

# OSI: Livello 4 (Transport)

---

Il livello transport è il cuore di tutta la gerarchia di protocolli. Il suo compito è di fornire un trasporto affidabile ed efficace dall'host di origine a quello di destinazione, indipendentemente dalla rete utilizzata:

- Servizi di livello transport:
  - servizi affidabili orientati alla connessione (tipici di questo livello);
  - servizi datagram (poco usati in questo livello).
- Specificare la QoS (Quality of Service) desiderata.
- Protocolli:
  - Controllo degli errori;
  - Controllo di flusso;
  - Riordino dei TPDU.
- Isolare i livelli superiori dai dettagli implementativi della subnet di comunicazione
- Primitive di definizione del servizio. Definiscono il modo di accedere ai servizi del livello:

Primitiva	TPDU spedito	Note
accept()	-	Si blocca finché qualcuno cerca di connettersi
connect()	conn.request	Cerca di stabilire una connessione
send()	dati	Invia dei dati
receive()	-	Si blocca finché arriva un TPDU
disconnect()	disconn.request	Chiede di terminare la connessione

Esempio di frammenti di codice di un'applicazione client-server:

CLIENT	SERVER
...	...
connect()	accept()
receive()	send ()
send()	receive()
...	...
disconnect()	

- Indirizzamento:

Quando si vuole attivare una connessione, si deve ovviamente specificare con chi la si desidera. Dunque, si deve decidere come è fatto l'**indirizzo di livello transport**, detto **TSAP address** (Transport Service Access Point address).

Tipicamente un TSAP address ha la forma:

**(NSAP address, informazione supplementare)**

Ad esempio, in Internet un TSAP address (ossia un indirizzo TCP o UDP) ha la forma:

**(IP address : port number)**

**dove IP address è il NSAP address, e port number è l'informazione supplementare.**

Questo meccanismo di formazione degli indirizzi dei TSAP ha il vantaggio di determinare implicitamente l'indirizzo di livello network da usare per stabilire la connessione.

In assenza di tale meccanismo, diviene necessario che l'architettura preveda un servizio per effettuare il mapping fra gli indirizzi di livello transport e i corrispondenti indirizzi di livello network.

FTP(control): 20; FTP(data): 21; SMTP: 25; POP3: 110; HTTP:80;

## Attivazione della connessione

Questa operazione sembra facile ma non lo è, perché la subnet può perdere o duplicare (a causa di ritardi interni) dei pacchetti sia durante la trasmissione sia durante l'attivazione.

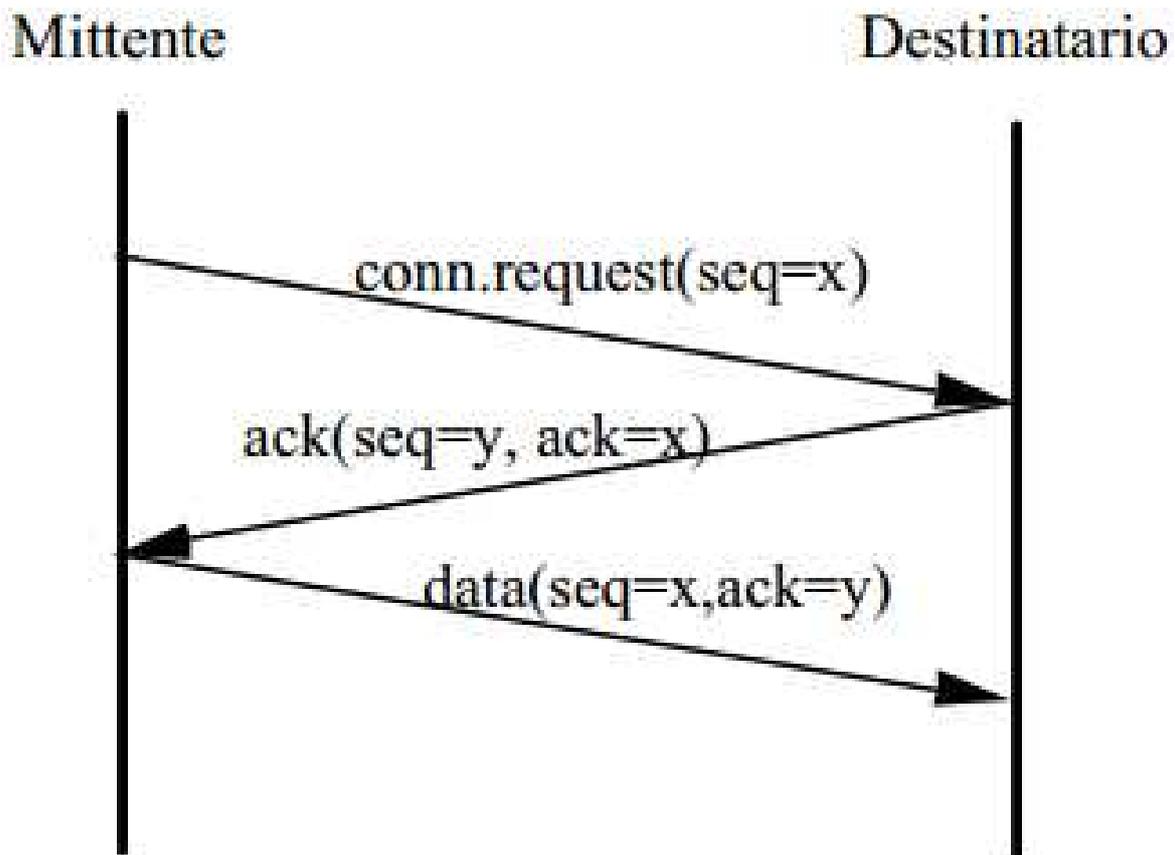
Per risolvere il problema dei duplicati relativi alla fase di attivazione della connessione esiste una soluzione detta **three-way handshaking** (Stretta di mano a tre vie - Tomlinson, 1975).

Il protocollo funziona così: 

1. il richiedente invia un TPDU di tipo **conn.request** con un numero **x** proposto come inizio della sequenza;
2. il destinatario invia un TPDU di tipo **ack** contenente:
  - a. la conferma di **x**;
  - b. la proposte di un proprio numero **y** di inizio sequenza;
3. il richiedente invia un TPDU di tipo dati contenente:
  - a. i primi **dati** del dialogo;
  - b. la conferma di **y**.

I valori **x** e **y** possono essere generati, ad esempio, sfruttando l'orologio di sistema, in modo da avere valori ogni volta diversi.

In assenza di errori, il funzionamento è il seguente:



Three-way handshake



## Rilascio di una connessione

Rilasciare una connessione presenta qualche piccolo problema. Ci sono due tipi di rilasci:

1. asimmetrico
2. simmetrico.

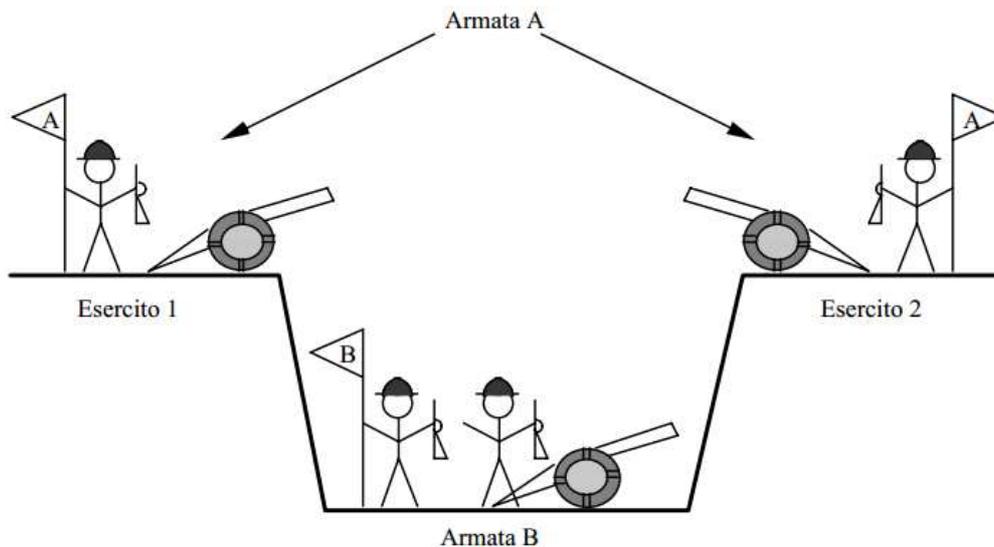
Nel primo caso (esemplificato dal sistema telefonico) quando una delle due parti "mette giù" si chiude immediatamente la connessione. Ciò però può portare alla perdita di dati, in particolare di tutti quelli che l'altra parte ha inviato e non sono ancora arrivati.

Nel secondo caso se invece le due entità vogliono essere d'accordo prima di rilasciare la connessione, un modo di raggiungere lo scopo potrebbe essere questo:



Semplice schema per il rilascio concordato della connessione

Purtroppo, le cose non sono così semplici: c'è un problema famoso in proposito, il **problema delle due armate**:



La definizione del problema è la seguente:

- i due eserciti che compongono l'armata A sono ciascuno più debole dell'esercito che costituisce l'armata B;
- l'armata A però nel suo complesso è più forte dell'armata B;
- i due eserciti dell'armata A possono vincere solo se attaccano contemporaneamente;
- i messaggi fra gli eserciti dell'armata A sono portati da messaggeri che devono attraversare il territorio dell'armata B, dove possono essere catturati.

Come fanno ad accordarsi gli eserciti dell'armata A sull'ora dell'attacco? Una possibilità è la seguente:

- il comandante dell'esercito 1 manda il messaggio "attacchiamo a mezzanotte. Siete d'accordo?";
- il messaggio arriva, un ok di risposta parte e arriva a destinazione, ma il comandante dell'esercito 2 esita perché non può essere sicuro che la sua risposta sia arrivata.

Si potrebbe pensare di risolvere il problema con un passaggio in più (ossia con un three-way handshake): l'arrivo della risposta dell'esercito 2 deve essere a sua volta confermato.

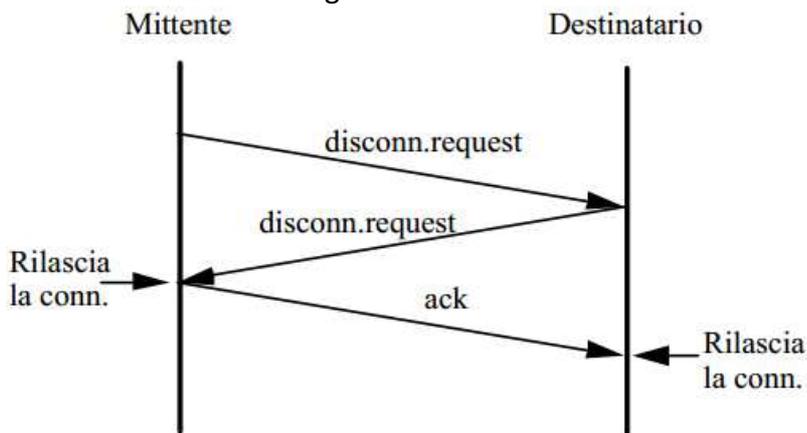
Ora però chi esita è il comandante dell'esercito 1, perché se tale conferma si perde, l'armata 2 non saprà che la sua conferma alla proposta di attaccare è arrivata e quindi non attaccherà.

Aggiungere ulteriori passaggi non aiuta, perché c'è sempre un messaggio di conferma che è l'ultimo, e chi lo spedisce non può essere sicuro che sia arrivato. Dunque, non esiste soluzione.

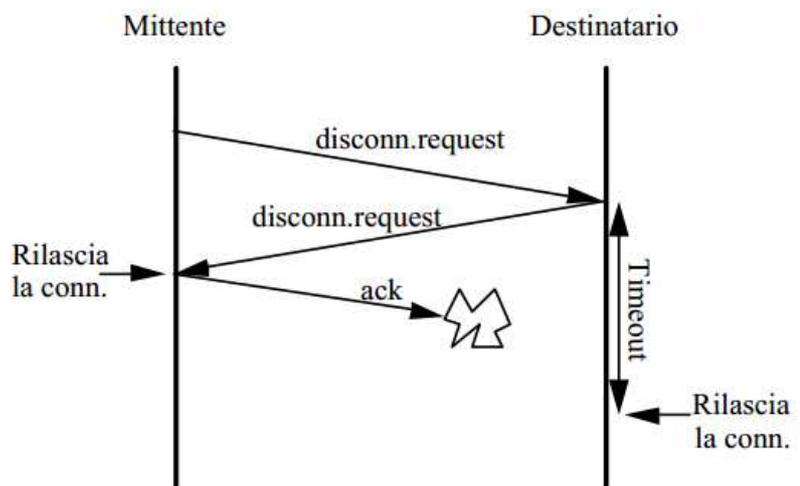
Se ora sostituiamo la frase "attacchiamo l'armata B" con "rilasciamo la connessione", vediamo che si rischia effettivamente di non rilasciare mai una connessione.

Un protocollo di tipo three-way handshaking arricchito con la gestione di timeout è considerato adeguato, anche se non infallibile:

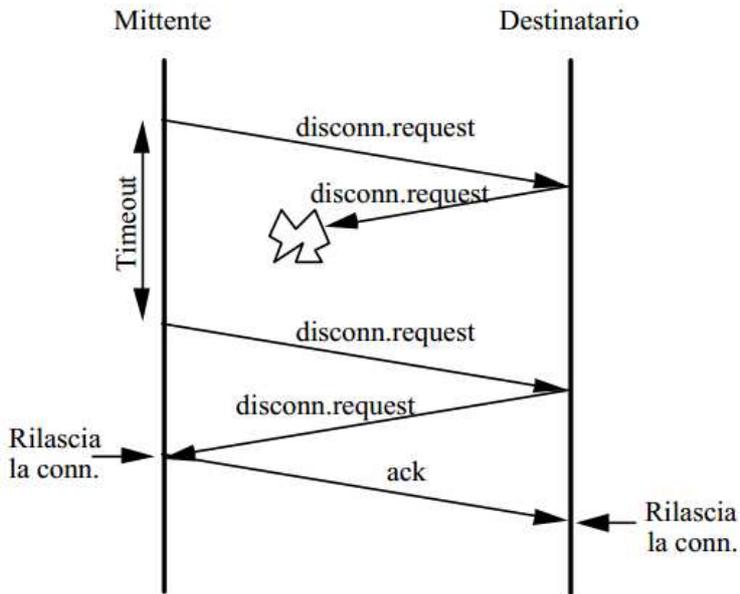
- il mittente invia un **disconn.request** e, se non arriva risposta entro un tempo prefissato (timeout), lo invia nuovamente per un massimo di n volte;
- appena arriva una risposta (disconn.request) rilascia la connessione in ingresso e invia un ack di conferma; se non arriva nessuna risposta, dopo l'ultimo timeout rilascia comunque la connessione in ingresso;
- il destinatario, quando riceve disconn.request, fa partire un timer, invia a sua volta un disconn.requeste attende l'ack di conferma. Quando arriva l'ack o scade il timer, rilascia la connessione in ingresso.



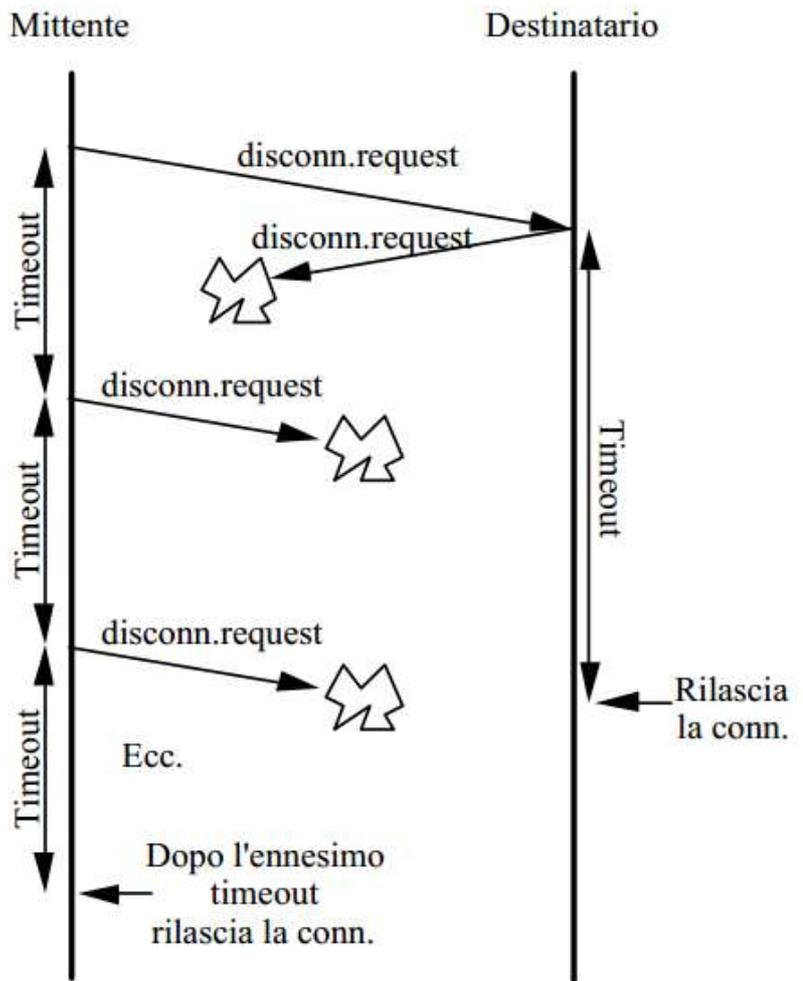
Rilascio concordato di una connessione transport



Perdita dell'ack durante il rilascio della connessione



Perdita della risposta alla proposta di chiusura durante il rilascio della connessione



Perdita di tutti i messaggi tranne il primo