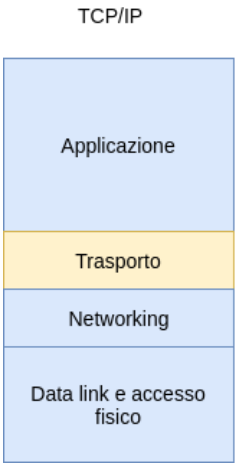
Appunti sul livello Trasporto

Livello Trasporto

In questo livello, i “messaggi” inviati tra gli host sono chiamati “**segmenti**”



Il livello 4 **estende le funzioni del livello 3 (IP)** fornendo un servizio di consegna più vicino al livello applicativo.

Rispetto al livello IP viene aggiunto:

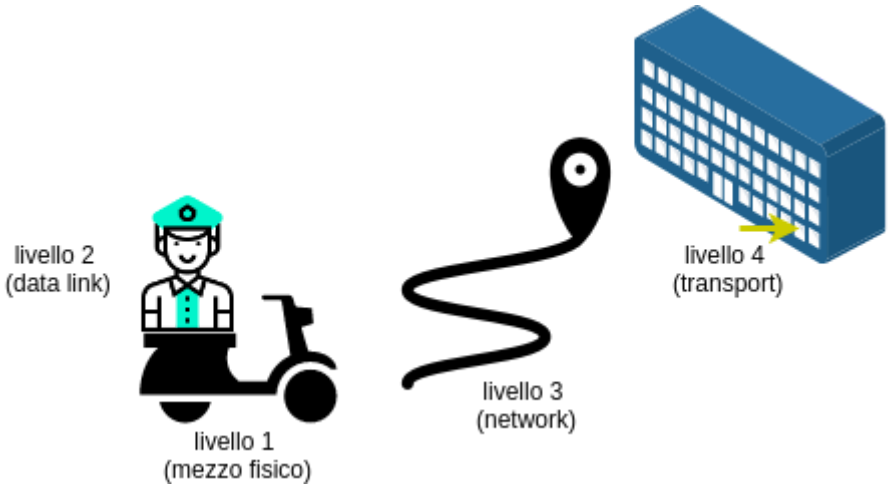
- Multiplazione/demultiplazione dei messaggi tra processi
- Rilevamento dell’errore

I protocolli implementati a livello trasporto sono

- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)

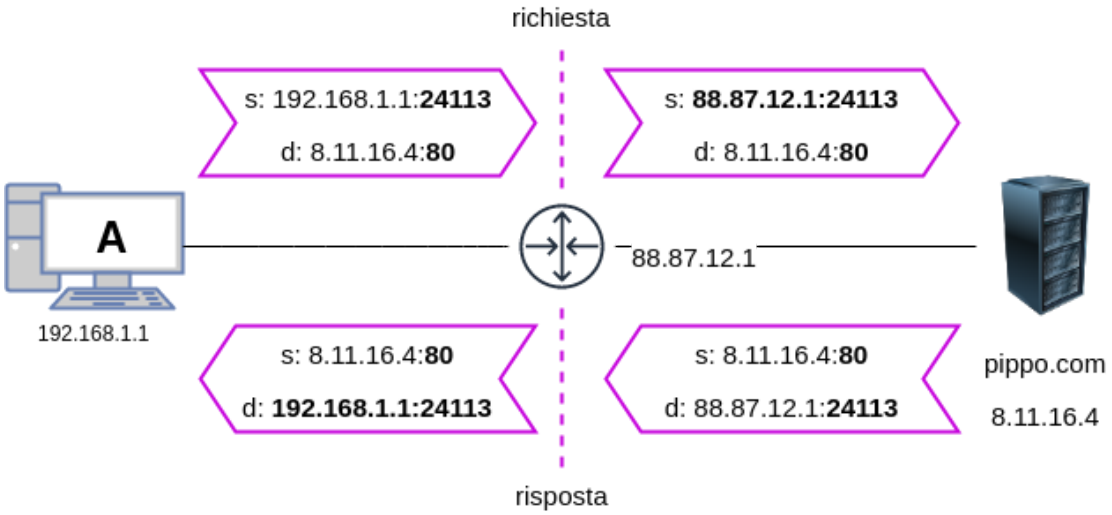
Multiplazione / demultiplazione

- Il livello 3 (IP) ha il solo compito di consegnare un messaggio da un host A a B.
- Ma sia su A che su B ci possono essere diverse applicazioni in esecuzione!
- I protocolli di livello 4 (TCP e UDP) introducono il concetto di “**porta**”: un meccanismo per identificare a quale applicazione dell’host destinatario inoltrare il messaggio.



Siccome la comunicazione può avvenire in maniera bidirezionale, **avremo un numero di porta del mittente e un numero di porta del destinatario**, così entrambi potranno far recapitare il messaggio al processo applicativo rispettivamente corretto.

Come avvengono multiplazione e demultiplazione



Il numero di porta è un numero a 16 bit quindi avremo 2^{16} porte tra 0 e 65535

le porte da 0 a 1023 sono riservate per i protocolli applicativi noti, ad esempio:

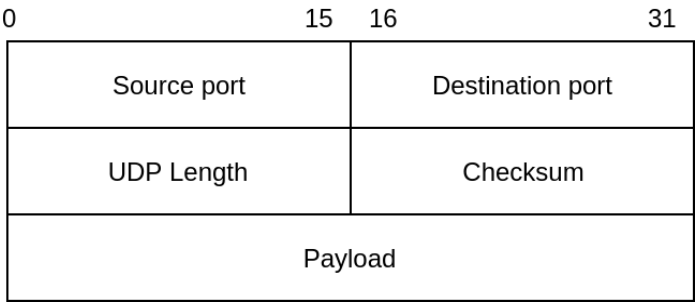
- HTTP : 80
- SMTP : 25
- DNS : 53
- ...

⇒ Le porte da 1024 a 49151 **non devono essere usate senza l'autorizzazione di IANA**,
⇒ le porte da 49152 a 65535 possono essere usate liberamente **senza autorizzazione**.

NAT + PAT

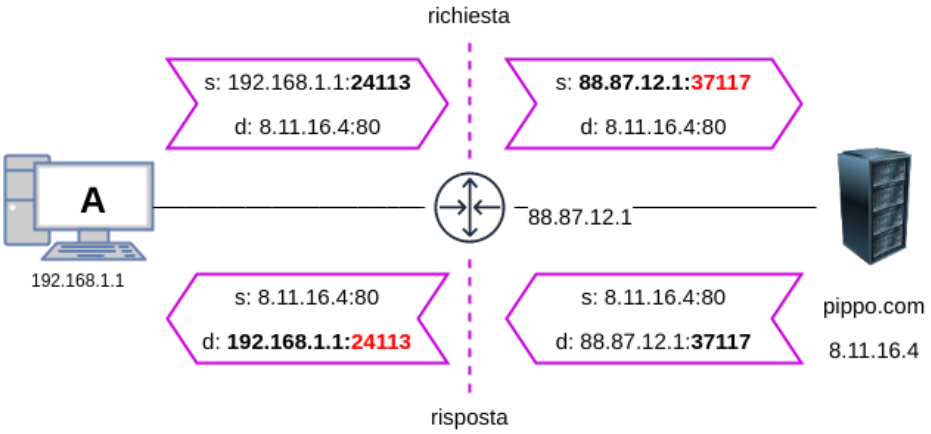
A livello 3 il Natting consiste nel cambiare gli indirizzi IP sorgente e destinazione passando da una rete privata ad una rete pubblica e vice versa. **A livello 4 abbiamo però una informazione in più: il numero di porta**.

Oltre al **NAT** avviene il **PAT** (**P**ort**A**ddress**T**ranslation): è rimappato anche il numero di porta:



segmento UDP: Header + Data

PAT:



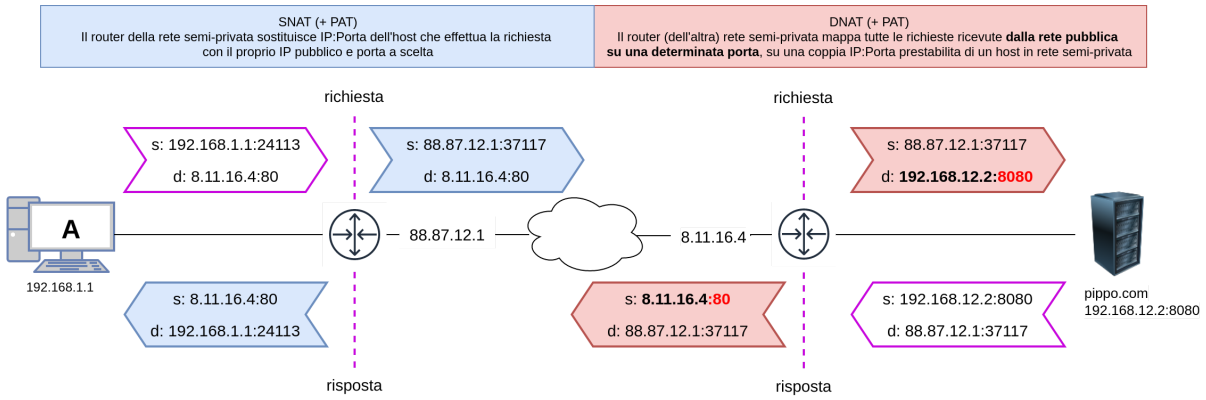
Siamo pronti a introdurre un'altra tipologia di natting: ***destination natting***.

Destination Natting

Lo facciamo quando vogliamo **esporre un servizio verso l'esterno**.

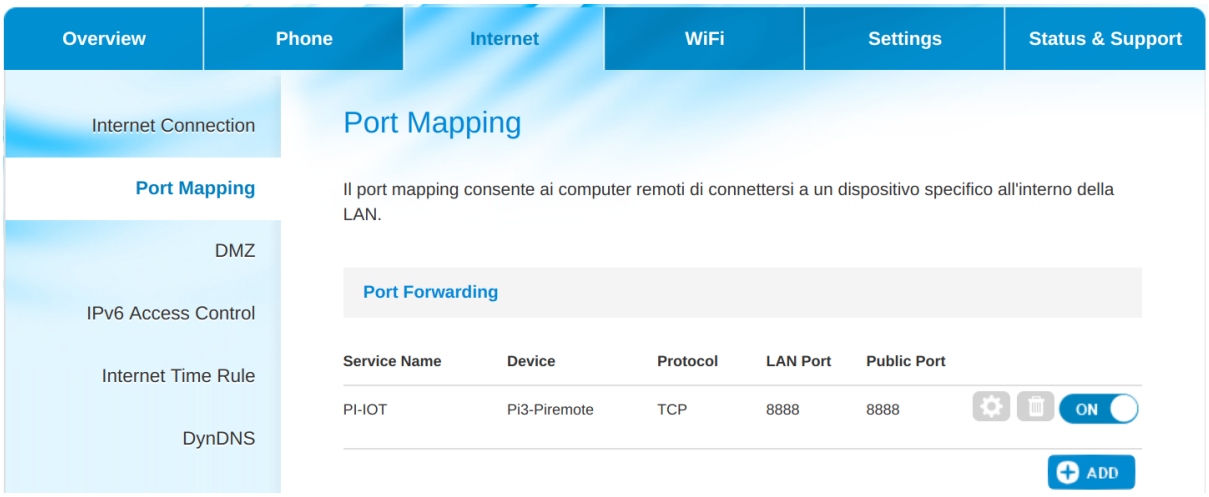
ES: il server che fornisce pippo.com: potrebbe essere in una rete semi-privata. **Come farà ad esporre la porta 80** (servizio web server) **verso l'esterno**?

Il destination natting consiste nel fatto che il router cambi IP:Porta di **destinazione**



*"Il **source natting** viene fatto automaticamente dal nostro router per permetterci di **navigare** normalmente, mentre il **destination natting** è da impostare sul router, indicando quale host della nostra rete vogliamo **esporre pubblicamente** e su che porta."*

La porta esposta pubblicamente e la porta dell'host da esporre **non devono per forza essere uguali**, come abbiamo visto nell'esempio di prima.

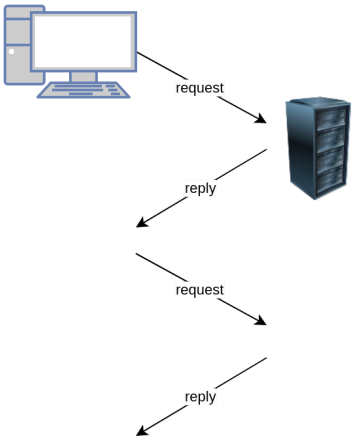


In questo esempio sul modem TIM, "**LAN Port**" si riferisce alla porta del server da esporre pubblicamente (può essere qualunque numero, anche 8000), mentre "**Public Port**" al numero di porta a cui il server deve essere raggiungibile dall'esterno (tipicamente 80 per HTTP e 443 per HTTPS)

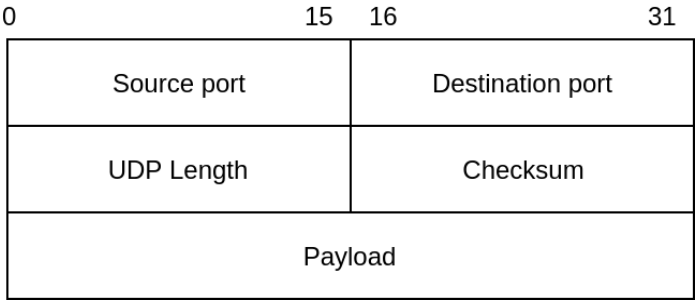
Caratteristiche di UDP

UDP (**U**ser **D**atagram **P**rotocol) è un protocollo di trasporto leggero, che fornisce solo le funzionalità minime del trasporto:

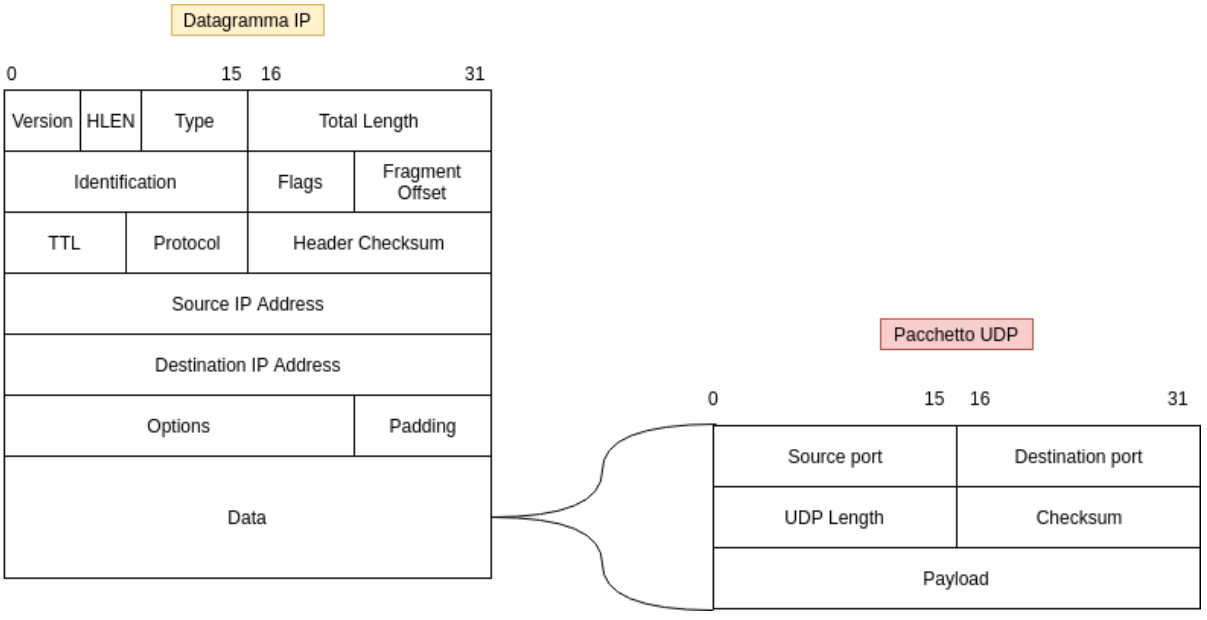
1. Servizio di **multiplazione/demultiplazione** (aggiunge il numero di porta)
2. **Controllo dell'errore** (campo checksum)
3. Servizio di **consegna non garantito** (i pacchetti possono essere persi, duplicati, consegnati fuori ordine.., come in IP)
4. **Servizio senza connessione** (ogni pacchetto UDP è trattato in modo indipendente dagli altri, come in IP)



Il pacchetto UDP



viene come sempre incapsulato a livello sottostante (3 - IP):



⇒ L'header del pacchetto UDP è di soli 8 byte (64 bit): un'aggiunta veramente minima ai dati dell'applicazione che andiamo a trasportare nel payload.

NB: Il **checksum** serve a controllare l'integrità dei dati, ma non viene fatta nessuna azione di recupero in caso di problemi!

- viene segnalato l'errore all'applicativo (del destinatario); o
- viene direttamente scartato il pacchetto

a seconda dell'implementazione.

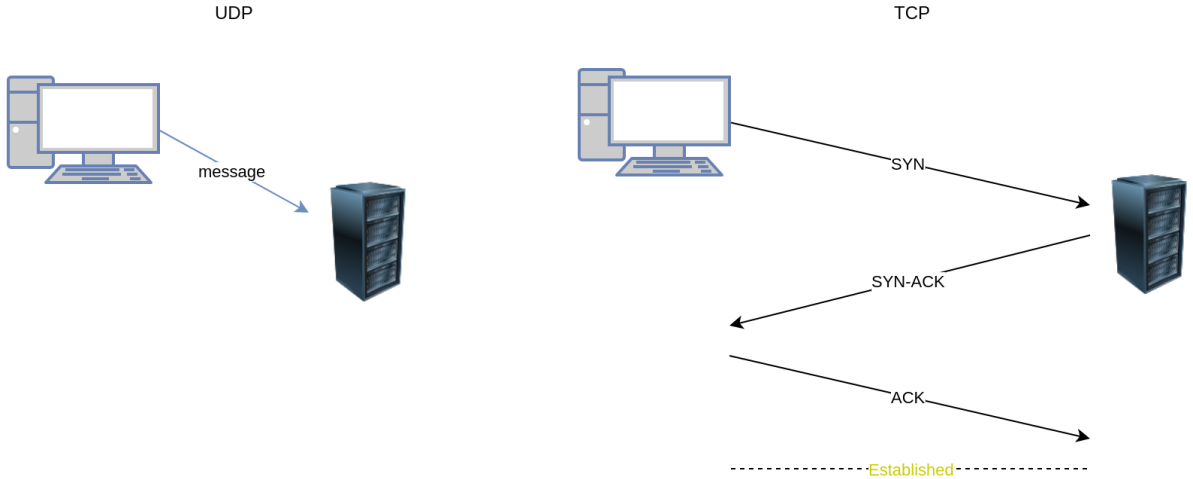
Caratteristiche di TCP

TCP (**T**ransmission **C**ontrol **P**rotocol) offre un servizio di **trasporto *affidabile* e *orientato alla connessione* su un canale trasmissivo inaffidabile**

Connessione TCP

Diciamo che **A** voglia comunicare con **B**.

⇒ In UDP avremmo direttamente inviato il messaggio **senza controllare che B fosse in ascolto e pronto a riceverlo**.
⇒ In TCP invece, prima di inviare il messaggio **apriamo una connessione**:



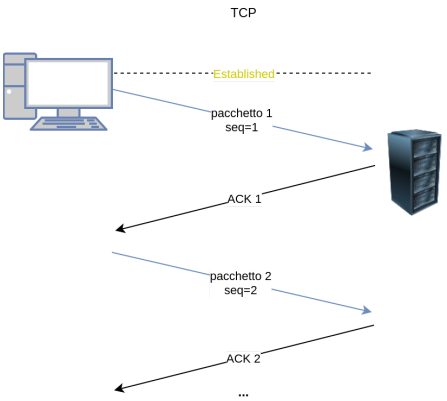
In questo esempio abbiamo un PC client ed un server.

- Il client invia al server una richiesta di connessione (segmento speciale SYN)
- Se il server è pronto e in ascolto, accetterà la richiesta, rispondendo con un altro segmento SYN di conferma
- A questo punto il client invia un'ultima conferma (ACK)
- Ora il trasferimento dei dati (invio del/dei messaggio/i) può avvenire.

Questo sistema si chiama "**Three Way Handshake**" (stretta di mano a tre vie) e assicura che sia il client che il server siano pronti a comunicare in un dato momento. Inoltre, tutti i messaggi scambiati da qui in avanti appartengono a questa sessione.

Affidabilità di TCP

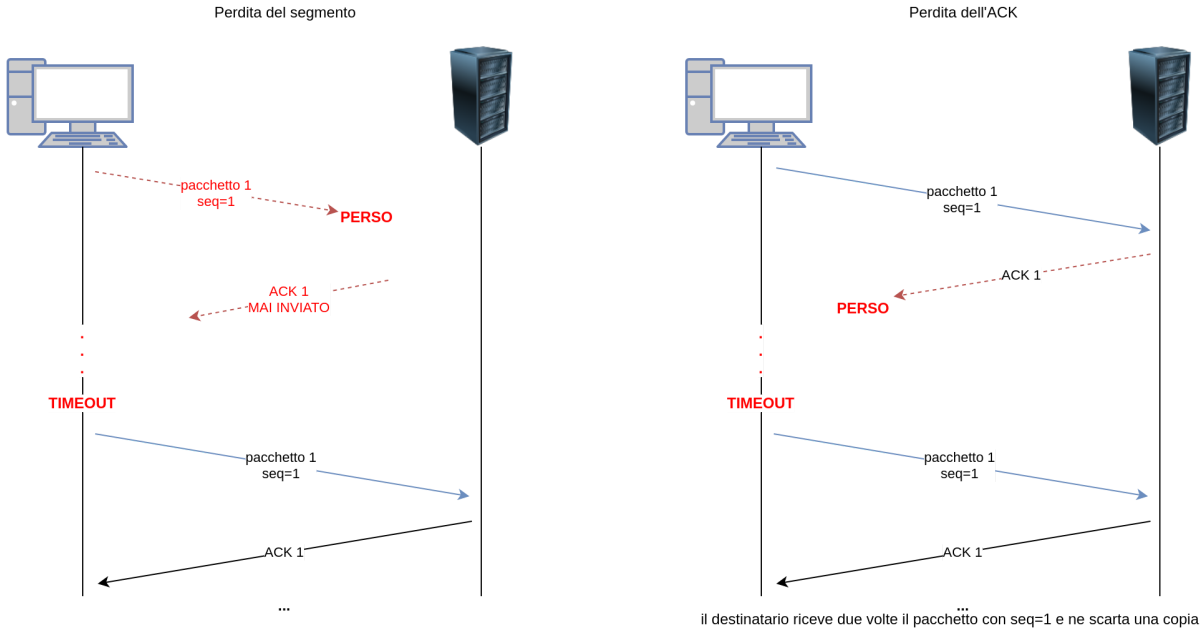
Una volta stabilita la connessione, **ad ogni messaggio inviato deve seguire una conferma** (ACK) **di avvenuta ricezione**.



Se ciò non avviene entro un certo periodo di tempo, il messaggio viene re-inviato automaticamente.

Questo rende affidabile la consegna dei messaggi.

Vediamo un esempio:



Al termine della trasmissione, il client può **chiudere la connessione**.

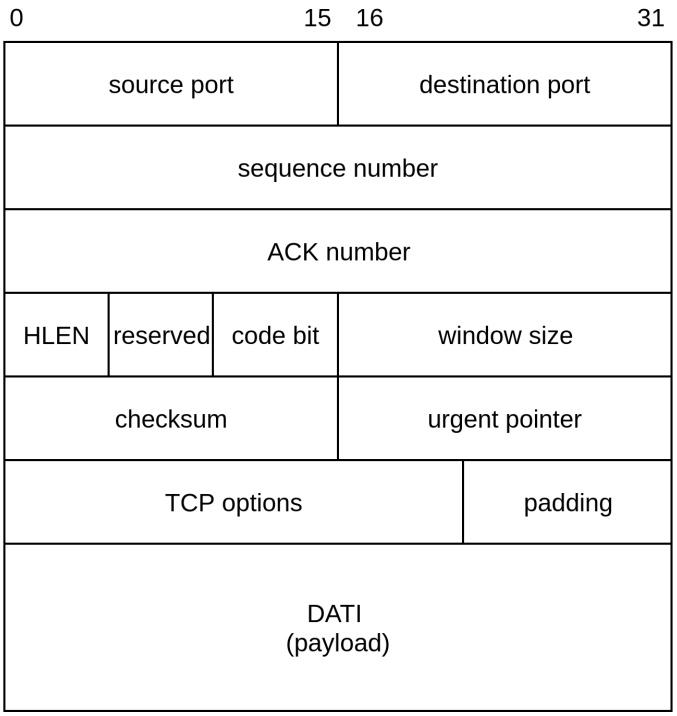
Chiusura "gentile" (standard):

- Il client invia un messaggio di **FIN**
- Il server dovrà confermare con un **ACK**, seguito da un messaggio di **FIN**
- Il client dovrà confermare la ricezione del messaggio **FIN** al server con un **ACK.**
- Dopo un certo intervallo di tempo se il client non riceve nessun messaggio, chiude a sua volta la connessione.

Chiusura "sbrigativa" (in caso di crash o sovraccarico):

- Invio di un messaggio di reset (**RST**).
  provoca l'immediata chiusura della connessione e il rilascio delle risorse.

Struttura di un segmento TCP



In particolare notiamo:

- **Sequence number**: identifica ogni segmento con un numero di sequenza, per riconoscere se un segmento viene ricevuto doppio, fuori ordine, o non viene affatto ricevuto (ricevo 8, ricevo 10, dov'è 9?)
- **ACK Number**: numero del segmento che confermo di aver ricevuto (ACK 8, ACK 10)
- **Code bit**: può essere SYN, FIN, RST, ecc…
- **Window size**: parametro che regola l'ampiezza della finestra di ricezione

Altri pregi e difetti: TCP vs UDP

- In **TCP** la connessione è **full-duplex**, ovvero è possibile trasferire dati contemporaneamente in entrambe le direzioni

- **UDP** è più adatto ad **applicazioni a bassa latenza** come lo streaming di video online (TCP non può garantire che le comunicazioni avvengano in tempo reale per via della sua complessità)

- **UDP** ha **meno overhead** per ogni messaggio inviato avendo un header molto più piccolo

- **UDP** delega al protocollo applicativo di livello superiore la gestione degli errori di trasmissione e altri aspetti che mancano rispetto a TCP