Relazione Corso di Ingegneria Di Internet E Web

Autori: Contino Pier Francesco, Ponzo Alex

Trasferimento file su UDP

Il progetto consiste nel realizzare un'applicazione per il trasferimento affidabile di file utilizzando il protocollo di trasporto UDP. Per farlo abbiamo utilizzato il linguaggio C, e più in particolare le API per la gestione dei socket di Berkeley.

Architettura utilizzata e scelte progettuali:

Il progetto si basa su un'architettura client-server, con il server in grado di elaborare contemporaneamente le richieste di molteplici client.

Tipologie di richieste

Il client può inviare tre diversi tipi di richieste:

1. **List**: consente di ottenere dal server la lista dei file contenuti nella cartella di download;
2. **Get**: consente di trasferire sul client (in una apposita cartella) un file presente nella cartella di download del server;
3. **Put**: consente di trasferire sul server (nella cartella di download) un file presente nella cartella locale del client.

Ricevuta la richiesta il server ne verifica la validità:
-nel caso non sia valida informa il client con un messaggio contente un codice di errore specifico

Tipologie di errore

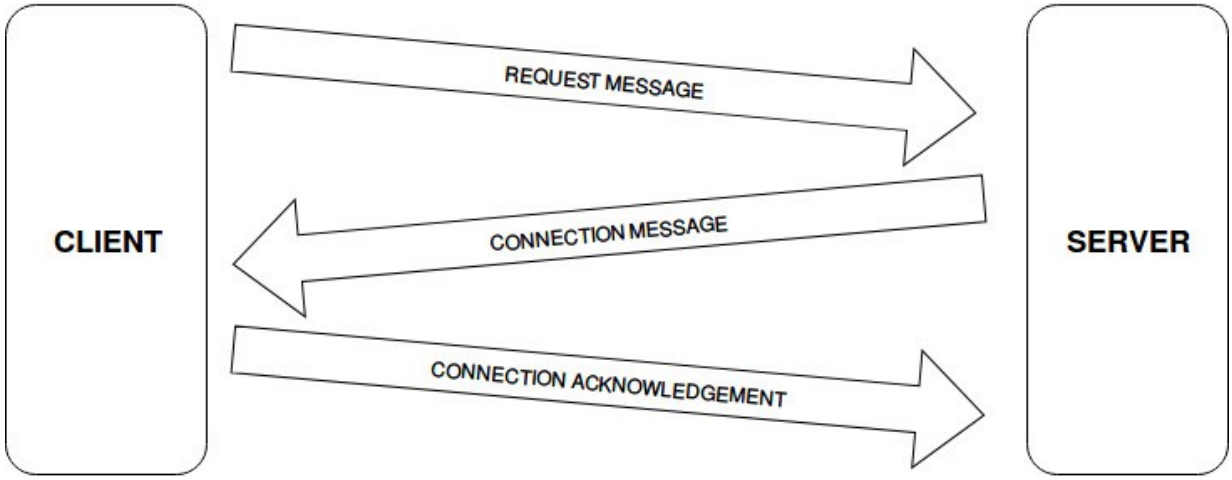
Codice errore	Richiesta	Causa
1	GET	Il file non è stato trovato nella cartella di download
2	PUT	Il file è già in stato di caricamento nella cartella del server da parte di un altro utente

- nel caso sia valida invia un messaggio di risposta in cui informa il client dell'avvenuta accettazione della richiesta e continua la procedura per l’adempimento della stessa.

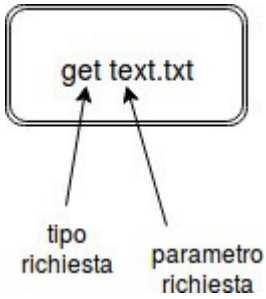
Fasi della connessione

1. Stabilire la connessione

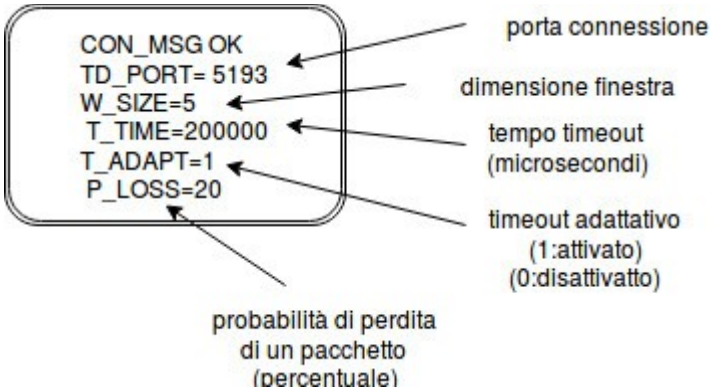
La connessione viene stabilita con un handshake a tre vie (come in figura):



Il ***request message*** contiene una richiesta tra quelle elencate precedentemente con i suoi parametri eventuali.



Il ***connection message*** contiene al suo interno dei parametri di connessione come la dimensione della finestra e la durata del timeout, così da assicurare che in tutte le trasmissioni la dimensione della finestra e la durata del timeout siano coerenti tra i vari client ed il server.

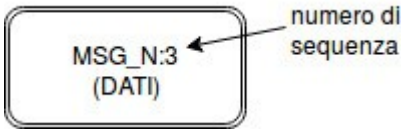


Una volta ricevuto il ***connection acknowledgement*** la comunicazione passa alla fase successiva.

2. Trasferire i dati

Il trasferimento dati avviene usando il protocollo a finestra *selective repeat al fine di garantire una comunicazione affidabile che possa usare la banda disponibile anche in presenza di tempi di latenza non trascurabili.*
La comunicazione è di tipo monodirezionale quindi si possono etichettare i due host come sender (quello che invia i dati) e come receiver (quello che invia solo le conferme di ricezione :ack).

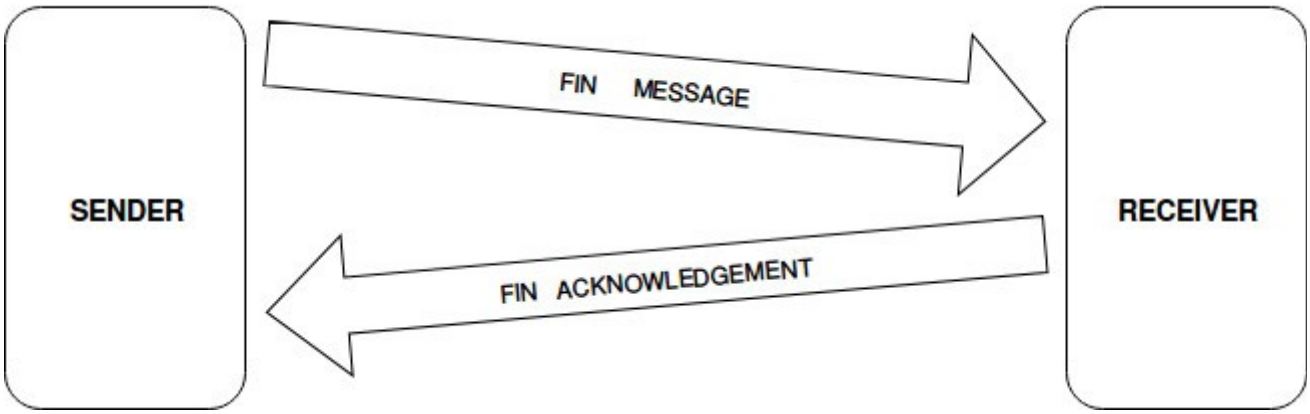
Ogni pacchetto ha una dimensione massima prefissata (1024 byte).



Formato di un messaggio contenente i dati da trasferire
Il numero di sequenza è compreso tra 0 e 2 volte la dimensione della finestra

3. Chiudere la connessione

Infine la chiusura della connessione viene stabilita dal sender che invia l'apposito messaggio "fin message" al quale il receiver risponde con il messaggio "fin acknowledgement"(immagine).



La trasmissione affidabile è garantita per tutte e tre le fasi della trasmissione.

Gestione file

La cartella del server contiene file comuni accessibili ad ogni client, dato che non vi è nessuna forma di autenticazione (come da direttive).
I client su host differenti avranno invece cartelle contenenti diversi file.

Abbiamo reso possibile la modifica dei path delle cartelle (sia lato client che lato server) e degli altri parametri fondamentali per la comunicazione che vengono caricati dal server all'avvio.

Descrizione dell'implementazione:

Server ricorsivo

Per ottenere la concorrenza del server vengono utilizzati molteplici thread, uno (principale) che si occupa di ricevere le richieste per passarle agli altri thread (generati al momento) i quali si occupano di eseguire le richieste.
Abbiamo utilizzato una porta principale per identificare il thread che esamina le richieste ed un'altra per i vari thread che si occupano del trasferimento dati.

All’inizio dell’operazione ogni thread effettua una connessione con il suo client tramite connect cosicché ogni messaggio di un’operazione viene identificato dalla coppia «indirizzo client , porta client» e viene ricevuto solo dal thread che si occupa di tale operazione.
Il client conosce solo la porta principale, quindi la porta per il trasferimento dati gli viene comunicata dal server nella prima fase della connessione.
Questo tipo di implementazione ci consente di non sovraccaricare il thread principale, in modo che sia sempre disponibile a gestire nuove richieste.
Inoltre appena un’operazione viene terminata il server rilascia tutte le risorse collegate al thread che si occupava di tale operazione.

Timeout

Il protocollo ***selective repeat*** prevede che ogni pacchetto inviato abbia un proprio timeout, il quale una volta scaduto porti alla ritrasmissione del relativo pacchetto; per implementare un sistema che facesse questo abbiamo deciso di utilizzare i timer POSIX.
I timer POSIX ci consentono inoltre di ottenere il tempo mancante alla loro scadenza quando vengono interrotti, in questo modo risulta più semplice ricavare una stima del round trip time.

Timeout adattativo

Per l’ implementazione del timeout adattativo ci siamo rifatti all’implementazione del TCP:

- misuriamo il round trip time di un pacchetto
- calcoliamo una media pesata dei vari round trip time:
  SRTT = (1-g) * SRTT + g * (RTT)
- calcoliamo la deviazione standard media:
  MSDEV = (1-h) * MSDEV + h * |RTT-SRTT|
- calcoliamo il nuovo timeout:
  RTO = SRTT + 4 * MSDEV

I valori usati per i parametri sono:
g=1/8
h=1/4

Questi parametri sono quelli usati di solito da TCP, ed essendo potenze di due rendono più rapide le operazioni di moltiplicazione che vengono implementate sotto forma di shift binario.
In caso di perdita di un pacchetto il round trip time stimato viene incrementato.
Bisogna considerare che nei vari test eseguiti la variazione dei ritardi è molto bassa poiché il sistema è stato testato in locale. Gli eventi di perdita sono dati principalmente dal parametro di fallimento dell’invio.

Configurazione dei parametri

Per la personalizzazione dei parametri vengono usati dei file di configurazione (contraddistinti dall'estensione .conf), uno per il client ed uno per il server.

Il file di configurazione del client comprende i parametri:
1. Porta del thread principale del server;
2. Indirizzo IP del server;
3. Path della cartella locale del client.

Il file di configurazione del server invece comprende i parametri:
1. Porta del thread principale;
2. Porta dei thread incaricati di evadere le richieste dei client;
3. Path della cartella di download del server;
4. Dimensione della finestra di spedizione/ricezione;
5. Scelta del tipo di timeout (fisso o adattativo);
6. Tempo di timeout, che nel caso di timeout adattativo sarà il valore di partenza ;
7. Probabilità di perdita di un pacchetto.

Struttura codice

Il progetto è stato suddiviso in moduli tramite la scrittura di file di libreria così da favorire l'implementazione parallela. I moduli sono i seguenti (ciascuno accompagnato dal proprio header file):
- mylib.c, contenente funzioni base di uso comune all'interno del progetto;
- mytrslib.c, contenente funzioni che si occupano della trasmissione su socket e della concorrenza;
- recv_window_lib.c, contenente funzioni e strutture dati utili alla ricezione affidabile;
- send_window_lib.c, contenente funzioni e strutture dati utili all'invio affidabile.
- timeout_lib.c, contenente funzioni per la gestione dei timeout

Viene inoltre utilizzato un header file per la gestione delle costanti comuni a tutti gli altri moduli (constant_lib.h).

Piattaforma software:

Sistema Operativo: Linux (Xubuntu);
Compilatore: Gcc;
Editor: Geany;

Esempi di funzionamento:

Al fine di permettere il funzionamento dell'architettura è necessario che il server sia sempre attivo (./server.o) prima dell'avvio della richiesta del client.

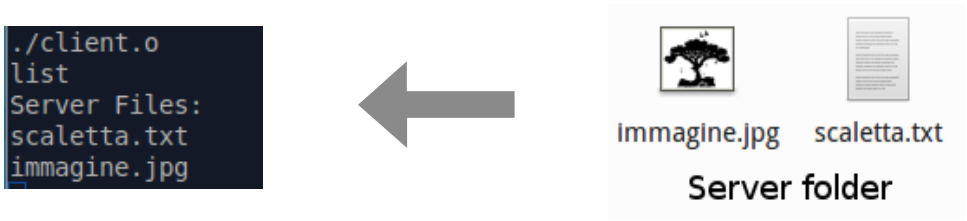
0) Esecuzione server
Passo preliminare necessario per il lancio dei client.

```
./server.o
```

1) Comando client errato
In caso di comando errato (diverso cioè da List, Get e Put) o di errato numero di argomenti per l'azione richiesta (1 per List, 2 per Get e Put) verrà stampata una panoramica delle funzioni e del loro metodo di utilizzo.

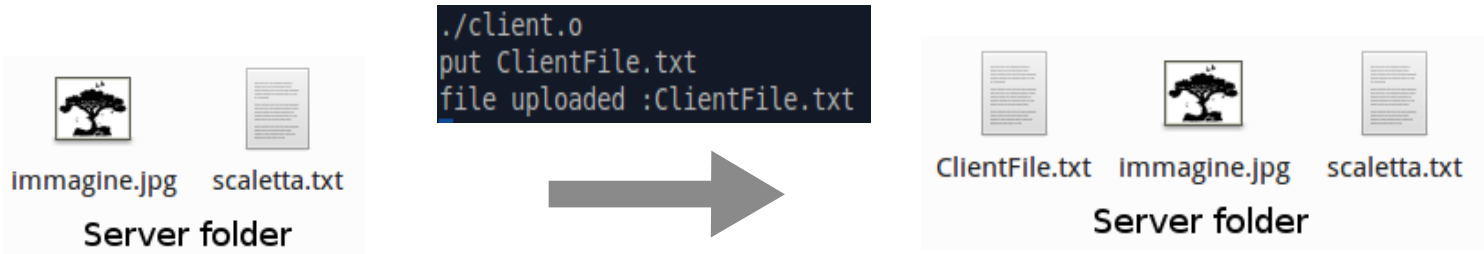
```
./client.o
wrong_function
Usage:
 list (get the list of files in server)
 get 'file_name' (download the file named 'file_name' from the server)
 put 'file_name' (upload the file named 'file_name' to the server)
```

2) Comando client *«list»*
Il client richiede al server la lista dei file presenti nella cartella di download.

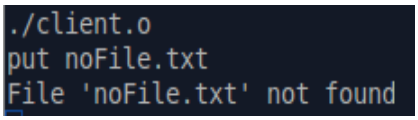


3) Comando client *«get immagine.jpg»*
Il client richiede il download dal server di un file (con nome 'immagine.jpg').

4) Comando client *«put ClientFile.txt»*
Il client esegue l'upload sul server di un file (con nome 'ClientFile.txt') presente nella propria directory.



5) Comando client *«put noFile.txt»* (corrispettivo *«get noFile.txt»*)
Il client tenta di eseguire l'upload sul server del file con nome 'noFile.txt' non presente nella propria directory.
Una risposta simile si otterrà nel caso il client richieda il download dal server di un file ivi non presente.



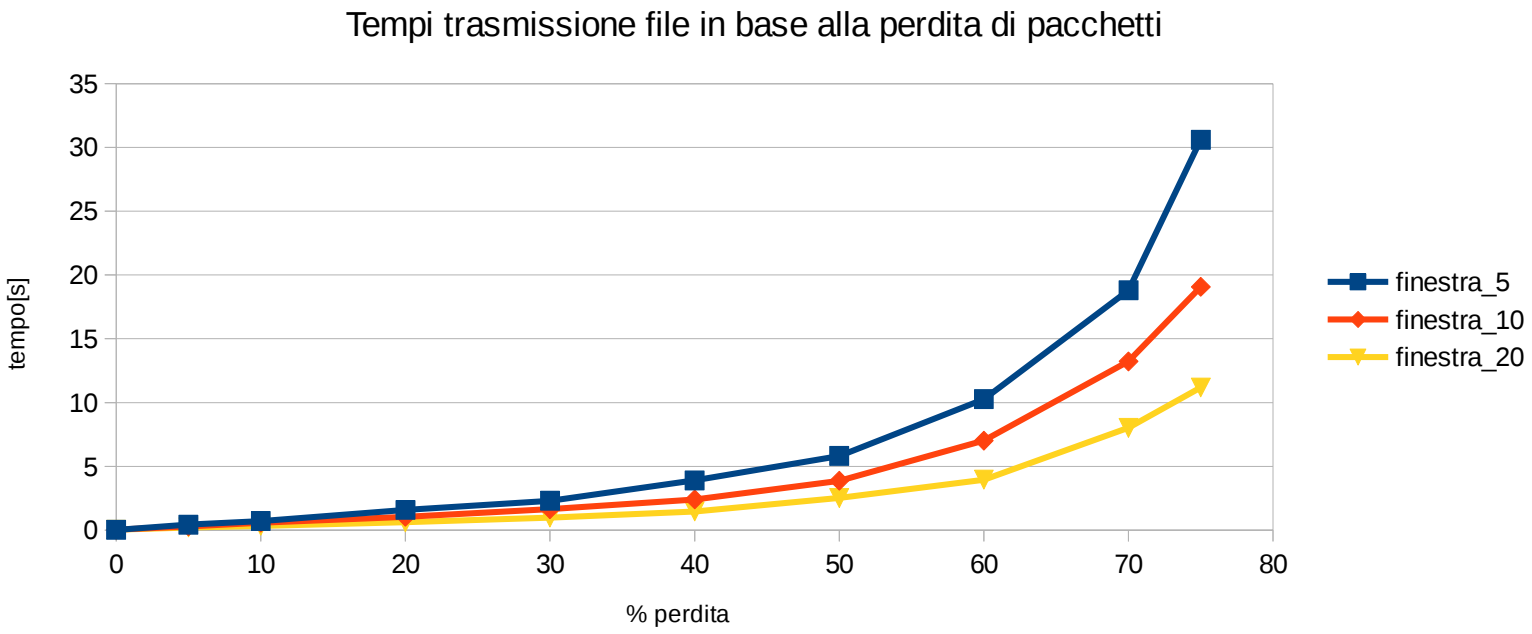
Prestazioni protocollo:

Modalità misurazioni

Al fine di simulare un reale impiego del sistema ed avere valori affidabili della misurazione delle prestazioni abbiamo effettuato i test con multipli client che inoltrano la medesima richiesta su file differenti; questo permette anche di valutare la risposta del server nel caso di più richieste in contemporanea.
I valori nei grafici sono ottenuti calcolando la media dei tempi misurati nelle quattro esecuzioni simultanee di ogni test.
I test sono effettuati in locale il che comporta una probabilità di perdita esterna all'applicazione praticamente nulla oltre ad un RTT molto basso e ad una banda molto elevata.
Il file utilizzato nelle misurazioni è di dimensione 1Mb.
I test sono effettuati sull'operazione di get (i tempi dell'operazione di put sono congruenti).

Grafico 1

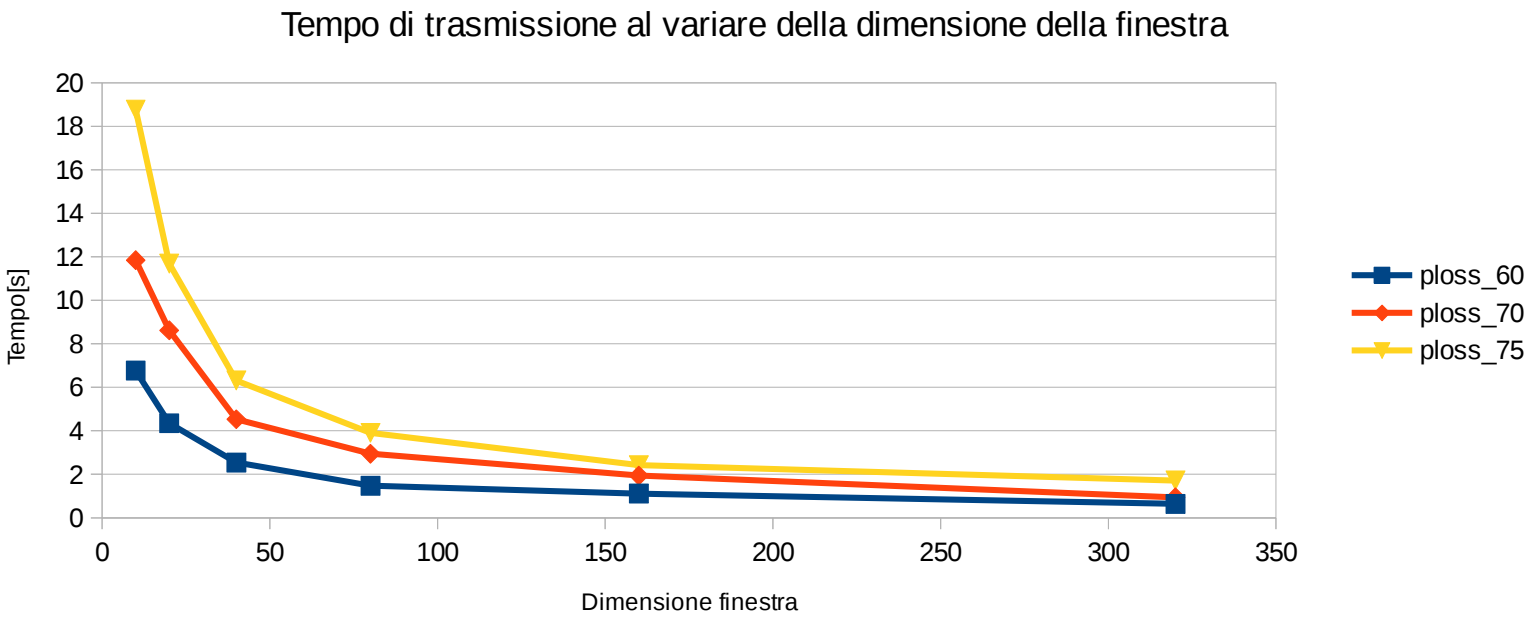
Timeout: 5 ms Adattativo: No



Da questo grafico si può vedere che il tempo di trasmissione cresce rapidamente rispetto al parametro di perdita dei pacchetti e ha un asintoto verticale quando quest'ultimo raggiunge il 100%.
Inoltre le prestazioni del protocollo migliorano rapidamente con l'aumentare della dimensione della finestra in condizione di alto tasso di errore.
Questo comportamento è dovuto alla trasmissione parallela, che va ad utilizzare il tempo in cui il protocollo sarebbe stato inattivo perché in attesa degli ack; il tempo di attesa dipende dal timeout e dal round trip time.

Grafico 2

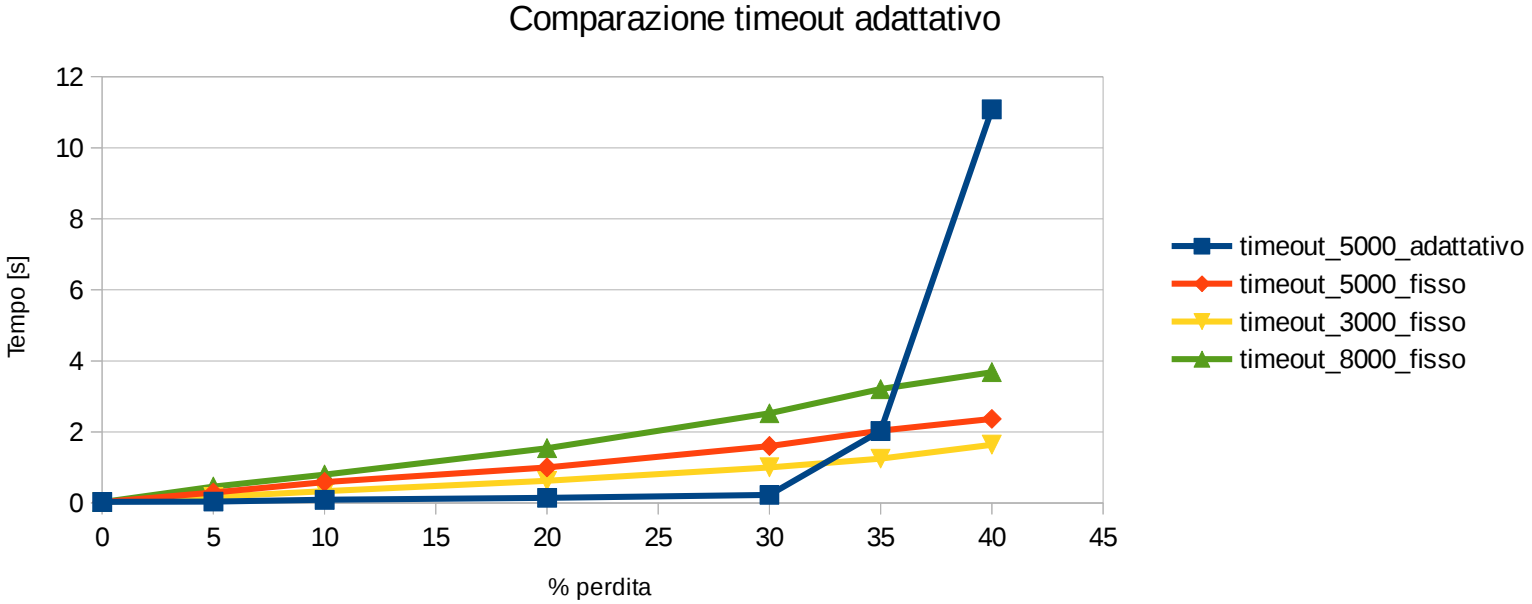
Timeout: 5 ms Adattativo: No



Possiamo vedere dal secondo grafico che il tempo d'invio diminuisce fino ad avvicinarsi asintoticamente a un valore minimo; questa soglia non può essere superata perché la trasmissione è vincolata dalla banda massima.
Si può vedere questo fenomeno come una trasmissione in cui si hanno così tanti pacchetti da trasmettere, prima di aspettare il loro riscontro, che questa diviene continua poiché il collo di bottiglia è dato dal tempo di trasferimento.

Grafico 3

Dimensione finestra: 10



Il timer adattativo assume valori esponenziali rispetto alla perdita dei pacchetti, quindi risulta essere notevolmente vantaggioso per valori “piccoli” di questo parametro per poi degradare immediatamente le prestazioni. Va evidenziato il fatto che la natura in locale del progetto rende il timer adattativo meno vantaggioso in quanto le perdite per timeout scaduto sono molto inferiori rispetto a quelle per pacchetto scartato dall’applicazione.

Analisi perdite

Considerando che il parametro di perdita dato all’applicazione è la perdita di un qualsiasi pacchetto , sia esso pacchetto dati o acknowledgement, si può calcolare la probabilità di ritrasmissione.
Per farlo consideriamo la probabilità di pacchetto spedito e riscontrato correttamente, quindi di pacchetto dati spedito e ack spedito:

P(Trasmissione riuscita) = P(invio pacchetto dati) * P(invio ack) = (1 - parametro perdita) ^ 2

Da qui segue che:

P(Trasmissione fallita) = 1 - (1 – parametro perdita) ^ 2

Parametro perdita	Trasmissione pacchetto riuscita	Trasmissione pacchetto fallita
0%	100%	0%
5%	90,25%	9,75%
10%	81%	19%
20%	64%	36%
30%	49%	51%
50%	25%	75%
70%	9%	91%
75%	6,25%	93,75%

Come possiamo vedere dalla tabella il parametro di perdita incrementa molto rapidamente la probabilità di ritrasmissione di un pacchetto; è quindi naturale che il protocollo rallenti sensibilmente in funzione di questo parametro.

Manuale d’uso progetto UDP affidabile

Installazione

1. Decomprimere e spostare nella directory d’installazione il file “progetto.zip”
2. Aprire la shell dei comandi dalla directory del progetto e lanciare i comandi:
   “make server”
   “make client”
3. tutti i sorgenti sono stati compilati il progetto è pronto all’esecuzione

Configurazione

Per prima cosa è possibile cambiare la posizione e il nome delle directory di default per i file del server e del client (quelle di default si chiamano “client_file” e “server_file”).

Ogni campo viene identificato da una parola chiave seguita dal valore , è possibile modificare solo quest’ultimo , quindi solo la parte compresa tra apici.

La modifica dei campi richiede il riavvio dell’applicazione ma non la ricompilazione.

Server

Una volta aperto con un editor di testo il file “server.conf” verranno visualizzati i seguenti campi:

- *SERVER_PORT='5193'* questo campo identifica la porta pubblica utilizzata dal thread principale del server.
  Questo valore deve essere una porta non utilizzata da applicazioni che condividano lo stesso indirizzo ip dell’applicazione (in genere compresa tra 1024 e 65535).
  Deve coincidere col corrispettivo campo nel file di configurazione del client.
- *THREAD_PORT='5194'* questo campo identifica la porta privata utilizzata dai thread secondari del server.
  Questo valore deve essere una porta non utilizzata da applicazioni che condividano lo stesso indirizzo ip dell’applicazione (in genere compresa tra 1024 e 65535).
- *FILE_DIR='./server_file/'* questo campo identifica il percorso della cartella che contiene i file presenti sul server
- *WINDOW_SIZE='30'* questo campo identifica la dimensione in pacchetti della finestra di spedizione/ricezione.
  Deve essere un valore intero positivo.
- *TIMEOUT_TIME='1000'* questo campo identifica il timeout di ritrasmissione di un pacchetto (espresso in microsecondi) in caso di timeout fisso, o il valore iniziale del timeout in caso di timeout adattativo.
- *TIMEOUT_ADAPT='0'* questo campo stabilisce il tipo di timeout utilizzato e può assumere solo i seguenti valori:
  “0” timeout fisso
  “1” timeout adattativo
- *P_LOSS='0'* questo campo identifica la probabilità di perdita di ogni pacchetto (espressa in percentuale).
  Può assumere un valore tra 0 e 100.

Client

Una volta aperto con un editor il file “client.conf” verranno visualizzati i seguenti campi:

- *SERVER_PORT='5193'* questo campo identifica la porta pubblica del server a cui il client invia la richiesta.
  Questo valore deve essere una porta non utilizzata da applicazioni che condividano lo stesso indirizzo ip dell’applicazione (in genere compresa tra 1024 e 65535).
  Deve coincidere col corrispettivo campo nel file di configurazione del server.
- *SERVER_IP='127.0.0.1'* questo campo identifica l’ indirizzo di rete del server a cui il client invia la richiesta. In questo caso abbiamo utilizzato il local host.
- *FILE_DIR='./client_file/'* questo campo identifica il percorso della cartella che contiene i file presenti sul client.

Esecuzione

1. Avviare il server : da shell lanciare il comando “./server.o”; il server rimane in esecuzione finché non viene terminato manualmente (ctrl + c).
2. Avviare il/i client: da shell lanciare il comando “./client.o”; ogni client rimane in attesa di comandi finché non viene chiuso lo standard input (ctrl + d) .
3. Lanciare i comandi desiderati dal client:
   get <nome_file> : avvia il download del file specificato dalla directory del server
   put <nome_file> : avvia l’upload del file specificato sulla directory del server
   list : visualizza l’elenco dei file disponibili al download (quelli presenti nella directory del server).

È possibile lanciare nuovi comandi sul client anche durante l’esecuzione di quelli precedenti.
La chiusura dello standard input del client non comporta l’ arresto delle operazioni pendenti, esse verranno eseguite e solo allora il client terminerà l’esecuzione.
Per forzare la chiusura del client ricorrere a ctrl + c.

Tipologie di errore

Data la natura multithread dell’applicazione ogni errore non ha ripercussioni sulle altre operazioni in corso.

Ambito	Messaggio d’errore	Causa	Possibile soluzione
comando get (lato client)	File not found	Il file richiesto non è presente nella cartella del server	Controllare che il nome del file sia stato digitato correttamente
comando put (lato client)	File cannot be uploaded because another user is already uploading it	È già in corso un upload di un file con lo stesso nome	Attendere che l’upload concorrente venga finito e poi riprovare
comando put (lato client)	File <file_name> not found	Il file che si vuol inviare non è presente nella cartella del client	Controllare che il nome del file sia stato digitato correttamente
Instaurazione connessione (lato server)	Connection aborted in handshake: too many packets lost	Sono falliti troppi tentativi di connessione consecutivamente	Provare a ridurre la probabilità di perdita dei pacchetti

Invio file (lato server o client)	Connection aborted in file transmission too many packets lost	Sono falliti troppi invii dei pacchetti consecutivamente	Provare a ridurre la probabilità di perdita dei pacchetti
Ricezione file (lato server o client)	Connection aborted in file reception,timeout expired	È da troppo tempo che non si riceve alcun pacchetto	Provare a ridurre la probabilità di perdita dei pacchetti