Relazione Progetto di Programmazione di Reti: Traccia nr.2

Pietro Tellarini
pietro.tellarini2@studio.unibo.it
matricola: 0000971367

Indice

Capitolo 1

Analisi del problema

1.1 Descrizione

L'obiettivo del progetto è quello di implementare, tramite linguaggio Python, un'applicazione per il trasferimento di file che impieghi il servizio di rete senza connessione(UDP come protocollo di strato di trasporto).
Il software deve permettere:

- Connessione client-server senza autenticazione;
- La visualizzazione sul client dei file disponibili sul server;
- Il download di un file dal server;
- L'upload di un file sul server;

Il server dovrà rimanere in ascolto rispondendo ai tre messaggi che può ricevere dal client, ovvero, la risposta al comando list mandando al richiedente la lista dei file scaricabili dal server; la risposta al comando get contente il file che l'utente vuole scaricare o eventuali messaggi di errore o buona riuscita di tale operazione. Inoltre il server deve poter ricevere file dal client in risposta al comando put.

Il client deve ricevere in input dall'utente quale delle tre operazione tra List, Get e Put egli voglia eseguire e comunicarla al server. Il server deve rispondere con messaggio ultimo l'esito delle operazioni.

Capitolo 2

Design

Il software è basato su un'architettura di tipo client-server.
A livello di trasporto si fa l'uso del protocollo UDP, quindi non vi è la garanzia della ricezione del messaggio e nemmeno dell'integrità di quest'ultimo.
Per ovviare a problematiche che possono nascere con la suddivisione di messaggi troppo grandi in vari pacchetti, come può essere la perdita di uno di questi, il mittente invia pacchetto finale con un header speciale e con datagram il checksum cosicché il ricevente possa verificare la corretta ricezione dell'intero messaggio.

2.1 Protocollo

Ad alto livello, client e server si scambiano dei messaggi tra di loro e, in base alla dimensioni dei messaggi, questi sono convertiti in uno o più pacchetti.
La struttura dati del pacchetto è una lista di due elementi: "HEADER" e "DATA".

Estitono 6 tipi di header:

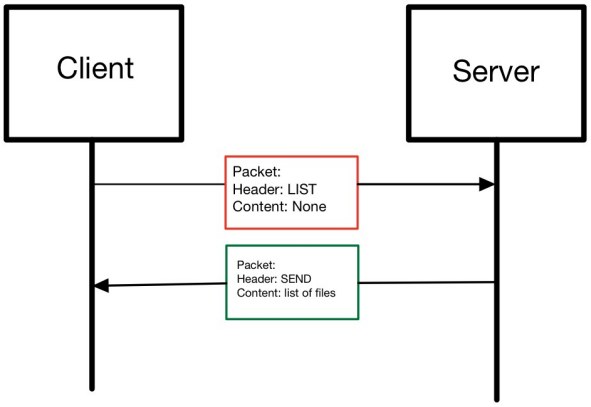
- LIST
- GET
- PUT
- SEND
- END
- ERR

I primi tre header vengono usati per inviare i messaggi con le richieste dei vari comandi, mentre gli altri tre sono utilizzati durante la trasmissione per effettuare controlli o segnalare lo stato del trasferimento dei file.

2.2 Comandi

2.2.1 List

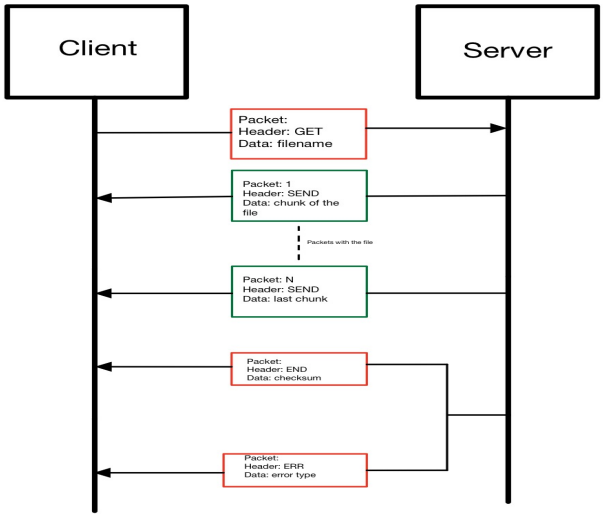
Il client manda un messaggio il cui header è LIST, così ha inizio questa operazione.



Come si può vedere in figura, il server riceve il pacchetto iniziale dove nell'header è specificato il tipo di operazione LIST, a questo punto questi risponderà mandando la lista dei file scaricabili disponibile.

2.2.2 Get

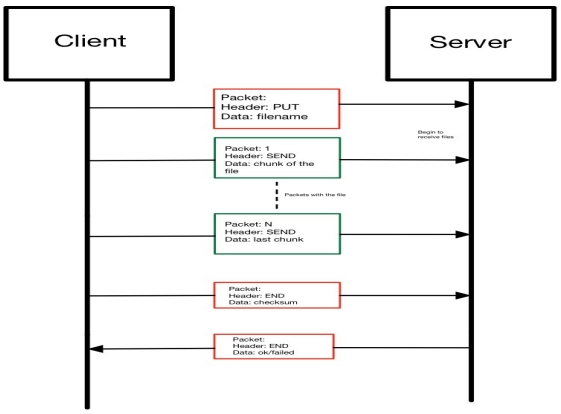
Il client manda un messaggio specificando nell'header l'operazione GET e con datagram il nome del file che l'utente vuole scaricare



Il server riceve il messaggio con l'operazione Get, legge il file, che se dovesse essere più pesante del buffer del client viene diviso in vari pacchetti con header SEND e come contenuto il file che si vuole spedire. L'ultimo pacchetto spedito dal server è sempre con header END in caso di corretta trasmissione e contenuto il checksum che verrà verificato dal client.
L'esecuzione effettiva di invio e ricezione dei file avviene tramite due metodi della classe Message.
Invece, se vi dovessero essere errori di ogni genere il server manda un pacchetto con header: ERR per segnalare l'errore e con contenuto la descrizione dell'errore, come potrebbe essere l'assenza del file nel server.

2.2.3 Put

Il client manda un pacchetto con header di tipo PUT e con data il nome del file, successivamente inizia l'invio del file. Quando il server riceve il primo pacchetto capisce che sta per essere eseguito il comando PUT, quindi si prepara per ricevere il file. Vengono utilizzati gli stessi metodi che usa il comando GET, solo che mittente e ricevente in questo caso sono invertiti.



Una volta ricevuto il file, il server verificherà l'ugualianza del checksum e questi invierà al client un pacchetto con header END e con data l'esito del controllo.

2.3 Headers

2.3.1 SEND

Viene usato durante la trasmissione di messaggi contenenti come data i pezzi di file che si vogliono trasferire.

2.3.2 END

Viene inviato sempre alla fine del trasferimento di un file per comunicarne appunto la condivisione, contiene come data il checksum del file per permettere al ricevente di effettuare un controllo di integrità del file. Viene usato anche dal server per comunicare al client l'esito di del comando put.

2.3.3 ERR

Viene usato per comunicare eventuali errori al client, contiene come data informazioni riguardo l'errore.

2.4 Codifica Pacchetti

Per poter comunicare tra di loro server e client si scambiano uno o più pacchetti contenente il messaggio che vogliono inoltrare. Un pacchetto corrisponde alla struttura dati lista di dimensione 2, con [0] = HEADER e [1] = DATA, il tutto viene trasformato in un JSON che prima di essere inoltrato viene codificato in utf-8.

Capitolo 3

Sviluppo

Il progetto si compone di tre file:

- **Server.py:** contiene il codice della classe Server per gestire le richieste al server
- **Client.py:** contiene il codice della classe Client per gestire le richieste al client
- **Common_Classes.py:** contiene due classi per poter semplificare la comunicazione e le stutture dati utilizzate sia dal server e dal client

3.1 Sever

Per la creazione del server si fa uso della libreria socket. Il server è caratterizzato da un processo che lo mantiene costantemente in ascolto, infatti, esso può essere terminato solamente con la combinazione di tasti Ctrl+C.
Quando il server riceve un messaggio dal client, lo riceve sottoforma di un pacchetto, quindi ne analizza l'header a cui corrisponde un'operazione da eseguire, infine, risponde con il contenuto richiesto da questa operazione, per poi ritornare in ascolto.
Ricezione e invio dei file avvengono tramite i metodi della classe Message: send_file() e receive().

3.2 Client

Il client è responsabile della creazione del socket UDP per comunicare con il server, invia le richieste a quest'ultimo iniziando la comunicazione.

Per fare uso dell'applicazione bisogna digitare i vari comandi.
Il client prende in input i comandi e inizia la comunicazione con il server sempre tramite l'uso dei metodi $\mathbf{send}_{f}ile()e\mathbf{receive()}presentiin\mathbf{Common\_Classes}$.

3.3 Common_Classes

3.3.1 Classe Message

In questa classe sono presenti la maggior parte dei metodi utilizzati per la comunicazione client-server e per l'esecuzione delle varie operazione richieste dall'utente. I metodi principali sono:

- **send_file()**
- **receive()**

Il metodo send_file() prende come argomenti il messaggio e l'indirizzo a cui spedire il primo. Il metodo si occupa di dividere il messaggio in vari pacchetti per evitare una congestione del buffer e gestisce, anche, l'invio di eventuale errori che vi possono essere da parte del mittente.

Il metodo receive() permette la ricezione dei vari pacchetti per poi riunirli nel messaggio o file che il mittente voleva trasmettere, si occupa inoltre di verificare l'integrità del messaggio utilizzando il controllo checksum.
Ci sono anche tre metodi:

- **command_list()**
- **command_get()**
- **command_put()**

utilizzati dal client per istanziare le varie operzioni.

3.3.2 Packet

La classe Packet viene utilizzata per la creazione di oggetti packet che vengono iniviati durante la comunicazione.

Capitolo 4

Guida Utente

4.1 Envoironment

Per poter eseguire i file presenti all'interno di questo progetto è necessario utilizzare una versione di Python $\geqslant 3.8$.

Non è necessario installare alcun pacchetto aggiuntivo in quanti vengono usati solo moduli presenti nella libreria standard di Python.

4.2 Server

Per avviare il server è sufficiente lanciarlo da qualsiasi IDE o eseguire il seguente comando:

python3 server.py

4.3 Client

Per avviare il client è sufficiente lanciarlo da qualsiasi IDE o eseguire il seguente comando:

python3 client.py

4.3.1 Comando Help

help

per visualizzare tutti i comandi disponibili

4.3.2 Comando list

list

4.3.3 Comando get

get filename filepath

il filepath di dove si vuole salvare il file, altrimenti solo il nome del file se si vuole salvare il file nella stessa cartella del client

4.3.4 Comando put

put filename filepath

il filepath solo se il file non si trova all'interno della cartella del client

4.4 E.G.

list
get lion.jpeg
get Lion-King-Script.pdf /User/Desktop
put lion2.jpeg