Progetto di Reti

Simulazione del protocollo Go-Back-N con socket UDP

Justin Carideo
Matricola: 0001115610

24 maggio 2025

Indice

Capitolo 1

Introduzione

Il protocollo Go-Back-N ARQ è un protocollo di comunicazione che lavora a livello del data link e permette la trasmissione di pacchetti dati in modo affidabile per mezzo di una implementazione particolare della tecnica della finestra scorrevole.

1.1 Analisi del protocollo

Nel protocollo Go-Back-N abbiamo due entità *mittente* e *ricevente* che comunicano scambiandosi pacchetti tra loro. In tutto ciò bisogna tenere conto di:

- *numerazione dei pacchetti*: la numerazione sequenziale permette l'identificazione e il riordino
- *timer di ritrasmissione*: se non viene trasmesso il pacchetto giusto ci dovrà essere un timer che ritrasmetterà dall'ultimo arrivato
- *finestra di trasmissione del mittente*: il mittente dovrà mandare più di un pacchetto per mezzo di una finestra di lunghezza predefinita.

Ai fini pratici analizzeremo anche quando la dimensione della finestra di trasmissione è 1, portando l'implementazione del Go-Back-N a simulare il comportamento del protocollo Stop-and-Wait.

1.2 Obiettivi

L'obiettivo è quello di implementare il protocollo tramite l'uso di socket UDP, quindi la connessione sarà più veloce ma meno sicura rispetto a TCP. In

particolare, il tipo di protocollo è *connectionless*, quindi non stabilisce una connessione tra mittente e destinatario, ma invia i pacchetti senza alcun tipo di garanzia su ordine e integrità.
Gli obiettivi specifici dell'implementazione includono:

- Ci siano due entità **server** e **client** che comunicano tra loro
- Fare in modo che i pacchetti siano numerati
- Gestire la finestra di trasmissione
- Implementare una funzione di timeout che permetta la ritrasmissione dei pacchetti persi
- Simulare la perdita di pacchetti

Capitolo 2

Progettazione

2.1 Discussione sull'implementazione

Per l'implementazione ho voluto dividere il lavoro tra `client` e server, dove sono rispettivamente mittente e destinatario. Il client viene diviso in tre thread, che hanno i seguenti ruoli:

- Spedire i pacchetti in maniera sequenziale (`main`)
- Ricevere gli ACK da parte del server (`receive_ack`)
- Gestire il timeout e la ritrasmissione (`handle_timeout`)

Il timer viene trattato come una risorsa condivisa tra i thread, e viene avviato, arrestato o resettato a seconda dell'invio o ricezione di pacchetti.
La finestra di trasmissione è definita tramite una costante che stabilisce quanti pacchetti possono essere inviati senza attendere ACK. Per simulare la perdita dei pacchetti si è utilizzata una costante `LOSS_PROBABILITY`, che in combinazione con `random` permette a `send_pack` di non spedire in modo casuale alcuni pacchetti.

2.2 Diagramma di flusso

Per non mancare di generalità di seguito viene riportato il diagramma di flusso:

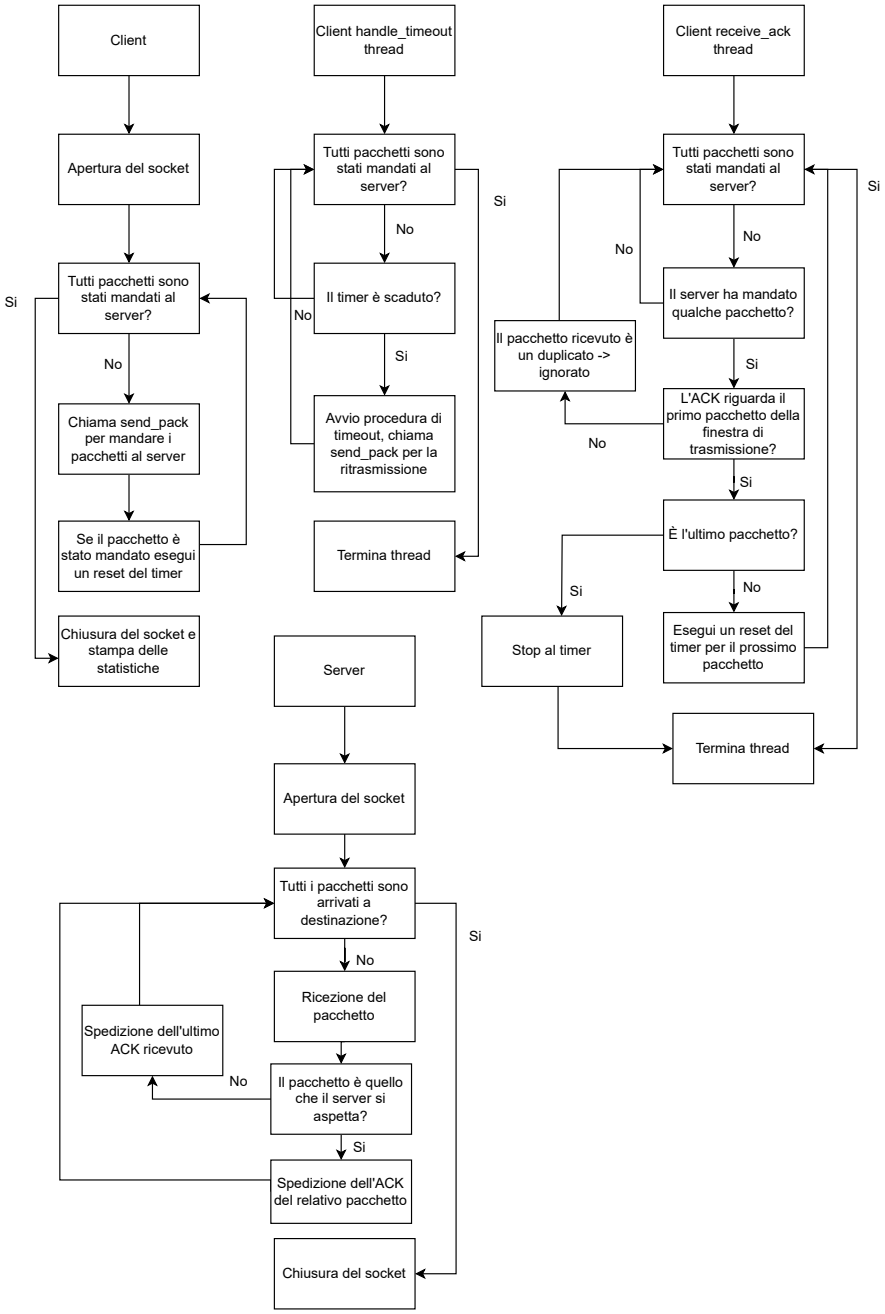

Figura 2.1: Diagramma di flusso

Capitolo 3

Conclusione

Alcune particolarità le possiamo trovare nel file di log. Se per esempio la dimensione della finestra di trasmissione è 1 si osserva che il controllo di flusso diventa quello di uno Stop-And-Wait, dove ad ogni pacchetto spedito aspettiamo il suo corrispettivo ACK prima di spedirne un altro.
Un altro dettaglio sottile riguarda la trasmissione del primo pacchetto. Se il primo non viene spedito e il thread di ricevimento dell'ACK prova a chiedere di quel pacchetto, succede che viene lanciata un'eccezione *WinError 10022*, che rappresenta il fatto che il server non abbia mandato nulla. Questo tipo di errore evidenzia l'inaffidabilità della comunicazione basata su UDP, poiché non viene garantita alcuna connessione o conferma di ricezione tra le due entità, al contrario del TCP che invece esegue un *Three-Ways-Handshake* per garantire connessione tra le entità.