Alessandro Sciarrillo, Niccolò Mussoni

Università di Bologna
Corso Programmazione di Reti

alessandr.sciarrill2@studio.unibo.it 0000970435
niccolo.mussoni@studio.unibo.it 0000970381

Progetto Droni

Giugno 2022

Indice

Scelte generali di progetto effettuate

Le principali scelte di progetto effettuate riguardano a che livello di astrazione avremmo dovuto gestire la simulazione delle comunicazioni e di conseguenza la gestione dei ruoli Client-Server sui diversi tipi di connessione tra le tre parti (Client, Gateway, droni).

Per la comunicazione tra Client e Gateway ci è sembrato logico assegnare i ruoli di Client-TCP al Client dell'applicazione e di Server-TCP al Gateway poichè quest'ultimo è colui che principalmente rimane in ascolto e gestisce le richieste del client.

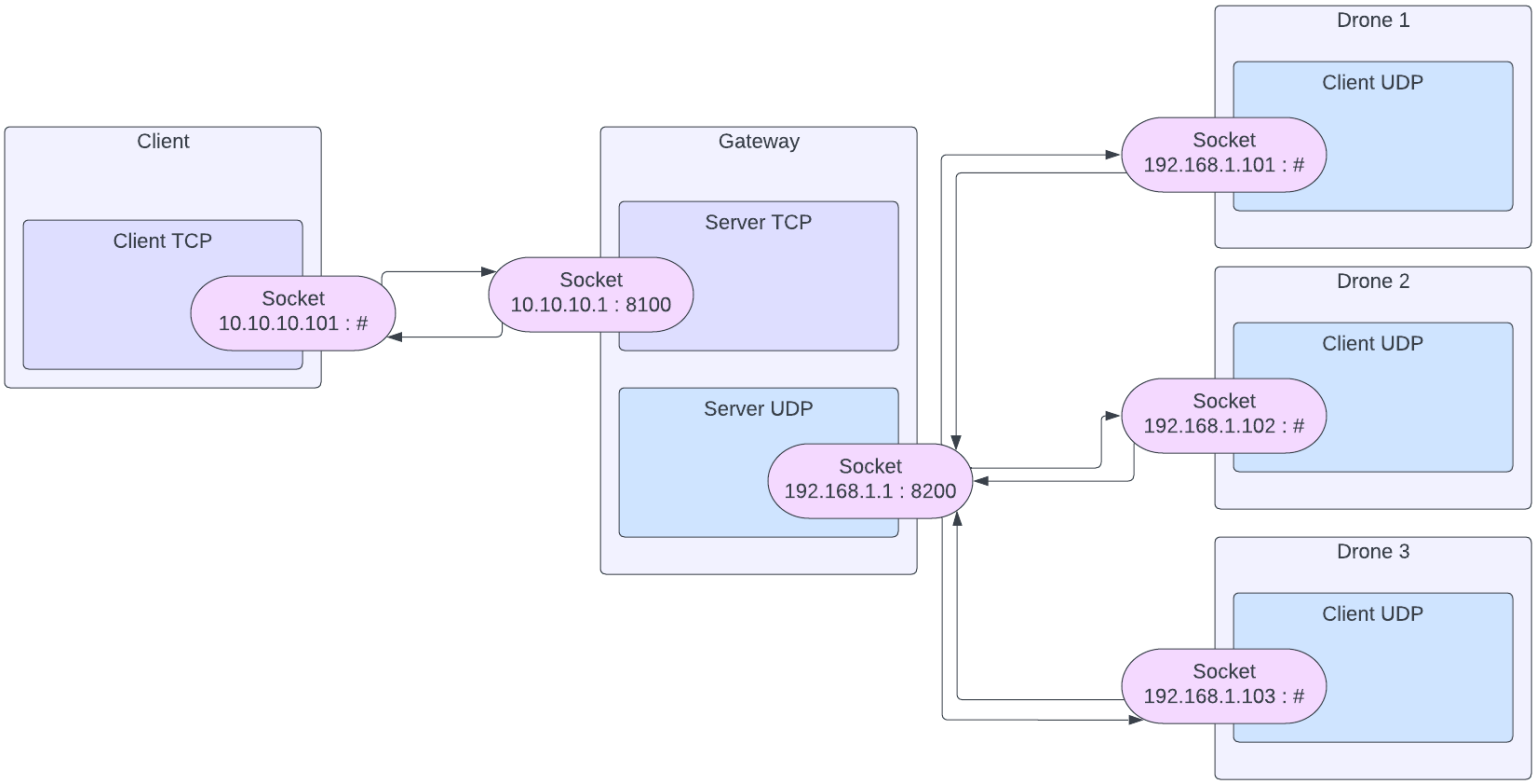
Per la gestione della comunicazione UDP tra Gateway e droni abbiamo optato per utilizzare il Gateway ancora con il ruolo di Server e i droni con il ruolo di client. La decisione è stata tratta dalla modalità con cui la logica del sistema doveva funzionare, cioè che i droni dovessero comunicare al Gateway quando erano disponibili, che quindi ci ha portati a definire in questo modo i ruoli.

Le valutazioni precedenti sono state effettuate considerando sempre che il risultato finale dovesse essere una simulazione e che quindi stessimo lavorando con un certo livello di astrazione dei particolari delle connessioni.

Assegnazione indirizzi IPV4

Schede di rete	**IPV4**	**Netmask**	**Subnet**
Client	10.10.10.101	255.255.255.0	Subnet 1
Gateway eth0	10.10.10.1	255.255.255.0	
Gateway eth1	192.168.1.1	255.255.255.0	Subnet 2
Drone 1	192.168.1.101	255.255.255.0	
Drone 2	192.168.1.102	255.255.255.0	
Drone 3	192.168.1.103	255.255.255.0	

Schema Client-Server con Sockets

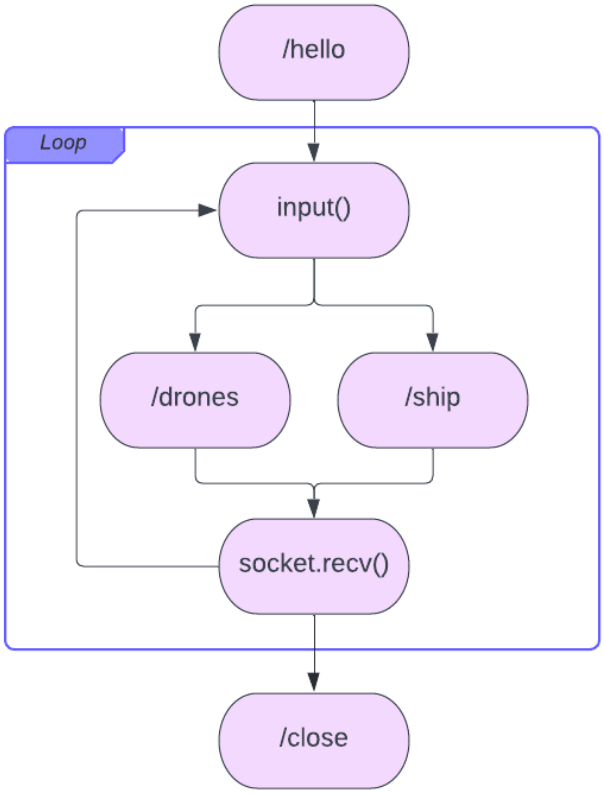


Implementazioni

Client

Il client, appena viene avviato, si connette al gateway tramite una connessione TCP. Utilizziamo il messaggio /hello <ipv4> per simulare lo stabilimento di connessione in cui il client fa conoscere il proprio indirizzo ip.

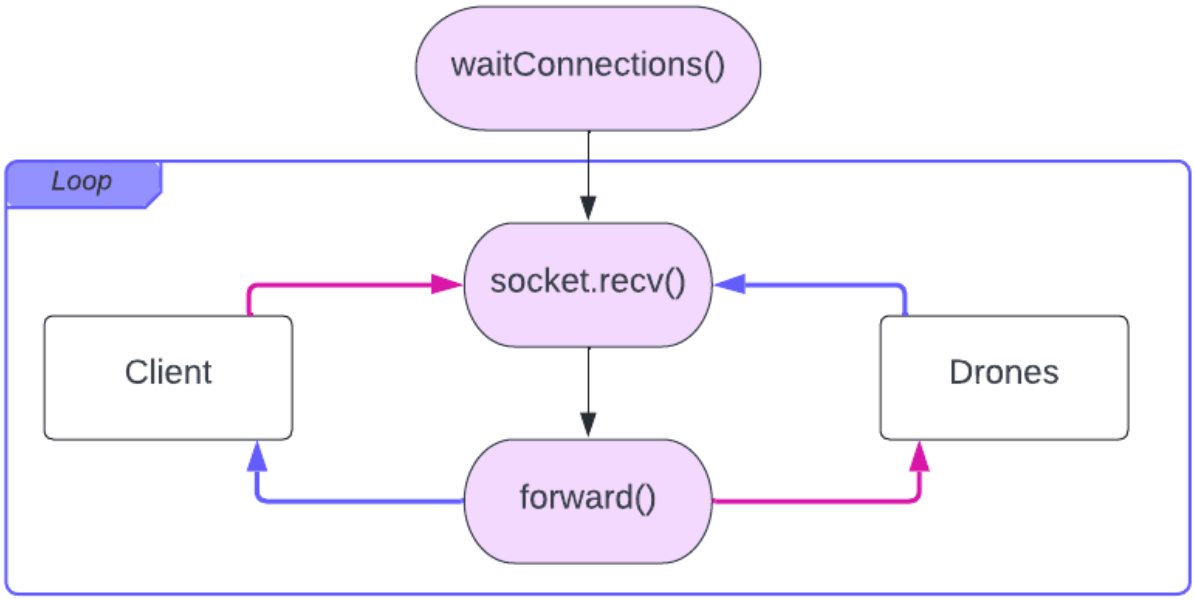
In seguito alla parte iniziale inizia un loop in cui il client rimane in attesa di comandi in input da console. In base al comando ricevuto viene inviato un messaggio al Gateway e viene messo un thread in ascolto della risposta che verrà poi stampata a video al suo arrivo. Il ciclo si ripete con l'attesa di un nuovo comando e terminerà solo alla disconnessione del client /close.



Gateway

Al suo avvio il gateway crea i due socket per le sue due interfacce di rete e le relative connessioni, quella TCP e quella UDP. In seguito lancia un thread che si mette in ascolto sul socket relativo alla connessione UDP per la ricezione dei messaggi che i droni invieranno al socket quando si connetteranno e saranno disponibili. In parallelo si attende anche la connessione del client.

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, e viceversa, messaggi che dai droni devono arrivare al client.

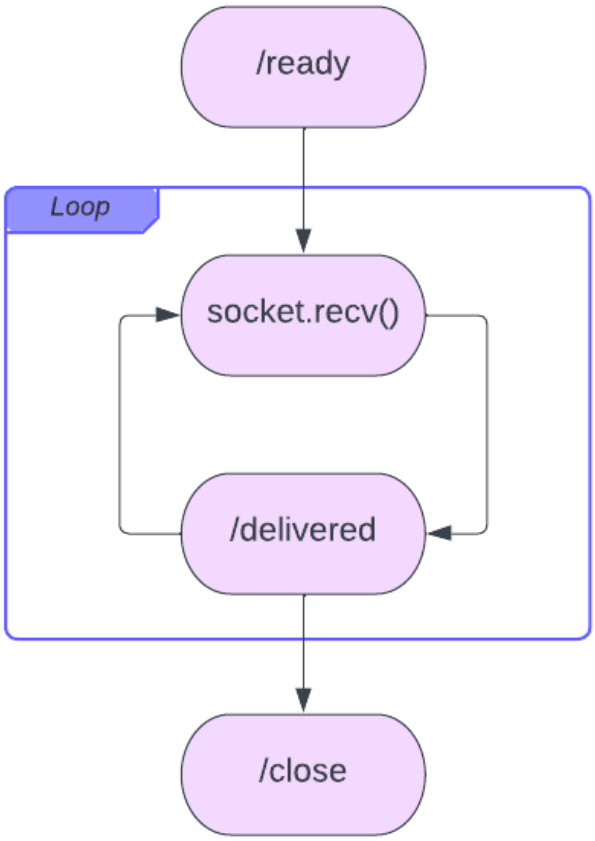


Drone

I droni per prima cosa si connettono al gateway e si identificano con il loro ID e il loro indirizzo IPV4 tramite un messaggio /ready <drone_id> <ipv4>.

Appena connessi al gateway avviano un loop in cui attendono spedizioni da effettuare stampandone l'indirizzo, in seguito comunicano l'avvenuta consegna attraverso il messaggio /delivered rendendosi nuovamente disponibili.

Il ciclo si interrompe quando il drone si disconnetterà con il messaggio /close.



Strutture dati utilizzate

1. Messaggi testuali

Ci siamo ispirati ai messaggi HTTP come metodologia di struttura dati con cui comunicare. Abbiamo strutturato i messaggi con una parte iniziale composta da "/comando" e di seguito i parametri del comando separati da uno spazio. Questa struttura ci ha permesso di rendere i comandi modulari come nel caso del comando /ship a disposizione del Client che come secondo parametro può accettare sia l'ID di un drone sia il suo indirizzo IP.

Soggetto	Header <parametro>	Descrizione
Client	/drones	Richiede la lista dei droni
	/ship <drone_id / drone_ip> <delivery_address>	Invia a un drone la richiesta di spedizione ad un indirizzo
	/hello <ipv4>	Nella simulazione di connessione al Gateway gli comunica il suo indirizzo IPV4
	/close	Segnala al Gateway che il client si è disconnesso
Drone	/ready <drone_id> <ipv4>	Nella simulazione segnala al Gateway che il drone si è connesso ed è pronto
	/delivered <drone_id>	Segnala che il drone ha concluso una spedizione
	/close <drone_id>	Segnala al Gateway che il drone si è disconnesso e di conseguenza non sarà più disponibile

2. Dizionari

Abbiamo utilizzato due dizionari all'interno del codice del Gateway per mantenere registrate le connessioni sia fisiche che logiche: il primo dizionario connections ha come chiavi gli ID dei droni e come valori una tupla composta da IP e porta con cui le comunicazioni effettive funzionano. Nel secondo dizionario logic_connections abbiamo invece salvato le associazioni logiche ID drone → IPV4 relative alla simulazione del gateway. In un utilizzo su una rete "reale" con schede di rete distinte ovviamente non ci sarebbe stato bisogno di quest'ultimo poichè le informazioni sarebbero già state mantenute da connections; in questo caso però, visto l'utilizzo del Loopback delle nostre macchine, gli IP in connections risultano tutti equivalenti a 127.0.0.1 e quindi era necessario logic_connections per il mantenimento delle informazioni per la simulazione degli indirizzi IPV4.

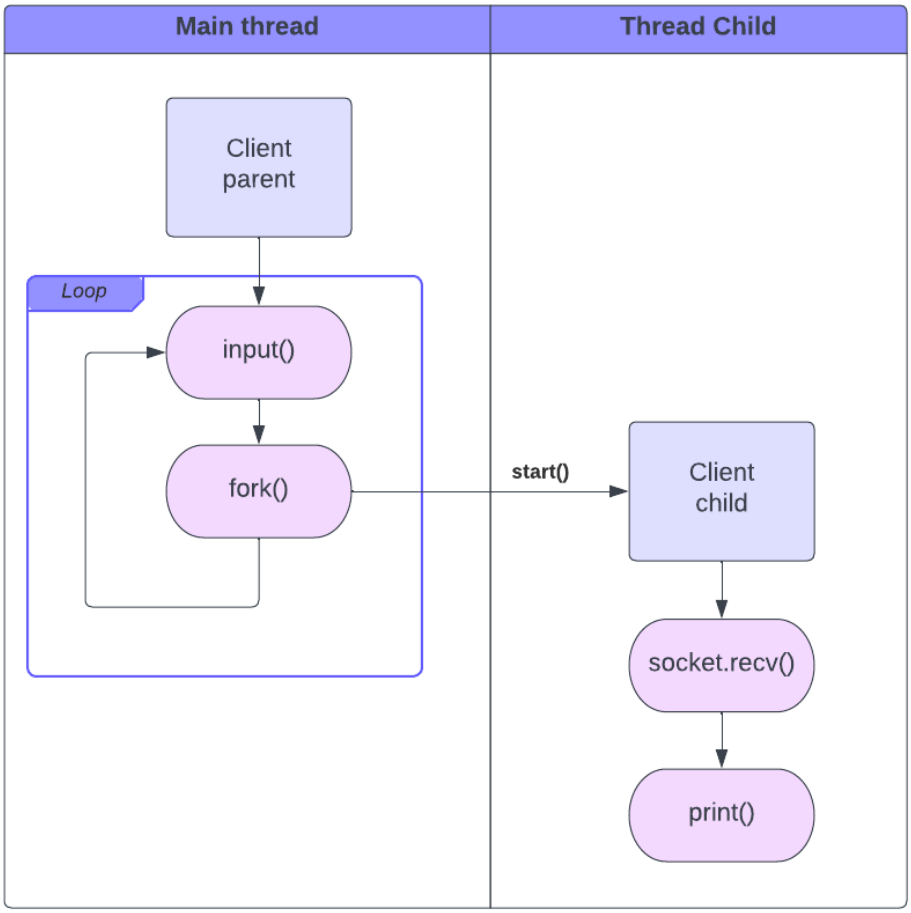
3. Liste

Abbiamo valutato di mantenere in una lista available_drones i droni disponibili all'effettuazione di una spedizione. Questa lista è mantenuta all'interno del Gateway per rendere più efficiente l'aggiornamento e la lettura da parte dei dispositivi connessi che puntualmente devono farla modificare (droni) o richiederla (Client). Ci è sembrato ragionevolmente utile implementare questa parte di logica direttamente nel Gateway; in una rete reale si sarebbe potuta implementare anche con la riprogrammazione di un vero gateway e mantenendo al suo interno una struttura dati simile.

Gestione dei Threads

Client

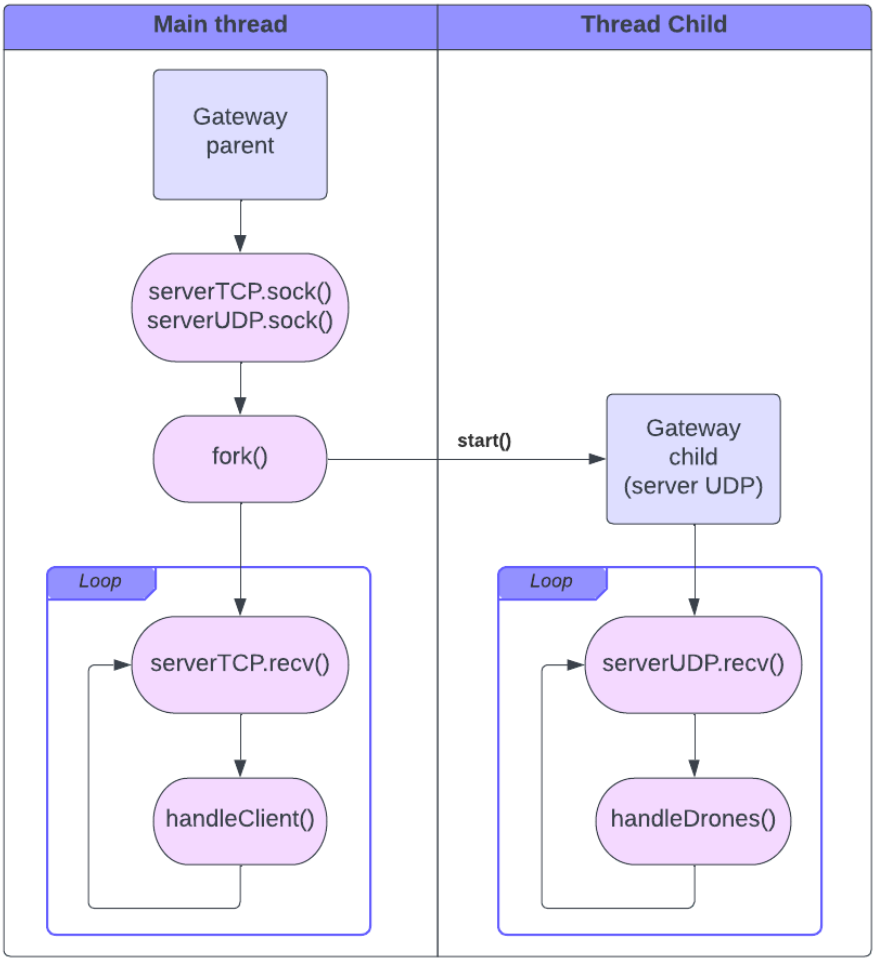
Per il client abbiamo deciso di mantenere il thread principale in attesa di input dall'utente. Appena un comando viene lanciato dalla console la richiesta viene inoltrata al Gateway e il thread principale crea un thread figlio che viene lasciato in attesa della risposta. In questo modo il thread padre può immediatamente rimettersi in attesa di input sulla console per nuovi comandi.



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.



Drone

Per i droni non abbiamo avuto la necessità di utilizzare altri thread oltre a quello principale che esegue quindi il flusso di operazioni già descritto nella sezione di Implementazione.

Gestione delle connessioni simulate

Buffer

Data la dimensione limitata dei messaggi che abbiamo necessità di inviare per la nostra simulazione abbiamo fatto un'analisi dimensionale sulla dimensione dei pacchetti e abbiamo valutato che dei buffer di 1024 bytes sarebbero stati proporzionati e sufficienti per entrambe le connessioni UDP e TCP.

Tempi

L'effettivo calcolo dei tempi relativi alle due connessioni non avrebbe reso evidenti le differenze tra i due protocolli in termini di costo di tempo per la comunicazione, in quanto utilizzando l'interfaccia di Loopback le misurazioni tra i due protocolli con la precisione temporale a disposizione sarebbero state pressochè identiche. Abbiamo quindi valutato di non mostrare i tempi in quanto sarebbero stati insignificanti e se li avessimo dovuti introdurre artificialmente sarebbe poi stato inutile calcolarli. Un calcolo dei tempi potrebbe essere effettuato con risultati che rispecchino le differenze tra i due protocolli se venissero attuati in un sistema reale con distanze più significative.

Guida Utente

È **raccomandato** l'utilizzo di Spyder per un'esecuzione ottimale.

Come prima cosa è necessario avviare il Gateway

Se non si usa Spyder, da terminale: $python3\ \ gateway.py$

In seguito si possono avviare in qualsiasi ordine:

- Client con:

  Se non si usa Spyder, da terminale: $python3\ \ client.py$

- DroneX (sostituire X con 1, 2, 3) con:

  Se non si usa Spyder, da terminale: $python3\ \ droneX.py$

I comandi utili per testare il tutto sono quelli lanciabili dalla console del Client e sono fruibili con il comando /help direttamente da essa.