

Relazione Progetto Programmazione di Reti

Traccia 1 - Progetto IoT

Riccardo Omiccioli
Matricola: 0000939201



Introduzione

Il progetto richiede di modellare uno scenario tipico del mondo IoT dove si ha un SoC (ad esempio un ESP-32) con capacità di collegamento tramite Wi-Fi che effettua misure e un sistema generalmente più performante che si occupa di raccogliere i dati dei dispositivi e inviarli ad un server oltre ad offrire altre possibili funzionalità opzionali (utilizzando ad esempio una Raspberry Pi).

Indirizzamento

Come scritto nelle specifiche i device e il gateway appartengono alla stessa rete con indirizzamento 192.168.1.0/24 e sarà poi il gateway ad inviare i dati al server attraverso una rete di classe 10.10.10.0/24. Anche se non influente per il progetto si capisce quindi che entrambe sono reti private dato che gli indirizzi citati sono registrati dalla IANA come indirizzi per reti private. Inoltre per emulare lo scenario completo su un solo dispositivo è necessario sfruttare l'interfaccia di loopback (indirizzo IP127.0.0.1). Gli indirizzi dei device, gateway e server saranno quindi simulati attraverso l'invio di ipotetici indirizzi IP sotto forma di stringhe mentre le socket utilizzate per le comunicazioni dovranno necessariamente fare riferimento all'indirizzo di loopback. La scelta fatta è stata di propagare solo gli indirizzi IP dei device in quanto richiesta la loro stampa nel server mentre gli indirizzi del gateway e del server non sono stati condivisi attraverso stringhe in quanto, almeno in questo caso, si hanno un solo gateway e server e si utilizza comunque l'interfaccia di loopback. Diversamente, nel caso di più gateway o server sarebbe stato utile visualizzare gli indirizzi IP di essi per rendere comprensibili gli scambi di dati che avvengono sulla rete.

Organizzazione del programma

È stato deciso di utilizzare una programmazione orientata al multi-threading per modellare i device, avere un thread per il controllo e interazione con l'utente e per le parti di codice responsabili della comunicazione sulle interfacce di rete. A questo scopo è stata realizzata una classe python chiamata ThreadedProgram che permette di gestire in modo semplice i thread favorendo il riuso di codice nei programmi dei device, gateway e server rendendo inoltre tali programmi molto più semplici, dato che sarà sufficiente implementare solamente la funzione target del thread che definirà il comportamento del dispositivo. ThreadedProgram si occupa anche di impostare i parametri della libreria logging per poter fare stampe sul terminale in maniera thread-safe. Vengono anche fornite delle variabili per sospendere o stoppare i thread (threads_active e stop_threads), una variabile di lock per eseguire parti di codice in mutua esclusione (lock) e una lista contenente i thread del programma (threads). Per utilizzare tale classe è necessario importarla attraverso l'istruzione `from ThreadedProgram import ThreadedProgram`

e poi creare un oggetto di tale classe con ad esempio `tp = ThreadedProgram()`. Creata un'istanza della classe `ThreadedProgram` è poi possibile aggiungere thread gestiti da tale classe con la funzione `creare_thread(name, target)` dove, come parametro `target`, andrà inserito il nome della funzione `target` da fare eseguire dal thread e che può essere definita direttamente nel file del programma relativo al dispositivo da modellare. Il codice dei device, gateway e server sarà quindi organizzato secondo la seguente struttura:

```
# imports

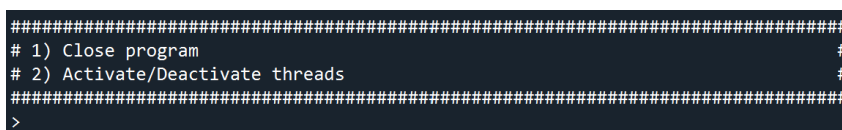
from ThreadedProgram import ThreadedProgram

# program settings variables

def target():
    # Thread setup code
    while True:
        if tp.stop_threads:
            break
        if tp.threads_active:
            # Code to execute while thread is running

tp = ThreadedProgram()
tp.create_thread("thread_name", target)
tp.start_UI()
```

`start_UI()` è una funzione della classe `ThreadedProgram` che sfruttando il `main_thread` di un programma python fornisce all'utente una semplice interfaccia grafica da terminale con la quale sospendere l'esecuzione dei thread o terminare il programma.



```
#####
# 1) Close program                                     #
# 2) Activate/Deactivate threads                       #
#####
>
```

Per terminare il programma, oltre a fare la `join` di tutti i thread è stato necessario eseguire un `sigkill` del programma per riavviare il kernel python ed eliminare la cache delle esecuzioni precedenti che manteneva traccia dei thread eseguiti precedentemente i quali venivano nuovamente messi in esecuzione insieme a quelli generati da nuove run del programma.

Programma device

Le misure dei device sono generate a partire da un valore medio `average_temperature` e `average_humidity` scelto opportunamente con valori plausibili di temperatura media e umidità media ai quali viene sommato un valore che varia tra `-delta` e `+delta` (dove `delta` delle temperature e delle misure di umidità è un parametro del programma) per produrre numeri randomici che si distribuiscono intorno ai valori medi. Se il programma è in fase di test le misure verranno generate casualmente ognuna separata dalle altre con un tempo variabile tra 1 e 5 secondi e vengono inviate al gateway ogni 10 secondi. La casualità nella generazione delle misure è stata inserita appositamente per dimostrare che il sistema non sfrutta strutture dati predefinite con dimensione nota a priori e può adattarsi senza problemi ad un numero variabile di dati inviati a patto di non inviare quantità di dati eccessivi che sovraccarichi la ricezione. Una volta trascorso un tempo maggiore di `day_duration` secondi si inviano al gateway, tramite socket UDP, tutte le misure effettuate nell'ultimo periodo. La variabile `day_duration` è pari a 10 secondi nella modalità di debug o 86400 secondi durante l'esecuzione reale, ovvero il numero di secondi presenti in 24h. Prima e dopo l'invio dei dati sul socket vengono prese misure di tempo per poi visualizzare il tempo necessario alla trasmissione come richiesto (trattandosi di trasferimenti in locale spesso l'invio di dati richiede un tempo trascurabile e non misurabile mentre altre volte vengono visualizzati dei valori nell'ordine di qualche millisecondo probabilmente dovuti a ritardi causati dallo scheduling del sistema operativo).

Programma gateway

Il gateway, alla ricezione dei dati, visualizzerà sul terminale un messaggio che riporta il numero di bytes ricevuti e l'indirizzo IP (finto) dei dispositivi che hanno inviato quei dati. Per memorizzare i dati ricevuti è stata utilizzata una variabile di tipo dictionary nella quale la chiave è la stringa con l'indirizzo IP del dispositivo e il valore la stringa contenente tutte le misure ricevute nell'ultima comunicazione con il device. Questa scelta è stata fatta per rendere la logica del gateway più semplice possibile infatti esso dovrà semplicemente andare a controllare la dimensione del dizionario e inviare al server i dati quando la dimensione del dizionario è uguale a `devices_number` ovvero quando sono state ricevute le misure di tutti i dispositivi. Si fa notare ancora una volta come gli indirizzi IP visualizzati siano programmati artificialmente dato che le comunicazioni effettive avvengono tutte sfruttando l'indirizzo di loopback. Nel caso il sistema debba essere utilizzato realmente basterà sfruttare la funzione `bind()` disponibile sui socket per impostare nel programma l'indirizzo dell'interfaccia di rete della macchina sul quale è in esecuzione. Ovviamente anche nelle funzioni di invio dei pacchetti andrà inserito l'indirizzo IP e la porta corrispondenti a quelli della macchina reale. Anche

nel gateway vengono prese misure di tempo prima e dopo l'invio dei dati sul socket TCP e ancora una volta, così come accadeva per il device, le misure sono solitamente trascurabili o nell'ordine di millisecondi per le stesse ragioni già citate.

Programma server

Il server, una volta che viene avviato, si mette in ascolto di connessioni sul socket TCP e, una volta ricevuta una richiesta, riceve i dati attraverso il socket dedicato alla connessione creato durante l'accettazione della richiesta. Successivamente viene usata più volte la funzione python `ast.literal_eval(data)` che converte le stringhe ricevute nelle opportune strutture dati di python, in questo modo, è poi estremamente semplice, eseguendo dei cicli, ricavare tutte le misure di un dispositivo e da esse ogni singola misura. Vengono poi stampate le misure nel formato richiesto dalle specifiche dopo aver fatto una semplice conversione da tempo epoch ad un formato date-time più comprensibile e presentabile. Alla fine, sempre attraverso il socket della connessione, il server risponde con un messaggio "OK" al gateway in ascolto.

Buffer

Nelle connessioni UDP tra device e gateway le informazioni scambiate tra i device e il gateway utilizzano messaggi strutturati nel seguente modo:

```
255.255.255.255 ['0123456789-00-00', '0123456789-00-00']
```

Dove 255.255.255.255 rappresenta l'IP di un dispositivo e '0123456789-00-00' una delle misure effettuate. Quindi il numero di caratteri inviati è nel caso peggiore uguale a 15 per l'ip 3 per lo spazio e le parentesi quadre e infine $20 \cdot (n-1) + 18$ per le varie misurazioni considerando anche i caratteri utilizzati per rappresentare sotto forma di stringa la struttura lista utilizzata per memorizzare le misure. Considerando il caso peggiore nel quale ogni carattere con codifica utf-8 occupi 4 Bytes e presupponendo di effettuare una misurazione ogni 30 minuti, in un giorno si invieranno 976 caratteri utf-8 che corrispondono a 3904 Bytes e quindi, scegliendo un buffer di 4096 Bytes (4 KiB) si ha la certezza di poter ricevere il pacchetto UDP contenente tutte le misure di un dispositivo. La connessione TCP tra il gateway e il server utilizza messaggi strutturati nel seguente modo:

```
{'255.255.255.255': "'0123456789-00-00', '0123456789-00-00'",  
'255.255.255.255': "'0123456789-00-00', '0123456789-00-00'"}
```

Dove ancora una volta 255.255.255.255 rappresenta l'IP di un dispositivo e '0123456789-00-00' una delle misure effettuate. Si utilizzeranno quindi 92 caratteri utf-8 per gli indirizzi ip e $(20 \cdot (n-1) + 18) \cdot 4$ per tutte le misure effettuate dai 4 dispositivi in un giorno considerando in questi numeri

anche i caratteri necessari a rappresentare sotto forma di stringa la struttura dictionary utilizzata per memorizzare tutte le misure effettuate dai 4 dispositivi. Considerando ancora una volta il caso peggiore di 4 Bytes a carattere si ottiene che inviare tutti i 3924 caratteri utf-8 complessivi richiede 15696 Bytes e quindi un buffer di 16384 Bytes (16 KiB) assicura la completa ricezione del pacchetto.

Immagini applicazioni

Di seguito verranno mostrati alcuni screenshot fatti durante l'esecuzione dei device, gateway e server:

```
(Thread-Device-1) ThreadID_2896 Started
(Thread-Device-2) ThreadID_16588 Started
(Thread-Device-3) ThreadID_12104 Started
(Thread-Device-4) ThreadID_292 Started
(Thread-Device-1) Measure added: 1629646020-13-43
(Thread-Device-2) Measure added: 1629646021-20-45
(Thread-Device-3) Measure added: 1629646022-14-59
(Thread-Device-1) Measure added: 1629646023-13-44
(Thread-Device-3) Measure added: 1629646024-20-57
(Thread-Device-2) Measure added: 1629646024-20-42
(Thread-Device-4) Measure added: 1629646024-15-41
(Thread-Device-4) Measure added: 1629646025-19-57
(Thread-Device-1) Measure added: 1629646025-12-43
(Thread-Device-2) Measure added: 1629646026-17-45
(Thread-Device-3) Measure added: 1629646027-21-48
(Thread-Device-1) Measure added: 1629646028-17-47
(Thread-Device-2) Measure added: 1629646029-14-52
(Thread-Device-4) Measure added: 1629646030-19-44
(Thread-Device-4) Sending measure
(Thread-Device-4) Sent data in: 3004.074µs
(Thread-Device-3) Measure added: 1629646031-15-51
(Thread-Device-4) Measure added: 1629646031-16-41
(Thread-Device-3) Sending measure
(Thread-Device-3) Sent data in: 0.000µs
(Thread-Device-1) Measure added: 1629646032-18-46
(Thread-Device-1) Sending measure
(Thread-Device-1) Sent data in: 0.000µs
(Thread-Device-2) Measure added: 1629646034-13-58
(Thread-Device-2) Sending measure
(Thread-Device-2) Sent data in: 0.000µs
```

Screenshot terminale del programma device.py in esecuzione

```
(Thread-UDP-Receiver) Received 72 bytes from 192.168.1.5
(Thread-UDP-Receiver) Received 92 bytes from 192.168.1.4
(Thread-UDP-Receiver) Received 112 bytes from 192.168.1.2
(Thread-UDP-Receiver) Received 112 bytes from 192.168.1.3
(Thread-TCP-Sender) Sent data in: 0.000µs
(Thread-TCP-Sender) Response from server: OK
(Thread-UDP-Receiver) Received 52 bytes from 192.168.1.4
(Thread-UDP-Receiver) Received 112 bytes from 192.168.1.5
(Thread-UDP-Receiver) Received 112 bytes from 192.168.1.2
(Thread-UDP-Receiver) Received 92 bytes from 192.168.1.3
(Thread-TCP-Sender) Sent data in: 0.000µs
(Thread-TCP-Sender) Response from server: OK
```

Screenshot terminale del programma gateway.py in esecuzione

```
(Thread-TCP) 192.168.1.5-2021-08-22 17:27:04-15-41
(Thread-TCP) 192.168.1.5-2021-08-22 17:27:05-19-57
(Thread-TCP) 192.168.1.5-2021-08-22 17:27:10-19-44
(Thread-TCP) 192.168.1.4-2021-08-22 17:27:02-14-59
(Thread-TCP) 192.168.1.4-2021-08-22 17:27:04-20-57
(Thread-TCP) 192.168.1.4-2021-08-22 17:27:07-21-48
(Thread-TCP) 192.168.1.4-2021-08-22 17:27:11-15-51
(Thread-TCP) 192.168.1.2-2021-08-22 17:27:00-13-43
(Thread-TCP) 192.168.1.2-2021-08-22 17:27:03-13-44
(Thread-TCP) 192.168.1.2-2021-08-22 17:27:05-12-43
(Thread-TCP) 192.168.1.2-2021-08-22 17:27:08-17-47
(Thread-TCP) 192.168.1.2-2021-08-22 17:27:12-18-46
(Thread-TCP) 192.168.1.3-2021-08-22 17:27:01-20-45
(Thread-TCP) 192.168.1.3-2021-08-22 17:27:04-20-42
(Thread-TCP) 192.168.1.3-2021-08-22 17:27:06-17-45
(Thread-TCP) 192.168.1.3-2021-08-22 17:27:09-14-52
(Thread-TCP) 192.168.1.3-2021-08-22 17:27:14-13-58
(Thread-TCP) Sending response
```

Screenshot terminale del programma server.py in esecuzione