Report Assignment 3

Baiardi Martina, Lombardini Alessandro

TASK 1

## Descrizione della FSM Server

Il server attende nello stato di *wait\_connection* sulla porta designata, una qualsiasi richiesta di connessione da parte di un client. Quando un client effettua una richiesta, dopo aver effettuato la connessione, il server passa allo stato di *wait\_hello\_message*, dove attende che venga inviato un hello message dal client.

Il client invia il messaggio e il server, che a seguito della ricezione, controlla che sia valido:

1. Se l’esito risulta **positivo,** invia un messaggio di conferma al client e passa allo stato di *wait\_probe\_message*. In questo caso il server passa allo stato di *wait\_probe\_message*, dove attende i messaggi di probe da parte del client.
2. Se l’esito risulta **negativo**, invia un messaggio di errore al client e chiude la connessione. Lo stato del server ritorna quello iniziale di *wait\_connection*.

All’arrivo di un nuovo messaggio, il server controlla la sua validità come nello stato precedente:

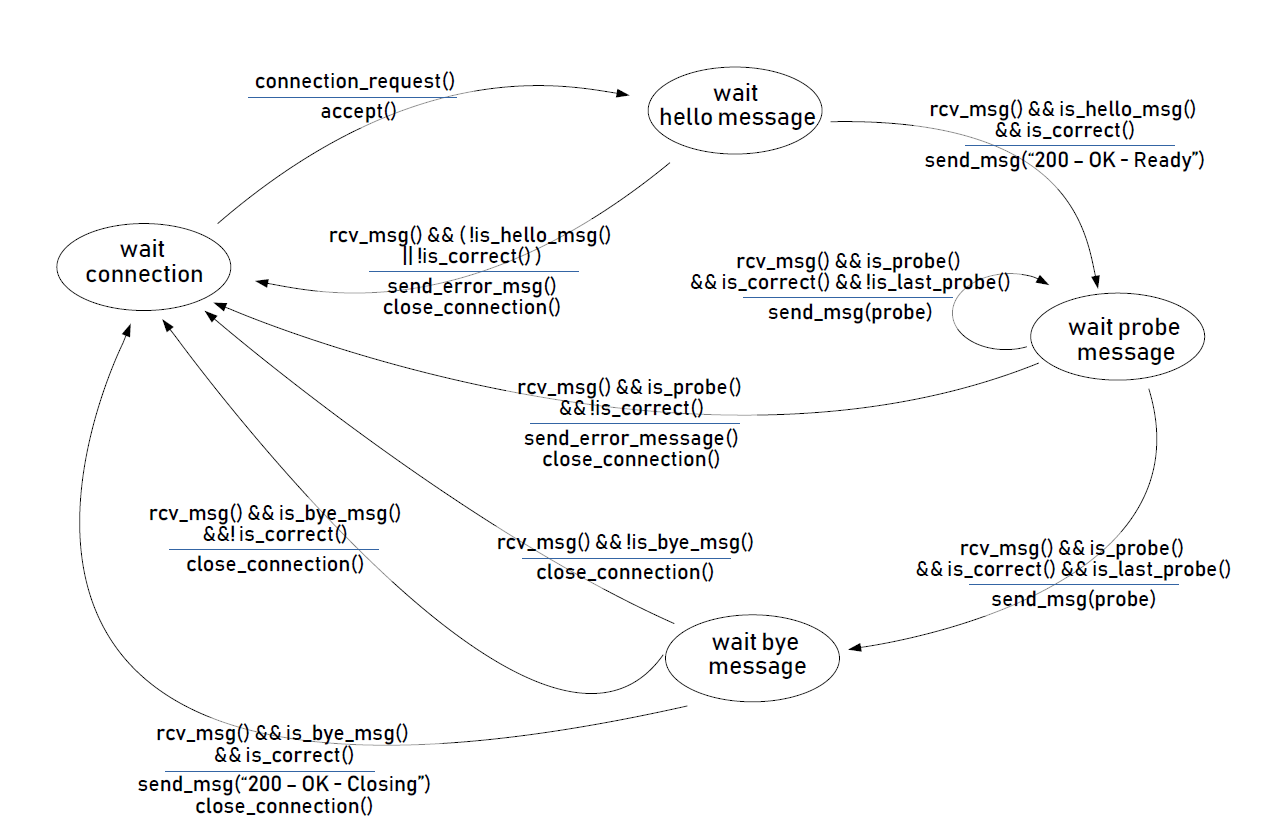
1. Se l’esito risulta **positivo,** invia indietro il messaggio ricevuto e attende di ricevere i messaggi di probe successivi;
2. Se l’esito risulta **negativo**, invia un messaggio di errore al client e chiude la connessione. Lo stato del server ritorna quello iniziale di *wait\_connection*.

Lo stato *wait\_probe\_message* non cambia finchè il messaggio ricevuto, oltre ad essere valido, non è l’ultimo fra quelli previsti, la quantità specificata nell’*hello message*.

Una volta arrivati tutti i messaggi di probe validi, lo stato cambia in *wait\_bye\_message*, ovvero il server si mette in attesa del *bye message.* Quando il server riceve un messaggio, controlla che questo sia effettivamente il *bye message* atteso:

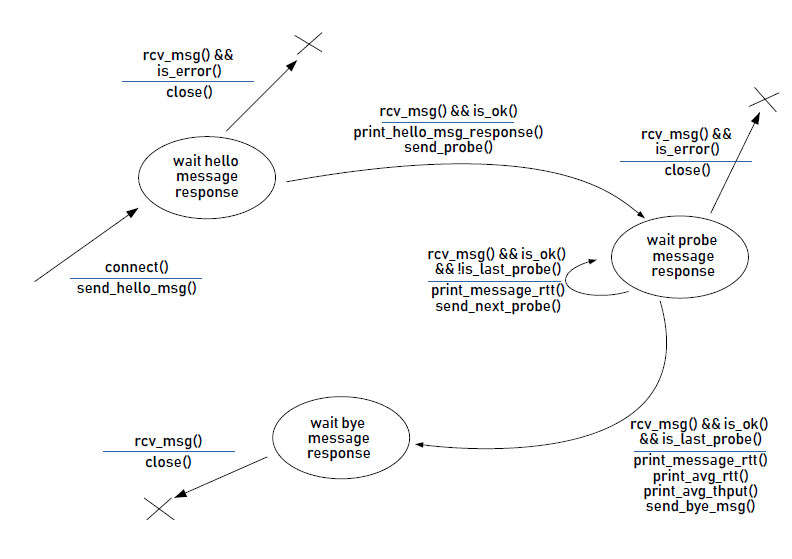
1. Se l’esito è **positivo**, invia un messaggio di conferma al client.
2. Se l’esito è **negativo**, invia al client un messaggio di errore.

Una volta inviata la risposta, il server procede con la chiusura della connessione e torna in ascolto di nuove richieste nello stato di *wait\_connection*.



## Descrizione della FSM del Client

Il client si connette al server e invia un *hello message*, posizionandosi nello stato di *wait\_hello\_message\_response*. Una risposta **negativa** implica un errore, dunque chiude la connessione e l’applicativo termina; una risposta **positiva** prevede lo spostamento nello stato di *wait\_probe\_message\_response* inviando un primo *probe message.* Se la risposta coincide con il messaggio inviato, il client può procedere con l’invio del probe successivo, altrimenti viene termianto l’applicativo. Non appena sono stati inviati un numero di *probe message* pari al valore indicato nell’*hello message* e per ciascuno quali ricevuto un echo back da parte del server, il client può cambiare stato in *wait\_bye\_message response*, inviando un *bye\_message* al server. In caso arrivi un messaggio di conferma positivo viene chiusa la connessione con successo, nel caso in cui il messaggio sia invece di errore, viene terminato il programma con un errore, chiudendo anche in questo caso la connessione.



# TASK 2

## Client behaviour

Il client si preoccupa di eseguire la richiesta per instaurare una connessione TCP con il server. Devono essere dunque forniti, all’atto dell’esecuzione, l’indirizzo IP e la porta su cui il server è in ascolto.

L’utente, per esprimere le proprie preferenze riguardo il servizio che vuole utilizzare, può modificare il file *init.conf.* Al suo interno devono essere presenti i dettagli riguardanti la misura da effettuare. Il formato della stringa contenuta al suo interno deve essere:

*<measure\_type><sp><n\_probes><sp><msg\_size><sp><server\_delay>*

*measure\_type:* il tipo di misura che su vuole effettuare; può essere ***thput*** o ***rtt***.

*n\_probes:* il numero di probe che desidera vengano inviati al server.

*msg\_size:* dimensione del payload dei messaggi di probe che vengono inviati al server.

*server\_delay*: tempo di attesa che deve svolgere il server prima di rispondere ai probe del client.

Il client legge queste informazioni e ne valuta la correttezza: se corrette, le estrae popolando una struttura dati interna, la quale viene tenuta aggiornata durante le diverse fasi d’esecuzione, se errate, il client non effettua richieste al server e termina.

La struttura dati è la seguente:

/\* Service structure definition \*/

typedef struct node {

char measureType[6]; /\* Service requested - RTT or THPUT \*/

int nProbes; /\* Number of probe messages to send \*/

int messageSize; /\* Size of probe message's payload \*/

int serverDelay; /\* Delay of server probe message echo (in milliseconds) \*/

int serverFD; /\* File descriptor of socket \*/

int phaseNumber; /\* 1: Hello phase

2: Probe phase

3: Bye phase \*/

} serviceNode;

serviceNode service;

Dove *phaseNumber* è una rappresentazione numerica delle fasi in cui si trova il client attualmente.

Se il contenuto del file è valido, il client esegue una richiesta di connessione al server e invia l’*hello message,* entrando poi nello stato di attesa della risposta da parte del server. Se tale risposta è positiva si prosegue nella creazione del messaggio di probe, altrimenti si chiude direttamente l’applicativo segnalando l’errore. Il *probe message* viene ricreato ad ogni ciclo di invio/ricezione, incrementando il suo **sequence number**, per permettere di far sapere al server lo stato di avanzamento dell’invio dei probe. Il payload viene creato inizialmente, formando una stringa con un numero di caratteri tale da ottenere un payload delle dimensioni richieste*.*

*qui*

Viene creata quindi una stringa sufficientemente lunga di caratteri *a.* A seguito dell’invio del messaggio viene fatto immediatamente partire un timer, il quale viene stoppato a seguito della ricezione della risposta. Viene quindi valutato il tempo in nanosecondi, convertito in millisecondi, e stampato. Ciascun ciclo mi genera un tempo: terminati i *probe message* viene calcolata una media. In base alla richiesta fornita dall’utente viene stampato il semplice valore RTT, come media degli RTT ottenuti, oppure viene valutata la dimensione del *probe message* in kilobit e l’RTT in secondi, e viene calcolato il throughtput. Viene poi spedito il messaggio di bye e se ne attende la risposta. Una volta ricevuta, la connessione viene chiusa e l’esecuzione dell’applicativo terminata.

Il server si pone in attesa di connessioni TCP sulla porta indicata dall’utente all’avvio dell’eseguibile. Ha a sua disposizione, per la gestione delle misurazioni, una struttura dati in cui vengono inserire le informazioni utili. Importantissimo, anche in questo caso, è il campo *phaseNumber,* che può assumere i valori: uno, due, tre e quattro, rispettivamente *wait\_connection*, *wait\_hello\_message*, *wait\_probe\_message*, *wait\_bye\_message*. Questi valori rappresentano i diversi stati in cui il server può trovarsi, prima fra tutte lo stato di attesa di connessione. Il server analizza la forma sintattica e semantica di tutti i messaggi che gli arrivano: in base allo stato in cui esso si trova si aspetta un certo tipo di messaggio, che rispetti il protocollo definito e che rispetti i dettagli che il client ha richiesto per effettuare la misurazione. Esistono tre pattern diversi che il server può accettare: uno per l’*hello message*, uno per il *probe message* e uno per il *bye message.* In funzione del valore di *phaseNumber* il server verifica che il messaggio arrivato soddisfi il pattern relativo allo stato in cui si trova. Stabilita una connessione, si passa allo stato successivo, l’attesa di ricezione dell’*hello message.* Ricevuto, si passa alla prima fase di analisi: il messaggio di hello ha una forte valenza semantica, si effettua quindi una verifica della sintassi del messaggio e vengono memorizzati nella struttura dati i valori dei campi. Se il controllo da buon esito viene inviato un messaggio di esito positivo e si passo allo stato successivo; in caso contrario il messaggio riporta un esito negativo, la connessione viene chiusa e si riposiziona nello stato iniziale. Si passa quindi allo stato successivo, in attesa di un *probe message.* Un dettaglio interessante è l’attributo *probeSequenceNumberAwaited*, presente nella struttura dati del server. Questo attributo indica il numero di sequenza che il server si aspetta di trovare all’interno del prossimo *probe message.* Arrivato il messaggio si verifica che sia esattamente come previsto, carattere per carattere. Se il messaggio non soddisfa il pattern viene segnalato al client, la connessione viene chiusa e si riposiziona nello stato iniziale. Terminati i *probe message* previsti, si passa all’ultimo stato: arrivato il messaggio si invia un feedback al client solo nel caso il messaggio sia corretto. Corretto o non corretto, il server termina la connessione e ritorna allo stato iniziale.

La ricezione di client e server prevede frammenti di codice utile per gestire più letture dal buffer di recezione con lo scopo di unire frazioni di testo che appartengono allo stesso messaggio ma che vengono lette in letture differenti. Per quanto riguarda il server, continua nella lettura fin quando non incontra un carattere *\n,* in quanto i messaggi del client terminano sempre con un *\n* finale*.* Il server invece invia messaggi che non prevedono l’uso finale del carattere *\n*. Per questo motivo il client, oltre a verificare che non sia arrivato un *\n,* verifica che il messaggio arrivato fino ad ora non coincida con uno dei messaggi previsti dal comportamento del server.