**LIVELLO DI TRASPORTO**

Il livello di trasporto è da considerarsi implementato end to end.

Svolge funzione di multiplexing/de-multiplexing per le applicazioni sugli host attraverso i numeri di porta (16 bit, da 0 a 65535).

* Numeri noti (0-1023) assegnati al lato server. Due client diversi possono accedere contemporaneamente alla stessa porta ma non c’è ambiguità perché le socket sono diverse.
* Numeri registrati (1024 – 49151) assegnati alle applicazioni che ne fanno richiesta tramite protocolli proprietari.
* Numeri dinamici (49152-65535) assegnati al lato client.

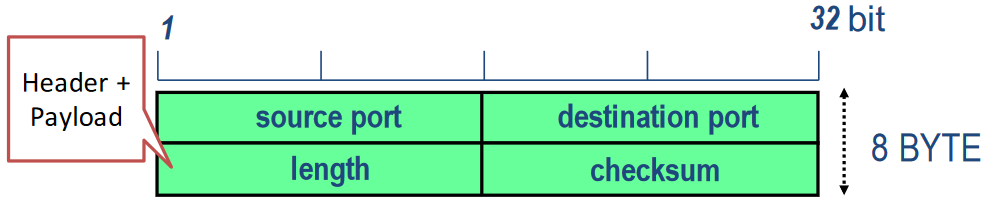
Il livello di trasporto implementa un sistema di buffering, il sistema operativo crea una coda in entrata e una in uscita.

**UDP (User Datagram Protocol) – RFC 768**

Ogni socket UDP è identificata solo da indirizzo e porta di destinazione, quindi due pacchetti provenienti da processi/host diversi ma con quei due campi uguali saranno indirizzati alla medesima socket sul lato server.

Header da 8 byte molto più leggero di quello di TCP (20 byte), quindi usato da applicativi che hanno bisogno di velocità come il DNS. Viene anche usato per trasportare informazioni di controllo e di gestone della rete in caso di stress/congestione.

Protocollo non orientato alla connessione, appena ha l’IP invia il dato senza aprire un canale. Non affidabile, non prevede controllo di flusso né rinvio di pacchetti persi o corrotti. Prevede comunque multiplazione e un minimo controllo d’errore sugli header ma senza correzione.



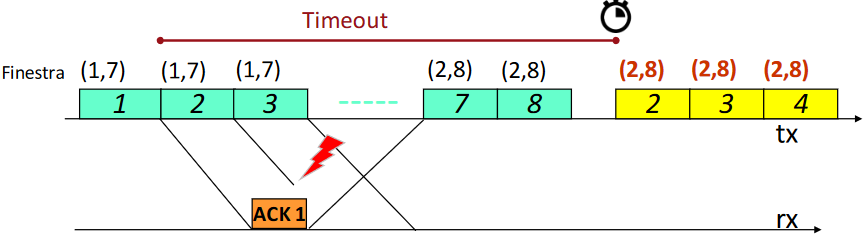
È un protocollo più veloce, semplice (no controllo connessione) e leggero (header minore) di TCP.

Il campo checksum (16 bit) contiene il risultato (calcolato dal mittente) di un calcolo che tiene conto di dati, pseudo-header e l’header; serve all’entità ricevente per controllare l’integrità del pacchetto. Mittente: sommo e complemento. Ricevente: sommo (compreso il checksum che è il complementare della somma): esce tutti 1, complemento ed esce 0 se è tutto corretto, altrimenti il pacchetto viene rifiutato.

**CONTROLLO D’ERRORE**

ACK = messaggio ricevuto correttamente, NACK = non correttamente = necessità di ritrasmettere. Questi messaggi di controllo hanno bisogno di un canale di trasmissione.

* Protocollo **Stop-And-Wait**: utilizza solo ACK e Timeout. Prevede la numerazione dei pacchetti e degli ACK per evitare di considerare pacchetti trasmessi più di una volta. Non adatto se ci sono alti ritardi di propagazione.
* Protocollo **Go-Back-N**: se il riscontro del primo pacchetto arriva entro la fine della finestra, questa scorre di una posizione. Se un pacchetto si perde, allora viene ritrasmesso tutto a partire dal pacchetto perso (back-N). Se non ci sono errori la trasmissione è continua (efficienza 100%). Quando termina la finestra, la trasmissione si blocca in attesa di un nuovo ACK o della scadenza del Timeout.



Ignora i pacchetti fuori sequenza nel caso ci sia un problema di trasmissione. Può esserci ACK collettivo. La finestra ottimale per la trasmissione continua corrisponde con il RTT.

Il NACK in teoria fa risparmiare tempo ma non posso essere sicuro del SN del pacchetto mancante a meno di valutare la sequenza (che non sempre è garantita).

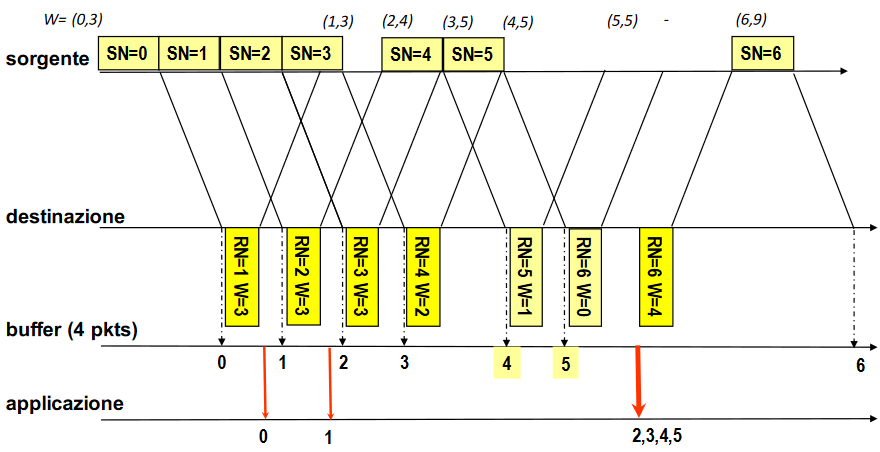
Gli ACK possono anche essere inseriti nei pacchetti che viaggiano in direzione opposta (piggy-backing). Contiene anche RN: numero di sequenza del pacchetto atteso in direzione opposta.

È necessario inizializzare RN e SN e scambiare questa informazione, quindi deve esserci un momento non equivocabile t=0.

**CONTROLLO DI FLUSSO**

Slinding Window Flow Control: controllo di flusso a finestra mobile. I riscontri vengono inviati quando i pacchetti vengono letti (tolti dal buffer) dal livello superiore. Se questo assorbimento è lento iniziano a scadere i time out.

Per evitare questo problema, viene diviso il controllo d’errore dal controllo di flusso tramite l’introduzione del campo W, che indica lo spazio rimanente nel buffer.



W può anche contenere un valore fittizio per regolare il flusso, o aspettare che il buffer sia pieno oltre una certa frazione prima di segnalarlo al trasmettitore.

Ripetizione selettiva: ACK specifici per ogni pacchetto vengono memorizzati in un buffer del mittente. Se ne manca qualcuno viene ritrasmesso il singolo pacchetto interessato.

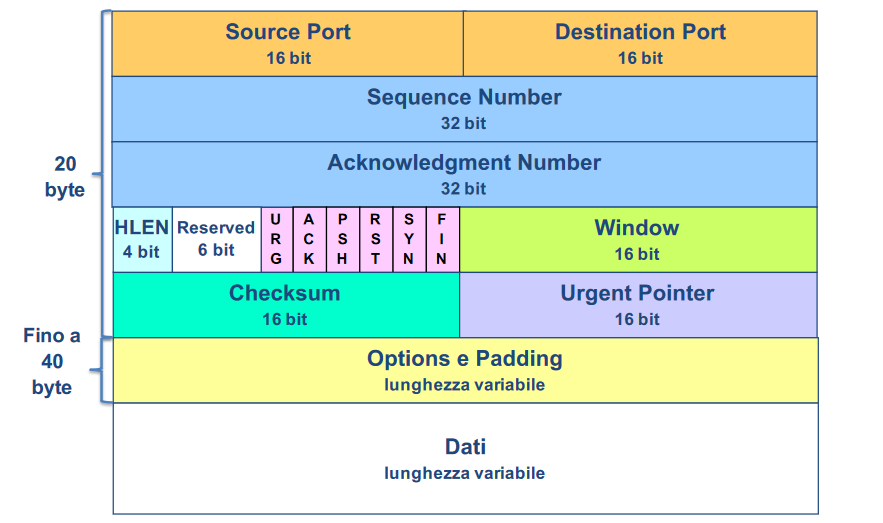
**TCP (Transmission Control Protocol) – RFC 793**

Trasmissione in sequenza, senza errori o perdite di dati.

Controllo di congestione end-to-end, che limita il traffico in rete e impone la condivisione equa delle risorse.

* Orientato alla connessione.
* Full duplex.
* Si appoggia su rete Datagram.
* Trasmette flussi continui di dati convertiti in segmenti di dimensione variabile che possono essere trasmessi in rete. Vengono formati frazionando periodicamente il contenuto del buffer.
* Numera ogni byte trasmesso. Implementa quindi sia SN che RN.

**SEGMENTI TCP**



**Sequence Number**: numero di sequenza del primo byte nel payload.

**ACK Number**: (se flag ACK attivo) numero del prossimo byte che si intende ricevere.

**HLEN**: Specifica lunghezza header TCP che deve essere un multiplo intero di 32 bit.

**URG**: Se vi sono dati urgenti viene attivato e il campo Urgent Pointer punta all’ultimo byte urgente nei dati.

**PSH**: attivo quando il trasmettitore intende usare il comando di push. Il pacchetto acquista priorità speciale, per questo è deprecato e viene ignorato. Viene attivato nell’ultimo segmento che svuota il buffer.

**RST**: resetta la connessione senza un tear down esplicito. Indica che la porta richiesta è chiusa.

**SYN**: usato durante il setup della connessione.

**FIN**: chiusura esplicita della connessione.

**Options & Padding**: riempimento fino a multipli di 32 bit e varie altre funzionalità, ad esempio comunicare il MSS durante il setup (default 536 byte) o il fattore di scala della finestra (il valore di default è 1. Il campo Window viene moltiplicato per 2 elevato al valore del campo fattore di scala).

**SETUP DELLE CONNESSIONI**

Viene detto “Handshake a tre vie”:

1. Il server fa una Passive Open comunicando al TCP locale che è pronto per accettare connessioni. Apre una socket costante verso una porta nota e rimane in ascolto. Il client comunica al TCP locale che desidera effettuare una connessione verso un dato socket (apre socket dinamica verso socket costante). Successivamente, manda un numero di sequenza iniziale attivando il flag SYN, specificando i parametri della connessione.
2. Il server riceve il messaggio e svolge le operazioni di routine (allocazione buffer, variabili TCP) oltre a rispondere riscontrando il numero di sequenza lato client e aggiungendo il proprio numero estratto casualmente per la connessione in senso opposto (anche questo pacchetto vede il flag SYN attivato).
3. Al momento della ricezione del secondo pacchetto, il mittente imposta la connessione e alloca il buffer. Procede infine con il riscontro del numero di sequenza iniziale dal lato server. In questo terzo pacchetto il flag SYN sarà 0 e il campo di dati può già contenere informazioni. Da questo momento, le applicazioni da entrambi i lati possono accedere ai buffer.

**PORTE NOTE**

22 **SSH**

21 **FTP** controllo

20 **FTP** dati

23 **Telnet**

25 **SMTP**

53 **DNS**

110 **POP**

143 **IMAP**

**TEAR DOWN**

L’entità TCP che chiude la connessione invia un messaggio con il flag di FIN attivo, che deve essere riscontrato da un ACK.

Se la connessione viene chiusa in una direzione, questo non implica la chiusura dall’altro lato. Per chiuderla del tutto serve appunto un altro messaggio FIN (riscontrato).

Se il flag RST è attivo, la connessione viene considerata non affidabile e di conseguenza viene chiusa automaticamente in entrambe le direzioni.

**CONTROLLO DI FLUSSO TCP**

Il TCP ricevente controlla il flusso di quello trasmittente.

Lo spazio libero nel buffer ricevente si chiama Receive Window (RCWND), è la dimensione del buffer – i dati non ancora assorbiti dall’applicazione.

Dal lato del trasmettitore la finestra non è un parametro fisso, viene modulata dinamicamente in base ai riscontri.

Ricevitore:

ultimo byte ricevuto – ultimo byte letto <= dimensione buffer

dimensione buffer – byte ricevuti = RCWND

Mittente:

ultimo byte mandato – ultimo byte riscontrato <= RCWND

* Silly Window Syndrome (lato ricevitore): buffer di ricezione lento e sempre pieno. La finestra sarà molto piccola e il trasmettitore dovrà sempre trasmettere pochi dati con molto over head. Si risolve con l’algoritmo di Clark: il ricevitore mente indicando finestra nulla fino a che non si è svuotato abbastanza il buffer (dimensione MSS di solito).

Quando la finestra è nulla il mittente deve comunque continuare a inviare byte di dati, in modo da poter ricevere aggiornamenti sulla dimensione della finestra da parte del ricevitore, in quanto questa informazione la si ottiene solo dagli ACK.

* Silly Window Syndrome (lato trasmettitore): l’applicazione genera dati lentamente e i segmenti sono piccoli. Viene risolto con l’algoritmo di Nagle: dopo un primo invio di pochi dati, questi vengono inviati solo se il pacchetto è grande abbastanza da riempire un MSS oppure solo quando si riceve un ACK.

**CONTROLLO D’ERRORE TCP**

Serve a recuperare pacchetti persi in rete la cui causa principale è l’overflow di una delle code dei router a causa della congestione. Implementa il Go-Back-N / RS con Timeout.

Differenze con UDP:

* TCP mantiene nel buffer i segmenti finché non sono in sequenza, aspettando quelli che gli mancano in mezzo.
* ACK collettivi.

Il valore del Timeout è un parametro critico, il valore ottimale viene calcolato stimando il RTT utilizzando gli algoritmi di Karn e Jacobson (si basano sulla stima del RTT medio e deviazione standard dal lato del trasmettitore: Timeout = SRTT + 4SDEV, valore di default 1s).In caso di ritrasmissione, il calcolo del Timeout viene affidato all’algoritmo di Karn: RTT non viene aggiornato, il Timeout cresce esponenzialmente fino a un valore massimo, dopo un numero massimo di ritrasmissioni la connessione viene chiusa.

Se i rallentamenti fossero dovuti a cause esterne e non a congestioni (esempio: collegamenti radio) questo sistema di controllo causerebbe più problemi che altro. TCP assume che quando qualcosa viene perso è a causa della congestione.

**PERSISTENZA**

Se RCWND viene fissata = 0 la trasmissione viene interrotta fino ad ulteriore avviso.

Se si perde l’ACK che segnala la riapertura della finestra la connessione rimane bloccata, quindi per sicurezza si usa un timer di persistenza. Alla scadenza del timer viene inviato un segmento di sonda (Probe).

**CONTROLLO DI CONGESTIONE TCP**

I meccanismi di controllo di congestione sono implementati solo nelle entità TCP locali, quindi è di tipo end-to-end. Di conseguenza, ogni perdita è considerata dall’entità TCP come conseguenza di congestione della rete.

*Esistono meccanismi di controllo di congestione supportati parzialmente dal livello di rete. In questo scenario, i router possono:*

* *Mandare un “choke packet” al mittente, indicando il proprio stato di congestione.*
* *Attivare un flag nel pacchetto in modo da notificare il destinatario della connessione della congestione su quel nodo (informazione che dovrà essere rimandata al mittente). Questo metodo richiede almeno un RTT per la notifica.*

*A volte il router comunica ai connessi la frequenza trasmissiva che può supportare.*

Nella sua implementazione standard, TCP regola la finestra di trasmissione implementando la CWIND (Congestion Window) che varia in base agli eventi che osserva. Il trasmettitore non può inviare più dati del minimo tra RCVWND e CWND.

Esistono due fasi:

**Slow Start (CWND < SSTHRESH):**

* All’inizio della trasmissione, il trasmettitore pone la CWND = 1 MSS di default e SSTHRESH a un valore molto alto, per partire in slow start.
* Ad ogni ACK ricevuto CWND++, aumento esponenziale.
* Continua così fino al primo evento di congestione o fino a che non raggiunge SSTHRESH.
* Durante questa fase il Rate istantaneo è R = CWND/RTT.

**Evento di congestione:**

* Su un link, la somma dei ritmi di trasmissione è maggiore della sua capacità.
* Scade un Timeout di ritrasmissione:
  + SSTHRESH = max (2MSS, FlightSize/2). FlightSize è circa = CWND. Questo valore è una stima della finestra ottimale che eviterebbe futuri eventi di congestione.
  + CWND = 1MSS
  + Si ritorna in Slow Start.
  + Si attiva il Go-Back-N per il segmento il cui timer era scaduto.

**Congestion Avoidance (CWND > SSTHRESH)**

* Ad ogni ACK, CWND viene incrementato di 1/CWND: incremento lineare della finestra di congestione per ogni blocco di dati inviati.

