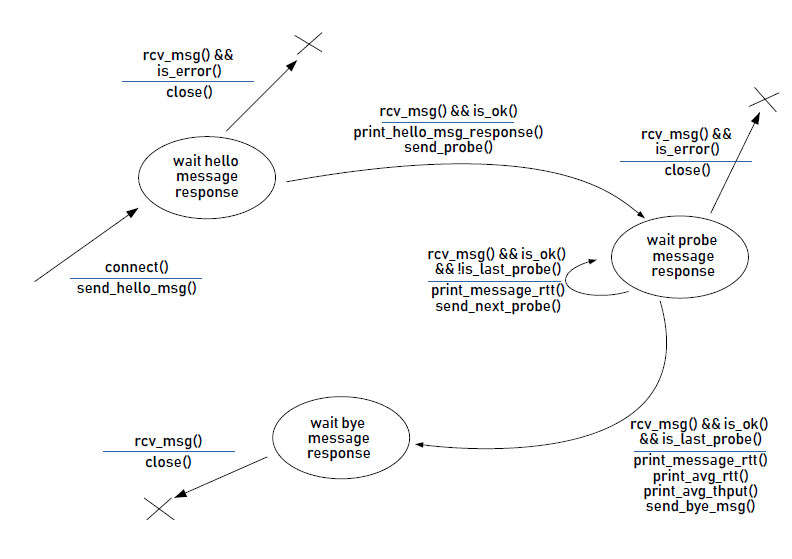


Il server attende, al di fuori delle fasi necessarie per ottemperare alle misurazioni, nello stato *wait\_connection*: rimane in attesa di richieste di connessione TCP. Quando un client vuole effettuare una misurazione esegue una richiesta di connessione; il server si pone dunque nel suo successivo stato, *wait\_hello\_message*. L’attesa questa volta è per un *hello message.* Il server, a seguito della ricezione, controlla che sia stato inviato un *hello message* valido; se l’esito risulta positivo è predisposto l’invio di un messaggio di conferma, diversamente la risposta è di errore, la connessione viene chiusa e lo stato ritorna quello iniziale. In caso di messaggio valido il server passa al nuovo stato: *wait\_probe\_message*. Si attende ora un *probe message.* Come per lo stato precedente, all’arrivo del messaggio questo viene controllato e si ottiene un esito, positivo o negativo. Se negativo invia un messaggio di errore, chiude la connessione e si riposiziona nello stato iniziale; se positivo rinvia al mittente il messaggio ricevuto. Lo stato non cambia se il messaggio ricevuto, oltre ad essere valido, non è l’ultimo fra quelli previsti, ovvero non ne sono stati ancora inviati un numero pari alla quantità specificata nell’*hello message*; si ha quindi uno stato con una transizione circolare. In caso non ci si aspetti più *probe message* lo stato cambia. Siamo ora nell’ultimo stato possibile: *wait\_bye\_message*, ovvero in attesa del *bye message.* Il server prevede una chiusura amichevole della connessione, e attende quindi il messaggio con cui il client ne fa richiesta. In fase di ricezione il messaggio viene verificato: se valido invia una conferma. Sia nel caso in cui esso sia sintatticamente corretto che non, la connessione viene chiusa e il server ritorna nello stato iniziale.



Il client attua delle procedure speculari a quelle appena descritte: si connette al server e invia un *hello message*. Si posiziona dunque nel suo primo stato: *wait\_hello\_message\_response*. Una risposta negativa implica un errore, dunque chiude la connessione e l’applicativo termina; una risposta positiva prevede l’invio di un primo *probe message* e lo spostamento in uno stato successivo: *wait\_probe\_message\_response*. Se la risposta coincide con il messaggio inviato abbiamo due possibilità: non sono ancora stati inviati un numero di *probe message* pari al valore indicato nell’*hello message*, quindi ne invia un altro e si riposiziona sul medesimo stato; oppure il *probe message* inviato era l’ultimo, dunque invia un *bye\_message* e si sposta sull’ultimo stato: *wait\_bye\_message\_response*. In questo caso può arrivare una risposta, nel caso di messaggio valido, come nessuna, nel caso contrario.

TASK 2

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 inoltre, per esprimere le proprie preferenze riguardo il servizio, deve modificare il file *init.conf.* Al suo interno devono essere presenti i dettagli riguardanti la misurazione da effettuare. Il formato della stringa contenuta al suo interno deve essere:

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

Il client ha il compito di leggerla, valutarne la correttezza sintattica e semantica, e, se corretta, estrarne le informazioni contenute. Le informazioni vengono memorizzate all'interno di una struttura dati che viene tenuta aggiornata durante le diverse fasi dell’esecuzione. Valore estremamente importante presente all’interno della struttura è il *phaseNumber*, valore numerico intero i cui valori possibili sono: uno, due e tre, definiti come *wait\_hello\_message\_response*, *wait\_probe\_message\_response* e *wait\_bye\_message\_response*. Ciascuno di questi valori rappresenta uno stato, in particolare uno stato di attesa di uno specifico messaggio in risposta ad uno precedentemente inviato. Il valore uno rappresenta l’attesa della risposta all’*hello message*, il due al *probe message* e il tre al *bye message*. Se il contenuto del file è valido, il client esegue una richiesta di connessione e invia l’*hello message,* del quale si attende la risposta da parte del server. Se positiva si prosegue nella creazione di un messaggio di probe, altrimenti si chiude direttamente l’applicativo segnalando l’errore. Il *probe message* viene ricreato ad ogni ciclo di invio/ricezione, unendo le sue varie componenti. Abbiamo un solo elemento che varia, il numero di sequenza dei messaggi. Il payload è creato inizialmente, formando una stringa con un numero di caratteri tale da ottenere un payload delle dimensioni richieste*.* 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.