“IPortbook”
Progetto SETI 2025/2026

Attenzione: Prendetevi il tempo necessario per leggere attentamente TUTTO il documento nei minimi dettagli. Rileggete più volte, anche in gruppo. Non lanciatevi immediatamente nella realizzazione a meno che non abbiate già familiarità con lo sviluppo di reti e la comunicazione tra applicazioni. Quindi create dei diagrammi, immaginate scenari di comunicazione tra entità, cercate di analizzarne la portata, i problemi, le informazioni di cui le entità hanno bisogno nel corso della loro vita, ecc. Cercate innanzitutto di astrarvi dai problemi tecnici.

Nota: Nel documento il segno ␣ rappresenterà un semplice carattere di spaziatura (ASCII 32). Inoltre, i messaggi in circolazione sono indicati tra parentesi quadre, **le parentesi quadre non fanno parte del messaggio**. Alla fine del documento viene descritto in modo preciso il formato delle parti che costituiscono i messaggi.

Introduzione

Lo scopo di questo progetto è quello di programmare un sistema che consenta a diverse entità clienti di stabilire relazioni e comunicare tra loro. Il funzionamento si ispira agli attuali social network.

In questo sistema esistono due tipi di entità:

1. I server
2. I clienti

Intuitivamente, i clienti possono registrarsi sul server, stringere amicizie e scambiarsi messaggi. Il server dispone dell’elenco di tutti i suoi clienti e delle loro amicizie, ha il compito di inviare notifiche ai clienti e trasmettere i messaggi tra i clienti.

Il funzionamento è il seguente: il server gestisce per ciascuno dei suoi clienti un elenco di flussi. Un flusso può essere una richiesta di amicizia o un messaggio da un altro cliente o una risposta a una richiesta di amicizia o un messaggio di flood. Ad ogni nuovo flusso creato, il server invia una notifica sotto forma di messaggio UDP al cliente interessato, codificato su 3 carattere e nella forma `[YXX]` dove `Y` è un codice per il flusso in questione e `XX` è la codifica in esadecimale in *litte-endian* del numero di flussi non consultati. Il cliente connesso può consultare i propri flussi grazie al comando `CONSU`.

L’obiettivo di questo progetto sarà quello di programmare tali entità nel rispetto del protocollo corrispondente.

Descrizione dettagliata del protocollo

I clienti

Le caratteristiche di un cliente sono le seguenti:

- Un identificativo (di 8 caratteri alfanumerici)
- Una porta UDP (inferiore a 9999) per ricevere le notifiche
- Una password (un numero compreso tra 0 e 65535)

Quando un cliente è nuovo su un server, si registrerà fornendo le proprie informazioni. La porta UDP serve a ricevere notifiche per i nuovi flussi in arrivo (nuovi messaggi e nuove richieste di amicizia).

I server

Le caratteristiche di un server sono le seguenti:

- Una porta TCP (inferiore a 9999) di ascolto per comunicare con i client

Interazione cliente-server

Registrarsi. Per registrarsi a un server, un client si connette al server e gli invia un messaggio del tipo [REGIS␣id␣port␣mdp+++] dove id è l'identità del client , port è la sua porta UDP e mdp è la sua password. Il server risponde con un messaggio [WELCO+++] se accetta che quel cliente sia un nuovo cliente, oppure con un messaggio [GOBYE+++] se lo rifiuta (sta a voi capire perché un server potrebbe rifiutare un cliente). Il numero di clienti su un server sarà limitato a 100. Se il server accetta il cliente, quest'ultimo può continuare a comunicare, altrimenti il server chiude la connessione. Quando un server accetta il cliente, memorizza ovviamente il suo IP e le sue caratteristiche per poter comunicare con lui. Un cliente accettato può quindi eseguire le seguenti operazioni: ottenere l'elenco dei clienti registrati, inviare richieste di amicizia, inviare messaggi ai propri amici, inviare messaggi di flooding e consultare i propri flussi

Connetersi. Quando un client accettato da un server si è disconnesso e desidera riconnettersi, si connette tramite TCP e invia il messaggio [CONNE␣id␣mdp+++] con la propria identità id e la propria password mdp. Se il server lo riconosce, lo accetta inviandogli un messaggio [HELLO+++] oppure lo rifiuta inviandogli un messaggio [GOBYE+++] e chiudendo la connessione.

Le richieste di amicizia. Un cliente può chiedere di diventare amico di un altro cliente. Per farlo, invia il seguente messaggio: [FRIE?␣id+++] dove id è l'identità del cliente con cui desidera diventare amico. Il server risponde quindi con il messaggio [FRIE>+++] per comunicare che ha trasmesso la richiesta di amicizia sotto forma di flusso oppure [FRIE<+++] se non conosce il cliente id. Nel primo caso, il server invia una notifica sulla porta UDP del cliente id sotto forma di messaggio [0XX] e aggiunge al flusso di id la richiesta di amicizia.

L'invio di messaggi. Un cliente può inviare un messaggio ad un amico. Per farlo, invia [MESS?␣id ␣mess+++] dove id corrisponde all'identità dell'amico con cui si desidera comunicare e mess corrisponde al messaggio ed è una stringa di caratteri di massimo 200 caratteri. Quando il server ha ricevuto il messaggio, risponde inviando [MESS>+++] per segnalare che ha trasmesso il messaggio a id o [MESS<+++] per segnalare che non ha trasmesso il messaggio, un motivo potrebbe essere che id non è amico del cliente che ha inviato il messaggio (Nel primo caso, il messaggio viene quindi aggiunto all'elenco dei flussi che id deve consultare). Quando un client riceve un messaggio di questo tipo, il server gli invia sulla sua porta UDP una notifica del tipo [3XX] . In questo modo il client potrà consultare il messaggio quando consulterà le sue notifiche. (Il server memorizza quindi sotto forma di flusso per ogni cliente i messaggi che gli vengono inviati fino a quando questi non richiede di leggerli).

Flood. Un cliente può inviare un messaggio a tutti i suoi amici e agli amici dei suoi amici, agli amici degli amici dei suoi amici ecc. A tal fine, il cliente invia un messaggio [FLOO?␣mess+++] dove mess è il messaggio inviato di massimo 200 caratteri. Si noti che in questo caso i messaggi di flooding non sono mai lunghi. Il server risponde quindi con [FLOO>+++] per indicare che ha trasmesso il messaggio. Ai clienti interessati, il server invia sulla loro porta UDP una notifica del tipo [4XX] e aggiunge il messaggio di flooding al flusso di tutti questi client

L'elenco dei clienti. Un client può richiedere di visualizzare l'elenco dei client di un server inviando una richiesta del tipo [LIST?+++], alla quale il server risponde inviando prima un messaggio [RLIST␣num-item+++] dove num-item è il numero di client presenti nell'elenco. Successivamente invia num-item messaggi del tipo [LINUM␣id+++] con l'id di ciascuno dei clienti

Consultazione. Un cliente può richiedere di consultare l'elenco dei propri flussi. A tal fine, invia un messaggio del tipo [CONSU+++] che gli consente di consultare (e quindi di rimuovere dall'elenco uno dei flussi). Il server risponde quindi nel modo seguente, a seconda dei casi :

- Se si tratta di un flusso corrispondente al messaggio di un cliente, il server invia [SSEM>␣id␣mess+++] dove id è l'identità del cliente che ha inviato il messaggio e mess è una stringa di caratteri di massimo 200 caratteri contenente il messaggio.
- Se si tratta di un flusso corrispondente a un messaggio di inondazione, il server invia [OOLF>␣id␣mess+++] dove id è l'identità del cliente che ha inviato il messaggio e mess è una stringa di caratteri di massimo 200 caratteri contenente il messaggio.
- Se si tratta di una richiesta di amicizia, il server invia [EIRF>␣id+++] per segnalare che c'è una richiesta di amicizia proveniente dal client id. Il client risponde quindi con [OKIRF+++] se accetta la richiesta o con [NOKRF+++] se la rifiuta. Il server risponde quindi con [ACKRF+++] e invia una notifica UDP del tipo [1XX] al client che ha effettuato la richiesta di amicizia se la richiesta è stata accettata o [2XX] se la richiesta è stata rifiutata e aggiunge un flusso contenente la risposta all'elenco dei flussi del client id.
- Se si tratta di un'accettazione o di un rifiuto di una richiesta di amicizia, il server invia il messaggio [FRIEN␣id+++] per segnalare che id accetta una richiesta o, se si tratta di un rifiuto, il server invia il messaggio [NOFRI␣id+++].
- Se non ci sono più flussi da consultare, il server invia [NOCON+++].

Quindi, ogni volta che il client desidera consultare uno di questi flussi, invia il messaggio [CONSU+++]. I flussi trasmessi vengono quindi cancellati.

Disconnettersi. Quando un cliente desidera disconnettersi, invia un messaggio [IQUIT+++] a cui il server risponde con [GOBYE+++] e chiude la connessione.

Esempio di esecuzione. Supponiamo che ci siano due clienti bob12345 e alice123 (che chiameremo Alice e Bob). Alice è già registrata su un server e ha consultato tutti i suoi flussi. Bob si connette per la prima volta al server e si registra inviando un messaggio [REGIS␣bob12345␣07878␣1010+++] (dove 1010 corrisponde a due byte contenenti ciascuno il numero intero 10), il server accetta inviando [WELCO+++]. Bob rimane quindi connesso. Alice può essere connessa o meno. Bob invia una richiesta di amicizia inviando al server il messaggio [FRIE?␣alice123+++]. Il server invia quindi alla porta UDP di Alice un messaggio [010] (il primo 0 per indicare che ha ricevuto una richiesta di amicizia, i due caratteri 1 e 0 per indicare che ha un flusso da consultare). A questo punto può connettersi al server e inviare il messaggio [CONSU+++] per consultare i suoi flussi. Poiché ha un solo flusso in attesa, ovvero la richiesta di amicizia, il server

le invia [EIRF>␣bob12345+++]. Supponiamo che lei accetti questa richiesta di amicizia: risponderà con [OKIRF+++], il server confermerà inviando [ACKRF+++] e invierà a Bob sulla sua porta UDP un messaggio [110] il primo carattere 1 significa che qualcuno ha accettato una delle sue richieste di amicizia e i caratteri 1 e 0 significano che ha un flusso da consultare (10 essendo la codifica su in esadecimale di 1 in little endian). A questo punto (se è ancora connesso) può inviare [CONSU+++] al server che invierà il messaggio [FRIEN␣alice123] indicando che Alice ha accettato la sua richiesta di amicizia.

Specifiche relative alla forma dei messaggi.

Affinché i vostri progetti possano comunicare tra loro o con altri, è importante che tutti i messaggi rispettino una forma ben precisa. In questa sezione descriviamo in modo più dettagliato il formato di ciascuno dei messaggi.

Specifiche dei messaggi TCP.

Come avrete notato, i primi cinque byte di un messaggio serviranno a codificare in una stringa di caratteri corrispondente al tipo di messaggio. I tipi di messaggi sono: REGIS, WELCO, GOBYE, CONNE, HELLO, FRIE?, FRIE>, FRIE<, MESS?, MESS>, MESS<, FLOO>, FLOO?, LIST?, RLIST, LINUM,CONSU, SSEM>, OOLF>, EIRF>, OKIRF, NOKRF, ACKRF, FRIEN, NOFRI e NOCON.

Ogni messaggio TCP termina inoltre con tre byte contenenti la codifica in stringa di caratteri del segno aritmetico per l'addizione (ASCII 43).

Ecco ora le caratteristiche di ciascuno dei campi contenuti nei messaggi :

- `id` è codificato su 8 byte e contiene la stringa di caratteri corrispondente all'identificativo di un cliente (l'identificativo non può contenere tre + consecutivi).
- `port` è codificato su 4 byte e contiene la stringa di caratteri corrispondente al numero di porta (completata con degli 0 all'inizio se necessario), ad esempio per la porta 52, la codifica sarà la stringa di caratteri 0052.
- `mdp` è codificato su due byte e corrisponde alla codifica `little-endian` della password.
- `mess` è una stringa di caratteri che non contiene tre + consecutivi e la cui lunghezza è al massimo di 200 caratteri.
- `num-item` è codificato su 3 byte e contiene la stringa di caratteri corrispondente al numero di clienti nell'elenco (completata con degli 0 all'inizio se necessario), ad esempio per 8 clienti la codifica sarà la stringa di caratteri 008.

Specifiche dei messaggi UDP

I messaggi UDP corrispondenti alle notifiche dai server ai client sono composti da tre caratteri e hanno sempre la seguente forma [YXX] dove Y è un carattere contenente il codice della notifica e XX è la codifica in esadecimale su due caratteri in *little endian* del numero di flussi non consultati dal client.

Implementazione

La realizzazione dovrà essere necessariamente in C. Un gruppo dovrà programmare almeno un client ed un server che eseguano l'intero protocollo

È indispensabile rispettare scrupolosamente le specifiche fornite nell'oggetto e i formati dei messaggi. Qualsiasi violazione sarà giudicata molto sfavorevolmente!

Creeremo un forum su Aulaweb per consentirvi di comunicare tra voi e con noi, ad esempio per segnalare imprecisioni nell'argomento o per segnalare che avete un'entità funzionale e dare così l'opportunità ai vostri colleghi di fare test insieme a voi. **Qualsiasi richiesta relativa al progetto dovrà passare attraverso il forum.** Fate attenzione a non fidarvi delle voci, l'unica fonte di informazioni affidabile sarà il forum su Aulaweb! In caso di dubbi, non esitate a pubblicare la vostra domanda!

La comunicazione verbale tra i gruppi non solo è consentita, ma anche incoraggiata, tuttavia è **severamente vietato** scambiarsi il codice; ciò sarebbe considerato plagio e quindi giudicato severamente. Le discussioni devono quindi riguardare solo il funzionamento del protocollo e la sua interpretazione, o i formati dei messaggi; è quindi meglio evitare di dare troppe indicazioni su come codificare le funzionalità.

Un **programma che si esegue non è sufficiente**! È necessario scrivere un programma che si comporti come indicato; se non comunica con il programma scritto da altri, significa che non avete compreso cosa sia la programmazione di rete.

Per l'orale, sarà **necessario prevedere una modalità di funzionamento verbosa** in cui sullo schermo siano disponibili informazioni sufficienti per comprendere cosa sta succedendo (visualizzazione dei messaggi in circolazione, ecc.).

I vostri programmi dovranno necessariamente poter essere eseguiti sui computer delle aule di laboratorio. Qualsiasi soluzione che non rispetti questo criterio sarà considerata non valida.

Non è richiesto di fornire un'interfaccia grafica; si raccomanda addirittura di astenersi dal crearla (qualunque sia la vostra opinione in merito). Non vi porterà alcun vantaggio, vi farà perdere tempo e non impressionerà in alcun modo gli esaminatori; e soprattutto **non rientra assolutamente nel programma di questo corso**, pertanto insistiamo: **nessuna interfaccia grafica**.

Il vostro progetto dovrà ovviamente essere robusto (cioè privo di errori) e dovrà essere in grado di gestire i messaggi errati senza bloccarsi.

Il progetto sarà realizzato da gruppi composti da **al massimo due** studenti. Ovviamente, tutti i membri di un gruppo dovranno lavorare e non è escluso che gli studenti dello stesso gruppo ottengano voti finali diversi.

La composizione dei gruppi dovrà essere inviata via e-mail all'indirizzo `arnaud.sangnier@unige.it` entro **mercoledì 17 dicembre 2025 alle ore 23:59**. Chiunque non abbia presentato un gruppo entro tale data rischia di non potere presentarsi all'orale.

Il progetto dovrà essere consegnato due giorni prima dell'orale, la cui data vi sarà comunicata in seguito. Anche le informazioni relative alla discussione (ordine di presentazione e istruzioni) saranno fornite in seguito.