Laboratorio di ingegneria informatica

Anno accademico 2015-2016

Davide Talon 1075692

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

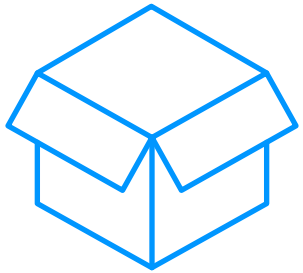
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Link Application

Send files over local network

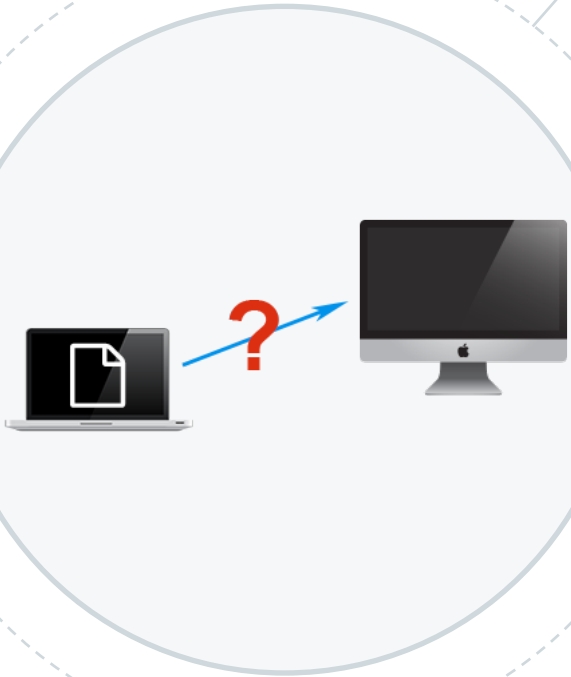
Introduzione

Motivazioni e obiettivi del progetto



Caso di interesse

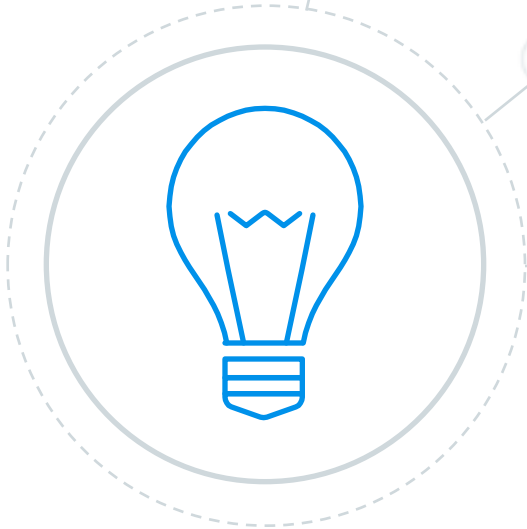
Quante volte capita, in casa o in ufficio, di dover inviare dei file da un computer all'altro?



Obiettivi

- Sviluppare un' applicazione che permetta lo scambio di file tramite la rete locale
- Rendere l'applicazione intuitiva e veloce
- Acquisire conoscenze in merito alle reti e al loro funzionamento

Progettazione



Principi di funzionamento e architettura software

I socket

I socket fanno parte della IPC di Unix e rappresentano il metodo con cui si ha lo scambio di dati, oltre che sulla stessa macchina, anche tra macchine connesse in rete.

SOCKET STREAM

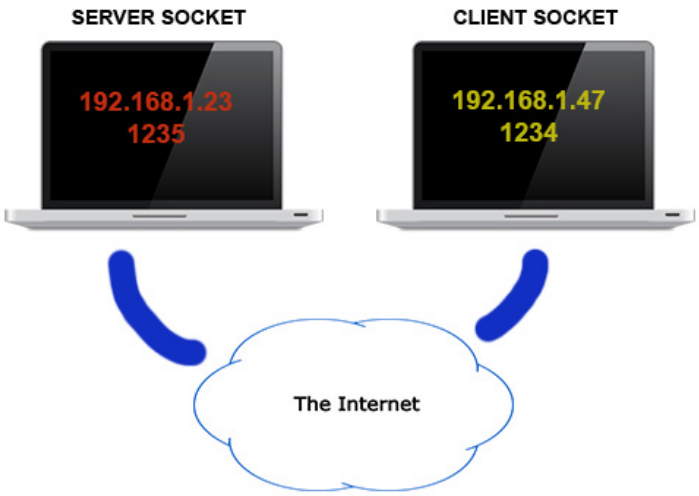
Permettono di gestire una trasmissione TCP ovvero affidabile, orientata alla connessione e senza limiti di dimensioni nel trasferimento dati.

SOCKET DGRAM

Implementano una trasmissione UDP, consentono un trasferimento dati veloce senza assicurare che la trasmissione sia avvenuto in modo corretto, i dati possono arrivare in ordine inverso, sbagliati o addirittura non arrivare.

Il funzionamento dei socket

- Ogni applicazione, sia lato client che lato server, apre un socket , ovvero un interfaccia che permette alle applicazioni di comunicare l'una con l'altra.
- La trasmissione è identificata dalla coppia di socket.



Le due librerie principali

Master

la piattaforma si mette in ascolto su una specifica porta UDP e aspetta di ricevere una richiesta da parte di qualche mittente, ricevuta la richiesta apre un socket TCP e riceve i dati in ingresso.

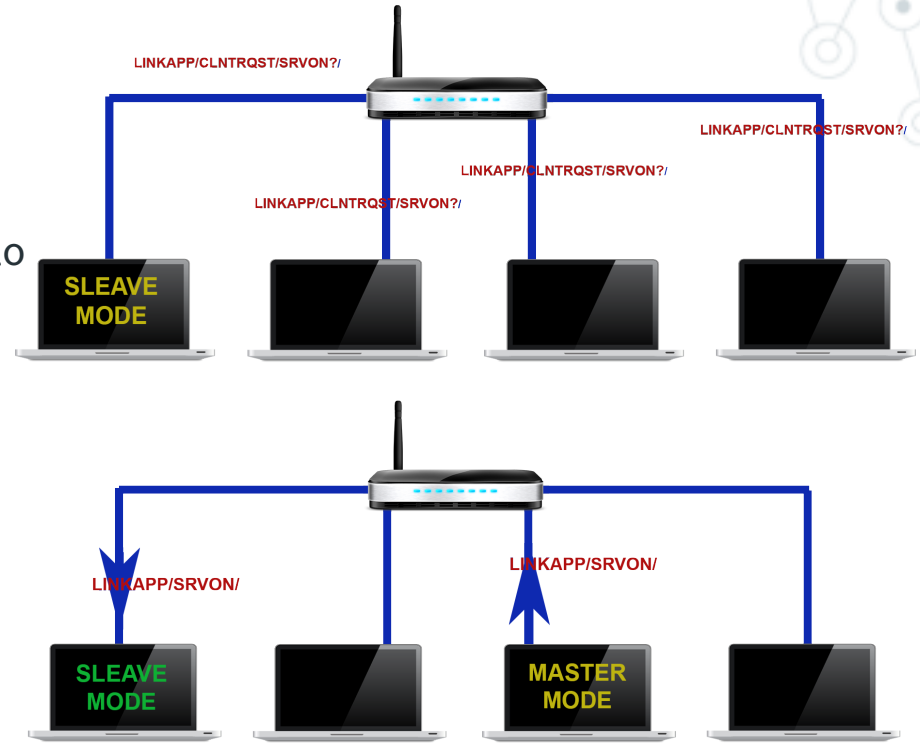
Slave

modalita` con cui è possibile inviare un file, infatti, entrati in modalita` slave si cercano tutti i master in ascolto all'interno della rete e si inviano i dati.

Slave

Entrati in slave mode l'applicazione esegue alcuni semplici passi:

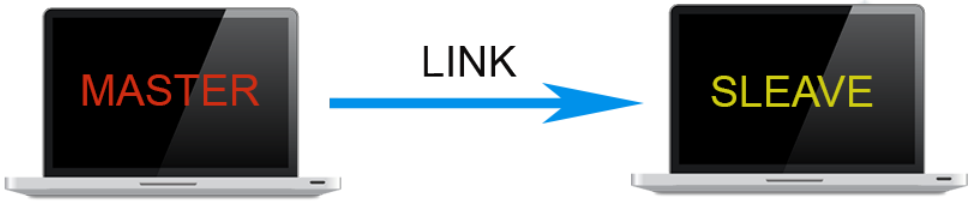
- Invia un messaggio in broadcast per rilevare i master presenti nella rete
- Attende la risposta dei vari master disponibili
- Prepara il file per l'invio comprimendolo
- Instaura una connessione TCP con il master scelto
- Attende che il destinatario accetti il file
- Invia il file

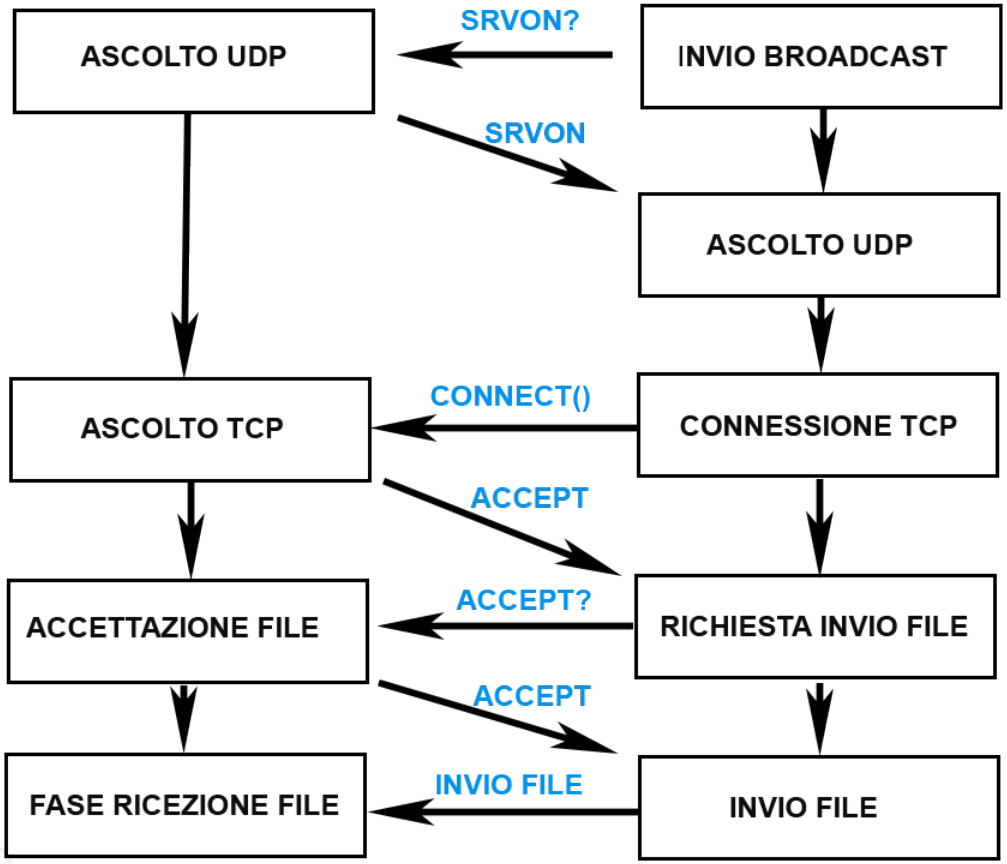


Master

La master mode, invece :

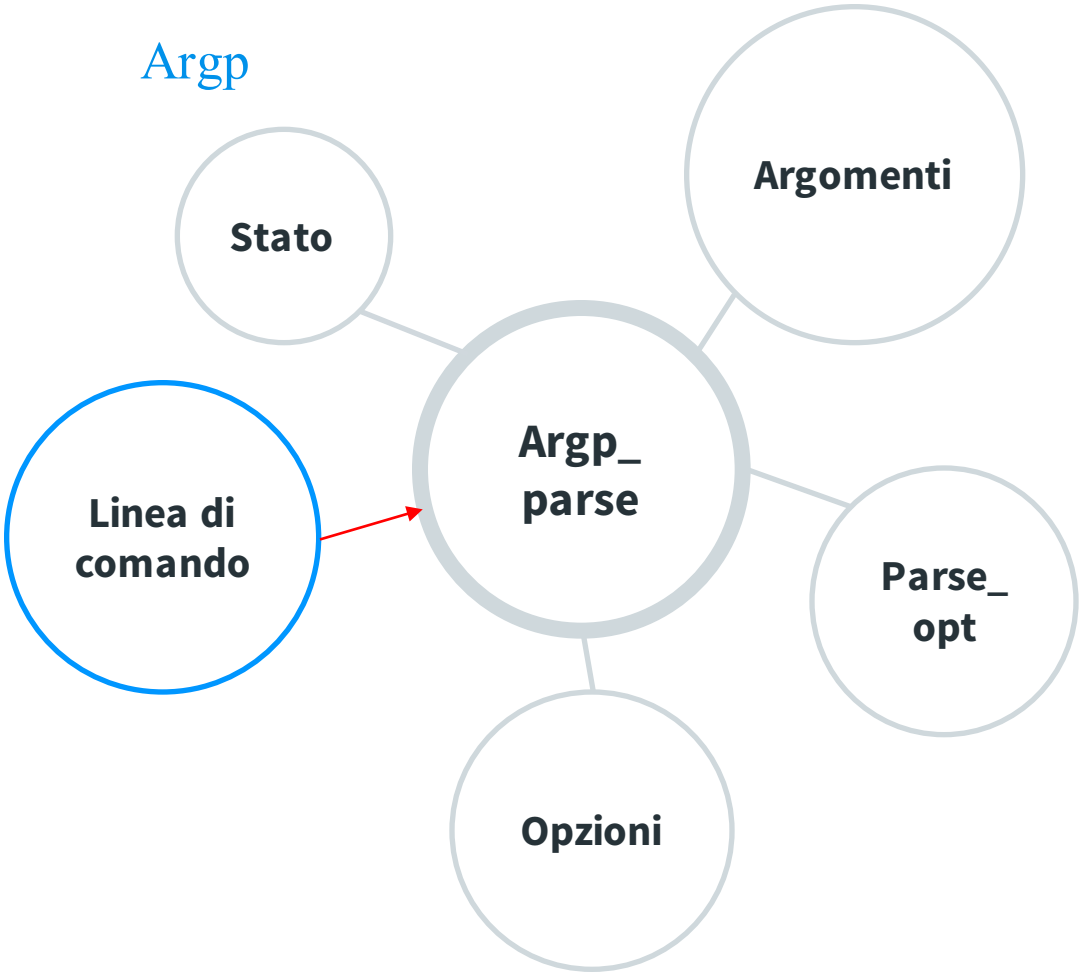
- Apre un socket UDP in attesa di eventuali richieste da parte di uno slave
- Risponde alla richiesta dello slave comunicando il proprio nome
- Apre un socket TCP e si pone in ascolto
- Dopo aver accettato la connessione riceve le informazioni sul mittente e il file
- Chiede all'utente se accettare il file
- Riceve il file





FILE RICEVUTO ✅

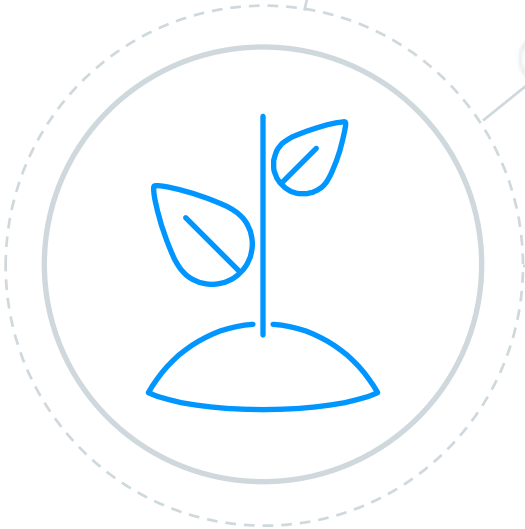
Argp



Per il parsing dei comandi da linea di comando viene utilizzata una ridotta libreria derivante da **Argp** di GNU.

Le opzioni utilizzabili sono passati alla funzione **argp_parse** la quale, attraverso un registro di stato e la funzione **parse_opt** si occupa di effettuare il parsing in modo elegante e sicuro modificando gli argomenti di esecuzione.

Sviluppo



Realizzazione e implementazione

Scoprire i master nella rete

I master trovati vengono salvati nella struttura **struct srv** così definita in **linkutils.h**

```
struct srv {
        char name[MAX_NAME_LENGTH];
        struct sockaddr_in sockAddr;
        int sockAddrLen;
};
```

CODE

Per scoprire i master disponibili nella rete si può utilizzare la funzione **srvsInNet()** in **sleave.c** passandogli il socket UDP da utilizzare e un array di **srv**

```
int srvsInNet(const int udpClntSock, srv *srvs)
```

CODE

La quale setta il socket in **NONBLOCK**

```
//set socket non blocking
fcntl(udpClntSock, F_SETFL, O_NONBLOCK);
```

CODE

Scoprire i master nella rete

Rimane dunque in attesa di messaggi da parte dei master disponibili finché non è stato superato il numero di server massimo o è scaduto il tempo della ricerca.
Durante il ciclo valuta la validità della risposta ricevuta, ottiene le informazioni relative al master trovato e lo aggiunge all'array **srvs**.

```
while ((nSrvs <= MAX_NUMBER_SERVERS ) && (time(0) < excededTime) ) {

        //checking received message
        recvfrom(udpClntSock, buffer, SERVICE_BUFFER_SIZE, 0,
          (struct sockaddr *) &currentAddr, &currentAddrLen);

        //CODE for handle error from recvfrom

        //checking received message
        if (strncmp(buffer, VALID_SERVER_ON, 14) == 0) {
                //CODE for getting server name
                strName = strtok(NULL, "/");

                //CODE for getting server address
                currentSrv.sockAddr = currentAddr;

                //adding current server
                srvs[nSrvs] = currentSrv;
                nSrvs++;
        }
}
```

CODE

Apertura connessione TCP

L'apertura della connessione TCP avviene in **masterMode()** dove prima viene creato un socket TCP e poi gli si assegna un indirizzo noto tramite la funzione **bind()**.

```
//server TCP address
struct sockaddr_in tcpSrvSockAddr;
memset(&tcpSrvSockAddr, 0, sizeof(tcpSrvSockAddr));

tcpSrvSockAddr.sin_family = AF_INET;
tcpSrvSockAddr.sin_port = htons(TCP_SERVER_PORT);
tcpSrvSockAddr.sin_addr.s_addr = htonl(INADDR_ANY);

//CODE for client TCP address
//create TCP server socket
int tcpSrvSock = socket (AF_INET, SOCK_STREAM, 0);

// CODE for handle socket error
// CODE for setting SO_REUSEADDR

if (bind(tcpSrvSock, (struct sockaddr *) &tcpSrvSockAddr,
 tcpClntSockAddrLen) < 0) {
          perror("\nCannot bind TCP server socket: ");
          return -1;
}

int tcpSrvIsOpen = openTcpSrv(tcpSrvSock, (struct sockaddr_in *) &tcpSrvSockAddr,
 tcpSrvSockAddrLen, (struct sockaddr_in *) &tcpClntSockAddr, tcpClntSockAddrLen,
 &connectionSock);
```

CODE

Apertura connessione TCP

Con la chiamata della funzione **openTcpSrv()** si pone il master in ascolto imponendogli una **backlog** di 5, ovvero limitando il numero di connessioni in sospeso nella coda del socket. Si accettano tutte le richieste di connessione**.**

```
if(listen(tcpSrvSock, 5) < 0) {
        printf("Error starting listening\n");
}

//search for a connection request
int request;
while (1) {

        printf("ConnectionSock: %d\n", connectionSock);
        //accept connection with sleave
        request = accept(tcpSrvSock, (struct sockaddr *) &tcpClntSockAddr,
          &tcpClntSockAddrLen);

        *connectionSock = request;

        if(connectionSock < 0) {
                perror("Connection with sleave not accepted: ");
                return -1;
        }

        return 0;
}
```

CODE

Parsing dei comandi

Argp.h parte della ominima libreria di GNU mette a disposizione la struttura **argp_option** in cui vengono definiti i comandi accettabili

```
struct argp_option {
    char *name;
    char  key;
    char *arg;
    int   flags;
    char *doc;
    int   group;
};
```

CODE

```
static struct argp_option options[] = {
  {"verbose",  'v', 0, 0, "Produce verbose output" },
  {"listen", 'l', 0, 0, "Start listening" },
  {"send", 's', "<FILENAME>", 0, "Send file"},
  {"setname", 'n', "<NEWNAME>", 0, "Set user name" },
  {"getname", 'g', 0, 0, "Get user name"},
  { 0 }
};
```

CODE

Definiamo quindi **options**, un array di comandi che accettiamo da riga di comando

E definiamo la struttura struttura **arguments**, che verra` passata ad **Argp,** e viene utilizzata dal **main()** per comunicare con la funzione **argp_opt**

```
struct arguments {
  char *args[2];
  int listen, verbose, getName;
  char *sendFile;
  char *newName;
};
```

CODE

Parsing dei comandi

```
static error_t parse_opt (int key, char *arg, struct argp_state *state) {
        struct arguments *arguments = state->input;

        switch (key) {
        case 'l':
                arguments->listen = 1;
                break;
        // CODE for other cases

        default:
                return ARGP_ERR_UNKNOWN;
   }

  return 0;

}
```

CODE

La funzione **parse_opt()** verifica un parametro alla volta quanto passato e fissa lo stato del parsing

Una volta inizializzata la struttura **arguments** possiamo dunque costruire la struttura da passare ad **Argp** e invocare, all'interno del **main() argp_parse**

```
static struct argp argp = { options, parse_opt,
  args_doc, doc };
```

CODE

```
argp_parse (&argp, argc, argv, 0, 0, &arguments);
```

CODE

Valutazione

Testing e collaudo

Testing: avvio Master mode e Slave mode

La fase di sviluppo della piattaforma è stato tutto un susseguirsi di implementazione delle nuove funzionalità e di testing.

Durante le prove sono state sfruttate le diverse porte utilizzate dalla slave mode e dalla master mode, è stato possibile testare la correttezza dei risultati semplicemente utilizzando due terminali.

```
Davide:master  davidetalon$  link ––listen

Waiting  for sleave...UDP  request:
```

BASH

```
Davide:sleave  davidetalon$  link –send
canzone.mp3
```

BASH

Testing: scoprire i master nella rete

Mentre dal lato master, dopo aver risposto alla richiesta dello slave si apre un socket TCP, nel lato slave, ricevute le informazioni sui master presenti nella rete, si da la possibilità all'utente di scegliere il destinatario del file.

```
Davide:master davidetalon$ link ––listen

Waiting for sleave...UDP request:
LINKAPP/CLNTRQT/SRVON?
Server response:LINKAPP/SRVON/Tallo
Starting TCP server...
ConnectionSock: 1514626604
```

BASH

```
Davide:sleave davidetalon$ link ––send
canzone.mp3
Broadcast sent: LINKAPP/CLNTRQT/SRVON?
Waiting for master response...
1 masters found

0. Tallo
Scegliere un master valido:
```

BASH

Testing: invio header e accettazione del file

Nella slave mode possiamo notare come sia stato inviato l'header contente le informazioni relative al nome utente e al file da inviare, nella master mode, invece, si da la possibilità all'utente di accettare o meno il file.

```
Davide:master davidetalon$ link –listen
Waiting for sleave...
UDP request: LINKAPP/CLNTRQT/SRVON?
Server response:LINKAPP/SRVON/Tallo
Starting TCP server...
ConnectionSock: 1514626604
ConnectionSock after accept: 6
Connection enstabilished
ConnectionSock: 6
Header:LINKAPP/SLVNAME/Tallo/FNAME/canzone.
mp3.tar.gz/
Accept file canzone.mp3.tar.gz from Tallo?
(Y/N)
```

BASH

```
Davide:sleave davidetalon$ link ––send
canzone.mp3
Broadcast sent: LINKAPP/CLNTRQT/SRVON?
Waiting for master response...
1 masters found

0. TalloScegliere un master valido:0
Connetcting to Tallo...
Connection enstabilished.
tarCommand command: tar –zcf
canzone.mp3.tar.gz canzone.mp3
Headersent:LINKAPP/SLVNAME/Tallo/FNAME/canz
one.mp3.tar.gz/
Header size: 336
```

BASH

Testing: invio del file

Si procede poi con l'invio del file.

```
Davide:master davidetalon$ link –listen
Waiting for sleave...
UDP request: LINKAPP/CLNTRQT/SRVON?
Server response:LINKAPP/SRVON/Tallo
Starting TCP server...
ConnectionSock: 1514626604
ConnectionSock after accept: 6
Connection enstabilished
ConnectionSock: 6
Header:LINKAPP/SLVNAME/Tallo/FNAME/canzone.
mp3.tar.gz/
Accept file canzone.mp3.tar.gz from Tallo?
(Y/N) y
Receiving file canzone.mp3.tar.gz from
Tallo...
Serverresponse:LINKAPP/SEND/ACCEPTED/buffer
: 7210143
File size to receive: 7210143
Davide:master davidetalon$ ls
canzone.mp3.tar.gz
```

BASH

```
Davide:sleave davidetalon$ link ––send
canzone.mp3
Broadcast sent:LINKAPP/CLNTRQT/SRVON?
Waiting for master response...
1 masters found

0. Tallo
Scegliere un master valido: 0
Connetcting to Tallo...
Connection enstabilished.
tarCommand command: tar –zcf
canzone.mp3.tar.gz canzone.mp3
Header sent:LINKAPP/SLVNAME/Tallo/FNAME/
canzone.mp3.tar.gz/
Header size: 336
Server response:
LINKAPP/SEND/ACCEPTED/
Opening file canzone.mp3.tar.gz ...
size: 7210143
Remove command: rm –r canzone.mp3.tar.gz
File succefully sent.
```

BASH

Conclusione

Considerazioni
e lavoro futuro

Lavoro futuro

Risoluzione bug
Stabilizzare l'applicazione risolvendo i problemi riscontrati nella versione corrente.

Libertà dell'utente
Lasciare maggior libertà all'utente permettendogli di scegliere tra più formati di compressione

Porting Windows
Rendere la piattaforma disponibile anche per gli utenti Windows effettuando il porting.

Crittografia dati
Rendere più sicura la trasmissione dei dati introducendo la crittografia end-to-end.

Risultati raggiunti

- Raggiungimento degli obiettivi prefissati
- Utilizzo costante di nuove tecnologie, in particolare git e cmake
- Comprensione della necessità di documentare il codice
- Insonnia

Grazie.