Progetto di Reti di Calcolatori:
Trasporto affidabile multi-percorso

Orgest Shehaj <orgest.shehaj@studio.unibo.it>
Matricola: 0000577768
Michele Goitom Tuccu <goitomtu@studio.unibo.it>
Matricola: 0000593026

Indice

1 Introduzione

1.1 La simulazione tcp/udp

E necessario la realizzazione di un software per la simulazione della comunicazione tcp/udp.
Questo progetto e formato da 5 software, il sender, il psender, il Ritardatore , il preceiver e il receiver.

1.2 Funzionalità del psender/preceiver

L'attuale progetto consiste nel realizzare il psender e il preceiver.
Il psendere e il preceiver sono dei programmi che si interpongono

tra un client (sender) e un server (receiver).
Il psendere stabilisce una comunicazione tcp con il sender e rimane in attesa di eventuali pacchetti.

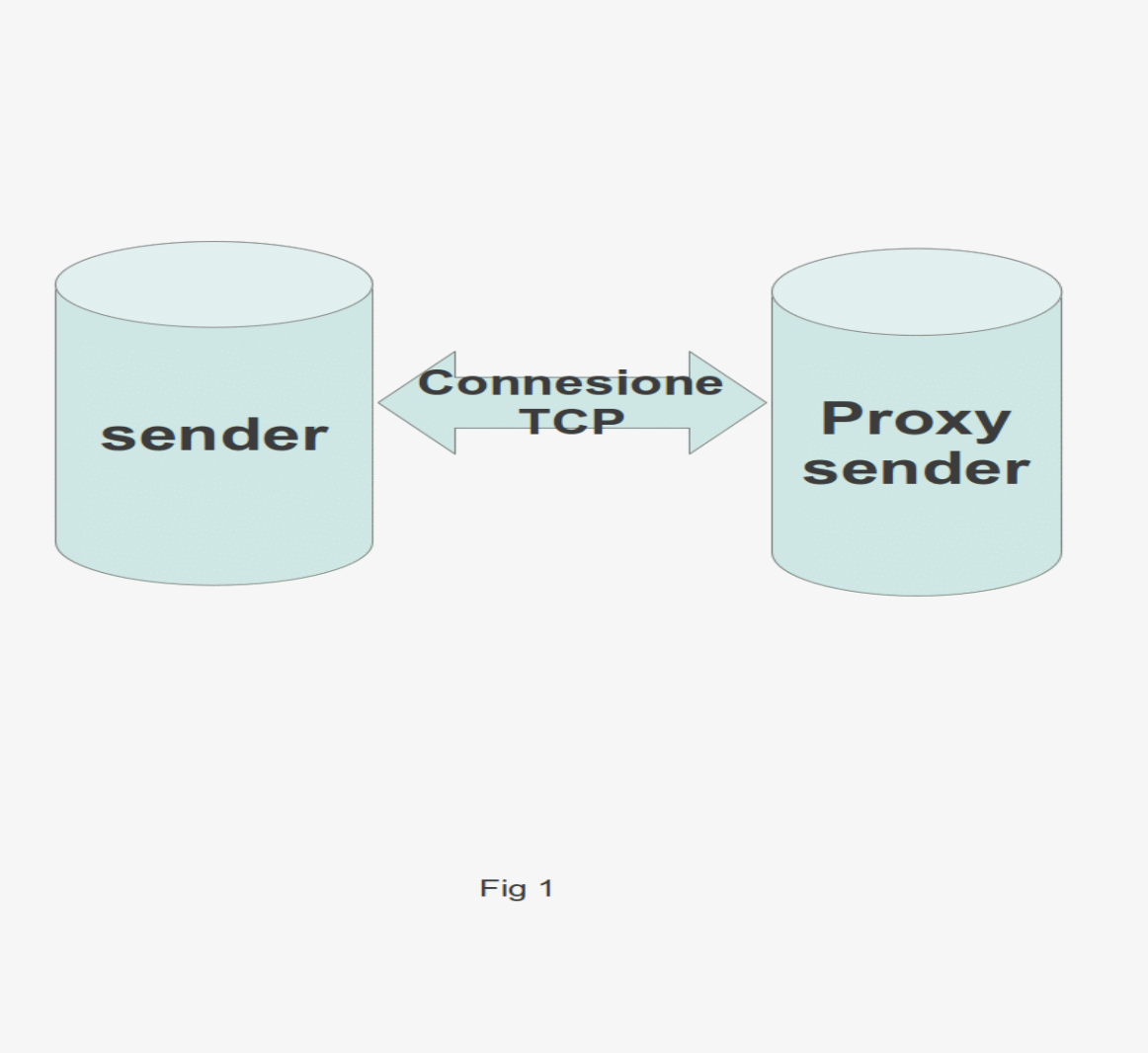


Fig 1

A sua volta il psender tutti i datti che riceve dal sender li invia al preceiver attraverso il Ritardatore, e il preceiver a sua volta invia tutti i dati al receiver (ovvero il nostro server).Per simulare il funzionamento di una rete reale il Ritardatore può ritardare oppure perdere i pacchetti derivanti dal psendere oppure dal preceiver. Il psender resta in ascolta per eventuali dati.
Il psender accumula i pacchetti che riceve dal sender in una lista dopo di che spedisce tutti i pacchetti che riceve dal sender al preceiver attraverso una delle tre porte del Ritardatore.
Il preceiver invia un ack al psender per ogni pacchetto che riceve attraverso una delle tre porte del Ritardatore, dopo di che spedisce quel pacchetto al receiver dove verrà ricomposto il messaggio iniziale del sender.
Tra il preceiver e il receiver si instaura una connessione tcp, secondo il seguente schema :

Fig 2

Il psender e il preceiver possono comunicare tra di loro esclusivamente tramite il Ritardatore. Il Ritardatore ha 6 porte udp in totale, tre delle quali vengono utilizzate per comunicare con il psender e le altre tre per comunicare con il preceiver.

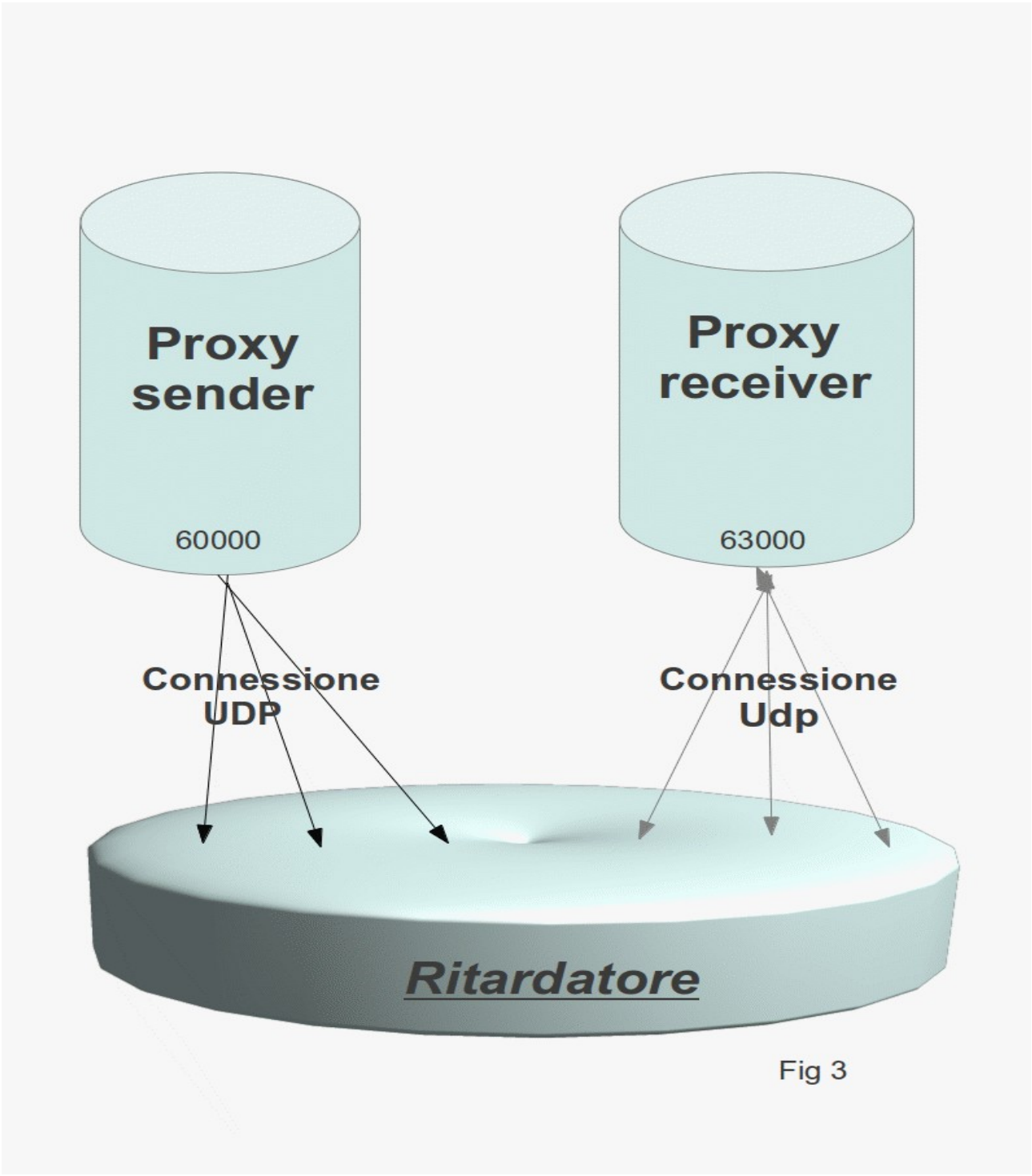
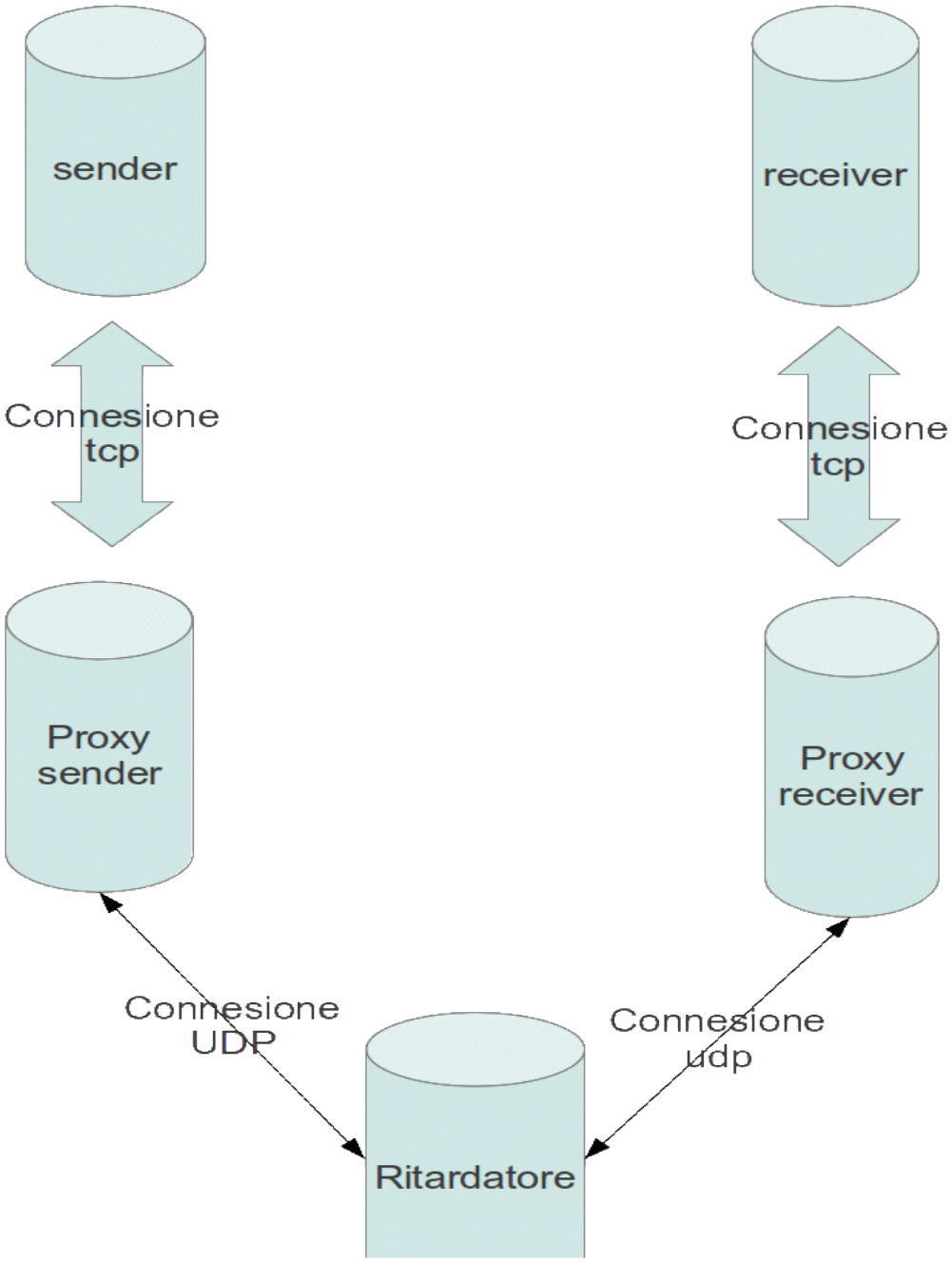


Fig 3

Quindi il nostro schema generala sarà il seguente :



Mentre il receiver una volta che acquisisce i dati attraverso il preceiver, ricostruisce il messaggio .

1.3 **Aspetti tecnici**

1.3.1 Compilazione

Il progetto è fornito di un script compila.sh e di un Makefile (diverso da quello originale) in modo da facilitarne la compilazione. Per compilare il progetto è sufficiente digitare da terminale il seguente comando:

$ ./compila.sh
oppure :
$ make

Per cancellare tutti i file oggetto digitare da terminale:
$ make clean

1.3.2 Documentazione

La presente documentazione (.PDF) spiega quelle che sono state le scelte di implementazione del preceiver/psender, per visionare la documentazione del codice C fare riferimento alla documentazione doxygen. Per creare la documentazione digitare dal terminale:

$ make doc

1.3.3 Uso di CPU

Per misurare l'utilizzo della CPU durante l'esecuzione di un trasferimento, abbiamo utilizzato il programma htop,
che visualizza le statistiche di tutti i processi del sistema.
L' utilizzo della CPU naturalmente dipende dalla quantitá di pacchetti in circolazione, un dato che si può dedurre dal numero di pacchetti presenti nel psender e nel preceiver.

2 Implementazione

2.1 Sender

Come sender per il progetto usiamo il sender in dotazione con i sorgenti (cliTCP) al quale sono state apportate delle piccole modifiche per sfruttare al meglio le sue capacità.

Qualunque sender si sceglie di usare il progetto funziona correttamente .

2.2 psender

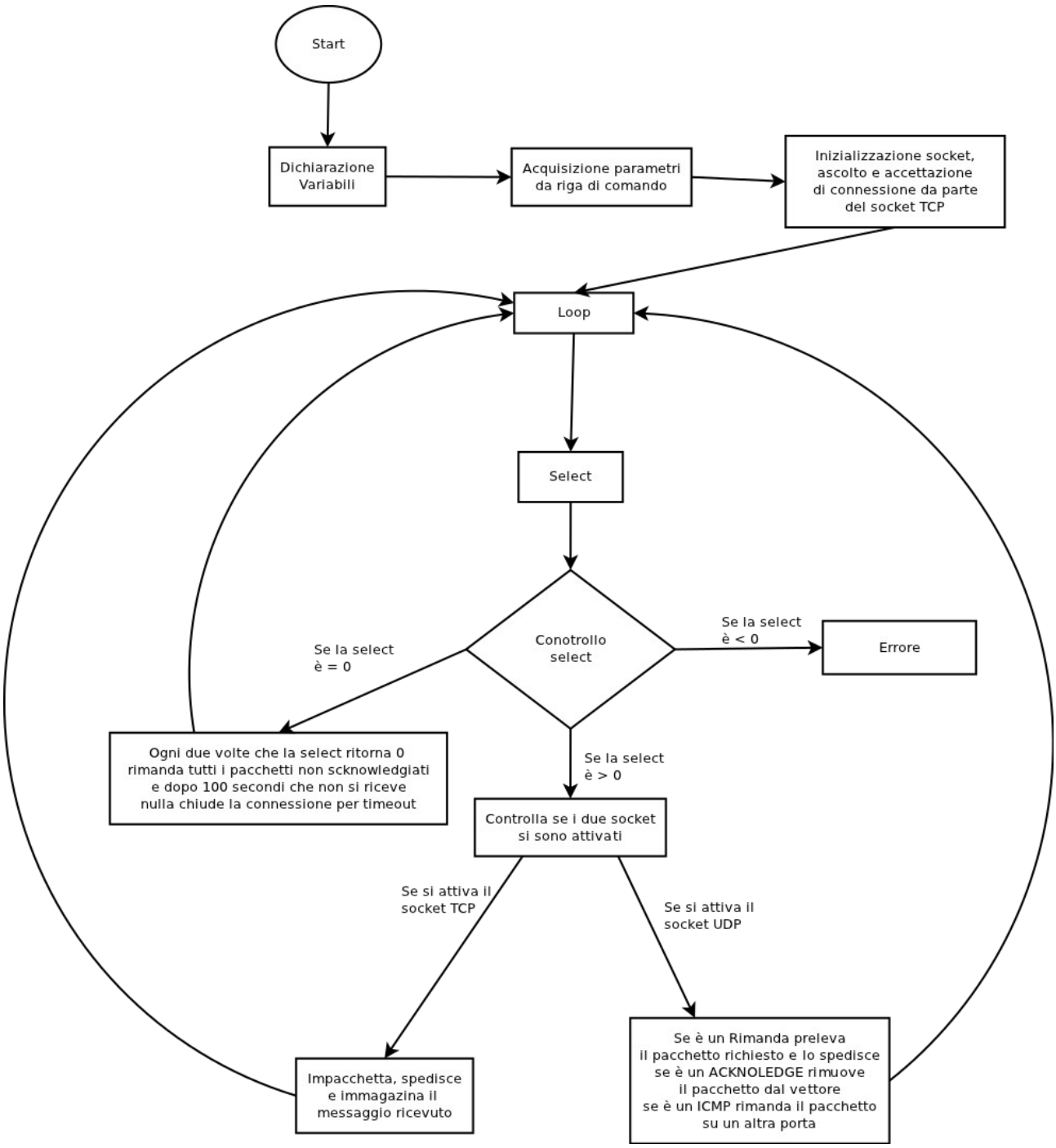
Tra psender e Proxy_recevier viene implementato un protocollo UDP che permette di trasportare un flusso TCP tre i due HOST mantenedolo invariato.

La comunicazione tra i due proxy avviene attraverso il Ritardatore, il quale decide se far passare il flusso di dati integri oppure scartare dei pacchetti casualmente, ogni volta che avviene lo scarto di un pacchetto il Ritardatore può avvisare uno dei due Proxy che e avventuo lo scarto oppure non lo notifica per niente questo avvenimento. Tocca ai due proxy garantire l'integrità del flusso di dati.

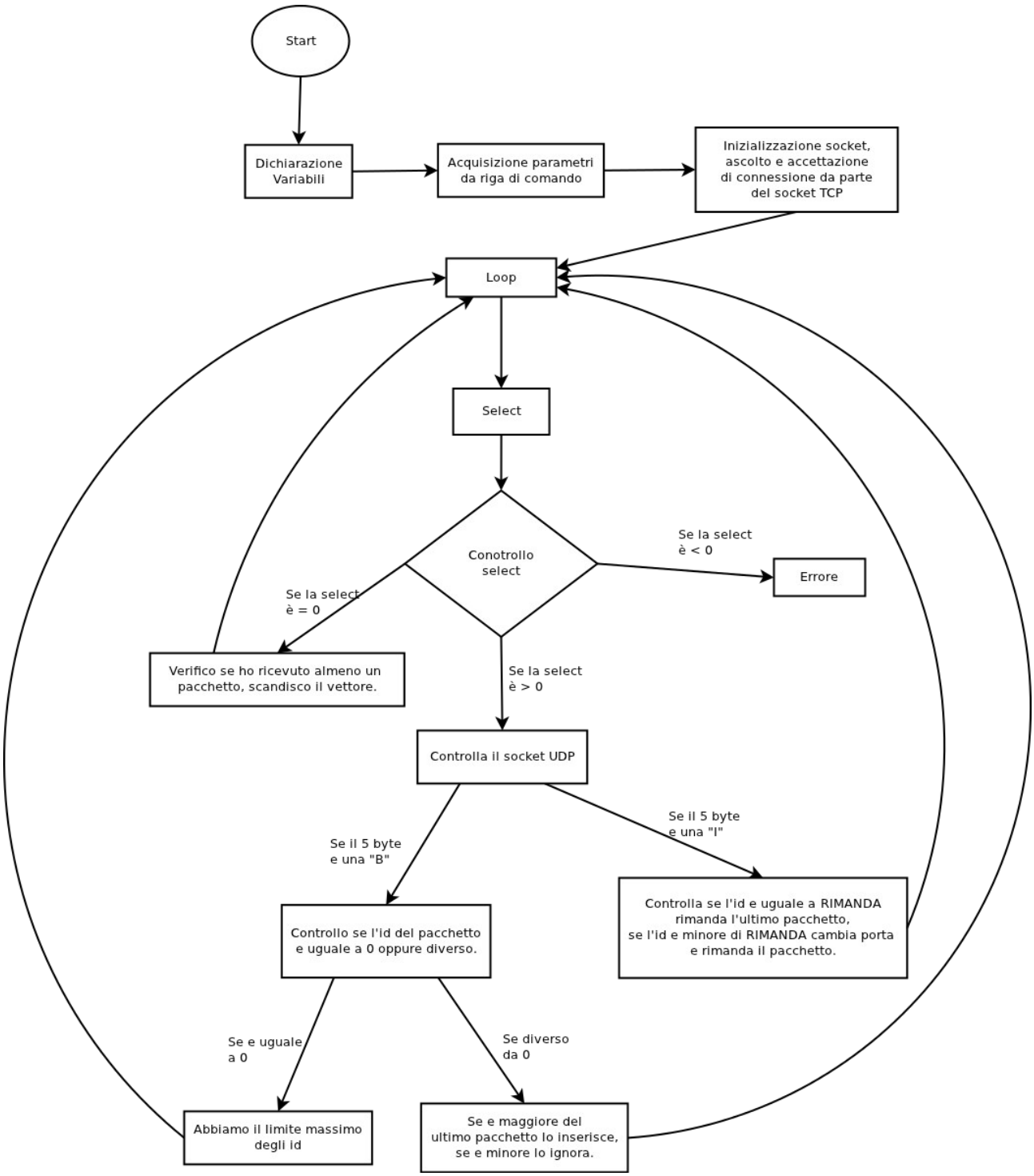
> I datagram UDP possono arrivare in ordine diverso da quello inviato quindi il protocollo deve anche garantire il riordinamento dei pacchetti.

Una volta che il psendere si connette con il sendere inizia il trasferimento dei dati dalla parte del sender.

Ogni dato che il psender riceve dal sender viene salvato in un vettore e dopo viene inviato al preceiver.



2.3 preceiver



2.4 Ritardatore

Il ritardatore e rimasto invariato, non sono state aportate modifiche di alcun tipo.

2.5 Receiver

Il receiver (receiverTCP.c) riceve i pacchetti dal preceiver e forma il messaggio iniziale che e partito dal sender stampandolo sul terminale .
Anche qui qualunque sia il receiver scelto per fare le prove, il progetto funziona correttamente.

2.6 utility.c && utility.h

In utility.c ci sono tutte le funzioni fatte da noi e usate in psender/preceiver, in più ci sono anche le strutture dati create per tenere conto di tutti i dettagli.

2.6.1 Strutture dati

Le strutture create da noi sono tre:

```
/* struct utiliziate */
typedef struct {
  uint32_t id;
  char tipo; /* B = body, I = icmp */
  char ack; /* N = normal, X = fine */
  int msg_size;   /*Dimensione del messaggio*/
  char msg[MSGSIZE];
  } __attribute__((packed)) PACCO;

/*struttura ICMP*/
typedef struct {
  uint32_t id;
  char tipo;
  uint32_t id_pkt;
} __attribute__((packed)) ICMPACK;
```

```
/*struttura contenente informazioni*/
typedef struct {
    int tot; /*totale byte */
    int idmax;
    int pkt_counter;
    int icmp;
    int ack; /* ack spediti(pr)/ricevuti(ps) */
```

```
    int rimanda; /* richieste spedite(pr)/ricevute(ps) */
    time_t ini;
    time_t fin;
} INFO;
```

Le procedure realizzate sono le seguenti:

```
void init_info (INFO *info);
uint16_t scegli_door (uint16_t origine, uint16_t old);

/* Funzioni per la preparazione dei socket */
int tcudp_setting (int *sockfd, uint16_t porta, int tipo_sock);
void sblocca (int *sockfd);

/* Funzioni per la spedizione */
int send_udp (struct sockaddr_in dest, char *ip_ricevente, unsigned short
porta_ricevente, int sockfd, PACCO msg);
int send_tcp (int sockfd, char *msg, int len);
int send_ack (struct sockaddr_in dest, char *ip_ricevente, unsigned short
porta_ricevente, int sockfd, ICMPACK msg);

/* scansione vettore */
int multi_sendtcp (PACCO* vett[], int inizio, int nfine, struct sockaddr_in dest, char
*ip_dest, unsigned short porta_dest, int sockfd, INFO *info);
int scan_null(PACCO *vett[], int inizio, int fine);
void libera_null(PACCO *vett[],int inizio, int fine)

/* Funzioni per impacchettare e spacchettare */
void pkt_udp(char* buf, int i, int dim_buf, PACCO *pacco);
void spkt_udp(char *buf, PACCO *pacco);
void spkt_icmpack (char *buf, ICMPACK *pkt);
void fine_pkt (PACCO *pack, uint32_t id);
```

Costanti:

```
#define MSGSIZE 64497 /* lettura del proxysender */
#define BUFSIZE 65507 /* lettura del proxyreceiver */
#define RIMANDA  (unsigned int)2000000000 /* piu' 128 tera*/
#define SOCKET_ERROR   ((int)-1)
#define IDFINE 0 /* id pacchetto finale */
#define VECT_SIZE 10000 /* dimensione iniziale vettore */
```

3 The Source Code