INGEGNERIA DI INTERNET E WEB

TRASFERIMENTO FILE SU UDP - TCP

A cura di

ALESSANDRO CALOMINO, ALESSANDRO CHILLOTTI

Indice

1. Introduzione

Lo scopo del progetto è quello di progettare ed implementare in linguaggio C usando l'API del socket di Berkeley un'applicazione client-server per il trasferimento di file che impieghi il servizio di rete senza connessione. Il software deve permettere:

- Connessione client-server senza autenticazione
- La visualizzazione sul client dei file disponibili sul server (comando list)
- Il download di un file dal server (comando get)
- L'upload di un file sul server (comando put)
- Il trasferimento file in modo affidabile.

La comunicazione tra client e server deve avvenire tramite un opportuno protocollo. Il protocollo di comunicazione deve prevedere lo scambio di due tipi di messaggi:

- messaggi di comando: vengono inviati dal client al server per richiedere l'esecuzione delle diverse operazioni
- messaggi di risposta: vengono inviati dal server al client in risposta ad un comando con l'esito dell'operazione.

Il server, di tipo concorrente, deve fornire le seguenti funzionalità:

- L'invio del messaggio di risposta al comando list al client richiedente; il messaggio di risposta contiene la filelist, ovvero la lista dei nomi dei file disponibili per la condivisione;
- L'invio del messaggio di risposta al comando get contenente il file richiesto, se presente, od un opportuno messaggio di errore;
- La ricezione di un messaggio put contenente il file da caricare sul server e l'invio di un messaggio di risposta con l'esito dell'operazione.

I client devono fornire le seguenti funzionalità:

- L'invio del messaggio list per richiedere la lista dei nomi dei file disponibili
- L'invio del messaggio get per ottenere un file
- La ricezione di un file richiesta tramite il messaggio di get o la gestione dell'eventuale errore
- L'invio del messaggio put per effettuare l'upload di un file sul server e la ricezione del messaggio di risposta con l'esito dell'operazione.

2. Architettura del sistema e scelte progettuali

Architettura del sistema

Per lo sviluppo del progetto sono state utilizzati seguenti macchine:

- Thinkpad X200
    - CPU: Intel Core 2 Duo CPU P8400 (2 core)
    - RAM: 3GB
- HP Laptop 15-bs1xx
    - CPU: Intel Core i7-8550U CPU (4 core)
    - RAM: 8 GB

Scelte progettuali

Si è scelto, dovendo progettare un server concorrente, di costruire un'applicazione multithread per garantire flessibilità ed efficienza all'applicazione. Lo schema di funzionamento del server concorrente è il seguente:

- Il mainthread ha il compito di accogliere nuovi client e di stabilire una connessione con essi. Esso, una volta stabilita la connessione, crea un thread figlio che sarà dedicato esclusivamente a quel client.
- Ogni thread figlio ha una propria socket con il quale comunica esclusivamente con il client, gestendone le richieste. Ha inoltre il compito di gestire un'eventuale richiesta di chiusura.

Per ovviare al rischio di avere troppi threads che spesso lavorano poco a causa della scarsa interattività del client è stato attutato il seguente meccanismo: ogni thread, dopo un periodo di tempo che non riceve più richieste dal client con cui comunica, chiude la connessione con esso lasciando spazio per nuove connessioni.

3. Descrizione dell'implementazione

Lo scambio di messaggi avviene usando un servizio di comunicazione non affidabile (UDP), al fine di garantire la corretta spedizione/ricezione dei messaggi e dei file sia i client che il server implementano a livello applicativo il protocollo di comunicazione affidabile di TCP. Di seguito sono riportate le tecniche e le implementazioni sviluppate.

Handshaking a 3 vie

E' stato implementato l'handshaking a 3 vie che permette di risolvere il problema derivante da un handshaking a 2 vie, ovvero avere connessioni stabilite lato server quando non è presente nessun client, infatti è possibile che un client invii un messaggio di richiesta connessione al server(flag SYN = 1), quest'ultimo accetta la richiesta del client(flag SYN=1 , flag ACKbit = 1) ma il client non è presente, lasciando così una connessione pendente al server. L'handshaking a 3 vie richiede al client di inviare al server un riscontro (ACK) del messaggio di accettazione della richiesta di connesione ricevuto dal server, solo quando il server riceve questo riscontro stabilisce la connessione verso il client. Questa fase è stata utilizzata inoltre per permettere al server di inviare al client la porta del thread che servirà le sue richieste. Inoltre, il main thread comunica al client la dimensione della finestra adottata per la comunicazione.

La struttura dei pacchetti

Per garantire la comunicazione affidabile a livello applicativo sono stati creati tre tipi di pacchetti utilizzati a seconda di cosa si sta trasferendo:

- Il pacchetto di tipo dati, contenente i seguenti campi:
    - Il numero di sequenza (il primo byte del campo dati): utile per capire l'ordine dei messaggi
    - Dati
    - La dimensione del campo dati: utile quando il campo dati non è stato utilizzato al pieno (512 byte). Usare la funzione strlen() non avrebbe portato al risultato desiderato quando non si trasmettono array di byte(stringhe) ma si vogliono trasferire file come ad esempio foto, video ecc.
    - Il flag ultimo pacchetto: indica se un pacchetto è l'ultimo
    - Il flag di riscontro: indica se un pacchetto ha ricevuto riscontro
    - Il flag ritrasmesso: indica se un pacchetto è stato ritrasmesso
    - Il numero di ACK: indica fino a quale pacchetto si è ricevuto (ACK N indica che ho ricevuto N-1 byte e mi aspetto il pacchetto con numero di sequenza N)

    - Il flag di validità ACK: indica se il pacchetto contiene un ACK valido
    - Il flag di SYN: utilizzato per stabilire la connessione
    - Il flag di FIN: utilizzato per terminare la connessione

- Il pacchetto di tipo richiesta, contenente i seguenti campi:
    - Il numero di sequenza (il primo byte del campo dati): utile per capire l'ordine dei messaggi
    - Richiesta: contenente la richiesta effettiva
    - Il flag di FIN: utilizzato per terminare la connessione
- Il pacchetto di tipo ACK, contenente i seguenti campi:
    - Il numero di ACK: indica fino a quale pacchetto si è ricevuto (ACK N indica che ho ricevuto N-1 byte e mi aspetto il pacchetto con numero di sequenza N)

Il pacchetto di tipo ACK è stato creato per motivi di efficienza, infatti tutte quelle volte in cui si deve riscontrare un pacchetto senza dover inviare dati, si invia il pacchetto di ACK evitando così di inviare un pacchetto di tipo dati che avrebbe avuto molti campi vuoti peccando così in efficienza.

Il trasferimento dei file

La fase di trasferimento file è divisa in più fasi:

- Una prima fase dove il client invia pacchetti di richiesta (get, put, list) e il server li riceve.
- Una seconda fase in cui il server (se è stata fatta una richiesta di get o list) o il client (se è stata fatta una richiesta di put) calcolano il numero di pacchetti di tipo dati, tramite la funzione prepare_packets, che servono per trasferire i dati necessari a soddisfare la richiesta e invia questo numero calcolato al rispettivo ricevente. Questa fase serve per evitare di sprecare memoria lato receiver, infatti in questo modo il ricevente prima di ricevere il file effettivo riceverà il numero di pacchetti, a questo punto sapendo il numero reale dei pacchetti che riceverà può allocare memoria in modo preciso e senza sprechi.
- Una terza fase in cui avviene il trasferimento in modo affidabile, attraverso il protocollo TCP, realizzato tramite le funzioni sendto_reliable e recvfrom_reliable. Le finestre di spedizione e ricezione del protocollo TCP sono state implementate con un array di dimensione pari alla grandezza della finestra (valore configurabile) e tramite un indice (sendBase o rcvBase) che tengono traccia della reale prima cella della finestra. Una volta che i segmenti

sono stati riscontrati (lato sender) o che i segmenti sono stati bufferizzati in ordine (lato receiver), vengono aggiornati gli indici simulando così la traslazione della finestra. Questa tecnica simula l'esistenza di un array di dimensione "infinite" con dimensioni limitate alla grandezza della finestra configurata.

- Una quarta fase (esclusiva della richiesta put) in cui il server invia l'esito della richiesta put al client.

Timer per i singoli pacchetti

Per poter gestire il timer singolo per ogni pacchetto, con un unico timer hardware, è stata usata la seguente tecnica: è stato allocato un array di timer di dimensioni della finestra sender che ha il compito di tener traccia di quando un pacchetto è stato inviato. L'idea è quella di controllare il timer del pacchetto più vecchio (è stato inviato per prima e quindi il suo timer scadrà prima degli altri). Ogni qualvolta si riscontra uno o più pacchetti, o scade il timer relativo al pacchetto più vecchio si aggiorna l'array di timer in modo da controllare sempre e soltanto un unico timer, quello del pacchetto con timer più piccolo. Si simula così di avere più timer, uno per ogni pacchetto, avendo soltanto un timer fisico.

Gestione accesso concorrente ai file del server

Per sincronizzare l'accesso ai file presenti nel server è stata usata la primitiva di sincronizzazione di read/write lock. Un blocco read/write consente l'accesso simultaneo per operazioni di sola lettura, mentre le operazioni di scrittura richiedono un accesso esclusivo. Ciò significa che più thread possono leggere i dati in parallelo ma è necessario un blocco esclusivo per la scrittura o la modifica dei dati. Per poter permettere che un thread scriva su file e un altro thread legga/scriva su un altro file diverso dal primo senza dover attendere che il primo thread abbia finito di scrivere su quel file è stato necessario collegare la primitiva di sincronizzazione di read/write lock ai singoli files. Per fare ciò è stata progettata una struttura contenente il nome del file e la rispettiva primitiva di sincronizzazione. All'avvio il main thread crea una lista collegata di queste strutture, una per ogni file presente nel server. Per ogni richiesta di get del client, il thread verifica se il lock per quel file è disponibile o meno; per ogni richiesta di put il thread verifica se il file è presente sul server scorrendo la lista collegata create dal main thread, se lo è, verifica se il lock è disponibile o meno, se il file non è presente sul server, il thread inserisce in coda alla lista collegata il nuovo file con il rispettivo lock. L'inserimento nella lista collegata è stato sincronizzato tramite un mutex, evitando così inserimenti nella stessa posizione di due file diversi. L'attuazione di questa tecnica ha permesso di avere lock sui singoli file migliorando l'efficienza delle richieste e garantendo la sincronizzazione in lettura e scrittura, a fronte di utilizzare pochi byte per la creazione di queste strutture.

Terminazione connessione

La terminazione di connessione può avvenire in due modi:

- Il client fa chiusura attiva, ovvero invia una richiesta di fine connessione (flag FIN = 1), il server invierà l'ACK per riscontro il segmento di FIN. Quando il

server è pronto a chiudere la connessione con il client invierà a sua volta una richiesta di fine connessione, il client invierà l'ACK del FIN e si metterà in attesa per un tempo di 4 minuti per riscontrare eventuali ritrasmissioni del server.

- Il server manda un segmento di FIN al client se non riceve da quest'ultimo nessun tipo di richiesta entro un determinato tempo (il valore è stato settato a 1 minuto ma è configurabile a seconda delle esigenze). Questa tecnica evita di avere il server carico di client che non stanno interagendo con esso per troppo tempo consecutivo.

Altre tecniche e meccanismi implementati

- La tecnica del Fast retransmit
- La tecnica del Deleyed Ack
- RTO adattativo TCP
- Bufferizzazione dei segmenti fuori ordine
- Finestre lato sender e receiver

4. Limitazioni riscontrate

Una limitazione è stata riscontrata quando l'ultimo riscontro da parte del ricevitore veniva perso: il ricevitore, quindi, aveva ricevuto tutti i pacchetti e aveva terminato la comunicazione mentre il mittente doveva ricevere l'ultimo riscontro e quindi si metteva in attesa del timer e continuava a trasmettere al suo scadere. Ciò poteva essere risolto mettendo un tempo massimo di comunicazione e far sì che dopo un periodo ti tempo che si stava comunicando, si dava la comunicazione fallita. Questo, però, avrebbe causato che appena si perde l'ultimo riscontro la comunicazione non andava mai a buon fine. Si è preferito supporre che l'ultimo riscontro non venga mai perso.

5. Piattaforma software per lo sviluppo e testing del sistema

Il sistema è stato realizzato sul sistema operativo Ubuntu 18.04.
Per potere compilare ed eseguire il codice è necessario aver installato all'interno della propria piattaforma GNU Compiler Collection (GCC), ovvero una raccolta open source di compilatori e librerie che supportano i linguaggi di programmazione come C e C ++.

6. Esempi di funzionamento

All' avvio dell'applicazione il client contatta il server per cercare di stabilire una connessione con esso.

Potrebbe essere che la connessione non si stabilisca a causa dell'assenza del server. Dopo un timer di cinque secondi (timer configurabile) in cui il client starà tentando di stabilire connessione, esso sarà avvisato da questo messaggio dell'impossibilità di stabilire connessione.

Invece, il secondo motivo, è che il server sia funzionante ma abbia tutte le porte disponibili per la comunicazione occupate. Allora, il client si troverà questo messaggio.

```
Il server non e' in grado di accettare la connessione.
```

Se la connessione al server è avvenuta con successo, l'utente vedrà comparire un messaggio di colore verde che permetterà ad esso di capire di aver stabilito una connessione con il server in modo corretto.
A quel punto, l'utente potrà leggere il menù per capire come poter utilizzare il software.

```
Connessione stabilita con il server.

Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

Assumendo la connessione stabilita, ora verranno mostrati alcuni scenari in base alle richieste dell'utente.
All'invio della richiesta 'list', l'utente potrà vedere il seguente messaggio.

```
Lista file disponibili Server:

•4k_best.jpg
•magia.jpg
•4k.jpg
•finestra.mov
•sonic.png
•tirocinio.pdf


Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

All'invio della richiesta 'put filetest.zip', l'utente potrà vedere il seguente messaggio se l'esito dell'operazione è positivo.

```
Il file 'filetest.zip' e' stato caricato correttamente.

Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

Per completezza, viene mostrato di nuovo il risultato dopo il comando 'list' per vedere l'effettiva aggiunta del file 'filetest.zip'.

```
Lista file disponibili Server:

•4k_best.jpg
•magia.jpg
•4k.jpg
•finestra.mov
•sonic.png
•filetest.zip
•tirocinio.pdf


Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

All'invio della richiesta 'get tirocinio.pdf', l'utente potrà vedere il seguente messaggio se l'esito dell'operazione è positivo.

```
Il file 'tirocinio.pdf' e' stato scaricato correttamente.

Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

Ora, verrà mostrato come reagisce il sistema in caso di input errati ovvero comandi non conosciuti dall'applicativo.

```
L'input inserito non e' corretto


TEMPO DI TRASFERIMENTO FILE: 0 sec 0 usec
Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

Nella seguente immagine viene mostrato come il server, in caso di richiesta di file non in possesso da esso, invii un messaggio al client nel comunica che il file richiesto non è disponibile.

```
Il file 'file_non_esistente' richiesto non e' presente nel server.

Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

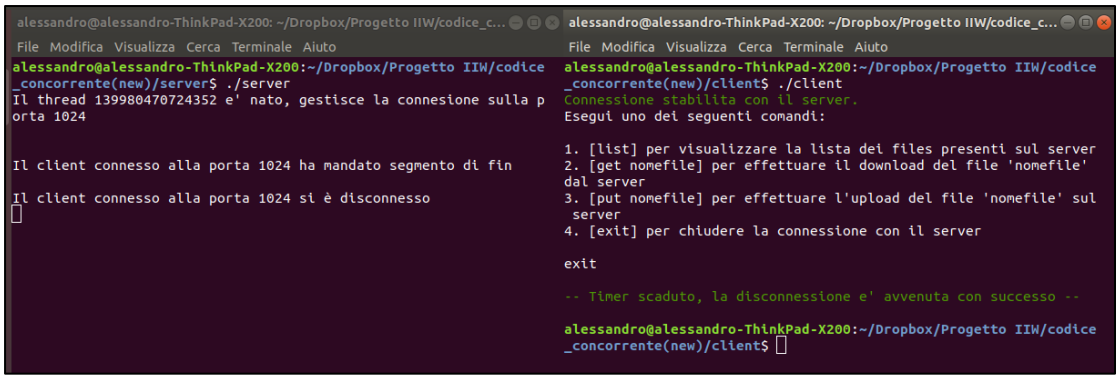
Nella seguente immagine viene mostrato come l'applicativo, in caso di put di file non in possesso dal client, mostri un messaggio al client che comunica che il file che si vuole inserire non è disponibile.

```
Il file 'file_non_esistente' non esiste.

Esegui uno dei seguenti comandi:

1. [list] per visualizzare la lista dei files presenti sul server
2. [get nomefile] per effettuare il download del file 'nomefile' dal server
3. [put nomefile] per effettuare l'upload del file 'nomefile' sul server
4. [exit] per chiudere la connessione con il server
```

Di seguito viene mostrata la scherma che si ottiene con una corretta chiusura della connessione da parte del server e da parte del client (che ha fatto chiusura attiva e deve quindi attendere circa 4 minuti)



Inoltre, il seguente link mostra un video che permette di visualizzare il regolamentare funzione dell'applicazione.
www.dropbox.com/s/74whatnohfpxk6z/progetto%20iiw.mp4?dl=0

7. Valutazioni delle prestazioni del protocollo

Di seguito sono riportati alcuni test sull'efficienza del protocollo, in termini di tempo al variare della dimensione del file, della dimensione della finestra, della probabilità di perdita messaggi e RTO. Denotiamo con N la grandezza della finestra e con P la probabilità di perdita dei messaggi.

Risultati dei test effettuati

Trasferimento file di dimensione 21 MB

N	P	RTO	Tempo 1	Tempo 2	Tempo 3	Tempo medio
4	1	Adattivo	0s 259ms	0s 262ms	0s 198ms	0s 240ms
8	1	Adattivo	0s 154ms	0s 144ms	0s 153ms	0s 150ms
12	1	Adattivo	0s 186ms	0s 152ms	0s 169ms	0s 169ms
4	3	Adattivo	0s 373ms	0s 317ms	0s 412ms	0s 367ms
8	3	Adattivo	0s 266ms	0s 367ms	0s 255ms	0s 296ms
12	3	Adattivo	0s 268ms	0s 203ms	0s 212ms	0s 228ms
4	5	Adattivo	0s 557ms	0s 610ms	0s 553ms	0s 573ms
8	5	Adattivo	0s 355ms	0s 330ms	0s 307ms	0s 331ms

12	5	Adattivo	0s 359ms	0s 339ms	0s 366ms	0s 355ms
4	1	0,1 ms	0s 383ms	0s 247ms	0s 332ms	0s 321ms
8	1	0,1 ms	0s 161ms	0s 158ms	0s 149ms	0s 156ms
12	1	0,1 ms	0s 164ms	0s 230ms	0s 248ms	0s 214ms
4	3	0,1 ms	0s 500ms	0s 544ms	0s 516ms	0s 520ms
8	3	0,1 ms	0s 241ms	0s 237ms	0s 271ms	0s 250ms
12	3	0,1 ms	0s 230ms	0s 252ms	0s 212ms	0s 231ms
4	5	0,1 ms	0s 771ms	0s 810ms	0s 873ms	0s 818ms
8	5	0,1 ms	0s 452ms	0s 436ms	0s 466ms	0s 451ms
12	5	0,1 ms	0s 414ms	0s 415ms	0s 380ms	0s 403ms
4	1	0,2 ms	0s 400ms	0s 479ms	0s 414ms	0s 431ms
8	1	0,2 ms	0s 187ms	0s 163ms	0s 304ms	0s 218ms
12	1	0,2 ms	0s 159ms	0s 168ms	0s 167ms	0s 165ms
4	3	0,2 ms	0s 833ms	1s 111ms	0s 905ms	0s 950ms
8	3	0,2 ms	0s 427ms	0s 418ms	0s 348ms	0s 398ms
12	3	0,2 ms	0s 307ms	0s 307ms	0s 288ms	0s 301ms
4	5	0,2 ms	1s 24ms	1s 577ms	1s 491ms	1s 364ms
8	5	0,2 ms	0s 596ms	0s 670ms	0s 612ms	0s 626ms
12	5	0,2 ms	0s 608ms	0s 618ms	0s 705ms	0s 643ms
4	1	0,5 ms	0s 789ms	0s 814ms	0s 757ms	0s 787ms
8	1	0,5 ms	0s 274ms	0s 235ms	0s 242ms	0s 250ms
12	1	0,5 ms	0s 208ms	0s 218ms	0s 248ms	0s 224ms
4	3	0,5 ms	1s 412ms	1s 377ms	1s 724ms	1s 518ms
8	3	0,5 ms	0s 578ms	0s 584ms	0s 478ms	0s 547ms
12	3	0,5 ms	0s 635ms	0s 462ms	0s 398ms	0s 498ms
4	5	0,5 ms	2s 461ms	2s 709ms	2s 414ms	2s 528ms
8	5	0,5 ms	1s 858ms	1s 636ms	1s 734ms	1s 743ms
12	5	0,5 ms	1s 300ms	1s 449ms	1s 121ms	1s 290ms

Trasferimento file di dimensione 209,7 MB

N	P	RTO	Tempo 1	Tempo 2	Tempo 3	Tempo medio
4	1	Adattivo	1s 932ms	1s 921ms	1s 885ms	1s 913ms
8	1	Adattivo	1s 441ms	1s 442ms	1s 447ms	1s 443ms
12	1	Adattivo	1s 890ms	2s 173ms	2s 318ms	2s 127ms
4	3	Adattivo	4s 957ms	4s 872ms	4s 908ms	4s 912ms
8	3	Adattivo	2s 894ms	3s 582ms	3s 253ms	3s 243ms
12	3	Adattivo	2s 103ms	2s 594ms	3s 567ms	2s 754ms
4	5	Adattivo	7s 14ms	7s 364ms	6s 695ms	7s 24ms
8	5	Adattivo	4s 176ms	4s 657ms	5s 473ms	4s 768ms
12	5	Adattivo	4s 697ms	4s 429ms	4s 960ms	4s 695ms
4	1	0,1 ms	3s 318ms	3s 51ms	3s 796ms	3s 388ms
8	1	0,1 ms	2s 148ms	1s 597ms	2s 764ms	2s 169ms
12	1	0,1 ms	1s 890ms	1s 738ms	2s 180ms	1s 936ms
4	3	0,1 ms	5s 957ms	6s 32ms	6s 908ms	6s 299ms
8	3	0,1 ms	3s 365ms	3s 199ms	2s 518ms	3s 27ms

12	3	0,1 ms	2s 122ms	2s 134ms	2s 84ms	2s 113ms
4	5	0,1 ms	9s 659ms	9s 92ms	9s 695ms	9s 475ms
8	5	0,1 ms	4s 235ms	4s 552ms	4s 468ms	4s 418ms
12	5	0,1 ms	4s 669ms	4s 430ms	4s 800ms	4s 633ms
4	1	0,2 ms	4s 679ms	4s 275ms	4s 409ms	4s 454ms
8	1	0,2 ms	2s 127ms	1s 601ms	3s 160ms	2s 296ms
12	1	0,2 ms	2s 324ms	1s 897ms	2s 591ms	2s 270ms
4	3	0,2 ms	8s 670ms	8s 979ms	8s 827ms	8s 825ms
8	3	0,2 ms	3s 860ms	3s 926ms	4s 131ms	3s 972ms
12	3	0,2 ms	3s 917ms	3s 885ms	3s 758ms	3s 853ms
4	5	0,2 ms	13s 577ms	13s 702ms	13s 208ms	13s 496ms
8	5	0,2 ms	7s 125ms	7s 254ms	7s 58ms	7s 146ms
12	5	0,2 ms	6s 194ms	6s 806ms	6s 64ms	6s 354ms
4	1	0,5 ms	6s 747ms	7s 357ms	7s 52ms	6s 688ms
8	1	0,5 ms	1s 935ms	1s 859ms	1s 876ms	1s 890ms
12	1	0,5 ms	2s 277ms	2s 901ms	1s 792ms	2s 323ms
4	3	0,5 ms	20s 837ms	15s 772ms	15s 217ms	17s 275ms
8	3	0,5 ms	5s 990ms	6s 127ms	5s 832ms	5s 983ms
12	3	0,5 ms	5s 283ms	4s 849ms	5s 591ms	5s 241ms
4	5	0,5 ms	25s 473ms	26s 194ms	26s 114ms	25s 927ms
8	5	0,5 ms	12s 540ms	11s 138ms	14s 878ms	12s 852ms
12	5	0,5 ms	9s 728ms	9s 678ms	9s 945ms	9s 783ms

Commenti dei risultati ottenuti

Dai test effettuati si evince come il timer adattivo permetta la maggior efficienza per il trasferimento file. Infatti, all'aumentare del valore del RTO, i tempi di trasferimento diventano sempre più grandi. Si è deciso di fermarsi ad un timer di 0,5ms poiché l'aumentare di questa quantità avrebbe solamente fatto allungare ancora di più i tempi di trasferimento. Questo è dovuto al fatto che il timer adattivo viene calcolato in base al comportamento che si sta avendo durante la comunicazione mentre fissando timer sempre più alti il valore si allontanerà sempre di più dal valore di timer che effettivamente serve alla comunicazione.
Si vede, invece, come la dimensione della finestra influisca in base ai parametri della comunicazione. Infatti, se la probabilità di pacchetti è bassa la finestra di dimensioni maggiori non porta grandi vantaggi, anzi a volte è addirittura meno efficiente della finestra con dimensione minore: questo è dovuto al fatto che non si ha realmente bisogno di una finestra di maggiori dimensioni e quindi viene influenzato, in negativo, il tempo di trasferimento del file a causa dei tempi maggiori di allocazione di memoria e di elaborazione dei dati al suo interno come ad esempio il calcolo del tempo di trasmissione minimo dei pacchetti all'interno della finestra. Si noti, se la probabilità di perdita è più alta, che una dimensione maggiore della finestra influenza positivamente il tempo di trasferimento poiché in questo caso si ha realmente bisogno di una finestra più grande.
Infine, si può notare come in entrambi i files trasferiti abbiamo avuto sia il miglior tempo (in verde) di trasferimento che il peggior tempo (in rosso) nelle medesime configurazioni di parametri della comunicazione.

8. Manuale per l'installazione e per l'utilizzo del sistema

Manuale per l'installazione

L'installazione del sistema consiste nell'estrarre il contenuto del file zip scaricato.
Una volta estratto, l'utente si troverà due cartelle:

- CLIENT: all'interno ci sarà una cartella FILES che dovrà contenere i files di cui si vuole fare l'upload sul server e sarà la destinazione dei file di cui si fa il download. Inoltre ci sarà l'applicativo che sarà eseguibile digitando da terminale il comando './client'.

- SERVER: all'interno ci sarà una cartella FILES che conterrà i files di cui si vuole fare il download dal server oppure i files di cui è stato fatto l'upload dai client ed inoltre ci sarà l'applicativo che sarà eseguibile digitando da terminale il comando './server N' dove N è la dimensione della finestra di spedizione e ricezione.

L'applicazione è presente in diverse versioni. L'applicazione standard prevede un RTO adattativo. È possibile utilizzare le altre versioni in questo modo compilando il file con i seguenti comandi:

- make: versione standard
- make no_adaptive_rto: versione con il timeout costante
- make test: versione con il timeout costante e test*
- make test_adaptive_rto: versione standard e test*

*La versione test è una versione dell'applicazione che permette di visualizzare l'esito dell'invio di ogni pacchetto, la finestra di spedizione e la finestra di ricezione dopo ogni invio di pacchetto e riscontri, tempi di trasferimento dei file, la concorrenza dei diversi threads, l'evolversi del RTO e altre informazioni relative al funzionamento generale dell'applicazione.

Manuale per l'utilizzo del sistema

Per utilizzare il sistema è necessaria la conoscenza di tre comandi:

- LIST
  - Funzione: visualizzare a schermo la lista dei files che risiedono nel server
  - Sintassi: [list]
- GET
  - Funzione: effettuare il download del file 'filename' nella cartella files
  - Sintassi: [get filename]

- PUT
  - Funzione: effettuare l'upload del file 'filename' sul server
  - Sintassi: [put filename]
- EXIT*
  - Funzione: disconnessione dal server e chiusura dell'applicazione
  - Sintassi: [exit]

*E' possibile svolgere la funzione del comando exit anche attraverso i consueti comandi di terminazione dell'applicazione da terminale (ad esempio Ctrl + C).