Reti di Calcolatori ed Ingegneria del Web - A.A. 2019/20
Progetto B1: Trasferimento file su UDP

Relazione

Giovanni Pica
0253606

Indice

1 Introduzione

Lo scopo di questo progetto è di implementare in linguaggio C, un'applicazione client-server per il trasferimento di file tramite UDP (socket tipo SOCK_DGRAM). Il software deve permettere:

- Connessione client-server senza autenticazione;
- La visualizzazione sul client dei file disponibili sul server dunque l'equivalente del comando List;
- Il download di un file dal server dunque l'equivalente del comando Get;
- L'upload di un file dal server dunque l'equivalente del comando Put;
- Il trasferimento file in modo affidabile.

Per garantire il trasferimento affidabile dei messaggi spediti o ricevuti sia il client che il server implementano a livello applicativo il protocollo Go-Back-N. Per simulare la perdita dei messaggi in rete (evento alquanto improbabile in una rete locale per non parlare di quando client e server sono eseguiti sullo stesso host), si assume che ogni messaggio sia scartato dal mittente con probabilità p. La dimensione della finestra di spedizione N, la probabilità di perdita dei messaggi p, e la durata del timeout T, sono tre costanti configurabili ed uguali per tutti i processi. Oltre all’uso di un timeout fisso, deve essere possibile scegliere l’uso di un valore per il timeout adattativo calcolato dinamicamente in base alla evoluzione dei ritardi di rete osservati. Dopo questa piccola introduzione di ciò che si è andato ad implementare ora si andrà a spiegare ciò che è stato implementato

2 Descrizione dell'Architettura

Il sistema funziona mediante l'utilizzo di una socket d'accoglienza con un numero di porta prestabilito cioè 8080, utilizzata per ricevere le varie richieste del client, infatti esso tramite un menù (con uno switch di opzioni disponibili) potrà scegliere cosa fare cioè:

- Exit: Il cilient immettendo 1 uscirà dall'applicazione e invierà un messaggio di uscita al server, che appena lo riceverà manderà un ACK ed uscirà dall'applicazione e appena il client riceverà l'ACK uscirà definitivamente;
- List: il client immettendo 2 invierà la richiesta "List" al server, dunque gli richiederà la lista di file presenti in esso, il server non appena riceve questa richiesta invierà al client come messaggio di risposta la lista dei file;
- Get: il client immettendo 3 invierà la richiesta "Get" al server e scriverà il nome del file da scaricare, il server una volta ricevuto il nome del file,

controllerà se è presente nella sua directory e in caso positivo invierà il file in pacchetti con Go-Back-N e il client manderà ACK cumulativi quando li avrà ricevuti, in caso negativo invece il server inviera un messaggio "Not Exists" al client;

- Upload: il client immettendo 4 invierà la richiesta "Upload" al server e scriverà il nome del file da inviare (sempre che sia presente all'interno della sua directory), il server chiederà alcune informazioni inerenti alla taglia del file e riceverà poi tramite pacchetti Go-Back-N il file dal client ed invierà ACK cumulativi quando li avrà ricevuti.

I dettagli di queste operazioni sia al lato Client che al lato Server verranno spiegate nel prossimo capitolo in cui si andrà a descrivere l'implementazione vera e propria dell'applicazione.

3 Implementazione

In questo capitolo verranno spiegate tutte le scelte implementative fatte per il corretto funzionamento del sistema.

3.1 Go-Back-N

RIguardo all'implementazione del protocollo di trasporto Go-Back-N sono state utilizzate due funzioni **gbnRecv()** e **gbnSend()** che si occuperanno appunto della ricezione e dell'invio di questi messaggi. La dimensione di questi messaggi è stata prefissata a 536 Bytes, che è il minimo Maximum Segment Size (da Wikipedia) e riguardo ai Timeout dei pacchetti ci può essere sia un timeout fisso e sia quello adattativo, che non è nient'altro che la formula del TimeoutInterval di TCP tramite distribuzione Gaussiana che consiste nel calcolare un SampleRTT che è la misura del tempo che intercorre tra l'invio del messaggio e la ricezione dell'ACK (ignorando segmenti ritrasmessi), poi si stimerà l'RTT con:
EstimatedRTT = (1 - α)*EstimatedRTT + α*SampleRTT
con α = 0.125 che è un valore tipico scelto perchè basta fare uno shift verso destra di 3. Poi si calcola la deviazione standart dell'RTT:
DevRTT = (1 - β)*DevRTT + β*|SampleRTT - EstimatedRTT|
con β = 0.25 che è un altro valore tipico scelto perchè basta fare uno shift verso destra di 2. Una volta calcolati questi valori nelle tabelle della distribuzione Gaussiana si trova lo z-value per calcolare il TimeoutInterval che sarà:
TimeoutInterval = EstimatedRTT + 4*DevRTT.
Le funzioni **gbnRecv()** e **gbnSend()** sono state implementate seguendo l'automa presente nel libro di Kurose & Ross “Reti di Calcolatori e Internet” e sono presenti sia nella parte Client sia nella parte Server, in entrambe le parti hanno le stesse funzionalità, cambiano soltanto alcune cose in fase di stampa su standart output

dell'intera operazione di invio o ricezione dei messaggi per riconoscere quale delle due parti sta facendo cosa. La funzione **gbnRecv()** avrà una variabile intera settata inizialmente a 0, che servirà per riconoscere se il pacchetto ricevuto è quello atteso ed all'interno di un ciclo while avverrà la ricezione del numero del pacchetto e del contenuto di esso (tramite due **recvfrom()**), che verrà scritto all'interno di un array di stringhe lunghe MSS (quindi conoscendo il numero del pacchetto sappiamo a che posizione della lista dobbiamo scrivere il contenuto di esso) e che potrà contenere tante stringhe a seconda del numero di pacchetti, comoda quando si andrà a creare il file nella directory del ricevente. Una volta fatto ciò si controlla se il numero del pacchetto è uguale al numero del pacchetto atteso, in caso positivo si manda l'ACK al mittente con il numero del pacchetto tramite una **sendto()** con una certa probabilità di perdita e si incrementa il valore in cui è salvato il numero del pacchetto atteso. In caso negativo invece, si ha un pacchetto "fuori ordine" dunque si manderà al mittente l'ACK cumulativo con il numero dell'ultimo pacchetto ricevuto (quindi il pacchetto atteso - 1), sempre tramite **sendto()** e sempre con una certa probabilità di perdità. In entrambi i casi dopo la **sendto()** c'è un controllo che stabilisce se quel pacchetto di cui si sta inviando il riscontro sia l'ultimo in maniera tale da uscire dal ciclo while. Mentre per quanto riguarda la **gbnSend()** si chiameranno prima di tutto delle variabili **struct timeval** per il calcolo del tempo di invio dei pacchetti e per il SampleRTT, un array di stringhe lunghe MSS che potrà contenere tante stringhe a seconda del numero di pacchetti, in cui si andrà a scrivere il contenuto del file letto a "chunks" di 536 bytes, un array di interi che servirà a capire se un pacchetto è stato ritrasmesso o meno (utile per ignorare pacchetti ritrasmessi), un array di interi che servirà a contare quante volte viene ricevuto l'ACK di quel pacchetto perchè appunto se quell'ACK viene ricevuto 2 volte non sarà più utile ai fini del calcolo del SampleRTT ed una volta popolati questi array si passa all'algoritmo vero e proprio di invio dei pacchetti e ricezione dei relativi ACK. Prima di tutto si settano due variabili a 0 che indicano rispettivamente, il pacchetto inviato più "vecchio" non ancora riscontrato e il prossimo pacchetto da inviare, si fa partire il timer con **gettimeofday()** e si incominciano ad inviare i pacchetti all'interno di un ciclo while. All'interno di esso prima di tutto vengono inviati tutti i pacchetti all'interno della Window Size N tramite **sendto()**, verranno inviati numero del pacchetto prima e contenuto del pacchetto poi con una certa probabilità di perdità, si fa partire il timer del SampleRTT con la **gettimeofday()** e si andrà ad incrementare la variabile in cui si ha il prossimo pacchetto da inviare. Una volta inviati tali pacchetti partiranno i timeout per le varie ricezioni degli ACK con la funzione **timeout()** che chiamerà una **struct timeval** in cui andrà a settare come **tv_sec = 0** e **tv_usec = Tout** (Tout è il TimeoutInterval citato precedentemente che partirà come valore iniziale uguale a quello del TIMEOUT prefissato) oppure **tv_usec = TIMEOUT** (timeout prefissato) e si setta questa struct all'interno della

setsockopt() con **SO_RCVTIMEO**, quando il timeout scadrà la variabile **errno** verrà settata a **EAGAIN** dunque quando la **recvfrom()** per la ricezione dell'ACK ritornerà un valore negativo (falllisce), allora si controlla anche se **errno** è uguale a **EAGAIN**, in modo tale da ritrasmettere tutti quei pacchetti scaduto quindi inviati ma non ancora riscontrati e si setta l'intero all'interno dell'array di pacchetti ritrasmessi uguale al numero del pacchetto che si sta ritrasmettendo, per indicare che quel pacchetto è stato ritrasmesso. Quando l'ACK verrà ricevuto senza errori, si incrementerà l'intero all'interno dell'array di ack ricevuti, in maniera tale da controllare sia che quel pacchetto non è stato ritrasmesso e sia che quell'ACK non è gia stato ricevuto precendemente (controllando quindi gli interi all'interno degli array dichiarati inizialmente), se appunto è così allora si può calcolare il SampleRTT quindi stoppando il timer dichiarato all'invio dei pacchetti e facendo una differenza. Fatto ciò verranno calcolati l'EstimatedRTT, il DevRTT ed infine il Tout che sarà appunto adattativo a seconda dei ritardi. Si controlla se questo è l'ultimo ACK per uscire dal ciclo while infinito, si setta la variabile che ha il numero del pacchetto più "vecchio" non ancora riscontrato uguale al numero dell'ACK appena ricevuto e si controlla se questo numero è uguale al numero del prossimo pacchetto da inviare, in caso positivo si ferma il timeout.

3.2 Client

La parte implementativa del Client è formata da una funzione **main()**, che è colei che si occuperà di istanziare la socket e di creare il menù citato precedentemente, tramite una switch che a seconda dell'opzione inserita dall'utente, eseguirà un certo tipo di operazione. Le scelte come accennate al capitolo precedente sono 4:

- Exit: sempre nella funzione **main()** verrà inviato al server la richiesta di uscita dall'applicazione tramite una **sendto()** e si controllerà che tutto vada a buon fine, alla ricezione dell'ACK da parte del server tramite una **recvfrom()**, si controllerà se questa chiamata a questa funzione sia andata a buon fine ed infine il client potrà uscire dall'applicazione che verrà terminata;
- List: alla ricezione di questo comando la funzione **main()** chiamerà un'altra funzione **list_files()**, che si occuperà di inviare tramite una **sendto()** la richiesta "List" al Server (sempre controllando che non ci siano errori), alla ricezione della lista dei file da parte del Server tramite una **recvfrom()** (controllando che non ci siano errori) stamperà su schermo la lista dei file ricevuta;
- Get: alla ricezione di questo comando la funzione **main()** chiamerà un'altra funzione **get()**, che si occuperà di inviare la richiesta "Get" al Server tramite una **sendto()**, farà inserire su schermo il nome del file da scaricare e lo

invierà al Server tramite **sendto()**, riceve dal Server un messaggio che indicherà se tale file esiste o meno tramite **recvfrom()**. Una volta ricevuto questo messaggio il client manda un messaggio "Size?" al server tramite una **sendto()** per sapere quanto sia grande il file per poi scriverlo nella propria directory ed anche per trovare il numero di pacchetti che riceverà che serviranno a loro volta a riempire la memoria condivisa in cui si andranno a scrivere i vari contenuti dei pacchetti ricevuti. Riceverà tramite **recvfrom()** la taglia del file e da qui riceverà dal server i pacchetti Go-Back-N tramite una funzione **gbnRecv()**, una volta ricevuti tutti i pacchetti andrà a scrivere un file all'interno della sua directory con tutto il contenuto dei pacchetti salvati all'interno della memoria condivisa citata precedentemente;

- Upload: alla ricezione di questo comando la funzione **main()** chiamerà un'altra funzione **upload()**, che si occuperà di inviare la richiesta "Upload" al Server tramite una **sendto()**, farà inserire su schermo il nome del file da caricare controllando se questo file esiste o meno nella propria directory, se non esiste fa riscrivere al client il nome. Se il file esiste, esso verrà inviato tramite una **sendto()** al Server, riceverà il messaggio "Size?" dal Server tramite **recvfrom()**, quindi verranno calcolate taglia e numero di pacchetti da inviare, la taglia del file verrà inviata tramite **sendto()** al Server e poi si potrà passare alla chiamata della **gbnSend()** quindi all'invio dei pacchetti Go-Back-N al Server.

3.3 Server

La parte implentativa del Server è formata da una funzione **main()**, che è colei che si occuperà di istanziare la socket e di eseguire la **bind()** di essa, inoltre riceverà dal Client la sua richiesta tramite **recvfrom()** e controllerà che tipo di richiesta è stata inviata:

- "Exit": la funzione **main()** invierà tramite **sendto()** il messaggio di uscita "Exit" e uscirà subito dopo dall'applicazione;
- "List": la funzione **main()** chiamerà un'altra funzione **list()**, che si occuperà di aprire e leggere il file list.txt, subito dopo invierà al client il contenuto di questo file tramite **sendto()**;
- "Get": la funzione **main()** chiamerà un'altra funzione **sendFile()**, che si occuperà di ricevere tramite **recvfrom()** il nome del file da inviare, aprirà

questo file e invierà un messaggio "Not Exists" se il file non esiste nella sua directory o il nome del file se esiste, sempre tramite **sendto()**. Riceverà il messaggio "Size?" dal Client tramite **recvfrom()**, dunque calcolerà la grandezza del file e la invierà al Client tramite **sendto()** e calcolerà il numero di pacchetti da inviare, una volta fatto questo chiamerà la funzione **gbnSend()**;

- "Upload": la funzione **main()** chiamerà un'altra funzione chiamata **putFile()** che si occuperà di ricevere il nome del file che dovrà caricare nella directory tramite **recvfrom()**, invierà il messaggio "Size?" tramite **sendto()** al Client, riceverà la taglia del file tramite **recvfrom()**, così per calcolare il numero di pacchetti che dovrà ricevere così per riempire la memoria condivisa in cui si andranno a scrivere i vari contenuti dei pacchetti ricevuti. Dopo ciò chiamerà la **gbnRecv()** e una volta ricevuti tutti i pacchetti Go-Back-N, allora potrà scrivere un nuovo file nella propria directory che avrà come contenuto tutto ciò che è stato ricevuto dal Client, inoltre verrà aggiornato il file list.txt.

4 Limitazioni Riscontrate

Ci sono state alcune limitazioni poi risolte con vari test effettuati sul codice, uno di questi è stato il limite di 10^6 microsecondi nella struct timeval, il timeout in microsecondi era così alto perchè il SampleRTT veniva calcolato erroneamente. Infatti esso considerava anche ritrasmissioni o ACK duplicati, dunque per risolvere ciò sono stati utilizzati due array di interi per le ritrasmissioni e per ACK gia ricevuti che sono stati gia citati nel capitolo precedente. Si sono avute anche limitazioni nella parte Server, cioè quando il Client faceva due richieste consecutive di Get o Upload, il Server chiudeva l'applicazione per un errore nella recvfrom(). Per risolvere queste problema sono state aggiunte delle etichette in maniera tale da rimandare il programma in quella etichetta così da rifare la recvfrom() quando fallisce e quando ha come errno EAGAIN. Un'altra limitazione riscontrata è stata quella di aggiornare il file list.txt con un nome di un file gia presente, quindi il file aveva al suo interno delle stringhe duplicate, si è risolto ciò tramite una fscanf che

legge appunto il contenuto del file e controlla se quel file è gia presente all'interno di list.txt. Un'altra limitazione è stata quella di inviare "chunks" di file e di salvarli per poi scriverli all'interno di un file. Per risolvere ciò per l'invio sono stati utilizzati array di tante stringhe lunghe 536 bytes, a seconda di quanti pacchetti vengono inviati e per la ricezione è stata utilizzata una memoria condivisa che è stata già esplicata nel capitolo precedente.

5 Software utilizzato

In questo capitolo verranno spiegati quali software sono stati utilizzato per lo sviluppo e per il testing del sistema creato.

5.1 Sviluppo

Per lo sviluppo del codice è stato utilizzato come IDE **Sublime Text** e come Sistema Operativo **Linux Kubuntu 18.04**.

5.2 Testing

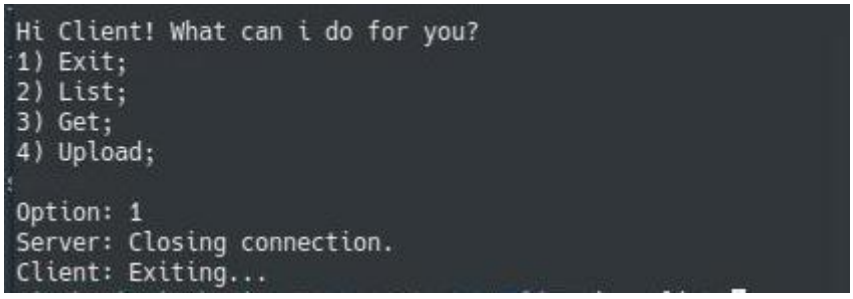
Per il testing è stato utilizzato **Kate Editor** per scrivere i vari risultati dei test svolti e per eseguire questi test è stata utilizzata la shell di linux.

6 Esempi di funzionamento

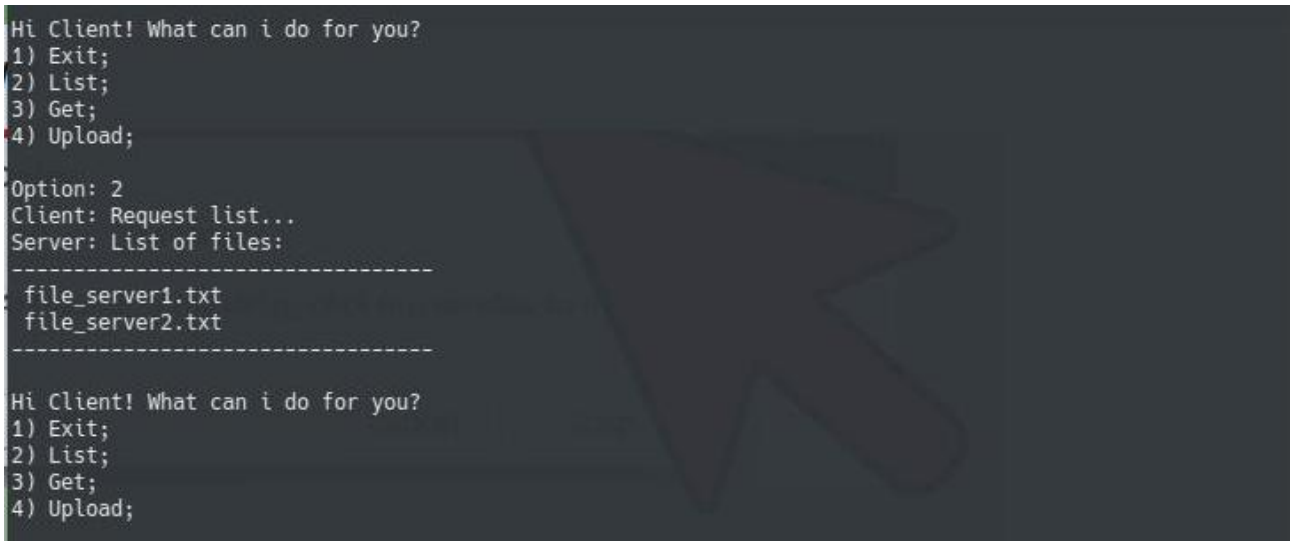
In questo capitolo si andranno a far vedere immagini di esempio di come si presenta il sistema, sia lato client che lato server.

6.1 Client

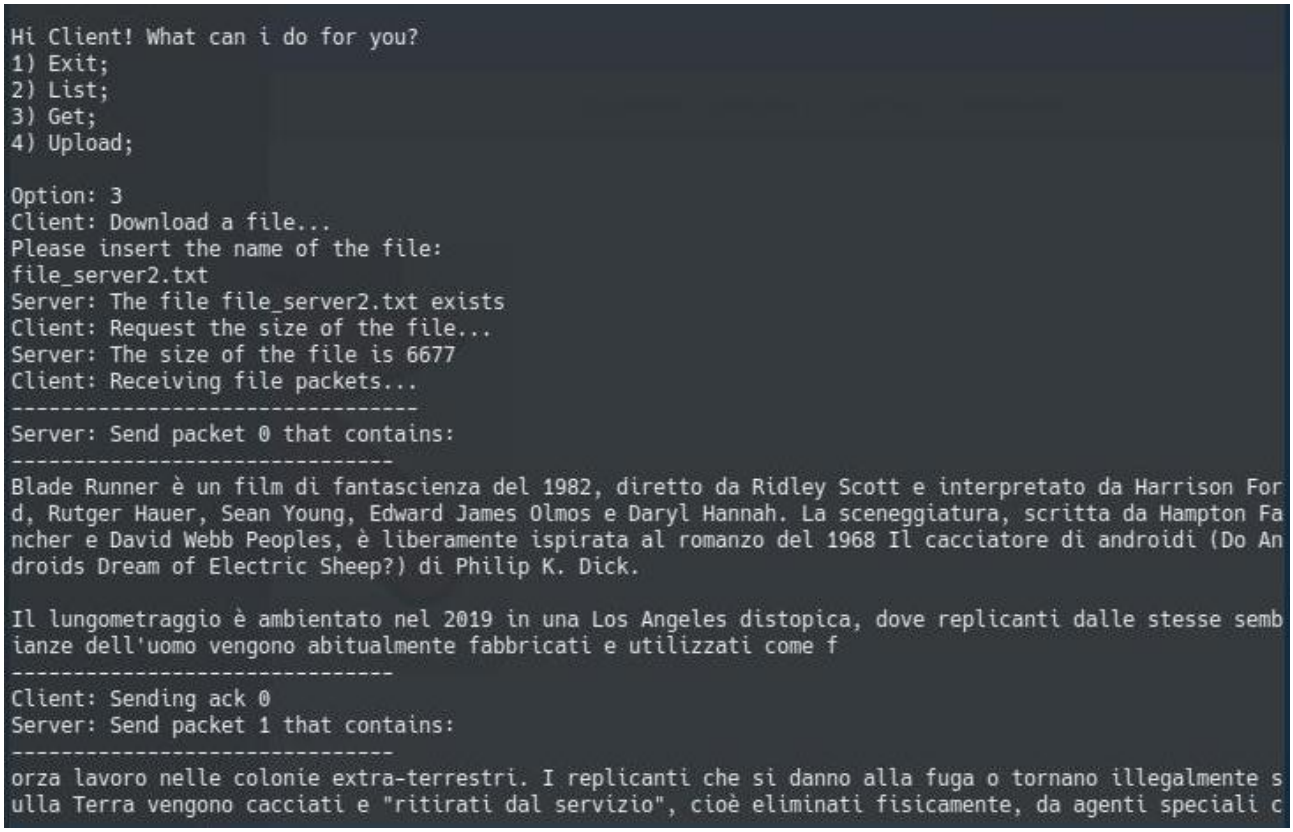
Exit:



List:



Get: Quando il client immette 3 e tutta la fase iniziale



Get: La fase finale con la ricezione dei pacchetti e l'invio degli ACK in questo caso anche persi;

```
 sulle sue performance in tre film di Pa
--------------------------------
Client: Lost Ack for packet 10
Server: Send packet 11 that contains:
--------------------------------
ul Verhoeven: Keetje Tippel, Soldato d'Orange e Fiore di carne. Philip K. Dick stesso approvò la scelta
, descrivendo Hauer come «il perfetto Roy Batty. Freddo, ariano e senza difetti». Dei molti film girati
 da Hauer Blade Runner rimane il suo preferito. Come ha avuto modo di affermare nel 2001: «Blade Runner
 non ha bisogno di spiegazioni. Semplicemente [è]. [...] Non esiste niente di simile. Essere parte di u
n vero capolavoro che ha cambiato il modo di pensare del mondo. È magnifico».

Io ne ho viste cose che voi umani non
--------------------------------
Client: Sending ack 11
Server: Send packet 12 that contains:
--------------------------------
 potreste immaginarvi. Navi da combattimento in fiamme al largo dei bastioni di Orione. E ho visto i ra
ggi B balenare nel buio vicino alle porte di Tannhäuser. E tutti quei momenti andranno perduti nel temp
o come lacrime nella pioggia. È tempo di morire.

--------------------------------
Client: Sending ack 12
--------------------------------
Download successful!

Hi Client! What can i do for you?
1) Exit;
2) List;
3) Get;
4) Upload;
```

Upload: Quando il client inserisce il nome e tutta la fase di invio dei pacchetti;

```
Hi Client! What can i do for you?
1) Exit;
2) List;
3) Get;
4) Upload;

Option: 4
Client: Upload a file...
Please insert the name of the file:
file_client1.txt
File file_client1.txt exists
Server: Requesting -> Size?
Number of packets to send -> 5
Size of the file -> 2520
Client: Sending file packets...
--------------------------------
Client: Sending packet 0
Client: Sending packet 1
Client: Sending packet 2
Client: Lost packet 3
Client: Sending packet 4
Server: Ack packet 0
TIMEOUT IS -> 360.000000 us
Server: Ack packet 1
TIMEOUT IS -> 625.250000 us
Server: Ack packet 2
TIMEOUT IS -> 802.937500 us
Client: Timeout packet 3
Client: Resending packet 3
Client: Timeout packet 4
```

Upload: La fase finale dell'Upload.

6.2 Server

Exit:

List:

Get: Praticamente identica ad Upload di Client

```
Hi Server!
Client: Requesting Get
Client: Request filename -> file_server2.txt
Client: Requesting -> Size?
Number of packets to send -> 13
Size of the file -> 6677
Server: Sending file packets...
--------------------------------
Server: Sending packet 0
Server: Sending packet 1
Server: Sending packet 2
Server: Lost packet 3
Server: Sending packet 4
Client: Ack packet 0
TIMEOUT IS -> 281.250000 us
Server: Sending packet 5
Client: Ack packet 1
TIMEOUT IS -> 223.218750 us
Server: Sending packet 6
Client: Ack packet 2
TIMEOUT IS -> 189.128906 us
Server: Sending packet 7
Client: Ack packet 2
TIMEOUT IS -> 161.284668 us
Client: Ack packet 2
TIMEOUT IS -> 138.159241 us
Client: Ack packet 2
TIMEOUT IS -> 119.603203 us
Server: Timeout packet 3
Server: Resending packet 3
Server: Timeout packet 4
```

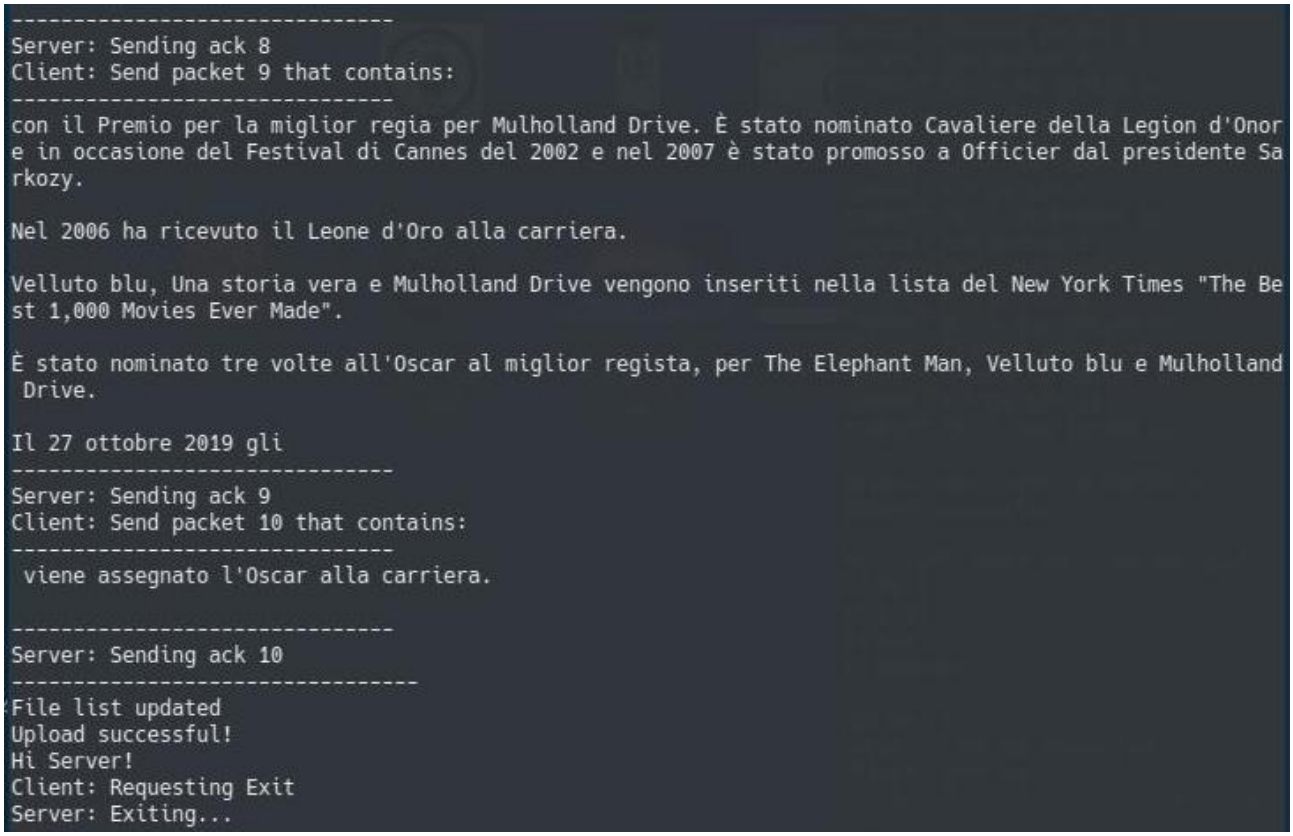
Upload: Fase iniziale di richiesta della grandezza per il resto simile a Get di Client;

```
Hi Server!
Client: Requesting Upload
Client: Request filename -> file_client2.txt
Server: Request the size of the file...
Client: The size of the file is 5390
Packets are 11
Server: Receiving file packets...
--------------------------------
Client: Send packet 0 that contains:
--------------------------------
David Keith Lynch (Missoula, 20 gennaio 1946) è un regista, sceneggiatore, attore, musicista, produttor
e cinematografico e pittore statunitense.

È stato descritto da The Guardian come "il regista più importante di quest'epoca",mentre AllMovie l'ha
definito "l'uomo del Rinascimento del Cinema moderno americano".

Nasce come pittore; le sue opere sono attualmente esposte in musei e gallerie d'arte come il Museum of
Modern Art di New York e la Pennsylvania Academy of the Fine Arts di Philadelphia. Successivamente entr
a nel mondo
--------------------------------
Server: Sending ack 0
Client: Send packet 1 that contains:
--------------------------------
 del cinema divenendo regista, sceneggiatore e produttore, spesso anche nel ruolo di montatore, scenogr
afo, progettista del suono e attore nei suoi film. È anche musicista, cantante e scrittore. Nonostante
non riscuota sempre successo ai box office, Lynch è apprezzato dai critici e gode di un cospicuo seguit
o di fan.Nel corso degli anni ha ricevuto tre nomination al Premio Oscar per la regia (per The Elephant
 Man, Velluto blu e Mulholland Drive),la Palma d'oro al Festival di Cannes 1990 per Cuore selvaggio,il
Prix de la mise en
--------------------------------
Server: Sending ack 1
```

Upload: Fase finale;



7 Prestazioni protocollo

In questo capitolo si andranno a valutare le prestazioni del protocollo implementato, come vari tipi di test si è scelto di variare la dimensione della finestra di spedizione, la probabilità di perdità e il timeout fisso o adattivo. Questo capitolo verrà diviso in tre paragrafi in cui andremo a mettere test con due timeout fissi ed un test con timeout adattivo. Questi test sono stati effettuati in quattro file che sono file_server1.txt grande 2084, file_server2.txt grande 6677, file_client1.txt grande 2520 e file_client2.txt grande 5390 e le operazioni sotto test sono le Get e le Upload. Sono stati fatti otto test differenti per ogni tipo di Timeout scelto, quattro con la finestra di spedizione che varia di 1, 5, 10, 50 come dimensione, la probabilità di perdita settata al 25%, mentre gli altri quattro test hanno stessa variazione della dimensione della finestra di spedizione ma con probabilità di perdita al 75%. I timeout scelti per il testing sono stati uno di gran lunga più grande rispetto al RTT medio, uno di poco più grande al RTT medio ed infine il timeout adattativo di TCP accennato nei precedenti capitoli.

7.1 Timeout fisso a 400 millisecondi

1. N = 1, P = 25%,
    1. Download file_server1.txt -> 0.822180 s
    2. Download file_server2.txt -> 0.822180 s
    3. Upload file_client1.txt -> 0.834508 s
    4. Upload file_client2.txt -> 1.676001 s
2. N = 5, P = 25%
    1. Download file_server1.txt -> 0.817781 s
    2. Download file_server2.txt -> 1.237661 s
    3. Upload file_client1.txt -> 0.418599 s
    4. Upload file_client2.txt -> 1.667363 s
3. N = 10, P = 25%
    1. Download file_server1.txt -> 0.841994 s
    2. Download file_server2.txt -> 0.842128 s
    3. Upload file_client1.txt -> 0.416375 s
    4. Upload file_client2.txt -> 0.415346 s
4. N = 50, P = 25%
    1. Download file_server1.txt -> 0.840884 s
    2. Download file_server2.txt -> 0.829209 s
    3. Upload file_client1.txt -> 0.402895 s
    4. Upload file_client2.txt -> 0.413951 s
5. N = 1, P = 75%
    1. Download file_server1.txt -> 1.245109 s
    2. Download file_server2.txt -> 86.519625 s
    3. Upload file_client1.txt -> 3.315949 s
    4. Upload file_client2.txt -> 72.376805 s

6. N = 5, P = 75%
    1. Download file_server1.txt -> 1.258283 s
    2. Download file_server2.txt -> 19.960308 s
    3. Upload file_client1.txt -> 1.680836 s
    4. Upload file_client2.txt -> 18.297869 s
7. N = 10, P = 75%
    1. Download file_server1.txt -> 1.257771 s
    2. Download file_server2.txt -> 15.794441 s
    3. Upload file_client1.txt -> 1.658167 s
    4. Upload file_client2.txt -> 14.132391 s
8. N = 50, P = 75%
    1. Download file_server1.txt -> 1.264158 s
    2. Download file_server2.txt -> 15.796837 s
    3. Upload file_client1.txt -> 1.678693 s
    4. Upload file_client2.txt -> 14.144617 s

7.2 Timeout fisso 400 microsecondi

1. N = 1, P = 25%,
    1. Download file_server1.txt -> 0.020582 s
    2. Download file_server2.txt -> 0.031786 s
    3. Upload file_client1.txt -> 0.017123 s
    4. Upload file_client2.txt -> 0.035342 s
2. N = 5, P = 25%
    1. Download file_server1.txt -> 0.015146 s
    2. Download file_server2.txt -> 0.026751 s
    3. Upload file_client1.txt -> 0.007230 s
    4. Upload file_client2.txt -> 0.024981 s
3. N = 10, P = 25%
    1. Download file_server1.txt -> 0.016612 s

2. Download file_server2.txt -> 0.015432 s
3. Upload file_client1.txt -> 0.007174 s
4. Upload file_client2.txt -> 0.007624 s

4. N = 50, P = 25%
    1. Download file_server1.txt -> 0.014032 s
    2. Download file_server2.txt -> 0.016098 s
    3. Upload file_client1.txt -> 0.007938 s
    4. Upload file_client2.txt -> 0.007161 s
5. N = 1, P = 75%
    1. Download file_server1.txt -> 0.024185 s
    2. Download file_server2.txt -> 1.706143 s
    3. Upload file_client1.txt -> 0.070966 s
    4. Upload file_client2.txt -> 1.465355 s
6. N = 5, P = 75%
    1. Download file_server1.txt -> 0.022725 s
    2. Download file_server2.txt -> 0.392346 s
    3. Upload file_client1.txt -> 0.031806 s
    4. Upload file_client2.txt -> 0.378259 s
7. N = 10, P = 75%
    1. Download file_server1.txt -> 0.021736 s
    2. Download file_server2.txt -> 0.311830 s
    3. Upload file_client1.txt -> 0.033780 s
    4. Upload file_client2.txt -> 0.281190 s
8. N = 50, P = 75%
    1. Download file_server1.txt -> 0.023738 s
    2. Download file_server2.txt -> 0.318418 s
    3. Upload file_client1.txt -> 0.031120 s
    4. Upload file_client2.txt -> 0.294759 s

7.3 Timeout Adattativo

1. N = 1, P = 25%,
    1. Download file_server1.txt -> 0.014245 s
    2. Download file_server2.txt -> 0.039553 s
    3. Upload file_client1.txt -> 0.017856 s
    4. Upload file_client2.txt -> 0.032091 s
2. N = 5, P = 25%
    1. Download file_server1.txt -> 0.016868 s
    2. Download file_server2.txt -> 0.027747 s
    3. Upload file_client1.txt -> 0.008052 s
    4. Upload file_client2.txt -> 0.025909 s
3. N = 10, P = 25%
    1. Download file_server1.txt -> 0.017112 s
    2. Download file_server2.txt -> 0.015805 s
    3. Upload file_client1.txt -> 0.009140 s
    4. Upload file_client2.txt -> 0.011494 s
4. N = 50, P = 25%
    1. Download file_server1.txt -> 0.016374 s
    2. Download file_server2.txt -> 0.019919 s
    3. Upload file_client1.txt -> 0.006514 s
    4. Upload file_client2.txt -> 0.008857 s
5. N = 1, P = 75%
    1. Download file_server1.txt -> 0.027717 s
    2. Download file_server2.txt -> 1.744982 s
    3. Upload file_client1.txt -> 0.068173 s
    4. Upload file_client2.txt -> 1.427554 s
6. N = 5, P = 75%
    1. Download file_server1.txt -> 0.027771 s

2. Download file_server2.txt -> 0.393972 s
3. Upload file_client1.txt -> 0.034190 s
4. Upload file_client2.txt -> 0.376040 s

7. N = 10, P = 75%
    1. Download file_server1.txt -> 0.024357 s
    2. Download file_server2.txt -> 0.330054 s
    3. Upload file_client1.txt -> 0.032346 s
    4. Upload file_client2.txt -> 0.277501 s
8. N = 50, P = 75%
    1. Download file_server1.txt -> 0.021472 s
    2. Download file_server2.txt -> 0.315451 s
    3. Upload file_client1.txt -> 0.031824 s
    4. Upload file_client2.txt -> 0.286285 s

7.4 Commenti

Si nota dai test effettuati che scegliere nel primo caso un timeout troppo lungo può causare una lenta ripresa ai segmenti persi e più si ha una maggiore probabilità di perdita e soprattutto una dimensione di finestra molto bassa, più il tempo di esecuzione sarà altissimo come successo per esempio nel test 5 del 7.1, dunque la scelta di un timeout fisso di gran lunga più grande del RTT medio non si è rilevata conveniente. Riguardo invece alla scelta di un timeout di poco più grande del RTT medio, si nota come sia estremamente efficace, infatti esso e il timeout adattativo in alcuni casi quasi si equivalgono, nonostante ci siano alcune ritrasmissioni inutili. Dunque la soluzione del timeout adattativo trasmette più sicurezza rispetto a un timeout fisso abbastanza piccolo, però comunque anche la seconda opzione può rilevarsi vantaggiosa. Inoltre il protocollo implementato ha delle ottime prestazioni in quanto all'aumentare della dimensione della finestra di spedizione si vede un netto miglioramento del tempo di esecuzione.

8 Manuale

Questo paragrafo sarà dedicato all'installazione e alla configurazione del sistema creato e il suo relativo utilizzo. Per la corretta esecuzione del programma bisogna aprire due terminali uno per la parte Client ed uno per la parte Server.

8.1 Configurazione

Per compilare il programma basta andare nella cartella Client o Server, aprire una shell e digitare

gcc -o client client.c

gcc -o server server.c.

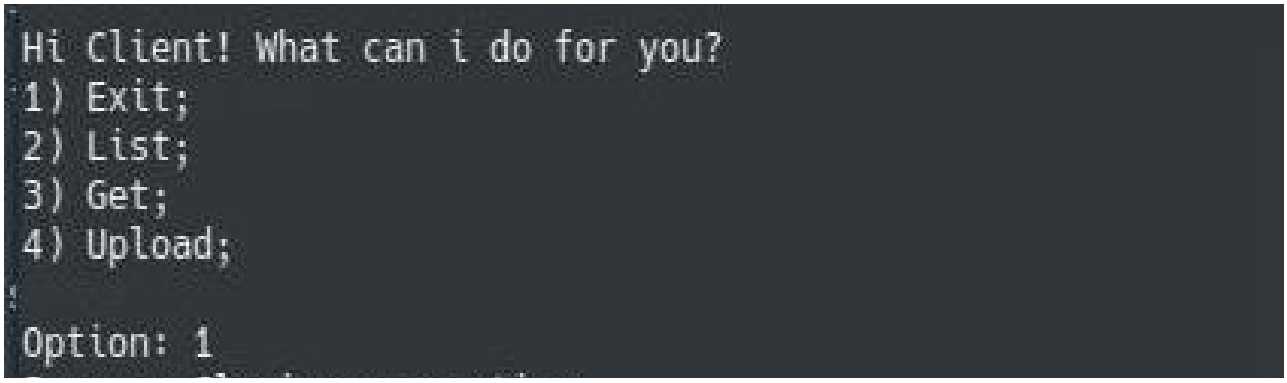
Una volta compilato il programma basta digitare subito dopo

./client

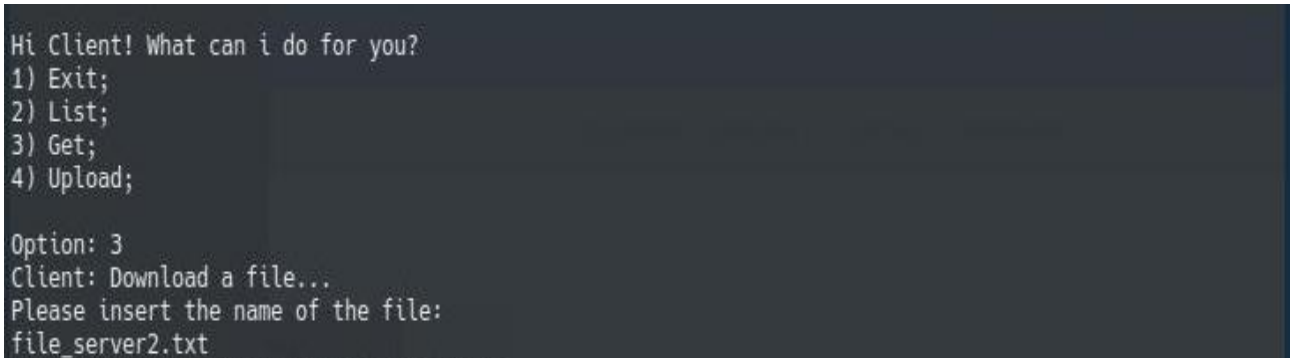
./server.

8.2 Utilizzo

Nel lato server non si dovrà inserire nulla da standard input e eseguirà tutto autonomamente, le varie esecuzioni del server si possono trovare nel capitolo 6.2 inerente ai vari esempi di esecuzione. Una volta entrati nel lato Client bisognerà inserire un intero dopo "Option" tra 1 e 4 per scegliere ciò che si vuole fare.



Se si inserisce 3 si genererà un sottomenù che ti farà inserire da standard input il nome del file da scaricare sotto la scritta "Please insert the name of the file".



Se si inserisce 4 si genererà un sottomenù che ti farà inserire da standard input il nome del file da caricare nella directory del server sotto la scritta "Please insert the name of the file".