Progetto di Programmazione di Reti 2021/2022

Alessandro Venturini, matr. 0000914531
e-mail. alessandro.venturin6@studio.unibo.it
29/06/2022

Indicazioni per l'utilizzo

È necessario eseguire i programmi con una versione piuttosto recente di Python (es: Python 3.9) in quanto sono state utilizzate alcune nuove librerie di *typehinting*. È inoltre necessario aver installato *Tkinter*.

I file sorgenti sono situati nella cartella "src", si assume che tutti i seguenti comandi vengano eseguiti all'interno di tale cartella.

Si consiglia di:

- Avviare il Gateway: *python3 gateway.py*
- Avviare il Client: *python3 client.py*
- Avviare i Droni: *python3 drone.py* (ognuno su una console dedicata)

  È possibile in realtà avviare un numero a piacere di Droni. (3 se si vuole restare fedeli alla traccia)
  È possibile connettere nuovi Droni in qualsiasi momento.

Il client può connettersi e disconnettersi a piacimento. Per quanto riguarda invece i Droni si è assunto che il sistema funzioni senza interruzioni e che a questi, una volta connessi, non sia consentito disconnettersi.

Per chiudere il sistema è sufficiente utilizzare la scorciatoia da tastiera Ctrl+C sulla console del Gateway e su quelle dei Droni.

Nelle prime righe dei file gateway.py e drone.py sono state predisposte due variabili per favorire il testing del sistema:

- PRINT_DEBUG: se settata a True attiva il log di informazioni utili e più approfondite.
- SIMULATE_PACKET_LOSS: se settata a True simulerà la perdita di pacchetti UDP e attiverà il log di informazioni utili e più approfondite.

Client (client.py)

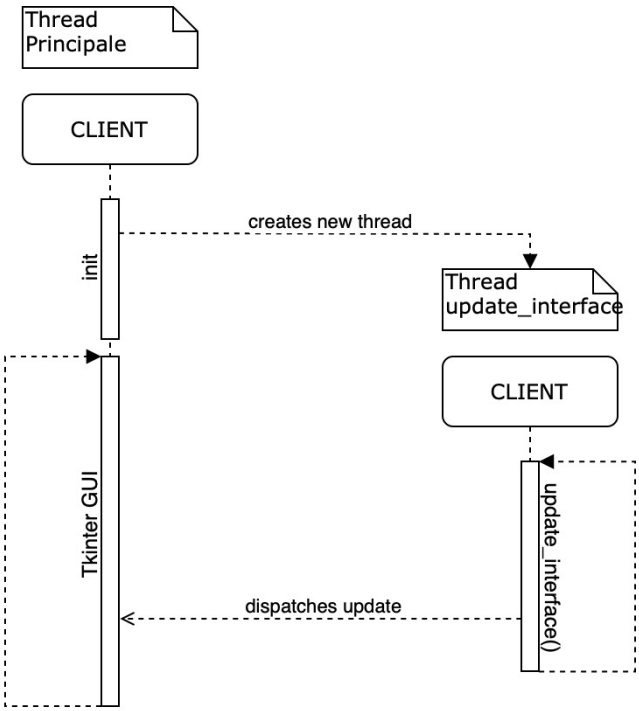
Per rendere l'utilizzo del sistema più fruibile è stata realizzata una semplice interfaccia grafica con *Tkinter*.

È possibile vedere lo stato di ogni Drone ed effettuare una nuova richiesta di spedizione inserendo l'identificativo del drone in quesitone, l'indirizzo e cliccando il tasto Invio.

Sul *thread* principale viene stabilita la connessione con il Gateway e vengono gestiti l'interfaccia grafica, gli eventi e l'invio delle richieste di spedizione.

Viene poi predisposto un *thread* secondario che si occupa di ricevere messaggi dal Gateway, inserirli in una coda e che viene controllata ciclicamente dal *thread* principale al fine di mantenere l'interfaccia aggiornata.

Diagramma dei *thread* del Client:



Comunicazione TCP (utils.py)

La comunicazione tra Client e Gateway avviene attraverso una *socket* TCP, questo rende lo scambio di messaggi piuttosto semplice in quanto si può contare su un protocollo che garantisce affidabilità.

Il tipo di comunicazione offerto dalle *socket* TCP è di tipologia *stream*, essendo però più semplice lavorare con messaggi "discreti" si è deciso di implementare delle funzioni di utilità nel file "utils.py".

Assumendo che la dimensione dei messaggi sia rappresentabile con 4 byte, è stato deciso che ogni messaggio includa 4 byte iniziali con cui descrivere la lunghezza effettiva del messaggio, in questo modo è possibile accedere allo *stream* di dati sotto forma di messaggi "discreti". Quindi i buffer utilizzati nella *socket* TCP sono sempre il minimo indispensabile per la ricezione dell'intero messaggio.

Drone (drone.py)

A parte per la logica di comunicazione tra Drone e Gateway, che verrà illustrata in maniera completa più avanti, il resto del codice del Drone è banale e quasi auto-esplicativo.

Viene utilizzato solo e unicamente il *thread* principale. Il Drone, una volta effettuata la connessione con il Gateway, entra in un ciclo nel quale:

1. Segnala al Gateway di essere disponibile per una nuova consegna.
2. Attende una richiesta di spedizione.
3. Ricevuta una richiesta simula l'esecuzione e le tempistiche di una spedizione.
4. Torna al punto 1.

Gateway (gateway.py)

Il Gateway è il componente del sistema nel quale risiede la maggior parte della logica di funzionamento.

All'inizio dello script vengono settate alcune variabili globali:

- TCP/UDP_ADDRESS: sono gli indirizzi a cui legare le *socket* e sui quali il Gateway accetterà le connessioni dal Client e dai Droni.
- ACCEPTING_DRONES_THREAD: utilizzata come riferimento al *thread* che si occupa di accettare nuove connessioni con i Droni, è necessaria per poter poi chiudere correttamente il *thread* prima di interrompere l'esecuzione.
- running: mantiene l'informazione riguardante lo stato del Gateway ovvero se è in normale esecuzione o se sta terminando.
- client_is_connected: viene utilizzata per tenere traccia del fatto che il Client sia connesso o meno.
- client_socket: socket attraverso la quale comunicare con il Client.
- should_update_client_console: se settata a True il *thread* principale si occuperà di aggiornare l'interfaccia del client il prima possibile.
- connected_drones: è un dizionario che associa ad ogni Drone il suo indirizzo IP. (Per mantenere lo stato di ogni drone si è utilizzata la classe Drone (situata in DTOs.py)

```python
TCP_ADDRESS: Address = ('127.0.0.1', 8080)
UDP_ADDRESS: Address = ('127.0.0.1', 8081)
ACCEPTING_DRONES_THREAD: Thread
running: bool = True
client_is_connected: bool = False
client_socket: socket
should_update_client_console: bool = False # used to notify the main thread that client's console needs an update.

connected_drones: dict[Address, Drone] = {}
```

La prima cosa che avviene poi è la creazione di un *thread* che si occupi di accettare le connessioni di nuovi Droni. (Più avanti si troverà una descrizione più dettagliata)

Viene poi inizializzata la *socket* per comunicare con il Client, si attende la connessione di quest'ultimo e poi si entra in un loop che gestisce l'aggiornamento dell'interfaccia del Client e la ricezione di richieste di spedizione. Ogni comunicazione con il Client avviene sempre sul *thread* principale.

Il normale flusso di comunicazione tra Client e Gateway è il seguente:

1. Gateway, se necessario, aggiorna l'interfaccia del Client attraverso un tipo di messaggio ben definito (GatewayInterfaceDTO situato in DTOs.py).
2. Poi attende per mezzo secondo messaggi dal Client.
3. Se non ne riceve torna al punto 1, in caso contrario procede a gestire la richiesta di spedizione (ShippingRequestDTO situata in DTOs.py).
4. Torna al punto 1.

Comunicazione tra Droni e Gateway

Questa è la parte più complessa del sistema, per realizzarla si sono presi alcuni concetti del protocollo TCP e si è tentato di emularli, solo fino ad un certo livello di fedeltà, utilizzando delle *socket* UDP.

Sono state effettuate delle assunzioni:

- Essendo il sistema, chiaramente, non di tipo *Real Time* è accettabile gestire alcune situazioni andando a sacrificare un po' di reattività in favore di una implementazione meno complessa.
- Il *Round Trip Time* sarà sempre inferiore ad un secondo.
- Le connessioni tra Droni e Gateway avvengono sempre e solamente una alla volta. (Nel caso in cui durante *l'hanshake* con il Drone X anche un Drone Y dovesse tentare di collegarsi i suoi pacchetti verrebbero ignorati.
- Siccome è possibile che i pacchetti vengano persi si assume che prima o poi almeno uno di essi arrivi a destinazione, questa assunzione consente di non dover inserire dei *timeout* sulle operazioni principali in quanto esse prima o poi andranno a buon fine.
- Si è assunto che per la dimensione del buffer utilizzata per lo scambio di pacchetti con i Droni siano sufficienti 4096 byte. (Questo, quindi, limita la lunghezza dell'indirizzo di spedizione, anche se il quantitativo di memoria è già molto generoso.)
- Non è stata predisposta la possibilità di un Drone di disconnettersi una volta connesso.
- Il sistema non sarà obiettivo di attacchi malevoli. (Ad esempio, i *sequence number* dei pacchetti non hanno un inizio casuale unicamente per semplicità di comprensione del protocollo)

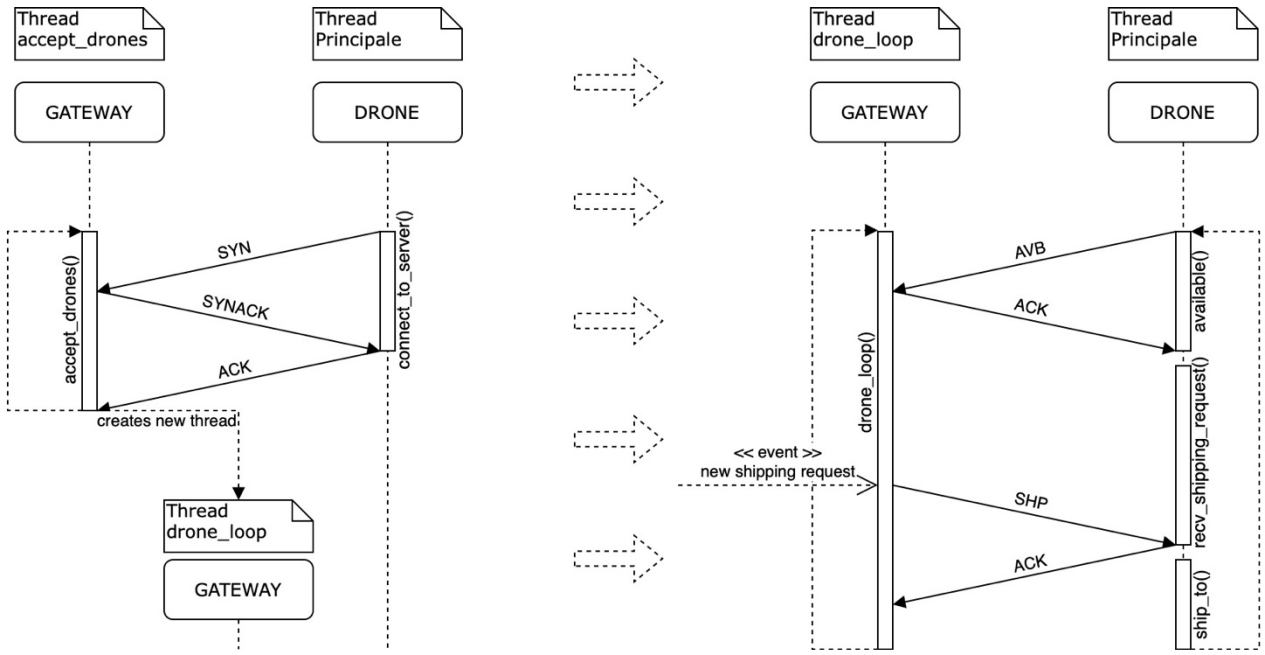
I Pacchetti (Packet.py):

La classe "Packet" (situata in Packet.py) rappresenta l'unica tipologia di messaggio scambiata tra Droni e Gateway, i tipi effettivamente disponibili sono:

- SYN (Handshake)
- SYNACK (Handshake)
- ACK (Handshake e Conferma di ricezione degli altri pacchetti)
- AVB (Pacchetto inviato dal Drone che segnala di essere DISPONIBILE)
- SHP (Pacchetto inviato dal Gateway per effettuare una richiesta di spedizione)

ognuno con i suoi attributi peculiari.

Il diagramma qui sotto mostra a sinistra la procedura di handshake e a destra il normale flusso di comunicazione che invece si ripete all'infinito una volta instaurata la connessione.



Fase di connessione (handshake):

All'avvio il Gateway crea un thread sul quale verrà eseguita in maniera continuativa la funzione "accept_drones()". In questa funzione viene inizializzata una socket ("drones_socket") che avrà il solo scopo di instaurare una connessione con i Droni e non di comunicarci.

Avviene quindi un classico *Three-Way Handshake* tra Drone e Gateway, con l'unica differenza che nel pacchetto SYNACK viene aggiunta l'indicazione di una porta (il perché verrà spiegato qui sotto).

Una volta che l'handshake è terminato con successo, il Gateway avrà già creato una nuova *socket* che verrà utilizzata per comunicare esclusivamente con il Drone che si è connesso. Anche il Drone d'ora in poi utilizzerà l'indirizzo della nuova socket che conosce grazie all'informazione ricevuta nel pacchetto SYNACK. Il Gateway si occupa anche di creare un *thread* dedicato esclusivamente al Drone nel quale verrà eseguita la funzione "drone_loop()"

La possibilità che l'ultimo ACK dell'handshake venga perso è gestita all'interno delle altre funzioni di comunicazione.

Flusso di comunicazione:

1. Il Gateway attende il messaggio AVB dal Drone che indica la DISPONIBILITA' per una nuova consegna.
2. Il Drone invia AVB (avaliable()) e attende l'ACK del Gateway.
3. Il Gateway ricevuto AVB invia l'ACK.
4. Il Drone ricevuto l'ACK aspetta SHP (recv_shipping_request())
5. Il Gateway invia SHP e attende l'ACK del Drone
6. Il Drone invia l'ACK e parte per la spedizione (ship_to()).
7. Il Gateway ricevuto l'ACK torna al punto 1.
8. Il Drone terminata la spedizione torna al punto 2.

Se non viene ricevuto l'ACK entro un secondo dall'invio dei pacchetti AVB o SHP essi vengono considerati persi e ritrasmessi ad oltranza fino alla ricezione di un ACK.

La situazione si complica nel caso in cui i pacchetti AVB o SHP vengono effettivamente recapitati correttamente ma l'ACK viene perso, in questo caso la gestione viene effettuata nel seguente modo e sfrutta i "seq_num" dei pacchetti AVB o SHP e gli "ACK_num" dei pacchetti ACK.

Situazioni non ordinarie possibili e relativa gestione:

1) Ipotizzando la seguente situazione:

[GATEWAY] -----> SHP -----> [DRONE]

[GATEWAY] X----- ACK <----- [DRONE]

Siccome l'ACK è stato perso il Gateway continuerà ad inviare SHP ma il Drone si sarà già spostato nella fase di spedizione. Per gestire questa situazione il Drone continuerà ad ascoltare per eventuali SHP duplicati e ritrasmetterà l'ACK se necessario. (Come osservabile qui sotto)

[GATEWAY] -----> dSHP -----> [DRONE]

[GATEWAY] <----- dACK <----- [DRONE]

L'utilizzo dei *sequence number* per riconoscere eventuali pacchetti duplicati consente anche di gestire il caso in cui la spedizione dovesse durare talmente poco da non permettere al drone di ritrasmettere l'ACK, in quel caso verrebbe mandata nuovamente un'AVB, questa verrebbe trattata come se fosse l'ACK mancante.

2) Ipotizzando quest'altra situazione:

[GATEWAY] <----- AVB <----- [DRONE]

[GATEWAY] -----> ACK -----X [DRONE]

Siccome l'ACK è stato perso il Drone continuerà ad inviare AVB ma il Gateway si sarà già spostato nella fase di inoltro di una spedizione. Per gestire questa situazione se il Gateway, dopo aver inviato una SHP dovesse ricevere un AVB duplicato allora lo ignorerebbe, contando sul fatto che il Drone interpreterà SHP come l'ACK che ha perso. (Come osservabile qui sotto)

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] -----> SHP -----> [DRONE]

[GATEWAY] Ignora l'AVB duplicato in quanto SHP verrà interpretato dal drone come ACK.

[DRONE] Interpreta SHP come l'ACK perso.

[GATEWAY] <----- ACK <----- [DRONE]

L'utilizzo dei *sequence number* per riconoscere eventuali pacchetti duplicati consente anche di gestire il caso in cui, il Gateway dovesse inviare immediatamente dopo l'ACK (perso) un pacchetto SHP al quale il Drone potrebbe rispondere con un AVB duplicato che, grazie al "seq_num" verrà ignorato invece che accettato erroneamente.

3) Un caso particolare sarebbe quello nel quale l'ACK dell'hanshake venisse perso. Il Drone invierebbe subito un AVB ma questo non verrebbe ricevuto dal Gateway che ancora aspetta l'ACK e che continuerà a mandare SYNACK duplicati e questo porterebbe il Drone a mandare continuamente AVB duplicati che si accumulerebbero nella socket. Sempre i *sequence number* consentono al Gateway di scartare gli AVB accumulati durante questa fase.

[GATEWAY] <----- SYN <----- [DRONE]

[GATEWAY] -----> SYNACK -----> [DRONE]

[GATEWAY] X----- ACK <----- [DRONE]

[GATEWAY] <----- AVB <----- [DRONE]

[GATEWAY] -----> dSYNACK -----> [DRONE]

[GATEWAY] X----- dACK <----- [DRONE]

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] -----> dSYNACK -----> [DRONE]

[GATEWAY] <----- dACK <----- [DRONE]

[GATEWAY] -----> SHP -----> [DRONE]

[GATEWAY] Ignora il dAVB accumulato sulla socket dedicata del drone.

[GATEWAY] <----- ACK <----- [DRONE]

4) Una situazione che non causa problemi gravi e che per questo non è stata gestita è la seguente:

[GATEWAY] <----- SYN <----- [DRONE]

[GATEWAY] -----> SYNACK -----> [DRONE]

[GATEWAY] X----- ACK <----- [DRONE]

[GATEWAY] <----- AVB <----- [DRONE]

[GATEWAY] -----> dSYNACK -----> [DRONE]

[GATEWAY] <----- dACK <----- [DRONE]

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] -----> ACK -----X [DRONE]

Il Drone continuerebbe ad inviare AVB duplicati, si è deciso appunto per semplicità di non ascoltare per AVB duplicati in questa fase ma semplicemente di inviare una SHP quando richiesto, in questo modo il Drone la interpreterà come l'ACK perso e il tutto infine sarà stato impercettibile dal Client. L'unico svantaggio sarebbe appunto un piccolo spreco di banda causato dal continuo invio di AVB del drone fino alla ricezione di SHP. (Come osservabile qui sotto)

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] <----- dAVB <----- [DRONE]

[GATEWAY] -----> SHP -----> [DRONE]

[GATEWAY] Ignora i quattro AVB accumulati sulla socket dedicata del drone.

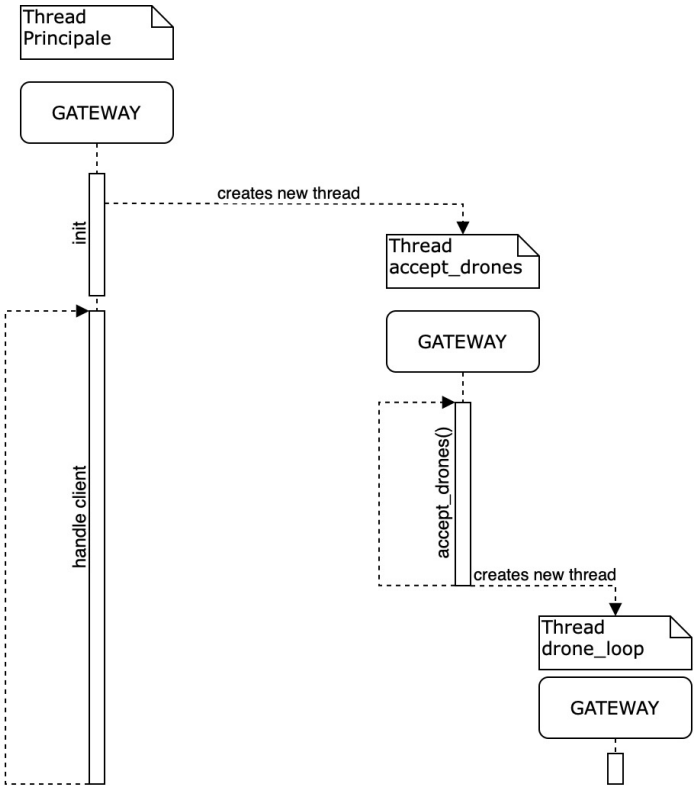
[GATEWAY] <----- ACK <----- [DRONE]

Infine, è da specificare che in realtà tutte le volte in cui un pacchetto non duplicato che non sia un ACK viene interpretato come ACK, esso non viene processato subito… Esso viene ignorato, il ricevente si sposterà nella fase successiva dove lo gestirà correttamente quando il mittente lo ritrasmetterà. Questo anche se causa lo spreco di un pacchetto consente di rendere molto più semplice il codice che gestisce questa situazione.

Tutte queste, e anche altre, situazioni possono comunque essere riprodotte (e monitorate attraverso l'output nelle console) settando a True la variabile globale "SIMULATE_PACKET_LOSS" in entrambi i file

(gateway.py e drone.py). Se si volesse aumentare la probabilità di perdita dei pacchetti basterebbe modificare conseguentemente la funzione “should_be_lost()”.

Diagramma dei *thread* del Gateway:



Tempistiche di trasmissione dei pacchetti TCP e UDP.

Come già detto in precedenza, non essendo un sistema *Real Time*, queste tempistiche sono poco rilevanti.

Comunque si è ipotizzato di avere il Gateway in prossimità della stazione di partenza dei Droni e invece non si sono fatte assunzioni sulla distanza del Client dal Gateway. I Droni in questione sono realizzati per effettuare consegne a corto raggio e quindi si presuppone che il *RTT* sia sempre molto basso.

Una considerazione però da fare è che il protocollo realizzato basato su UDP rallenterà in maniera molto maggiore rispetto al TCP nei casi in cui la rete possa non funzionare in maniera ottimale, questo è chiaramente causato dai *timeout* di circa un secondo che si sono utilizzati nel protocollo basato su UDP.