Progetto droni

Corso di Programmazione di reti
Ingegneria e scienze informatiche
Università di Bologna

A.A. 2021-2022

Nome	Email	Matricola
Maicol Battistini	maicol.battistini@studio.unibo.it	0000988616

18 agosto 2022

Indice

1 | Introduzione

Il progetto droni si pone l'obiettivo di realizzare un semplice simulatore Python per lo scambio di messaggi tra client e droni mediante l'utilizzo di un gateway come intermediario tra i due.

2 | Guida utente

2.1 | Requisiti software

Per poter utilizzare l'emulatore è necessario assicurarsi di aver installato nel proprio dispositivo:

- Python 3.10+.
  Nota: il software *potrebbe* funzionare comunque su versioni precedenti, ma non è garantito il corretto comportamento con versioni precedenti alla 3.10.0)
- Utilità a riga di comando **pip** (per poter installare le librerie necessarie al funzionamento dell'applicazione, solitamente installato insieme al linguaggio Python)

2.2 | Download, installazione e avvio

È possibile scaricare il software tramite il repository Github del progetto.
Da ora in avanti vengono riportate le indicazioni per un utilizzo da emulatore di terminale. Non è garantita la corretta visualizzazione del software tramite altri software
Prima di avviare il software, occorre installare le librerie utilizzate dal simulatore tramite il comando pip:

```
pip install -r requirements.txt
```

È ora possibile avviare qualsiasi componente del progetto (client, drone, gateway) ma le prime due restitueranno un errore se il gateway non è in esecuzione. Si riporta una tabella con i comandi per eseguire il componente e il numero massimo di istanze che possono essere eseguite contemporaneamente in un sistema client-gateway-drone/i:

Componente	Comando	Numero massimo di istanze
Client	`python ./client.py`	1
Drone	`python ./drone.py`	253[1]
Gateway	`python ./gateway.py`	1

Tabella 2.1: Comandi per eseguire i componenti del progetto.

[1] Come da specifiche, un drone può essere assegnato a un indirizzo IP nella rete 192.168.1.0/24. Pertanto, i possibili indirizzi IP assegnabili ad un drone, considerando il gateway come primo dispositivo di questa rete con IP 192.168.1.1, sono 253.

2.3 | Utilizzo dei componenti

Per ogni componente descritto di seguito, vale la combinazione di tasti **CTRL+C** per uscire dal software e ritornare al terminale.

2.3.1 | Gateway

All'avvio il gateway rimarrà in attesa del client o droni che vogliono collegarsi. Verranno notificati d'ora in poi tutti i messaggi scambiati tra i client e i droni collegati ed anche quelli tra client-gateway (ad esempio la richiesta dei droni disponibili).

2.3.2 | Drone

All'avvio il drone richiederà un identificatore da assegnare al dispositivo, scelto liberamente, e di un indirizzo IP valido: verrà effettuata una validazione per verificare che l'indirizzo IP non sia già utilizzato da un altro drone o che appartenga alla rete. D'ora in poi il drone rimarrà in attesa di consegne da effettuare.

2.3.3 | Client

Il client è il componente più interattivo del progetto: dopo aver instaurato una connessione con il gateway, viene presentato un menù simile:

```
Select a command:
        s. Send shipment
        g. Get available drones
        e. Exit

 [s/g/e]:
```

nel quale è possibile scegliere tra 3 opzioni:

- **Send shipment** - Identificato dalla lettera **s**: permette di avviare la procedura per effettuare una spedizione con un drone disponibile
- **Get available drones** - Identificato dalla lettera **g**: recupera la lista dei droni disponibili per effettuare una spedizione e li mostra sotto forma di tabella, visualizzando ID, Indirizzo IP e stato (Disponibile/Non disponibile) del drone.
- **Exit** - Identificato dalla lettera **e**: esce dal software chiudendo la connessione con il gateway.

3 | Scelte progettuali

Per gestire le connessioni tra le varie parti si è ricorso al modello client-server tra i vari componenti. In particolare:

- Nella comunicazione TCP tra client e gateway, quest'ultimo è utilizzato come server TCP dato che rimane in ascolto delle richieste del client, che di fatto è il client dell'applicazione (colui che invia le richieste al server TCP del gateway)
- Nella comunicazione UDP tra droni e gateway, i droni svolgono il ruolo di client, mentre il gateway è il server UDP. Questa scelta è dovuta dal fatto che il gateway è il primo dispositivo che si avvia nel sistema, mentre i droni possono avviarsi successivamente e si rendono disponibili ad effettuare nuove consegne. Secondo questa logica i droni comunicano al gateway il loro stato, e rimangono in attesa di una risposta (una consegna da effettuare). In questo modo è stato anche possibile effettuare connessioni multiple al gateway tramite diversi droni, grazie al modello client-server (1 solo server (il gateway), più client (i droni))

Si è inoltre optato per una gestione dei droni non limitata a solo tre dispositivi per facilitare future estensioni del progetto, se saranno necessarie, e per non irrigidire il client e il gateway supportando al massimo tre droni. Infine, l'indirizzo IP assegnato al client (10.10.10.2) è stato scelto in quanto il gateway, primo dispositivo collegato alla rete, ha verosimilmente l'indirizzo IP 10.10.10.1. Si è quindi andati in ordine crescente prendendo il primo indirizzo IP disponibile dopo il gateway.

Di seguito una schematizzazione delle due sottoreti del gateway:

Tabella 3.1: Sottorete 1: Client-Gateway.

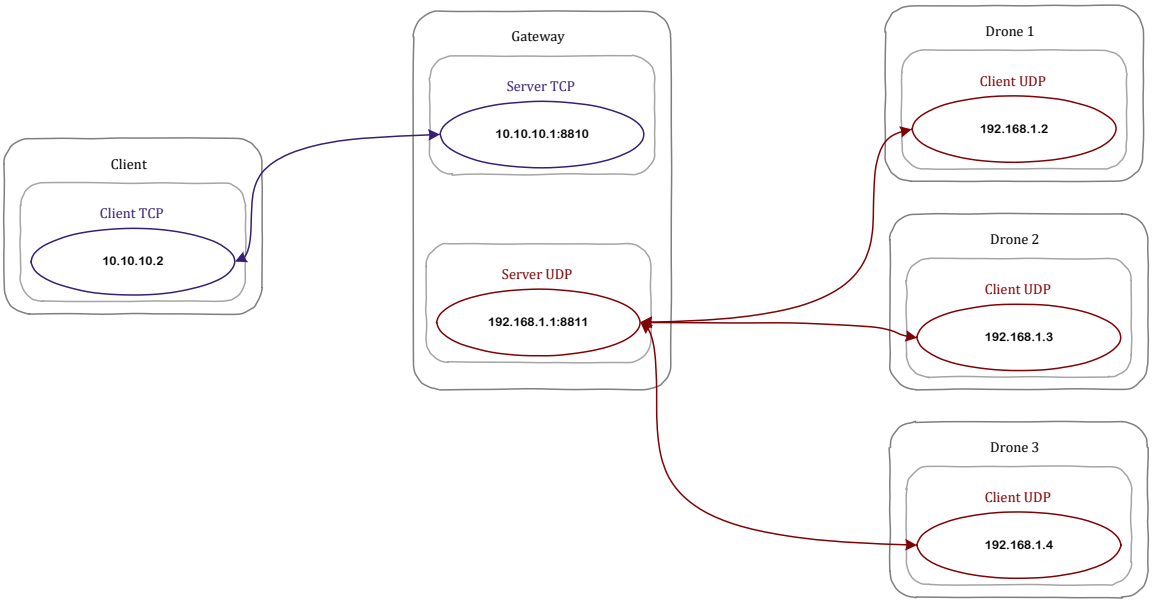
Dispositivo	Indirizzo IP (IPv4)	Maschera sottorete
Client	10.10.10.2	255.255.255.0
Gateway	10.10.10.1	255.255.255.0

Tabella 3.2: Sottorete 2: Droni-Gateway.

Dispositivo	Indirizzo IP (IPv4)	Maschera sottorete
Client	10.10.10.2	255.255.255.0
Gateway	10.10.10.1	255.255.255.0

Per brevità, nella tabella 3.2 vengono mostrati solo tre droni.

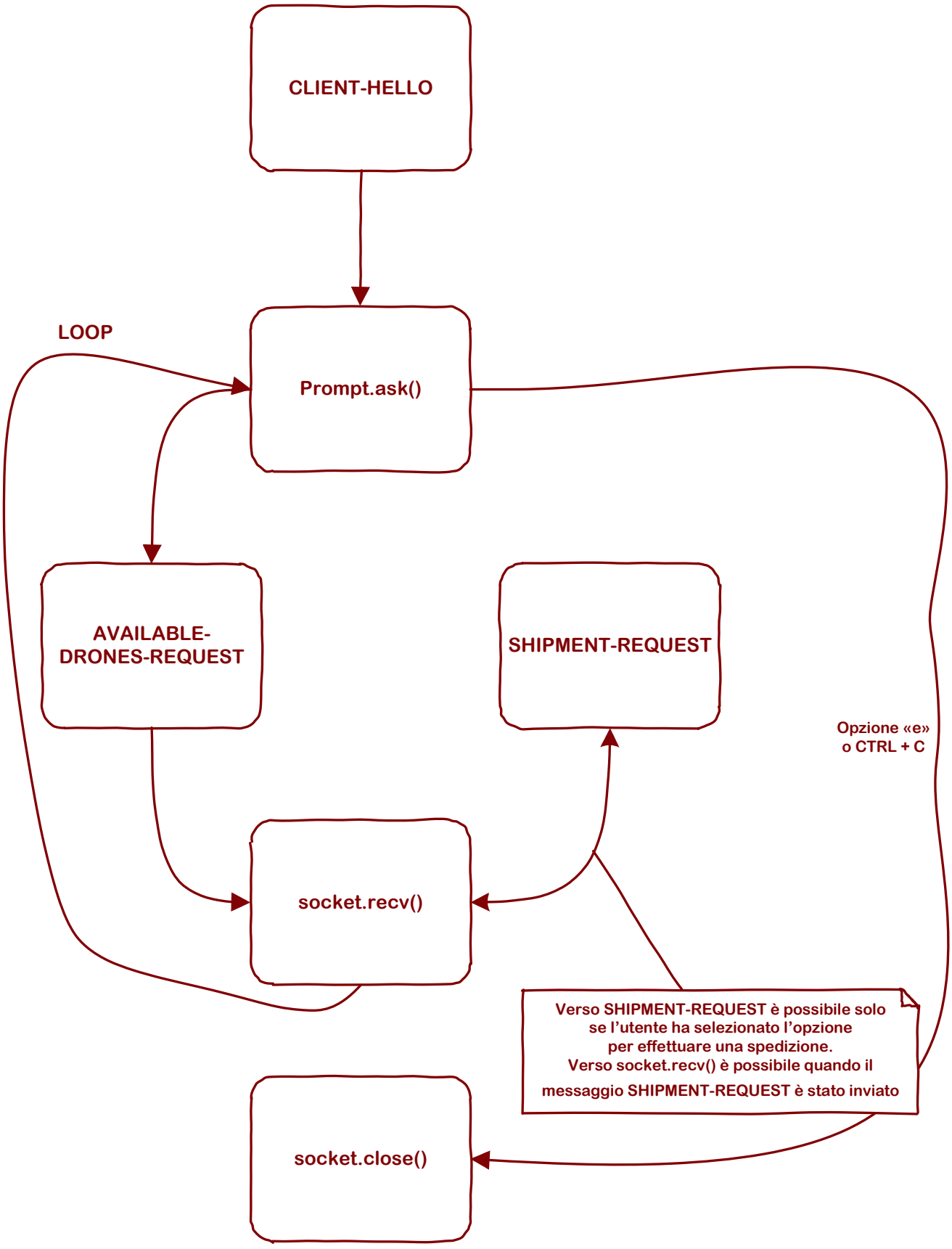
Infine, una schematizzazione dell'architettura client-server dell'applicazione, realizzato tramite socket:



4 | Implementazioni

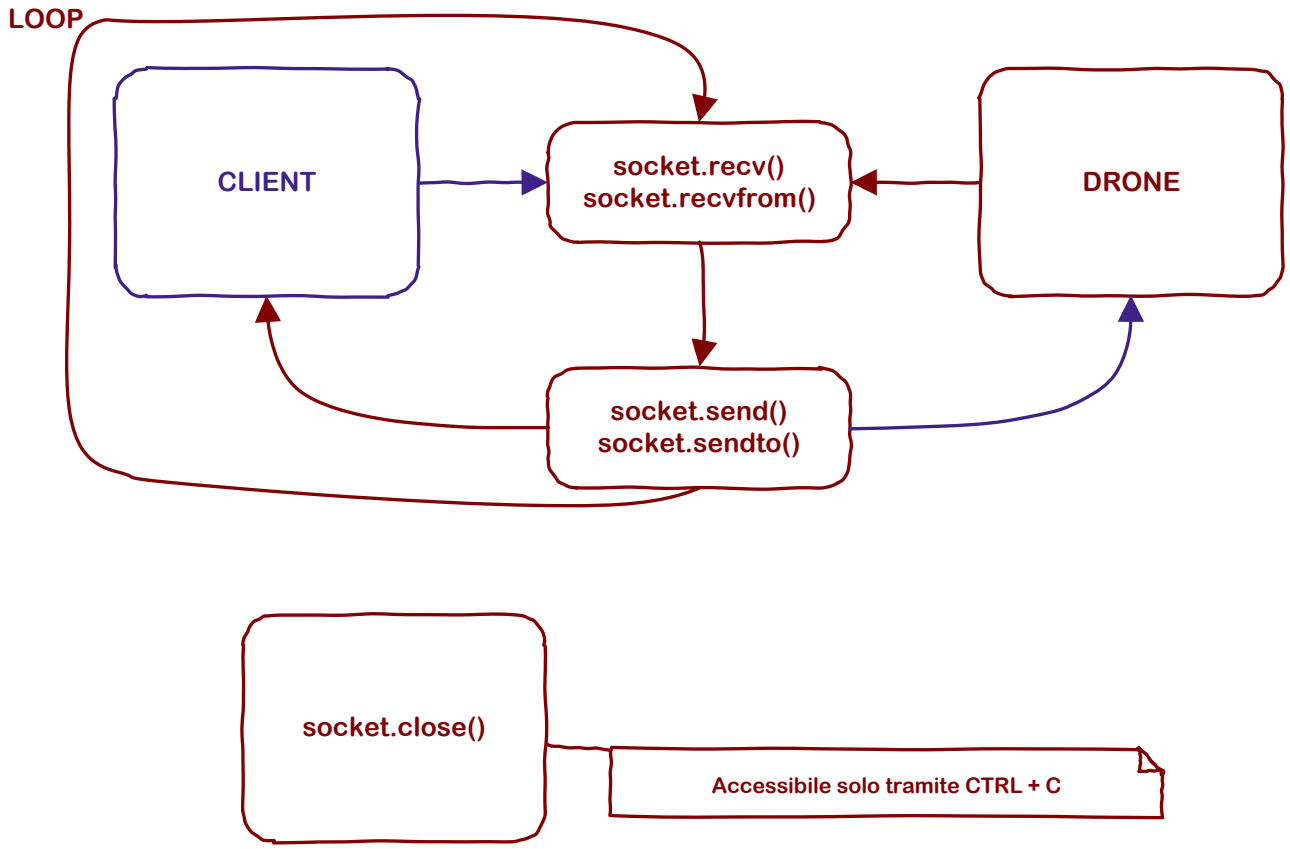
4.1 | Client

Il client appena viene eseguito instaura una connessione TCP con il gateway, segnalandoli il suo IP tramite il messaggio `CLIENT-HELLO <client_ip>` (simulando la connessione e il momento in cui il client fa conoscere il suo IP al gateway). Successivamente inizia un loop infinito in cui viene chiesta all'utente un'azione da effettuare, come già descritto nella Guida utente. In base al comando ricevuto viene inviato un messaggio al gateway e viene messo un thread in ascolto della risposta che verrà poi stampata a video al suo arrivo. Per il messaggio di richiesta dei droni disponibili viene "aspettato" il thread, cioè viene attesa una risposta dal gateway, in quanto è necessaria sia per poter presentare la lista dei droni con le loro disponibilità sia per poter effettuare una spedizione (dato che vengono mostrati i droni disponibili tra cui scegliere con quale effettuare la consegna).
Il ciclo si ripete con l'attesa di un nuovo comando e terminerà solo alla disconnessione del client tramite combinazione da tastiera (CTRL + C) o selezionando l'opzione di uscita tra i comandi disponibili.


CLIENT-HELLO
LOOP
Prompt.ask()
AVAILABLE-
DRONES-REQUEST
SHIPMENT-REQUEST
Opzione «e»
o CTRL + C
socket.recv()
Verso SHIPMENT-REQUEST è possibile solo
se l'utente ha selezionato l'opzione
per effettuare una spedizione.
Verso socket.recv() è possibile quando il
messaggio SHIPMENT-REQUEST è stato inviato
socket.close()

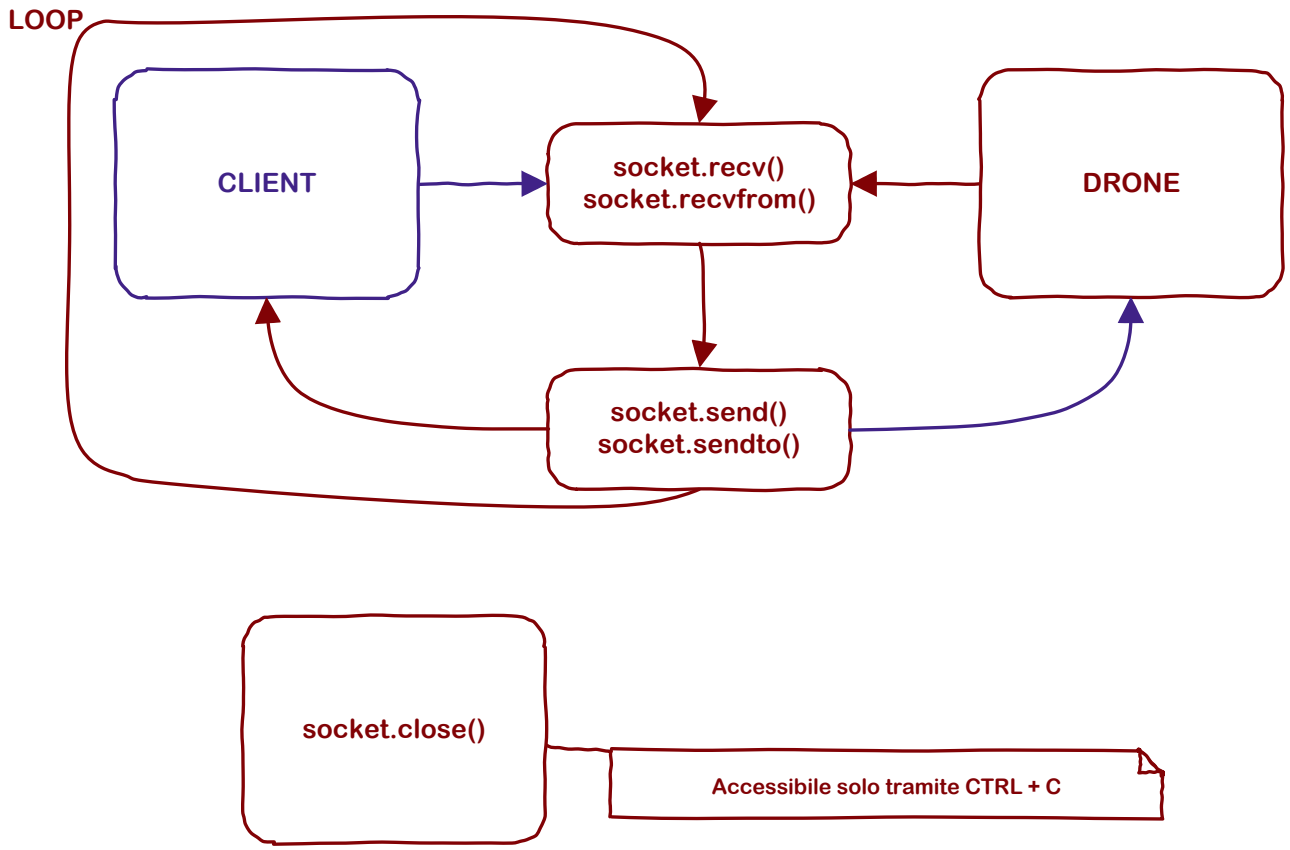
4.2 | Gateway

Al suo avvio il gateway crea i due socket per la connessione TCP con il client e quella UDP con i droni. In seguito viene creato un thread che si mette in ascolto sul socket UDP per la ricezione dei messaggi che i droni invieranno al socket quando si connetteranno e saranno disponibili e quando avranno effettuato una consegna assegnatali. In parallelo si attende anche la connessione del client nel loop principale: infatti, una volta che la connessione con il client è stata stabilita inizia un loop in cui si attendono richieste dal client che eventualmente sono da propagare ai droni, mentre nel thread creato in precedenza si attendono i messaggi che dai droni devono eventualmente arrivare al client.



4.3 | Drone

I droni per prima cosa si connettono al gateway e si identificano con il loro ID e il loro indirizzo IPV4, richiesti all'utente che inizializza il drone, tramite un messaggio `DRONE-READY <drone_id> <drone_ip>`. L'indirizzo IP viene validato sia dal drone sia dal gateway in fase di connessione tramite una espressione regex per verificarne l'appartenenza alla classe di indirizzi 192.168.1.0/24 e per verificare che sia valido (cioè che non coincida con l'IP del gateway (idealmente 192.168.1.1) e di broadcast (192.168.1.255). Appena connessi al gateway viene avviato un loop in cui attendono le spedizioni da effettuare stampandone l'indirizzo, in seguito comunicano l'avvenuta consegna attraverso il messaggio `DRONE-DELIVERED <drone_ip>` rendendosi nuovamente disponibili. Il ciclo si interrompe quando il drone si disconnetterà (simulato con la combinazione di tasti CTRL + C) con il messaggio `DRONE-CONNECTION-CLOSED <drone_ip>`.


LOOP
CLIENT
socket.recv()
socket.recvfrom()
DRONE
socket.send()
socket.sendto()
socket.close()
Accessibile solo tramite CTRL + C

5 | Strutture dati utilizzate

5.1 | Messaggi testuali

I messaggi sono stati strutturati seguendo il formato:

```
COMANDO <arg1> <arg2> ...
```

dove:

- `COMANDO` è il comando inviato, rappresentato in maiuscolo e con trattini se il comando è composto da più parole
- `<arg1>` e `<arg2>` sono due argomenti che possono essere passati al comando

Ogni parte del messaggio è quindi suddivisa da uno spazio: in questo modo i comandi possono essere composti da potenzialmente infiniti argomenti.

Di seguito le tabelle riassuntive dei comandi che possono inviare il client e i droni:

Tabella 5.1: Comandi utilizzati nella comunicazione tra client e gateway.

Comando	Descrizione
`AVAILABLE-DRONES-REQUEST`	Richiede al gateway la lista dei droni connessi, includendo anche la loro disponibilità.
`SHIPMENT-REQUEST <drone_ip> / <drone_id> <delivery_address>`	Invia al gateway una richiesta di spedizione all'indirizzo `<delivery_address>` da effettuare con il drone `<drone_ip>` / `<drone_id>`.
`DRONE-SHIPMENT-DELIVERED <drone_ip>`	Messaggio usato dal gateway per notificare al client l'effettiva consegna di una spedizione da parte del drone con IP `<drone_ip>`.
`DRONE-NOT-AVAILABLE <drone_ip>`	Messaggio usato dal gateway per notificare al client l'impossibilità di effettuare la spedizione utilizzando il drone con IP `<drone_ip>` in quanto non disponibile al momento.
`CLIENT-HELLO <client_ip>`	Comando utilizzato al client per notificare al gateway il prioprio IP (indicato dall'argomento `<client_ip>`).
`AVAILABLE-DRONES-RESPONSE <...drones_json>`	Messaggio usato dal gateway per restituire al client la lista dei droni disponibili (e non) in formato JSON tramite l'argomento `<...drone_json>`[1].

[1] L'argomento è volutamente *rest* (indicato con i tre puntini) per indicare che occupa più di un argomento nel comando.

Tabella 5.2: Comandi utilizzati nella comunicazione tra drone e gateway.

Comando	Descrizione
`DRONE-READY <drone_ip> <drone_id>`	Notifica al gateway la connessione effettuata dal drone con esso, indicando anche il proprio ID e IP.
`SHIPMENT-REQUEST <drone_ip> <delivery_address>`	Messaggio inviato dal gateway richiedendo una spedizione all'indirizzo `<delivery_address>`. Viene anche ricordato al drone il suo IP.
`DRONE-DELIVERED <drone_ip>`	Inviato al gateway per notificare il client dell'effettiva consegna di una spedizione da parte del drone con IP `<drone_ip>`.
`DRONE-CONNECTION-CLOSED <drone_ip>`	Inviato al gateway per notificare la disconessione del drone con IP `<drone_ip>` e la sua chiusura della connessione.
`DRONE-CONFIRMED`	Messaggio inviato al drone dal gateway per notificargli l'avvenuta connessione al gateway con ID e indirizzo IP specificati al momento della connessione.
`DRONE-ALREADY-CONNECTED`	Messaggio inviato al drone dal gateway per notificargli che è già connesso un drone con lo stesso indirizzo IP o ID specificati al momento della connessione.

5.2 | Dizionari

I due dizionari principali utilizzanti all'interno dell'emulatore sono nel client e nel gateway:

- Il dizionario nel client mantiene in memoria la lista dei droni disponibili e non. Viene aggiornato ad ogni richiesta con le informazioni ricevute (caricando il JSON come dizionario). Il dizionario ha la seguente struttura:

  ```
  dict[str, dict["id" | "available", str | bool]]
  ```

  Presenta quindi una struttura in cui la chiave è l'indirizzo IP del drone mentre il valore è un altro dizionario in cui sono presenti ID e disponibilità, accessibili tramite le relativi chiavi.

- Il dizionario nel gateway si occupa di tenere traccia dei droni connessi. Si tratta di un semplice dizionario dalla struttura chiave-valore dove la chiave è l'IP del drone e il valore la classe Drone, che tratteremo nel paragrafo sottostante:

  ```
  dict[str, Drone]
  ```

5.3 | Classi

La classe principale utilizzata nel gateway è Drone: è una semplice classe creata per raggruppare insieme le informazioni sul drone, così da approfittare di una maggiore tipizzazione rispetto ai dizionari.

5.3.1 | Enum

Gli enum vengono utilizzati per raggruppare i comandi in base al dispositivo (client o drone) ed è un ottimo modo per evitare di scrivere ogni volta i comandi in modo "hard-coded", cioè scrivendo il comando sotto forma di stringa. Con gli enum si chiama sempre una certa chiave (costante) che ha come valore il comando effettivo sotto forma di stringa

5.4 | Altre strutture minori

Vengono utilizzate delle liste e le tuple per il parsing dei comandi ricevuti, così da mantenere il comando come stringa e la lista degli argomenti come tupla

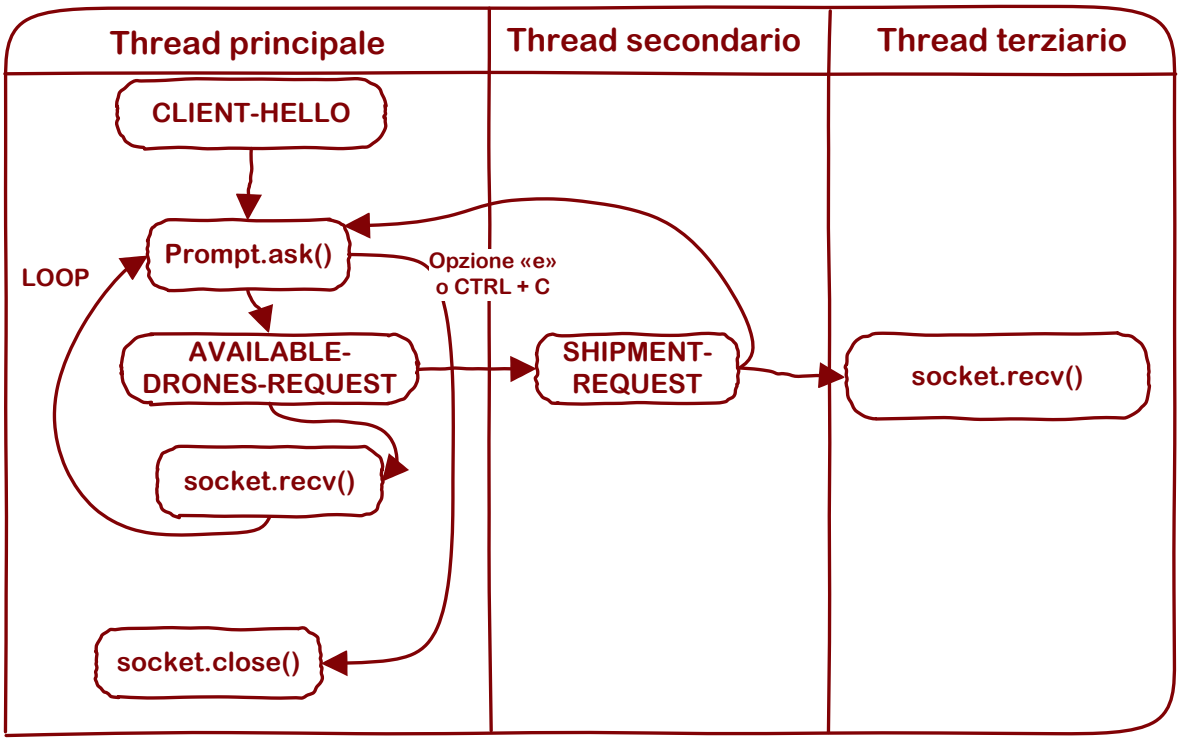
6 | Gestione dei threads

6.1 | Client

Nel client il thread principale rimane in attesa di input dall'utente. Appena un comando viene scelto dall'utente la richiesta viene inoltrata al gateway all'interno di un thread secondario per poter eseguire subito altri comandi. Il thread secondario che invia la richiesta al gateway crea un thread figlio che viene lasciato in attesa della risposta. In questo modo il thread padre può immediatamente rimettersi in attesa di input sulla console per nuovi comandi. Questo non vale se la richiesta da effettuare è recuperare i droni disponibili. In quel caso la risposta verrà aspettata anche dal thread principale, in tutti e due i casi:

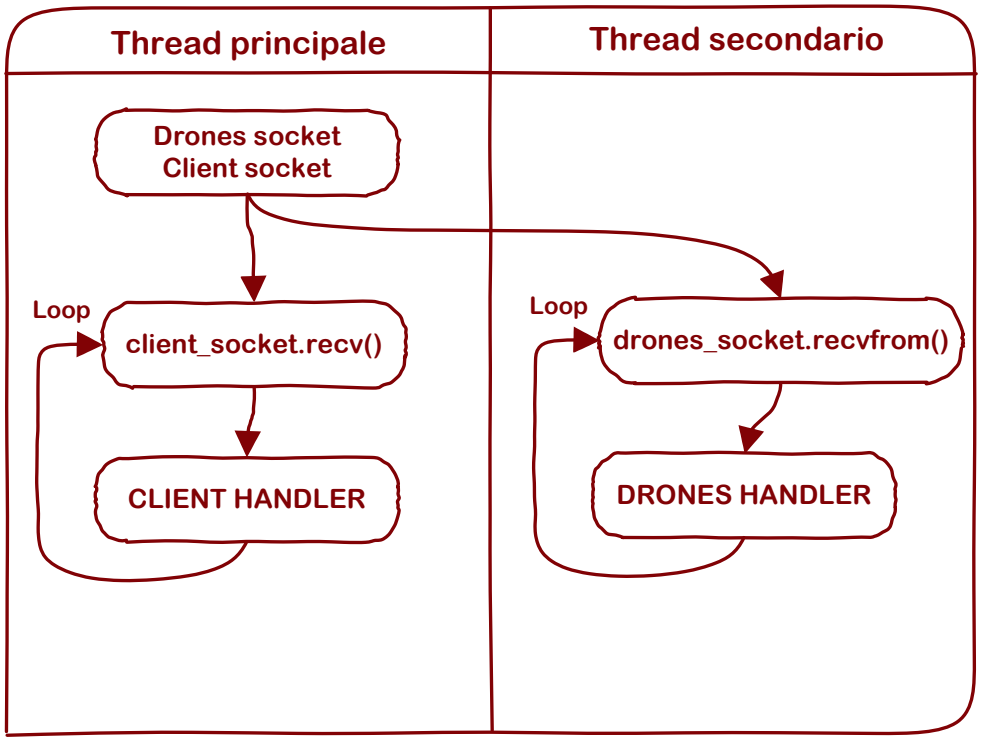
1. Visualizzazione dei droni disponibili - In questo caso è necessario attendere la risposta per poter "renderizzare" la tabella dei droni con le relative informazioni e disponibilità.
2. Recupero dei droni disponibili per effettuare una consegna - In questo caso è necessario attendere la risposta per poter offrire all'utente la scelta tra i droni disponibili (ovvero, mostrare quali droni possono essere utilizzati e quali no)

Di seguito una schematizzazione dell'organizzazione dei thread del client:



6.2 | Gateway

Il gateway ha invece un thread principale che rimane in ascolto sul socket relativo alla connessione TCP, il quale resta in attesa di messaggi dal client che dovranno poi ad esempio essere inoltrati ai droni oppure necessiteranno di una risposta diretta. Un thread figlio invece è utilizzato per rimanere contemporaneamente in ascolto sul socket relativo alla connessione UDP, in modo da poter ricevere e in seguito gestire i messaggi provenienti dai droni.



6.3 | Drone

Per i droni non c'è stata la necessità di utilizzare altri thread oltre a quello principale che esegue le operazioni già descritte nella sezione di Implementazione.

7 | Gestione delle connessioni

7.1 | Buffer

Per la maggior parte dei messaggi vengono utilizzati dei buffer di 1024 byte sia per le connessioni UDP, sia per quelle TCP. Solo il client utilizza un buffer più capiente (di ben 15 KB = 15360 byte) per riuscire a ricevere completamente anche la risposta della richiesta dei droni disponibili: ciò perché, analizzando la dimensione del messaggio nel caso limite in cui si ha il numero massimo di droni collegati alla rete, 1024 byte risultavano in un buffer troppo poco capiente. Si è quindi preferito aumentare il buffer a 15 KB al fine di evitare problemi di troncamento e di caricamento del messaggio.

7.2 | Tempi

Utilizzando l'interfaccia di loopback del PC il calcolo dei tempi non rende evidenti le differenze tra i due protocolli, dato che la connessione locale azzera la differenza tra i tempi, essendo praticamente identici. Quindi i tempi non vengono mostrati in quanto non sarebbero un dato significativo da analizzare. In un sistema reale, in cui i dispositivi sono distanti fra loro, si possono calcolare i tempi, mostrando così le differenze tra i due protocolli.