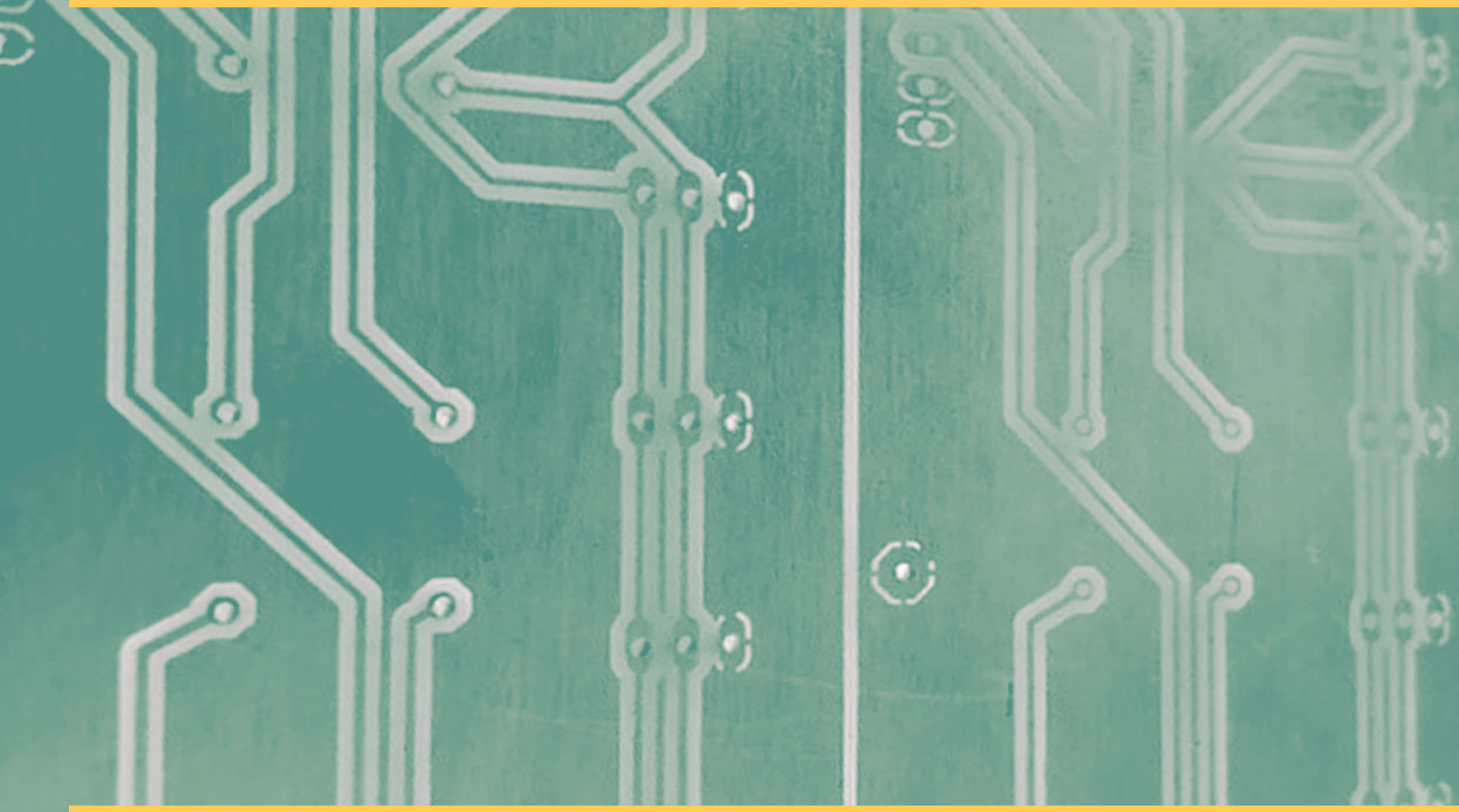
ISSN 2039-7941
anno 2022 numero 444



RAPPORTI TECNICI INGV

Un sistema modulare hardware/software a basso costo per il monitoraggio della sala CED

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

444

REGISTRAZIONE AL TRIBUNALE DI ROMA N.174 | 2014, 23 LUGLIO

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

ISSN 2039-7941
anno **2022** numero **444**

RAPPORTI TECNICI INGV

Un sistema modulare hardware/software a basso costo per il monitoraggio della sala CED

A low-cost hardware/software modular system for datacenter monitoring

Stefano Chiappini

INGV | Istituto Nazionale di Geofisica e Vulcanologia, Sezione Geomagnetismo, Aeronomia e Geofisica Ambientale

Accettato 2 dicembre 2021 | *Accepted 2 December 2021*

Come citare | *How to cite* Chiappini S., (2022). Un sistema modulare hardware/software a basso costo per il monitoraggio della sala CED. Rapp. Tec. INGV, 444: 1-30, https://doi.org/10.13127/rpt/444

In copertina Circuito stampato a cui collegare le sonde DS18B20; elaborazione di B. Angioni | *Cover Circuit board where the DS18B20 probes are connected; processed by B. Angioni*

444

INDICE

Riassunto

In questo documento viene presentata la progettazione e realizzazione di un sistema, a basso impatto economico, per il controllo dei parametri critici di funzionamento di una sala CED, quali le temperature nei punti più rappresentativi del locale dove sono ospitati i *rack* dei *server*, lo stato di funzionamento del sistema di raffreddamento e le grandezze fisiche inerenti l'alimentazione elettrica, al fine di valutarne il grado di efficienza e di qualità. Tale soluzione, personalizzata specificatamente per la nuova sala CED del Centro Nazionale Dati (IT-NDC) ubicato presso la sede distaccata di Roma 2, è in realtà concepita come una struttura modulare, alla quale possono essere facilmente aggiunte componenti, sia *hardware* che *software*, finalizzate al controllo di parametri non esplicitamente contemplati in questo specifico lavoro.

Abstract

In this paper, the design and development of a low-cost monitoring system is reported, in order to control the critical operating parameters of a data center, such as temperatures in the most representative spots of the server room, the cooling system efficiency, power supply related physical measurables useful to evaluate its quality and efficiency level. This solution, specifically customized for the new data center of the Italian National Data Center (IT-NDC) located at the Roma 2 branch office, is actually designed with a modular structure, where hardware and software components can be easily added, even to detect parameters not explicitly planned in this specific work.

Keywords CED; Monitoraggio; Sistema modulare | Datacenter; Monitoring; Modular system.

Introduzione

L'attività descritta in questo documento, è stata originata dalla necessità di realizzare un nuovo sistema di controllo dei parametri della sala CED operante presso la sede distaccata della sezione Roma 2, sita in zona Flaminio a Roma Nord. Nonostante il vecchio sistema abbia funzionato egregiamente fino al giorno della sua disattivazione, un suo rinnovamento si è reso indispensabile a causa della migrazione della sala *server* all'interno della sede (dal primo piano al piano terra), nell'ambito del più generale piano di ampliamento ed ammodernamento del CED. Il cambio di locale avrebbe comportato lo spostamento e la stesura di nuovi e più lunghi cavi speciali per le termocoppie, nonché di una sostanziale modifica del programma di acquisizione dati e gestione degli allarmi, sviluppato nel 2006 utilizzando un ambiente di sviluppo oramai obsoleto (MS Visual Studio). Pertanto, la necessità di introdurre la possibilità di controllare anche altri parametri, eventualmente aggiunti successivamente a sistema già funzionante, ha spinto verso una completa riprogettazione del sistema, il quale fosse basato su più moderne tecnologie *hardware* e *software*, con l'impiego per quanto possibile di componentistica a basso costo di facile reperibilità, gestibile con l'ausilio di soluzioni *software* preferibilmente *open source*, modulare, facilmente manutenibile e scalabile per future espansioni.

1. Obiettivi

I requisiti principali che hanno indirizzato la progettazione sono stati sostanzialmente i seguenti:

- possibilità di aggiungere sensori di temperatura secondo necessità, superando il limite numerico imposto dal vecchio modulo di conversione A/D della National Instruments a cui erano collegate le termocoppie nel vecchio sistema;
- capacità di controllare parametri legati alla qualità dell'alimentazione di rete elettrica, oltre che alla semplice presenza/assenza come nel sistema precedente, in quanto la zona è tradizionalmente servita da una linea abbastanza instabile, specialmente nei mesi estivi, e si verifica l'entrata in funzione dell'UPS anche non in presenza di veri e propri *blackout*, bensì a causa di microinterruzioni e frequenti sottotensioni;
- componenti elettronici standard e di facile reperibilità sul mercato, possibilmente a basso costo, sia per i *budget* sempre più ridotti sia per rendere più agevole l'approvvigionamento di eventuali ricambi in futuro;
- componente *software* basata su ambienti di sviluppo e soluzioni *open source*, per evitare di dipendere in futuro da scelte e strategie di mercato dettate da società private (fenomeno del *lock in*).

2. Componenti *hardware*

2.1 Sensori di temperatura

La prima decisione ha riguardato quali sonde termiche utilizzare e la scelta è caduta sugli integrati DS18B20 (Figura 1). Come primo vantaggio, non è più indispensabile la presenza un convertitore A/D (*analog-to-digital*) dedicato, in quanto la conversione avviene direttamente all'interno dell'integrato DS18B20 ed il dato è trasmesso già in forma numerica all'esterno. Questo rende possibile intervenire in maniera più rapida, efficiente ed economica in caso di guasti, i circuiti di acquisizione sono indipendenti ed il malfunzionamento di una sonda non ha impatto negativo sulle altre.

Figura 1 La nuova sonda digitale di temperatura adottata.
Figure 1 *The new digital thermometer selected.*

Come si può osservare in Figura 1, questi termometri digitali richiedono, oltre ad un'alimentazione compresa tra 3V e 5.5V con relativa massa, un solo cavo per la connessione dati (protocollo "*1-Wire bus*", inventato dalla Dallas Semiconductors), e comunicano serialmente con un microprocessore esterno che si occupa di interrogare il sensore ed ottenere la lettura di temperatura. Ogni integrato [Maxim Integrated, 2021] ha cablato al suo interno un codice

identificativo univoco a 64 bit, che consente l'indirizzamento diretto da parte del microprocessore; per questo motivo tutti gli integrati possono condividere lo stesso *1-Wire bus* senza rischio di collisioni o interferenze, semplificando il cablaggio. La risoluzione di partenza (all'accensione) delle misure è di 12 bit, che corrisponde ad una discretizzazione pari a 0.0625 °C di differenza minima tra due valori diversi, mentre l'errore di misura nell'intervallo di temperature di interesse è pari a ± 0.5 °C.
In Figura 2 è riportato lo schema per un corretto cablaggio dell'integrato DS18B20, il quale necessita solamente di una resistenza (R1) cosiddetta di *pull-up* da 4.7 kΩ tra l'alimentazione V_{cc} e la linea dati, oltre ai tre fili indicati in figura. Per consentire il collegamento di più sensori utilizzando un unico *1-Wire bus*, è stata disegnata e realizzata una basetta in rame (Figura 3), il cui schema è stato sviluppato mediante la versione *free download* del programma Autodesk EAGLE 9.4.2 [Autodesk, 2019], mentre la realizzazione delle piste in rame tramite il tradizionale procedimento che fa uso di un reagente (cloruro ferrico) per corrodere le porzioni di metallo appositamente non protette dall'inchiostro.

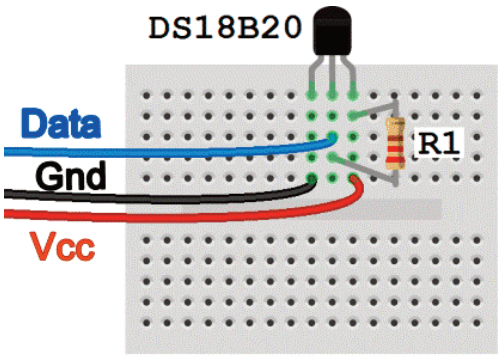


Figura 2 Schema di collegamento del DS18B20.
Figure 2 *Connection diagram for the DS18B20.*

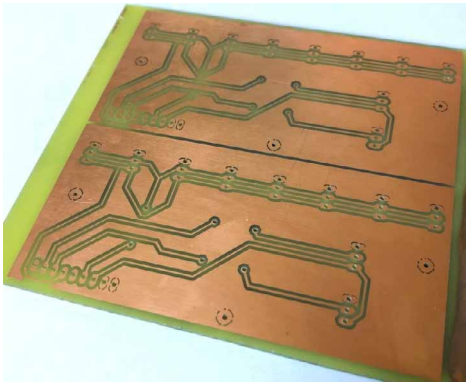


Figura 3 Circuito stampato a cui collegare le sonde DS18B20.
Figure 3 *Circuit board where the DS18B20 probes are connected.*

Nella versione completa riportata in Figura 4, sono visibili gli 8 connettori bianchi a 3 pin scelti per il collegamento degli integrati DS18B20, i quali consentono di connettere e disconnettere i sensori anche a sistema alimentato, senza che ciò possa provocare interruzioni alla porzione di impianto non interessato. Ho optato per questo tipo di attacco M-F (Figura 5) in quanto compatto e provvisto di un fermo laterale di sicurezza, il quale impedisce l'inserimento nel verso errato, in cui verrebbero invertiti la sorgente di alimentazione con la massa. Nella parte destra vi sono anche due connettori a 3 pin neri e due resistenze da 1 kΩ, questa parte del circuito serve a pilotare due interruttori optoelettronici SRD-05VDC-SL-C, i quali possono essere controllati da una tensione continua a 5V e sono in grado di aprire/chiudere circuiti con correnti alternate fino a 250V e 10A. Questi interruttori si sono rivelati molto utili durante la fase di sviluppo e *test*, e restano ora disponibili per eventuali scopi futuri.

Figura 4 Basetta con tutti i componenti saldati su di essa.
Figure 4 *Board with all the components soldered.*

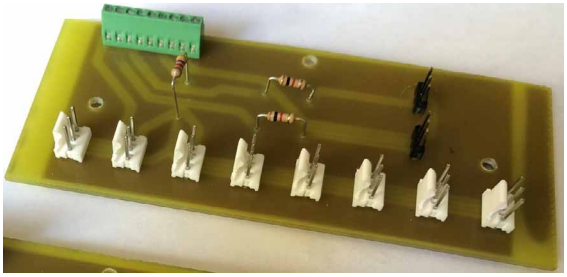


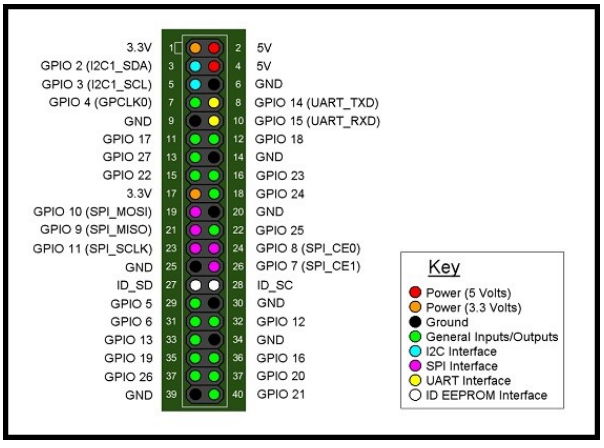
Figura 5 Connettore per DS18B20.
Figure 5 *Connector for DS18B20.*

2.2 Il Raspberry Pi

Una configurazione particolarmente diffusa per controllare gli integrati DS18B20 si ottiene tramite un microprocessore Arduino o similare, in questo caso si è preferito affidarsi al Raspberry Pi [Raspberry Pi Foundation, 2020] (Figura 6). Anche su di esso è presente un bus (GPIO, Figura 7) con i pin necessari ad alimentare i DS18B20, inoltre, essendo gestito da un sistema operativo *Linux* (distribuzione *Raspbian*, derivata da *Debian*), è possibile sfruttare tutte le funzionalità tipiche di un *personal computer*, quali ad esempio ambienti di sviluppo nel linguaggio preferito, presenza di *stack* di rete, pianificazione delle attività, presenza di porte USB per collegamento di periferiche.

Figura 6 Raspberry Pi model B.
Figure 6 *Raspberry Pi model B.*

Figura 7 Il bus GPIO.
Figure 7 *The GPIO bus.*



Il circuito stampato descritto al paragrafo precedente si collega quindi al connettore GPIO del Raspberry in Figura 7, utilizzando i seguenti sette pin:

- pin 1 [+3.3 V]
- pin 6 [Gnd]
- pin 7 [GPIO4]

} sensori di temperatura DS18B20

- pin 2 [+5 V]
- pin 9 [Gnd]
- pin 11 [GPIO17]
- pin 12 [GPIO18]

} interruttori optoelettronici

Il collegamento tra il bus GPIO del Raspberry e la basetta autocostruita avviene quindi tramite un cavo a 8 fili (per praticità è stato utilizzato un comune cavo UTP cat. 5E solitamente presente nelle connessioni di rete), di cui se ne usano solamente 6, poiché i pin 6 e 9 usano lo stesso filo. L'intero sistema è stato quindi assemblato in dei contenitori ottenuti da lastre di compensato sagomate mediante un apposito macchinario per il taglio laser, ed è illustrato in Figura 8 con tutti i cablaggi completati. In particolare, nella figura di sinistra si vede il modulo "accessori" composto dalla basetta di collegamento dei sensori di temperature e due interruttori controllati via *software* da remoto; nella parte destra c'è il modulo "centrale" contenente il Raspberry. I connettori a forma di cubo nero sono le porte RJ45 che servono, come anticipato prima, a collegare fino a due moduli "accessori" a quello "centrale".

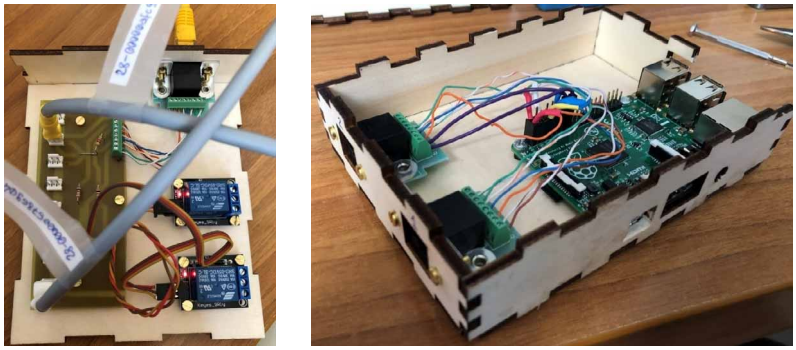


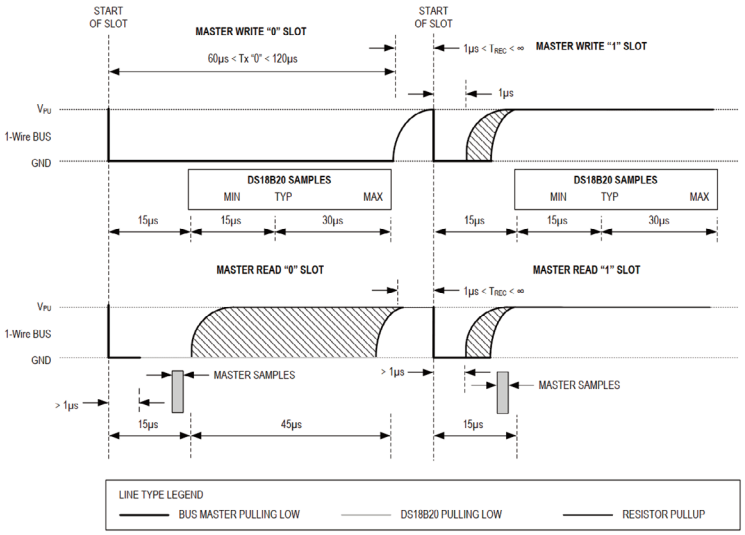
Figura 8 Moduli "centrale" (a destra) e "accessori" con 2 sonde collegate via cavo (a sinistra).
***Figure 8** "Main" module (right) and "additional" module (left) with 2 wired probes.*

2.3 Criticità

Durante tutta la fase di sviluppo e *test*, l'intero sistema è stato tenuto in funzione con pieno successo, alternando gli integrati DS18B20 ad esso collegati, anche due o tre contemporaneamente, aggiungendoli e rimuovendoli "a caldo" per verificare l'affidabilità del sistema. Successivamente, l'installazione è stata effettuata nella nuova sala CED, collegando i primi quattro sensori di temperatura presso i due *rack* interamente popolati (per ogni armadio una sonda per misurare l'aria in ingresso ed una posteriormente per la temperatura dell'aria in uscita), più altri due sensori nella sala UPS, uno per misurare la temperatura in uscita dal condizionatore ed uno all'interno dell'armadio batterie. In tale configurazione, nessuna sonda era più funzionante, il Raspberry non riusciva a connettersi stabilmente ai DS18B20, i quali apparivano e sparivano ad intermittenza dalla *directory* di sistema operativo in cui poter leggere le misure (si veda il paragrafo 3.2 più avanti). Riducendo invece il numero di integrati collegati a 2, tutto tornava a funzionare.

Se invece si collegavano i 2 integrati della sala UPS, con un cavo di oltre 8m ognuno, il sistema tornava di nuovo instabile, anche con un solo sensore. Alla fine si è capito che il problema si verifica quando la somma delle lunghezze dei cavi a cui sono saldati i DS18B20 supera all'incirca gli 8 m complessivi. Dopo un'analisi più approfondita sul funzionamento del protocollo *1-Wire bus*, è emerso che, per una corretta inizializzazione ed esecuzione della misura da parte del DS18B20, è necessario che vengano rispettate delle temporizzazioni minime (Figura 9) durante il processamento dei segnali all'interno dell'integrato stesso [Maxim Integrated, 2021], dell'ordine tipicamente delle decine di µs. Con l'uso di cavi di diversi metri di lunghezza, iniziano ad essere apprezzabili le componenti capacitiva ed induttiva delle impedenze passive, le quali introducono attenuazioni alle frequenze più alte. Ciò si traduce in una deformazione della forma degli impulsi, in cui i fronti di salita e discesa si allontanano dall'ideale "onda quadra" per assumere l'aspetto più o meno pronunciato di esponenziali crescenti o decrescenti. Quando questi tempi di transizione dallo stato 1 allo 0 e viceversa si allungano eccessivamente, andando ad alterare le temporizzazioni indicate in Figura 9, le operazioni di inizializzazione e lettura/scrittura non vanno più a buon fine. Una conferma visiva di tale interpretazione si è avuta osservando su un oscilloscopio la forma del segnale presente sul pin "Data" comune a tutti i DS18B20, dove i fronti di salita e discesa degli impulsi risultavano effettivamente molto deformati.

Figura 9 Diagramma delle temporizzazioni in lettura e scrittura.
Figure 9 *Read/write timing graph.*



Un primo livello di soluzione al problema, più a breve termine, si è avuto sostituendo i cavi saldati alle sonde con altri caratterizzati da valori di impedenza passiva minori. Allo scopo, sono state misurate con un capacimetro varie tipologie di cavi a disposizione, selezionando quello col valore inferiore. In questo modo è stato possibile aumentare la portata massima dei cavi dai precedenti 8 m fino a circa 15 m. Posizionando poi i moduli direttamente sulla guida passacavi sopra i *rack*, è stato possibile connettere le sonde con cavi al massimo di 2.5 m ognuno. Con questi accorgimenti è stato possibile far funzionare le quattro sonde dedicate al monitoraggio della temperatura dei *rack*, con la possibilità di espanderle a sei per coprire anche il terzo armadio; tuttavia si è persa la scalabilità sul numero di sensori installabili, che era uno dei requisiti di partenza.

2.4 Il modulo ESP-01

Dalla soluzione tampone descritta alla fine del paragrafo precedente, sono comunque rimaste escluse le sonde per il monitoraggio dell'UPS, posto in un locale di fianco alla sala CED.

Dovendo snodarsi fin sopra il controsoffitto per superare il tramezzo ignifugo, i cavi dei sensori superano di poco gli 8 m di lunghezza, eccedendo abbondantemente il limite consentito. La necessità di svincolarsi da un cavo fisico ha quindi suggerito l'utilizzo di una soluzione *wireless*. Il miglior candidato allo scopo si è rivelato il modulo ESP-01 (Figura 10), prodotto dall'azienda AI-Thinker e basato sul *chip* a basso costo ESP8266, costruito a sua volta dall'azienda cinese Espressif Systems [Espressif Systems, 2021]. Il modulo ESP-01 è in grado di connettersi a reti *Wi-Fi*, supporta completamente il protocollo TCP/IP ed ha a bordo un microcontrollore programmabile.

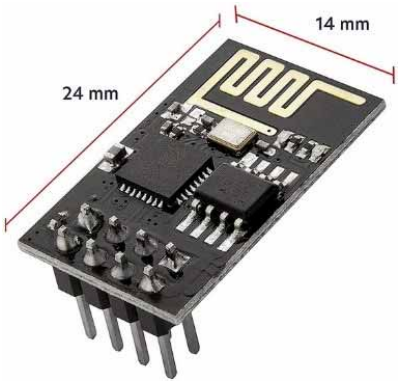


Figura 10 Il modulo di comunicazione wireless ESP-01.
Figure 10 *The ESP-01 wireless communication module.*

Ogni integrato DS18B20, pertanto, viene collegato ad una corrispondente schedina ESP-01, estremamente compatta, come si può verificare in Figura 10, la quale si connette alla rete LAN *wireless* del Centro Dati. Inoltre, il microprocessore è programmato con una applicazione che espone in un *server web* locale le temperature misurate della sonda associata. Ne risulta così un componente estremamente modulare, anch'esso di facile installazione e sostituzione; viene nettamente migliorato il vincolo di distanza della sonda, ora dipendente esclusivamente dalla portata del segnale radio *Wi-Fi*. Inoltre è stato rimosso anche il limite del numero di sonde attive contemporaneamente, in quanto ogni coppia ESP-01 + DS18B20 è completamente indipendente dalle altre ed ha un proprio indirizzo IP sulla LAN *wireless*. In realtà è sempre necessario stendere un cavo per alimentare la ESP-01 (che a sua volta alimenta il DS18B20), ma trattandosi di una tensione in continua a 3.3 V non si evidenziano le criticità espresse nel paragrafo 2.3. L'unico requisito è sull'alimentatore, che deve essere in grado di erogare una corrente sufficiente a far funzionare tutte le ESP-01 che decidiamo di collegare; ma considerato che il consumo medio è di circa 120 mA, un alimentatore da 3 A riesce a gestire fino a 25 moduli con sonda.

3. Componenti *software*

Una volta completata l'infrastruttura *hardware* del nuovo sistema di monitoraggio, si è provveduto a realizzare il componente centrale, vale a dire il sistema *software* di raccolta dati dai vari sensori, la loro archiviazione, la visualizzazione e la gestione degli allarmi.

3.1 Prometheus

Anche in questo caso si è deciso di utilizzare tecnologie più recenti, basando il nucleo centrale sul prodotto *open source* Prometheus, nato proprio per agevolare il controllo remoto nel tempo di sistemi complessi e variegati [Prometheus Authors, 2021]. Esso si basa su un *database* locale in cui vengono memorizzate le serie temporali acquisite, adotta un modello dati multi-

dimensionale ed essi vengono identificati mediante coppie chiave-valore, permette agevolmente elaborazioni ed aggregazioni mediante un proprio linguaggio (PromQL), vi sono sia strumenti interni che integrazione con applicazioni esterne per la visualizzazione, e per ultimo, ma non meno importante, ha un sistema di gestione degli allarmi perfettamente integrato.
L'architettura di Prometheus prevede un *server* centrale, nel nostro caso installato su di una macchina fisica separata, su cui risiedono il *database* e tutta la logica di raccolta, gestione dati ed eventuali analisi, quindi il sottosistema "AlertManager" per la gestione degli allarmi. La raccolta dei dati avviene mediante il metodo *pull*, ovvero dei piccoli programmi, chiamati *exporter*, raccolgono le informazioni sui sistemi di interesse e le mettono a disposizione tramite il protocollo HTTP sotto forma di "metriche", vale a dire serie temporali di valori numerici che cambiano al passare del tempo. Nell'ambito della già ampia comunità di utilizzatori di Prometheus, sono già stati sviluppati *exporter* per una vasta gamma di applicativi, quali *database* (ElasticSearch, PostgreSQL, Oracle DB), sistemi di messaggistica (Kafka, RabbitMQ), *storage* (Ceph, NetApp), dispositivi *hardware* (Dell, Fortigate, Nvidia GPU), *web server* (Apache, Nginx), sistemi operativi stessi e loro componenti (BIND, SSH, SMTP), solo per citarne alcuni. In tutti gli altri casi in cui non esiste ancora un *exporter*, è possibile svilupparlo autonomamente con l'ausilio di librerie disponibili per i principali ambienti di programmazione (Python, C/C++, Bash, ecc.). Tutta la documentazione necessaria è liberamente disponibile sul sito ufficiale [Prometheus Authors, 2021].

3.2 *Exporter* per le temperature

Dovendo raccogliere dati da un sistema costruito secondo necessità personalizzate, è stato necessario realizzare un *custom exporter* e si è scelto di scriverlo in Python3. Il primo passo è abilitare l'interfaccia *1-Wire* sul Raspberry, tramite lo strumento di sistema `raspi-config` [Hawkins, 2021]. Al successivo riavvio, all'interno della *directory* `/sys/devices/w1_bus_master1/` appaiono delle sotto-*directory*, una per ogni integrato DS18B20 connesso al *1-Wire bus* (Figura 11), identificate tramite il codice univoco a 64 bit associato ad ogni sensore. All'interno di ognuna di queste cartelle sono presenti diversi *files*, quello in cui è scritta la misura di temperatura corrente effettuata dalla sonda corrispondente è il *file* `w1_slave` (Figura 12).

```
pi@raspmonitor:~ $ ll /sys/bus/w1/devices/
totale 0
lrwxrwxrwx 1 root root 0 mag 24 15:42 28-00000afc50d1 -> ../../../devices/w1_bus_master1/28-00000afc50d1
lrwxrwxrwx 1 root root 0 mag 24 15:42 28-00000afc6e33 -> ../../../devices/w1_bus_master1/28-00000afc6e33
lrwxrwxrwx 1 root root 0 mag 24 15:42 28-00000afcd35e -> ../../../devices/w1_bus_master1/28-00000afcd35e
lrwxrwxrwx 1 root root 0 mag 24 15:42 28-00000afd0141 -> ../../../devices/w1_bus_master1/28-00000afd0141
lrwxrwxrwx 1 root root 0 mag 21 13:43 w1_bus_master1 -> ../../../devices/w1_bus_master1
```

Figura 11 Lista dei sensori di temperatura collegati.
***Figure 11** List of plugged temperature sensors.*

L'ultimo passo propedeutico alla creazione dell'*exporter* è l'installazione delle librerie *client* in ambiente Python, da effettuarsi tramite il comando: "`pip install prometheus-client`" [AA.VV., 2021]. Seguendo le semplici linee guida indicate sul sito GitHub, è possibile integrare la parte di codice che si occupa di leggere i valori presenti nei vari file `w1_slave` con quella destinata ad esportare le metriche verso Prometheus. Ne risulta un codice abbastanza semplice e compatto, condizione che aiuta a diminuire la probabilità di eventuali errori di programmazione (*bug*). Il listato completo è riportato in Appendice A. La lettura delle misure di temperatura viene effettuata ogni 13 secondi, per essere sicuri di non inserire nel *database* centrale due volte la

stessa identica misura, dal momento che l'*exporter* viene interrogato ogni 15 secondi. La porta utilizzata è solitamente la 8000 o comunque una diversa dalla 80, per non interferire con eventuali altri *web server* che dovessero essere attivi sulla macchina dove gira l'*exporter*.

```
pi@raspmonitor:/sys/devices/w1_bus_master1/28-00000afc50d1 $ ll
totale 0
lrwxrwxrwx 1 root root    0 mag 21 13:43 driver -> ../../../bus/w1/drivers/w1_slave_driver
drwxr-xr-x 3 root root    0 mag 21 13:43 hwmon
-r--r--r-- 1 root root 4096 ott 12 12:01 id
-r--r--r-- 1 root root 4096 ott 12 12:01 name
drwxr-xr-x 2 root root    0 ott 12 12:01 power
lrwxrwxrwx 1 root root    0 mag 21 13:43 subsystem -> ../../../bus/w1
-rw-r--r-- 1 root root 4096 mag 21 13:43 uevent
-rw-r--r-- 1 root root 4096 mag 24 15:42 w1_slave
pi@raspmonitor:/sys/devices/w1_bus_master1/28-00000afc50d1 $ cat w1_slave
71 01 4b 46 7f ff 0f 10 56 : crc=56 YES
71 01 4b 46 7f ff 0f 10 56 t=23062
pi@raspmonitor:/sys/devices/w1_bus_master1/28-00000afc50d1 $
```

Figura 12 Dove trovare la misura di temperatura.
***Figure 12** Where the temperature value is written.*

Per l'acquisizione dei valori misurati tramite i moduli ESP-01 il procedimento è sostanzialmente il medesimo, con l'unica differenza che in questo caso non si deve leggere un *file* su *filesystem*, bensì estrarre l'informazione da una pagina *web*. Poiché i *server* attivi sui moduli *Wi-Fi* sono configurati con indirizzi IP privati statici, è abbastanza semplice modificare la parte di codice interessato. Il listato finale è consultabile in Appendice B.

3.3 *Exporter* SNMP

I parametri legati alla qualità dell'alimentazione di rete elettrica sono misurati dall'UPS Borri B8033FXS, che fornisce la protezione da interruzioni di energia a tutto il Centro Dati. Il modo più pratico per estrarre tali parametri è attraverso il protocollo SNMP (*Simple Network Management Protocol*) [Stallings, 1999], supportato dal Borri. Per prima cosa occorre una lista di tutte le informazioni che l'UPS riesce a veicolare tramite SNMP, la quale può essere ottenuta mediante il comando riportato in Figura 13.

```
[stefano@prometheus ~]$ snmpwalk -v 2c -c public -O e <ip_borri>

SNMPv2-MIB::sysDescr.0 = STRING: CS121 v 5.12.70
SNMPv2-MIB::sysObjectID.0 = OID: SNMPv2-SMI::mib-2.33
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (710351200) 82 days, 5:11:52.00
SNMPv2-MIB::sysContact.0 = STRING: Stefano Chiappini
SNMPv2-MIB::sysName.0 = STRING: CS-121
SNMPv2-MIB::sysLocation.0 = STRING: Locale tecnico
SNMPv2-MIB::sysServices.0 = INTEGER: 72
IF-MIB::ifNumber.0 = INTEGER: 1
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifDescr.1 = STRING: CS121 Ethernet
…
```

Figura 13 Prime righe dell'elenco di tutte le informazioni restituite dall'UPS tramite query SNMP.
***Figure 13** First few lines returned by the SNMP query on the UPS.*

Tale comando restituisce la *Management Information Base* (MIB), ovvero un elenco di tutti gli oggetti dati utilizzati dalla particolare apparecchiatura interrogata, il questo caso l'UPS, strutturato in maniera gerarchica.
Il passo successivo è scorrere questa struttura ad albero e trovare l'associazione tra ogni singola voce, che prende il nome di OID (*Object IDentifier*), e la grandezza fisica corrispondente

(ad esempio: Input_Current_Phase_T_A). Gli OID, infatti, sono stringhe numeriche abbastanza lunghe e complesse, che hanno una forma del tipo: 1.3.66.9.281.2.33.1.3.3.1.4.3 ed identificano univocamente ogni dispositivo ed ogni stato ad esso associato: la prima parte dell'OID (es.: 1.3.66.9.281) identifica il dispositivo mentre la seconda parte (es.: 2.33.1.3.3.1.4.3) individua l'informazione riferita a quel dispositivo. L'assegnazione di un significato intellegibile a molti OID è stato un passaggio abbastanza lungo e noioso, causa la totale mancanza di documentazione al riguardo nella manualistica Borri. Una volta individuate tutte le grandezze di interesse, è stato elaborato un *exporter* dedicato, in analogia con quanto descritto nel paragrafo 3.2, sempre in linguaggio Python3 e con l'ausilio della libreria `prometheus-client`; anche questo listato è disponibile in appendice (Appendice C).

3.4 Visualizzazione con Grafana

Una volta che tutte le metriche di interesse sono perfettamente integrate all'interno di un unico *database*, è possibile effettuare subito dei grafici per visualizzare l'andamento temporale delle grandezze in esame. Sebbene Prometheus metta già a disposizione un suo strumento integrato, la maggiore flessibilità nella creazione di grafici interattivi e reportistica si ha con l'utilizzo di "Grafana" [Grafana Labs, 2021], una applicazione *software* sviluppata nell'ambito di un progetto *open source*, finalizzata alla visualizzazione di dati provenienti da fonti più disparate. L'interoperabilità fra questi due strumenti è molto stretta e pertanto creare una connessione tra Grafana ed un *database* Prometheus è un'operazione semplice ed immediata, ed una volta stabilita è altrettanto facile creare innumerevoli pannelli di controllo (denominati *dashboard*) contenenti tutti i tipi di grafici che si desidera inserire (a linee, a barre, a torta, ecc.), con ogni genere di personalizzazione riguardo stili, colori, legende, tipi di carattere e altro. Avvalendosi quindi di queste funzionalità, è stata creata una *dashboard* in Grafana contenente i grafici di temperature, tensioni, correnti e potenze elettriche, i quali sono poi stati importati all'interno di un sito *web* (Figura 14) realizzato con *Apache* [The Apache Software Foundation, 2021] per una migliore e più sicura consultazione; così facendo, si evita infatti l'accesso dell'utente alla *dashboard*, dove è possibile effettuare modifiche e si garantisce una visualizzazione dei grafici aggiornati ogni 15 secondi sulla base dei dati ricevuti in tempo reale.

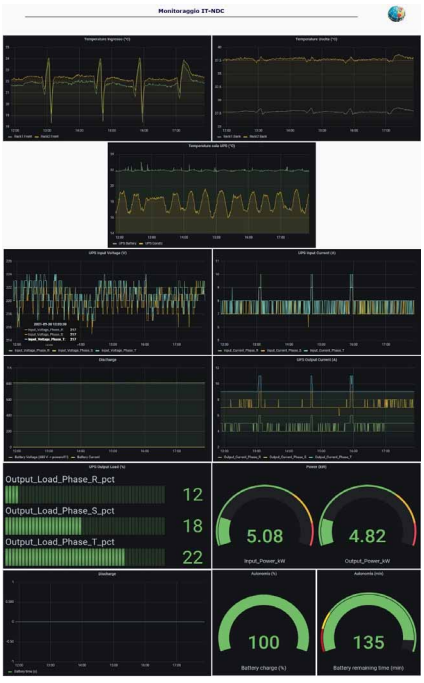


Figura 14 Schermata di visualizzazione in tempo reale dei parametri controllati.

Figure 14 *Dashboard with real-time display of the monitored parameters.*

3.5 *AlertManager* e gestione degli allarmi

Un grande valore aggiunto del sistema Prometheus è la presenza di una gestione degli allarmi completamente configurabile, suddivisa sostanzialmente in due componenti: la prima opera all'interno di Prometheus basandosi su una serie di regole definite dall'utente, il quale invia gli allarmi alla seconda componente, chiamata AlertManager, che si occupa della gestione degli stessi, quali l'invio ad uno o più destinatari secondo i canali prescelti (*email*, *sms*, *chat*, ecc.), applica eventuali politiche di aggregazione o inibizione. L'intero sistema si configura agendo sostanzialmente su questi tre *file* di testo:

1. prometheus.yml
2. prometheus_rules.yml
3. alertmanager.yml

Nel primo sono definiti tutti gli *exporter* interrogati dal *server*, identificati da indirizzo IP e porta di ascolto; nel secondo tutte le condizioni di allarme che devono essere gestite, nonché eventuali metriche composite ottenute mediante elaborazioni (operazioni, aggregazioni, statistiche) effettuate sulle metriche semplici; nel terzo sono definiti i gruppi di destinatari unitamente ai criteri di smistamento degli avvisi ed ai relativi *template*.
Nel caso specifico del sistema di monitoraggio del CED, sono stati impostati allarmi quando le sonde di temperatura misurano valori che eccedono soglie prefissate, quali ad esempio 23 °C per l'aria in ingresso ai *server* e 25 °C per la temperatura delle batterie dell'UPS; per quanto riguarda invece i parametri di alimentazione elettrica, sono generati avvisi (*warning*) quando la tensione in ingresso all'UPS assume valori inferiori a 214 V di picco (sottotensioni), dei veri e propri allarmi invece quando questa va a zero Volt su tutte e tre le fasi, che significa che si è verificato un *blackout*. Altre notifiche sono inviate al sottoscritto ed agli altri collaboratori del Centro quando l'UPS è in fase di scarica e la tensione delle batterie si avvicina al livello minimo, che corrisponde ad una autonomia residua prossima all'esaurimento. In Figura 15 si può vedere un particolare dell'interfaccia di gestione di Prometheus.

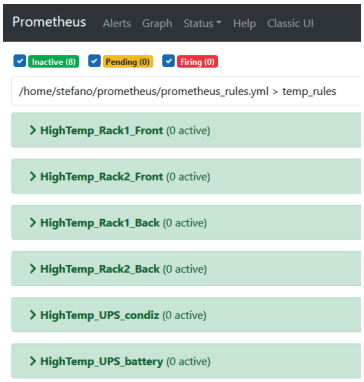


Figura 15 Interfaccia di controllo della configurazione e funzionamento di Prometheus.
Figure 15 *Prometheus management graphical interface.*

4. Conclusioni

Giunti al termine di questo lavoro, si può tranquillamente affermare che gli obiettivi prefissati sono stati pienamente raggiunti.
Il sistema è completamente modulare, sia per quanto riguarda la componente elettronica della sensoristica, sia per la corrispondente componente *software* che si occupa dell'acquisizione, memorizzazione e visualizzazione dati. Ad esempio, uno sviluppo in programma per il futuro è l'aggiunta di misuratori di umidità e rilevatori di fumo, da affiancare a quelli appartenenti al

sistema antincendio dell'edificio. Una volta messi in funzione i nuovi sensori è sufficiente sviluppare un *exporter* dedicato per integrare le nuove misure nel sistema.
L'acquisto del materiale utilizzato ha richiesto un *budget* di poche centinaia di Euro, il costo di un Raspberry oscilla fra i 60 e gli 80 € a seconda del modello scelto. Sul fronte software va anche meglio, su tutte le macchine utilizzate sono state installate distribuzioni di Linux (Raspbian e CentOS) a costo zero, ed anche gli applicativi Prometheus e Grafana sono stati utilizzati in versione *community*, ovvero senza costi di licenza.
Da ultimo, ma non meno importante, contrariamente a quanto accade nelle soluzioni commerciali, il sistema descritto è completamente aperto, una volta letta la documentazione chiunque è in grado di effettuare manutenzioni e apportare modifiche o anche aggiunte. Questa è probabilmente la caratteristica più importante in ambito scientifico, dove la collaborazione è più importante del *copyright*.

Ringraziamenti

L'autore desidera ringraziare il Dr. Roberto Carluccio per il prezioso contributo apportato durante l'analisi dei malfunzionamenti discussi nel paragrafo 2.3, insieme alla lunga esperienza nella realizzazione di circuiti stampati.

Bibliografia

AA.VV. (2021). Prometheus instrumentation library for Python applications. GitHub (ultima consultazione 08 04 2021). https://github.com/prometheus/client_python

Autodesk (2019). PCB design software for everyone. Autodesk. https://www.autodesk.com/products/eagle/free-download

Espressif Systems (2021). ESP8266 Wi-Fi MCU. Espressif Systems. https://www.espressif.com/en/products/socs/esp8266

Grafana Labs. (2021). Grafana: The openobservability platform. Grafana Labs. https://grafana.com/

Hawkins M. (2021). Enable 1-Wire Interface on the Raspberry Pi. Raspberry Pi Spy. https://www.raspberrypi-spy.co.uk/2018/02/enable-1-wire-interface-raspberry-pi/

Maxim Integrated (2021). DS18B20. Maxim Integrated. https://www.maximintegrated.com/en/products/sensors/DS18B20.html

Prometheus Authors (2021). Monitoring system & time series database. Prometheus. https://prometheus.io/

Raspberry Pi Foundation (2020). Raspberry Pi. Raspberry Pi Foundation, UK. https://www.raspberrypi.org/

Stallings W. (1999). SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. Addison-Wesley Professional, Inc., pp. 619, ISBN 978-0201485349.

The Apache Software Foundation (2021). Apache HTTP Server Project. Apache HTTP Server. The Apache Software Foundation. https://httpd.apache.org/

APPENDICI

Appendice A – *Exporter* per temperature da Raspberry

```
#———————————————————+
#  Exporter per far leggere a Prometheus |
#    i dati di utilizzo dell'UPS Borri   |
#                                        |
#  (c)  Stefano Chiappini                |
#                                        |
#     ultima modifica: 28-Apr-2021 18:30 |
#———————————————————+

#
# Prerequisiti OS:
#
#   sudo pip3 install prometheus_client
#   — add firewalld rule —
#

import os
import glob
import subprocess
import prometheus_client
import time

#
#  Constants
#
UPDATE_PERIOD = 13   # in seconds
HTTP_PORT = 8000


os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

device_folder_array = glob.glob('/sys/bus/w1/devices/28*')
num_sonde = len(device_folder_array)
device_file = []
head = []
etichetta = {
   '28-00000afd0141' : 'temp_rack2_back_C',
   '28-00000afc6e33' : 'temp_rack1_back_C',
   '28-00000afcd35e' : 'temp_rack2_front_C',
   '28-00000afc50d1' : 'temp_rack1_front_C'
}

for device_folder in device_folder_array:
   device_file.append(device_folder + '/w1_slave')
   head.append(device_folder.split('/')[-1])

def read_temp_raw(j):
```

```
    f = open(device_file[j], 'r')
    lines = f.readlines()
    f.close()
    return lines

def read_temps():
    temps = []
    for n in range(num_sonde):
        lines = read_temp_raw(n)
        while lines[0].strip()[-3:] != 'YES':
            time.sleep(0.2)
            lines = read_temp_raw()
        equals_pos = lines[1].find('t=')
        if equals_pos != -1 :
            temp_string = lines[1 ][equals_pos+2:]
            temp_c = float(temp_string) / 1000.0
        temps.append(temp_c)
     return temps, time.strftime('%Y-%m-%d %H:%M:%S')   # stampa la data in formato
completo

#
#  define Prometheus metrics
#

#SYSTEM_USAGE = prometheus_client.Gauge('system_usage',
#                                       'Hold current system resource usage',
#                                       ['resource_type'])

TEMPS_CED = prometheus_client.Gauge('temps_CED', 'Hold current rack temperatires',
                                    ['resource_type'])


if __name__ == '__main__':
   prometheus_client.start_http_server(HTTP_PORT)   # avvia server web sulla porta
desiderata


# ********************
#  routine principale
# ********************
while True:
    #
    #   Legge le temperature
    #
    my_temps, tempo = read_temps()
    #
    #   Inserisce i valori SNMP nelle metriche
    #
    n = 0
    for valore in my_temps:
```

```
      TEMPS_CED.labels(etichetta[head[n]]).set(valore)
      n = n + 1
   #print('...iterazione completata...')    # solo per sviluppo/debug
   time.sleep(UPDATE_PERIOD)
```

Appendice B – *Exporter* per temperature via *Wi-Fi*

```
#————————————————————+
#  Exporter per far leggere a Prometheus |
#    i dati delle sonde di temperatura   |
#    Wi-Fi.                              |
#                                        |
#  (c)  Stefano Chiappini                |
#                                        |
#     ultima modifica: 20-Lug-2021 16:00 |
#————————————————————+

#
# Prerequisiti OS:
#
#   sudo pip3 install urllib prometheus_client
#   — add firewalld rule —
#

import urllib.request
import prometheus_client
import time

#
#  Constants
#
UPDATE_PERIOD = 13   # in seconds
HTTP_PORT = 8001


#
#  Lista sonde da interrogare
#
sonde = {'192.168.4.151' : 'UPS_condiz_Temp',
         '192.168.4.152' : 'UPS_batterie_Temp' }
#print("IP = ", sonde)


#
#  define Prometheus metrics
#
UPS_TEMPS   =   prometheus_client.Gauge('temps_UPS',   'Hold   current   Borri   UPS
temperatures',
                                        ['resource_type'])
```

```
if __name__ == '__main__':
  prometheus_client.start_http_server(HTTP_PORT)   # avvia server web sulla porta
desiderata



# ********************
#  routine principale
# ********************
while True:
    for IP,Desc in sonde.items():
         web_content = urllib.request.urlopen("http://"+IP+"/").read().decode("utf-
8")

        posiz = web_content.find('temperaturec')
        #print("posiz = ", posiz)

        if (posiz>-1):
            temp = web_content[posiz+14:posiz+19]

        print(Desc, " = ", temp)
        UPS_TEMPS.labels(Desc).set(temp)
    time.sleep(UPDATE_PERIOD)
```

Appendice C – *Exporter* SNMP

```
#————————————————————+
#  Exporter per far leggere a Prometheus |
#    i dati di utilizzo dell'UPS Borri   |
#                                        |
#  (c)  Stefano Chiappini                |
#                                        |
#     ultima modifica: 16-Lug-2021 11:00 |
#————————————————————+

#
# Prerequisiti OS:
#
#   sudo yum -y install net-snmp net-snmp-utils
#   sudo pip3 install prometheus_client
#   — add firewalld rule port HTTP_PORT —
#

import subprocess
import prometheus_client
import time

#
```

```
#  Constants
#
UPDATE_PERIOD = 13   # in seconds
BORRI_IP = '10.230.15.249'
HTTP_PORT = 8000

#
#  define MIB entries to query
#
mib = {'mib-2.33.1.3.3.1.3.1' : 'Input_Voltage_Phase_R_V',
       'mib-2.33.1.3.3.1.3.2' : 'Input_Voltage_Phase_S_V',
       'mib-2.33.1.3.3.1.3.3' : 'Input_Voltage_Phase_T_V',
       'mib-2.33.1.3.3.1.4.1' : 'Input_Current_Phase_R_A',
       'mib-2.33.1.3.3.1.4.2' : 'Input_Current_Phase_S_A',
       'mib-2.33.1.3.3.1.4.3' : 'Input_Current_Phase_T_A',
       'mib-2.33.1.3.3.1.5.1' : 'Input_Power_Phase_R_kVA',
       'mib-2.33.1.3.3.1.5.2' : 'Input_Power_Phase_S_kVA',
       'mib-2.33.1.3.3.1.5.3' : 'Input_Power_Phase_T_kVA',
       'mib-2.33.1.4.2.0'     : 'Output_Frequency_*_10_Hz',
       'mib-2.33.1.4.4.1.2.1' : 'Output_Voltage_Phase_R_V',
       'mib-2.33.1.4.4.1.2.2' : 'Output_Voltage_Phase_S_V',
       'mib-2.33.1.4.4.1.2.3' : 'Output_Voltage_Phase_T_V',
       'mib-2.33.1.4.4.1.3.1' : 'Output_Current_Phase_R_A',
       'mib-2.33.1.4.4.1.3.2' : 'Output_Current_Phase_S_A',
       'mib-2.33.1.4.4.1.3.3' : 'Output_Current_Phase_T_A',
       'mib-2.33.1.4.4.1.4.1' : 'Output_Power_Phase_R_VA',
       'mib-2.33.1.4.4.1.4.2' : 'Output_Power_Phase_S_VA',
       'mib-2.33.1.4.4.1.4.3' : 'Output_Power_Phase_T_VA',
       'mib-2.33.1.4.4.1.5.1' : 'Output_Load_Phase_R_pct',
       'mib-2.33.1.4.4.1.5.2' : 'Output_Load_Phase_S_pct',
       'mib-2.33.1.4.4.1.5.3' : 'Output_Load_Phase_T_pct',
       'mib-2.33.1.5.1.0'     : 'Bypass_Frequency_HZ',
       'mib-2.33.1.5.3.1.2.1' : 'Bypass_Voltage_Phase_R_V',
       'mib-2.33.1.5.3.1.2.2' : 'Bypass_Voltage_Phase_S_V',
       'mib-2.33.1.5.3.1.2.3' : 'Bypass_Voltage_Phase_T_V',
       'mib-2.33.1.2.1.0'     : 'Battery_status',
       'mib-2.33.1.2.2.0'     : 'Ups_seconds_on_battery',
       'mib-2.33.1.2.3.0'     : 'Battery_remaining_time_min',
       'mib-2.33.1.2.4.0'     : 'Battery_charge_pct',
       'mib-2.33.1.2.5.0'     : 'Battery_Voltage_*_10_V',
       'mib-2.33.1.2.6.0'     : 'Battery_Current_A',
       'mib-2.33.1.6.1.0'     : 'Ups_alarm_present'
       }
results = {}


#
#  define Prometheus metrics
#
```

```
#SYSTEM_USAGE = prometheus_client.Gauge('system_usage',
#                                       'Hold current system resource usage',
#                                       ['resource_type'])

BORRI_USAGE = prometheus_client.Gauge('UPS_usage', 'Hold current Borri UPS resource
usage',
                                      ['resource_type'])


if __name__ == '__main__':
   prometheus_client.start_http_server(HTTP_PORT)   # avvia server web sulla porta
desiderata


# ********************
#  routine principale
# ********************
while True:
  #
  #   Crea le coppie metrica-valore
  #
  for key, desc in mib.items():
      process = subprocess.Popen(['snmpget','-v2c','-c','-c',BORRI_IP,key],
                                 stdout=subprocess.PIPE,
                                 stderr=subprocess.PIPE,
                                 universal_newlines=True)
      stdout, stderr = process.communicate()
      valore = stdout.split()
      try:
          results[desc] = valore[-1]
      except IndexError as err:
          print('Invalid operation ({})!'.format(err))
          continue    # skip loop iteration
#          print("stdout = ",stdout)     # solo per sviluppo/debug
#          print("valore = ",valore)     # solo per sviluppo/debug
#          results[desc] = 999           # solo per sviluppo/debug

  IVR = float(results['Input_Voltage_Phase_R_V'])
  IVS = float(results['Input_Voltage_Phase_S_V'])
  IVT = float(results['Input_Voltage_Phase_T_V'])
  ICR = float(results['Input_Current_Phase_R_A'])
  ICS = float(results['Input_Current_Phase_S_A'])
  ICT = float(results['Input_Current_Phase_T_A'])
  OVR = float(results['Output_Voltage_Phase_R_V'])
  OVS = float(results['Output_Voltage_Phase_S_V'])
  OVT = float(results['Output_Voltage_Phase_T_V'])
  OCR = float(results['Output_Current_Phase_R_A'])
  OCS = float(results['Output_Current_Phase_S_A'])
  OCT = float(results['Output_Current_Phase_T_A'])
  IPW = (IVR*ICR+IVS*ICS+IVT*ICT) / 1000
```

```
OPW = (OVR*OCR+OVS*OCS+OVT*OCT) / 1000
#print("IVR = ", IVR)           # solo per sviluppo/debug
#print("IVS = ", IVS)           # solo per sviluppo/debug
#print("IVT = ", IVT)           # solo per sviluppo/debug
#print("OVR = ", OVR)           # solo per sviluppo/debug
#print("OVS = ", OVS)           # solo per sviluppo/debug
#print("OVT = ", OVT)           # solo per sviluppo/debug
#print("Input power  = ", IPW)  # solo per sviluppo/debug
#print("Output power = ", OPW)  # solo per sviluppo/debug


#
#   Inserisce i valori SNMP nelle metriche
#
for desc, valore in results.items():
    BORRI_USAGE.labels(desc).set(valore)
BORRI_USAGE.labels('Input_Power_kW').set(IPW)
BORRI_USAGE.labels('Output_Power_kW').set(OPW)
#print('...iterazione completata...')    # solo per sviluppo/debug
time.sleep(UPDATE_PERIOD)
```

QUADERNI di GEOFISICA

ISSN 1590-2595 http://istituto.ingv.it/it/le-collane-editoriali-ingv/quaderni-di-geofisica.html/

I QUADERNI DI GEOFISICA (QUAD. GEOFIS.) accolgono lavori, sia in italiano che in inglese, che diano particolare risalto alla pubblicazione di dati, misure, osservazioni e loro elaborazioni anche preliminari che necessitano di rapida diffusione nella comunità scientifica nazionale ed internazionale. Per questo scopo la pubblicazione on-line è particolarmente utile e fornisce accesso immediato a tutti i possibili utenti. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi. I QUADERNI DI GEOFISICA sono presenti in "Emerging Sources Citation Index" di Clarivate Analytics, e in "Open Access Journals" di Scopus.

QUADERNI DI GEOFISICA (QUAD. GEOFIS.) welcome contributions, in Italian and/or in English, with special emphasis on preliminary elaborations of data, measures, and observations that need rapid and widespread diffusion in the scientific community. The on-line publication is particularly useful for this purpose, and a multidisciplinary Editorial Board with an accurate peer-review process provides the quality standard for the publication of the manuscripts. QUADERNI DI GEOFISICA are present in "Emerging Sources Citation Index" of Clarivate Analytics, and in "Open Access Journals" of Scopus.

RAPPORTI TECNICI INGV

ISSN 2039-7941 http://istituto.ingv.it/it/le-collane-editoriali-ingv/rapporti-tecnici-ingv.html/

I RAPPORTI TECNICI INGV (RAPP. TEC. INGV) pubblicano contributi, sia in italiano che in inglese, di tipo tecnologico come manuali, software, applicazioni ed innovazioni di strumentazioni, tecniche di raccolta dati di rilevante interesse tecnico-scientifico. I RAPPORTI TECNICI INGV sono pubblicati esclusivamente on-line per garantire agli autori rapidità di diffusione e agli utenti accesso immediato ai dati pubblicati. Un Editorial Board multidisciplinare ed un accurato processo di peer-review garantiscono i requisiti di qualità per la pubblicazione dei contributi.

RAPPORTI TECNICI INGV (RAPP. TEC. INGV) publish technological contributions (in Italian and/or in English) such as manuals, software, applications and implementations of instruments, and techniques of data collection. RAPPORTI TECNICI INGV are published online to guarantee celerity of diffusion and a prompt access to published data. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

MISCELLANEA INGV

ISSN 2039-6651 http://istituto.ingv.it/it/le-collane-editoriali-ingv/miscellanea-ingv.html

MISCELLANEA INGV (MISC. INGV) favorisce la pubblicazione di contributi scientifici riguardanti le attività svolte dall'INGV. In particolare, MISCELLANEA INGV raccoglie reports di progetti scientifici, proceedings di convegni, manuali, monografie di rilevante interesse, raccolte di articoli, ecc. La pubblicazione è esclusivamente on-line, completamente gratuita e garantisce tempi rapidi e grande diffusione sul web. L'Editorial Board INGV, grazie al suo carattere multidisciplinare, assicura i requisiti di qualità per la pubblicazione dei contributi sottomessi.

MISCELLANEA INGV (MISC. INGV) favours the publication of scientific contributions regarding the main activities carried out at INGV. In particular, MISCELLANEA INGV gathers reports of scientific projects, proceedings of meetings, manuals, relevant monographs, collections of articles etc. The journal is published online to guarantee celerity of diffusion on the internet. A multidisciplinary Editorial Board and an accurate peer-review process provide the quality standard for the publication of the contributions.

Coordinamento editoriale
Francesca DI STEFANO
Istituto Nazionale di Geofisica e Vulcanologia

Progetto grafico
Barbara ANGIONI
Istituto Nazionale di Geofisica e Vulcanologia

Impaginazione
Barbara ANGIONI
Patrizia PANTANI
Massimiliano CASCONE
Istituto Nazionale di Geofisica e Vulcanologia

Istituto Nazionale di Geofisica e Vulcanologia
Via di Vigna Murata, 605
00143 Roma
tel. +39 06518601

www.ingv.it

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA