Livello Network e Livello Transport nel TCP/IP

Introduzione TCP/IP

Il protocollo TCP/IP (Transmission Control Protocol / Internet Protocol) era stato sviluppato in origine per il Ministero della Difesa statunitense negli Anni '70 per essere poi utilizzato con quella che conosciamo come Internet. Il TCP/IP assicura la comunicazione fra reti formate da nodi basati su un'ampia gamma di hardware e architetture di sistemi operativi. In altre parole realizza efficacemente la connessione di sistemi e reti eterogenei. Oggi il TCP/IP è diventato il protocollo più utilizzato al mondo e supporta tutte le piattaforme di rete sul mercato.
TCP/IP è una specifica che definisce una serie di protocolli usati per standardizzare come due o più computer scambiano informazioni gli uni con gli altri.

IP – Livello Network

IP: collocazione nella pila OSI

OSI: Open Systems Interconnect, sistema per il raggruppamento logico di tutte le funzionalità degli apparati di una rete, dalle funzioni delle componenti hardware a quelle software.

Confronto con Stack OSI:

Application
Presentation
Session
Transport
Network ←→ IP
Data Link
Physical

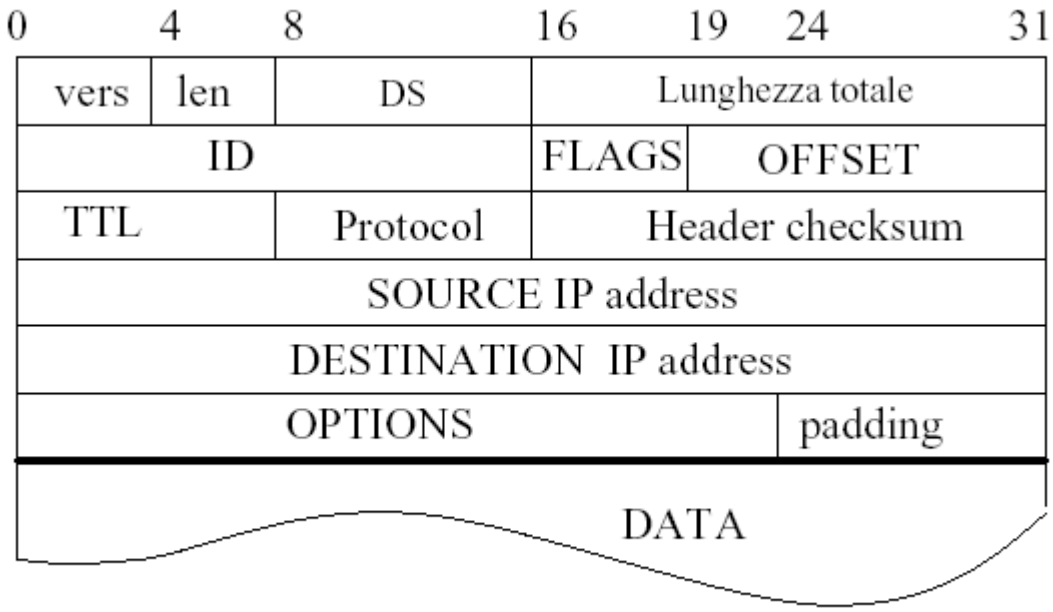
Alcune funzioni del livello *Network*:

- meccanismo di identificatione univoca dei nodi della rete;
- meccanismi per l'instradamento (routing) dei pacchetti, raccolta delle informazioni sulla topologia della rete;
- controllo di errori di trasmissione (CRC) sull'intestazione dei pacchetti;

Caratteristiche generali

IP offre un servizio di tipo connection-less, cioè una applicazione trasmette i pacchetti IP (detti anche "datagram") senza stabilire una connessione preliminare fisica o logica tra mittente e destinatario (a contrario di ciò che avviene nella telefonia). Questo implica che per esempio il cammino (path) di ciascun pacchetto è instradato in modo indipendente dagli altri. I pacchetti possono anche seguire cammini diversi e indipendenti.
Il protocollo è *unreliable*, ovvero IP non è in grado di riconoscere la perdita di un pacchetto. Tuttavia in caso di congestione e di perdita, è il protocollo di livello superiore (livello Transport) denominato:
Transmission Control Protocol il quale si incarica di effettuare la ritrasmissione delle porzioni di dati perse.

Ogni singolo pacchetto IP si suddivide in due porzioni:
- intestazione: per trasportare informazioni di controllo, cioè di utilità per il protocollo stesso
- dati: per il trasporto dei dati generati dall'applicazione (per esempio i dati di una pagina web, una porzione di file trasferito con ftp, il traffico interattivo generato da una sessione telnet etc.)



• VERS (4 bit): versione del protocollo IP utilizzata dal mittente
• Lunghezza (4 bit): lunghezza dell'INTESTAZIONE in numero di word (1 word = 32 bit)
• DS (8 bit): identifica la qualità di trasmissione richiesta dal pacchetto. DS significa "Differentiated Services", tale campo si divide ulteriormente in 2 sottocampi:
- DSCP: DS Code Point, il codice che identifica il servizio
- CU: Currently Unused, 2 bit riservati per uso futuro
• Lunghezza totale (16 bit):lunghezza complessiva (INTESTAZIONE+DATI) espressa in byte
• Source IP address (32 bit): indirizzo IP della sorgente
• Destination IP address (32 bit): indirizzo IP del destinatario
• ID (8 bit): identificatore del pacchetto. E' usato in caso di frammentazione del pacchetto, ovvero quando in un dato apparato di rete il pacchetto viene suddiviso in ulteriori pacchetti di lunghezza inferiore (ogni mezzo trasmissivo non ammette porzioni di dati superiori ad un dato valore detto MTU, Maximum Transfer Unit):
- ethernet: 1500 by, ATM: 9180 by, FDDI: 4352 by, X.25: 576 by
Tutti i frammenti derivanti dal medesimo pacchetto recano il medesimo identificatore (ID), che permette al destinatario di ricomporre il pacchetto IP originario.
• FLAGS (3bit): di cui
- 1 bit: indica se un dato pacchetto può essere frammentato oppure no
- more (1 bit): "1" significa che il frammento NON è l'ultimo della serie
• Fragment Offset (5 bit): se il pacchetto è un frammento allora indica la posizione (in byte) del frammento all'interno del pacchetto IP originario

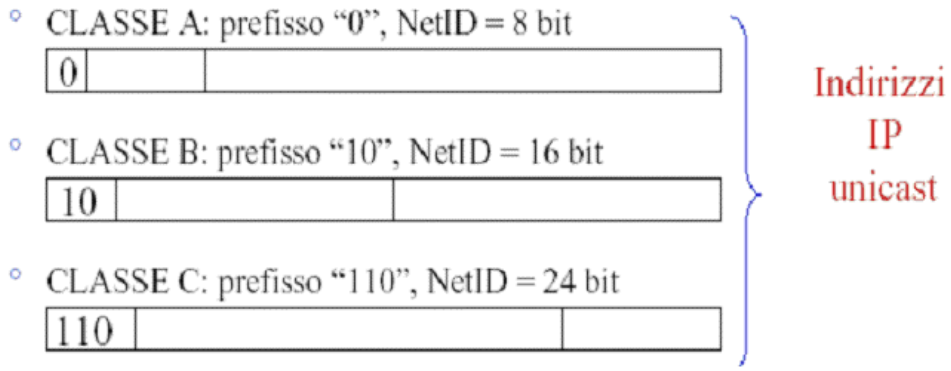
• TTL (8 bit): Time To Live. Numero massimo di volte in cui un dato pacchetto può passare attraverso un router (apparato intermedio che effettua l'instradamento dei pacchetti). Questo campo viene decrementato ogni volta che passa da un router. Se tale campo vale zero, il pacchetto viene scartato. *A cosa serve? Per evitare congestione.*
• Protocol (8 bit): codice standard che identifica il protocollo di livello superiore che ha generato il pacchetto (es. ARP, ICMP, IGMP…)
• Header Checksum (16 bit): campo per la verifica del controllo di correttezza dell'intestazione. Viene calcolato suddividendo l'intestazione in campi di 16 bit, facendone la somma con complemento 1, e calcolando il complemento binario del risultato.
• Options: di lunghezza arbitraria (ma sempre multipla di 32 byte), sono campi principalmente introdotti per fare test o monitoring della rete e per estendere le funzionalità del protocollo IP.

Struttura dell'indirizzo IP

• Indirizzo di 32 bit (IPv4) o di 128 bit (IPv6)
• perché IPv6? Soprattutto per risolvere il problema dell'esaurimento degli indirizzi
• suddiviso in due parti di lunghezza variabile:
° Network ID
° Host ID
• L'instradamento avviene attraverso la parte Network ID, mentre Host ID serve per identificare uno specifico sistema (router, PC, workstation, stampante…) collegato ad una rete.

Classi di indirizzi

• Esistono 4 classi di indirizzi, per ciascuna classe la lunghezza del Network ID è fissata. Il tipo di classe a cui appartiene un dato indirizzo viene identificato attraverso i primi 4 bit dell'indirizzo stesso.
° CLASSE A: prefisso "0", NetID = 8 bit (1-126)
° CLASSE B: prefisso "10", NetID = 16 bit (128-191)
° CLASSE C: prefisso "110", NetID = 24 bit (192-223)
° CLASSE D: prefisso "1110" identifica un indirizzo di multicast (cioè un insieme di più sistemi, ciascuno contraddistinto da un proprio indirizzo di classe A, B o C - indirizzo IP unicast) (224-239)



Indirizzi di rete e di *host*

• Gli indirizzi che recano una serie terminale di zeri sono riservati per indicare l'indirizzo di una rete:
es. 192.135.23.0 è l'indirizzo di una rete di classe C
131.154.0.0 è l'indirizzo di una rete di classe B
Non si possono assegnare indirizzi di rete a degli host.
• Tutti gli altri indirizzi rappresentano indirizzi di host.
Con il termine host si identificano tutti i singoli apparati di una rete che sono identificabili con un indirizzo IP, per esempio PC, workstation, stampanti e gli stessi router. Un *router* è dotato di *varie interfacce* di rete, le quali sono normalmente eterogenee (interfacce di linee seriali, ATM, Ethernet, FastEthernet, FDDI etc). Un router associa a ciascuna interfaccia indirizzi IP diversi. Gli indirizzi associati alle interfacce DEVONO appartenere a indirizzi di rete diversi. Una stessa interfaccia FISICA può recare uno o più indirizzi IP, questo equivale ad associare ad una interfaccia FISICA varie interfacce VIRTUALI, ciascuna recante un proprio indirizzo IP.

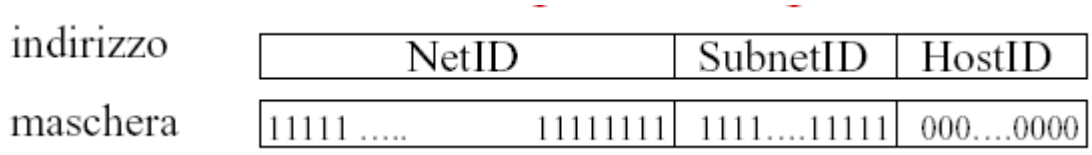
NOTA: se un host viene spostato da una rete ad un'altra, il suo indirizzo IP deve cambiare!

Per convenzione:
una rete è identificato con un indirizzo IP che ha un Host ID i cui bit sono tutti 0.

Subnetting (Classless IP)

• Problema: utilizzo degli indirizzi molto inefficiente! Per esempio, data una classe C, sono disponibili 255 indirizzi di host (il 256-esimo, costituito da una serie di zeri indica la classe stessa). Se in una data LAN è prsente un numero di host pari a 30, i rimanenti 225 indirizzi della classe non possono essere riutilizzati in alcuna altra parte di Internet!!
• Soluzione: tecnica di SUBNETTING (classless IP addressing)
Nel subnetting, dato un indirizzo di rete di qualsiasi classe, la lunghezza del prefisso (NetID) può essere estesa arbitrariamente, non deve essere necessariamente un multiplo di 8 bit.
Per indicare la lunghezza del prefisso, ogni indirizzo di rete viene accompagnato da una MASCHERA costituita da una sequenza di "1" il cui numero equivale alla lunghezza del prefisso.
Senza maschera NON è possibile individuare il SubnetID di un indirizzo.

Subnetting: terminologia



Maschera:
in binario -> sequenza di 1 e di 0 dove 1 indica che il bit appartiene al campo NetID o SubnetID, 0 che il bit fa parte dell'HostID.
in decimale: x.y.h.k/num, num è la maschera espressa in decimale che indica il numero di 1 nella maschera, es. 192.135.23.0/30

NetID: porzione dell'indirizzo la cui lunghezza è determinata dal prefisso (cioè dalla "classe"). Es. 192.135.23.5/30, NetID = 192.135.23.0
SubnetID: porzione di lunghezza variabile dell'indirizzo che segue il NetID e la cui lunghezza è definita dalla differenza fra la maschera (in decimale) e la lunghezza del NetID.
Es. 192.135.23.0/30, la lunghezza del SubNetID è di 6 bit
Sottorete: l'indirizzo ottenuto dalla concatenazione (NetID:SubnetID) ponendo nella rimanente parte *HostID* una sequenza di 0 e accompagnato dalla maschera
Indirizzo di rete: sinonimo di *Sottorete.* Se il subnetting non è utilizzato allora SubnetID è nullo, e quindi l'indirizzo di rete corrisponde al NetID
Indirizzo dell'host: la concatenazione (NetID:SubnetID:HostID)+maschera
Indirizzo: può essere sia un indirizzo di Host che di rete.

Subnetting (esempi)

• Esempio 1:
dato un indirizzo di CLASSE B applichiamo il subnetting estendendo il prefisso da 16 a 24:
indirizzo senza subnetting: 131.154.0.0
maschera standard: 255.255.0.0 ==> 256 * 256 indirizzi , [131.154.0.0 , 131.154.255.255]
maschera con subnetting: 255.255.255.0 ==> 256 indirizzi: di cui 1 di rete (131.154.0.0) e gli altri di host nel range [131.154.0.1 , 131.154.0.255]
Gli indirizzi di rete si calcolano ponendo l'HostID a 0.
Dall'indirizzo di CLASSE B 131.154.0.0 posso ricavare 256 sottoreti con maschera 255.255.255.0:
[131.154.0.0 , 131.154.255.0]

• Esempio 2:
indirizzo: 128.178.156.24
maschera: 255.255.255.0
classe dell'indirizzo: CLASSE B 128
NetID 128.178.0.0
Indirizzo di rete: NetID:SubnetID 128.178.156.0
HostID binario: 0.0.0.0001|1000, decimale: 0.0.0.24

• Esempio 3: 192.135.23.29/30
classe dell'indirizzo: CLASSE C
NetID: 192.135.23.0
indirizzo di rete = NetID:SubnetID: 192.135.23.28 (29 = **0001**|**11**01)
HostID binario: 0.0.0.0000|0001 decimale: 0.0.0.1
indirizzo dell'host: 192.135.23.29

Risoluzione degli indirizzi (**ARP e RARP**)

Sulla scheda di rete di un host è scritto un indirizzo fisico.

Se all'interno di una rete LAN ogni nodo conosce l'indirizzo IP di tutti gli altri nodi, ma non conosce i relativi indirizzi fisici, come può tale nodo raggiungere altri host senza utilizzare l'IP?
La soluzione è data da ARP (Address Resolution Protocol).
Come funziona?
Se un nodo deve trasmettere un pacchetto, prima trasmette una trama broadcast che contiene:

- indirizzo IP del mittente;
- indirizzo fisico del mittente;
- indirizzo IP del destinatario.

Essendo un msg broadcast, tutti i nodi ricevono la trama e analizzano i dati contenuti. Ogni nodo verifica se l'indirizzo IP del destinatario è il proprio. Il nodo che si riconosce come destinatario risponde con datagram contenente:

- indirizzo IP del mittente;
- inidirizzo fisico del mittente;
- indirizzo IP del destinatario (il nodo stesso);
- inidirizzo fisico del destinatario (il nodo stesso).

A questo punto l'host che aveva bisogno dell'indirizzo fisico per trasmettere può procedere alla trasmissione.
Ogni host della rete utilizza una **ARP-Table**, che contiene indirizzo-IP e indirizzo-fisico degli altri host. Ogni volta che un nodo utilizza ARP viene aggiunta una riga a tale tabella, quindi non è sempre necessario utilizzare ARP, prima i nodi possono consultare la propria ARP-Table.
La ARP-Table è come una memoria cache e quando è piena l'host rimpiazza una riga della tabella secondo il principio della **località temporale**.

Ottimizzazione: se viene collegato un nuovo host alla rete, questo invia sulla rete in broadcast il proprio indirizzo-IP e il proprio inidirizzo-fisico.

Conclusione: il protocollo ARP non viene quasi mai attivato.

Cosa accade se cambia l'indirizzo fisico di un host?
Siccome è come se fosse stato aggiunto un uovo host, questo invierà a tutta la rete i propri indirizzi (IP e fisico).
Problema: la ARP-Table può contenere più di una riga con lo stesso inidirizzo IP, manca l'univocità.
Soluzione:
Periodicamente viene invalidata una riga della ARP-Table; quale? Quando un host inserisce una nuova entry nella tabella, viene avviato un timer e al momento in cui scade si invalida il riferimento aggiunto.

NOTA: ARP non può essere usato con ATM, perché in ATM non esiste il broadcast.

Implementazione del Multicast

Quando serve?
Negli ultimi anni esistono applicazioni che richiede un indirizzamento multicast, come la video conferenza, le attività di workgroup o la televisione digitale.

Si utilizzano gli indirizzi IP in classe D. I primi 4 bit valgono 1110, e gli altri bit costituiscono un campo unico.
L'insieme degli host che ha lo stesso campo di multicast è chiamato host di gruppo. Il n. di host in un gruppo è infinito ed è dinamico, cioè un nodo può entrare o lasciare il gruppo.
Gli indirizzi multicast sono indirizzi alternativi. (anche a livello fisico – infatti tale indirizzo è scritto in un buffer della scheda di rete e l'host ha, oltre all'indirizzo fisico primario, tale indirizzo alternativo).

TCP e UDP– Livello Transport

IL LIVELLO QUATTRO (TRANSPORT) IN GENERALE

Questo è il livello in cui si gestisce per la prima volta (dal basso verso l'alto) una conversazione diretta, cioè senza intermediari, fra sorgente e destinazione.

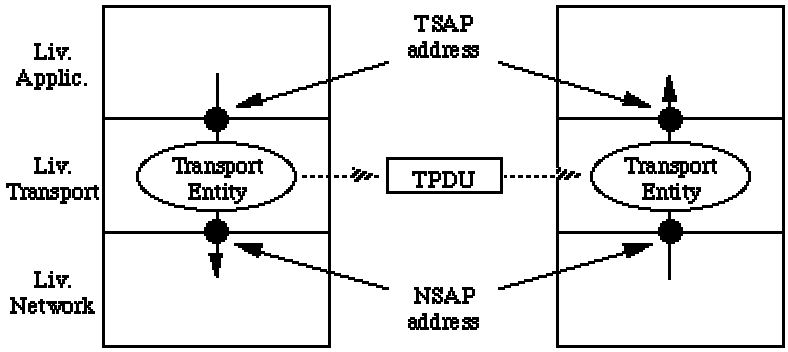
Il compito del livello transport (livello 4) è di fornire un trasporto efficace dall'host di origine a quello di destinazione, indipendentemente dalla rete utilizzata.

Da ciò discende che il software di livello transport è presente solo sugli host, e non nei router della subnet di comunicazione.

Servizi offerti dal livello transport

I servizi principali offerti ai livelli superiori sono vari tipi di trasporto delle informazioni fra una transport entity su un host e la sua peer entity su un altro host.

Naturalmente, tali servizi sono realizzati dal livello transport per mezzo dei servizi ad esso offerti dal livello network.



I servizi transport sono basati sui servizi network

Così come ci sono due tipi di servizi di livello network, ce ne sono due anche a livello transport:

- servizi affidabili orientati alla connessione (tipici di questo livello);
- servizi datagram (poco usati in questo livello).

Essi sono molto simili, come caratteristiche, a quelli corrispondenti del livello network, ed hanno gli analoghi vantaggi e svantaggi.

Il livello Transport ha un importante scopo, quello di isolare i livelli superiori dai dettagli implementativi della sottorete di comunicazione. Offre un insieme di primitive (di definizione dei servizi) semplici da utilizzare ed indipendenti dai servizi dei livelli sottostanti. Questo perché mentre solo poche persone scrivono componenti software che usano i servizi di livello network, molte scrivono applicazioni di rete, che si basano sui servizi di livello transport.

Un ultimo aspetto riguarda la possibilità di specificare la ***QoS*** (***Quality of Service***) desiderata. Questo in particolare è adatto soprattutto ai servizi connection oriented, nei quali il richiedente può specificare esigenze quali:

- massimo ritardo per l'attivazione della connessione;
- throughput richiesto;
- massimo ritardo di transito ammesso;
- tasso d'errore tollerato;
- tipo di protezione da accessi non autorizzati ai dati in transito.

In questo scenario:

- le peer entity avviano una fase di negoziazione per mettersi d'accordo sulla QoS, anche in funzione della qualità dei servizi di livello network di cui dispongono;
- quando l'accordo è raggiunto, esso vale per tutta la durata della connessione.

Protocolli di livello transport

I protocolli di livello transport (sulla base dei quali si implementano i servizi) assomigliano per certi aspetti a quelli di livello data link. Infatti si occupano, fra le altre cose, anche di:

- controllo degli errori;
- controllo di flusso;
- riordino dei TPDU (Transport Protocol Data Unit).

Ci sono però anche delle importanti differenze. Quella principale è che:

- nel livello data link fra le peer entity c'è un singolo canale di comunicazione;
- nel livello transport c'è di mezzo l'intera sottorete di comunicazione.

Questo implica che, a livello transport:

- è necessario indirizzare esplicitamente il destinatario;
- è più complicato stabilire la connessione;
- la rete ha una capacità di memorizzazione.

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 (TCP/IP) 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.

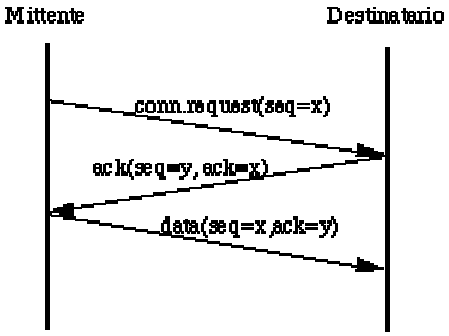
Three-way handshake (Tomlinson, 1975).

Il protocollo funziona così:

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

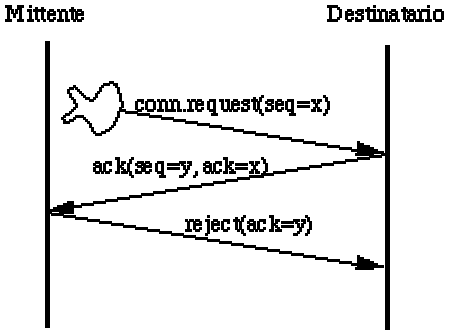
I valori x e y possono essere generati, ad esempio, sfruttando l'orologio di sistema, in modo da avere valori ogni volta diversi.

In assenza di errori, il funzionamento è il seguente:



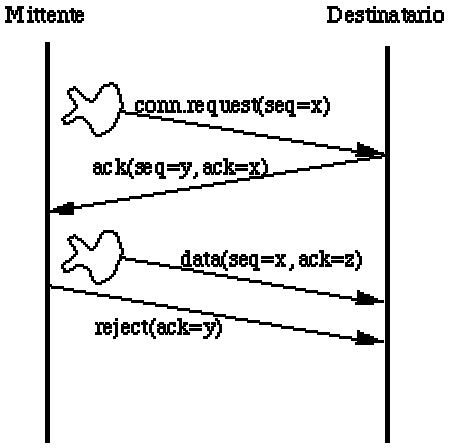
Three-way handshake

Se arriva a destinazione un duplicato della richiesta di attivazione, il destinatario risponde come prima ma il mittente, che sa di non aver richiesto una seconda connessione, lo informa dell'errore:



Duplicato della richiesta di attivazione

Se infine arrivano al destinatario sia un duplicato della richiesta di attivazione che un duplicato del primo TPDU dati, la situazione è la seguente:



Duplicati della richiesta di attivazione e del primo TPDU dati

Infatti, in questo caso:

- il mittente invia un reject alla risposta del destinatario, perché sa di non aver richiesto una seconda connessione (come nel caso precedente);
- il destinatario scarta il TPDU dati, perché questo reca un ack relativo ad un numero di sequenza (z) precedente e non a quello (y) da lui testé inviato.

Rilascio di una connessione

Rilasciare una connessione è più semplice che stabilirla, ma comunque qualche piccolo problema c'è anche in questa fase.

In questo contesto, rilasciare la connessione significa che l'entità di trasporto rimuove le informazioni sulla connessione dalle proprie tavole e informa l'utente di livello superiore che la connessione è chiusa.

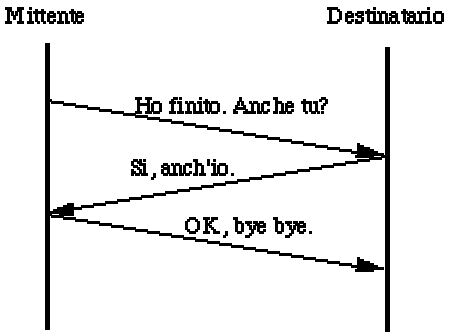
Ci sono due tipi di rilasci:

- ***asimmetrico***;
- ***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 si considera la connessione come una coppia di connessioni unidirezionali, che devono essere rilasciate indipendentemente. Quindi, lungo una direzione possono ancora scorrere dei dati anche se la connessione lungo l'altra direzione è stata chiusa. Il rilascio simmetrico è utile quando un processo sa esattamente quanti dati deve spedire, e quindi può autonomamente decidere quando rilasciare la sua connessione in uscita.

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



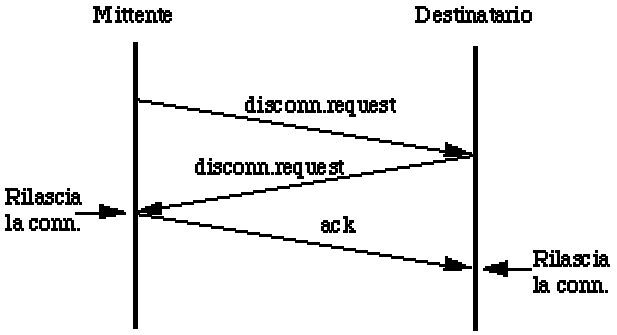
Semplice schema per il rilascio concordato della connessione

In realtà sia il mittente che il destinatario devono in due momenti diversi procedere al rilascio della connessione. Quindi 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;

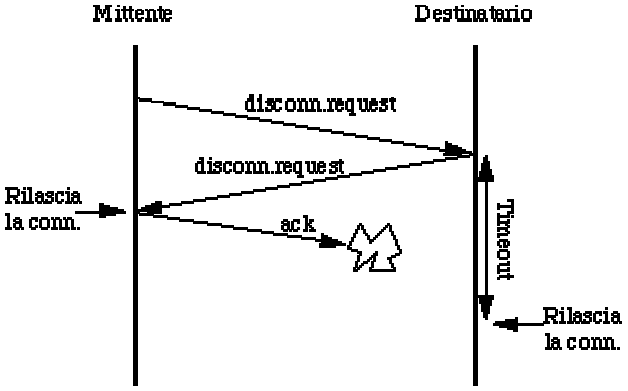
    - se non arriva nessuna risposta, dopo l'ultimo timeout rilascia comunque la connessione in ingresso;
- il destinatario, quando riceve disconn.request, fa partire un timer, invia a sua volta un disconn.request e attende l'ack di conferma. Quando arriva l'ack o scade il timer, rilascia la connessione in ingresso.

Normalmente il funzionamento è il seguente:



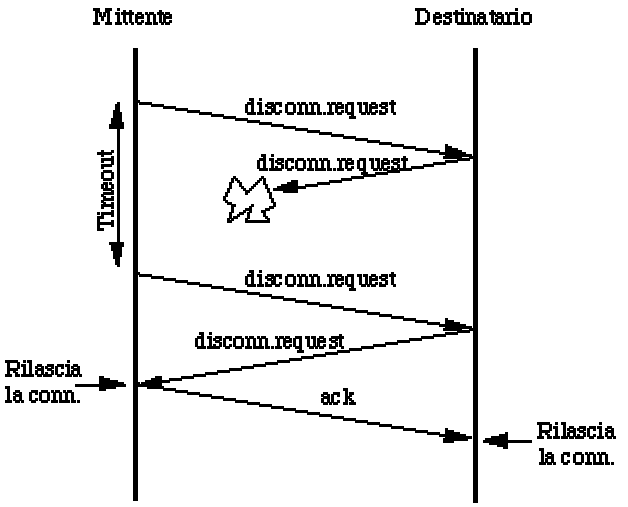
Rilascio concordato di una connessione transport

Se si perde l'ack:



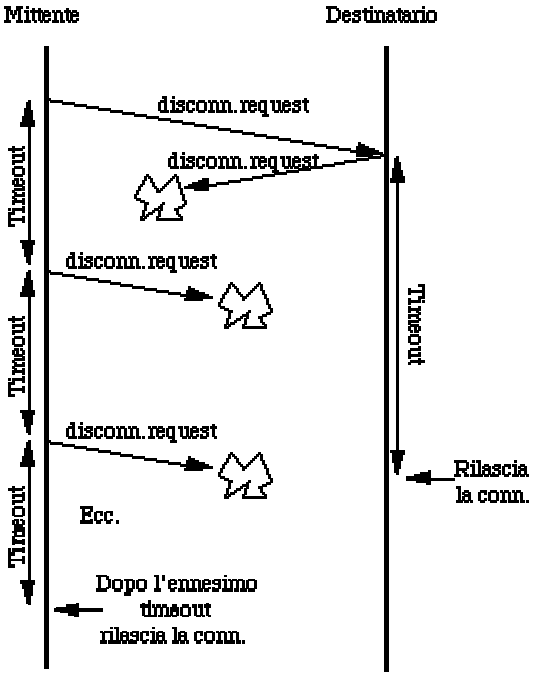
Perdita dell'ack durante il rilascio della connessione

Se invece si perde la risposta alla prima proposta di chiusura (supponendo che il timeout del destinatario sia molto più ampio di quello del mittente):



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

Infine, si può perdere la risposta alla prima proposta di chiusura e tutte le successive proposte di chiusura:



Perdita di tutti i messaggi tranne il primo

Il protocollo fallisce se tutte le trasmissioni di disconn.request della peer entity che inizia la procedura si perdono: in tal caso essa rilascia la connessione, ma l'altra entity non se ne accorge e la tiene aperta. Il risultato è una ***half-open connection***.

Un modo per risolvere il problema è che le parti rilascino una connessione quando passa un certo lasso di tempo (ad es. 60 secondi) senza che arrivino dati. Naturalmente, in tal caso, i processi devono scambiarsi dei ***dummy TPDU***, cioé dei TPDU vuoti, con una certa frequenza anche se non c'è necessità di comunicare alcunché, per evitare che la connessione cada.

Controllo di flusso e buffering

Per alcuni aspetti il controllo di flusso a livello transport è simile a quello di livello data link: è necessario un meccanismo, quale la gestione di una sliding window, per evitare che il ricevente sia sommerso di TPDU.

Però ci sono anche importanti differenze:

- il livello data link non ha alcun servizio, tranne la trasmissione fisica, a cui appoggiarsi. Il livello transport invece usa i servizi del livello network, che possono essere affidabili o no, e quindi può organizzarsi di conseguenza;
- il numero di connessioni data link è relativamente piccolo (uno per linea fisica) e stabile nel tempo, mentre le connessioni transport sono potenzialmente molte e di numero ampiamente variabile nel tempo;

- le dimensioni dei frame sono piuttosto stabili, quelle dei TPDU sono molto più variabili.

Se il livello transport dispone solo di servizi di livello network di tipo datagram:

- la transport entity sorgente deve mantenere in un buffer i TPDU spediti finché non sono confermati, come si fa nel livello data link;
- la transport entity di destinazione può anche non mantenere dei buffer specifici per ogni connessione, ma solo un pool globale: quando arriva un TPDU, se non c'è spazio nel pool, non lo accetta. Poiché il mittente sa che la subnet può perdere dei pacchetti, riproverà a trasmettere il TPDU finché non arriva la conferma. Al peggio, si perde un pò di efficienza.

Viceversa, ove siano disponibili servizi network di tipo affidabile ci possono essere altre possibilità per il livello transport:

- se il mittente sa che il ricevente ha sempre spazio disponibile nei buffer (e quindi è sempre in grado di accettare i dati che gli arrivano) può evitare di mantenere lui dei buffer, dato che ciò che spedisce:
  - arriva sicuramente a destinazione;
  - viene sempre accettato dal destinatario;
- se non c'è questa garanzia, il mittente deve comunque mantenere i buffer, dato che il destinatario riceverà sicuramente i TPDU spediti, ma potrebbe non essere in grado di accettarli.

Un problema legato al buffering è come gestirlo, vista la grande variabilità di dimensione dei TPDU:

- un pool di buffer tutti uguali: poco adatto, dato che comporta uno spreco di spazio per i TPDU di piccole dimensioni e complicazioni per quelli grandi;
- un pool di buffer di dimensioni variabili: più complicato da gestire ma efficace;
- un singolo array (piuttosto grande) per ogni connessione, gestito circolarmente: adatto per connessioni gravose, comporta spreco di spazio per quelle con poco scambio di dati.

In generale, data la grande variabilità di condizioni che si possono verificare, conviene adottare regole diverse di caso in caso:

- traffico bursty ma poco gravoso (ad esempio in una sessione di emulazione di terminale, che genera solo caratteri): niente buffer a destinazione, dove i TPDU si possono acquisire senza problemi man mano che arrivano. Di conseguenza, bastano buffer alla sorgente;
- traffico gravoso (ad esempio file transfer): buffer cospicui a destinazione, in modo da poter sempre accettare ciò che arriva.

Per gestire al meglio queste situazioni, di norma i protocolli di livello transport consentono alle peer entity di mettersi d'accordo in anticipo (al setup della connessione) sui meccanismi di bufferizzazione da usare.

Inoltre, spesso la gestione della dimensione delle finestre scorrevoli è disaccoppiata dalla gestione degli ack (vedremo il caso di TCP).

IL LIVELLO TRANSPORT IN INTERNET

Il livello transport di Internet è basato su due protocolli:

- ***TCP*** (***Transmission Control Protocol***)
- ***UDP*** (***User Data Protocol***)

Il secondo è di fatto IP con l'aggiunta di un breve header, e fornisce un servizio di trasporto datagram (quindi non affidabile).

Confronto con Stack OSI:

Application
Presentation
Session
Transport ←→ TCP e UDP
Network
Data Link
Physical

TCP

Servizio di trasmissione affidabile.

Caratteristiche:

INTERFACCIA BYTE STREAM: i programmi applicativi passano al protocollo un flusso "continuo" di byte;

ORIENTATO ALLA CONNESSIONE: prima si stabilisce la connessione, poi si procede alla trasmissione;

BUFFERED TRANSFER: sono necessari buffer sia in ricezione che in trasmissione. TX: è il protocollo che decide quando trasmettere i dati (in genere quando il buffer è pieno). RX: il protocollo può conoscere lo stato del buffer in ricezione (tramite ACK); se il buffer in ricezione è pieno è inutile trasmettere ancora!

UNSTRUCTURED STREAM: il flusso di dati trasmessi non è strutturato;

FULL DUPLEX: la comunicazione può essere bidirezionale. Conseguenza: le risorse allocate in TX devono essere duplicate in RX.

Il protocollo TCP è stato progettato per fornire un flusso di byte affidabile, da sorgente a destinazione, su una rete non affidabile.

Dunque, offre un servizio reliable e connection oriented, e si occupa di:

- accettare dati dal livello application;
- spezzarli in ***segment***, il nome usato per i TPDU (dimensione massima 64 Kbyte, tipicamente circa 1.500 byte);
- consegnarli al livello network, eventualmente ritrasmettendoli;
- ricevere segmenti dal livello network;
- rimetterli in ordine, eliminando buchi e doppioni;
- consegnare i dati, in ordine, al livello application.

E' un servizio full-duplex con gestione di ack e controllo del flusso.

Indirizzamento

I servizi di TCP si ottengono creando connessione di livello transport identificata da una coppia di punti d'accesso detti ***socket***. Ogni socket ha un ***socket number*** che consiste della coppia:

IP address: Port number

Il socket number costituisce il TSAP.

I port number hanno 16 bit. Quelli minori di 256 sono i cosidetti *well-known port*, riservati per i servizi standard. Ad esempio:

Port number	**Servizio**
7	Echo
20	Ftp (control)
21	Ftp (data)
23	Telnet
25	Smtp
80	Http
110	Pop versione 3

Poiché le connessioni TCP, che sono full duplex e point to point, sono identificate dalla coppia di socket number alle due estremità, è possibile che su un singolo host più connessioni siano attestate localmente sullo stesso socket number.

Le connessioni TCP trasportano un flusso di byte, non di messaggi: i confini fra messaggi non sono né definiti né preservati. Ad esempio, se il processo mittente (di livello application) invia 4 blocchi di 512 byte, quello destinatario può ricevere:

- 8 "pezzi" da 256 byte;
- 1 "pezzo" da 2.048 byte;
- ecc.

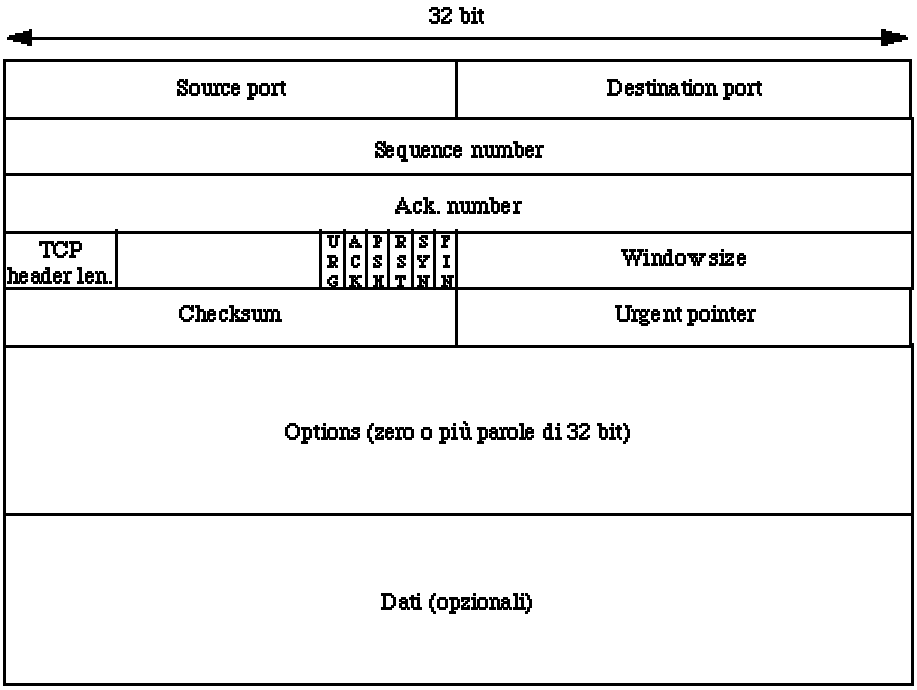
Ci pensano le entità TCP a suddividere il flusso in arrivo dal livello application in segmenti, a trasmetterli e a ricombinarli in un flusso che viene consegnato al livello application di destinazione.

C'è comunque la possibilità, per il livello application, di forzare l'invio immediato di dati; ciò causa l'invio di un flag ***urgent*** che, quando arriva dall'altra parte, fa sì che l'applicazione venga interrotta e si dedichi a esaminare i dati urgenti (questo succede quando, ad esempio, l'utente durante una sessione di emulazione di terminale digita il comando ABORT (CTRL-C) della computazione corrente).

Il protocollo TCP

Le caratteristiche più importanti sono le seguenti:

- ogni byte del flusso TCP è numerato con un numero d'ordine a 32 bit, usato sia per il controllo di flusso che per la gestione degli ack;
- un segmento TCP non può superare i 65.535 byte;
- un segmento TCP è formato da:
  - un'header, a sua volta costituito da:
    - una parte fissa di 20 byte;
    - una parte opzionale;
  - i dati da trasportare;
- TCP usa un meccanismo di sliding window di tipo go-back-n con timeout. Se questo scade, il segmento si ritrasmette. Si noti che le dimensioni della finestra scorrevole e i valori degli ack sono espressi in numero di byte, non in numero di segmenti.

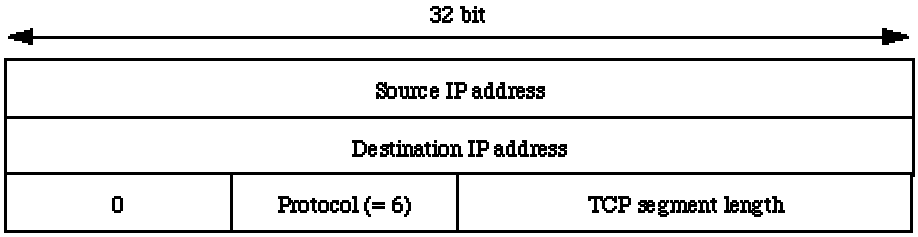


Formato del segmento TCP

I campi dell'header hanno le seguenti funzioni:

Source port, ***destination port***	identificano gli end point (locali ai due host) della connessione. Essi, assieme ai corrispondenti numeri IP, formano i due TSAP.
Sequence number	il numero d'ordine del primo byte contenuto nel campo dati.
Ack. number	il numero d'ordine del prossimo byte aspettato.
TCP header length	quante parole di 32 bit ci sono nell'header (necessario perché il campo options è di dimensione variabile).
URG	1 se urgent pointer è usato, 0 altrimenti.
ACK	1 se l'ack number è valido (cioè se si convoglia un ack), 0 altrimenti.
PSH	dati urgenti (***pushed data***), da consegnare senza aspettare che il buffer si riempia.
RST	richiesta di reset della connessione (ci sono problemi!).
SYN	usato nella fase di setup della connessione: • SYN=1 ACK=0 richiesta connessione; • SYN=1 ACK=1 accettata connessione.
FIN	usato per rilasciare una connessione.
Window size	il controllo di flusso è di tipo sliding window di dimensione variabile. Window size dice quanti byte possono essere spediti a partire da quello (compreso) che viene confermato con l'ack number. Un valore zero significa: fermati per un pò, riprenderai quando ti arriverà un uguale ack number con un valore di window size diverso da zero.
Checksum	simile a quello di IP; il calcolo include uno pseudoheader.
Urgent pointer	puntatore ai dati urgenti.
Options	fra le più importanti, negoziabili al setup: • dimensione massima dei segmenti da spedire; • uso di selective repeat invece che go-back-n; • uso di NAK.

Nel calcolo della checksum entra anche uno ***pseudoheader*** , in aperta violazione della gerarchia, dato che il livello TCP in questo calcolo opera su indirizzi IP.



Formato dello pseudoheader TCP

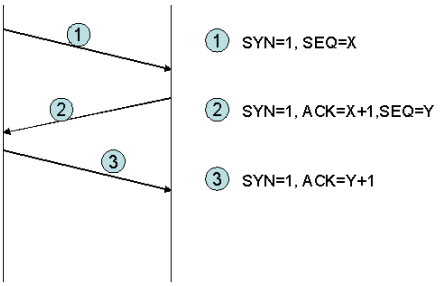
Lo pseudoheader non viene trasmesso, ma precede concettualmente l'header. I suoi campi hanno le seguenti funzioni:

Source IP address, destination IP address	indirizzi IP (a 32 bit) di sorgente e destinatario.
Protocol	il codice numerico del protocollo TCP (pari a 6).
TCP segment length	il numero di byte del segmento TCP, header compreso.

Attivazione della connessione

Si usa il three-way handshake visto precedentemente:

- una delle due parti (diciamo il server) esegue due primitive, listen() e poi accept() rimanendo così in attesa di una richiesta di connessione su un determinato port number e, quando essa arriva, accettandola;
- l'altra parte (diciamo un client) esegue la primitiva connect(), specificando host, port number e altri parametri quali la dimensione massima dei segmenti, per stabilire la connessione; tale primitiva causa l'invio di un segmento TCP col bit syn a uno e il bit ack a zero;
- quando tale segmento arriva a destinazione, l'entity di livello transport controlla se c'è un processo in ascolto sul port number in questione:
  - se non c'è nessuno in ascolto, invia un segmento di risposta col bit rst a uno, per rifiutare la connessione;
  - altrimenti, consegna il segmento arrivato al processo in ascolto; se esso accetta la connessione, l'entity invia un segmento di conferma, con entrambi i bit syn ed ack ad uno, secondo lo schema sotto riportato.



Attivazione di una connessione TCP

I valori di x e y in genere sono ricavati dagli host sulla base dei loro clock di sistema; il valore si incrementa di una unità ogni 4 microsecondi.

Rilascio della connessione (3-way-handshake modificato)

Il rilascio della connessione avviene considerando la connessione full-duplex come una coppia di connessioni simplex indipendenti, e si svolge nel seguente modo:

1. quando una delle due parti non ha più nulla da trasmettere, invia un fin;
2. quando esso viene confermato, la connessione in uscita viene rilasciata;

A questo punto è chiusa una metà della connessione.

3. quando anche l'altra parte completa lo stesso procedimento e rilascia la connessione nell'altra direzione, la connessione full-duplex termina.

Il protocollo di gestione delle connessioni si rappresenta comunemente come una ***macchina a stati finiti***. Questa è una rappresentazione molto usata nel campo dei protocolli, perché permette di definire, con una certa facilità e senza ambiguità, protocolli anche molto complessi.

Garanzia dell'affidabilità

Viene utilizzata una tecnica di tipo PAR (Positive Acknowledgement Retransmission): il mittente in via un segmento e si mette in attesa di un ACK; dopo l'invio attiva un timer e se allo scadere di tale tempo non è stato ricevuto nessun ACK, il mittente procede alla ritrasmissione.

IPOTESI: scade il timer, il mittente rispedisce il segmento e subito dopo riceve l'ACK precedente.

DOMANDA: si ha duplicazione in ricezione?

NO. Perché vengono mantenuti dal protocollo (presso il ricevente) dei numeri di sequenza che servono ad identificare i segmenti. Quindi quando il ricevente vede arrivare il secondo segmento (= al primo) lo può scartare.

COSTO della tecnica PAR: non si ha un buon sfruttamento della larghezza di banda. Tanto maggiore è il tempo tra un INVIO e la ricezione del suo ACK (RTT:Round Trip Time), tanto peggiore è l'utilizzazione della larghezza di banda e quindi le prestazioni della rete.

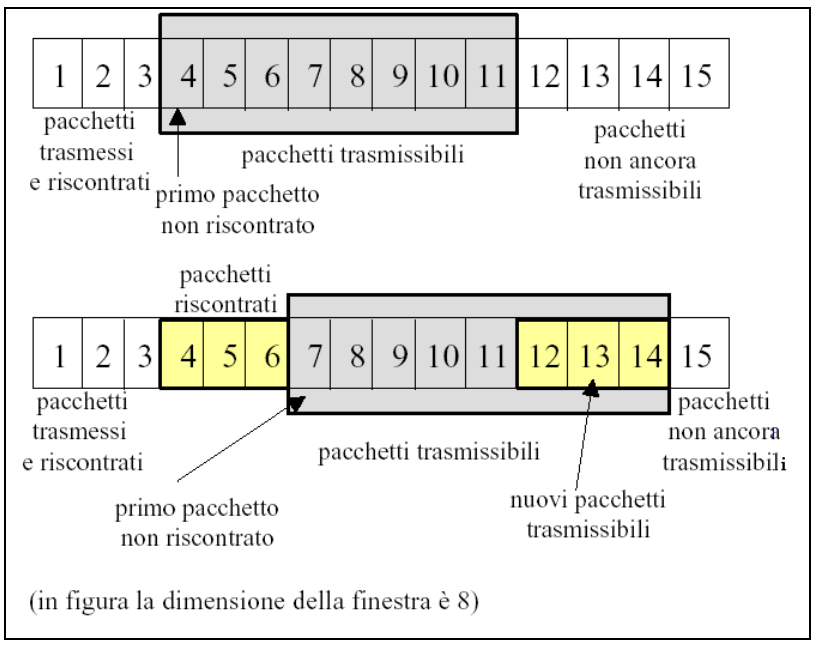
SOLUZIONE: **Sliding Window**

In fase di connessione viene stabilita la dimensione della finestra (n° di segmenti =N) e il mittente può inviare consecutivamente N segmenti senza attendere gli ACK. Quando il mittente riceve l'ACK del primo segmento procede all'invio dell'(N+1)-esimo, cioè sposta la finestra scorrevole di una posizione.

Dimensionare la Sliding Window:

1. al massimo può essere grande da saturare il buffer in ricezione;
2. se fissiamo la dimensione a 5, ma l'ack del primo segmento arriva quando sono stati trasmessi 3 segmenti ➔ la finestra è sovradimensionata rispetto al RTT.

La soluzione del TCP è utilizzare una finestra scorrevole di dimensione variabile, consentendo così il controllo del flusso.

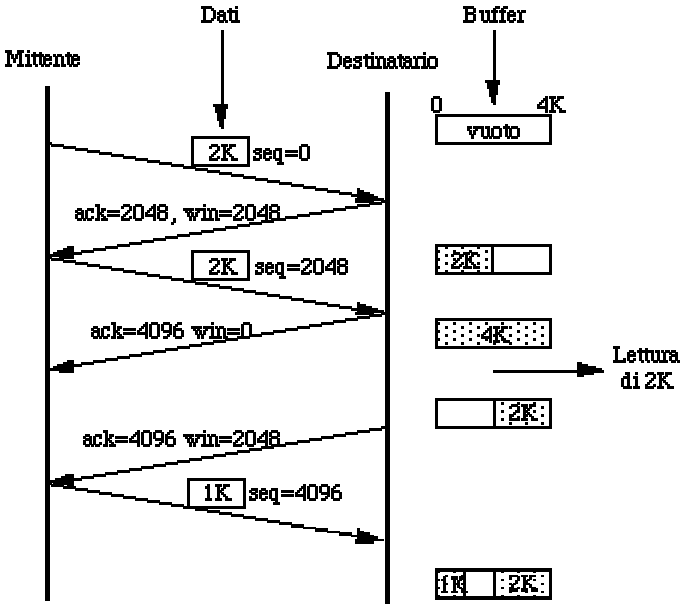


Politica di trasmissione del TCP con controllo del flusso

L'idea di fondo è la seguente: la dimensione delle finestre scorrevoli non è strettamente legata agli ack (come invece di solito avviene), ma viene continuamente adattata mediante un dialogo fra destinazione e sorgente.

In particolare, quando la destinazione invia un ack di conferma, dice anche quanti ulteriori byte possono essere spediti.

Nell'esempio che segue, le peer entity si sono preventivamente accordate su un buffer di 4K a destinazione.



Esempio di controllo del flusso TCP

In realtà: **ack = quantità dati ricevuti +1 !!!**

Anche se riceve win=0, il mittente può comunque inviare:

- dati urgenti;
- richieste di reinvio dell'ultimo ack spedito (per evitare il deadlock se esso si è perso).

NOTA: è possibile implementare l'ACK cumulativo: se ad esempio è stato perso l'ack del secondo segmentio, ma il mittente riceve l'ack del terzo segmento, ciò significa che anche il secondo segmento è giunto a destinazione senza errori.

Dimensione dell'area dati

La MSS (Maximum Segment Size) è la dimensione massima dei segmenti trasmessi ed è limitata da:

1. dimensione dei buffer in RX e in TX;
2. minima MTU (Maximum Transfer Unit) della rete di connessione e dei due end-points.

Durante la fase di connessione il valore della MSS viene negoziato dai due end-points.

Se gli end-points appartengono alla stessa rete fisica, il TCP calcola la MSS in modo che i datagrammi IP risultino coincidenti con la MTU della rete.

Se gli end-points appartengono a reti diverse e non si conosce la minima MTU delle reti attraversate, in genere viene scelta:

MSS: MTU (supportata dal primo router che viene attraversato per raggiungengere la destinazione) – Header TCP (20 byte) – Header IP (20 byte)

Quindi durante la negoziazione i due end-points si comunicano la loro MSS e sceglieranno la minima.

Controllo congestione

Il protocollo TCP assume che, se gli ack non tornano in tempo, ciò sia dovuto a congestione della subnet piuttosto che a errori di trasmissione (dato che le moderne linee di trasmissione sono molto affidabili).

Dunque, TCP è preparato ad affrontare due tipi di problemi:

- scarsità di buffer a destinazione;
- congestione della subnet.

Ciascuno dei problemi viene gestito da una specifica finestra mantenuta dal mittente:

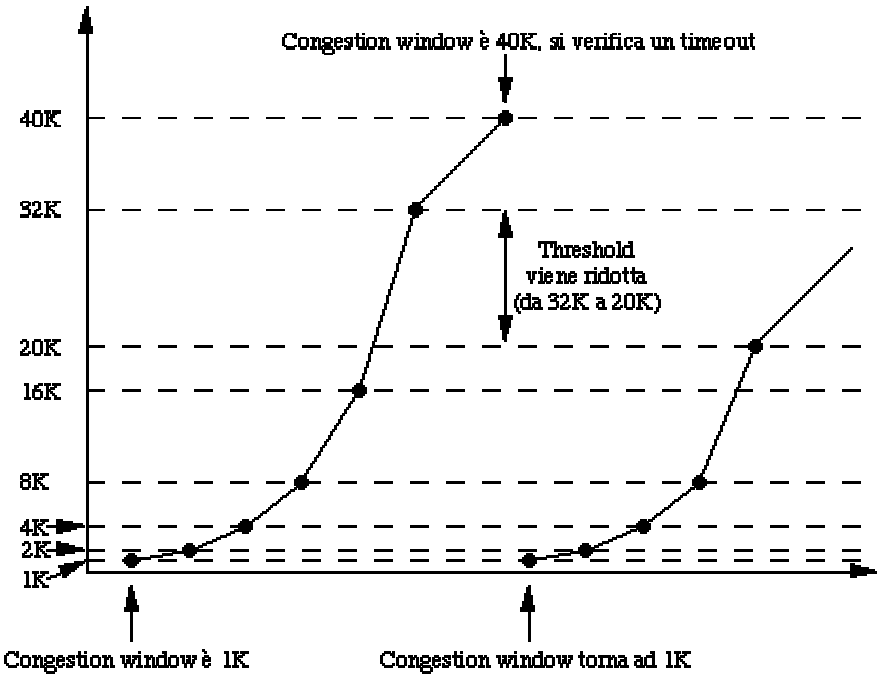
- la finestra del buffer del ricevitore (quella di cui all'esempio precedente);
- la ***congestion window***, che rappresenta quanto si può spedire senza causare congestione.

Il mittente si regola sulla più piccola delle due.

La congestion window viene gestita in questo modo:

- il valore iniziale è pari alla dimensione del massimo segmento usato nella connessione;
- ogni volta che un ack torna indietro in tempo la finestra si raddoppia, fino a un valore ***threshold***, inizialmente pari a 64 Kbyte, dopodiché aumenta linearmente di 1 segmento alla volta;
- quando si verifica un timeout per un segmento:
    - il valore di threshold viene impostato alla metà della dimensione della congestion window;
    - la dimensione della congestion window viene impostata alla dimensione del massimo segmento usato nella connessione.

Vediamo ora un esempio, con segmenti di dimensione 1 Kbyte, threshold a 32 Kbyte e congestion window arrivata a 40 Kbyte:

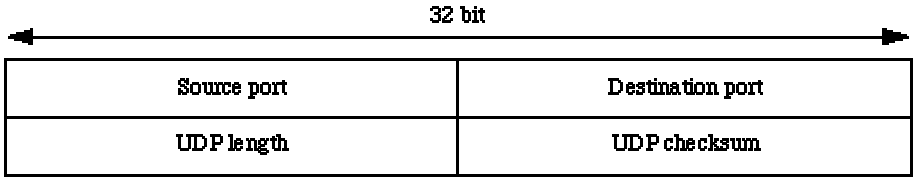


Esempio di controllo della congestione TCP

Il protocollo UDP

Il livello transport fornisce anche un protocollo non connesso e non affidabile, utile per inviare dati senza stabilire connessioni (ad esempio per applicazioni client-server).

L'header di un segmento UDP è molto semplice:



Formato dell'header UDP

La funzione di calcolo della checksum può essere disattivata, tipicamente nel caso di traffico in tempo reale (come voce e video) per il quale è in genere più importante mantenere un'elevato tasso di arrivo dei segmenti piuttosto che evitare i rari errori che possono accadere.