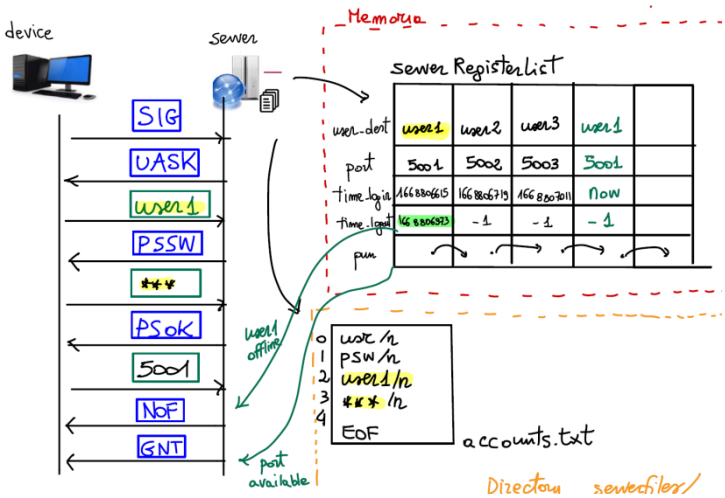
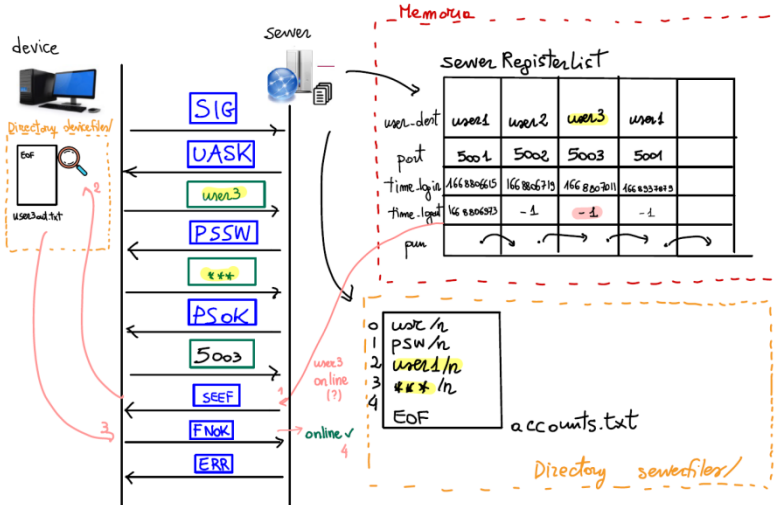
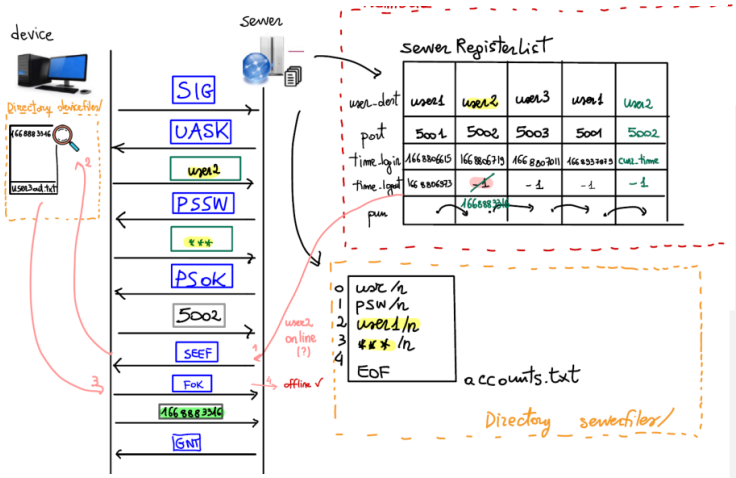
- Il server riesce a stabilire quando un device è offline/online scorrendo la lista serverRegisterList. Questa contiene un elemento per utente; se il campo timestamp_logout di un elemento è diverso da -1, allora sicuramente l'utente è offline. Altrimenti, solamente dal punto del server l'utente è online. Infatti, potrebbero essersi verificate disconnessioni improvvise.



Utente offline si riconnette al servizio



Utente già online: login rifiutato



Nuova *in* dopo disconnessione improvvisa

Il device è pronto per accettare connessioni solo dopo aver fatto login.

È più facile per il server recapitare il primo messaggio: si accorge se il device è ancora offline sulla porta prevista dal registro da ciò che ritorna connect().

In caso di disconnessione improvvisa, nessun altro device può connettersi alla porta sulla quale c'è stata la disconnessione inattesa fintantoché il device che si è disconnesso senza fare out non esegue un nuovo login (in quel caso il server se ne accorge e registra l'istante di disconnessione salvato in locale dal peer.

- *Nota*: Gli utenti user1, user2, user3 sono già registrati con password rispettivamente: "user1", "user2", "user3".

- Scambio dei messaggi in chat: scelto il protocollo **BINARY**

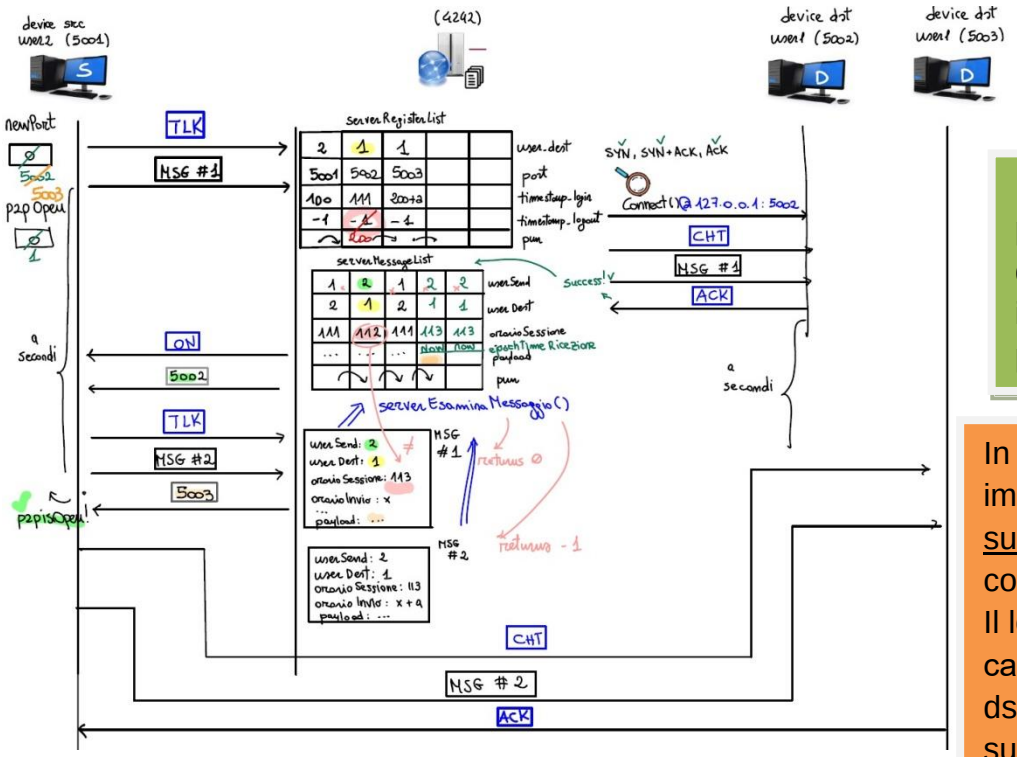
Maggiore segretezza dei messaggi che fluiscono in rete, minore occupazione di banda.

Per serializzare i timestamp in network order ho scelto di rappresentarli su 32 bit dato che per mantenere i 64 bit usati da time_t avrei dovuto ridefinire una struct con due campi, uno per i 32 bit più significativi, l'altro per i rimanenti bit. Avrei dovuto anche chiamare due volte htonl() ad ogni invio, rendendo più pesante il codice.

- Protocollo di trasporto: **TCP**

Il messaggio in chat arriva integro, c'è la certezza che arrivi, e che arrivi in ordine. Una volta che il messaggio arriva al dst, il device src stampa facilmente "(**)" vista l'affidabilità del protocollo di trasporto. Anche i messaggi bufferizzati sono mantenuti nell'ordine di arrivo.

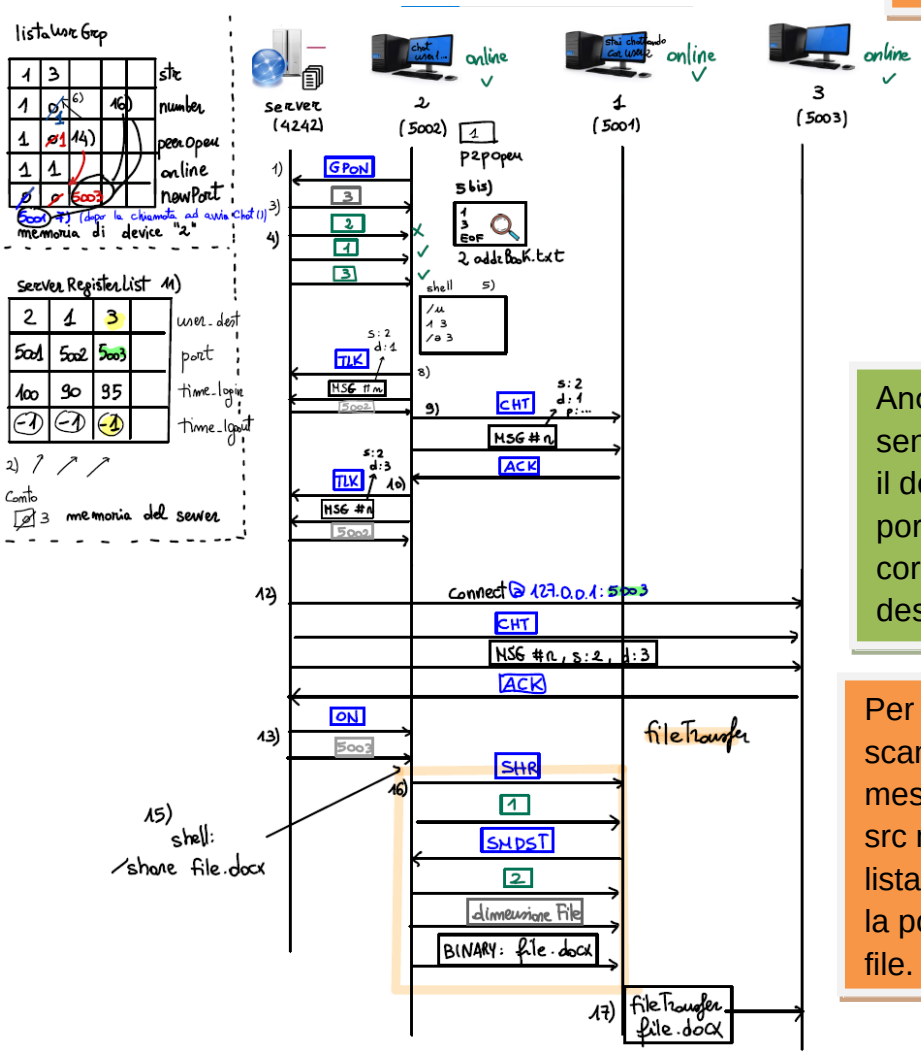
In caso di eccessivo traffico (e.g.: grande numero di utenti che richiedono di accedere al servizio mentre le chat sono in corso) si potrebbe avere un abbassamento del throughput, con tempi di consegna del messaggio superiori.



- La figura illustra come avviene lo scambio di messaggi in chat.

Le chat sono in grado di proseguire anche in caso di disconnessione del server e in caso di cambio di porta da parte del device destinatario.

In caso di disconnessione improvvisa, i messaggi bufferizzati sul server sono contrassegnati come recapitati.
Il loro timestampRicezione viene cambiato con “-1” quando device dst effettua un nuovo login successivo alla disconnessione improvvisa



- La figura illustra il funzionamento delle chat di gruppo e come ho realizzato l’invio dei file senza coinvolgere il server.

Anche durante l’invio dei file senza il contributo del server, il device src capisce se alla porta salvata in memoria corrisponde lo stesso destinatario.

Per inviare un file deve essere scambiato almeno un messaggio, altrimenti il device src non vede riempita la listaUsrGroup e non conosce la porta target a cui inviare il file.

Importante: Per il comando *signup* poiché da specifica non è indicata alcuna *srv_port*, ho supposto l’esistenza di una porta dedicata sul server (127.0.0.1:**3125**) atta ad accettare operazioni di creazione di nuovi account.