



RUA

Reliable UDP based Application

Lorenzo Zara, 0233381

Davide Palleschi, 0232356

INDICE

1. Specifiche del sistema	5
1.1 Formato dei messaggi RUA	5
2. Implementazione di base	8
2.1 Il server	8
2.2 I client	9
2.3 Implementazione delle operazioni	10
3. Implementazione dell'affidabilità	12
3.1 Struttura dei pacchetti	12
3.2 Finestra scorrevole	14
3.3 Invio dei pacchetti	14
3.4 Intervallo di timeout	18
3.5 Perdita dei pacchetti	19
4. Analisi delle prestazioni	20
4.1 Prestazioni al variare della dimensione della finestra di scorrimento (intervallo di timeout fisso)	21
4.2 Prestazioni al variare della probabilità di perdita (intervallo di timeout fisso)	22
4.3 Prestazioni al variare dell'intervallo di timeout (fisso)	23
4.4 Prestazioni al variare della dimensione della finestra (intervallo di timeout adattivo)	24
4.5 Prestazioni al variare della probabilità di perdita (intervallo di timeout adattivo)	25

4.6 Macchine utilizzate	26
5. Manuale	27
5.1 Installazione	27
5.2 Configurazione	28
5.3 Esecuzione	30
5.4 Utilizzo del client RUA	30
6. Esempi	31

1. SPECIFICHE DEL SISTEMA

RUA (Reliable UDP based Application) è un software rivolto al trasferimento di file che impiega UDP come protocollo al livello di trasporto, mantenendo tuttavia una affidabilità nella trasmissione dei dati grazie ad un'opportuna implementazione al livello applicativo.

L'architettura del sistema è di tipo client-server che, grazie ad un particolare protocollo di comunicazione, permette in una connessione senza autenticazione lo scambio di due tipi di messaggi:

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

1.1 FORMATO DEI MESSAGGI RUA

<i>C O M M A N D</i>	<i>put</i>
<i>F I L E N A M E</i>	<i>ReliableUDP1819.pdf</i>
<i>L E N G T H</i>	<i>272553</i>

Tipico messaggio di richiesta RUA.

I messaggi di richiesta sono composti da tre campi. La prima riga è la **riga di comando**, contenente il campo "metodo", e le altre due sono **righe di informazione**, contenenti rispettivamente i campi "nome file" e "lunghezza file".

Il campo "metodo" della prima riga permette di specificare il tipo di operazione che si vuole richiedere al server: tra queste troviamo "list", "get", "put" ed "exit".

Nel caso dell'operazione "list" il client richiede al server di inviargli una lista dei nomi dei file contenuti nella directory "file_server" residente in quest'ultimo (le righe di informazioni non contengono dati in questa operazione).

Per quanto riguarda l'operazione "get" essa contiene dati esclusivamente nel campo "nome file" con il quale il client specifica il nome del file di cui vuole eseguire il download.

Per ultimo troviamo il metodo "put" che richiede il riempimento sia del campo "nome file" che del campo "lunghezza file", con il quale il client specifica la dimensione in byte del file di cui vuole eseguire l'upload.

Un'ulteriore operazione è la "exit" con la quale il client specifica al server la volontà di terminare la connessione.

<i>S T A T U S C O D E</i>	<i>200</i>
<i>F I L E N A M E</i>	<i>ReliableUDP1819.pdf</i>
<i>L E N G T H</i>	<i>272553</i>

Tipico messaggio di risposta RUA.

Come i messaggi di richiesta, anche quelli di risposta sono composti da tre righe contenenti gli stessi tre campi, eccezion fatta per la prima riga, detta **riga di stato**, che contiene il codice di stato con cui il server comunica al client l'esito dell'operazione. Se il codice di stato è 200, il quale identifica il successo dell'operazione, il campo "nome file" conterrà il nome del file coinvolto nell'operazione richiesta dal client e, nel caso del metodo "get", il campo "lunghezza file" conterrà la dimensione in byte del file di cui il client vuole eseguire il download.

Di seguito sono riportati i codici di stato e i rispettivi significati:

- **200: OK** → l'operazione ha avuto esito positivo e il client può usufruire dei dati inviatigli dal server;
- **400: BAD REQUEST** → l'operazione ha avuto esito negativo in quanto il server non è stato in grado di interpretare il messaggio di richiesta;
- **404: NOT FOUND** → il file richiesto dal client non esiste nella directory del server;
- **406: NOT ACCEPTABLE** → la richiesta del client non può essere soddisfatta;
- **503: SERVICE UNAVAILABLE** → il servizio offerto dal server non è più disponibile.

2. IMPLEMENTAZIONE DI BASE

RUA è completamente implementata in linguaggio C ed è strutturata secondo l'architettura client-server, la cui comunicazione è basata sull'API del socket di Berkeley.

Nella figura 2.1 è riportata la creazione, l'assegnazione di indirizzo IP e numero di porta e il binding di una generica socket nella funzione `create_socket` nel file del codice sorgente `RUA/server/src/func.c`.

```
int create_socket(int s_port){
    struct sockaddr_in server_addr;
    // socket creation
    socket_fd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if (socket_fd == -1){
        printf("\nSocket %d creation failed.\n",s_port);
    }else {
        printf("\nSocket %d created succesfully.\n",s_port);
    }

    bzero(&server_addr, sizeof(server_addr));

    // IP address and port number assignment
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(s_port);

    // binding
    if (bind(socket_fd, (SA *) (&server_addr), sizeof(server_addr)) == -1){
        printf("Binding failed x\n");
    }else{
        printf("Binding success v\n");
    }
    return socket_fd;
}
```

Figura 2.1: Creazione di una socket.

2.1 IL SERVER

Il server, di tipo concorrente, è costituito da un processo padre che, all'esecuzione, rimane in attesa perenne delle richieste di connessione sulla socket con porta 1024, scelta non casuale tenendo conto che le porte che vanno da 0 a 1023 sono riservate per altri protocolli applicativi.

Una volta ricevuta una richiesta di connessione da un client, il server parent process si occupa di consultare una bitmap dei numeri di porta, che contiene tanti slot quanto è il valore del massimo numero di clients supportati (configurabile nel file `RUA/server/inc/config.h`). Ottenuto il numero di porta libero, il parent process esegue una fork per delegare a un processo figlio il

compito di sostenere una connessione persistente con il relativo processo client su una determinata porta.

La bitmap è stata realizzata tramite memoria condivisa per rendere noti ad ogni processo l'ottenimento ed il rilascio delle porte (figura 2.2).

```
free_port = (int *) mmap(NULL, MAX_CLIENTS, PROT_READ|PROT_WRITE,  
MAP_ANONYMOUS | MAP_SHARED, 0, 0);
```

Figura 2.2: Realizzazione della memoria condivisa per la bitmap delle porte.

2.2/CLIENT

I client, anch'essi di tipo concorrente, inviano un messaggio di richiesta di connessione al server non appena vengono eseguiti, attendendo, come già indicato, il numero di porta della socket attraverso il quale dovranno instaurare la connessione persistente. Una volta ottenuta la connessione, un generico client offre l'utilizzo delle operazioni "list", "put", "get" ed "exit".

```
Launching RUA client software...  
Initializing environment...  
Trying to connect to server...  
5 seconds left for connection...  
Client connected succesfully!  
  
*** WELCOME TO RUA ***  
With RUA you can download and upload files.  
  
Choose an operation:  
  
- list           [Usage >> list]  
- download      [Usage >> get filename]  
- upload        [Usage >> put filename]  
- exit          [Usage >> exit]
```

Figura 2.3: Interfaccia grafica minimale proposta all'utente.

2.3 IMPLEMENTAZIONE DELLE OPERAZIONI

Dopo l'avvenuta connessione l'utente può procedere con la digitazione di una delle operazioni supportate dall'applicazione. Il modo con cui tali operazioni devono essere digitate è indicato nella stessa interfaccia. Una volta richiesta l'operazione e dopo aver premuto INVIO i caratteri vengono letti tramite una `scanf` e confrontati con le diverse stringhe che rappresentano la corrispondente funzione nel codice sorgente. I messaggi di richiesta e di risposta sono poi formulati a seconda dell'operazione richiesta, mentre l'invio e la ricezione avvengono tramite le funzioni `"sendto"` e `"recvfrom"` attraverso l'opportuna socket.

Il metodo LIST

Digitando `"list"` il processo client formula il messaggio di richiesta inviando al server la stringa `"list"`. Il server, dal canto suo, riceve tale messaggio e formula il messaggio di risposta inviando al client, in caso di esito positivo, il codice di stato `"200"` concatenando un'ulteriore stringa contenente la lista dei file o, nel caso in cui non siano presenti file nella propria directory, esclusivamente il codice di stato `"404"` (`NOT_FOUND`). In entrambi i casi l'interfaccia grafica proposta dal processo client prevede un'illustrazione dei messaggi inviato e ricevuto, stampando la lista dei nomi dei file nel server in caso di esito positivo.

Il metodo GET

Digitando `"get <nomefile>"`, dove `<nomefile>` indica il file di cui si vuole eseguire il download, e premendo INVIO, il processo client formula il messaggio di richiesta contenente il metodo `"get"` ed il nome del file nel relativo campo.

Il server, che riceve il messaggio di richiesta di download del file, lo apre per ottenerne la lunghezza e formula, in caso di esito positivo, il messaggio di risposta `"200 <nomefile> <lunghezzafile>"`. Dopodiché vengono spediti i pacchetti dati al client, la cui dinamica è spiegata nel prossimo capitolo.

Il metodo PUT

Come il metodo `"get"` permette l'invio di pacchetti dati dal server al client, il metodo `"put"` funziona in modo del tutto speculare per il trasferimento di file dal client al server. All'utente è richiesto di digitare in input esclusivamente la

formula “put <nomefile>”, tuttavia il processo client si occupa di aprire una sessione di I/O verso il file tramite l’ottenimento dell’indice di scrittura/lettura immagazzinato nel descrittore del file. La chiamata di funzione lseek(temp_fd, 0, SEEK_END) ritorna il valore della dimensione del file e a questo punto è possibile formulare il messaggio di richiesta “put <nomefile> <lunghezzafile>”.

Il server, il quale riceve tale messaggio, risponderà con un messaggio di risposta con l’esito. Anche in questo caso, se l’esito è positivo, il client procede con il trasferimento del file tramite l’invio di pacchetti.

// metodo EXIT

Il metodo “exit” prevede l’invio di un messaggio al server da parte del client. Esso rappresenta la volontà di terminare la connessione e può avvenire sia in modo esplicito (tramite la digitazione di “exit”) che in modo implicito, cioè alla disconnessione del client. All’interno del processo client il segnale SIGINT (Ctrl + C) è gestito in maniera tale che alla sua ricezione il client stesso proceda con il metodo “exit”, ovvero inviando al server il messaggio “exit”.

3. IMPLEMENTAZIONE DELL’AFFIDABILITÀ

RUA impiega il servizio di rete senza connessione ed utilizza UDP come protocollo di strato di trasporto. Nonostante questo, essa offre comunque un servizio di trasferimento file affidabile tramite l’implementazione, al livello applicativo, del protocollo a finestra scorrevole Selective Repeat. Il vantaggio di tale scelta è dato dal fatto che il protocollo SR, più di altri, offre garanzia di velocità della connessione pur mantenendo il controllo sull’andamento di questa. Di seguito riportiamo le scelte progettuali effettuate per l’implementazione del Selective Repeat. Data la speculare implementazione delle operazioni di download e upload, per il resto del capitolo ci concentreremo sul metodo “put”, e più in particolare sul file RUA/client/src/cmd.c, attraverso il quale il processo client invia pacchetti al processo server.

3.1 STRUTTURA DEI PACCHETTI

Una volta a conoscenza del file di cui eseguire l’upload e della sua dimensione, si procede con lo splitting in pacchetti dati.

Ogni pacchetto è della forma

SEQ (64 bytes)	PAYLOAD (1024 bytes)
----------------	----------------------

dove SEQ rappresenta il numero di sequenza del pacchetto, mentre PAYLOAD sono i bytes in cui il file è stato suddiviso.

Per gestire la trasmissione dei pacchetti è stato definito un nuovo tipo di dato, “packet_man” (packet management), riportato nella figura 3.1.

Questo tipo di dato è una struct che contiene i campi:

- *seq*, per indicare il numero di sequenza del pacchetto;
- *payload*, un buffer che contiene i dati interessati nell’operazione;
- *ack*, un booleano che rappresenta l’avvenuta ricezione dell’acknowledgment per quel pacchetto;
- *ack_checked*, un booleano che rappresenta l’avvenuto controllo del campo *ack* da parte del processo;
- *initial* e *time_start* sono due variabili per il timestamp al momento dell’invio del pacchetto;

- *sampleRTT*, indica l'intervallo di tempo che intercorre tra l'invio di quel pacchetto e la ricezione del relativo ACK.

```
typedef struct _packet_man{
    // sequence number of packet
    int seq;

    char payload[PAYLOAD];

    // buffer that contains seq + payload
    char data[BUFFER_SIZE];

    // boolean [to know if packet has been sent]
    int sent;

    // boolean [to know if packet has been received]
    int ack;

    // boolean [to know if ack has been checked]
    int ack_checked;

    //timestamp
    struct timeval initial;
    long time_start;

    // timer infos
    long sampleRTT;
} packet_man;
```

Figura 3.1: Implementazione della struct per la gestione dei pacchetti.

3.2 LA FINESTRA SCORREVOLE

Si tenga presente che il protocollo a ripetizione selettiva evita le ritrasmissioni non necessarie facendo ritrasmettere al mittente solo quei pacchetti su cui esistono sospetti di errore, rappresentati nel nostro caso dalla simulazione di smarrimento. Questa forma di **ritrasmissione selettiva** costringe il destinatario a mandare acknowledgements specifici per i pacchetti ricevuti. A tal proposito è stata utilizzata una finestra di trasmissione con un'ampiezza configurabile in WINDOW nel file RUA/client/inc/config.h.

Il destinatario invia un riscontro per i pacchetti ricevuti sia in ordine sia fuori sequenza, eccezion fatta per quelli che ricadono al di fuori della finestra di ricezione, la cui dimensione è configurabile in WINDOW nel file RUA/server/inc/config.h. Questi vengono memorizzati in un buffer finché non sono stati ricevuti tutti i pacchetti mancanti (ossia quelli con numeri di sequenza più bassi), momento in cui un blocco di pacchetti può essere analizzato e si possono scrivere sul file i relativi payload in sequenza.

3.3 L'INVIO DEI PACCHETTI

Quando viene a conoscenza della lunghezza del file di cui si vuole eseguire l'upload, il processo client calcola il numero di pacchetti necessari (int num_pkt) per l'operazione ed alloca dinamicamente un array di puntatori a struct per la gestione di un singolo pacchetto, anch'esse allocate dinamicamente. Il motivo di tale allocazione dinamica risiede nel fatto che prima dell'invio di tali pacchetti il processo client, o meglio, il main thread, attiva un thread incaricato di mettersi in attesa degli acknowledgements dei pacchetti inviati. Questo thread, chiamato "ack_receiver", si mette in ricezione di un ACK ogni volta che riesce ad ottenere un token da un semaforo (int semaphore) condiviso col main thread: è quest'ultimo infatti che inserisce un token appena un pacchetto viene inviato.

La procedura di invio dei pacchetti è determinata da una logica che riportiamo di seguito sotto forma di pseudo codice:

```
while(ack_ricevuti < pkt_da_inviare) {
    for(i = base → base + window; i++) {
        if (pacchetto[i] non inviato) {
            sendto(socket, pacchetto[i]);
            pacchetto[i].time_start = time_stamp;
            signal(semaphore, 1);
        }
        if (pacchetto[i] ACKed) {
            if !(pacchetto[i] ACK controllato) {
                pacchetto[i] ACK controllato = 1;
                ack_ricevuti ++;
            }
            if (pacchetto[i].seq = base) base ++;
        } else { //pacchetto[i] ACKed == 0
            if (time_stamp - pacchetto[i].time_start > timeout){
                resendto(socket, pacchetto[i]);
                pacchetto[i].time_start = time_stamp;
            }
        }
    }
}
```

I pacchetti il cui numero di sequenza ricade all'interno della finestra di trasmissione vengono sottoposti ad un controllo ciclico che riguarda l'avvenuta ricezione dell'ACK e del tempo trascorso dall'invio del pacchetto.

Tratteremo nel prossimo paragrafo il calcolo dell'intervallo di timeout.

Vediamo piuttosto come scorre la finestra di trasmissione di dimensione N.

Tenendo conto che lo scorrimento della finestra è determinato dall'incremento del valore "base" (lo scorrimento parte da base = 0 fino a window+base=num_pkt), si ha che:

- i primi N pacchetti vengono inviati consecutivamente;
- la finestra non può scorrere finché l'ACK del pacchetto con numero di sequenza pari all'estremo inferiore della finestra (base) non è stato ricevuto;

- una volta ricevuto l'ACK di tale pacchetto, base viene incrementata di uno e la finestra scorre di un'unità.
- quando il valore $\text{window} + \text{base}$ raggiunge il valore num_pkt la finestra non può più scorrere.

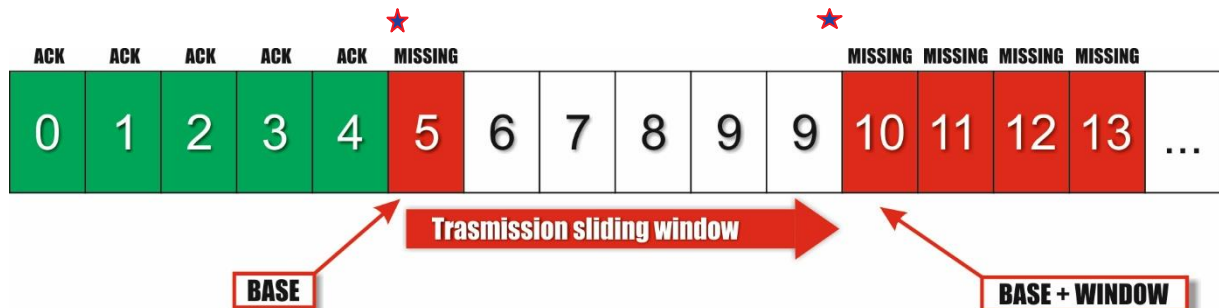


Figura 3.2: Scorrimento della finestra di trasmissione in base agli ack ricevuti.

Dall'altra parte, il ricevente allo stesso modo riceve i pacchetti e ne mantiene i dati all'interno dell'array di struct di gestione del pacchetto, mentre li scarta se:

- il loro numero di sequenza ricade al di fuori della finestra di trasmissione;
- il pacchetto in questione già è stato ricevuto;
- il pacchetto è stato perso (simulazione).

Forniamo di seguito anche lo pseudo codice dell'implementazione della ricezione dei pacchetti:

```
while (pacchetti_ricevuti < pacchetti_da_ricevere) {

rec:   recvfrom(soket, pacchetto);
      if (perdi(pacchetto)) goto rec;
      seq = estrai_sequenza(pacchetto);
      if ( pacchetto[seq] ricevuto) goto rec;    // pacchetto già ricevuto
      if (seq > window + base ) goto rec;        // pacchetto fuori dalla finestra di ricezione

      pacchetti_ricevuti ++;
      payload = estrai_payload(pacchetto);
      pacchetto[seq].payload = payload;
      sendto(socket, seq);                      // invio dell'ACK relativo al pacchetto con sequenza seq

      for(int i=base → base + window; i++){
          if (pacchetto[i] ricevuto AND i = base) {
              write(file, pacchetto[i].payload);
              base ++;
              if (base > pacchetti_da_ricevere – window) window --;
          }
      }
}
```

3.4 INTERVALLO DI TIMEOUT

Come già accennato, appena un pacchetto è inviato dal main thread del processo client, il valore di quell'istante di tempo viene immagazzinato nei campi `initial` e `time_start` attraverso la funzione `gettimeofday()` (figura 3.3).

```
// setting pkt starting time    (measure unit is microsec)
gettimeofday(&(pkts[i] -> initial), NULL);
pkts[i] -> time_start = (((long)((pkts[i] -> initial).tv_sec))*1000000) +
                        (long)((pkts[i] -> initial).tv_usec);
```

Figura 3.3: Time stamp del momento dell'invio del pacchetto.

Il campo `initial` è di tipo `struct timeval` (presente nella libreria `sys/time.h`) che presenta i campi `tv_sec` e `tv_usec`.

Tramite la funzione `gettimeofday(&initial, NULL)` i campi della struct `initial` conterranno i valori dei secondi e dei microsecondi trascorsi dalla mezzanotte del 1° gennaio 1970. Per comodità, in `time_start` sarà salvato quel tempo trascorso in microsecondi. Grazie a questa misurazione sarà possibile confrontare ad ogni controllo ciclico se il tempo trascorso dall'invio del pacchetto supera l'intervallo di timeout senza che sia stato ricevuto un ACK, dovendo così ritrasmettere il pacchetto.

RUA prevede un **intervallo di timeout adattivo** (per renderlo fisso bisogna spostarsi su `RUA/client/inc/config.h` e settare `ADAPTIVE_TIMER` a 0). Ma cos'è l'intervallo di timeout adattivo? Nel caso di intervallo di timeout adattivo, viene effettuata una stima calcolando una media dei valori `sampleRTT`, propri di ogni pacchetto inviato di cui si è ricevuto un ACK, chiamata `estimatedRTT`. Un `sampleRTT` non è altro che il valore dell'intervallo di tempo trascorso fra l'invio del pacchetto e la ricezione del relativo ACK.

Il thread `ack_receiver` si occupa di ottenere il valore dell'istante di tempo in cui l'ACK è stato ricevuto, di calcolare il valore del campione di tempo e di inserirlo nel campo `sampleRTT` della struct di gestione del pacchetto in considerazione.

La media è calcolata secondo la formula:

$$\text{estimatedRTT} = (1 - \alpha) \times \text{estimatedRTT} + \alpha \times \text{sampleRTT}$$

Il nuovo valore di estimatedRTT è una **EWMA (media mobile esponenziale ponderata)** del suo precedente valore e del nuovo valore di sampleRTT.

Tale combinazione attribuisce maggior peso ai campioni recenti rispetto a quelli vecchi. Il valore di default di α è 0.125 ed è configurabile in RUA/client/inc/config.h. Oltre ad avere una stima del Round Trip Time è importante possedere la misura della sua variabilità, che calcoliamo secondo la formula:

$$\text{devRTT} = (1 - \beta) \times \text{devRTT} + \beta \times |\text{sampleRTT} - \text{estimatedRTT}|$$

Il valore di default di β è 0.25 ed è configurabile in RUA/client/inc/config.h. L'intervallo di timeout adattivo di ritrasmissione segue la formula:

$$\text{timeout_interval} = \text{estimatedRTT} + 4 \times \text{devRTT}$$

Quando si verifica un timeout, timeout_interval viene raddoppiato per evitare un timeout prematuro riferito ad un pacchetto successivo per cui si riceverà presto un ACK.

3.5 LA PERDITA DEI PACCHETTI

Per simulare la perdita dei messaggi in rete (evento molto improbabile su rete locale per non parlare di quando client e server sono eseguiti sullo stesso host), si assume che ogni messaggio sia scartato dal ricevente con probabilità p (configurabile in PROBABILITY_LOSS nei file di configurazione config.h).

La funzione utilizzata per implementare lo scenario di perdita è:

```
int prob(int p){  
    return random()%100 < p;  
}
```

Fig. 3.4 – Funzione di probabilità per la simulazione della perdita di pacchetti

4. ANALISI DELLE PRESTAZIONI

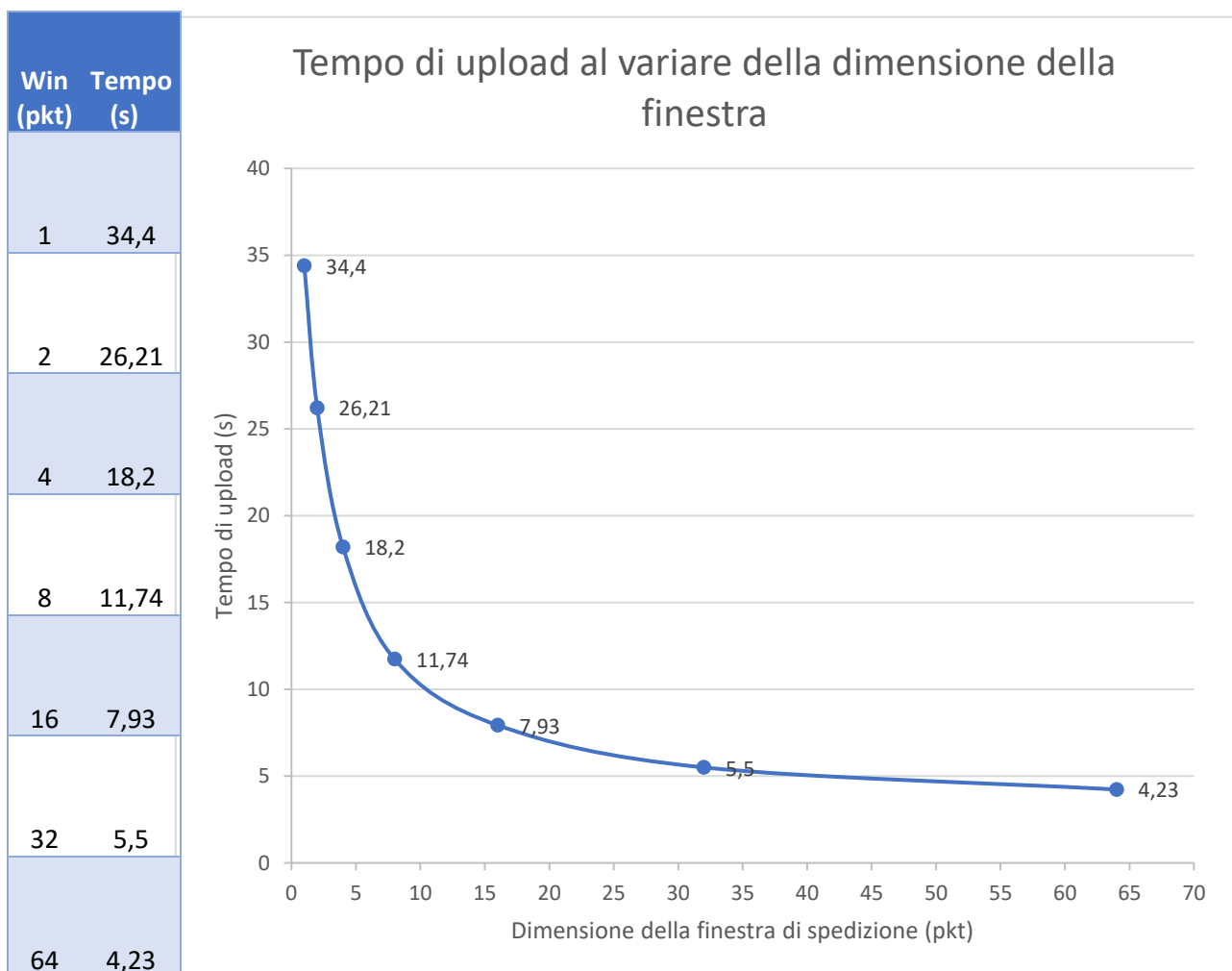
L'analisi delle prestazioni si basa principalmente sulla velocità di upload di un file sul server, dato che utilizziamo lo stesso algoritmo sia per inviare che per ricevere file. Per la fase di testing si è deciso di variare un singolo parametro per volta in modo tale da poter ottenere dei grafici bidimensionali. Inoltre, per evitare che i risultati non fossero falsati in alcun modo, per ogni valore è stata fatta una media dei valori ottenuti ripetendo il test 10 volte.

I parametri che abbiamo fatto variare sono stati:

1. La dimensione della finestra di scorrimento
2. La probabilità di perdita dei messaggi
3. La durata del timeout
4. L'intervallo del timeout (fisso/adattivo)

4.1 PRESTAZIONI AL VARIARE DELLA DIMENSIONE DELLA FINESTRA DI SCORRIMENTO (INTERVALLO DI TIMEOUT FISSO)

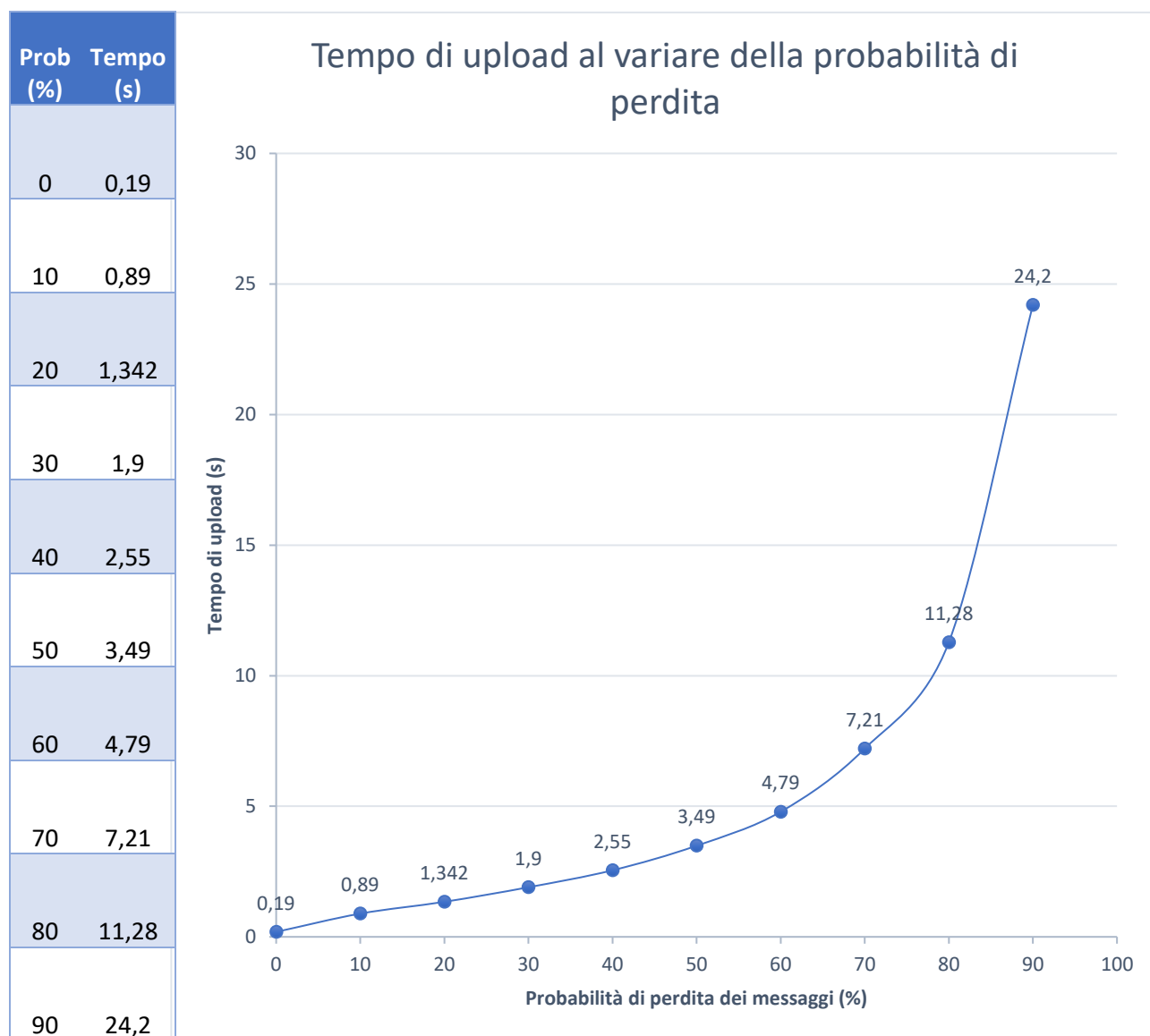
Per questo test è stata fatta variare la dimensione della finestra e sono stati utilizzati valori fissi per gli altri parametri: la probabilità di perdita è stata impostata al 50%, l'intervallo di timeout fisso a 10 ms ed è stato eseguito l'upload di un file di circa 3 MB.



Come è possibile vedere dal grafico, all'aumentare della dimensione della finestra il tempo di upload decresce con andamento logaritmico rendendo di fatto inutile utilizzare una finestra troppo grande.

4.2 PRESTAZIONI AL VARIARE DELLA PROBABILITÀ DI PERDITA (INTERVALLO DI TIMEOUT FISSO)

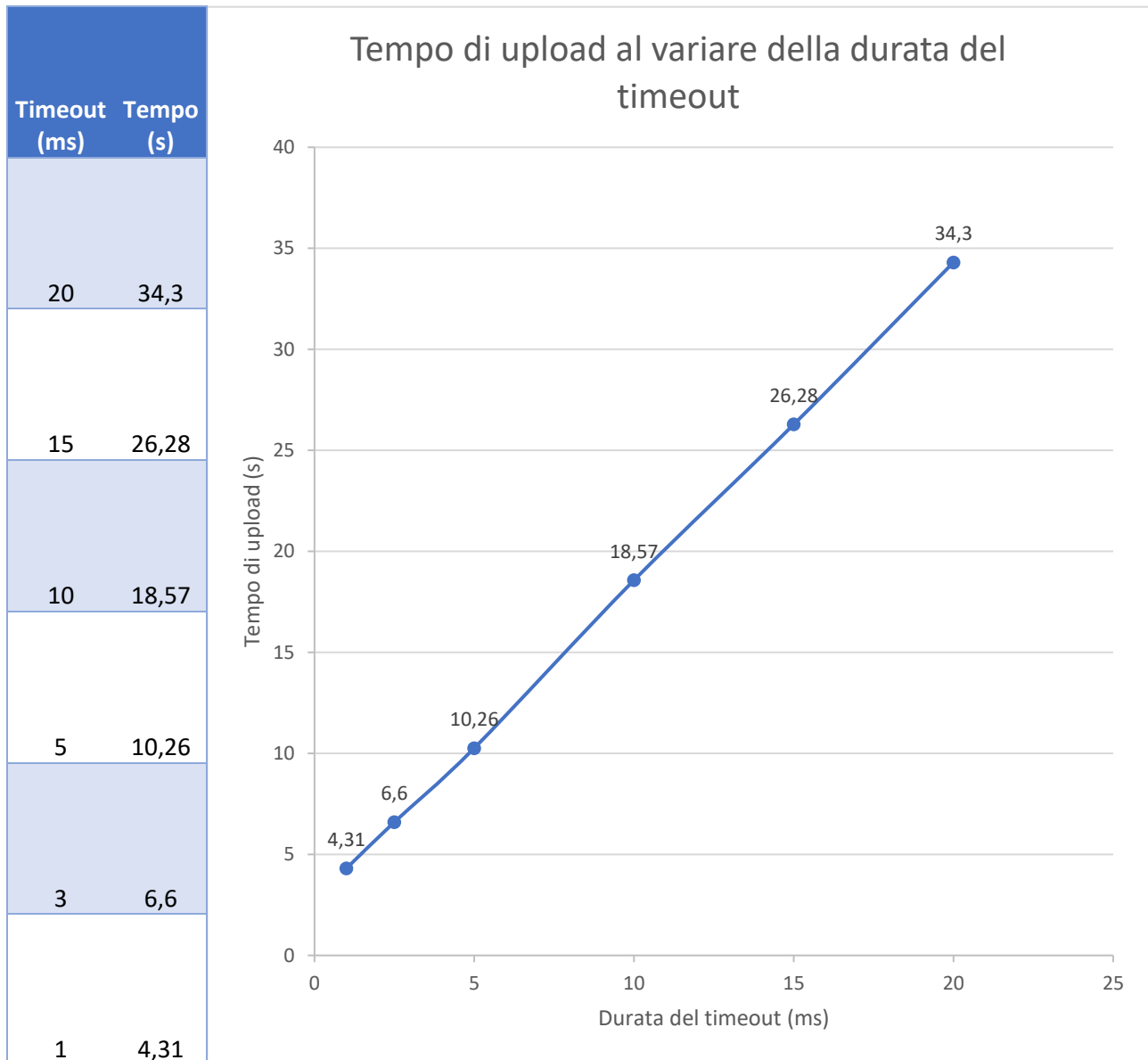
Per questo test è stata fatta variare la probabilità di perdita dei messaggi e sono stati utilizzati valori fissi per gli altri parametri: la dimensione della finestra a 16, l'intervallo di timeout fisso a 5 ms ed è stato eseguito l'upload di un file di circa 3 MB.



Come possiamo vedere nel grafico, all'aumentare della probabilità di perdita dei messaggi il tempo di upload aumenta in modo esponenziale presentando un asintoto verticale in 100.

4.3 PRESTAZIONI AL VARIARE DELL'INTERVALLO DI TIMEOUT (FISSO)

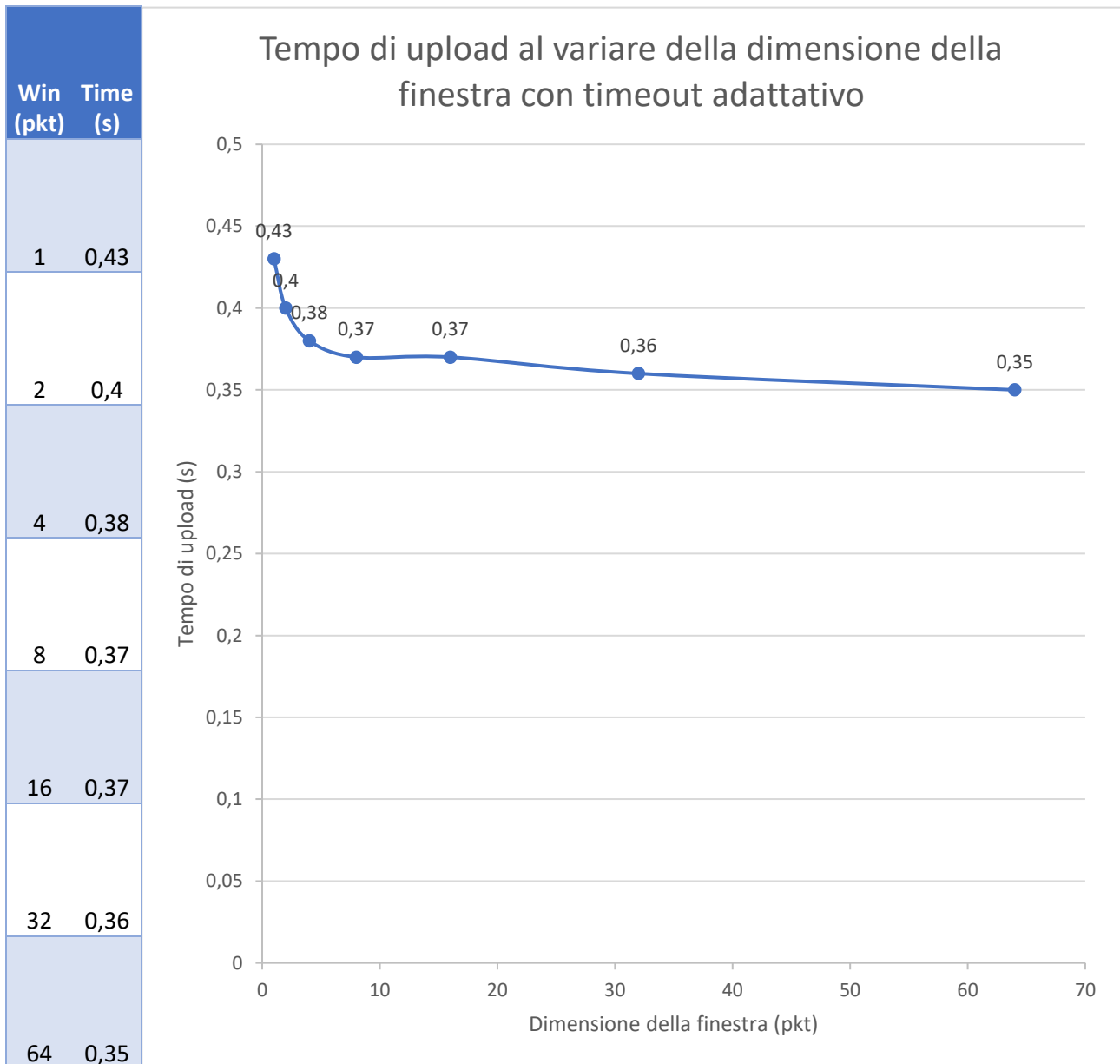
Per questo test è stata fatta variare la durata dell'intervallo di timeout e sono stati utilizzati valori fissi per gli altri parametri: la probabilità di perdita è stata impostata al 50%, la dimensione della finestra a 4 ed è stato eseguito l'upload di un file di circa 3 MB.



Come possiamo vedere nel grafico, all'aumentare della durata del timeout il tempo di upload aumenta in modo lineare.

4.4 PRESTAZIONI AL VARIARE DELLA DIMENSIONE DELLA FINESTRA (INTERVALLO DI TIMEOUT ADATTIVO)

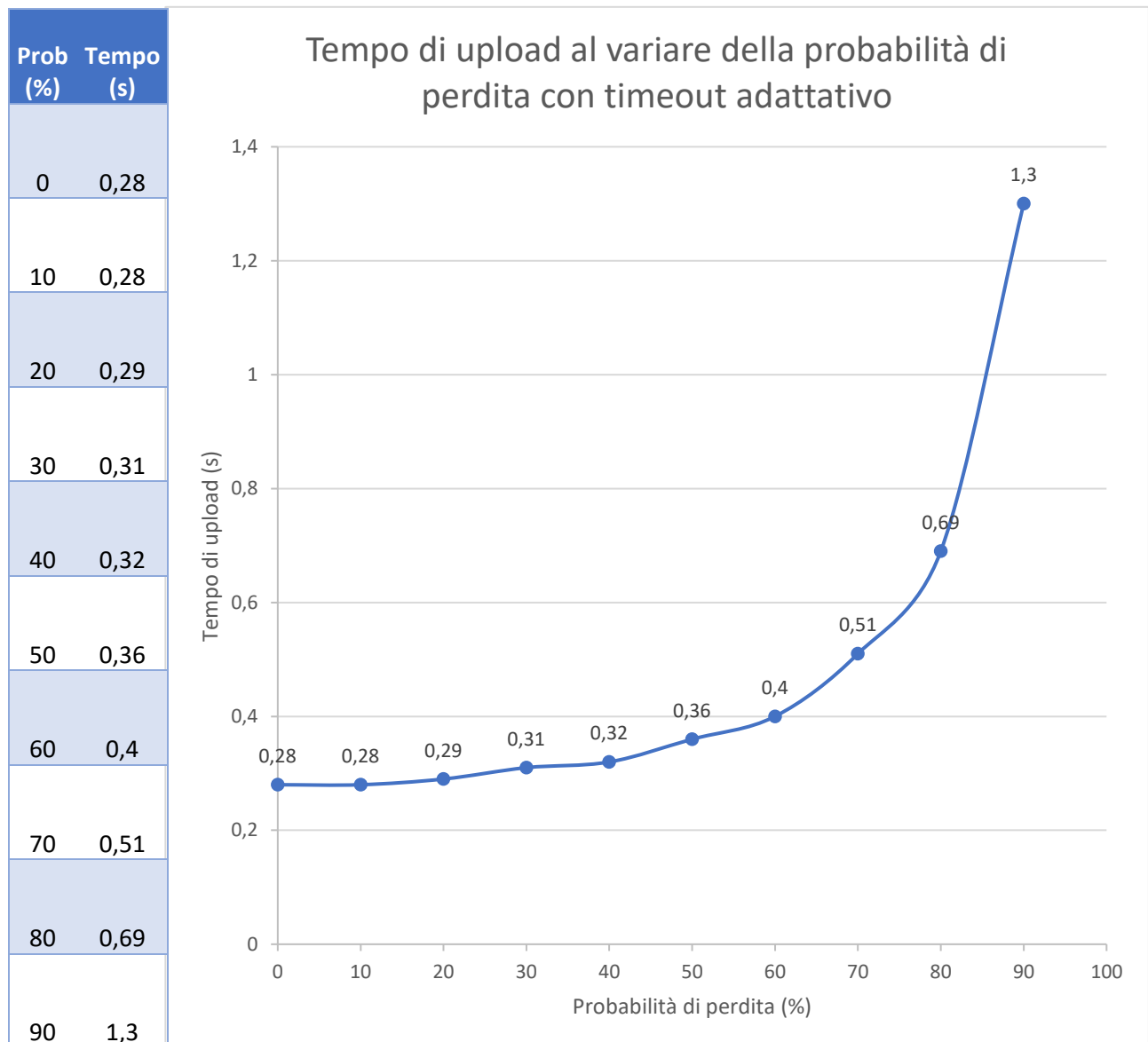
Per questo test, come prima, è stata fatta variare la dimensione della finestra e abbiamo utilizzato valori fissi per gli altri parametri, ma il timeout in questo caso è adattivo. La probabilità è stata impostata al 50% ed è stato eseguito l'upload di un file di circa 3 MB.



Come è possibile vedere il grafico è simile a quello ottenuto con il timeout fisso con la differenza che abbiamo tempi di upload minori e una variazione meno marcata.

4.5 PRESTAZIONI AL VARIARE DELLA PROBABILITÀ DI PERDITA (INTERVALLO DI TIMEOUT ADATTIVO)

Per questo test, come in precedenza, è stata fatta variare la probabilità di perdita dei messaggi e sono stati utilizzati valori fissi per gli altri parametri, ma il timeout in questo caso è adattivo. La dimensione della finestra di spedizione è stata impostata a 16 ed è stato eseguito l'upload di un file di circa 3 MB.



Come è possibile notare il grafico è simile a quello ottenuto con il timeout fisso con la differenza che abbiamo tempi di upload minori e una variazione meno marcata.

4.6 MACCHINE UTILIZZATE

La fase di progettazione e la fase di testing dell'applicazione sono state supportate dall'utilizzo di due macchine e dal software **Visual Studio Code v 1.42.1**.

MACCHINA 1

SISTEMA OPERATIVO: **Ubuntu 18.04.3 LTS @ 64-bit**
PROCESSORE: **Intel® Celeron® CPU N3050 @ 1.60GHz**
MEMORIA: **8 GB**
DISCO: **SSD @ 256 GB**

MACCHINA 2

SISTEMA OPERATIVO: **Windows 10 - macchina virtuale con Ubuntu 18.04.3 LTS @ 64-bit**
PROCESSORE: **Intel® Core i5-6300 2.4 GHz**
MEMORIA: **8 GB**
DISCO: **SSD @ 256 GB**

5. MANUALE

5.1 INSTALLAZIONE

Per l'installazione di RUA seguire le seguenti istruzioni.

1. Aprire il terminale.
2. Posizionarsi nella cartella RUA.

APPLICAZIONE SERVER

3. Posizionarsi all'interno della cartella server [`>> cd server`]
4. Compilare i file *.c digitando "make" [`>> make`]

APPLICAZIONE CLIENT

3. Posizionarsi all'interno della cartella client [`>> cd client`]
4. Compilare i file *.c digitando "make" [`>> make`]

In caso di problemi durante la compilazione digitare "make clean" [`>> make clean`] e tornare al punto 4.

5.2 CONFIGURAZIONE

Per la configurazione dei parametri salienti di RUA posizionarsi all'interno dei file di configurazione (in RUA/server/inc/config.h nel caso del server e in RUA/client/inc/config.h nel caso del client).

Il file di configurazione presenta i seguenti parametri:

SERVER

PARAMETRO	DESCRIZIONE	VALORI
TIMEOUT_INTERVAL	Intervallo di timeout: indica la durata di attesa di acknowledgement di un pacchetto prima della ritrasmissione.	1000000 (default) Il valore di default è 1 sec. Tale parametro è espresso in microsecondi.
ADAPTIVE_TIMER	Intervallo di timeout adattivo: indica il metodo di calcolo dell'intervallo di timeout.	1 (default) Per rendere l'intervallo di timeout fisso inserire il valore 0.
ALPHA	ALPHA: parametro relativo al calcolo dell'intervallo di timeout adattivo. Non assume alcuna importanza in caso di intervallo di timeout fisso.	0.125 (default) I valori che possono essere assunti vanno da 0 a 1.
BETA	BETA: parametro relativo al calcolo dell'intervallo di timeout adattivo. Non assume alcuna importanza in caso di intervallo di timeout fisso.	0.25 (default) I valori che possono essere assunti vanno da 0 a 1.
MAX_CLIENTS	Massimo numero di clients supportati.	10 (default)
MAX_FILE_SIZE	Massima dimensione dei file trasferibili.	2147483648 (default) Il valore di default è 2 GB. Tale parametro è espresso in bytes.
WINDOW	Dimensione della finestra di scorrimento del protocollo a ripetizione selettiva.	8 (default)
PROBABILITY_LOSS	Probabilità di perdita: indica la probabilità del verificarsi della simulazione di perdita di un pacchetto ricevuto.	30 (default) Il valore di default è 30%. Tale parametro è espresso in percentuale.

CLIENT

PARAMETRO	DESCRIZIONE	VALORI
TIMEOUT_CONNECTION	Intervallo di timeout di connessione: indica la durata di attesa massima senza ottenere una risposta dal server.	5 (default) Il valore di default è 5 sec. Tale parametro è espresso in secondi.
TIMEOUT_INTERVAL	Intervallo di timeout: indica la durata di attesa di acknowledgement di un pacchetto prima della ritrasmissione.	1000000 (default) Il valore di default è 1 sec. Tale parametro è espresso in microsecondi.
ADAPTIVE_TIMER	Intervallo di timeout adattivo: indica il metodo di calcolo dell'intervallo di timeout.	1 (default) Per rendere l'intervallo di timeout fisso inserire il valore 0.
ALPHA	ALPHA: parametro relativo al calcolo dell'intervallo di timeout adattivo. Non assume alcuna importanza in caso di intervallo di timeout fisso.	0.125 (default) I valori che possono essere assunti vanno da 0 a 1.
BETA	BETA: parametro relativo al calcolo dell'intervallo di timeout adattivo. Non assume alcuna importanza in caso di intervallo di timeout fisso.	0.25 (default) I valori che possono essere assunti vanno da 0 a 1.
WINDOW	Dimensione della finestra di scorrimento del protocollo a ripetizione selettiva.	8 (default)
PROBABILITY_LOSS	Probabilità di perdita: indica la probabilità del verificarsi della simulazione di perdita di un pacchetto ricevuto.	30 (default) Il valore di default è 30%. Tale parametro è espresso in percentuale.

5.3 ESECUZIONE

SERVER

Per l'esecuzione del server RUA aprire il terminale, posizionarsi all'interno della cartella RUA/server e digitare il comando "make run". Il server creerà automaticamente una socket con porta 1024 in attesa delle richieste di nuovi client.

CLIENT

Per l'esecuzione del client RUA aprire il terminale, posizionarsi all'interno della cartella RUA/client e digitare il comando "make run". Il client cercherà automaticamente di connettersi senza autenticazione al server. Nel caso in cui il server non sia attivo, è presente una deadline temporale (configurabile in TIMEOUT_CONNECTION in RUA/client/inc/config.h) che terminerà l'esecuzione del client in caso di mancata connessione.

5.4 UTILIZZO DEL CLIENT RUA

All'esecuzione del client RUA sarà possibile eseguire il download e l'upload di files. Per ottenere una lista dei file disponibili nel server digitare il comando list.

DOWNLOAD

Per eseguire il download di un file dal server RUA digitare "get <nomefile>". Il file verrà salvato nella cartella RUA/client/file_client. In caso di fallimento sarà riportato un opportuno messaggio di errore.

UPLOAD

Per eseguire l'upload di un file nel server RUA assicurarsi che il file in questione sia residente nella cartella RUA/client/file_client e digitare "put <nomefile>". Il file verrà salvato nella cartella RUA/server/file_server. In caso di fallimento sarà riportato un opportuno messaggio di errore.

6. ESEMPI

Esecuzione del Client riuscita

```
Launching RUA client software...
Initializating environment...
Trying to connect to server...
5 seconds left for connection...
Client connected succesfully!

*** WELCOME TO RUA ***
With RUA you can download and upload files.

Choose an operation:

- list           [Usage >> list]
- download       [Usage >> get filename]
- upload         [Usage >> put filename]
- exit           [Usage >> exit]
```

Esecuzione del Client fallita

(Il server non è in esecuzione)

```
Launching RUA client software...
Initializating environment...
Trying to connect to server...
5 seconds left for connection...
4 seconds left for connection...
3 seconds left for connection...
2 seconds left for connection...
1 seconds left for connection...
Couldn't connect to server.
```

Esecuzione del server

e successiva connessione instaurata con un client

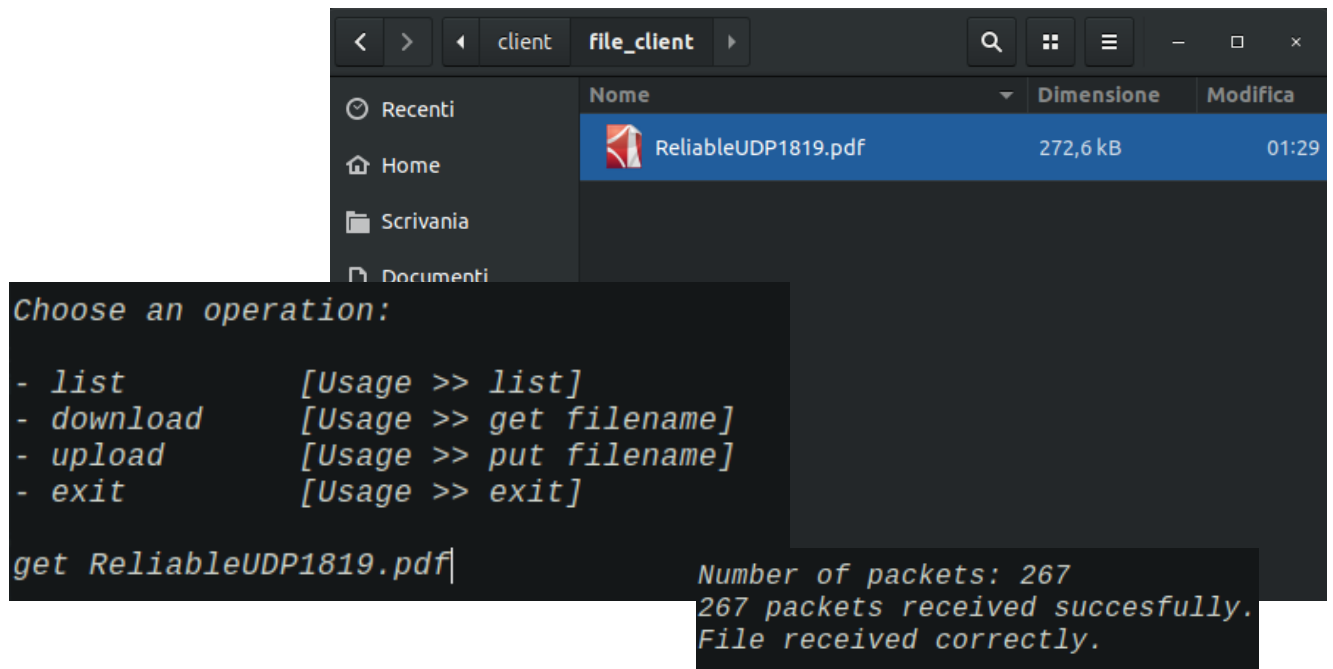
```
Starting server...  
  
Socket 1024 created succesfully.  
Binding success ✓  
Number of clients: 1  
  
Socket 1025 created succesfully.  
Binding success ✓
```

LIST (lato client)

```
Choose an operation:  
  
- list           [Usage >> list]  
- download      [Usage >> get filename]  
- upload        [Usage >> put filename]  
- exit          [Usage >> exit]  
  
list|
```

```
LIST OF SERVER AVAILABLE FILES:  
ReliableUDP1819.pdf
```

GET (lato client)



The image shows a file explorer window titled 'file_client' displaying a file named 'ReliableUDP1819.pdf' with a size of 272,6 kB and a modification time of 01:29. Below the file explorer, a terminal window displays the following text:

```
Choose an operation:
- list      [Usage >> list]
- download  [Usage >> get filename]
- upload    [Usage >> put filename]
- exit      [Usage >> exit]

get ReliableUDP1819.pdf|
Number of packets: 267
267 packets received succesfully.
File received correctly.
```

GET (lato server)

```
Processing request from client connected on port 1025.
GET request for ReliableUDP1819.pdf
Pathname is ./file_server/ReliableUDP1819.pdf
@@@ file_len = 272553
    payload = 1024
    num_pkt = ceil(272553/1024) = 267
All packets (267) have been built and file is closed.
Sending packets...
DOWNLOAD TIME    :    0.053582 seconds
THROUGHPUT       :    40.693218 Mbps [ 4983.016685 pkts / sec ]
```

PUT (lato client)

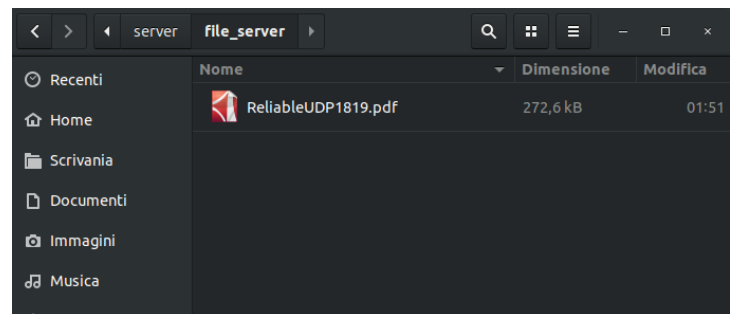
```
Choose an operation:
```

```
- list          [Usage >> list]
- download     [Usage >> get filename]
- upload       [Usage >> put filename]
- exit         [Usage >> exit]
```

```
put ReliableUDP1819.pdf|
```

```
ReliableUDP1819.pdf size is 272553. Splitting data in 267 packets.
@@@ All packets (267) have been built and file is closed.
Sending packets...
UPLOAD TIME :    0.073913 seconds
THROUGHPUT  :    29.499871 Mbps [ 3612.355066 pkts / sec ]
```

PUT (lato server)



```
Processing request from client connected on port 1025.
PUT request. Receiving ReliableUDP1819.pdf (272553 bytes) from port 1025
Number of packets: 267
267 packets received succesfully.
File received correctly.
```

Gestione concorrente di più Client (lato server) e sfruttamento della bitmap delle porte

```
Socket 1024 created succesfully.  
Binding success ✓  
Number of clients: 1  
  
Socket 1025 created succesfully.  
Binding success ✓  
Processing request from client connected on port 1025.  
Number of clients: 2  
  
Socket 1026 created succesfully.  
Binding success ✓  
Number of clients: 3  
  
Socket 1027 created succesfully.  
Binding success ✓  
Client exit from port 1025.  
  
Number of clients: 3  
  
Socket 1025 created succesfully.  
Binding success ✓
```