DOCUMENTAZIONE PROGETTO RETI INFORMATICHE 2021-22

ARDUINI LUCA, matricola 596307

SCELTE PROGETTUALI

● Utilizzo socket di tipo TCP

TCP assicura un trasporto affidabile dei dati e questo ci è particolarmente utile quando si deve condividere un file. Si è poi scelto di utilizzare questo tipo di socket per tutta l'applicazione.
Di conseguenza non è stato necessario stabilire una politica di controllo di avvenuta ricezione di messaggi tramite lo scambio di messaggi di "acknowledgement", che sarebbe invece necessaria nel caso di socket UDP.

● Connessioni TCP sempre attive

Si è scelto di mantenere le connessioni tra device-server o device-device sempre attive, anziché chiuderle dopo aver inviato un messaggio. Il motivo alla base di questa scelta è che si vuole mantenere l'applicazione il più reattiva possibile, dove i messaggi arrivano "in fretta" e in modo affidabile, senza dover stabilire una nuova connessione per ogni messaggio.
La scelta è stata fatta con la consapevolezza che questa non deve essere un'applicazione scalabile. Si è assunto che i client contemporaneamente online possano arrivare massimo a 50 unità, e che quindi la macchina server non saturi le sue risorse.

VANTAGGI:

- Quando un host1 deve comunicare con un host2 con il quale ha già scambiato pacchetti, non deve aprire una nuova connessione
- I device possono accorgersi facilmente che un altro device va offline. Altrimenti, senza connessioni persistenti, i device avrebbero dovuto continuamente accertarsi che i loro interlocutori non fossero andati offline tramite un periodico scambi di segnali.
  Questo aspetto è ancor più accentuato se si pensa alla disconnessione volontaria o involontaria del server. Senza connessioni persistenti, il server per comunicare la sua disconnessione avrebbe dovuto avvisare lui stesso tutti i device dell'imminente disconnessione, con tanto di eventuali attese nel caso non si riuscisse a mettere in contatto con un device nell'immediato.

SVANTAGGI:

- Applicazione non scalabile, in quanto avere molteplici connessioni TCP attive, richiede risorse, soprattutto sulla macchina server la quale è chiamata a mantenere una connessione con ognuno dei device attualmente online.
- Spreco di risorse. In un'applicazione di messagistica il "thinking time" è molto maggiore rispetto al tempo in cui effettivamente si inviano/ricevono dati, quindi avere queste connessioni sempre attive non è giustificato dal punto di vista dell'utilizzo di risorse.

● Molteplicità dei gruppi

Si è scelto di rendere possibile agli utenti di poter partecipare a più gruppi contemporaneamente, questo ha richiesto l'introduzione di un nuovo comando: "**\exit**" da utilizzare all'interno della schermata chat di un gruppo. Questo comando fa sì che l'utente esca dal gruppo, mentre il comando "\q" fa semplicemente tornare alla schermata del menù iniziale senza uscire dal gruppo. Un utente può quindi iniziare una chat singola, trasformarla in chat di gruppo aggiungendo un membro, tornare al menù principale ed iniziare una nuova chat singola con gli utenti che ora fanno parte del gruppo, oppure creare una nuova chat di gruppo, tutto senza uscire dal gruppo creato in precedenza.

● Invio e ricezione di file

Si è scelto ricevere i file in un loop *while*, anziché aprire il file, scrivere il chunk arrivato e chiuderlo. Questo ha indubbiamente lo svantaggio che mentre si riceve un file, l'utente non può ricevere notifiche di nuovi messaggi. Ma poiché si è assunto che i file inviati all'interno dell'applicazione abbiamo dimensioni limitate, si è preferito "bloccare" l'applicazione per qualche secondo invece

che fare molteplici aperture e chiusure del file che avrebbero richiesto comunque tempo andando a degradare le prestazioni.
Sempre con la supposizione che i file da inviare siano di piccole dimensioni, si è scelto che sia il mittente ad inviare in serie il file a tutti i membri del gruppo. Questo perché l'utente che invia un file, sa che questa operazione può richiedere un po' di tempo e quindi accetta che la sua applicazione possa fargli attendere qualche secondo prima di poter fare la prossima azione. Mentre se si fosse scelto di far inviare il file (una volta ricevuto) anche agli altri membri del gruppo, questi non sarebbero stati altrettanto contenti di vedere degradate le prestazioni della propria applicazione.

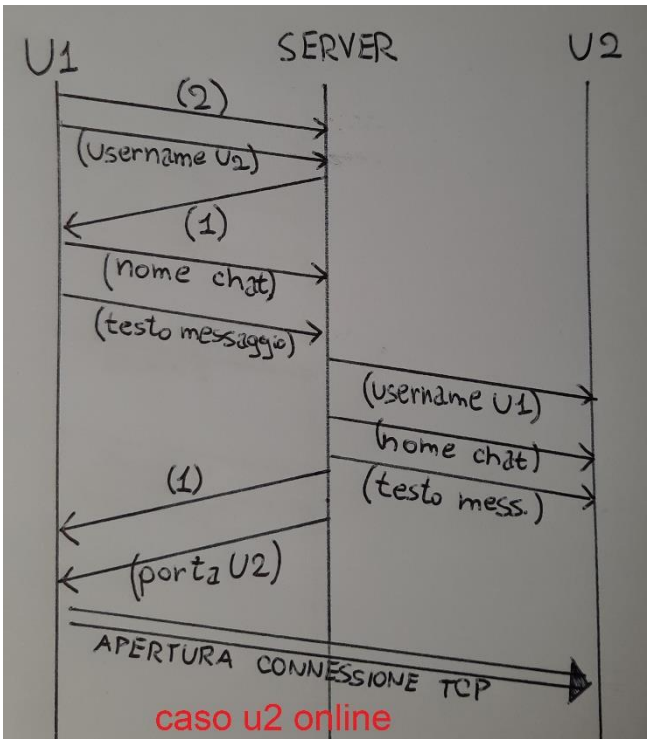
● **Protocolli con pacchetti di dimensioni minime**
Tutti i pacchetti di dati che vengono scambiati tra device-server o device-device non hanno un numero di byte predefinito, ma questi contengono volta per volta il numero strettamente necessario di informazioni. Come è ovvio, questa scelta è stata fatta per portare al minimo l'utilizzo della banda da parte dell'applicazione. A tal proposito è stato scelto di utilizzare un protocollo di tipo binary.

● **Protocollo di tipo binary**
Il motivo della scelta sta nella minimizzazione della banda utilizzata dall'applicazione. Inoltre questo tipo di protocollo evita le numerose codifiche e decodifiche che sono necessarie in un protocollo text.

PROTOCOLLO DI COMUNICAZIONE

Come anticipato, i pacchetti non hanno un formato standard, ma si limitano a contenere le informazioni strettamente necessarie. Questo porta molte volte ad inviare anche un solo byte, ad esempio quando un device vuole comunicare al server un'operazione da svolgere, gli invia un solo byte contente un codice.

● **Chat singola dove viene inviato un "primo" messaggio**
Se un utente (u1) vuole iniziare una chat con un utente (u2), dal momento che non è ancora stata stabilita una connessione TCP tra i due device, u1 per inviare il messaggio deve comunicare questa sua intenzione al server. Lo fa inviando al server (con il quale ha stabilito connessione TCP al momento del login) un codice "**2**" e l'username del destinatario (univoco in tutta l'applicazione). Il server controlla se l'utente è registrato nel database e risponde con "**1**" in caso affermativo, 0 altrimenti.
In caso di esito positivo, u1 comunica al server il nome della chat a cui appartiene il messaggio ed il testo del messaggio.

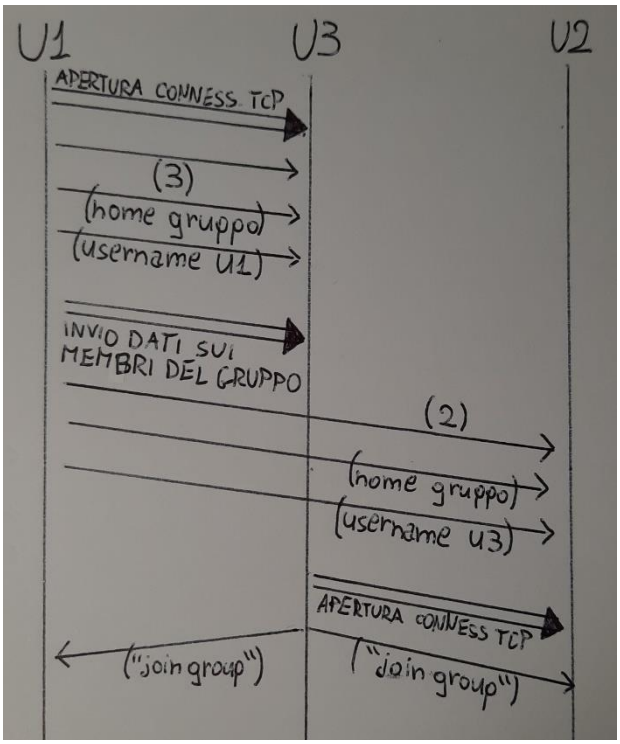


CASO u2 ONLINE: Il server controlla se u2 è attualmente online, se lo è, gli inoltra: username mittente, nome chat, testo del messaggio. Dopodiché invia l'esito dell'invio con "**1**" e gli invia la porta di ascolto di u2, con la quale u1 provvederà a stabilire una connessione TCP.

CASO u2 OFFLINE: Se u2 è offline, il messaggio viene memorizzato come pendente sul server e l'esito viene comunicato a u1. Quando u2 scaricherà i messaggi pendenti tramite l'uso della

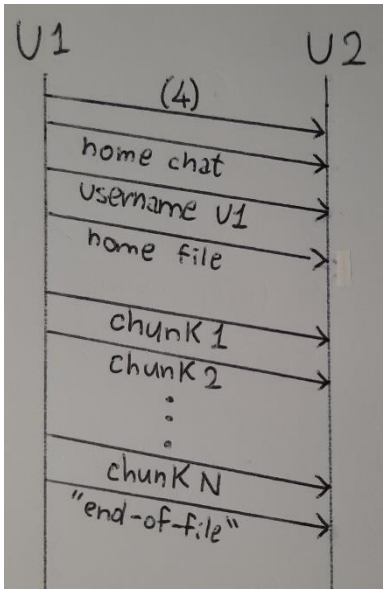
funzione “*show()*”, u1 ne verrà informato e segnerà nella sua memoria i messaggi come “recapitati”.

● Creazione chat di gruppo



Per creare una chat di gruppo si parte da una chat singola in cui i due interlocutori hanno scambiato almeno un messaggio (così da aver stabilito la connessione). Supponiamo che u1 voglia aggiungere u3 nella chat con u2, utilizzando il comando “*\a user3*”.
u1 attualmente potrebbe non aver una connessione con u3, quindi chiede la porta di ascolto di u3 al server (codice “**7**”) e una volta ricevuta vi ci stabilisce una connessione TCP. u1 invia ad user3 codice “**3**”, per indicargli che lo sta invitando in un gruppo, inoltre gli invia nome gruppo, il suo username e gli username di tutti i partecipanti al gruppo.
u1 invia agli altri membri del gruppo, un codice “**2**” per indicare di inserire un nuovo membro nel gruppo e invia il nome del gruppo nel quale inserirlo e l’username di u3.
u3 ricevute le informazioni necessarie, può ora inviare un messaggio speciale “join group”. In questo modo, se gli altri membri del gruppo non hanno una connessione stabilita con u3, il messaggio speciale sarà un “primo messaggio” e quindi proprio come se fosse un “primo messaggio” privato, questo passerà per il server e avverrà lo scambio di informazioni sul destinatario che si concluderà con una richiesta di connessione TCP da parte di u3.

● Condivisione di file



Per inviare un messaggio in una chat, singola o di gruppo che sia, un utente (supponiamo u1) deve utilizzare il comando “*share nome_file*” nella schermata della chat in questione. Ricordiamo che nel caso di invio un file, è necessario che il destinatario sia online.
In caso di chat singola, supponiamo che u1 voglia inviare un file a u2.
U1 invia un codice “**4**” per avvisare u2 dell’imminente arrivo di un file.
L’idea di base è che se un file è grande, questo deve essere inviato in più chunk. u1 quindi legge il file per un max di BUFFER_SIZE byte per volta e li invia a u2, questo viene ripetuto finchè il file non viene inviato tutto. Quando il ricevitore riceve “*end-of-file*” capisce che il file è terminato e che quindi non ha più nulla da ricevere.
Nel caso di chat di gruppo, il procedimento è lo stesso, solo che u1 dovrà ripetere lo stesso procedimento di invio per N utenti.