Trasferimento file su UDP

Ingegneria di Internet e del Web - A.A. 2018/19

Progetto B

Antonangeli Mattia, Trombetti Lisa

Indice

- Introduzione
- Ciclo client/server
- Trasferimento file affidabile
- Prestazioni

Introduzione

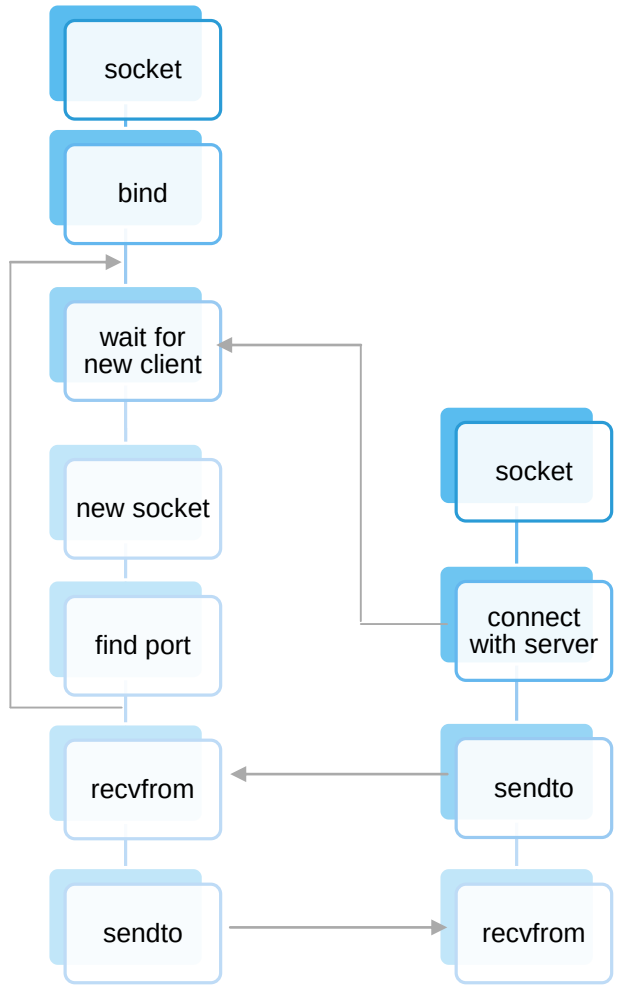
Lo scopo del progetto è quello di progettare e implementare in linguaggio C un'applicazione Client-Server per il trasferimento affidabile di file che impieghi UDP come protocollo di strato di trasporto.

Il sistema deve permettere l'utilizzo di tre comandi principali:

- List: visualizzazione sul client dei file disponibili sul server
- Get: download di un file dal server
- Put: upload di un file sul server

Il sistema è stato sviluppato in ambiente Unix, testato su un dispositivo con processore Intel Core I3-4005U e con SO basato su Ubuntu 18.04.

Ciclo client/server



Apertura della connessione

Parametri della comunicazione:

- numero di porta
- dimensione della finestra di ricezione
- Timeout
- flag del timeout adattivo

Ciclo client/server

La chiusura della connessione può avvenire in 3 modi:

- Input del comando quit da tastiera (q)
- Terminazione tramite interrupt (Control + C)
- Timeout, se non si inserisce un nuovo comando per più di 3 min

Il client richiama la funzione “quit_conn” che si occupa di inviare al server una richiesta di chiusura della comunicazione.

Trasferimento file affidabile

I messaggi che vengono scambiati fra client e server hanno tutti la stessa struttura:

```
struct message{
    char *cmd;      // comando
    char *mess;     // contenuto del messaggio
}
```

Le funzioni utilizzate per scambiare i messaggi sono ‘send_mess’ e ‘recv_mess’.

Trasferimento file affidabile

*int upload_file(int sd, struct sockaddr_in addr, FILE * fd, int N, int start_timeout, int adapt, int dim)*

- Descrittore della socket: sd
- Indirizzo a cui inviare i dati: addr
- Descrittore del file da inviare: fd
- Dimensione della finestra di invio: N
- Timeout iniziale: start_timeout
- Flag per il timeout adattivo: adapt
- Dimensione del file: dim

Trasferimento file affidabile

Implementazione timeout adattivo:

```
struct adaptive_tm{
        int est_rtt;
        int dev_rtt;
};
```

```
int get_timeout(adaptive_tm *atm, int new){
        atm->est_rtt = (1-ALPHA)*(atm->est_rtt) + ALPHA*new;
        atm->dev_rtt = (1-BETA)*(atm->dev_rtt) + BETA*abs(new - atm->est_rtt);
        return (atm->est_rtt + 4*(atm->dev_rtt));
}
```

Trasferimento file affidabile

*int download_file(int sd, struct sockaddr_in addr, FILE *fd, int N)*

- Descrittore della socket: sd
- Indirizzo da cui ricevere i dati: addr
- Descrittore del file su cui scrivere i dati ricevuti: fd
- Dimensione della finestra di ricezione: N

Trasferimento file affidabile

*int listFunc(int sd, struct sockaddr_in client, int N, int start_timeout, int adapt, char *path)*

- Descrittore della socket: sd
- Indirizzo del client: client
- Dimensione della finestra di invio: N
- Timeout iniziale: start_timeout
- Flag per il timeout adattivo: adapt
- Path della cartella di file del server: path

Prestazioni

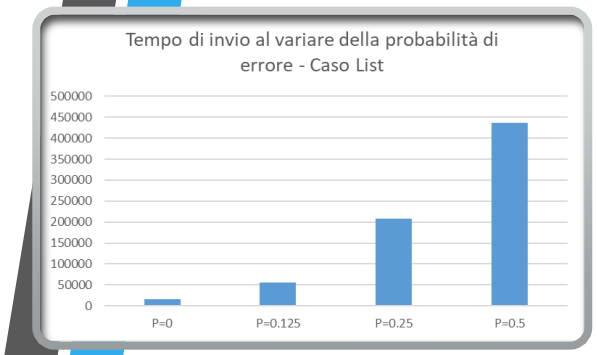
In fase di test è stato utilizzato, per il trasferimento, un file di dimensione 1 MB. I test sono stati effettuati con variazioni di:

- Tipo di timeout (normale / adattivo);
- Timeout iniziale (10000 / 20000 / 30000);
- Probabilità di errore (0 / 0,125 / 0,25 / 0,5);
- Dimensione finestra di ricezione (1 / 3 / 5 / 10 / 20);

Per vedere l'andamento dei tempi di trasmissione e ricezione, sono stati considerati, separatamente, i tempi in caso di:

- Timeout fisso (10000 ms), relativi alla dimensione della finestra N = 10, con tutte le probabilità
- Timeout fisso (10000 ms), relativi alla probabilità P = 0.125, con tutte le dimensioni delle finestre.
- Variazione del timeout in caso di TO fisso e adattivo, con N = 10 e P = 0,125

Prestazioni – List

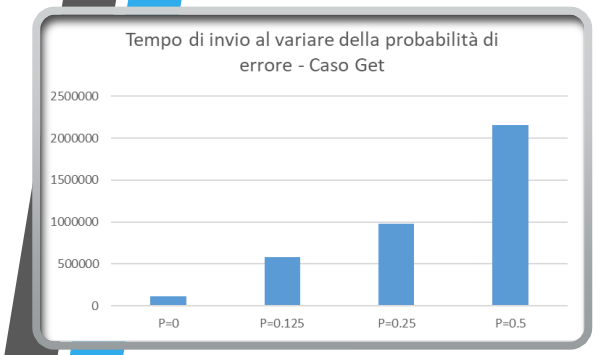


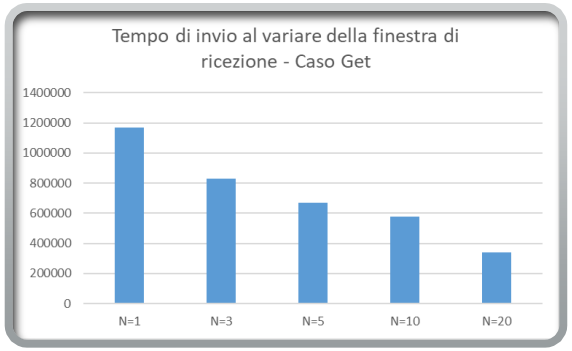
Var. T. Fisso	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	55554,88	185663,2	238771,4

Var. T. Adattivo	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	86959,31	199802,2	311502,3

Il tempo di invio diminuisce all'aumentare della finestra e aumenta all'aumentare della probabilità di errore. In caso di comando 'list', il timeout adattivo impiega più tempo del timeout fisso poiché ha bisogno di inviare diversi messaggi per adattarsi all'effettivo ritardo di rete, e nel caso di list non ne invia abbastanza.

Prestazioni – Get



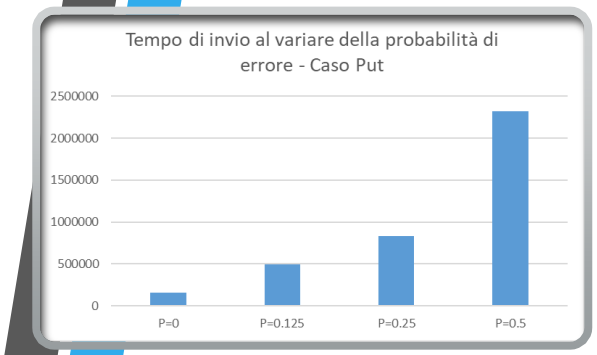


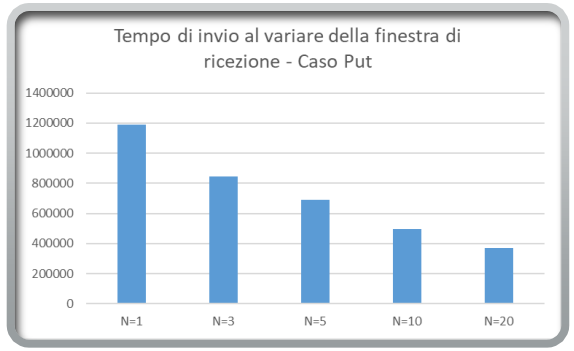
Var. T. Fisso	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	579793,9	889450,9	1265725

Var. T. Adattivo	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	475458,1	464810,1	479795,5

Il tempo di invio diminuisce all'aumentare della finestra e aumenta all'aumentare della probabilità di errore. A differenza del comando precedente, in caso di timeout adattivo il tempo di invio è simile al variare di T.

Prestazioni – Put





Var. T. Fisso	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	496387,4	966866,8	1487836

Var. T. Adattivo	T = 10000	T = 20000	T = 30000
N = 10 P = 0.125	438179,5	496912,7	454558,9

Le considerazioni sono analoghe a quelle fatte in caso di comando Get, poiché i due comandi utilizzano le stesse funzioni.

Prestazioni – VS TCP

Anche nel caso di TCP è stato trasferito un file di dimensione 1 MB. Si può notare dai valori della tabella sopra che il TCP è molto più veloce del Selective Repeat UDP (un ordine di grandezza)

È stato considerato, per il tempo di SR UDP, il test con P=0, N=10, T=10000.

SR UDP	TCP
110879,3	18532,67