DIY: timerPDM per ozono

Molti generatori di ozono sono privi di PDM (Pulse Duration Modulation) e quindi non possono regolare l'ozono prodotto nell'unità di tempo. Alcuni generatori economici sono addirittura privi di timer. Ma sia PDM che il timer sono praticamente indispensabili per un corretto uso dell'ozono nella sanificazione domestica. Per un approfondimento vedi introduzione sull'ozono https://github.com/msillano/Ozone-coronavirus-sonoff/blob/master/Ozono_Coronavirus_Sonoff_it.pdf ed il simulatore per l'ozono https://github.com/msillano/Ozone-coronavirus-sonoff/tree/master/PROJECTS-DIY/simulOzone

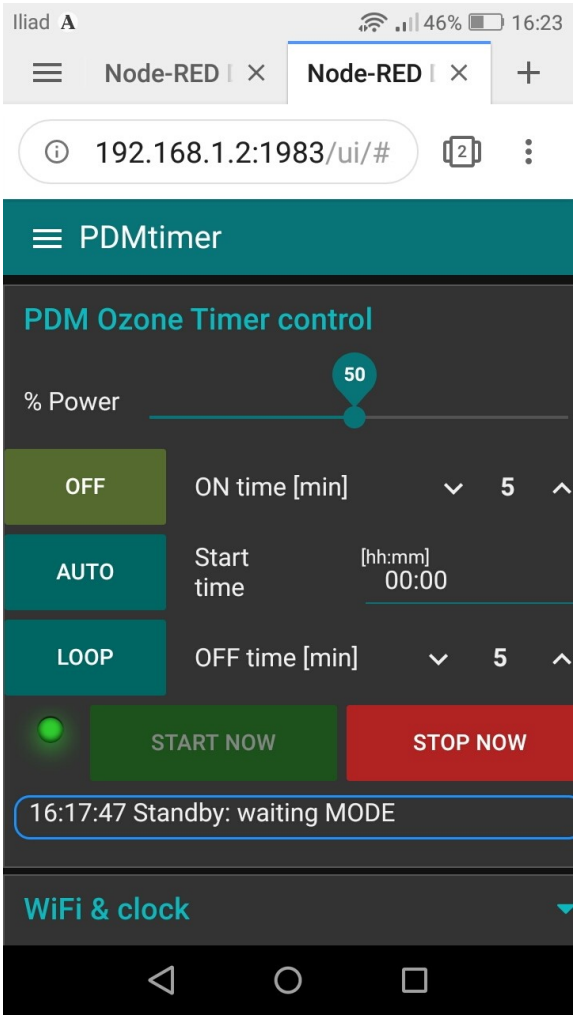
*La soluzione in questi casi è questo progetto DIY **TimerPDM**: un timer PDM per generatori di ozono, progettato e realizzato secondo lo stato dell'arte, migliore di molti timer presenti in ozonizzatori anche di fascia alta.*

Ozono in modalità air purify

Il **timerPDM** permette un un primo controllo dell'ozono prodotto tramite PDM (2%..100%) con un ciclo di un minuto. Inoltre, in modo LOOP, permette di regolare separatamente i tempi di accensione (1min..3h) e di riposo (1min..24h) con intervalli di 1 minuto.

Ozono in modalità chock

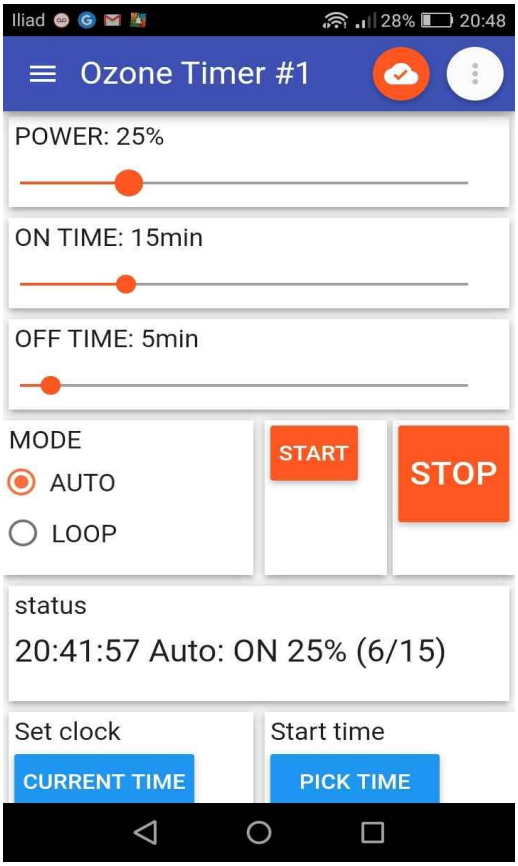
In modo AUTO, oltre al controllo PDM e l'impostazione della durata di accensione è possibile programmare ad orario l'avvio dell'ozonizzatore (hh:mm). Inoltre il pulsante sul **timerPDM** permette l'avvio manuale del modo AUTO con un ritardo iniziale di 1 minuto per allontanarsi.



Questo **timerPDM** può essere controllato direttamente o da remoto: per scambiare dati e comandi con altre applicazioni esterne segue la logica IOT ed il protocollo MQTT.
Per la connettività contiene uno STA (client WIFI) per collegarsi ad internet tramite una rete WiFi esistente, un AP (Access Point WiFi) per permettere a smartphone e PC di collegarsi al timer anche in assenza di rete e router.
Per la logica di funzionamento contiene un client MQTT ed un server (broker) MQTT, dotati di un proprio linguaggio di programmazione (script).

Concludendo in questo progetto la logica di funzionamento è tutta interna: non è richiesto un server MQTT esterno (tipo mosquitto) attivo 24/7. Questo **timerPDM** è autonomo, ma nel contempo

compatibile con client e broker MQTT standard esterni: pertanto può essere usato standalone o essere controllato da lontano o anche essere integrato in progetti più complessi, di domotica e case intelligenti.
Il monitoraggio e la configurazione di base di **timerPDM** possono essere effettuati con client MQTT con dashboard, sia su PC che su smartphone Android; infatti sono disponibili due applicazioni:



TimerPDM-red per PC, utilizzando *node-red*

PDM timer per smartphone Android, usando una app standard (*IoT MQTT Panel*)

Pur usando componenti aggiornati e di qualità, questo timer è molto economico: costo totale inferiore a 10 € (13 US$) !

Tutto questo è reso possibile dalla felice unione di due prodotti fenomenali:

> *Sonoff S20* economico interruttore WIFI con un ottimo fattore di forma, alimentatore, processore EPS8266 e relais
> *esp_MQTT*, un firmware per EPS8266 creato da Martin-ger, che trasforma *Sonoff* in un carrarmato.

*Sonoff S20 è uno straordinario interruttore WiFi, ma ha prestazioni non specializzate per il controllo di ozonizzatori. Con esp_MQTT si può riprogrammare la logica di funzionamento di S20, ottenendo così un **timerPDM** funzionalmente completo.*

Prestazioni:

- Power: regolazione **PDM** 2..100 %
- Durata acceso: **ON time** (minuti)
- Intervallo spento: **OFF time** (minuti)
- Orario di avvio: **Start time** (hh:mm)
- *Modalità di funzionamento:*

0. **OFF** standby di sicurezza.

1. **AUTO**: start a tempo, un solo ciclo di durata **ON time,** per applicazioni *shock*.
   accende: automaticamente a **Start time**, oppure pulsanti **START** (immediato), **TOGGLE** (ritardato)
   spegne: dopo **ON time**. Oppure **STOP**, **TOGGLE** (immediati)

2. **LOOP:** ciclo infinito **ON time** (acceso) + **OFF time** (spento) per applicazioni *air purify* .
   inizia: **START** (immediato), **TOGGLE** (ritardato) .
   termina: **STOP**, **TOGGLE** (immediati).

- Il **timerPDM** ha due spie:

    - VERDE lampeggia in modo OFF, mentre è accesa fissa in modo AUTO o LOOP
    - ROSSA è accesa quando il carico (ozonizzatore) è alimentato.

- Nei periodi ON, il carico è modulato dal PDM, con base 60 secondi. 100% equivale a sempre acceso.

- I valori massimi delle durate ON e OFF (in minuti) sono definiti nell'UI e non nel timer. Possono quindi essere facilmente modificati:
    - Window: **ON time** max 6H, **OFF time** max 24H per poter avere cicli giornalieri.
    - Android: **ON time** max 1H, **OFF time** max 1H per una più agevole impostazione.

- In modo **OFF**, il pulsante **STOP** aggiorna tutti i valori presentatati sullo schermo delle app, ad esempio in caso di nuovo collegamento.

- Pulsante su Sonoff: funzione di **TOGGLE** (AUTO e LOOP). Quando agisce come **START** l'azione è ritardata di 1 minuto per permettere l'allontanamento dell'utente, mentre come **STOP** l'azione è immediata.
- In modo **OFF**, premendo il pulsante **TOGGLE** una volta, si passa al modo AUTO e il led VERDE da lampeggiante diventa fisso. Premere il pulsante una seconda volta equivale ad uno START (ritardato 1 minuto) che utilizza i parametri memorizzati. Una terza premuta agisce da STOP (il led VERDE lampeggia).

Caratteristiche:

- ✔ *Autonomo*: contiene un server MQTT con la logica di funzionamento (script).
- ✔ *Compatibile*: può essere usato con client MQTT e server MQTT standard.
- ✔ *Autostart*: conserva status e configurazione in flash RAM: in caso di reset o blackout ripristina lo status precedente in modo autonomo.
- ✔ *NTP-client*: con il collegamento al WIFI domestico, ed accesso ad Internet, usa NTP per avere l'ora esatta. Comando timezone per ora estiva/invernale. In assenza di collegamento NTP, l'ora può essere impostata dall'orologio del PC o dello smartphone.
- ✔ *Console seriale:* per installazione e debug, via WiFi (telnet) o cavo seriale (COM)
- ✔ *Aggiornamento OTA*: la logica di funzionamento (script) è aggiornabile via OTA (Over The Air), senza spostare od aprire il timer.
- ✔ *Client MQTT:* attualmente 2 applicazioni per monitoraggio e configurazione:
    - "***timerPDM-red***" per PC, (usa *node-red)*
    - "***PDM timer***" per Android, (usa *IoT MQTT Panel*).
- ✔ *Alimentazione: 110-240 AC*
- ✔ *Consumo in stand-by*: non misurabile (< 1 W)
- ✔ *Max carico:* 240 V 10A/300W (http://www.fanhar.com/en/products/21.html)

Software

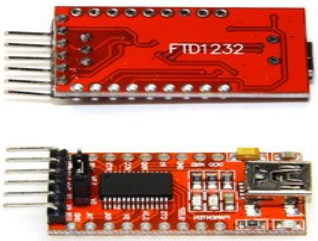
*Ringrazio gli autori dei seguenti software utilizzati nella realizzazione del **timerPDM:***

esp_MQTT https://github.com/martin-ger/esp_mqtt
esptool https://github.com/espressif/esptool
puTTY https://putty.org/
TCP Telnet Terminal: https://play.google.com/store/apps/details?id=project.telnettcpterminal&hl=en
Notepad++ https://notepad-plus-plus.org/
wpp_pampa http://www.winpenpack.com/en/index.php
node-red https://nodered.org/docs/getting-started/installation
IoT MQTT Panel https://play.google.com/store/apps/details?id=snr.lab.iotmqttpanel.prod&hl=en

Materiali

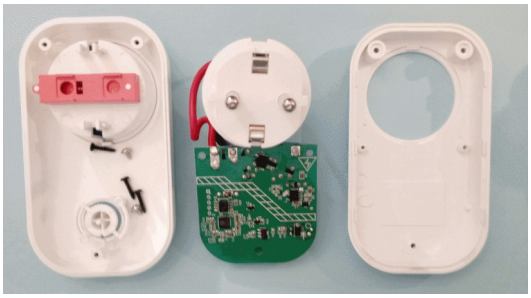
	Itead Sonoff S20 Presa Intelligente Wifi Spina EU-F (anche Ita-10A) https://it.aliexpress.com/item/4000382230184.html (alternativa: mod. S26, ma presenta collegamenti più difficili)	$ 8,52
	Alternativa più economica, con solo LED verde, adatta ad essere montata sul generatore di ozono: Sonoff basic WiFi wireless switch https://www.itead.cc/smart-home/sonoff-wifi-wireless-switch.html	€ 4.85
	4/5 pin per c.s.	$ 0.10
	4 Female To Female Jumper Cable Dupont	$ 0.10
	FE-SP-HDR23-100/22 Filtro per carico induttivo (Surge protector) opzionale. Richiesto per carichi elevati.	$ 4.16

Strumenti

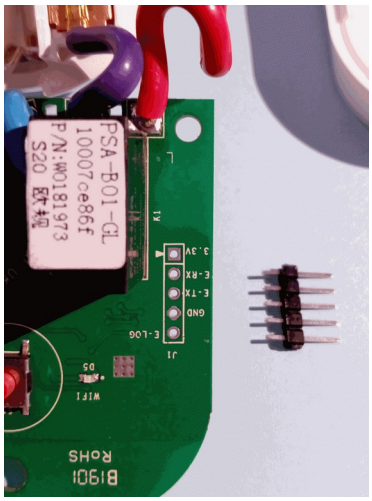
	Saldatore a punta sottile (uso PBLK 6 A1 Parkside)
	FTDI USB 3.3 V 5.5 V Modulo Adattatore Seriale TTL per Arduino (US$ 2.41)

Costruzione

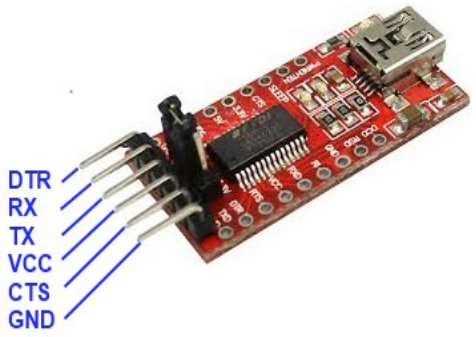
1) preparazione dell'interruttore S20



1 **Estrarre la scheda S20 dal contenitore.**
Tre viti uniscono i due gusci del contenitore.
Altre due viti bloccano il circuito stampato

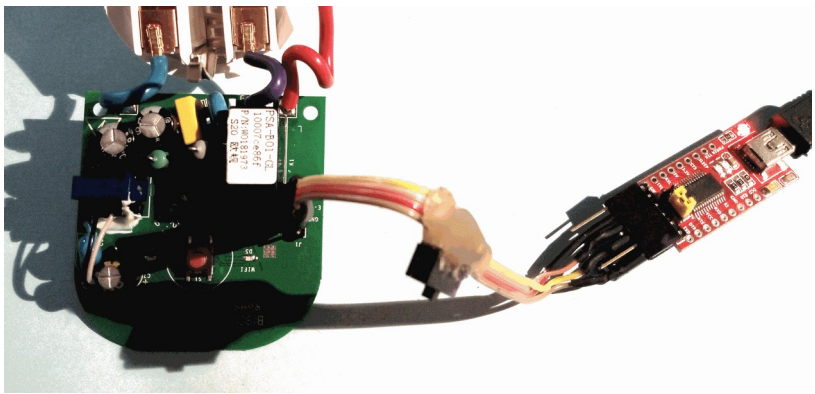


2 Saldare un connettore a 4/5 pin nelle piazzole di Sonoff-S20 (serve per caricare il firmware)



3 Preparare il collegamento a 4 fili FTDI ↔ Sonoff:

Posizionare il jumper su FTDI su 3.3 V (non su 5V, distrugge S20).



Collegamento:

FTDI		**S20**
VCC	→	3.3
RX	→	TX
TX	→	RX
GND	→	GND

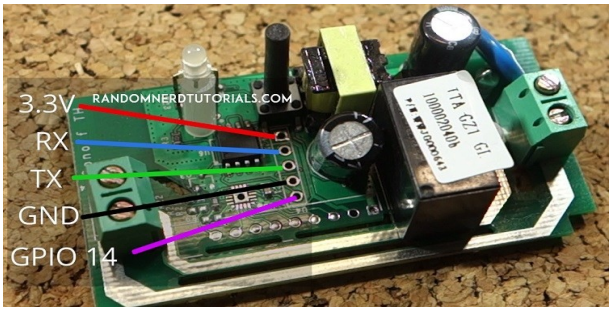
Se si usano connettori al posto di cavi volanti è utile un interruttore sul collegamento VCC → 3.3

nota: *per istruzioni più complete sul cavo consultare:* [*http://randomnerdtutorials.com/how-to-flash-a-custom-firmware-to-sonoff/*](http://randomnerdtutorials.com/how-to-flash-a-custom-firmware-to-sonoff/)

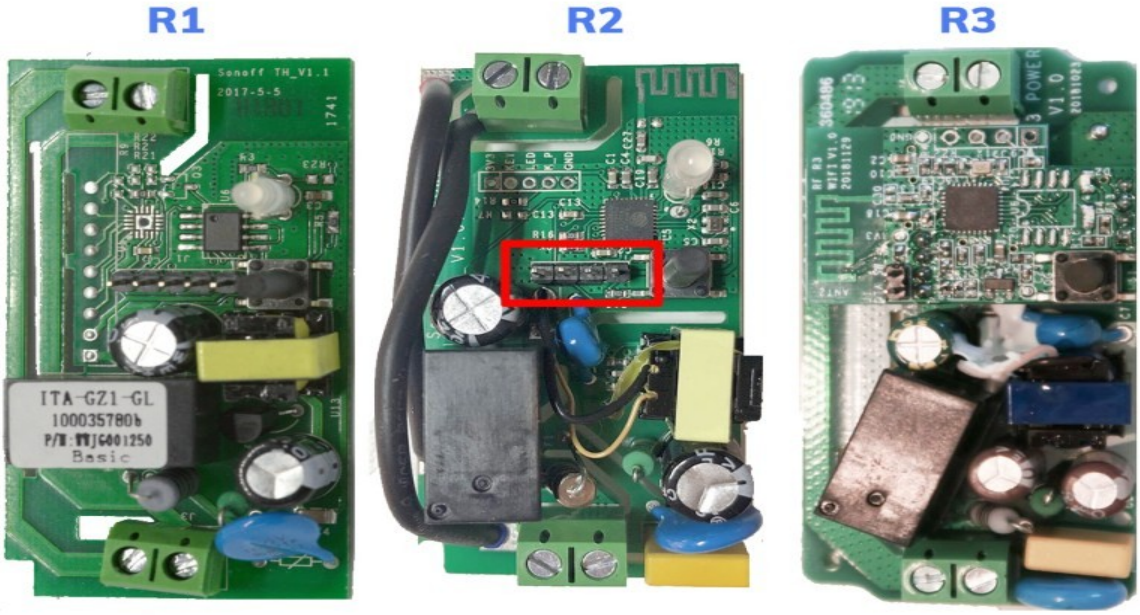
Uso di S26

Questo modello è analogo al modello S20, ma più piccolo e non usa un connettore. Occorre quindi saldare 4 fili direttamente sullo stampato.
vedi: https://diyprojects.io/hack-sonoff-s26-wifi-smart-plug-tasmota-firmware-installation/#.XtaeYzozZPY

Uso di Sonoff-basic



Esistono 3 versioni di Sonoff basic, ma tutte usano un connettore a 4 o 5 pin.
1.
Note: *Sonoff basic dispone di un solo LED, quello VERDE*



Sonoff-basic è particolarmete adatto ad essere montato direttamente su generatori di ozono privi i timer.

2) firmware esp_MQTT (windows)

esp_MQTT non deve essere compilato: *martin_ger*, oltre a pubblicare i sorgenti, pubblica anche i file 'bin' pronti per essere caricati su Sonoff.

- Scaricare ed installare esptool (https://github.com/espressif/esptool, necessario per copiare il firmware. (richiede anche python).

- Scaricare da https://github.com/martin-ger/esp_mqtt/tree/master/firmware i 2 file 'bin' con il firmare compilato, metterli nella stessa dir di *esptool.py (*nel mio caso in `C:\Program Files (x86)\Python38-32\Scripts\)`

Disconnettere Sonoff (S20/S26/basic) dalla rete a 220V !!!

- Collegare FTDI via USB al PC. Controllare il numero di porta usato (Pannello di controllo, Sistema, Gestione dispositivi, Porte) Nel mio caso: COM6.

- Controllare che FTDI sia a 3.3 V e poi collegare Sonoff.

- Tenere premuto il pulsante su Sonoff *prima* di fornire alimentazione a Sonoff (usare l'interruttore presente sul collegamento FTDI-Sonoff, o, più scomodo, scollegare e ricollegare il cavetto VCC → 3.3V). Rilasciare il pulsante su Sonoff 1 o 2 secondi dopo aver fornito tensione.

Usare il seguente comando per pulire la scheda Sonoff (COM6 può cambiare). Io uso un piccolo file BAT (vedi `esptool/write-erase-flash.bat)`:

```
esptool.py-script.py  --port COM6 erase_flash
```

Poi usare il seguente comando per copiare il firmware in Sonoff-basic (COM6 può cambiare). Io uso un piccolo file BAT (vedi `esptool/write-esp-Sonoff.bat)`:

```
esptool.py-script.py --port COM6 write_flash -fs 1MB -fm dout 0x00000 0x00000.bin
0x10000 0x10000.bin
```

nota: *Il firmware originale Itead è ripristinabile con qualche complicazione (vedi* https://wiki.almeroth.com/doku.php?id=projects:sonoff *). Ma in genere non è necessario: infatti lo script "script.sonoff" di martin-get emula perfettamente il funzionamento originale di Sonoff Basic (vedi –* https://github.com/martin-ger/esp_mqtt/tree/master/scripts*).*

Nota: *Non sempre la scrittura del firmware riesce al primo colpo. Provare più volte.*
Il problema è spesso imputabile alla poca corrente fornita da FTDI a 3.3V. Una semplice soluzione (per me ha funzionato) è quella di inserire un condensatore elettrolitico tra VCC e GND (ho usato 100 µF). Una soluzione più drastica è usare un alimentatore separato a 3.3 V (e.g. uno step-down 5⟶3.3). Un'altra soluzione è alimentare la scheda Sonoff a 5V, collegandosi a massa ed ad un lato della bobina del relay, quella con il (+) del diodo (striscia nera).

NON ALIMENTARE A 220V Sonoff quando FTDI è collegato!! *Una svista (un cacciavite che cade sul circuito) e il vostro PC (nel miglior caso) è da buttare! Ma potreste anche rischiare la vita!!*

nota: Per maggiori informazioni, per Linux etc. vedi martin-ger: "Building and Flashing": *https://github.com/martin-ger/esp_mqtt*

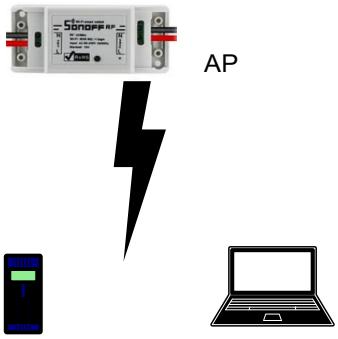
3) Uso: architetture di rete

Standalone

Il timer è usato da solo, controllandolo con il suo pulsante.

- E' usabile solo il modo AUTO, immediato oppure ad orario.
  La prima pressione attiva AUTO, la seconda START (dopo un minuto)
- I tre valori: **ON time, PDM** e **Start time** devono essere programmati precedentemente con una delle app. Sono memorizzati dal timer, anche in assenza di tensione.

SoloAP



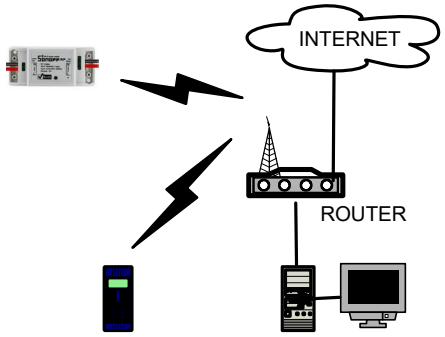
Il timer è configurato come AP, i client devono avere il WiFi connesso direttamente all'AP del timer.

- SSID: "timerPDM" (o timerPDM01, timerPDM02 nel caso di più timer)
- Password: none (open)
- L'ora attuale deve essere impostata dall'app.

Vantaggi:

- Portatile, si possono usare tutte le app.
- Il broker MQTT ha l'I.P. fisso 192.168.4.1, port 1883
- Si può usare, come telecomando dedicato, uno smartphone senza SIM.

LanRouter



Il timer è configurato come STA (+AP) e si collega al WiFi del router.

- Tutti i dispositivi della rete possono connettersi al timerPDM
- L'I.P. del broker MQTT è dato dal router, port 1883
- La funzione AP è attiva: è possibile collegarsi WiFi direttamente al timer (come nel caso **soloAP**)

Vantaggi:

- Permette l'aggiornamento OTA dello script
- Maggiore copertura WiFi, inoltre non serve il WiFi nei dispositivi collegati via LAN
- Il timer può usare un server NTP per avere l'ora esatta. Gestire con timezone l'ora legale (DST)
- Se su un PC della rete è attivo *node-red* e il flow "***timerPDM-red***", gli smartphone o altri dispositivi in rete possono usarlo connettendosi a `http://<IP_pc_red>:<port_red>/ui`
- L'accesso è locale via WiFi/LAN, ma può essere reso globale da Internet, usando un servizio gratuito di DNS dinamico (esempio ***noip***: https://www.noip.com/).

La configurazione iniziale è ***LanRouter,*** necessaria per usare la funzione OTA e caricare lo script ma tramite le app è possibile riconfigurare il timer (solitamente non necessario). Per disabilitare il modo ***LanRouter:*** con ***ssid*** impostare un nome qualunque, ma NON impostare ***pass***: questo blocca la funzione STA e la ricerca a vuoto di un router che non esiste. Per abilitare il modo ***LanRouter*** o cambiare il router: nna volta connessi ***soloAP***, con ***ssid*** impostare il nome dell'*AP del router*, e con ***pass*** impostare la *password del router.* Il collegamento è automatico.

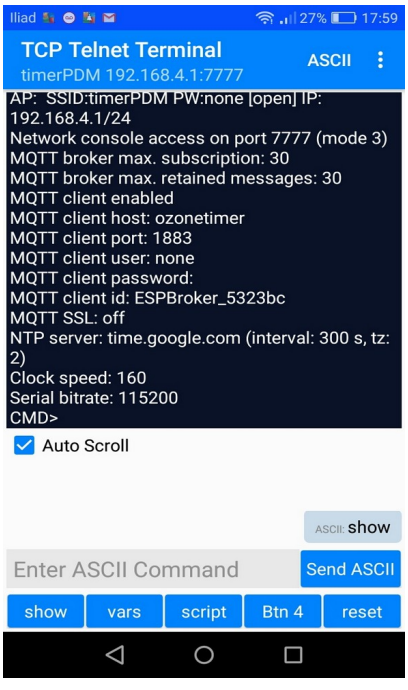
4) Configurazione, primo run

```
D:\Program Files (x86)\Python38-32\Scripts>esptool.py-script.py --port COM6 writ
e_flash -fs 1MB -fm dout  0x00000 0x00000.bin 0x10000 0x10000.bin
esptool.py v2.8
Serial port COM6
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: cc:50:e3:53:23:bc
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Flash params set to 0x0320
Compressed 42416 bytes to 29051...
Wrote 42416 bytes (29051 compressed) at 0x00000000 in 2.6 seconds (effective 131
.1 kbit/s)...
Hash of data verified.
Compressed 310940 bytes to 226936...
Wrote 310940 bytes (226936 compressed) at 0x00010000 in 23.7 seconds (effective
105.1 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

D:\Program Files (x86)\Python38-32\Scripts>pause
Premere un tasto per continuare . . .
```

Quando i messaggi ci informano della corretta scrittura della memoria flash, spegnere e riaccendere Sonoff con l'interruttore (*senza* premere il pulsante), lasciando collegato FTDI.

1. *La configurazione iniziale di base può essere fatta con la console seriale.*

   Usare FTDI per il collegamento ed il terminale Window *puTTY* (https://putty.org/) con la configurazione: 'serial', COM6, 115200.
   Premendo [ENTER] deve apparire il prompt '`CMD>`' di *esp_MQTT* che ci conferma il corretto funzionamento. Ora possiamo dare tutti i comandi console accettati da *esp_MQTT (per i comandi usabili vedi* https://github.com/martin-ger/esp_mqtt)



2. *Alternativa con Telnet*

   Se si vuol modificare la configurazione, senza usare FTDI, per esempio con il timer chiuso, si può usare un portatile con WiFi, e collegarsi con il timer: AP "MyAP" (default iniziale) oppure "timerPDM" (dopo la configurazione).
   Usare ora PuTTY, installato preventivamente sul portatile, per connettersi (Telnet, IP: 192.168.4.1, port: 7777).

   Oppure si può usare anche uno smartphone Android e l'app ***TCP Telnet Terminal*** (o altro client telnet) : https://play.google.com/store/apps/details?id=project.telnettcpterminal&hl=en

Per i nostri scopi, vogliamo che ***il timer*** si configuri come STA e AP:

```
CMD>set system_output 2
CMD>set speed 160
CMD>set ssid <router_name>
CMD>set password <router_pass>
CMD>set ap_ssid timerPDM  (o timerPDM01, timerPDM02)
CMD>set mqtt_host ozonetimer

CMD>save
CMD>reset
```

Dopo il reset ***Sonoff+esp_MQTT*** sarà raggiungibile come AP con nome ***timerPDM*** (per semplicità senza password, ma volendo può essere configurata una password con:

CMD>set ap_password <secret>).

Possiamo scollegare definitivamente FDTI.

La scheda Sonoff è pronta: lo script con la logica di funzionamento sarà caricato via OTA, quindi possiamo richiudere con le viti l'interruttore S20.

nota: *usare un filtro per carico induttivo solo per elevati assorbimenti con basso fattore di potenza. Collegarlo sui morsetti di uscita, in parallelo al carico, oppure direttamente sul generatore di ozono: aumenta la vita utile del relay.*
Con i miei due generatori di ozono (11W, pf: 0.70 e 103W, pf: 0.95) il filtro non è necessario.

5) logica di funzionamento (script)

Il file "`timerPDM02.cde`" è lo script usato per il Timer PDM.

Per creare lo script ho usato *Notepad++* (https://notepad-plus-plus.org/), ottimo text editor per programmatori. Nota: *il file script deve essere codificato ANSI*.

Per rimanere nei limiti di memoria (4'000 byte), nello script ho cercato di minimizzare i caratteri usati:

1) ridotta l'indentazione
2) eliminati i commenti
3) limitatati i '`println`' di debug
4) ridotti i comandi '`config`' iniziali
5) usati nomi di 2-3 lettere per le variabili.
6) riusato la variabile $tmp più volte.
7) eliminati i controlli sui valori di configurazione ricevuti
8) eliminati spazi, apici ["] e parentesi [(],[)] superflui
9) usato formato UNIX per caratteri di fine linea

Con queste avvertenze lo script è circa 3990 byte, ma risulta di non agevole lettura.

Personalizzazioni nello script `timerPDM02.cde`:

```
2       %CMD>script http://192.168.178.23:88/www/sonoff/timerPMD02.cde
                                        comando OTA: adattare all'URL usata
4       config ntp_server time.google.com   server NTP usato (internet)
5       config ntp_timezone 1               default timezone (si può cambiare dall' app)
6       config @6 /ozone/timerpdm/1         per cambiare ID (1) in caso di più timers
```

Nota: *In modalità **SoloAP** il clock del timerPDM deve essere aggiornato dalle app. In questo caso è usata direttamente l'ora inserita e **timezone** è posto automaticamente a 0.*
*In modalità **LanRouter** il timerPDM cerca di usare il server NTP definito nello script a linea 4. In questo caso deve essere aggiornato **timezone** con le app, per visualizzare l'ora locale esatta e tener conto del DST.*

Nota: *Se l'ora indicata è "99:99:99" l'aggiornamento NTP è fallito. In alcuni router o firewall occorre esplicitamente abilitare l'accesso alla porta 123 (default per NTP). E' comunque sempre possibile l'impostazione del clock tramite le app.*

Il modo più semplice per caricare uno script in ***Sonoff+esp_MQTT*** in modo OTA è quello di usare un server WEB.

In fase di sviluppo è comodo avere il server locale, nel PC principale. Nel mio caso (Windows) ho usato il server portatile "*wpp_pampa*" http://www.winpenpack.com/en/index.php contenuto in una chiavetta USB "*winPenPack*". Altrimenti si può usare un qualunque server *WAMP/LAMP* oppure anche un server remoto su Internet, ad esempio scaricando direttamente da Github. Per Linux vedi https://github.com/martin-ger/esp_mqtt,

In ogni caso il Timer deve essere in modo ***LanRouter***, per poter raggiungere il WEB server.

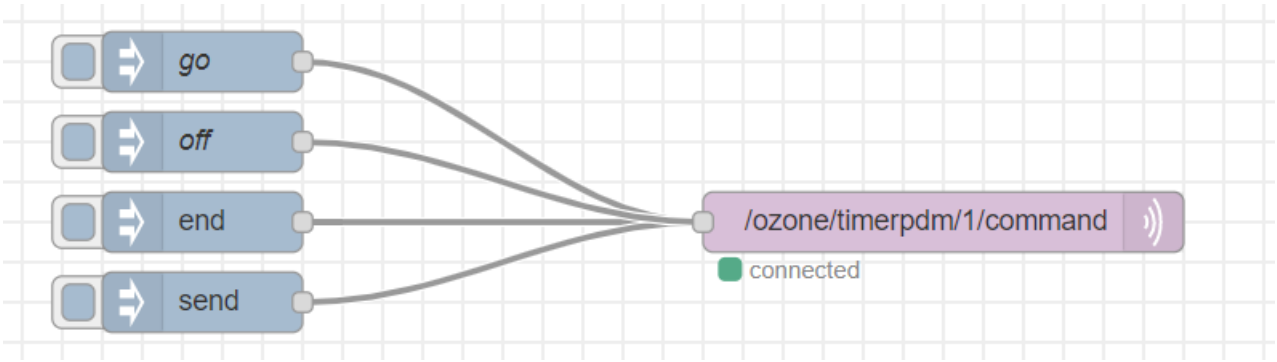
Usando la console remota (*telnet: Putty o TCP Telnet Terminal*) dare il comando

`CMD>script <url_to_script>`

Per comodità il comando da usare sulla console è scritto nella seconda riga del file: modificarlo per adattarlo alla configurazione usata, poi usare copia-incolla.

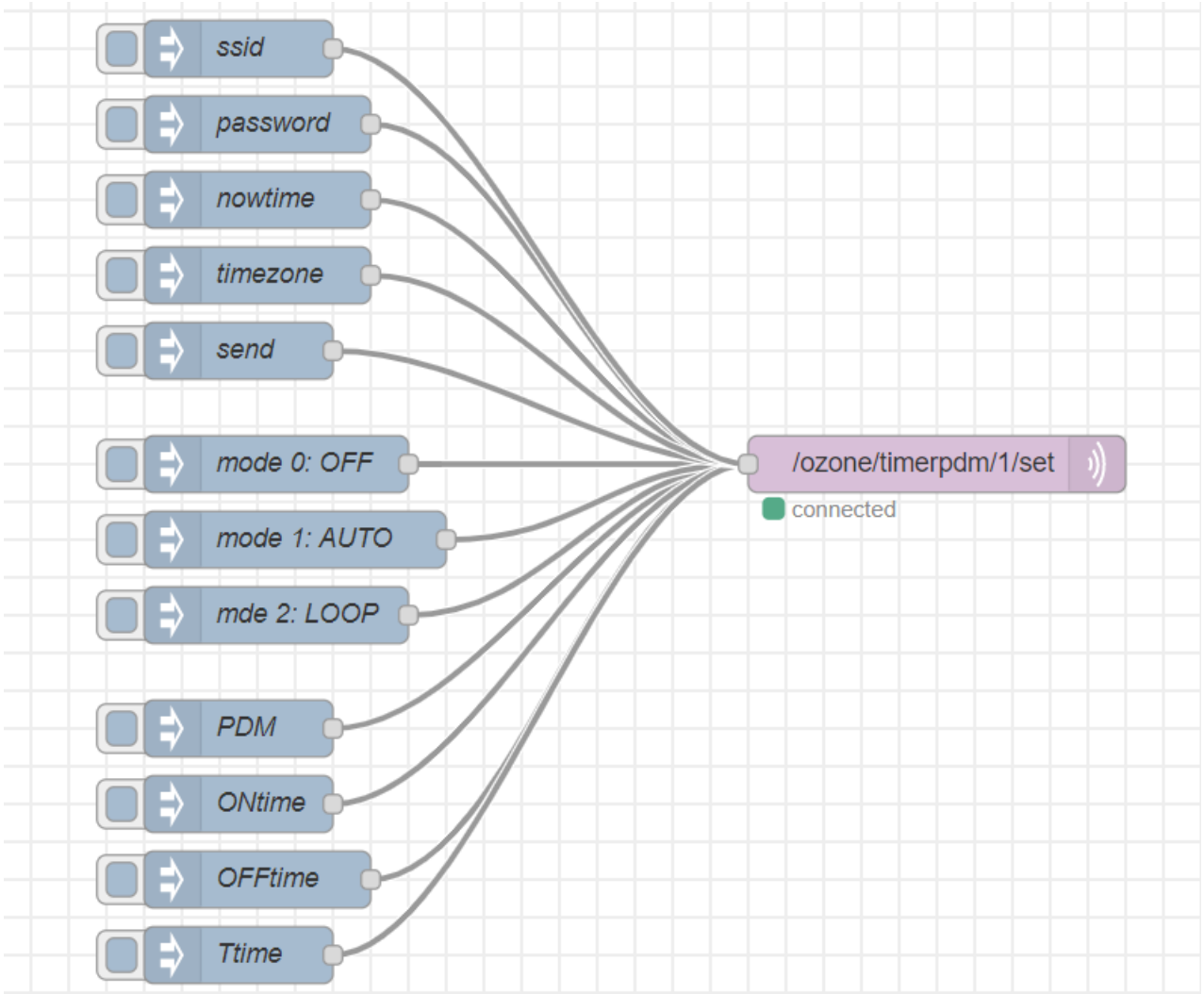
Riferimento: MQTT messaggi input

Sintassi e semantica dei messaggi MQTT inviati/eseguiti dal timer.



Subscribe Comandi: `/ozone/timerpdm/<num>/command`

`go` : START non condizionato, solo in modo AUTO e LOOP
`off` : STOP non condizionato, mantiene il modo attuale (AUTO oppure LOOP)
`end` : STOP non condizionato, riporta in modo OFF (standby)
`send` : (refresh) nessuna azione, rinvia status/output



Subscribe Configurazione: `/ozone/timerpdm/<num>/set`

`{"data":"ssid", "value":"<wifi-id>"}` : nome del router a cui collegarsi per internet. Blocca STA: resta attivo solo AP

`{"data":"pssw", "value":"<wifi-password>"}`: password di accesso al router, attiva STA

`{"data":"timezone", "value":<h>}` : set timezone. <h> é aggiunto all'ora fornita da NTP

`{"data":"nowtime", "value":"<HH:MM:SS>"}` : mancando NTP, aggiusta orologio (e mette timezone = 0)

`send` : (refresh) nessuna azione, rinvia status/config

`{"data":"mode", "value":0|1|2 }` : modo di funzionamento (0..2)

`{"data":"PDM", "value":<2..100> }` : controllo generatore, in % rispetto al max.

`{"data":"ONtime", "value":<min> }` : durata accensione, in minuti (AUTO e LOOP)

`{"data":"OFFtime", "value":<min> }` : durata spegnimento, in minuti (solo LOOP)

`{"data":"Ttime", "value":"<HH:MM>" }` : orario di accensione (solo AUTO)

nota: *I valori inviati sono cablati nelle proprietà dei nodi, quindi questi flow servono solo da esempio e test.*

Riferimento: MQTT messaggi output

Publish Status-configurazione: `/ozone/timerpdm/<num>/status/config`
(inviato dopo ogni modifica e dopo 'config send'):

```
{"mode":<0|1|2>,"Ontime":<min>,"OFFtime":<min>,"PDM":<%>,"Ttime":"HH:MM"}
```

Publish Status-output: `/ozone/timerpdm/<num>/status/output`
(inviato dopo ogni comando e dopo 'command/send')

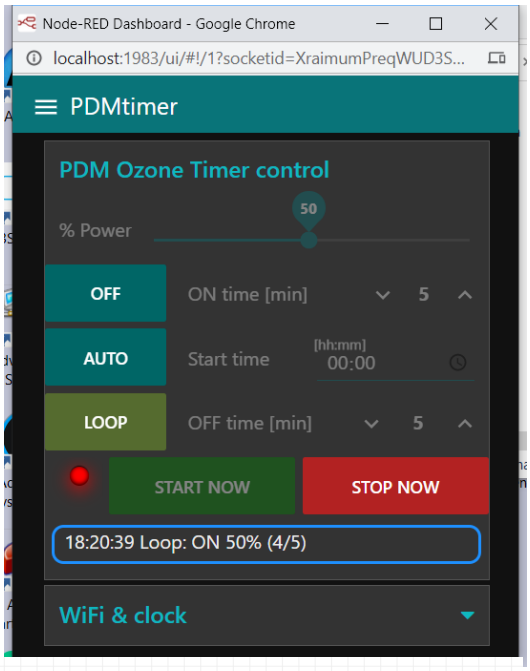
```
{"Time":"<timestamp>","mode":<0|1|2>,"PDM":<%>,"output":<0|1>,"count":<min>}
```

Publish Status-info: `/ozone/timerpdm/<num>/status/info`
(auto ogni secondo, array: [0] stringa double-escaped per visualizzazione
[1] led status (0..2))

```
["<timestamp> <mode>: <output> [<PDM>% (<count>/<time>])",<led_status>]
```

TimerPDM-red: node-red client

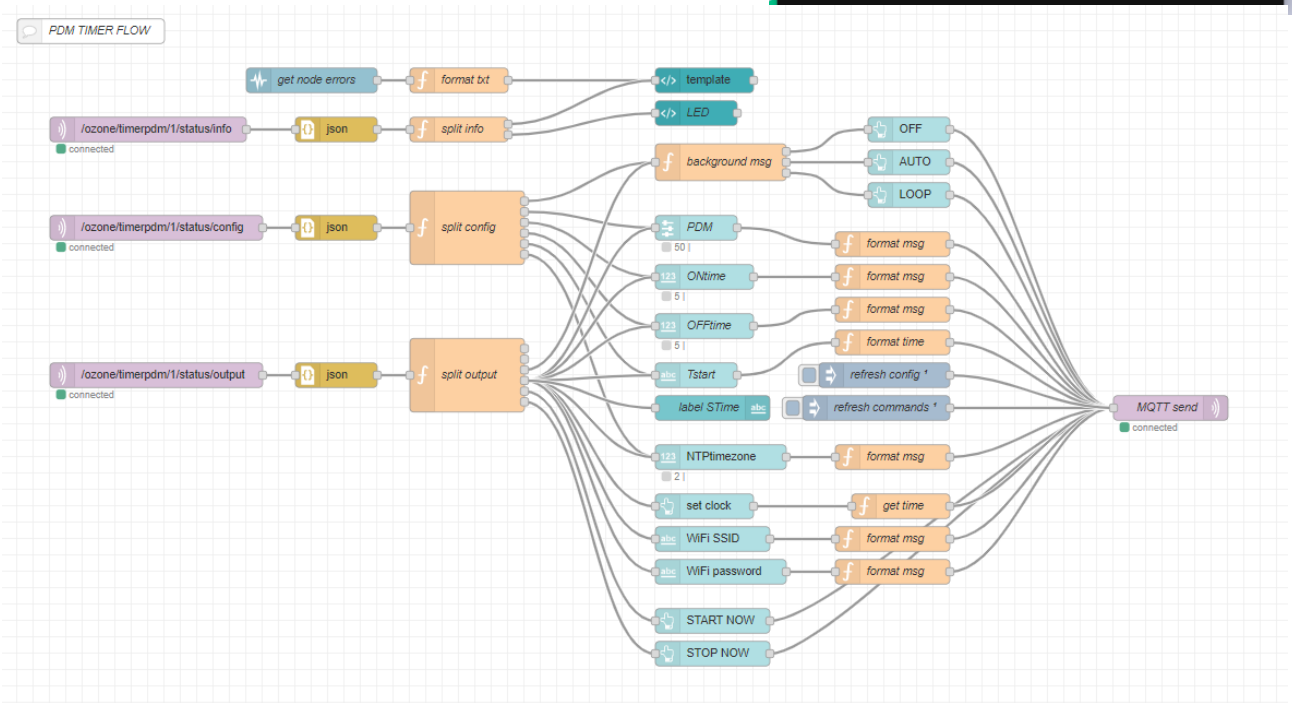
Questa app è adatta ad applicazioni fisse. Poiché node-red si comporta come un WEB server, tutti i dispositivi collegati alla rete, compresi gli smartphone, posso utilizzarla con un browser



Non è stata implementata la memorizzazione dei dati su DB, ma, se desiderata, è semplice aggiungerla al flow.

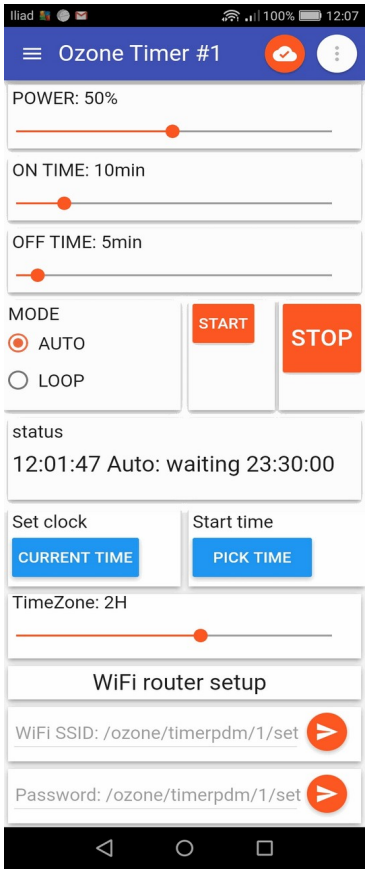
Node-red può essere installato su ogni PC (Win/Linux) ed anche su *Raspberry pi 3* se si desidera un computer dedicato a basso costo (per installazione di *node-red*, vedi https://nodered.org/docs/getting-started/installation)

- Se non installata di default, importare la library '`node-red-dashboard`'.
- Copiare nel clipboard il contenuto del file "`node-red/timerPDM.json`" ed importarlo in un nuovo flow in *node-red.*
- Configurare l'accesso al ***broker MQTT*** del timer: <IP_timer>:1883



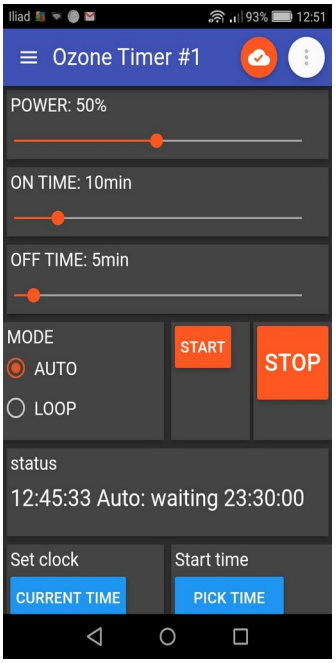
Per comodità nella dir `node-red` ci sono anche due script 'BAT' che uso in ambiente Windows per lanciare *node-red*. Questi script permettono di avere numerosi flow contemporaneamente, usando port e path differenti (port: 1980,1981…)
Nel mio caso la dir che contiene i flow di timerPDM è `D:/node-red/flow-1983`, ovviamente dovete modificare i file BAT e HTML per adattarli ai path che usate. I file principali sono:
`start-red-3ui-ozone.bat`, per lanciare la sola interfaccia utente (usa `dashboard-1983.html`).
`start-red-debug3.bat`, per lanciare node-red.
Lanciate *node-red* usando `start-red-debug3.bat`, e poi importate il file `timerPDM.json` in un nuovo flow: sarà salvato automaticamente nella dir: `D:/node-red/flow-1983`.

PDM timer: Android client



Questo semplice client permette di controllare il timer usando uno smartphone Android collegato WiFi, sia in modo ***soloAP*** che in modo ***lanRouter.*** Soluzione comoda e soprattutto portabile, agisce come un telecomando, consentendo di attivare il generatore di ozono da una zona sicura.

Per questo client ho scelto "*IoT MQTT Dashboard*" https://play.google.com/store/apps/details?id=com.thn.iotmqttdashboard free, perché è ricco di input ed esteticamente gradevole, ma ci sono anche altre alternative. Potete cercare in *Google Play,* l'importante è che gestisca JSON payload.



Realizziamo prima la versione **soloAP.**

Purtroppo non è possibile importare un dashboard in *IoT MQTT Dashboard* da un file: occorre crearlo interattivamente.
Fare riferimento all'help della app, ed alle figure e commenti seguenti:

1) *Creare una nuova connessione*
Pulsante più (+) rosso.

2) *Edit Connection* (v. img)

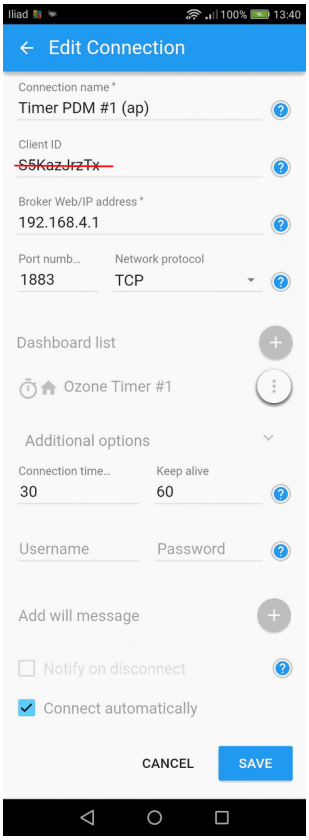
- Lasciare in bianco Client ID: é assegnato automaticamente
- IP predefinito (**soloAP**): 192.168.4.1, protocollo TCP

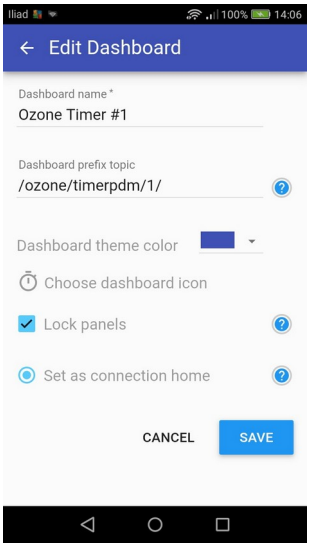
nota: *in modo **soloAP** il WiFi dello smartphone deve essere collegato all'AP del timer, SSID **timerPDM.***

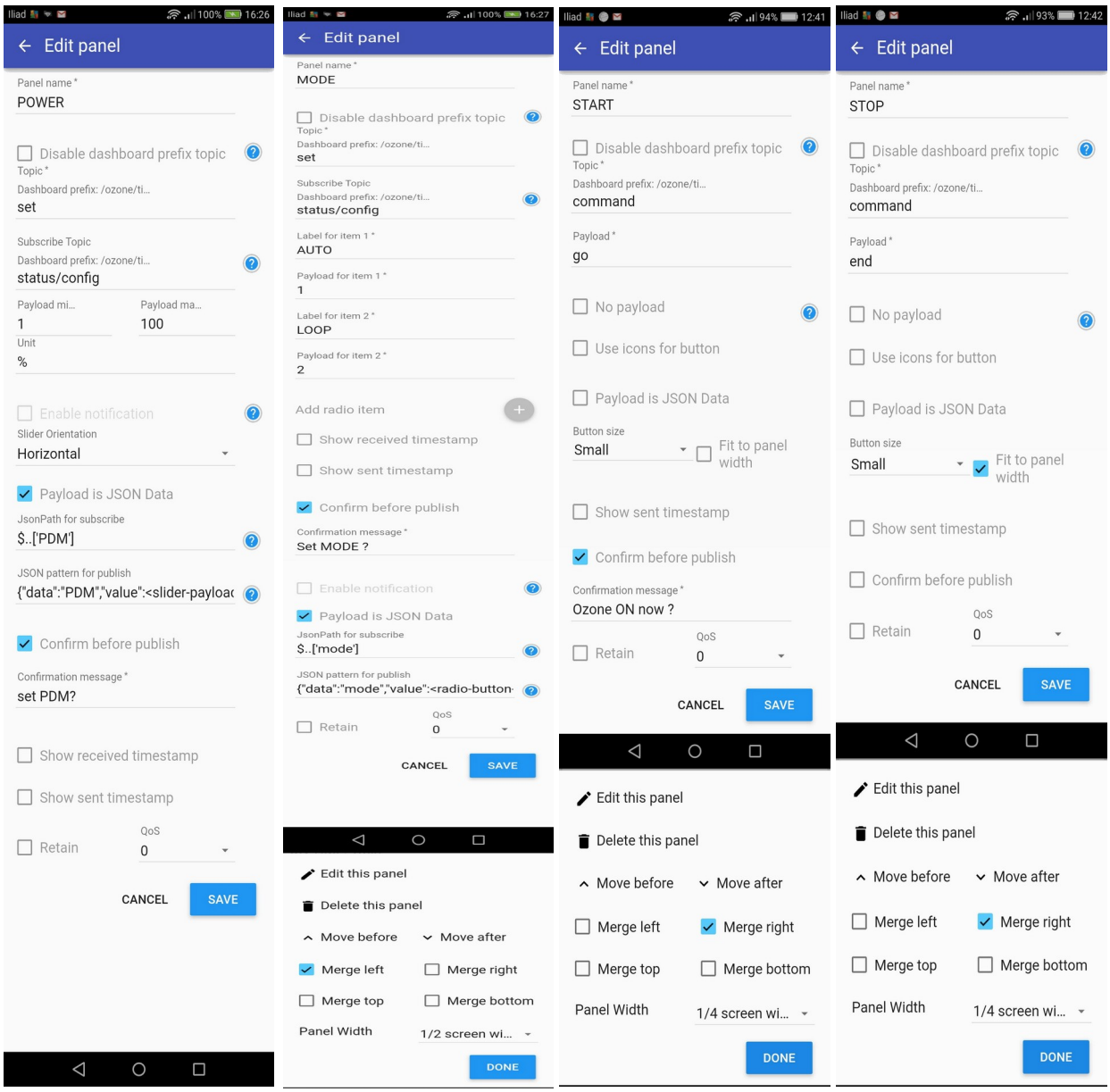
3) *Edit Dashboard* (v.img)

- Il "Dashboard prefix topic" rende più semplice la successiva definizione dei pannelli: /ozone/timerpdm/1/
- "lock panels" attivare solo alla fine.

4) *A*ggiungere i seguenti panel (Pulsante rosso):







4.1 *slider POWER (v. img)*

Topic: set
Subscrive topic: status/config
min, max, unit: 1, 100, %
JsonPhat for subscribe: $..['PDM']
JSON pattern for publish: {"data":"PDM","value":<slider-payload>}
confirm: set PDM?

4.2 *slider ON TIME (analogo a POWER)*

Topic: set
Subscrive topic: status/config
min, max, unit: 1, 60, min
JsonPhat for subscribe: $..['ONtime']
JSON pattern for publish: {"data":"ONtime","value":<slider-payload>}
confirm: set ON time?

4.3 *slider OFF TIME (analogo a POWER)*

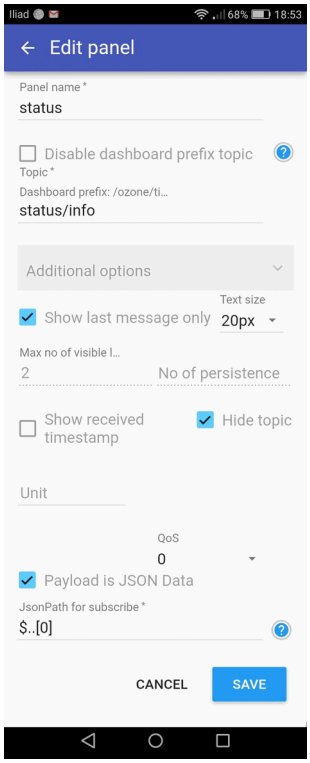
Topic: set
Subscrive topic: status/config

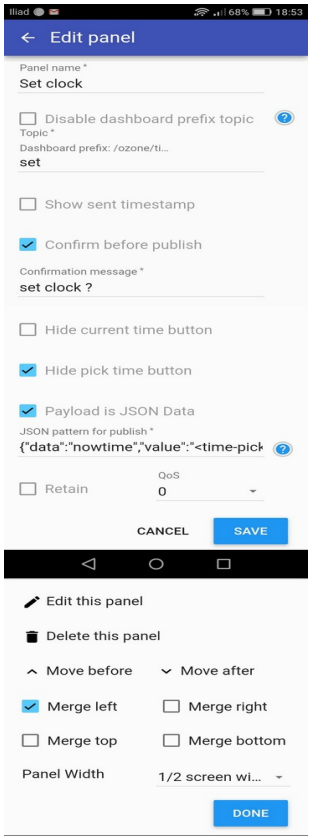
min, max, unit: 1, 60, min
JsonPhat for subscribe: $..[‘OFFtime’]
JSON pattern for publish {“data”:”OFFtime”,”value”:<slider-payload>}
confirm: set OFF time?

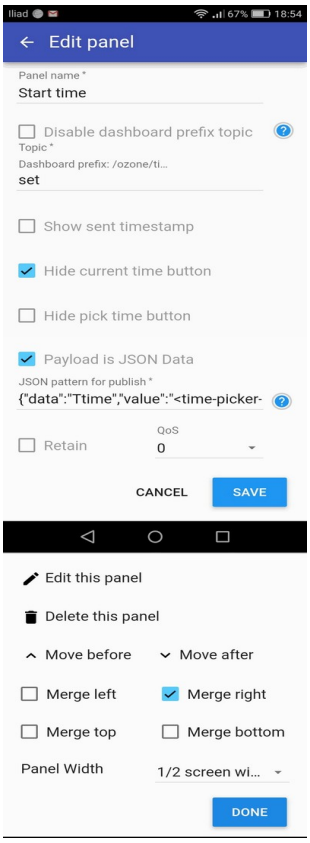
4.4 *Radio.button MODE (v. img)*
Topic: set
Subscrive topic: status/config
JsonPhat for subscribe: $..[‘mode’]
JSON pattern for publish:
{“data”:”mode”,”value”:<radio-button-payload>}
dimensioni, ½; a sinistra

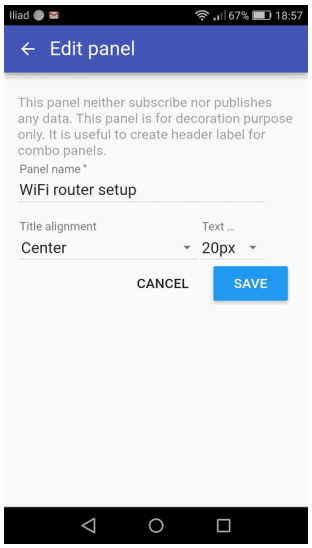
4.5 *Button START (v. img)*
Topic: command
Payload: go
small, *dimensioni: ¼; a destra*
confirm: Ozone ON now ??

4.6 *Button END (v. img)*
Topic: command
Payload: go
medium, *dimensioni: ¼; a destra*









4.7 *Text Log: status (v. img)*
Topic: status/info
JsonPhat for subscribe: $..[0]

4.8 Time Picker *Set clock (v. img)*

Topic: set
Payload is JSON data: {“data”:”nowtime”,”value”:“<time-picker-payload>”}
dimensioni, ½; a sinistra

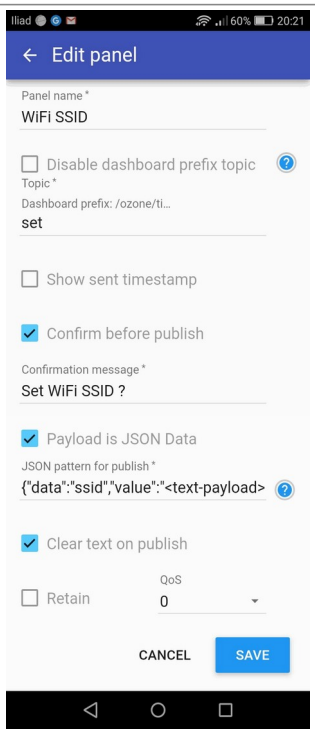
4.9 Time Picker *Start time (v. img)*
Topic: set
Payload is JSON data:
{“data”:”Ttime”,”value”:“<time-picker-payload>”}
dimensioni, ½; a destra

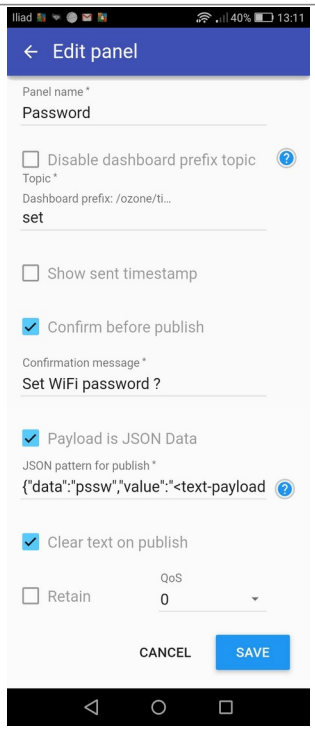
4.10 Layout Decorator *(v. img)*
- *usato come separatore*

4.11 Text Input *WiFDi SSID (v. img)*
Topic: set
Payload is JSON data:
{“data”:”ssid”,”value”:“<text-payload>”}
confirm: set WiFi SSID?

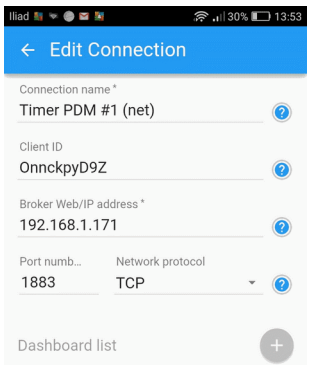
4.12 Text Input *Password (v. img)*
Topic: set
Payload is JSON data:
{“data”:”pssw”,”value”:“<text-payload>”}
confirm: set WiFi password?

5) Questi 12 panel formano la app. Quando tutto funziona potete bloccare i panel in *Edit Dashboard*.





Ovviamente la soluzione qui proposta è la più semplice: può essere personalizzata in infinite maniere.



Se desiderate un’app su smartphone anche per il modo **LanRouter** non dovete ricominciare da capo, ma potete usare il comando “*clone connection*” che duplica una connessione. Poi in *Edit Connection* effettuare i seguenti cambiamenti:

1. cambiare *Connection name*
2. cancellare il *Client ID:* ne verrà generato uno nuovo
3. cambiare *Broker Web/IP address*, usando l’IP assegnato dal router

Nota: *Ritengo si trovino app con funzione di MQTT+dashboard anche per S.O. diversi da Android, importante è che gestiscano JSON payload.*