

Reliable UDP File Transfer

Cinelli Alessia, Cinfrignini Filippo, Tozzi Andrea

[Introduzione](#)

[Manuale per l'installazione e l'uso](#)

[Configurazione](#)

[Comandi disponibili ed esecuzione](#)

[Manuale per il codice del progetto](#)

[Architettura generale del sistema](#)

[Struttura delle cartelle](#)

[Meccanismi comuni ad entrambi i programmi](#)

[Timer adattivo](#)

[Sliding Window](#)

[Acknowledgement](#)

[Threeway handshake](#)

[Codice del Make](#)

[Approfondimento sulle singole funzioni](#)

[Funzionamento requestControl \(Server\)](#)

[Funzionamento threeWay \(Server\)](#)

[Funzionamento listRequestManager \(Server\)](#)

[Funzionamento gestisciRichiesta \(Client\)](#)

[Funzionamento gestisciAttesa \(Client\)](#)

[Funzionamento getRequestManager \(Server\) e putRequest \(Client\)](#)

[Funzionamento putRequestManager \(Server\) e getRequest \(Client\)](#)

[Funzionamento ackReceiver \(Server, Client\)](#)

[Funzionamento gestisciPacchetto \(Server, Client\)](#)

[Funzionamento gestisciAck \(Server, Client\)](#)

[Funzionamento writeOnFile \(Server,Client\)](#)

[Limitazioni](#)

[Piattaforma di sviluppo e strumenti di testing](#)

[Analisi delle Prove](#)

Introduzione

Progetto reliable UDP file transfer (Client - Server Web Application).

Progetto di Ingegneria di Internet e Web implementato in linguaggio C usando l'API del socket di Berkeley su sistemi operativi Unix. Lo scopo è realizzare un protocollo affidabile senza connessione per il trasferimento dei file basandosi sul protocollo UDP. Vengono sfruttati principalmente i seguenti meccanismi:

1. Sliding window, sfruttato sia sul client che sul server, analogamente a quanto fa TCP nel transport layer, per assicurare il corretto ordine dei pacchetti ricevuti.

2. Meccanismo di Acknowledgement (Selective Repeat), per assicurare che i pacchetti inviati non vengano mai smarriti.

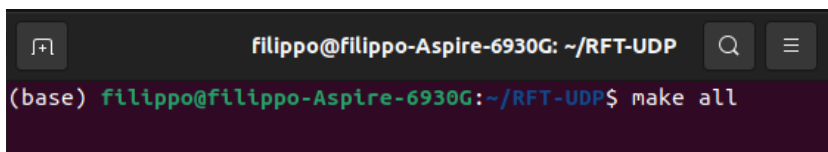
(Questi meccanismi vengono successivamente analizzati in maggiore dettaglio).

Il codice è organizzato in due programmi multi-thread client e server, i quali sfruttano entrambi funzioni aggiuntive esterne tramite libreria.h.

Manuale per l'installazione e l'uso

Configurazione

Per compilare i due programmi individualmente è necessario aprire due terminali, navigare fino alla cartella che contiene i file sorgenti e poi digitare `make client` nel primo terminale e `make server` nel secondo terminale. È poi possibile lanciare i due programmi digitando `./client <indirizzo ip del server>` e `./server`.



```
filippo@filippo-Aspire-6930G: ~/RFT-UDP
(base) filippo@filippo-Aspire-6930G:~/RFT-UDP$ make all
```

Altre opzioni di compilazione: con `make all` è possibile compilare entrambi i programmi contemporaneamente, con

`make client_print` e `make server_print` è possibile compilare i programmi con la macro PRINT in modo che i programmi stampino a schermo informazioni riguardo la loro esecuzione.

Comandi disponibili ed esecuzione

Sul Server non sono disponibili comandi per l'utente. Sul client, l'utente ha a disposizione i seguenti comandi:

- `get_filename`: permette di scaricare il file "filename" dal server e aggiungerlo nella cartella file del client
- `put_filename`: permette di inserire il file "filename" nella cartella file del server
- `list`: questo comando mostra quali file ha il server nella sua cartella

Per eseguire questi comandi è sufficiente digitarli sul terminale dopo aver lanciato l'applicazione. È possibile scrivere più comandi separati da uno spazio per poi premere invio affinché vengano gestiti contemporaneamente.

Manuale per il codice del progetto

Architettura generale del sistema

Il Server, al momento del primo avvio, crea una socket tramite cui rimane in attesa di nuove richieste. Ad ogni richiesta ricevuta nella socket di ascolto viene creato un thread che lavorerà nella funzione `requestControl`. `requestControl` ha il compito di: creare la socket per

interagire con il Client, verificare la legittimità della richiesta, avviare il thread per la connessione a tre vie (threeWay). Stabilita la connessione, il controllo passa al thread specifico per il comando richiesto dal client: list, get o put. Quest ultimo si occupa di servire il client avvalendosi di altri thread a supporto: gestorePacchetto per inviare i pacchetti e reinviarli alla scadenza del timer, gestoreAck analogo a gestisciPacchetto invia gli ack ma senza il timer per la ritrasmissione. readN e writeN sono thread utili alla lettura e scrittura dei file gestiti dal server e infine ackReceiver che resta in ascolto degli ack e gestisce la sliding window.

Il Client attende costantemente comandi inseriti dall'utente. Ad ogni comando inserito, ne verifica la validità e crea una socket per quel comando. Attiva un thread che si occupa di effettuare il threeway handshake. Stabilita la connessione, il controllo passa al thread specializzato nel gestire il comando richiesto: list, get o put. Questo ultimo thread che si occupa di scrivere e rispondere al Server si avvale di altri thread a supporto: gestorePacchetto si occupa di inviare i pacchetti e tiene conto del timer di ritrasmissione, gestoreAck è un thread analogo che si occupa però degli ack, readN e writeN sono thread utili alla lettura e scrittura dei file gestiti dal server, ackReceiver resta in ascolto degli ack e gestisce la sliding window.

Struttura delle cartelle

Nella cartella Progetto sono raccolti tutti i file relativi al progetto. Ci sono i codici sorgente in C del client e del server, c'è il makefile, c'è l'header file libreria.h, la cartella funzioniAusiliari e ci sono le cartelle Files_Client e Files_Server. Quando si compila con il make, gli eseguibili del client e del server vengono aggiunti nella cartella del progetto. Libreria.h permette di usare le funzioni aggiuntive presenti nella cartella funzioniAusiliari. I file che vengono scambiati tra client e server sono quelli contenuti nelle rispettive cartelle Files_Client e Files_Server.

Meccanismi comuni ad entrambi i programmi

Timer adattivo

Il timer adattivo è gestito, sia per il client che per il server, dal thread time_milli_print. È un thread che viene attivato all'inizio dai rispettivi main e rimane in esecuzione in background durante l'esecuzione. Il meccanismo del timer adattivo è semplice, al momento dell'avvio dei programmi c'è già un timeout base, scaduto il quale, avviene la ritrasmissione del pacchetto. Quando vengono ricevuti un numero di ack pari ai pacchetti inviati, viene stabilito un bonus che riduce il tempo di attesa base. time_milli_print è il thread che assegna il bonus (o il malus in caso di molte perdite) periodicamente sommandolo alla variabile di partenza.

Sliding Window

È un vettore di dimensione fissa (la stessa dimensione per client e per il server) che contiene i pacchetti ricevuti o i pacchetti inviati. Questa window permette di salvare, come un buffer, i pacchetti che arrivano che non necessariamente sono nell'ordine atteso. In questo modo si evita di scartare molti pacchetti solo perché non sono quelli che il programma si aspettava. Sono salvati in un buffer anche i pacchetti inviati, in modo tale che se non si

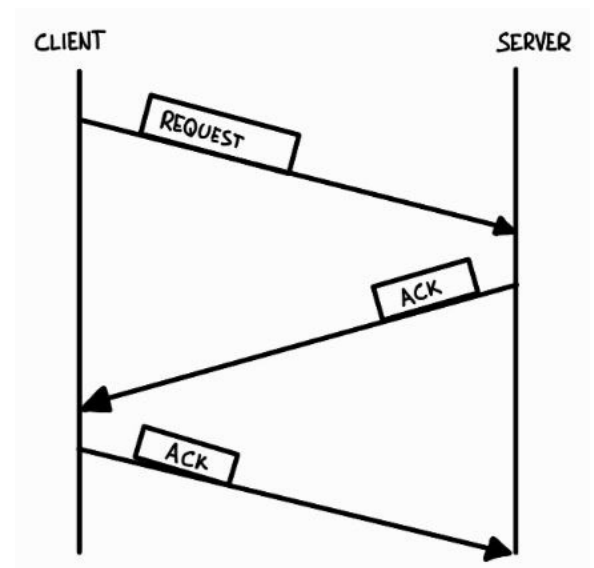
riceve l'ack per il relativo pacchetto, senza rileggere il file, si può inviare il pacchetto salvato nella sliding window. Il meccanismo di "sliding" per la window permette di rimuovere dalla finestra i pacchetti ricevuti che sono già stati copiati nel rispettivo file o i pacchetti inviati di cui si è già ricevuto l'ack, in modo tale da liberare spazio per nuovi pacchetti. La sliding window è stata implementata all'interno del nostro programma con un buffer circolare.

Acknowledgement

Il meccanismo degli acknowledgment è il meccanismo che garantisce che nessun pacchetto venga smarrito in rete. Per ogni pacchetto con una porzione di file inviato, il programma attende dall'altro un ack, ovvero un pacchetto di dimensioni ridotte che conferma la ricezione di quel contenuto. Se il pacchetto viene smarrito o se la conferma di ricezione viene smarrita, viene nuovamente inviato il contenuto. Il contenuto viene inviato fino a quando non si ha la conferma di ricezione. In questo modo è garantito che nessun pacchetto possa venire smarrito lasciando il file trasferito incompleto.

Threeway handshake

Per iniziare un nuovo scambio, server e client si scambiano tre messaggi per assicurarsi che entrambi siano pronti, attivi e sincronizzati su quali messaggi scambiarsi. Il primo messaggio viene inviato dal Client al Server ed è un messaggio contenente la richiesta dell'utente (list, get o put con eventuale nome del file). Il Server risponde al Client con un acknowledgement della richiesta. Il Client conclude l'handshake assicurando il Server di aver ricevuto il suo acknowledgement tramite un ulteriore ack. A questo punto può iniziare lo scambio effettivo dei pacchetti relativi alla richiesta.



Codice del Make

Makefile semplifica alcune azioni che vengono svolte periodicamente. Ad esempio il Makefile permette di: rimuovere col comando `clean` gli eseguibili del client e del server contemporaneamente. Permette poi di compilare i programmi con alcune opzioni: i programmi possono essere compilati insieme o individualmente e possono essere compilati con la macro PRINT attivata o disattivata.

	Insieme	Individualmente
PRINT attiva	<code>make all_print</code>	<code>make client_print, make server_print</code>
PRINT disattivata	<code>make all</code>	<code>make client, make server</code>

Approfondimento sulle singole funzioni

Funzionamento requestControl (Server)

1. Crea una nuova socket per la comunicazione con il client per la singola richiesta, il Client poi, prende (con la `recvFrom`) il nuovo IP/Porta del primo messaggio di ack ricevuto e invierà il restante dei pacchetti al nuovo indirizzo; la socket riservata all'ascolto delle richieste rimane attiva per le nuove richieste
2. Verifica la legittimità della richiesta del client:
 - a. Get: verifica se il file richiesto è presente sul Server
 - b. Put: verifica se non esiste già un file con lo stesso nome sul Server
3. Nel caso in cui le verifiche non vengono superate, viene inviato un Nack al Client e viene chiusa la connessione
4. Se la richiesta è valida, si passa allo stato di "Connessione" tramite la creazione di un nuovo thread

Funzionamento threeWay (Server)

1. Invia un Ack al Client con la socket creata nella funzione descritta sopra, per comunicargli che la sua richiesta è stata accettata
2. Attende la ricezione dell'ultimo Ack (relativo al messaggio di accettazione richiesta) prima di iniziare lo scambio del contenuto.
3. Se l'ultimo ack da parte del Client viene ricevuto, parte un nuovo thread in base alla richiesta effettuata (List, Get, Put)

Funzionamento listRequestManager (Server)

1. Creazione buffer e vettori attui al funzionamento del Selective Repeat
2. Creazione del thread "ackReceiver" che verifica quali pacchetti in volo hanno ricevuto un ack, scorre la Finestra e libera lo spazio in buffer e vettori per i prossimi pacchetti.
3. Entra nella cartella contenente i Files disponibili ai vari Clients
4. Legge uno ad uno i nomi dei Files e li concatena ad una stringa seguiti da "\n"
5. Riempito lo spazio disponibile per il pacchetto, il pacchetto viene inviato con la funzione "gestisciPacchetto" e l'ID relativo salvato in un vettore, per poter essere chiuso alla ricezione dell'ack.
6. Una volta finiti i nomi da poter leggere, all'ultimo pacchetto verrà impostato il terzo bit del primo byte dell'header pari ad 1.

Funzionamento gestisciRichiesta (Client)

1. Controlla che il comando inserito dall'utente sia un comando valido scritto nel formato adeguato
2. Crea il pacchetto con la richiesta da mandare al server
3. Crea la socket con cui scrivere al client

4. Invia il pacchetto e gestisce il threeway handshake
5. Invoca il thread corrispondente alla richiesta inserita dall'utente per continuare a dialogare col server

Funzionamento gestisciAttesa (Client)

Si occupa di chiudere i thread attivi nel momento in cui il threeway handshake non va a buon fine. Si occupa poi di far partire un altro threeway handshake con la stessa richiesta

1. Acquisisci la richiesta inserita nel Client dall'utente
2. Rimani nello stato sleep
3. Se non sono stato terminato da qualche altro thread (perché la threeway è andata a buon fine) termina i thread attivi
4. Fai partire un altro handshake
5. Termina ed esci

Funzionamento getRequestManager (Server) e putRequest (Client)

In questo thread verranno utilizzate le funzioni: `ackReceiver`, `readN`, e `gestisciPacchetto`.

Variabili in comune con `ackReceiver`: `threadIds`, `cong_win`.

Come prima attività viene inizializzata la richiesta, prendendo il nome del file, facendone la open e prendendone le dimensioni totali, e viene attivato il thread `ackReceiver`.

La struttura si basa su due while concatenati: Il while esterno controlla che ci siano pacchetti da inviare (`size > 0`) **oppure** ack da ricevere (`cong_win > 0`). Il while interno controlla se ci sono dei pacchetti da inviare **e** se può inviarli.

While Interno:

Crea il pacchetto da inviare controllando il size rimanente e aggiornando la struttura che verrà passata alla `readN`, di cui verrà fatta la join per creare il buffer da passare a `gestisciPacchetto`.

While Esterno:

Non ha ulteriori compiti, ma "vive" fin quando `ackReceiver` non ha ricevuto tutti gli ack dal client (`cong_win = 0`).

Funzionamento putRequestManager (Server) e getRequest (Client)

Utilizza `writeOnFile`, svolge il suo lavoro basandosi su due variabili, `sequenceNumber` e `rcv_base` (Numero di pacchetto atteso).

Variabili:

1. `payload_fill_vector`, buffer circolare di 0\1 (libero\occupato) relativo a `packet_pointer`
2. `packet_pointer`, buffer circolare contenente i puntatori alle aree di memoria per bufferizzare i pacchetti ricevuti.

Per ogni pacchetto ricevuto, viene creato un pacchetto di ack e vengono distinti 3 casi:

1. Se il pacchetto ricevuto è nella mia finestra di ricezione
 - a. Se posso bufferizzare il pacchetto lo faccio e chiamo restisciAck
 - b. Altrimenti ignoro il pacchetto ricevuto
2. Se il pacchetto ricevuto è nella scorsa finestra di ricezione
 - a. Ho già bufferizzato quel pacchetto, invio solamente un ack
3. Caso ignora (Il sender è davanti al receiver)

Funzionamento ackReceiver (Server, Client)

1. Variabili:
 - a. send_base, (locale) l'indice del primo pacchetto in volo senza ack
 - b. cong_win, (in comune)
 - c. acked[WINDOWSIZE], (locale) buffer circolare di 0/1 per identificare se un pacchetto è stato ricevuto o meno
 - d. threadIds, (in comune) buffer circolare contenente gli id dei thread gestisciPacchetto.
2. Entra in while infinito all'interno del quale, tramite la recvFrom aspetta pacchetti di ack.
3. Se è la prima volta che ricevo un Ack con quel determinato numero di sequenza, chiudo il thread "gestisciPacchetto" relativo.

```
if(threadIds[seqNum % WINDOWSIZE] != 0){
    pthread_cancel(threadIds[seqNum % WINDOWSIZE]); // Cancello il thread se esiste
    acked[seqNum % WINDOWSIZE] = '1';                // Imposto l'ack come ricevuto
    threadIds[seqNum % WINDOWSIZE] = 0;              // Resetto l'id del thread cancellato
}
```

Successivamente viene controllato se l'ack del primo pacchetto in volo non riscontrato per aggiornare send_base e diminuire la finestra di congestione. Il thread verrà chiuso dalle stesse funzioni che lo inizializzano.

Funzionamento gestisciPacchetto (Server, Client)

1. Utilizzando la funzione "scarta", viene deciso se dovrà essere eseguita o meno la sendto del pacchetto contenuto nel buffer.
2. Se il risultato è 0, inviamo il pacchetto.
3. Entra in sleep per timeoutSec
4. Una volta "svegliato" si ritorna al passaggio 1.

La funzione va avanti finché il thread “ackReceiver” non riceve l’ack relativo al giusto numero di sequenza. Dopo la ricezione, il thread gestisciPacchetto viene cancellato.

Funzionamento gestisciAck (Server, Client)

1. Utilizzando la funzione “scarta”, viene deciso se dovrà essere eseguita o meno la sendto del pacchetto contenuto nel buffer.
2. Se il pacchetto di ack è da scartare, il thread termina altrimenti termina dopo la sendto

Funzionamento writeOnFile (Server,Client)

Variabili:

1. rcv_base, (in comune) indica il numero di pacchetto che bisogna scrivere sul file
2. fill_vector, (in comune) da putRequestManager o getRequest

Viene controllato in un while infinito se il pacchetto che voglio scrivere è stato bufferizzato:

1. Viene creato il thread writeN e si aspetta che termini.
2. Impostiamo fill_vector a 0 per indicare che abbiamo scritto il file.
3. Aggiorna rcv_base.
4. Controlla se si è scritto l’ultimo pacchetto così da chiudere il thread.

Limitazioni

Nel software sono presenti bug minori, legati alla gestione di molti thread contemporaneamente. Inoltre impostare il timer per il rinvio dei pacchetti non riscontrati ad un valore troppo basso provoca spesso degli errori indesiderati.

Piattaforma di sviluppo e strumenti di testing

Il codice è stato scritto con Visual Studio Code. Il progetto è stato mantenuto online su GitHub. Abbiamo sfruttato l’integrazione tra la piattaforma di GitHub e l’editor Visual Studio per rendere lo sviluppo del codice collaborativo, mantenendo un controllo sulle versioni del progetto e una copia di backup online in caso di rottura dei computer. La documentazione è stata prodotta usando Google Documents, principalmente per ragioni di facilità d’uso e di possibilità di collaborazione.

Per il testing dei programmi abbiamo usato i terminali delle rispettive macchine e per la verifica delle performance abbiamo usato delle funzioni disponibili in C e il software per l’analisi della rete Wireshark.

Analisi delle Prove

Riportate su github.