

Relazione progettazione di rete (MQTT - IoT project)

Requisiti:

- Realizzare un sistema distribuito per il monitoraggio della connettività di rete e dello stato di funzionamento di un insieme di "host" mobili/fissi sparso in reti private e connessi tramite Internet.

Specifiche:

1. Gli host periodicamente comunicano con un server centrale segnalando i loro dati di rete ed il tempo di invio del messaggio.
2. Il server memorizza i dati degli host in un repository aggiungendo ai dati segnalati il ritardo nella propagazione;
3. Il server può essere interrogato da un client visualizzando lo stato dei vari host;
4. La comunicazione tra host, server e client deve avvenire tramite MQTT via un message broker in rete pubblica (193.206.55.23);
5. Bisogna poter richiedere un test dei vari host, o di tutti o di uno nello specifico;

Linguaggi di programmazione:

- Per il modulo risiedente sugli host e per il server: Python3 con libreria Paho MQTT;
- Per l'interfaccia web ho utilizzato HTML5, CSS3, BootStrap4, PHP;

Strutturazione del progetto dal punto di vista tecnologico:

Ho sviluppato il progetto su un sistema Unix (Posix certified) qual è MacOS. Su di esso ho costruito un sistema che rispetta le specifiche di cui sopra.

Inoltre il sistema è principalmente sviluppato in **Python** (versione 3+) ad eccezione della parte di interazione con l'utente (una web interface sviluppata con tecnologie front-end quali HTML5, CSS3, BootStrap4 e tecnologie pack-end quali PHP).

Per portare a termine il progetto nel migliore dei modi, ho utilizzato librerie ampiamente diffuse e testate.

A seguire ne riporto la lista, con dei riferimenti e una piccola spiegazione riguardo a “**dove**” sono state utilizzate e “**per cosa**”:

- **Bootstrap**, libreria che permette di importare classi CSS ampiamente diffuse e testate per rendere la parte front-end esteticamente più piacevole. Da me è stata utilizzata proprio con questo scopo. <https://getbootstrap.com>;
- **Paho MQTT**, è la libreria cardine del progetto, consente di creare agevolmente dei client MQTT (che nel contesto vuol dire poter creare un modulo che funge da server e n moduli che fungono da client) che facciano sia **subscriber** che **publisher** e siano ampiamente supportati in quanto questa libreria è sviluppata in maniera open source dalla Eclipse Foundation, che mantiene anche il broker **Mosquitto**. <http://www.eclipse.org/paho/>;
- **re**, è la libreria di Python che permette di lavorare con le regex. Nel mio progetto ne ho fatto ampio uso per poter fare dei controlli sui topic dei messaggi in arrivo sui vari topic.
- **time**, è la libreria di Python che permette di gestire “i tempi” nel senso di istanti della giornata o lavorare con millisecondi, secondi, minuti e ore. Da me è stata utilizzata per calcolare il tempo di invio di ogni messaggio, il tempo di ricezione e tramite un calcolo di differenza tra di essi, calcolare il tempo di propagazione in millisecondi;
- **getmac**, è una libreria Python che permette di lavorare ed estrarre i Mac Address di una determinata interfaccia. È lo scopo per cui l’ho utilizzata anch’io. <https://pypi.org/project/getmac/>;
- **json**, è la libreria Python che permette di costruire messaggi JSON. Io l’ho utilizzata per costruire da prima dei dizionari Python e poi convertirli in messaggi JSON per mandarli come “payload” da publisher a subscriber e dall’altra parte convertire da messaggi JSON a dizionari Python per poi poter lavorare con le informazioni trasmesse. <https://docs.python.org/3/library/json.html>
- **socket**, è la libreria Python che mi ha permesso di estrarre le informazioni relative all’hostname e l’indirizzo IP da ogni host monitorato all’interno della rete;
- **threading**, è la libreria Python che permette di gestire l’interazioni tra thread e la creazione di nuovi thread per permettere una gestione più moderna e ottimizzata dei progetti (multithreading). <https://docs.python.org/3/library/threading.html>;
- **mysql.connector**, è il connettore per far dialogare Python e MySQL. L’ho utilizzato per inserire le informazioni degli host in un Database MySQL versione 8+ ospitato su MacOS. Più avanti parlerò più approfonditamente della gestione del DB;

Gestione, struttura e descrizione del database e del client per comunicare

Per quanto riguarda la parte di database, ho utilizzato MySQL come DBMS. Riporto che avendo una struttura molto semplice, in realtà un database relazionale è fin troppo, tuttavia ho voluto fare questa scelta per aumentare le potenzialità del progetto per futuri eventuali upgrade.

Il database si chiama “**networkMonitoring**”. Le tabelle presenti sono solo 2:

- **hostInformation**: al suo interno vado a memorizzare le informazioni che arrivano dai messaggi periodici (ogni 40 secondi) da parte dei client. In questa tabella nello specifico memorizzo L'indirizzo IP, l'hostname, il time stamp di quando il messaggio è partito lato publisher (TimeStampReceived), il time stamp di quando il messaggio è arrivato lato subscriber (TimeStampStored), il tempo di propagazione in millisecondi e il Mac address della macchina che ha inviato il messaggio;
- **hostSeen**: questa tabella è utilizzata per memorizzare quali host ho visto passare nel corso del tempo. A regime qui avrò tutti (e i soli) host che sto monitorando. Questa tabella è utilizzata lato PHP per scaricare le informazioni da mostrare sulla pagina web per selezionare uno specifico host a cui mandare un messaggio di “check”. Al suo interno ho il Mac address dell'interfaccia da cui si è ricevuto un messaggio;

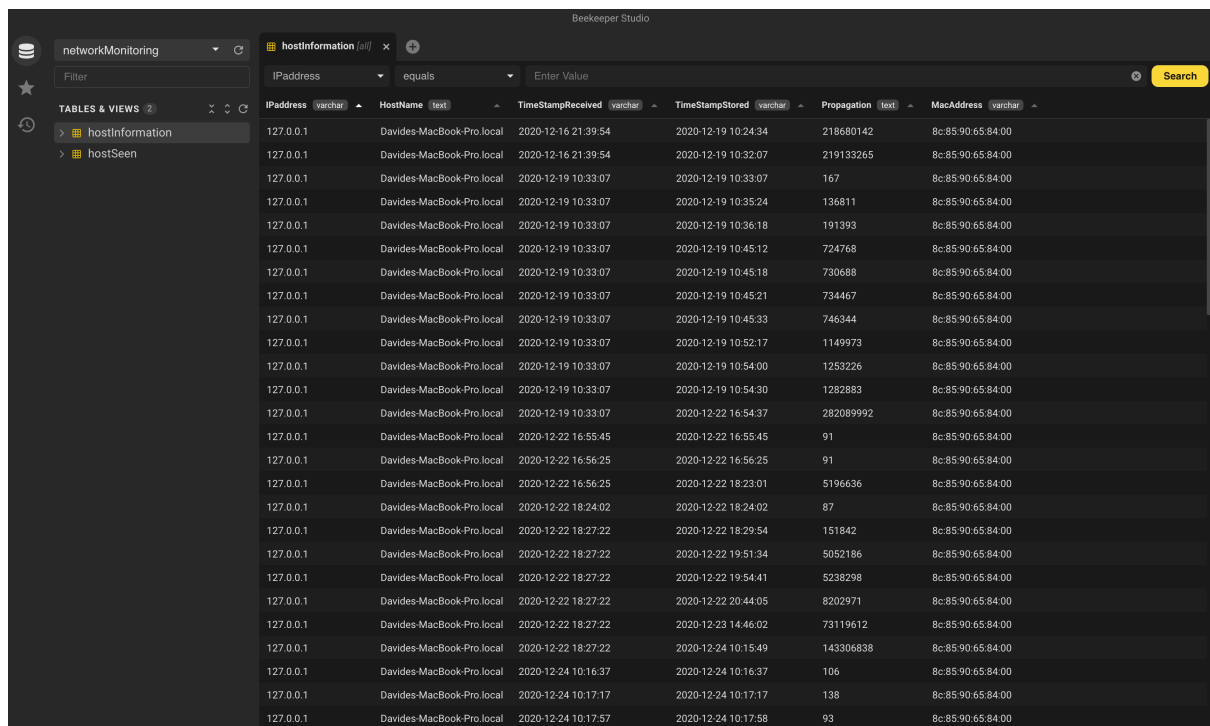
Riporto di seguito il codice SQL (nello specifico MySQL in quanto esso non rispetta lo standard SQL ANSI/ISO):

```
1 CREATE TABLE `hostSeen` (  
2     `MacAddress` VARCHAR(17) NOT NULL,  
3  
4     PRIMARY KEY (`MacAddress`)  
5 )  
6  
7 CREATE TABLE `hostInformation` (  
8     `IPaddress` VARCHAR(15) NOT NULL,  
9     `HostName` TEXT,  
10    `TimeStampReceived` VARCHAR(19) NOT NULL,  
11    `TimeStampStored` VARCHAR(19) NOT NULL,  
12    `Propagation` TEXT,  
13    `MacAddress` VARCHAR(17) NOT NULL,  
14  
15    PRIMARY KEY (`MacAddress`, `timeStampReceived`,  
16    `timeStampStored`)
```

N.B: questo sorgente è contenuto nel file “**queryDB.sql**”.

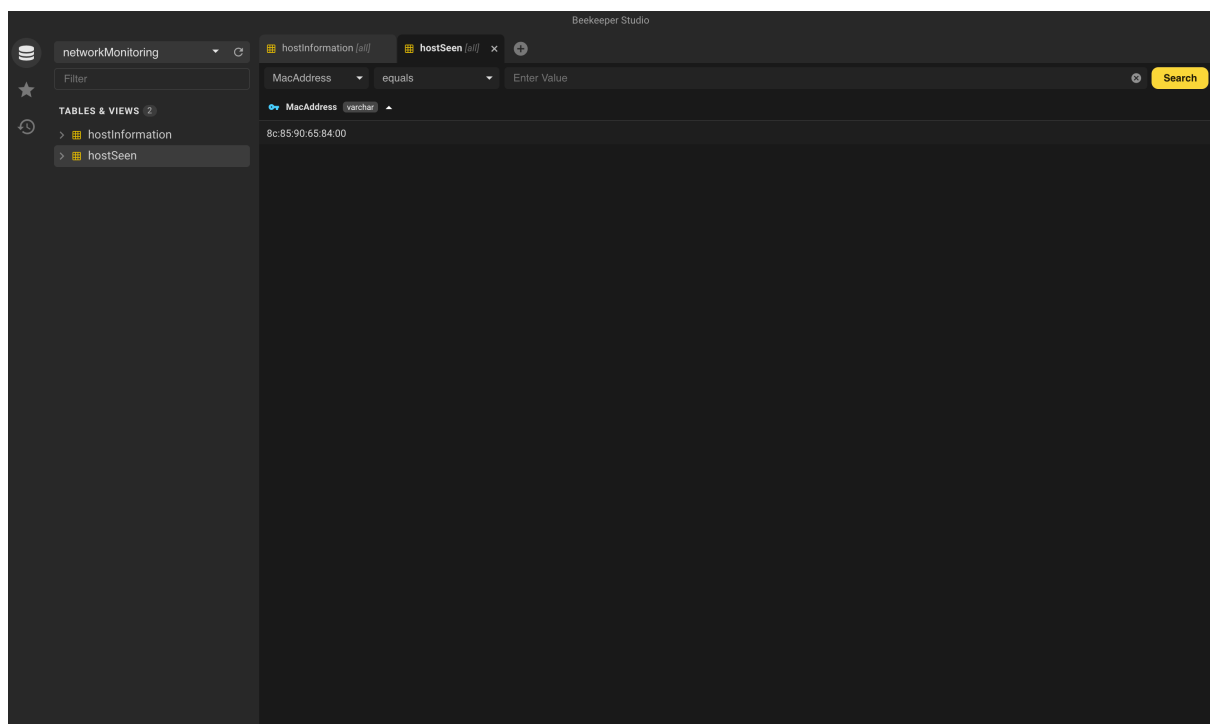
Per interfacciarmi con il database non da interfaccia a linea di comando ma utilizzando qualcosa di più strutturato e moderno, ho utilizzato un ottimo software open source sviluppato da **Matthew Rathbone** e **Gregory Garden**, di cui riporto il link ma sul quale non mi dilungo in quanto non di nostro interesse, in quanto ognuno può interfacciarsi al DB nel modo che preferisce. Link: <https://www.beekeeperstudio.io>.

Riporto, tuttavia, degli screenshot che mostrano dall'interno del suddetto software i dati memorizzati (durante le fasi di test) all'interno del DB:



The screenshot shows the Beekeeper Studio interface with the 'hostInformation' table selected. The table has columns: IPAddress, HostName, TimeStampReceived, TimeStampStored, Propagation, and MacAddress. The data shows multiple entries for 'Davides-MacBook-Pro.local' with various timestamps and propagation values.

IPAddress	HostName	TimeStampReceived	TimeStampStored	Propagation	MacAddress
127.0.0.1	Davides-MacBook-Pro.local	2020-12-16 21:39:54	2020-12-19 10:24:34	218680142	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-16 21:39:54	2020-12-19 10:32:07	219133265	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:33:07	167	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:35:24	136811	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:36:18	191393	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:45:12	724768	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:45:18	730688	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:45:21	734467	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:45:33	746344	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:52:17	1149973	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:54:00	1253226	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-19 10:54:30	1282883	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-19 10:33:07	2020-12-22 16:54:37	282089992	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 16:55:45	2020-12-22 16:55:45	91	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 16:56:25	2020-12-22 16:56:25	91	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 16:56:25	2020-12-22 18:23:01	5196636	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:24:02	2020-12-22 18:24:02	87	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-22 18:29:54	151842	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-22 19:51:34	5052186	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-22 19:54:41	5238298	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-22 20:44:05	8202971	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-23 14:46:02	73119612	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-22 18:27:22	2020-12-24 10:15:49	143306838	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-24 10:16:37	2020-12-24 10:16:37	106	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-24 10:17:17	2020-12-24 10:17:17	138	8c:85:90:65:84:00
127.0.0.1	Davides-MacBook-Pro.local	2020-12-24 10:17:57	2020-12-24 10:17:58	93	8c:85:90:65:84:00



The screenshot shows the Beekeeper Studio interface with the 'hostSeen' table selected. The table has columns: MacAddress. The data shows a single entry for '8c:85:90:65:84:00'.

MacAddress
8c:85:90:65:84:00

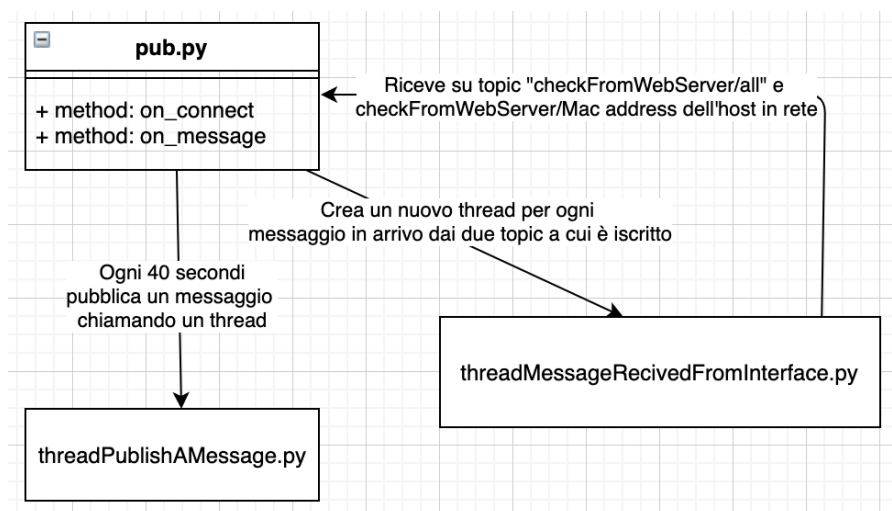
Non avendo, il DB, una delle relazioni particolari tra le tabelle non riporterò uno schema relazionale.

Struttura del sistema MQTT

Il progetto, come da specifiche, prevede un modulo Python “**pub.py**” eseguibile su tutti gli host che vogliono essere monitorati in rete. Questo modulo ha la seguente struttura:

1. Tramite la libreria Paho MQTT viene creato un oggetto client, che prende in input come identificativo il Mac address dell’interfaccia che pubblicherà;
2. Vi è un’iscrizione ad una tupla di topic (nello specifico 2 topic) in quanto questo publisher è vero che principalmente pubblica ma è vero anche che funge anche da subscriber, in particolare per ricevere messaggi dal modulo PHP che a sua volta intercetta comandi dall’interfaccia web con cui l’utente può richiedere dei check. I due topic a cui è iscritto sono: “checkFromWebServer/all” e “checkFromWebServer/ + str(gma())”. Il primo topic è il canale su cui riceve messaggi ogni qual volta dall’interfaccia, l’amministratore richiede un check a tutti gli host. L’altro è il canale che viene utilizzato quando viene richiesto un check allo specifico host (si noti la chiamata gma() che serve per estrarre il Mac address dell’interfaccia);
3. Inoltre, questo programma associa ad ogni client due callback, 1 che viene richiamata al momento della connessione, ed una che viene richiamata all’arrivo di un messaggio;
4. Ogni client ogni 40 secondi di attesa pubblica un messaggio tramite un thread separato che andrà gestito dal subscriber di controllo;
5. Anche la gestione dei messaggi in arrivo viene fatta con un thread separato;

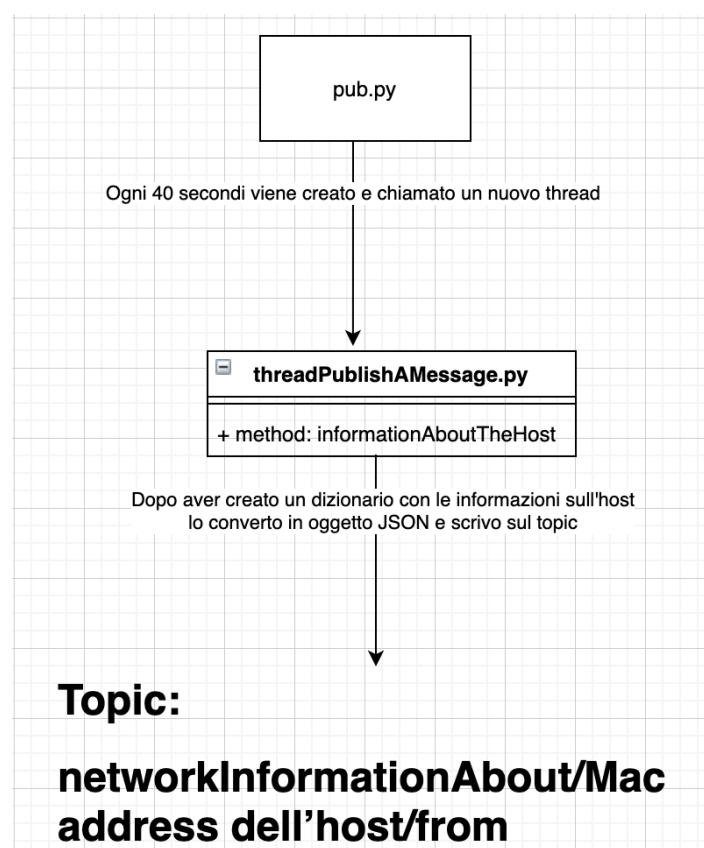
Per fornire un’idea più chiara del software che gira sui client in rete riporto uno schema grafico del tutto:



Per quanto riguarda i due thread sopra accennati, ora ne darò un approfondimento per descriverne il funzionamento:

threadPublishAMessage.py: è il thread che ho costruito per occuparsi della pubblicazione dei messaggi periodici (cioè quei messaggi che vengono pubblicati ogni 40 secondi dagli host sparsi nella rete). Il thread ha la seguente struttura:

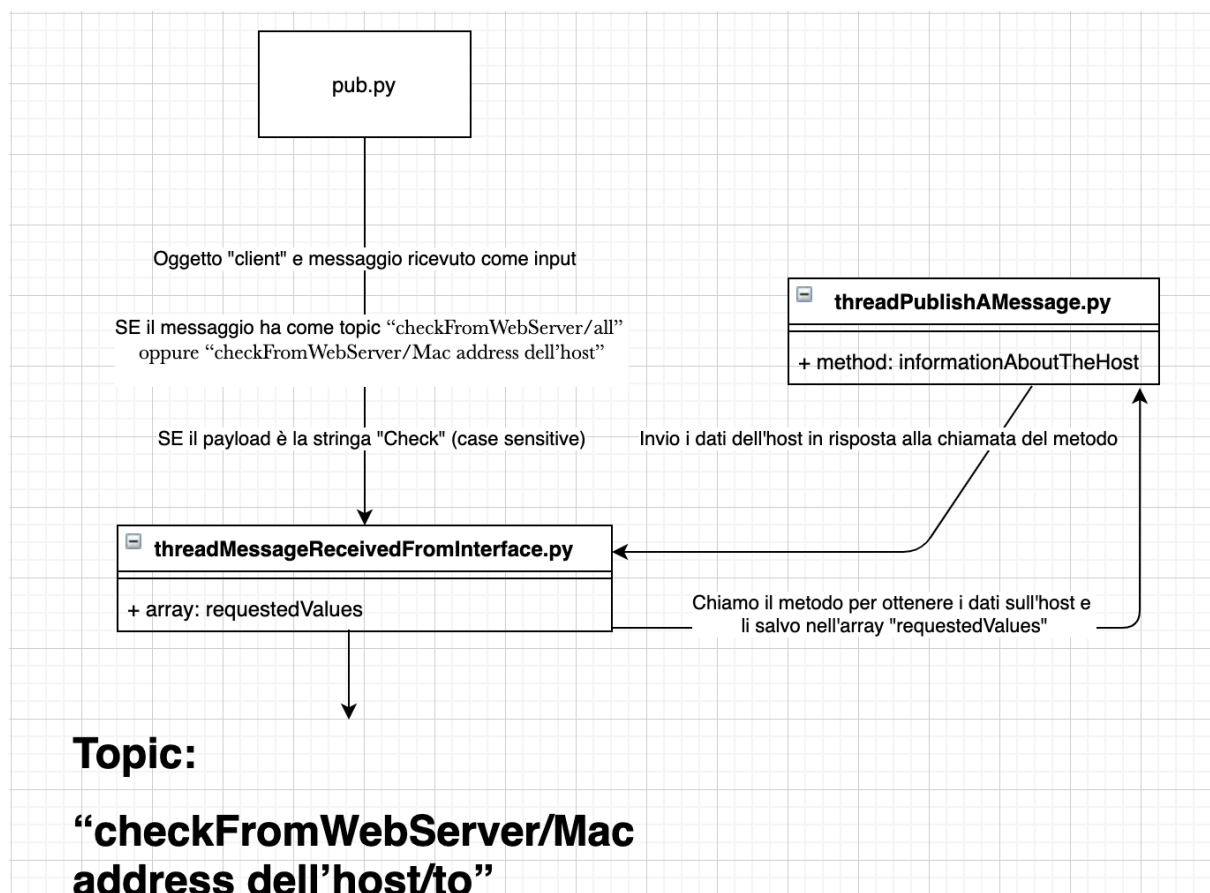
1. Com'è visibile dallo schema di cui sopra, il thread viene creato e lanciato dal “pub.py” ogni 40 secondi;
2. Il thread prende in input un oggetto client (creato precedentemente all'interno del “pub.py” tramite la libreria Paho MQTT);
3. Il thread richiama una funzione per estrarre le informazioni dell'host sul quale sta operando, tali informazioni sono: hostname, indirizzo IP, Mac address, start time (che come già precedentemente spiegato è il momento in cui vengono estratte queste informazioni);
4. Da prima costruisco un dizionario, all'interno del thread, con le informazioni appena descritte e tramite l'utilizzo della libreria “json” di Python, trasformo il dizionario in un oggetto JSON;
5. Infine, vado a pubblicare l'oggetto JSON sul topic “networkInformationAbout/Mac address dell'host/from”;



threadMessageReceivedFromInterface.py: è il thread che si occupa di gestire i messaggi che vengono ricevuti dal “pub.py”. In particolare questo thread viene creato e

lanciato quando viene chiamata la callback `on_message`. Vediamone nel dettaglio il suo funzionamento:

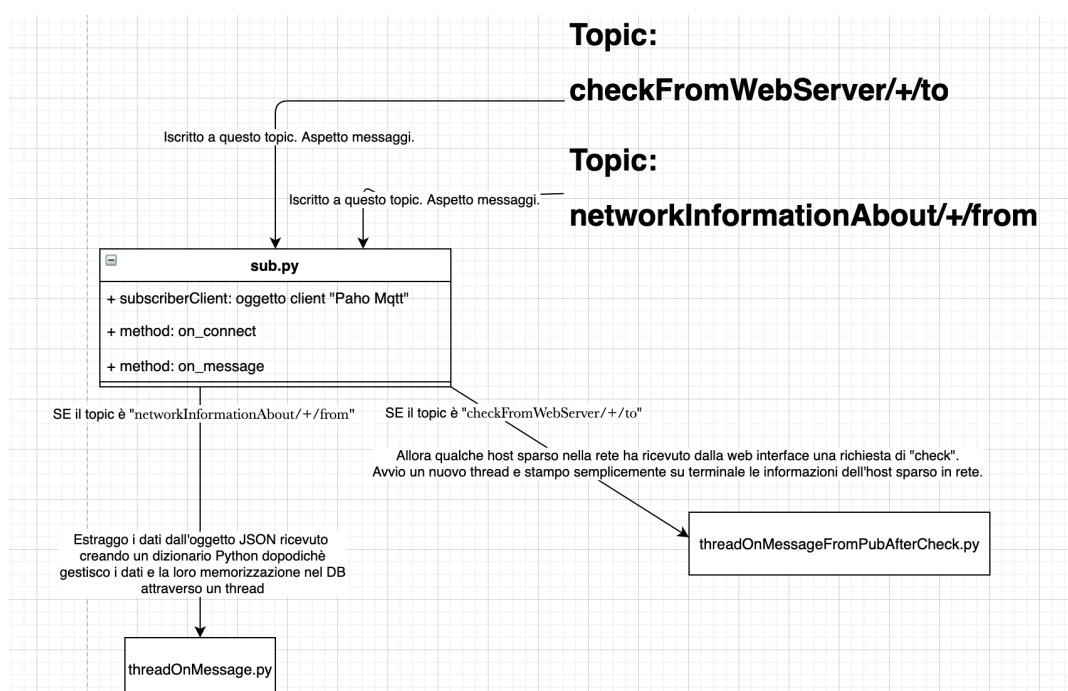
1. Il thread prende in input il messaggio ricevuto dall'interfaccia e un oggetto client (costruito con Paho MQTT). Dopodiché vi è un controllo sul topic associato al messaggio ricevuto sfruttando le regex (espressioni regolari);
2. Se effettivamente i topic associati al messaggio sono quelli di nostro interesse ("`checkFromWebServer/all`" oppure "`checkFromWebServer/Mac address dell'host`") allora vi è un ulteriore controllo sul payload del messaggio. Esso infatti dev'essere la stringa "`Check`" (case sensitive);
3. Se entrambi i controllo (payload e topic) sono superati, allora estraggo le informazioni dell'host richiamando il metodo contenuto nel thread "**`threadPublishAMessage.py`**" che è visibile dallo schema sopra riportato, il metodo in particolare è chiamato "`informationAboutTheHost`" e restituisce un array contenente il Mac address, il tempo di creazione dell'array (il solito `startTime`), l'indirizzo IP e l'hostname;
4. Recuperate le suddette informazione, viene effettuata una publish sul topic "`checkFromWebServer/Mac address dell'host/to`";



Per tutta la parte “server”, ove in questo contesto intendiamo indicare un host MQTT che principalmente si occupa di leggere i dati di tutti gli altri host in rete (prendendo i valori scritti sui topic da questi ultimi), ho creato un programma Python denominato “sub.py” che presenta alcune caratteristiche e complessità che andrò a spiegare.

sub.py: è il modulo, come appena detto, che legge i dati pubblicati sui topic dagli host sparsi in rete:

1. Al solito viene creato un oggetto client, tramite la solita libreria Paho MQTT, che prende l'identificativo di “subscriberClient”;
2. A questo client sono associate 2 callback (anche qui denominate “on_connect” che viene “agganciata” quando avviene una connessione e “on_message” che si attiva alla ricezione di un messaggio);
3. Il client aspetta messaggi sui seguenti topics: “checkFromWebServer/+/to” ove la wildcard + permette di iscriversi a tutti e soli i topic che hanno quella struttura ma a livello del “+” presentano un MAC address e su “networkInformationAbout/+/from”;
4. Nella callback “on_message” tramite l'utilizzo di regex avviene il controllo sui topic. Se il messaggio ricevuto ha come topic associato “networkInformationAbout/+/from” allora ho ricevuto un messaggio JSON con i dati periodici di uno degli host sparsi nella rete. Quindi in questa casistica converto il JSON object in un dizionario Python e creo e avvio un thread che gestisca i dati e il loro salvataggio nel database (in particolare il thread chiamato è “threadOnMessage.py”). Se il messaggio ricevuto invece ha come topic “checkFromWebServer/+/to” allora il messaggio ricevuto è quello di controllo che ha ricevuto un host sparso in rete dalla web interface. In questo caso avvio il thread “threadOnMessageFromPubAfterCheck.py” per gestire la stampa sul terminale delle informazioni ricevute senza fare null'altro;

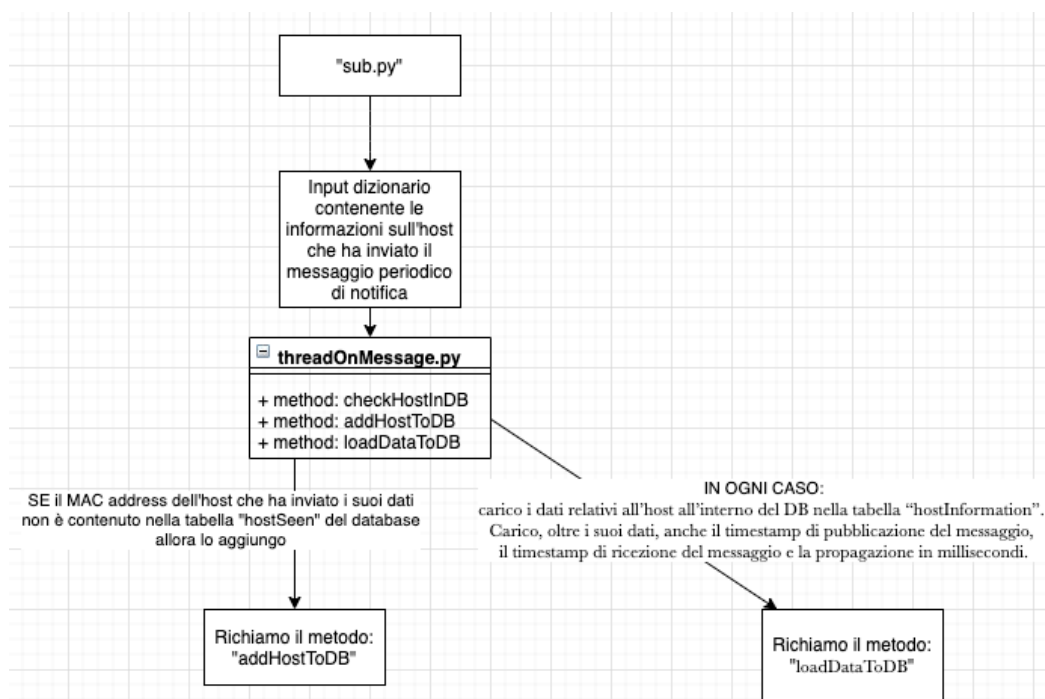


Ora vediamo nel dettaglio il funzionamento dei sopracitati threads che vengono creati e richiamati all'interno della on_message di "sub.py".

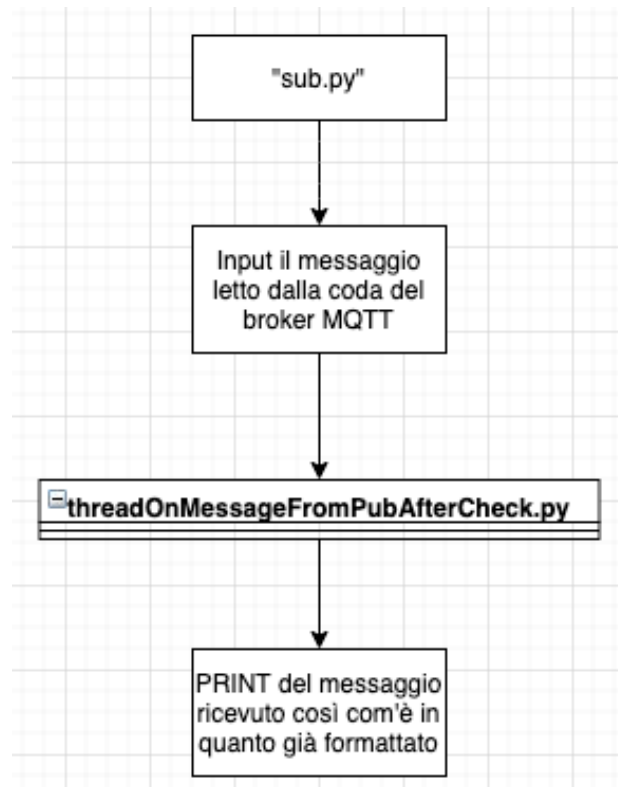
threadOnMessage.py: Questo thread viene creato e chiamato (come ben visibile dallo schema qui sopra riportato) dal "sub.py" quando si riceve un oggetto JSON contenente i dati dell'host che li pubblica all'interno della rete. In particolare viene creato un thread di questo tipo quando viene passato il controllo che il messaggio ricevuto abbia come topic associato "networkInformationAbout/+(Mac address dell'host che pubblica)/from". Dal momento che questo controllo è superato so per certo che il messaggio pubblicato sarà un oggetto JSON e che ci ritroviamo nel caso di un messaggio di quelli periodici in arrivo (i messaggi che gli host in rete pubblicano ogni 40 secondi).

All'interno di questo thread vengono svolte le seguenti azioni:

1. Il thread prende in input il messaggio ricevuto. Esso sarà nel formato dizionario Python in quanto in "sub.py" è stato convertito in oggetto dizionario da oggetto JSON;
2. Estraggo i dati quali hostname, indirizzo IP, MAC address, e tempo di pubblicazione (start Time) in delle variabili;
3. Faccio il controllo che l'host non sia mai stato visto prima (non ho mai ricevuto un messaggio da quel determinato host). Se così è tramite la chiamata al metodo "addHostToDB" inserisco il MAC address dell'host nella tabella "hostSeen" del database. Altrimenti se l'ho già visto in precedenza non lo aggiungo nuovamente. Il check viene fatto con un metodo chiamato "checkHostInDB";
4. In entrambi i casi carico i dati relativi all'host all'interno del DB nella tabella "hostInformation". Carico, oltre i suoi dati, anche il timestamp di pubblicazione del messaggio, il timestamp di ricezione del messaggio e la propagazione in millisecondi. Lo faccio richiamando il metodo "loadDataToDB";

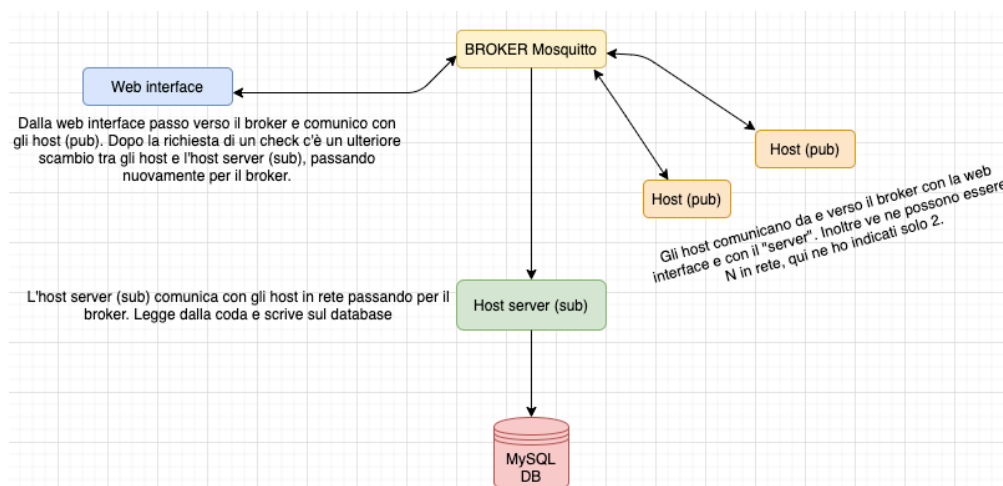


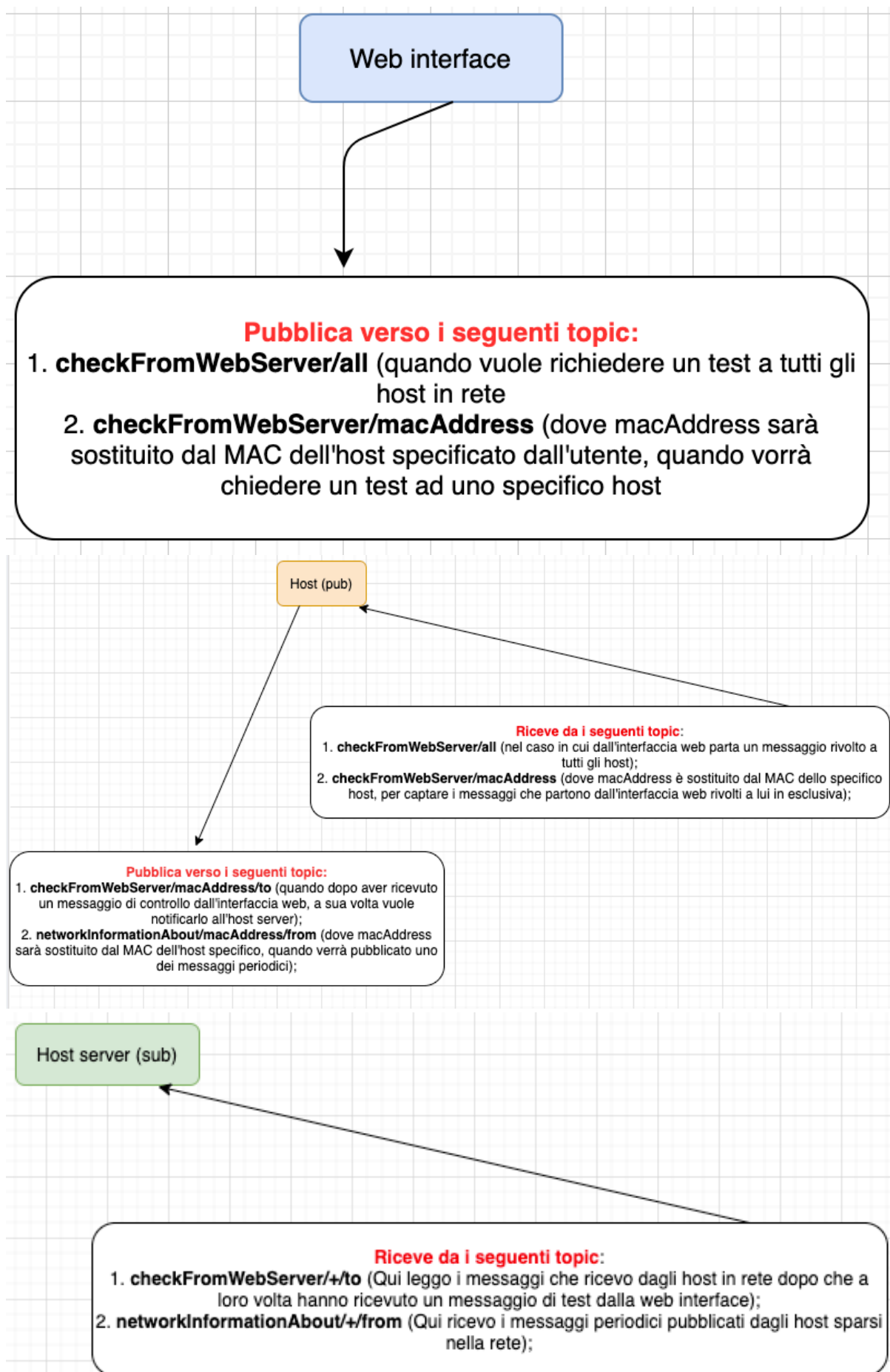
threadOnMessageFromPubAfterCheck.py: è il thread che viene creato e richiamato nel momento in cui si riceve un messaggio da un host in rete che a sua volta ha ricevuto un messaggio di controllo dall'interfaccia web. È un thread estremamente semplice in quanto le informazioni vengono ricevute già dal publisher formattate per una stampa efficace. Quindi sostanzialmente quello che fa questo thread essere creato e avviato alla ricezione di un messaggio, prendere in input il messaggio ricevuto e fare una print del messaggio stesso.



Da questi schemi ed inoltre dalla spiegazione delle caratteristiche di ogni codice sorgente, se ne può intuire il funzionamento e la logica.

Ora, però, presenterò alcuni schemi riassuntivi sulla **struttura dei topic** (che già sono stati descritti) per dare un'idea più schematizzata. Inoltre, schematizzerò anche la struttura degli oggetti che ruotano intorno alla **struttura MQTT**.

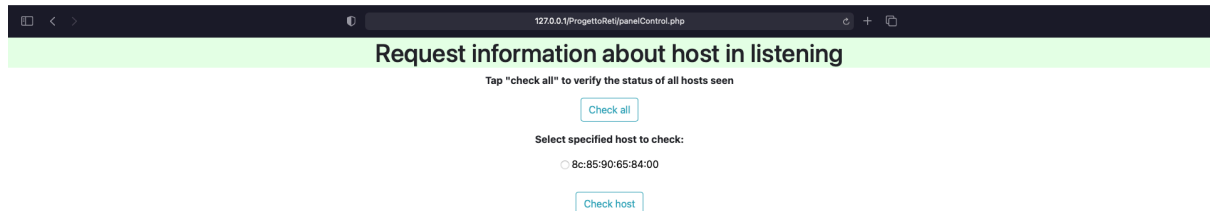




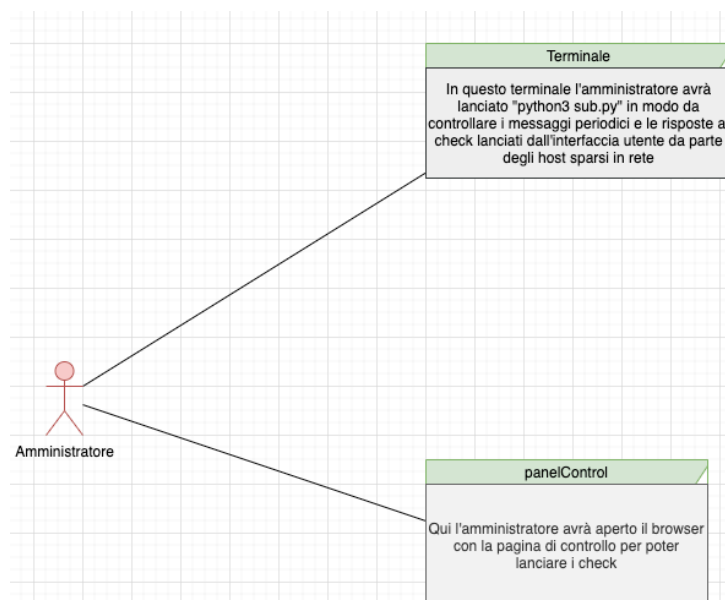
Interfaccia web per l'amministratore (panelControl)

Ho costruito un'interfaccia che fosse il più semplice possibile e visivamente intuitiva. Le tecnologie che ho utilizzato mi hanno permesso una facile scrittura ed inoltre un ottimo supporto nel tempo.

In particolare per il markup e la struttura della pagina ho usato il solito HTML, ho scritto poco CSS perché poi ho importato tutte le classi CSS dal framework Bootstrap.



L'interfaccia presenta due tasti, uno per richiedere un test a tutti gli host incontrati fino ad ora (cioè tutti gli host che compaiono nella tabella hostSeen) ed un altro che permette di selezionare prima un host dalla lista di quelli forniti (nell'esempio è visibile solo un host) e poi richiederne il check. La lista, al solito, fa una query al DB per estrarre tutti i MAC address degli host incontrati (cioè di nuovo prende tutti i MAC address che compaiono nella tabella hostSeen).



La parte di backend della piattaforma web è stata scritta in PHP. In particolare tramite PHP ho potuto sia fare la parte di chiamate e query verso il database, sia la parte di invio dei messaggi MQTT verso il broker utilizzando una libreria open source di cui riporto la pagina GitHub. Non starò a descrivere nel dettaglio il funzionamento della libreria (che in realtà è molto simile alla Paho MQTT solo che supporta PHP) e nemmeno a riportare il codice delle query da PHP verso il database MySQL in quanto sarebbe una ripetizione del codice già scritto in “panelControl.php” e “functionPHP.php”.

GitHub libreria utilizzata: <https://github.com/php-mqtt/client>

La libreria è mantenuta da: Marvin Mall

Licenza del progetto

MIT License

Copyright (c) 2020 Davide Netti

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

