Relazione

Livello trasporto

Ho scelto il protocollo TCP perché per un’applicazione di messaggistica è preferibile l’affidabilità al throughput

Livello applicazione

Ogni pacchetto è composto da un header e da un payload

|  |  |
| --- | --- |
| 1B | 4B |
| Tipo del messaggio | Lunghezza del payload |

* Header:
* Payload: cambia struttura a seconda del tipo del messaggio, questa è documentata nel dettaglio in *global.d/network\_tools/network\_tools.h*

Protocollo di scambio dei messaggi

Un pacchetto di tipo MESSAGE\_DATA contiene un messaggo di testo o file (specificato all’interno del pacchetto) e può essere inviato al destinatario (modalità P2P) o al server (modalità relay), in entrambi i casi arriverà al destinatario (in modalità relay dovrà essere richiesto con un pacchetto di tipo MESSAGE\_HANGING contenente l’username del mittente)

Se si utilizza la modalità relay si riceverà un messaggio MESSAGE\_SYNCREAD nel momento in cui il destinatario riceve il messaggio o quando faremo login se il messaggio è stato ricevuto mentre eravamo offline

Protocollo di ricezione dei messaggi inviati in modalità relay

Per ricevere una lista di messaggi in attesa da un dato utente dobbiamo mandare al server un pacchetto di tipo MESSAGE\_HANGING contenente l’username dell’utente da cui vogliamo riceve i messaggi, a questo punto dobbiamo ricevere una serie di pacchetti di tipo MESSAGE\_DATA terminata da un pacchetto di tipo MESSAGE\_RESPONSE (senza errori)

Opzionalmente possiamo mandare un pacchetto di tipo MESSAGE\_HANGING vuoto per ottenere una lista degli utenti che ci hanno mandato un messaggio in modalità relay, la risposta sarà formata da una lista di pacchetti di tipo MESSAGE\_HANGING contenenti ognuno un username, questa è terminata da un pacchetto di tipo MESSAGE\_RESPONSE (senza errori)

Protocollo di connessione P2P

Si dovrà richiedere la porta su cui sta in ascolto l’altro peer al server tramite un pacchetto di tipo MESSAGE\_USERINFO\_REQ e dovremo attendere un pacchetto di tipo MESSAGE\_USERINFO\_RES come risposta. Una volta ottenuta la porta dell’altro peer va aperta una connessione TCP con esso e va inviato un pacchetto di tipo MESSAGE\_LOGIN (senza password e porta) per fornire il nostro username. A questo punto possiamo inviare e ricevere pacchetti di tipo MESSAGE\_DATA su quella connessione

Gestione degli errori

Se inviamo un pacchetto al server e questo genera un errore di qualche tipo (signup con uno username già reistrato, credenziali di login errate, ecc…) viene inviato una pacchetto di tipo MESSAGE\_RESPONSE (con errore) invece della risposta attesa

Gestione delle richieste lato server

Il server gestisce le richieste in modo **sequenziale** con *select*, ho fatto questa scelta per evitare la complessità che deriva dalla programmazione concorrente e per facilitare la condivisione delle strutture dati comuni a tutte le connessioni.

Per evitare che il server rimanga in idle attendendo un intero messaggio da un device lento, il contenuto del messaggio viene bufferizzato in una struttura globale e utilizzato solo quando si riceve l’ultimo byte.

Gestione della modalità P2P dei device

Anche per la gestione dei messaggi ricevuti in P2P dai device viene utilizzato lo stesso codice che il server usa per gestire le richieste, in questo modo più messaggi possono essere ricevuti contemporaneamente.

Gestione della connessione al server da parte del device

I messaggi da parte del server sono gestiti in due modi diversi:

* Se non abbiamo mandato alcuna richiesta al server viene fatto un controllo grazie a *select* per controllare se ci sono messaggi da ricevere
* Se abbiamo fatto una richiesta al server il device si **blocca** in attesa della risposta

Ho scelto questo comportamento perché non posso sapere a priori quando il server invia una notifica ma se invio una richiesta non posso continuare l’esecuzione finché non ricevo la risposta (o non si verifica un errore)

Se perde la connessione con il server, il device tenta di riconnettersi periodicamente, senza però interrompere le altre attività