tabella, un campo aggiuntivo indicante il numero di record previsti per completare la ricezione. Al raggiungimento di questo numero i dati ricevuti andranno ad aggiornare la tabella 'loranodes'.

mqtt_send_all.py

Applicazione residente in directory ./source può essere lanciata se abbiamo interesse che altri recepiscano il nostro punto di vista. Il topic di default è fissato in 'meshtastic/vinloren' ma può essere impostato du meshtastic/xxxxxx dove xxxxx viene scelto da noi. In questo caso ovviamente occorre che altri utenti interessati a questo topic ne siano messi a conoscenza. mqtt_send.py si occupa di leggere, con periodicità di 5 minuti, la tabella 'meshnodes' (contiene una riga per nodo visto in rete dal nosto punto di visuale) e una riga alla volta provvede a inviarla al server broker.emqx.io sul topic prescelto. Per motivi di protocollo dettati da broker.emqx.io occorre che ci sia un intervallo di tempo intorno ai 200-250mS fra gli invii dei record in tabella che, per quello che concerne la mia posizione in rete, consiste in 280 nodi visibili portando il tempo di trasmissione a circa 70 secondi ogni 5 minuti.

Realizzazione obiettivo lettura log

Il server flask è dotato di un log degli eventi che hanno a che vedere con possibili problemi che si possono incontrare nei suoi processi oppure per indicare percorsi decisionali operati. A questo proposito diventa utile per l'operatore di meshtastic_meshcontroller rilevare attraverso la lettura del log node_id e nomi dei nodi presenti in 'meshnodes' e/o in 'loranodes' che non sono visibili in mappa perché non autorizzati. Può capitare che uno o più di questi nodi siano divenuti interessanti per noi e vogliamo autorizzarli; il log in questo caso ci viene in aiuto.

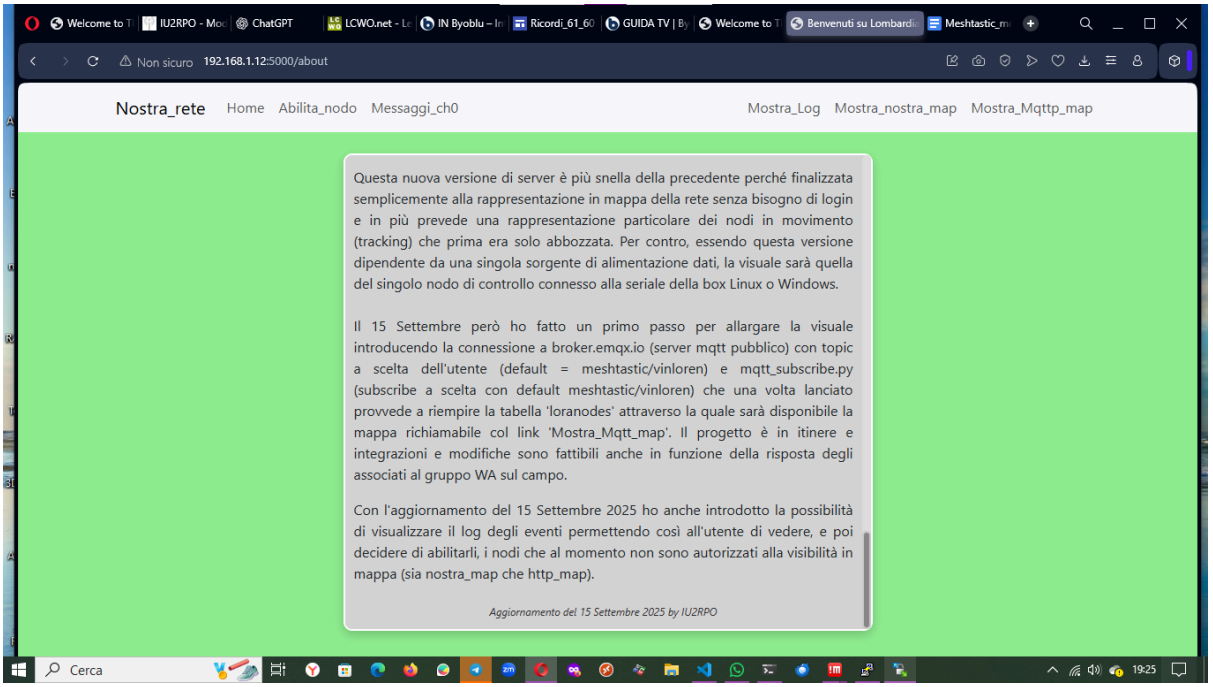
Opzioni di lancio

1. **mqtt_subscribe.py**: python mqtt_subscribe.py –channel=meshtastic/xxxxx dove xxxxx è il subtopic di nostro interesse (-- sono due trattini). Se si lancia semplicemente python mqtt_subscribe.py ci si aggancia al topic di default che è meshtastic/vinloren

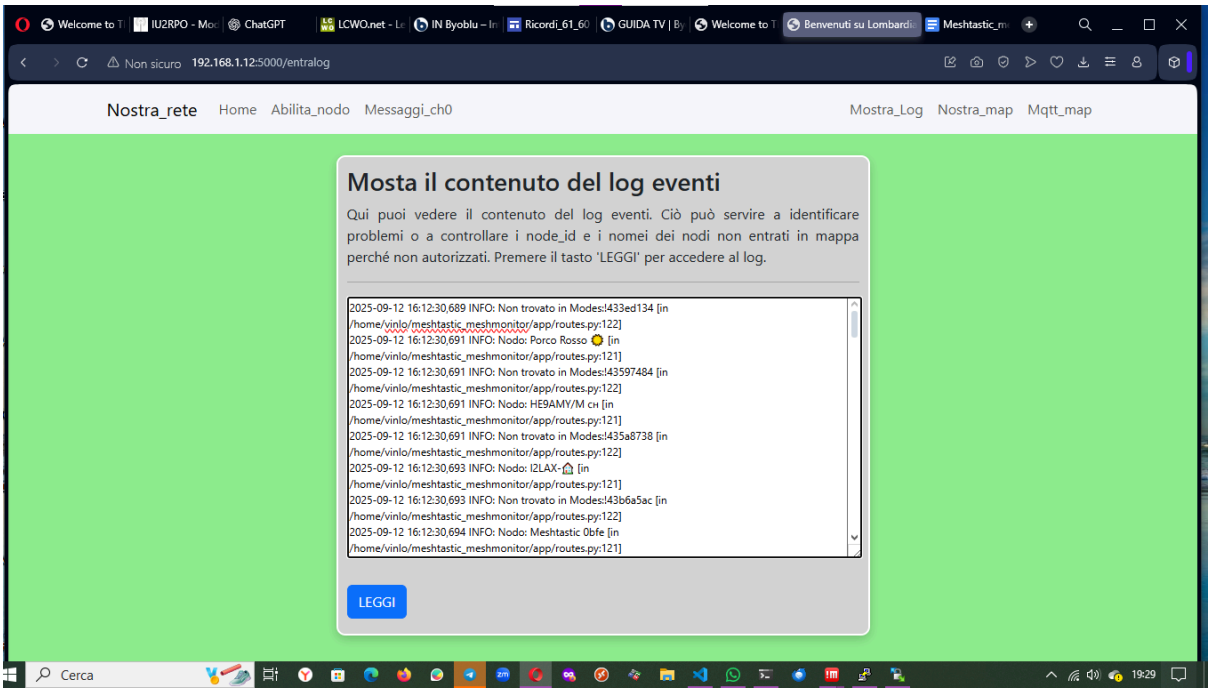
2. **mqtt_send_all.py**: python mqtt_send_all.py xxxx dove xxxx produrrà il topic meshtastic/xxxx se si vuole usarlo al posto del default meshtastic/vinloren che si ottiene lanciando senza parametri

Screen shots nuove pagine

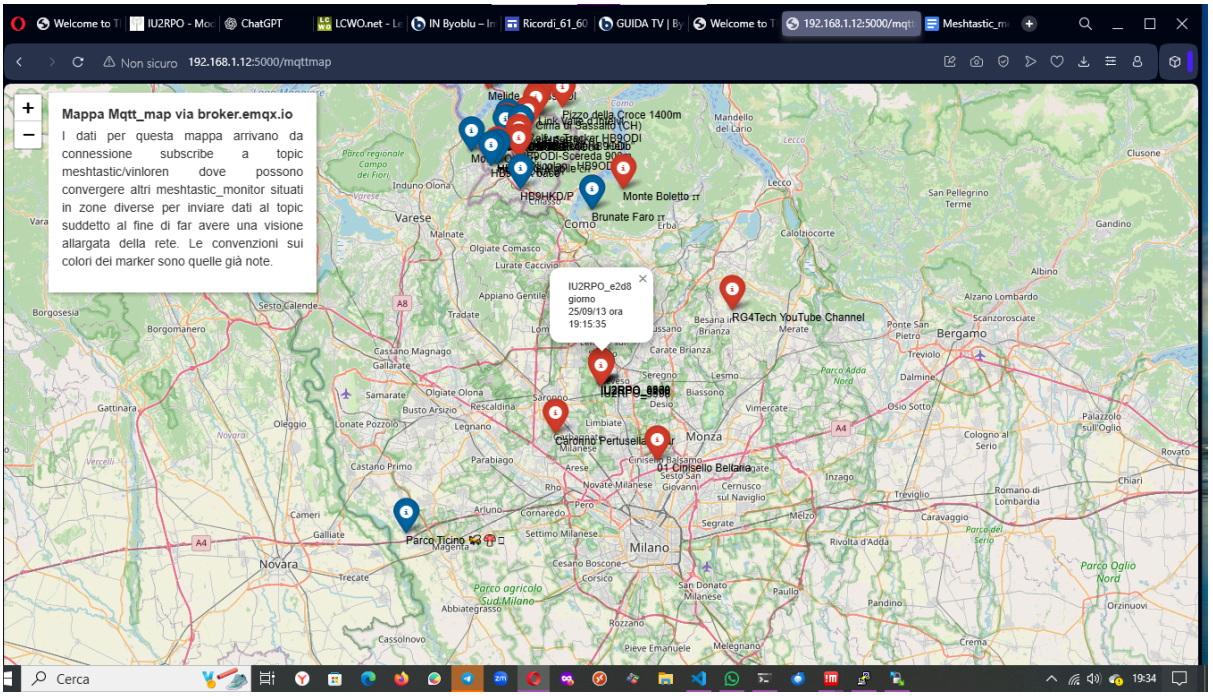
Pagina Nostra_rete



Pagina Mostra_Log



Pagina mqtt_map



Note di aggiornamento installazione

Per non dover rifare un lavoro di ricostruzione dei propri dati ma conservarli aggiungendo nel Db la nuova tabella necessaria al corretto funzionamento della Ver. 1.1 consiglio quanto segue:

1. Da una nuova directory clonare https://github.com/vinloren/meshtastic_monitor
2. Nella nostra installazione aggiungere la definizione della nuova tabella prendendola dal Db del nuovo clone via DB Browser Sqlite attraverso la sua definizione “CREATE TABLE "loranodes" (
   "data" VARCHAR(8) NOT NULL,
   "ora" VARCHAR(8) NOT NULL,
   "node_id" VARCHAR(9) NOT NULL UNIQUE,
   "longname" VARCHAR(28) NOT NULL,
   "alt" INTEGER,
   "lat" FLOAT,
   "lon" FLOAT,
   "batt" INTEGER,
   "snr" FLOAT,
   "pressione" FLOAT,
   "temperat" FLOAT,
   "umidita"FLOAT,
   PRIMARY KEY("node_id")
) da eseguire in “Esegui Sql”
3. a questo punto copiare, dalla directory base verso quella della nostra installazione, le directory source, app, Doc, per sostituire i vecchi contenuti coi nuovi
   essendo entrati in ambiente virtuale eseguire **pip install flask_session** nel caso la vostra installazione non sia quella del 9 Settembre (in ogni caso questa esecuzione male non fa)
4. **pip ip install mqtt.paho** per supporto mqtt