TRACCIA 2: Architettura client-server UDP per trasferimento file

L'obiettivo del progetto consiste nel creare un applicazione in Python per il trasferimento file tra server e client tramite il protocollo **UDP**.

Il software prodotto garantisce le seguenti funzionalità:

- Connessione senza autenticazione
- Visualizzazione dei file disponibili per il trasferimento
- Download di un file selezionato sul client
- Upload di un file selezionato sul server

Client e Server comunicano scambiandosi 2 tipi di messaggi:

- messaggi di comando, i quali il client usa per chiedere l'esecuzione di un comando
- messaggi di risposta, i quali il server usa per comunicare l'esito del comando ricevuto

Funzionalità del server:

- Invio di messaggi di risposta contenenti l'esito del comando ricevuto dal client
- Un messaggio di risposta contenente la lista dei file disponibili al trasferimento
- Un messaggio di risposta al comando "get" se è possibile il trasferimento od un messaggio di errore ove il download non fosse possibile
- Un messaggio di risposta al comando "put" se è possibile il trasferimento ed un messaggio di risposta con l'esito dell'operazione

Funzionalità del client:

- Invio del comando "list" per richiedere la lista di file disponibili al trasferimento
- Invio del comando "get" per scaricare un file selezionato e la gestione di un eventuale messaggio di errore
- Invio del comando "put" per caricare un file selezionato e la gestione di un eventuale messaggio di errore

Analisi e modello del dominio

Uno dei primi obiettivi è definire i diversi messaggi che client e server scambiano relativamente alla situazione.
Identifichiamo 3 diversi tipi di messaggio:

- POSITIVE ACK
- NEGATIVE ACK
- FINISHED TRANSMISSION ACK

```
FINISHED_TRANSMISSION_ACKNOWLEDGEMENT = 400
NEGATIVE_ACKNOWLEDGEMENT = 300
POSITIVE_ACKNOWLEDGEMENT = 200
```

POSITIVE ACK indica un'operazione eseguita con successo.
NEGATIVE ACK indica un'operazione non eseguita con successo.
FINISHED TRANSMISSION ACK indica l'interruzione del trasferimento file dal mittente

Per quanto riguarda i messaggi di comando ne distinguiamo 3:

- “LIST” che richiede l’elenco dei file accessibili
- “GET” che richiede un file selezionato
- “PUT” che carica un file selezionato

Bisogna inoltre identificare diversi messaggi di risposta ai possibili comandi:

- “LIST” riceverà sempre **POSITIVE ACK** a patto che il server possa accedere ai file
- “GET” riceverà **POSITIVE ACK** nel caso sia possibile scaricare il file altrimenti riceverà **NEGATIVE ACK** nel caso il file selezionato non sia presente nell’archivio
- “PUT” avviserà il client se il file selezionato non è presente nel proprio archivio altrimenti riceverà un **POSITIVE ACK**

Sviluppo

Per il corretto funzionamento di tutte le funzionalità specificate ho individuato alcuni sotto-programmi per migliorare la qualità del codice:

1. Utilizzo del modulo “**os**”

```
def get_files():
    files = []
    for file in os.scandir(path):
        files.append(file.name)
    return files
```

2. Utilizzo della lettura/scrittura di file, necessari per il trasferimento

```
f_in = open(path+file_name,'rb')
```
```
file = open(path+title,'wb')
```

3. Creazione di una classe **packet** che incapsula i messaggi di comando e risposta

```
class packet:
    def __init__(self,command,subject,ack,data):
        self.command = command
        self.subject = subject
        self.ack = ack
        self.data : bytes = data
```

4. Funzioni di controllo dell’integrità dei file inviati
5. Funzioni di controllo dell’integrità dei dati ricevuti
6. Funzioni di logging delle richieste ricevute
7. Funzioni di performance dei trasferimenti

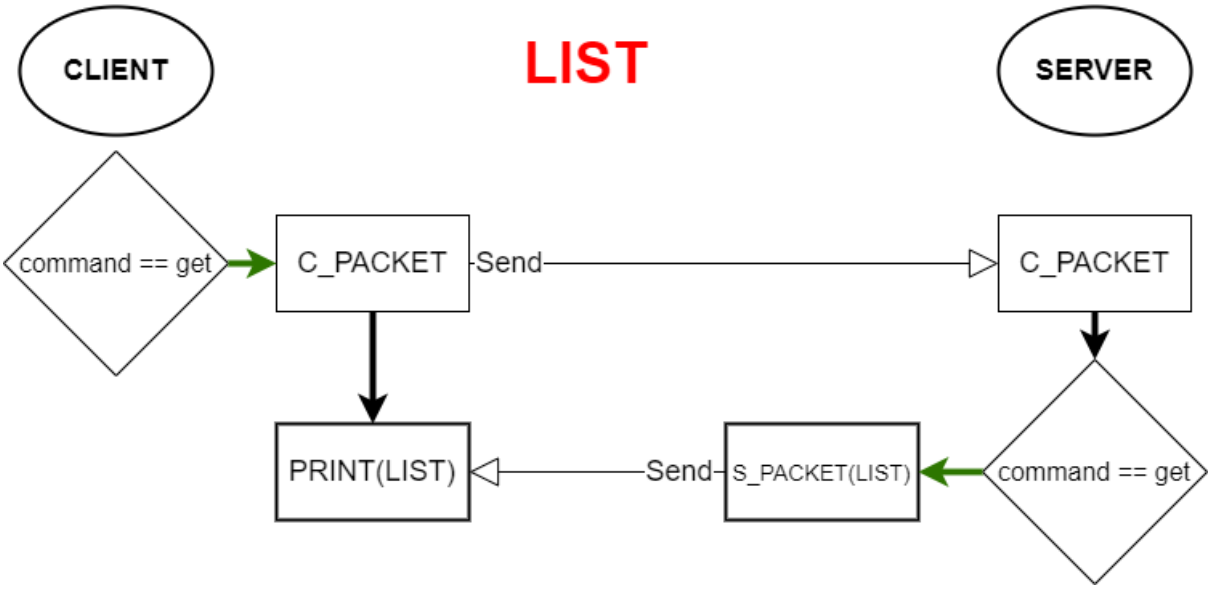
1) Ho ampiamente usato il modulo **os** specialmente il sottomodulo **path** per:
    a) Accesso a cartelle
    b) Esistenza di file
    c) Dimensione di file
    d) Ottenere percorsi
2) Ho scelto di leggere/scrivere i file mediante la “binary mode” offerta da Python che permette la lettura/scrittura di blocchi di byte
3) Ho creato una classe **packet** che permette la comunicazione tra client e server **packet** comprende le seguenti funzionalità:
    a) un metodo statico “from message” che analizza l’input del client e lo trasforma in un oggetto **packet**
    b) un custom **encoder** che ne permette la trasformazione per essere inviato

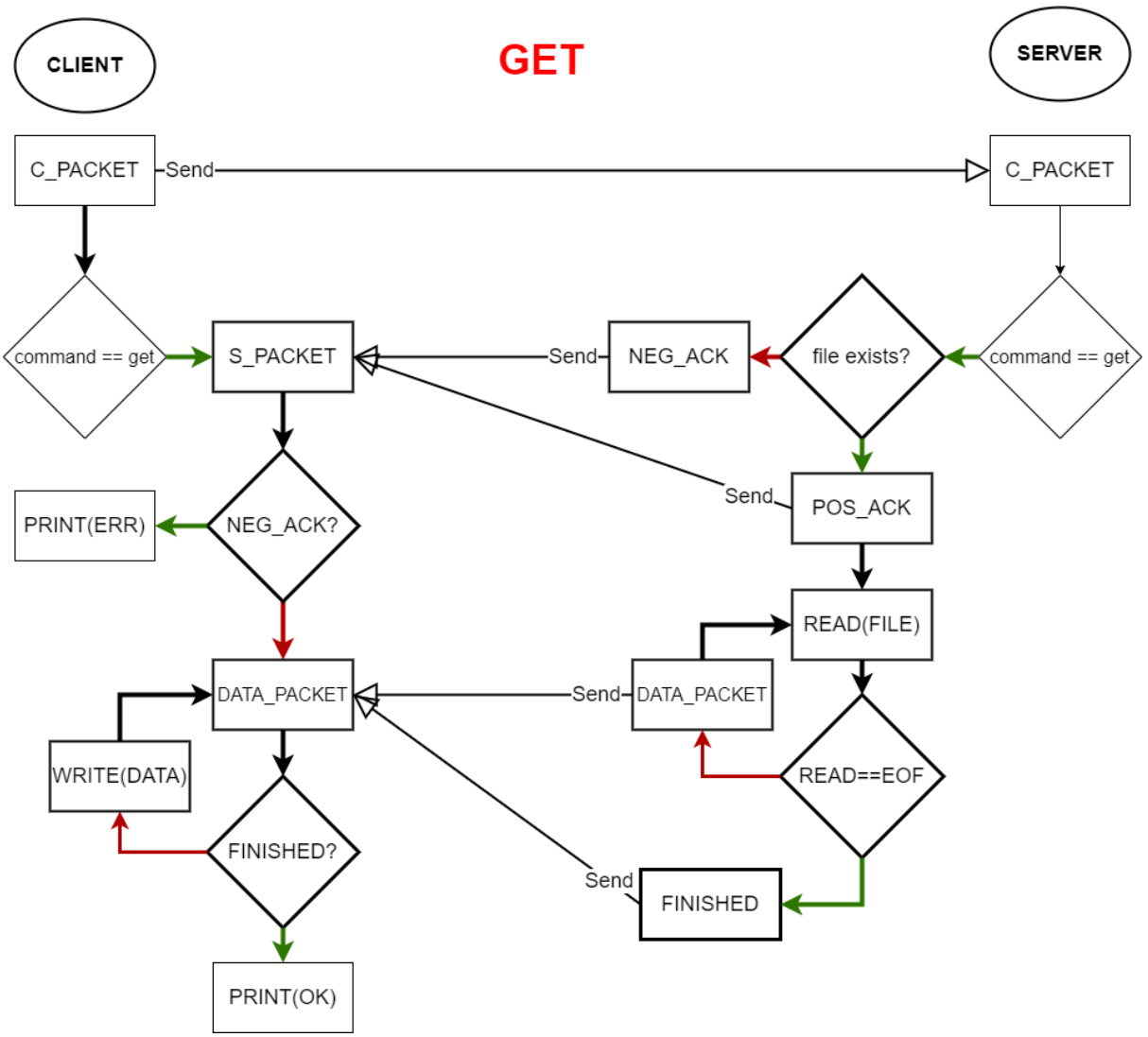
c) un custom **decoder** che permette la conversione in packet di dati ricevuti

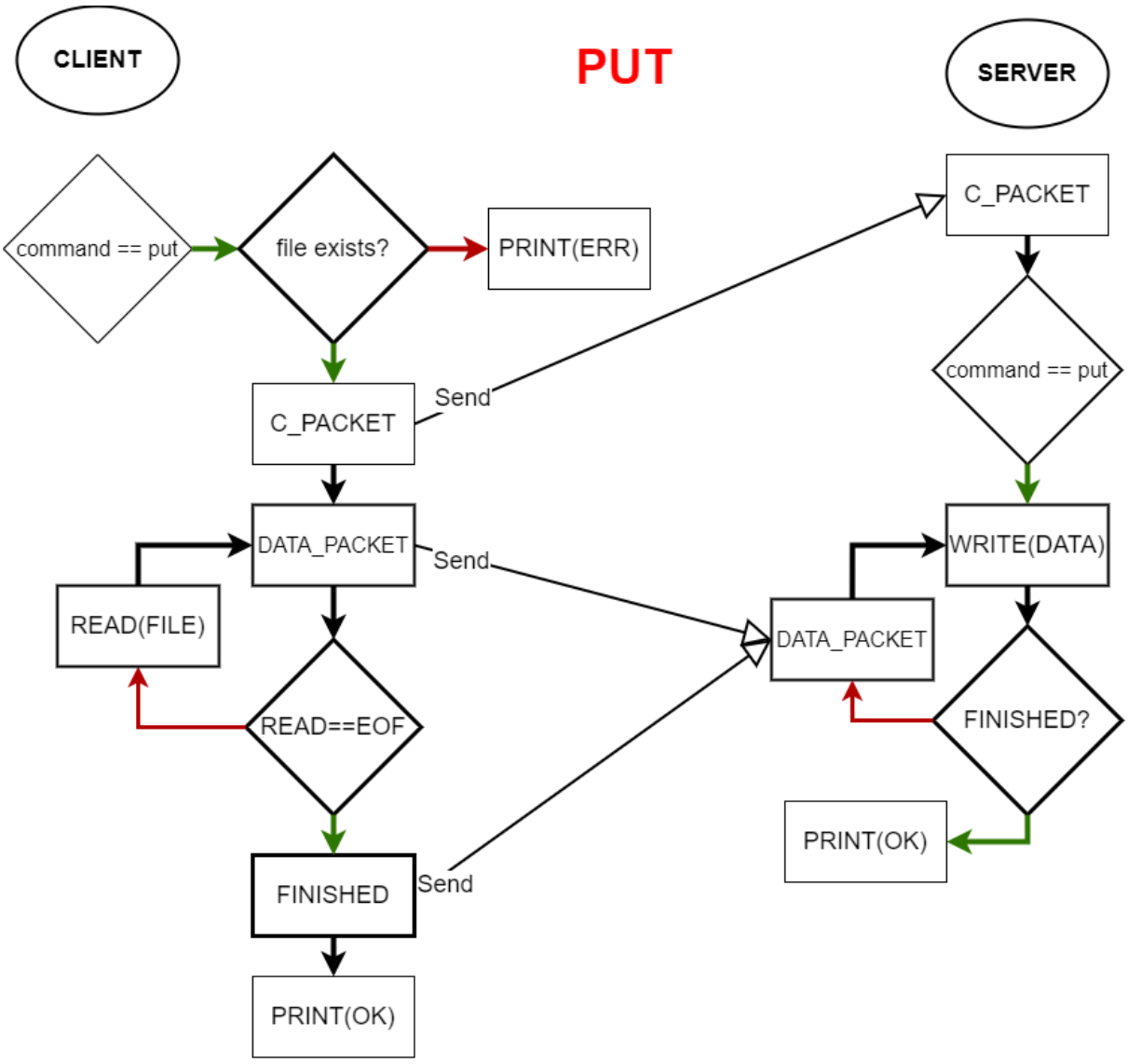
4) Dopo un download/upload mittente e destinatario scambiano le dimensioni del file selezionato e controllano la coerenza della dimensione
5) Ogni messaggio di risposta viene controllato se ricevuto correttamente
6) Ogni richiesta che riceve il server viene loggata in un apposito file di log privato
7) Mediante il modulo **time** vengono cronometrati gli upload ed i download per stabilire il rate di trasmissione

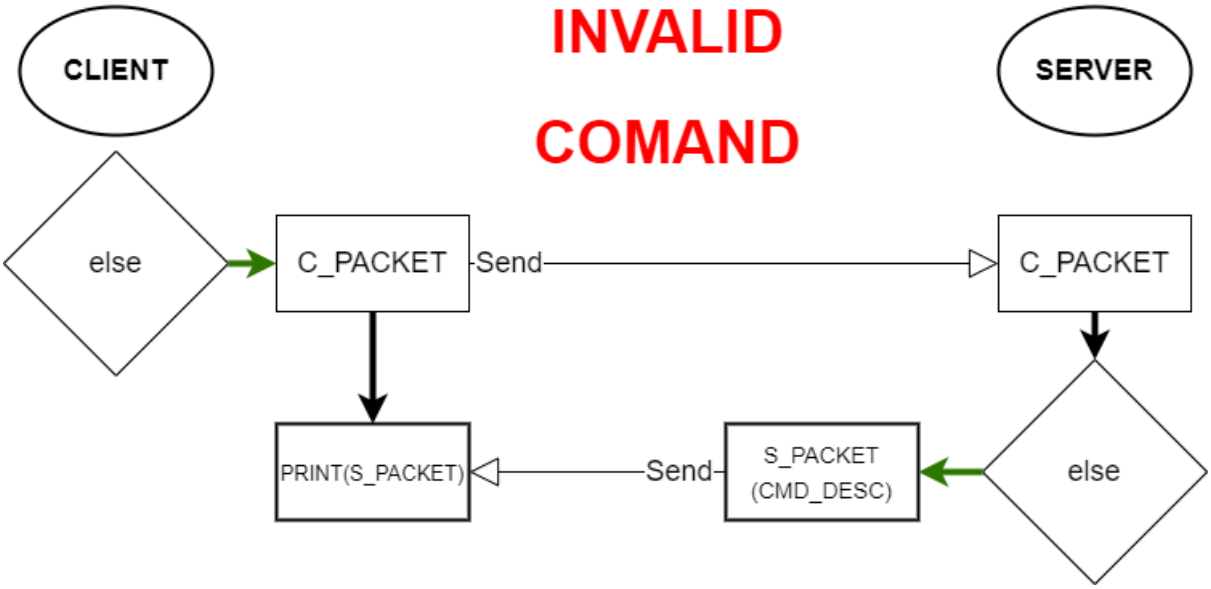
Modelli

Qui vi sono illustrati diagrammi riepilogativi del codice e del suo funzionamento:




GET
CLIENT
SERVER
C_PACKET
Send
C_PACKET
command == get
S_PACKET
Send
NEG_ACK
file exists?
command == get
Send
POS_ACK
PRINT(ERR)
NEG_ACK?
READ(FILE)
DATA_PACKET
Send
DATA_PACKET
WRITE(DATA)
READ==EOF
FINISHED?
Send
FINISHED
PRINT(OK)


CLIENT
PUT
SERVER
command == put
file exists?
PRINT(ERR)
C_PACKET
Send
C_PACKET
command == put
DATA_PACKET
Send
WRITE(DATA)
READ(FILE)
DATA_PACKET
READ==EOF
FINISHED?
PRINT(OK)
FINISHED
Send
PRINT(OK)


CLIENT
INVALID
COMAND
SERVER
else
C_PACKET
Send
C_PACKET
PRINT(S_PACKET)
Send
S_PACKET
(CMD_DESC)
else

Interfaccia

Il software comprende due script: **client.py** e **server.py** entrambi si appoggiano su un modulo **packet.py**.

I due script vanno avviati uno di seguito all'altro: prima **server.py** poi **client.py**.

All'avvio di client.py l'interfaccia presenterà una finestra di input dove inserire il messaggio di comando diretto al server.

```
[2022-06-27 13:49:31]
[please input a command]
```

Il software prevede la conoscenza dei principali comandi anche se all'invio di un comando non valido, il server provvederà ad informare il client dei comandi disponibili.

```
[2022-06-27 13:53:12]
[please input a command] help
1 or list ... lists all avaible files
2 or get ["name"] ... downloads the selected file if it exists
3 or put ["name"] ... uploads the selected file if it exists

[2022-06-27 13:53:14]
[please input a command] test
1 or list ... lists all avaible files
2 or get ["name"] ... downloads the selected file if it exists
3 or put ["name"] ... uploads the selected file if it exists
```

In seguito è riportata una lista dei comandi e le loro interazioni:

"1" o "list" fornisce i file contenuti nell'archivio del server incorporando un indice per file e la dimensione

```
[2022-06-27 14:16:08]
[please input a command] list
[1] immagine.png   64.87KB
[2] testo.txt   61B
[3] YEEEEE.mp3   44.06KB
```

"2" o "get" seguito dall'indice del file o dal nome del file stesso permette il download del file se presente

```
[2022-06-27 14:16:39]
[please input a command] get immagine.png
Successfully received:  immagine.png
Size: 64.87KB Time elapsed: 0.01s Rate: 7.92MB/s
```

```
[2022-06-27 15:39:52]
[please input a command] get 1
file already exists, substituting with current
Successfully received:  immagine.png
Size: 64.87KB Time elapsed: 0.01s Rate: 9.06MB/s
```

"3" o "put" seguito dal nome del file ne permette l'upload se presente nell'archivio del client

```
[2022-06-27 15:42:56]
[please input a command] put c_file.exe
file successfully sent
Successfully sent:  c_file.exe
Size: 52.76KB Time elapsed: 0.01s Rate: 7.37MB/s
```

Testing

In seguito si elencano situazioni specifiche e la gestione di errori:

- **Input nullo:**
  - nel caso l'input sia vuoto, l'input verrà considerato come un comando invalido

```
[2022-06-27 15:45:11]
[please input a command]
1 or list ... lists all avaible files
2 or get ["name"] ... downloads the selected file if it exists
3 or put ["name"] ... uploads the selected file if it exists
```

- **put/get di un file inesistente:**
  - nel caso il file non sia presente il server ed il client visualizzano messaggi di errore

```
[2022-06-27 15:45:12]
[please input a command] get file_inesistente
File not found
```

```
[2022-06-27 15:48:56]
[please input a command] put file_inesistente
File not found
```

- **put/get di un file già presente nell'archivio del destinatario:**
  - nel caso il file sia già presente esso verrà cancellato e riscritto.

```
[2022-06-27 16:44:33]
[please input a command] get 1
file already exists, substituting with current
Successfully received:  c_file.exe
Size: 52.76KB Time elapsed: 0.01s Rate: 6.44MB/s
```

- **put/get di un file di grosse dimensioni:**
  - durante il download/upload di file di grosse dimensioni vi è la possibilità che il file venga danneggiato

```
[2022-06-27 16:49:07]
[please input a command] get 3
file already exists, substituting with current
expected file size:  85.85MB
current file size:  84.74MB
File was not received correctly, file size mismatch
```