Relazione progetto programmazione di reti 2021

Benazzi Daniel

Per il progetto ho scelto la **traccia numero uno**, quindi il progetto IoT, che chiede di simulare 4 dispositivi IoT, un Gateway ed un Server; i dispositivi raccolgono dati sull'ambiente, questi dati sono inviati al gateway che li elabora ed inoltra al server.
Il gateway fa da ponte tra due sottoreti diverse.

Struttura generale

Organizzazione dei moduli

Il progetto si suddivide in **cinque moduli python**, di cui: due di utilità e tre contenenti il codice effettivo. (fig. 1)

I tre moduli fondamentali sono: **iot_device.py**, **gateway.py** e **server.py**

In **iot_device.py** sono simulati i quattro dispositivi IoT sotto forma di thread di esecuzione; questi inviano dati al gateway tramite una **connessione UDP** ogni n secondi dove n viene deciso dalla variabile ***__seconds_to_next_send.***

In **gateway.py** vengono ricevuti i dati dai quattro device IoT attraverso una connessione UDP aperta in ascolto sulla porta 50.000. I dati vengono salvati momentaneamente in un dizionario organizzato per indirizzo IP simulato dei dispositivi.
Ricevuti i dati da tutti i dispositivi vengono uniti in un unico messaggio che sarà inoltrato al server attraverso una connessione TCP.

In **server.py** si attende una **connesione TCP** sulla porta 51.000; all'apertura di una nuova connessione viene letto il messaggio in arrivo e verificato se il mittente era il gateway, in caso affermativo i dati contenuti vengono **stampati a video**.

---

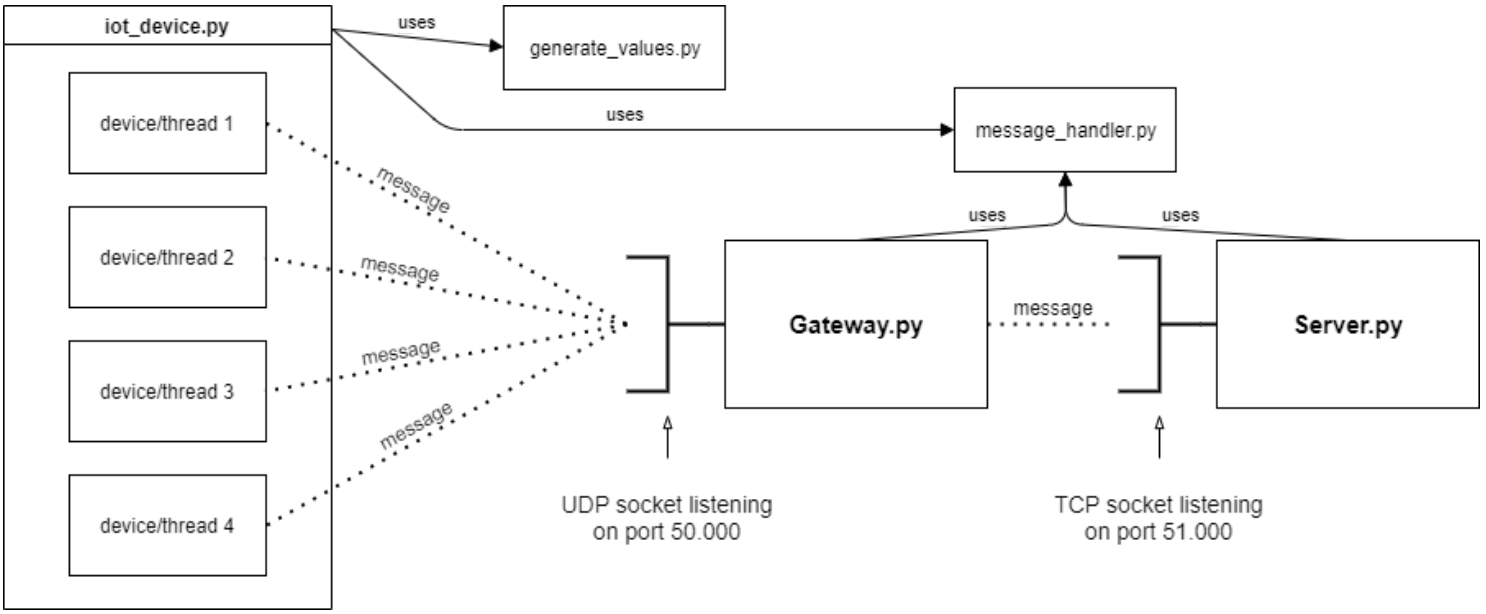
dimensione dei buffer

Il buffer della connessione UDP nel gateway ha dimensione **1024 byte**.
Il buffer delle connessioni TCP nel server ha dimensione **4096 byte**.

I 2 moduli di utilità sono **generate_values.py** e **message_handler.py**

In **generate_values** sono generati in maniera casuale i valori per temperatura ed umidità. Il range dei valori viene specificato da quattro variabili.

In **message_handler** vengono gestite le intestazioni simulate dei messaggi; al messaggio effettivo sono aggiunti e rimossi gli **indirizzi IP e MAC** simulati dei dispositivi.



Struttura dei messaggi scambiati

Siccome i dispositivi sono virtuali, e comunicano tramite l'**interfaccia di loopback** della macchina, per simulare la distribuzione dei dispositivi in sottoreti diverse **viene aggiunto un header** ai messaggi scambiati.
L'header contiene l'**indirizzo IP** e l'**indirizzo MAC** simulato del mittente e del destinatario.
L'aggiunta e la rimozione dell'header vengono gestite a livello applicazione prima di un invio e a seguito di una ricezione.
I messaggi hanno la forma mostrata nella tabella sottostante.

source MAC address	destination MAC address
source IP address	destination IP address
message data ...	

strutture singole

Seguono le descrizioni in dettaglio degli elementi principali del progetto. In particolare i thread attivi ed il loro funzionamento.

dispositivo iot

funzioni

- *create_values_for_a_day*: ritorna una stringa contenente i valori casuali dei dati di temperatura ed umidità
- *send_to_gateway*: funzione utilizzata dai thread per inviare al gateway un messaggio. Utilizza una variabile **mutex** per evitare che thread diversi inviino messaggi nello stesso istante.
- *device*: funzione che identifica un singolo dispositivo IoT. esegue un loop infinito contenente: attesa, creazione messaggio, invio.
- *signal_handler*: funzione delegata per la gestione di SIGINT. alla ricezione imposta la flag *__stop_devices* a True.

compiti dei thread

Ad esecuzione del programma sono attivi cinque thread: il **thread principale** e altri **quattro thread** che simulano i singoli dispositivi iot.

Il **thread principale** all'avvio del programma lancia gli altri thread, definisce la funzione per la gestione di SIGINT (ctrl + c), ed attende in un loop infinito per gestire i segnali.

I **thread secondari** simulano l'azione dei dispositivi IoT. Ogni 24 ore generano un messaggio e lo mandano al gateway

variabili utilizzate

__seconds_to_next_send: definisce ogni quanti secondi i dispositivi IoT inviano dati al gateway.

__stop_devices: variabile controllata dai vari thread per sapere quando terminare l'esecuzione.

__gateway_ip/mac/port: informazioni sul gateway per la connessione.

__devices_address_map: dizionario contenente indirizzi MAC ed IP dei dispositivi IoT.

gateway

funzioni

- *process_message*: questa funzione **esegue in un thread dedicato** creato da *receive_from_device*.
  La funzione **processa il messaggio** ricevuto come argomento;

aggiunge ad ogni riga del messaggio l'indirizzo IP del dispositivo specificato e lo salva il messaggio nel dizionario *__client_message_map.*
Se il numero di messaggi salvati corrisponde al numero di dispositivi IoT conosciuti sblocca il thread in attesa nella funzione *send_to_server* impostando l'evento *__send_wait*, poi elimina i messaggi precedenti dal dizionario*.*

- *receive_from_device*: Attende di ricevere dei dati dal socket UDP e, se il mittente era uno dei dispositivi IoT, crea un thread per gestirli.
- *send_to_server*: La funzione attende in un loop che la variabile *__send_wait* venga sbloccata poi esegue i seguenti compiti: crea un nuovo socket TCP, costruisce il messaggio completo dai singoli presenti in *__client_message_map*, si connette al server ed invia il messaggio.
- *signal_handler*: funzione delegata per la gestione di SIGINT. imposta la flag *__stop_gateway* a True e sblocca *send_to_server.*

compiti dei thread

Nel gateway sono attivi **tre thread** più un numero variabile di thread dipendente per la **gestione dei messaggi** ricevuti dai dispositivi IoT.
I tre thread sono: **thread principale**, **thread di ricezione**, **thread di invio**.

Il **thread principale** crea un socket di ascolto UDP che viene passato al thread di ascolto, crea ed avvia i thread di ricezione ed ascolto e poi attende in un loop infinito per la gestione degli interrupt.

Il **thread di ricezione** attende i messaggi dalla porta UDP e crea thread per la loro gestione.

Il **thread di invio** attende lo sblocco della variabile *__send_wait* per inviare i dati al server attraverso una connessione TCP creata sul momento.

variabili utilizzate

__stop_gateway: variabile controllata dai thread per sapere se interrompere l'esecuzione.
__server_ip/mac/port: informazione per la connessione al server.
__gateway_client_ip/mac e *__udp_address*: informazioni dell'interfaccia del gateway verso i dispositivi IoT.
__gateway_server_ip/mac: informazioni dell'interfaccia del gateway verso il server.
__client_message_map: dizionario che lega i dispositivi ai dati inviati.
__clients_number: numero di dispositivi conosciuti dal gateway.
__clients_lock: mutex per l'accesso a *__client_message_map* e *__clients_served.*
__clients_served: numero di client di cui il messaggio si trova nel dizionario.

__send_wait/end: eventi per la sincronizzazione dei thread di invio e di processamento del messaggio.

server

funzioni

- *receive_message*: legge i dati dalla connessione passata come argomento. Utilizza un selector per non attendere in maniera bloccante sulla connessione TCP, ai client viene data una **finestra di dieci secondi** per inviare i dati. Se il mittente del messaggio era il gateway stampa a video le informazioni contenute.
- *accept_connections*: funzione per accettare nuove richieste di connessione TCP; Attende sulla variabile *__accept_wait* fino a che non viene sbloccata dalla funzione *connection_request_notifier* poi crea un nuovo thread per la gestione del messaggio.
- *connection_request_notifier*: controlla per mezzo di un selettore se sono presenti nuove richieste di connessione TCP, in caso lo siano sblocca la variabile *__accept_wait*.
- *signal_handler*: funzione delegata per la gestione di SIGINT. imposta la flag *__stop_server* a True.

compiti dei thread

Nel server sono attivi **tre thread** più un numero variabile di thread per la **gestione dei messaggi** ricevuti sulla porta TCP.
I tre thread sono: **thread principale**, **thread per l'apertura delle connessioni**, **thread di notifica delle nuove connessioni**.

Il **thread principale** crea la connessione TCP e la passa ai thread per la connessione e la notifica di nuove connessioni, crea gli altri thread ed attende in un loop per la gestione degli interrupt.

Il **thread per l'apertura delle connessioni** attende la richiesta di una nuova connessione e delega un thread per la sua gestione
Il **thread di notifica delle nuove connessioni** gestisce la variabile *__accept_wait* sbloccandola ad una nuova richiesta di connessione.

variabili utilizzate

__stop_server: variabile controllata dai thread per sapere se interrompere l'esecuzione.

__gateway_ip/mac: informazioni dell'interfaccia di rete del server.

__*tcp_address*: indirizzo del socket TCP.

__*accept_wait*: variabile per la sincronizzazione.