

Livello quattro (Livello transport)

6. Generalità:

Questo è il livello più importante, si occupa di garantire un trasporto affidabile al livello precedente, in questo livello per la prima volta si gestisce una conversazione diretta, tra sorgente e destinazione.

I servizi offerti da questo livello sono di due tipi (come per il livello network) quelli orientati alla connessione e affidabili e quelli di tipo datagram, questi tipi di servizi sono del tutto simili sia per vantaggi che svantaggi a quelli del livello network, l'unica ragione per la quale si è ritenuto necessario dover duplicare questi servizi, è che l'utente deve poter utilizzare un servizio affidabile indipendentemente dalla subnet. Un altro importante scopo è quello di isolare i livelli superiori dai dettagli implementativi della subnet di comunicazione. Questo livello consente inoltre di specificare la QoS (Quality of Service) desiderata, in particolare questo è molto utile nei servizi di tipo connection oriented nei quali il richiedente può specificare:

- a) massimo ritardo per l'attivazione della connessione;
- b) throughput richiesto;
- c) massimo ritardo di transito ammesso;
- d) tasso d'errore tollerato;
- e) tipo di protezione da accessi non autorizzati;

In questo scenario le peer entity quando avviano una negoziazione nella quale si accordano sul QoS.

6.1 Protocolli di livello transport:

I protocolli di questo livello assomigliano per certi aspetti a quelli del livello data link, si occupano di controllare gli errori, controllare il flusso dei dati, e riordinare le TPDU. Le differenze più importanti sono che nel livello data link tra le peer entity c'è un singolo canale di comunicazione, e che nel livello transport c'è di mezzo l'intera subnet di comunicazione.

Tutto questo comporta che nel livello transport:

- a) è necessario indirizzare esplicitamente il destinatario;
- b) è più complicato stabilire la connessione;
- c) la rete ha la capacità di memorizzare i pacchetti quindi a destinazione potrebbero arrivare TPDU in ogni momento;
- d) a causa del numero molto variabile di connessioni sia il buffering che il controllo di flusso richiedono approcci differenti rispetto a quelli del livello data link.

6.2 Indirizzamento:

Come abbiamo già accennato l'indirizzamento in questo livello deve essere esplicito, tipicamente un indirizzo TSAP (Transport Service Access Point) è formato dalla coppia (*NSAP Address, Informazione supplementare*) ad esempio su internet un TSAP Address è costituito da (IP: port number).

6.3 Principi dei protocolli di livello transport

6.3.1. Attivazione della connessione:

L'attivazione di una connessione tra peer entity è un processo molto delicato, difatti dovendo i pacchetti attraversare la subnet di comunicazione il rischio di perdita o di duplicazione è molto elevato (basta pensare a ripetuti ritardi nell'invio degli ack di conferma con conseguente invio di duplicati di pacchetti già arrivati). Già sappiamo che una volta che la connessione è stata stabilita un qualunque protocollo a finestra scorrevole consente di risolvere il problema dei duplicati, difatti le due peer entity si sono già accordate sul numero iniziale di sequenza e quindi sono in grado di stabilire se un pacchetto è un duplicato, e di conseguenza scartarlo. Ma se il duplicato ci fosse tra i pacchetti di attivazione della connessione? Per risolvere questo problema esiste la tecnica del *three way hand shaking* che consiste in una fase di attivazione composta da tre fasi distinte:

- a) il richiedente invia un TPDU di tipo *conn.request* con un numero x proposto per inizio sequenza;
- b) il destinatario risponde con un ack con il numero di sequenza x e invia una propria proposta di numero di sequenza y;
- c) il richiedente risponde con un altro ack che riporta i dati iniziali del dialogo più la conferma del numero y;

Il valore di x e y può essere scelto facendo uso dell'orologio di sistema, in linea di massima comunque i valori sono ciclici con una lunghezza tale da coprire 2 volte il tempo massimo di vita di un pacchetto.

Con questo protocollo è possibile individuare duplicati errati di richieste di connessione.

6.3.2. Rilascio della connessione:

Il rilascio della connessione è un operazione estremamente più semplice rispetto a quella dell'inizializzazione.

In pratica quello che succede al momento del rilascio è che l'entità di trasporto rimuove tutte le informazioni sulla connessione dalle proprie tavole ed informa l'utente di livello superiore che la connessione è stata chiusa.

Ci sono due tipi di rilasci:

- a) asimmetrico
- b) simmetrico

Il primo dei due avviene quando una sola delle due parti chiude la connessione e in questo caso ci possono essere grandi perdite di dati, nell'altro caso la connessione viene considerata come una coppia di connessioni unidirezionali, che debbono essere rilasciate indipendentemente. Quello che può succedere è che in una direzione scorrano ancora dei dati mentre nell'altra direzione la connessione sia stata chiusa. Il metodo simmetrico è utile quando si è a conoscenza dell'esatta quantità di dati da trasmettere e si può quindi stabilire autonomamente quando rilasciare la connessione.

Se le due entità si debbono mettere d'accordo prima di rilasciare la connessione un metodo sicuro è che si scambino dei messaggi di conferma come all'avvio.

Purtroppo questo tipo di accordo cade facilmente nel *problema dei due eserciti* nel quale due eserciti che compongono l'armata A si trovano divisi dall'armata B più forte dei due separati; l'armata A nel suo complesso però è più forte di B occorre quindi che i due eserciti si accordino circa l'ora dell'attacco, ma cosa succede se strada facendo i messaggeri dei due eserciti di A vengono catturati da B? La soluzione migliore a questo problema anche se non del tutto risolutiva è un protocollo three way hand shake con controllo di timeout.

Il protocollo funziona in questo modo:

- a) il mittente invia un `disconn.request` e se non arriva risposta entro un tempo t lo rispedisce per un massimo di n volte;
 - a.1) appena arriva una risposta rilascia la connessione in ingresso ed invia un `ack`
 - a.2) se non arriva risposta dopo l'ultimo timeout rilascia la connessione
- b) il destinatario quando riceve la richiesta fa partire un timer ed invia una `disconn.request` e attende l'`ack`, se l'`ack` arriva o il timer scade rilascia la connessione.

Questo protocollo fallisce se tutte le `disconn.request` della peer entity che inizia la procedura si perdono, in tal caso questa peer entity rilascia la connessione ma dall'altra parte nessuno se ne accorge portando così ad un *half open connection*. La soluzione a questo problema sta nel rilasciare forzatamente la connessione dopo un certo tempo di inattività, ovviamente per effettuare questo controllo occorre che le peer entity si scambino dei dummy TPDU nei momenti di inattività nei quali è comunque richiesto che la connessione sia attiva.

6.3.3. Controllo di flusso e buffering:

Il controllo di flusso di questo livello è estremamente simile a quello del livello data link, entrambi di fatto hanno bisogno di un meccanismo come quello delle sliding window per evitare di sommergere di TPDU il ricevente.

Nonostante questa similitudine esistono anche importanti differenze, come:

- a) il livello data link non ha alcun servizio ad eccezione della trasmissione fisica, a cui appoggiarsi, il livello transport d'altra parte usa i servizi del livello network che come abbiamo visto possono essere di tipo affidabile o no;
 - b) il numero di connessioni data link è piccolo (uno per linea fisica), mentre per quanto riguarda il livello transport sono potenzialmente moltissime, e di numero variabile nel tempo;
 - c) le dimensioni del frame sono stabili nel data link mentre molto variabili quelle di TPDU;
- nel caso in cui il livello network offra solo servizi di tipo datagram il livello transport:
- a) la transport entity sorgente dovrà mantenere in un buffer tutte le TPDU inviate fino al momento della conferma come nel data link;
 - b) le transport entity di destinazione può anche non mantenere dei buffer specifici per ogni connessione, ma solamente un pool globale, e quando arriva una TPDU e non c'è spazio nel pool semplicemente viene scartata tanto il mittente sapendo che i pacchetti possono andare persi nella subnet provvederà a rispedirla;

nel caso in cui il livello network offra anche servizi di tipo connection oriented:

- a) se il mittente sa che il ricevente ha sempre spazio nei buffer non è tenuto a memorizzare i pacchetti inviati, dato che ciò che spedisce:
 - a.1) arriva sempre a destinazione;
 - a.2) viene sempre accettato;
- b) se non c'è garanzia, il mittente deve sempre mantenere dei buffer per la memorizzazione dato che il destinatario riceverà sicuramente i TPDU spediti ma non è detto che li possa accettare;

Un problema legato alla gestione dei buffer si presenta poiché le dimensioni dei TPDU possono variare sensibilmente, la scelta quindi di un pool di dimensione fissa è in realtà una scelta poco conveniente poiché in linea di massima comporta un grande spreco di memoria.

E' sicuramente più indicato un pool di buffer di dimensione variabile, anche se questo presenta un complicazione eccessiva nella gestione.

Infine un singolo array (abbastanza grande) per ogni connessione, gestito circolarmente, è adatto a connessioni gravose, ma comporta spreco di spazio nelle connessioni con poco scambio dati.

In generale data la grande variabilità delle situazioni che il livello transport deve affrontare occorrono soluzioni pensate per la specifica occasione, quindi in caso di traffico *bursty* ma poco

gravoso (emulazione sessione di terminale) non conviene mantenere un buffer a destinazione dove i TPDU possono essere acquisiti man mano che arrivano, è invece utile mantenere un buffer dal mittente.

Se invece il traffico è gravoso (file transfer) dei buffer abbastanza grandi sono l'unica soluzione accettabile.

Per poter gestire tutte queste situazioni le peer entity di livello transport normalmente si accordano in anticipo sulla politica da seguire.

6.3.4. Multiplexing:

Una tecnica estremamente utile nelle conversazioni tra livello transport e network è il multiplexing.

Con questa tecnica possiamo convogliare più conversazioni transport su un'unica connessione network, riducendo così i “costi” di connessioni pesanti.

Se si vuole invece ottenere una maggiore banda di quella consentita da una singola connessione network, conviene dividere una conversazione transport su più connessioni network.

6.4 Il livello transport in Internet:

Il livello transport in internet si basa su due protocolli, *TCP* e *UDP*.

UDP è di fatto IP con l'aggiunta di un breve header, è di tipo datagram e quindi non affidabile, TCP invece è un protocollo connection oriented che offre un trasporto affidabile su una rete inaffidabile.

TCP dunque offre i seguenti servizi:

- a) accetta dati dal livello application;
- b) li spezza in *segment* (il nome dei TPDU con dimensione max 64 Kbyte);
- c) li consegna al livello network, eventualmente ritrasmettendoli;
- d) riceve i segmenti dal livello network;
- e) riordina i segmenti ricevuti, eliminando i doppi e i buchi;
- f) li trasmette al livello application;

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

6.4.1. Indirizzamento:

I servizi TCP si ottengono creando delle connessioni di livello transport, queste connessioni sono identificate da una coppia di punti d'accesso detti *socket*; questi socket vengono identificati da un socket number composto dalla coppia di valori (IP Address : Port Number) che costituisce il TSAP, port number è un numero a 16 bit i valori al di sotto di 256 sono detti *well known port* e sono riservati ai servizi standard (web 80, ftp 20-21, telnet 23, smtp 25 ...). Le connessioni TCP trasportano un flusso di byte non dei messaggi (nel senso che quello che viene trasportato non è delimitato in alcun modo) ne consegue che se il mittente invia 4 blocchi da 512 byte il destinatario può ricevere 8 blocchi da 256 byte o 1 blocco da 2048 byte, saranno poi le transport entity TCP a suddividere il flusso in arrivo dal livello application in segmenti e a trasmetterli o ricombinarli in flusso che viene poi consegnato al livello application di destinazione.

Esiste la possibilità di abilitare un flag *urgent* che forza l'invio immediato lato mittente e la lettura immediata lato destinatario che bloccherà l'applicazione per esaminare i dati urgenti.

6.4.2. Il protocollo TCP:

Le caratteristiche più importanti del protocollo sono:

- a) ogni byte è numerato con un numero d'ordine a 32 bit, usato sia per il controllo di flusso che per gli ack;
- b) un segmento TCP non può superare i 65.535 byte;
- c) il segmento TCP è costituito da:
 - c.1) un header che a sua volta è costituito:
 - c.1.1) una parte fissa di 20 byte;
 - c.1.2) una parte opzionale;
 - c.2) i dati da trasportare;
- d) TCP usa un meccanismo di sliding window di tipo go-back-n con timeout o selective repeat.

L'header TCP analizzato in dettaglio è il seguente:

32 bit							
Source Port	Destination Port						
Sequence Number							
Ack Number							
TCP header length	U	A	P	R	S	F	Window Size
	R	C	S	S	Y	I	
	G	K	H	T	N	N	
Checksum	Urgent Pointer						
Options (zero o più parole da 32 bit)							
Dati (opzionali)							

I campi dell'header hanno le seguenti funzioni:

<i>Source port, destination port</i>	identificano gli end point della connessione, insieme ai corrispondenti IP formano i TSAP
<i>Sequence number</i>	il numero di sequenza del primo byte contenuto nel campo dati
<i>Ack number</i>	il numero di sequenza del prossimo byte atteso
<i>TCP header length</i>	quante parole di 32 bit ci sono nell'header
<i>URG</i>	bit flag urgent, 1 se Urgent Pointer è usato
<i>ACK</i>	bit flag per indicare se il segmento contiene un ack
<i>PSH</i>	bit flag dati urgenti (<i>pushed data</i>) da consegnare senza attese
<i>RST</i>	richiesta di reset della connessione
<i>SYN</i>	usato nella fase di connessione : a) SYN=1, ACK=0 richiesta connessione b) SYN=1, ACK=1 connessione accettata
<i>FIN</i>	usato per rilasciare una connessione

<i>Window size</i>	usato per il controllo di flusso, essendo questo di tipo sliding window questo campo specifica la dimensione della finestra
<i>Checksum</i>	simile a quello di IP, il controllo include lo pseudoheader
<i>Urgent pointer</i>	puntatore ai dati urgenti
<i>Options</i>	fra le più importanti troviamo: a) dimensione massima dei segmenti da spedire b) uso di selective repeat o del go-back-n c) uso di NAK

Nel calcolo del checksum entra anche uno *pseudo header* in violazione della gerarchia, dato che TCP in questo calcolo considera anche gli indirizzi IP:

32 bit		
Source IP Address (indirizzo a 32 bit)		
Destination IP Address (indirizzo a 32 bit)		
0	Protocol (=6) (codice numerico del protocollo, fisso a 6)	TCP Segment Length (il numero di byte del segmento TCP header compreso)

6.4.2.1. Attivazione connessione:

Si utilizza il three way handshake:

- a) una delle due parti esegue le primitive *listen()* e *accept()*, rimanendo così in attesa di una richiesta di connessione su un determinato port number, e se arriva accettandola;
- b) l'altra parte esegue la primitiva *connect()* specificando host e port number di destinazione insieme a tutti i parametri utili, come la dimensione max del segmento, questa primitiva consiste nell'invio di un segmento TCP con il bit SYN=1 e ACK=0;
- c) quando il destinatario riceve il segmento l'entity di livello transport controlla che ci sia un processo in ascolto sulla porta specificata, se c'è consegna il segmento e risponde con un segmento SYN=1, ACK=1 altrimenti risponde con un segmento con RST=1;

6.4.2.2. Rilascio connessione:

Il rilascio della connessione avviene considerando la connessione full duplex come una coppia distinta di connessioni simplex, e avviene in questo modo:

- a) una delle due parti non ha più nulla da trasmettere e invia un segmento con FYN=1;
- b) quando il segmento viene confermato la connessione viene rilasciata;
- c) quando anche l'altra parte completa lo stesso procedimento e rilascia la connessione la connessione full duplex è terminata;

Per risolvere il problema dei due esercizi si usano i timer impostati al doppio del tempo di vita massimo di un pacchetto.

6.4.2.3. *Politica di trasmissione:*

E' interessante notare che a differenza dei normali protocolli a finestra scorrevole, in quello di TCP la dimensione della finestra non dipende dal numero di ack inviati, ma viene stabilita mano mano attraverso un dialogo tra sorgente e destinazione; difatti quando la destinazione invia un ack, specifica attraverso il campo Window Size quanti altri byte possono essere spediti.

6.4.2.4. *Controllo congestione:*

TCP sa che se gli ack non tornato in tempo ciò è dovuto alla congestione della subnet piuttosto che ad errori di trasmissione, è dunque preparato ad affrontare problemi di scarsità di buffer a destinazione e di congestione della subnet.

Il primo problema viene gestito con la finestra del buffer del ricevitore (vedi politica di trasmissione) il secondo invece è gestito mediante la finestra di congestione (*congestion window*), solitamente il mittente si regola sulla più piccola delle due.

La congestion window viene gestita in questo modo:

- a) il valore iniziale è pari alla dimensione del massimo segmento usato nella connessione;
- b) ogni volta che un ack ritorna in tempo la finestra aumenta del doppio fino ad un valore soglia (*threshold*) inizialmente paria a 64 Kbyte, dopodiché aumenta linearmente di 1 segmento alla volta;
- c) quando si verifica un timeout:
 - c.1) il valore di threshold viene impostato alla metà della dimensione della congestion window;
 - c.2) la dimensione della congestion window viene impostata alla dimensione del massimo segmento usato nella connessione;

6.4.3. *Il protocollo UDP:*

E' un protocollo non connesso e inaffidabile utile per inviare dati senza stabilire le connessioni (ad esempio in applicazioni di streaming) l'header è composto da:

32 bit	
Source Port	Destination Port
UDP Length	UDP Checksum

Per poter ottenere traffico ancora più veloce la funzione di calcolo del checksum può essere disabilitata.