

Progetto Ingegneria di Internet e Web

Slide di Presentazione

Protocollo Selective Repeat

Il protocollo Selective Repeat appartiene alla famiglia di protocolli di comunicazione affidabile, che gestiscono la ritrasmissione dei pacchetti, in caso di perdita di questi ultimi. In particolare, nel caso in esame, viene rinviato solo il pacchetto perso. Ogni qual volta una ricezione avvenga con successo, viene inviato un ack di conferma.

Le caratteristiche peculiari di tale protocollo sono riassumibili nella descrizione di:

- Finestra
- Ack
- Numero di sequenza
- Timer
- Probabilità di perdita

Timer

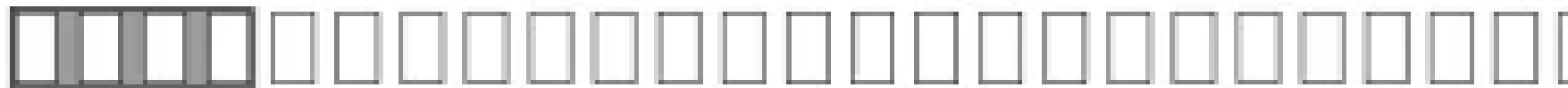
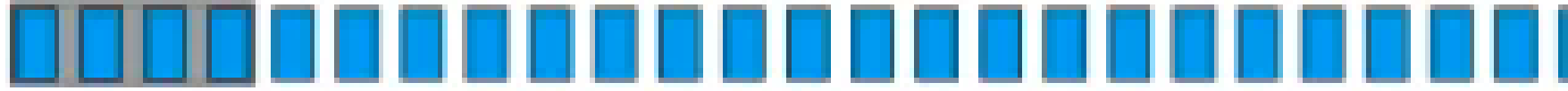
Per la gestione del timeout si è fatto riferimento alle formule:

$$\text{EstimatedRTT} = (1-\alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

$$\text{DevRTT} = (1-2\alpha) \times \text{DevRTT} + 2\alpha \times |\text{SampleRTT} - \text{EstimatedRTT}|$$

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

Simulazione Selective Repeat



I pacchetti e gli ack sono stati incapsulati in due apposite strutture:

```
struct packet{  
    int seq;  
    int ack;  
    int dim_data ;  
    char data[DIM_DATA_BLOK];  
};
```

```
struct ack{  
    int seq;  
    int ack;  
};
```

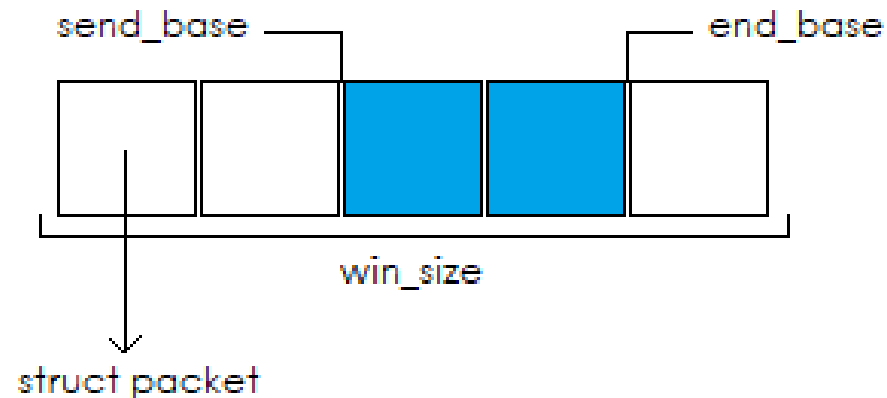
L'implementazione del protocollo Selective Repeat si compone, essenzialmente, di 4 funzioni principali, contenute in SRprotocol.c :

- writeSR
- receiveSRack
- receiveSR
- receive_routine

Ruolo principale: **invio pacchetti, lato mittente, sul socket dedicato.**

Altri compiti svolti:

- I. Creazione del timer del pacchetto;
- II. Gestione della probabilità di perdita di un pacchetto, tramite la funzione *get_loss_probability*;
- III. Gestione della finestra.



Ruolo principale: **gestione ricezione degli ack.**

Altri compiti svolti:

- I. Si valuta se l'ack è riferito ad un pacchetto avente numero di sequenza nell'intervallo accettabile;
- II. Aggiornamento TimeoutInterval;
- III. Gestione della finestra.

Ruolo principale: **ricezione pacchetti sul socket dedicato.**

Altri compiti svolti:

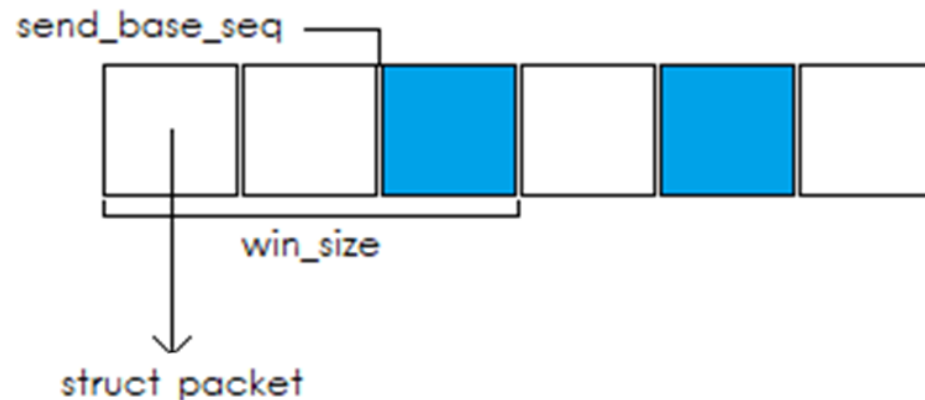
- I. La funzione chiama *la receive_routine*.

receive_routine

Ruolo principale: **scrittura dei pacchetti su file.**

Altri compiti svolti:

- I. Si valuta se la sequenza del pacchetto considerato sia contenuta nell'intervallo accettabile;
- II. Si gestisce l'ultimo pacchetto e la chiusura della connessione;
- III. Invio degli ack sulla socket ad essi dedicata;
- IV. Gestione della finestra.



Probabilità di perdita

Per simulare la perdita dei messaggi in rete è stato implementato un meccanismo apposito.

In particolare, in timer.c è stata creata la funzione:

```
int get_loss_probability(double *loss_p) {  
    double n = (double)rand() / (double)RAND_MAX;  
    if (n < *(loss_p)){  
        return 1;  
    }  
    return 0;  
}
```

I timeout sono necessari per cautelarsi contro la perdita di pacchetti.

Ad ogni trasmissione viene associato un timer e al verificarsi dell'evento di timeout si procede alla ritrasmissione del pacchetto.

All'interno del progetto si distinguono due tipologie di timer:

- Adattivo
- Non adattivo

Nel programma un timer è rappresentato dalla struttura:

```
struct timer {  
    timer_t * id_timer;  
    struct packet* pkt;  
    struct timer* next_timer;  
    struct protocolSR* protocol;  
    pthread_mutex_t* mutex;  
    int arrived;  
    long to;  
};
```

Le funzioni fondamentali alla gestione dei timer, contenute nel file timer.c, sono:

- create_timer
- timer_handler
- to_calculate

Ruolo principale: **associazione di un timer a ciascun pacchetto.**

Passi fondamentali:

- I. Inizializzazione parametri della struttura timer attraverso l'ausilio della funzione *make_timer*;
- II. Creazione timer POSIX;
- III. Inserimento timer all'interno della lista collegata.

Ruolo principale: **rinvio dei pacchetti**.

Passi fondamentali:

- I. Si controlla se l'ack del pacchetto sia stato ricevuto;
- II. Si invia il pacchetto, nel caso in cui questo non sia ancora stato riscontrato.

Ruolo principale: **calcolo dei nuovi valori di timeout.**

Tale funzione viene chiamata soltanto nel caso in cui l'utente scelga di utilizzare timer adattivo, cioè soltanto nel caso in cui il campo flag_timer sia posto a 1.

Per l'esecuzione di tale calcolo si è fatto riferimento alle formule viste nella slide 3.

Gestione concorrenza

Sono stati utilizzati, al fine di gestire la concorrenza, semafori e pthread mutex di tipo POSIX.

In particolare, sono stati utilizzati:

- Un pthread mutex in SRprotocol.c;
- Un semaforo in SRprotocol.c;
- Un pthread mutex in timer.c e linked_list.c.

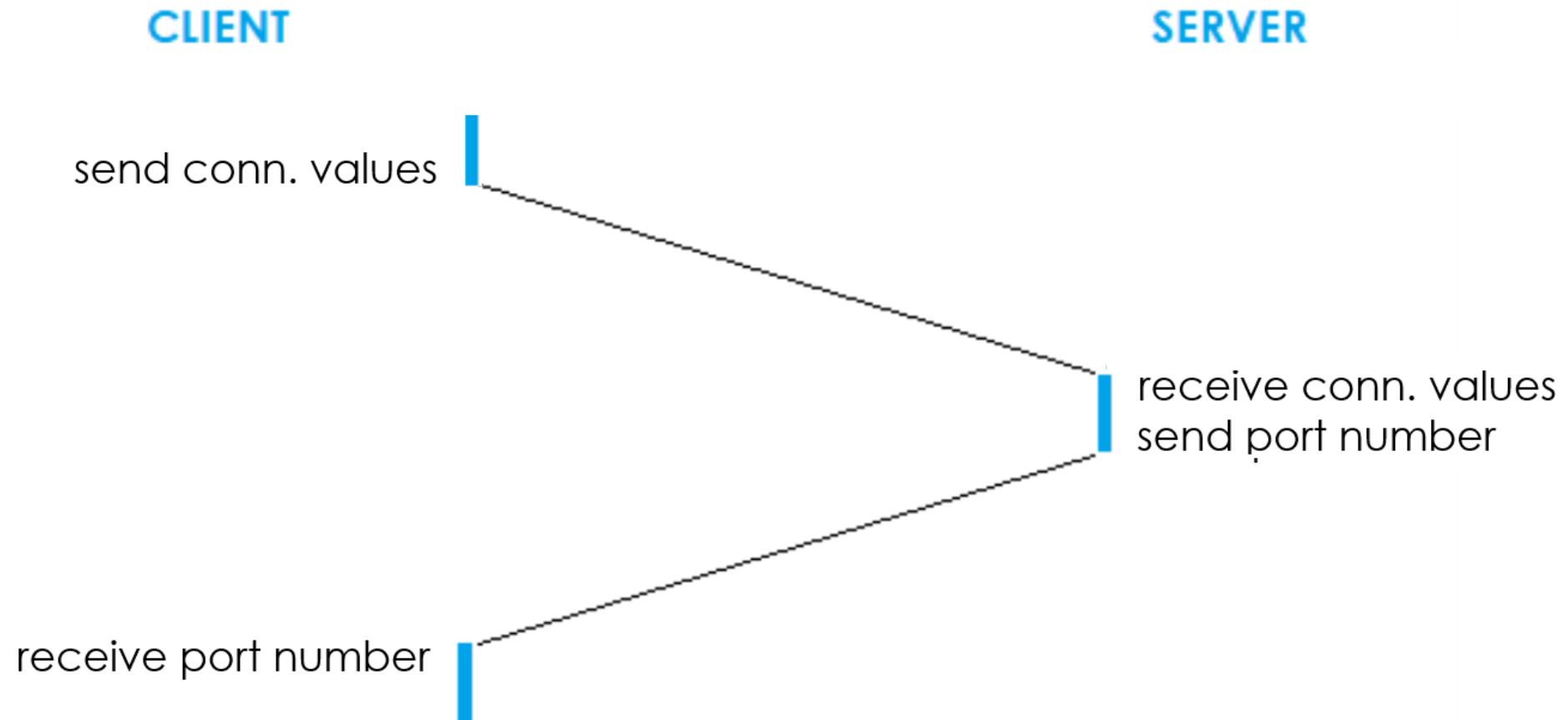
Architettura Client-Server

Client e Server creano, ambedue, un processo figlio tramite `fork()`.

Si occupano principalmente di:

- *Instaurazione della connessione*
- *Gestione dei comandi*

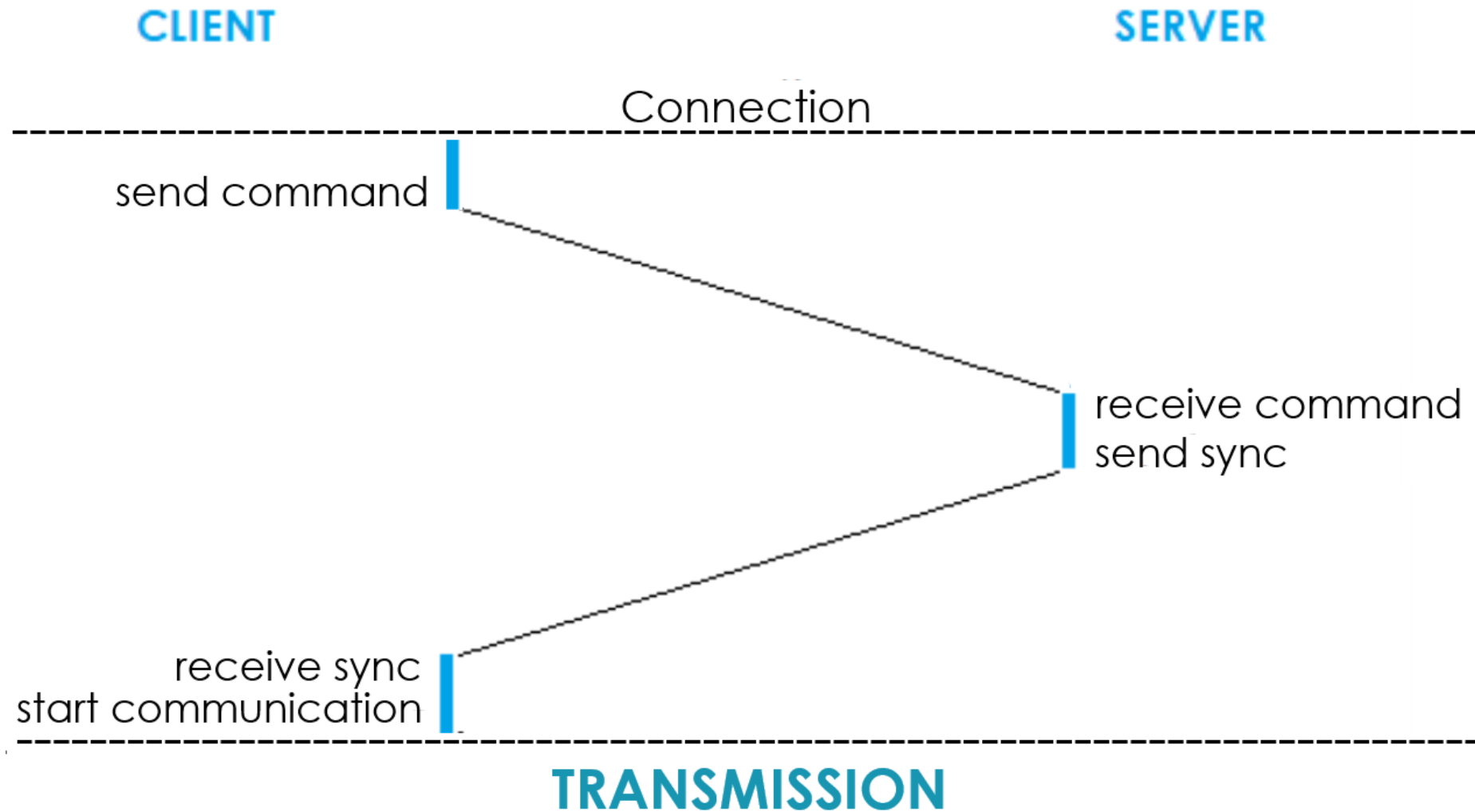
Instaurazione connessione



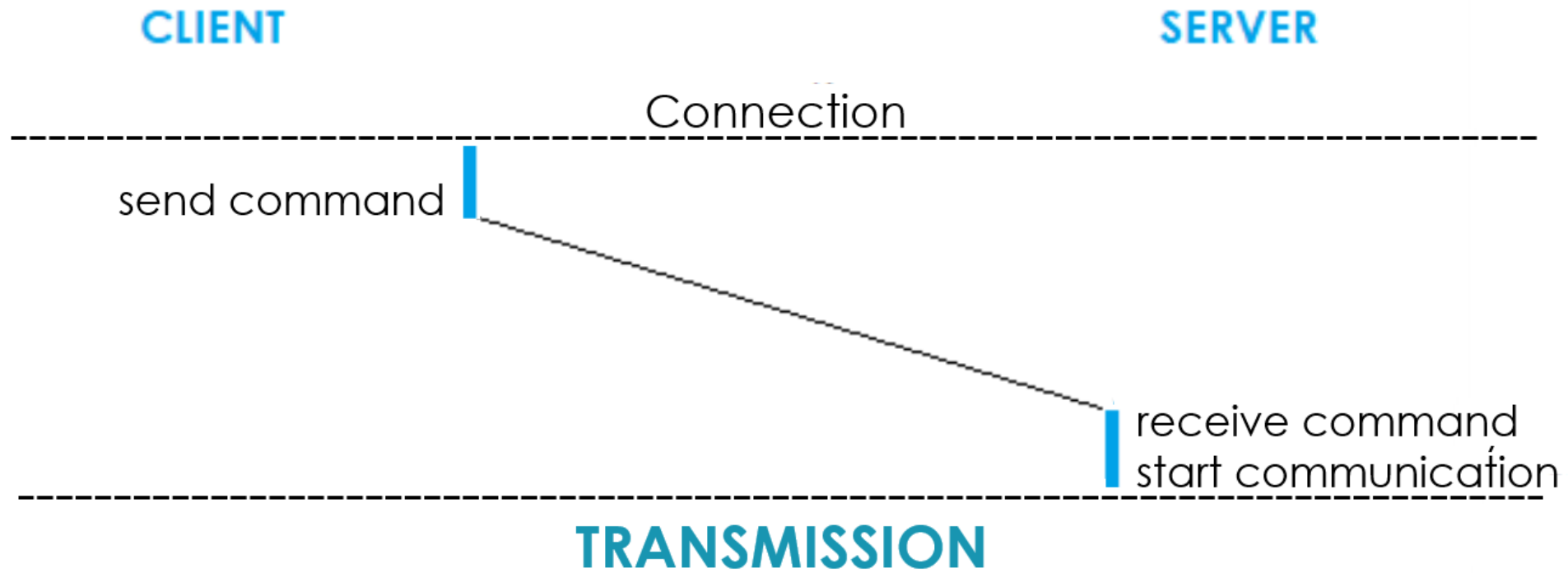
L'utente, una volta connesso, può digitare uno dei seguenti comandi:

- ***put <file>***
- ***get <file>***
- ***list***
- *quit*

put <file>



get <file> & list



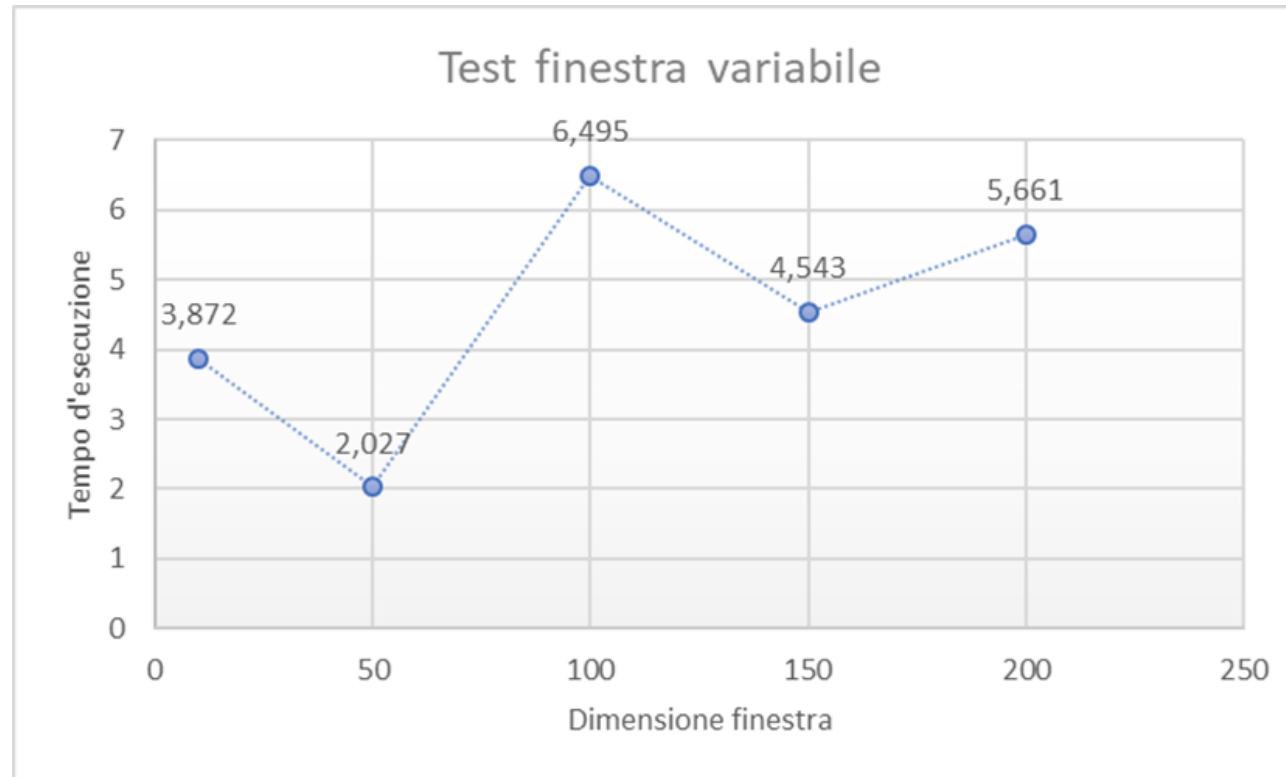
Test prestazionali

I Test eseguiti sono prettamente prestazionali; sono state, infatti, eseguite molteplici prove per misurare il tempo impiegato dal programma per portare a termine le operazioni scelte dall'utente.

Di seguito sono mostrati le tabelle e i grafici ottenuti facendo variare un singolo parametro, mantenendo costanti gli altri, in modo tale da analizzare gli effetti di quest'ultimo sui tempi di trasmissione dei dati.

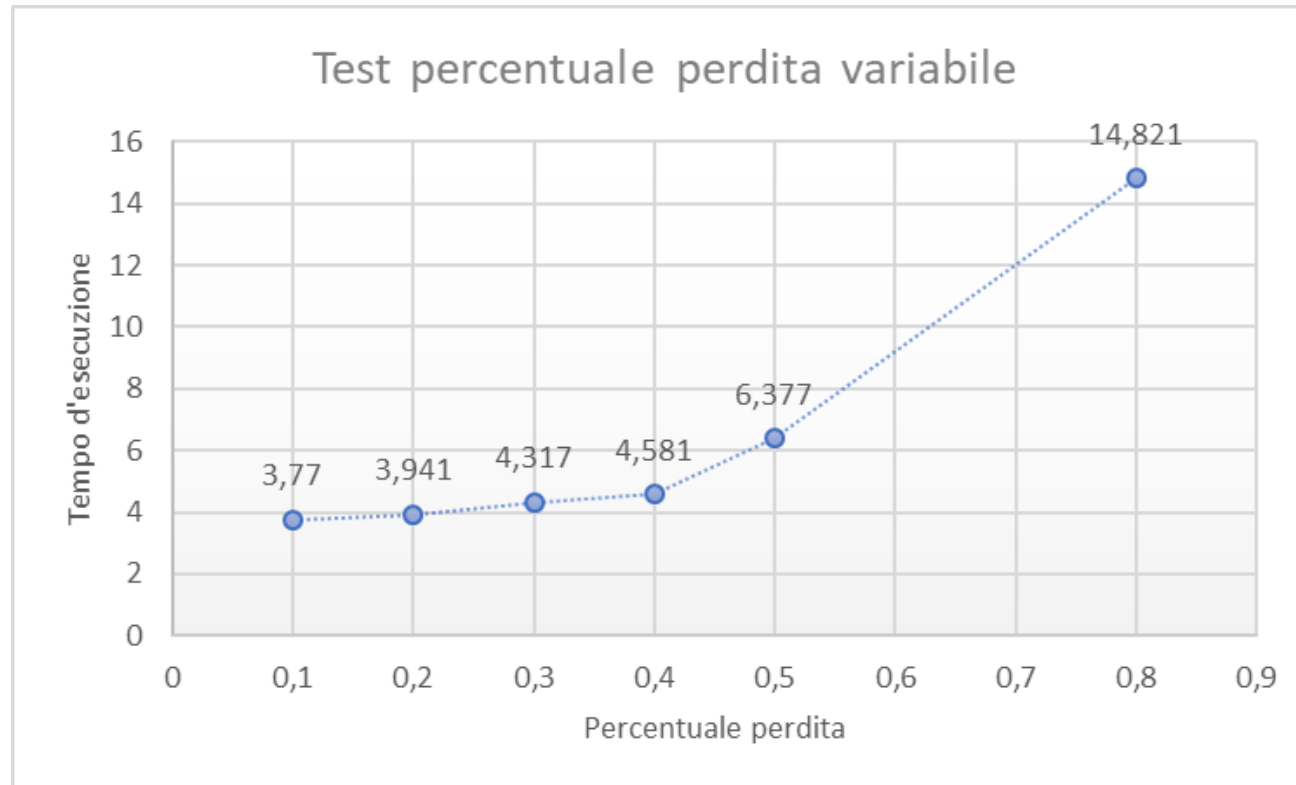
Test 1

Dimensione finestra	Percentuale di perdita del pacchetto	Timer adattivo	Valore timeout (ms)	Dimensione file (MB)	Durata processo (sec)
10	0	1	200	10	3,872
50	0	1	200	10	2,027
100	0	1	200	10	6,495
150	0	1	200	10	4,543
200	0	1	200	10	5,661



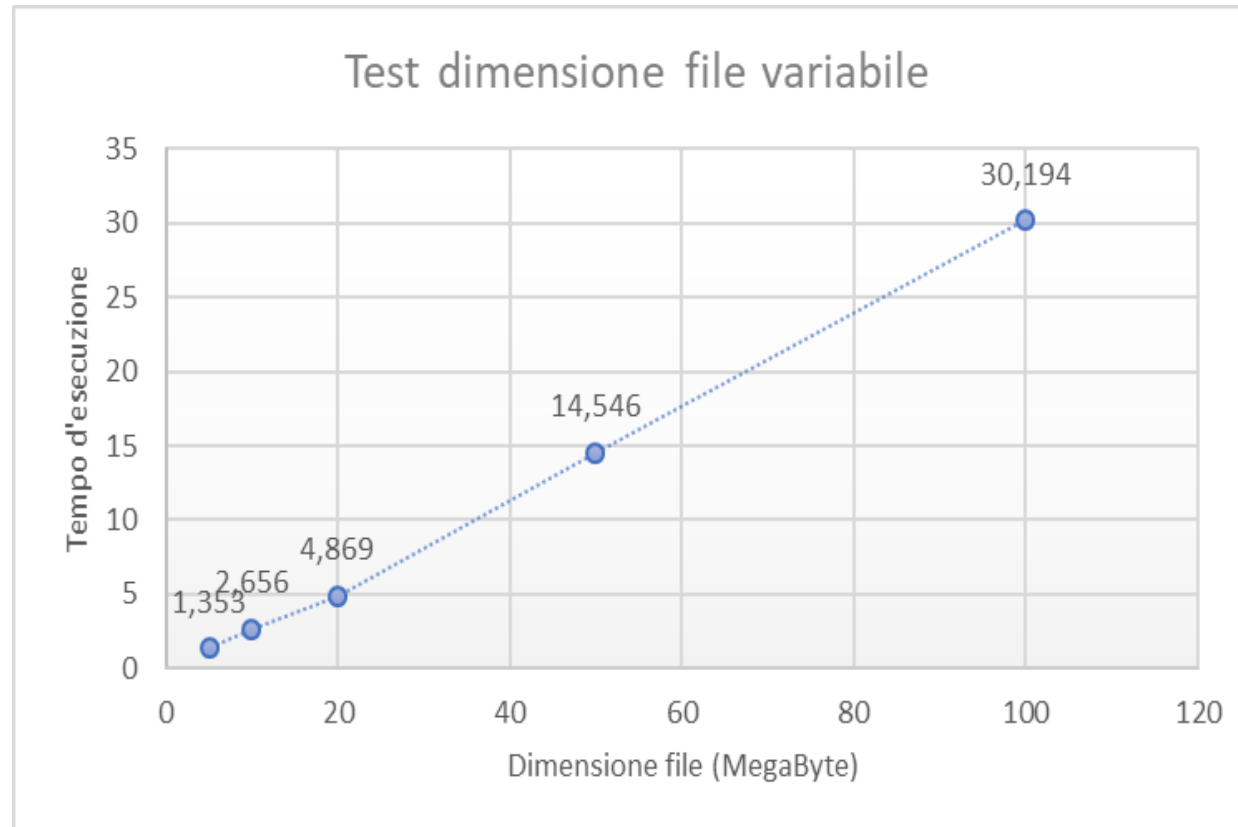
Test 2

Dimensione finestra	Percentuale perdita del pacchetto	Timer adattivo	Valore timeout (ms)	Dimensione file (MB)	Durata processo (sec)
10	0,1	1	200	10	3,770
10	0,2	1	200	10	3,941
10	0,3	1	200	10	4,317
10	0,4	1	200	10	4,581
10	0,5	1	200	10	6,377
10	0,8	1	200	10	14,821



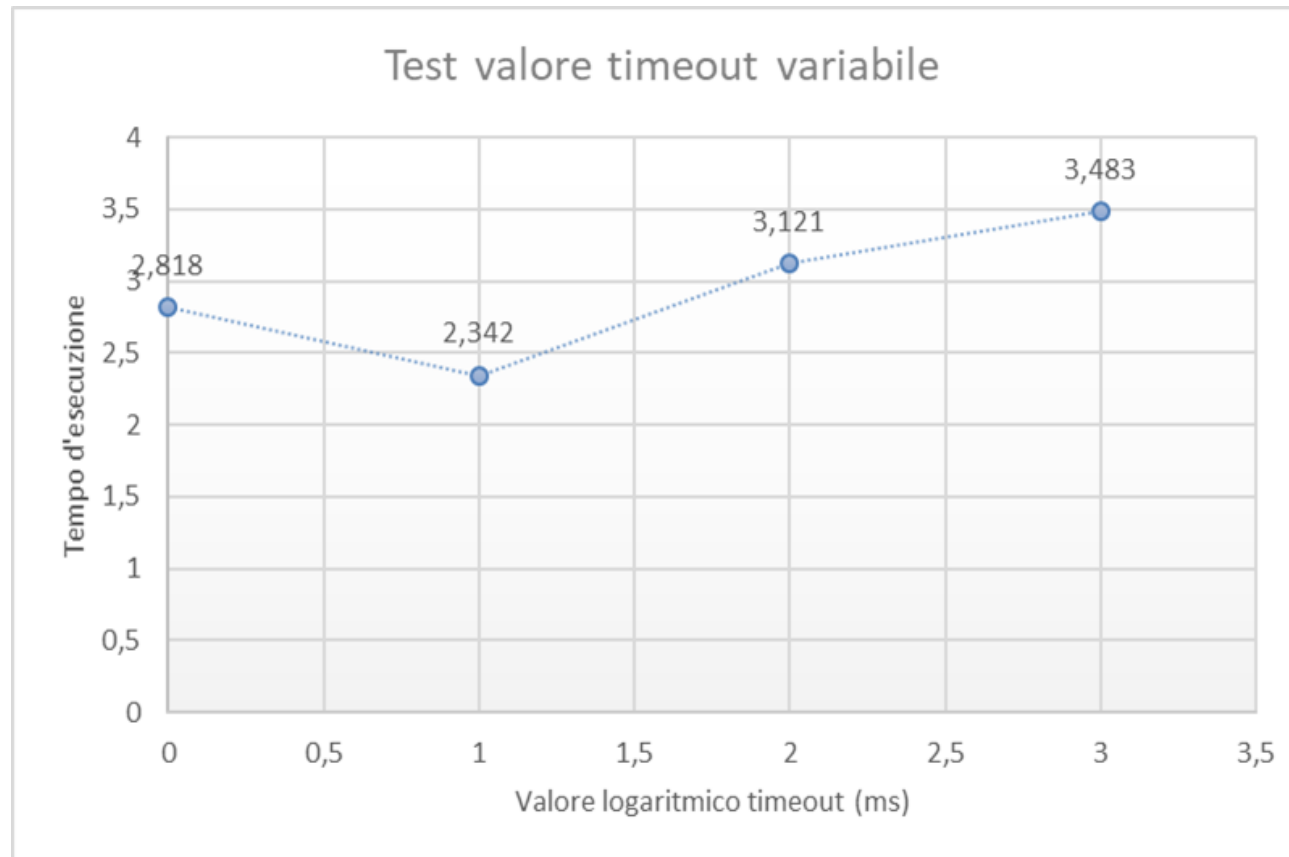
Test 3

Dimensione finestra	Percentuale perdita del pacchetto	Timer adattivo	Valore timeout (ms)	Dimensione file (MB)	Durata processo (sec)
10	0	1	200	5 MB	1,353
10	0	1	200	10 MB	2,656
10	0	1	200	20 MB	4,869
10	0	1	200	50 MB	14,546
10	0	1	200	100 MB	30,194



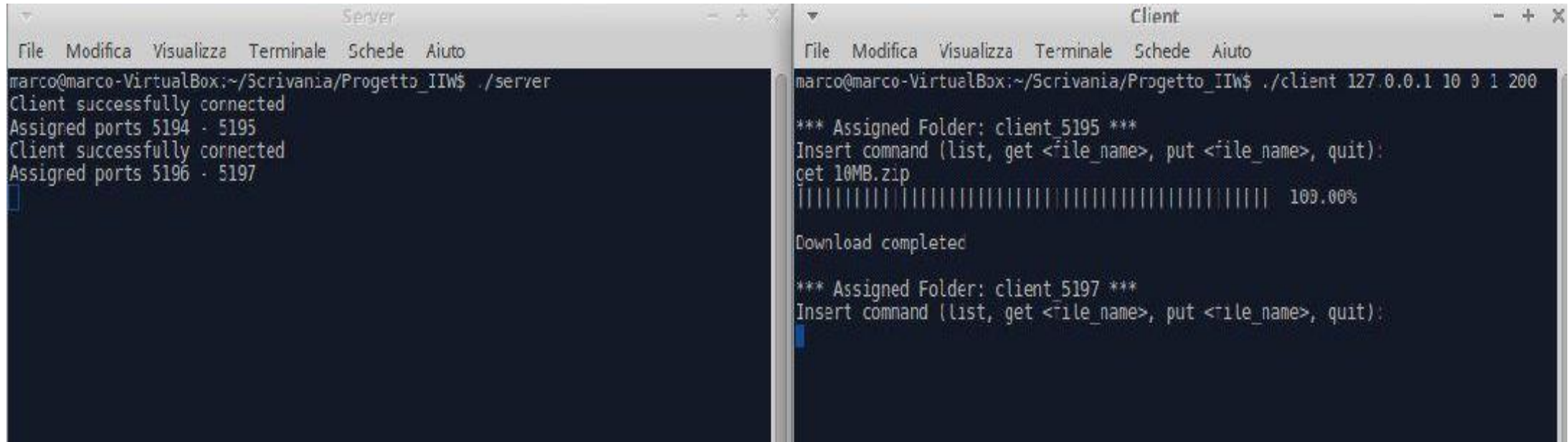
Test 4

Dimensione finestra	Percentuale perdita del pacchetto	Timer adattivo	Valore timeout (ms)	Dimensione file (MB)	Durata processo (sec)
10	0	1	1	10	2,818
10	0	1	10	10	2,342
10	0	1	100	10	3,121
10	0	1	1000	10	3,483



Esempi d'utilizzo

- Utilizzo del comando **get**



The image shows two terminal windows side-by-side. The left window is titled 'Server' and the right window is titled 'Client'. Both windows have a menu bar with 'File', 'Modifica', 'Visualizza', 'Terminale', 'Schede', and 'Aiuto'. The 'Server' window shows the command `./server` being executed, resulting in two successful client connections and assigned port ranges. The 'Client' window shows the command `./client 127.0.0.1 10 0 1 200` being executed, followed by a file transfer of `10MB.zip` using the `get` command. The transfer progress is shown as a bar of 100.00% and the download is completed. The client then shows the next assigned folder and the prompt to insert a command.

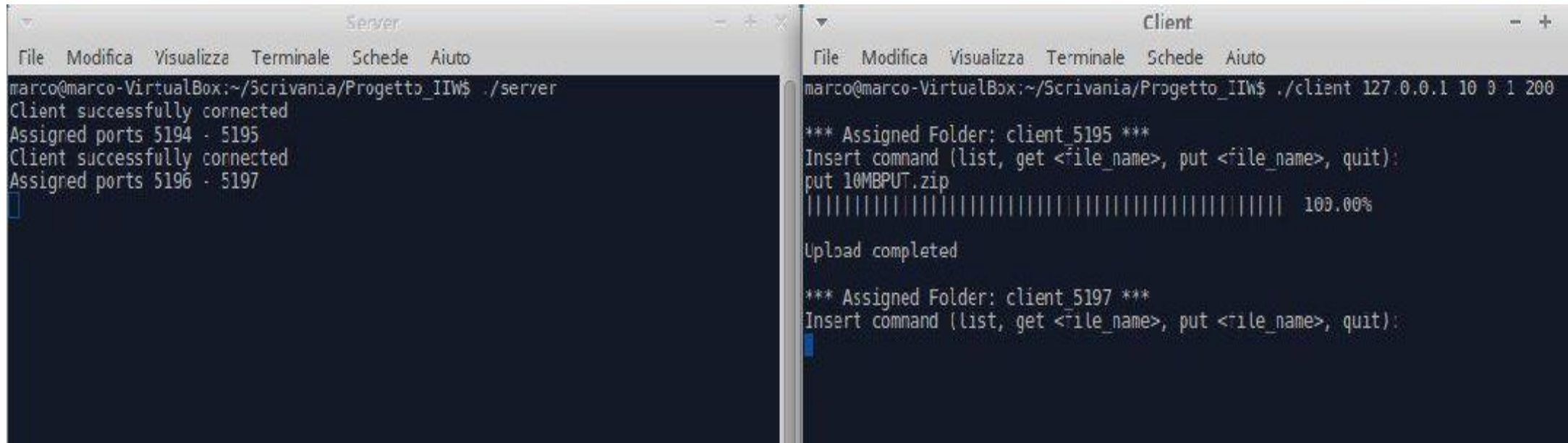
```
marco@marco-VirtualBox:~/Scrivania/Progetto_IIT$ ./server
Client successfully connected
Assigned ports 5194 - 5195
Client successfully connected
Assigned ports 5196 - 5197

marco@marco-VirtualBox:~/Scrivania/Progetto_IIT$ ./client 127.0.0.1 10 0 1 200
*** Assigned Folder: client_5195 ***
Insert command (list, get <file_name>, put <file_name>, quit):
get 10MB.zip
||||| 100.00%

Download completed

*** Assigned Folder: client_5197 ***
Insert command (list, get <file_name>, put <file_name>, quit):
```

- Utilizzo del comando **put**



```
Server
marco@marco-VirtualBox:~/Scrivania/Progetto_IIW$ ./server
Client successfully connected
Assigned ports 5194 - 5195
Client successfully connected
Assigned ports 5196 - 5197

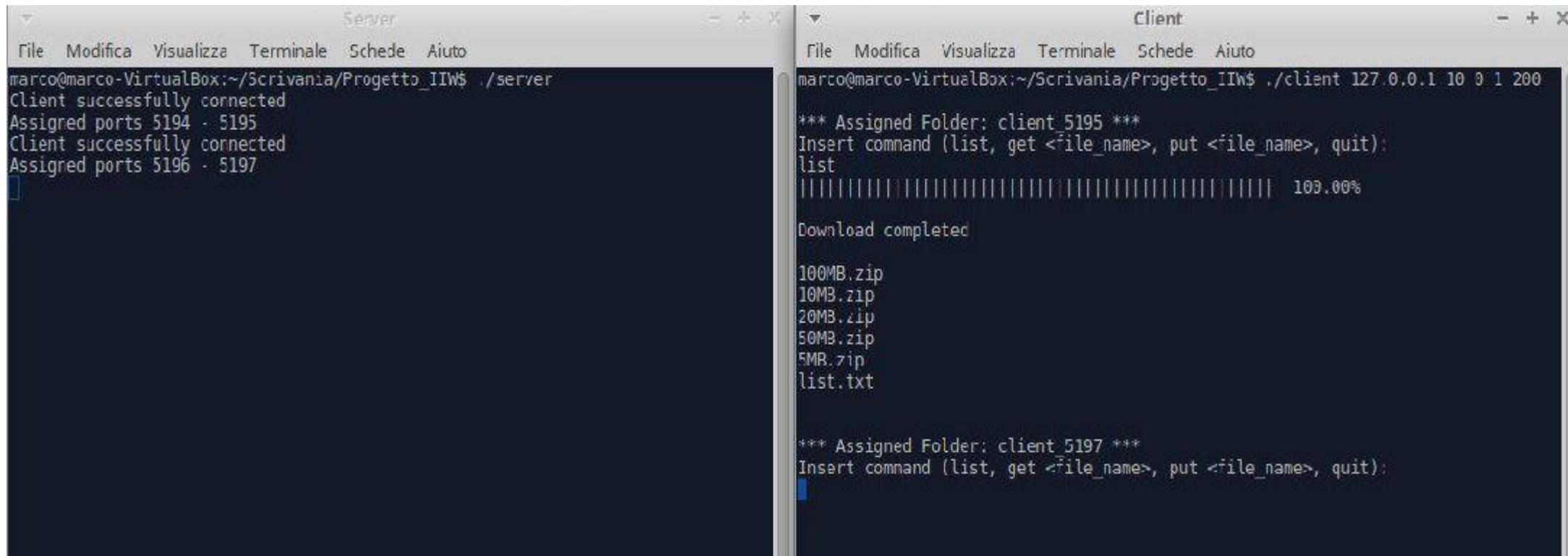
Client
marco@marco-VirtualBox:~/Scrivania/Progetto_IIW$ ./client 127.0.0.1 10 3 1 200

*** Assigned Folder: client 5195 ***
Insert command (list, get <file_name>, put <file_name>, quit):
put 10MBPUT.zip
||||| 100.00%

Upload completed

*** Assigned Folder: client 5197 ***
Insert command (list, get <file_name>, put <file_name>, quit):
```

- Utilizzo del comando ***list***



```
Server
File Modifica Visualizza Terminale Schede Aiuto
marco@marco-VirtualBox:~/Scrivania/Progetto_IIW$ ./server
Client successfully connected
Assigned ports 5194 - 5195
Client successfully connected
Assigned ports 5196 - 5197
█

Client
File Modifica Visualizza Terminale Schede Aiuto
marco@marco-VirtualBox:~/Scrivania/Progetto_IIW$ ./client 127.0.0.1 10 0 1 200

*** Assigned Folder: client 5195 ***
Insert command (list, get <file_name>, put <file_name>, quit):
list
||||| 100.00%

Download completed

100MB.zip
10MB.zip
20MB.zip
50MB.zip
5MB.zip
list.txt

*** Assigned Folder: client 5197 ***
Insert command (list, get <file_name>, put <file_name>, quit):
█
```