LIVELLO TRASPORTO

Consente di rendere disponibili al layer applicazioni le due modalità connectionless e connection oriented (che non è supportata dal livello IP)

- Permette connessioni tra *processi* (mentre l'IP permetteva connessioni tra host) su base $end\text{-}to\text{-}end$
  I pacchetti a questo livello sono chiamati datagrammi oppure più correttemente *segmenti*

I due protocolli sono:

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

UDP

Ideato per servizi $connection\text{-}less$

- Si appoggia per questo direttamente a IP

Caratteristiche

- Più leggero e veloce (a discapito di alcune funzionalità): utilizzato per quei servizi in cui si richiede un tempo di ritardo end-to-end basso (es. streaming - a scapito magari della qualità)
- Non affidabile (un datagramma viene scartato se arriva corrotto ad esempio) [no controlli flusso o congestione]
- Non aggiunge funzionalità specifiche rispetto al livello IP
- One-Way (non prevede messaggio di conferma - interessa quindi curare bene la parte destinazione)
- Connectionless

Se il livello IP permette di identificare l'interfacce di rete (dispositivi a cui i flussi informativi sono destinati) senza però distinguere la tipologia del dato consegnato, ecco che il livello TCP permette di identificare il processo software di riferimento che si richiede (attraverso la lettura del numero di porta)

$$\begin{array}{ccccc} Rete(IP) & \longrightarrow & infrastruttura\ (interfaccia) & \longrightarrow & \text{aggregazione (Multiplexing)} \\ Trasporto(TCP) & \longrightarrow & processo\ software\ (lettura\ porta) & \longrightarrow & \text{separazione flussi (Demultiplexing)} \end{array}$$

Demultiplexing cioè separazione dei flussi verso le **porte logiche** [socket] (identificate ciascuna con un numero) che sono etichette che identificano uno specifico processo che risiede nella macchina di destinazione. Ad esempio servizio mail, accesso database, accesso remoto... questa suddivisione logica ci permette di avere queste sessioni attive contemporaneamente

Utilizzi:

- Streaming audio/video
- Aggiornamento e gestione tabella routing
  - Supponendo invio di pacchetti brevi (vedi dopo)
- Invio pacchettini in broadcast in operazioni tipo DHCP

> *Per limitare gli errori, conviene **trasmettere pacchetti piccoli** (si riduce così la probabilità di errore)*

PACCHETTO

$\longleftarrow$ 32 bit $\longrightarrow$

- Porta sorgente: gestione multiplazione
- Porta destinazione: gestione demultiplazione
- Length: lunghezza totale in bit di cosa vogliamo fare
- Checksum: controlla gli errori solo dell'$header$ (non dei dati) - giusto per capire se i datagrammi in arrivo sono corrotti circa le informazioni fondamentali ovvero indirizzo IP sorgente/destinazione oppure numero di porta. Se questo controllo non ha esito positivo, il datagramma viene scartato. In particolare in UDP per il calcolo di checksum si utilizza lo **pseudo-header** (vedi dopo)

PSEUDO HEADER (CONTROLLO ERRORI PER IMPLEMENTARE IL CHECKSUM)

Per identificare in modo univoco la destinazione finale dei segmenti in arrivo, si deve eseguire il seguente abbinamento logico:

$$sorgente + numero\ di\ porta\ destinazione$$

- Basta solo la porta di destinazione in UDP perché questo protocollo è pensato per essere one-way (monodirezionale) [a differenza di TCP]

Lo pseudo-header per calcolare il campo di checksum utilizza le seguenti informazioni:

- Indirizzo sorgente
- Indirizzo destinazione
- Informazioni lunghezza datagramma UDP (escluso lo pseudo-header)
- Campo di riempimento (fittizi), detto ZERO
- Numero di porta

> *Viene costruito alla sorgente e ripristinata poi alla destinazione, ma non viene trasferita nel datagramma vero e proprio che viaggia (esistenza locale su base end-to-end)*

PERCHE' SI IMPLEMENTA?

- Non sempre si può contare "alla cieca" sulla qualità dei mezzi trasmissivi, meglio fare un controllo rapido

Ricordiamo comunque che tali controlli di errori non sono volti a una successiva correzione dei dati errati: in caso di errore come già detto i datagrammi "corrotti" vengono scartati (quindi non ci si allontana dalla filosofia UDP)

TCP

$Transmission\ Control\ Protocol$
SCOPO: realizzare collegamenti tra **processi** remoti, su base $end\text{-}to\text{-}end$

- Permette di implementare un servizio **affidabile** (e quindi) $connection\text{-}oriented$
- Il collegamento è full-duplex, cioè bidirezionale

Il protocollo prevede una fase iniziale di *setup* per condividere le condizioni operative circa le modalità per un corretto (e sicuro) scambio di informazioni. Essa prende il nome di

$$\mathrm{HANDSHAKE}$$

- Per ridurre i tempi di setup (handshake) iniziali, i messaggi devono essere brevi

CONNECTION-ORIENTED: COME SI IMPLEMENTA?

Per gestire la modalità connection oriented, si ricorre alla seguente soluzione:

- in fase di *trasmissione* ci si deve preparare a trasmettere e ritrasmettere in caso ci fossero errori: infatti il protocollo è affidabile, quindi se alla destinazione arriva un datagramma corrotto, se ne chiede il re-invio
  - Pertanto dovremo salvare le informazioni finché non arriva la conferma dell'arrivo corretto

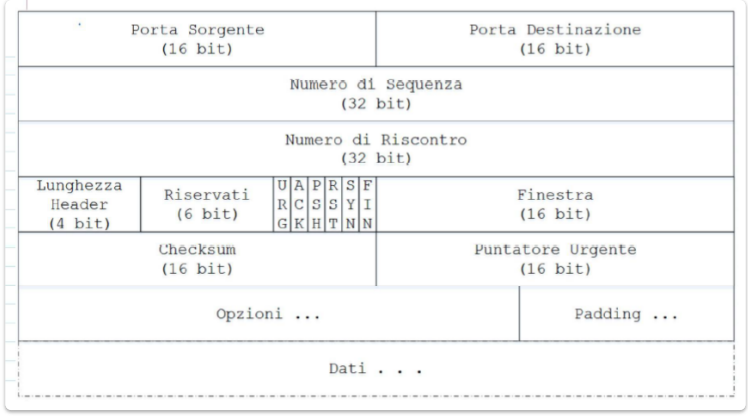
> ✓ **Due buffer distinti in trasmissione**
>
> Dobbiamo implementare **due buffer**:
>
> > *Il primo relativo ai pacchetti che devono essere trasmessi*
> >
> > *Il secondo relativo ai pacchetti che sono già stati trasmessi e siamo in attesa della conferma*

- in fase di *ricezione* è più semplice

> **Un solo buffer in ricezione**
>
> > *Basta un solo buffer di **riordino** dove i pacchetti vengono memorizzati secondo il corretto ordine di generazione*

> *Nota: il livello TCP non effettua la frammentazione. Quando questa è richiesta viene ancora gestita su base end-to-end utilizzando un algoritmo MTU (le dimensioni massime vengono eventualmente stabilite in fase di setup/handshake)*

DATAGRAMMA (SEGMENTO) TCP



Lunghezza testata: 5 campi di 4 byte $\rightarrow$ 20 byte

Le due porte individuano in maniera univoca il preciso trasferimento tra due istanze:

- Porta sorgente (16 bit): serve per individuare il *processo* interessato alla connessione (lato sorgente) - è una etichetta numerica
- Porta destinazione (16 bit): indica il processo destinazione
- Numero di sequenza (32 bit): Serve per associare al pacchetto il numero corrispondente alla sua posizione nel flusso informativo complessivo. Sarà utile poi in ricezione per riordinare in modo coerente l'intero flusso (pacchetto), basandosi appunto su questi indici

  > - In genere il primo datagramma si sceglie un numero tra 0 e $2^{32}-1$ e poi dai successivi si ordinano a partire da quel numero
  > - Ricordiamo che il processo di ordinamento non è previsto dai livelli inferiori (IP), quindi questa funzionalità agisce da "maschera", per implementare un servizio non previsto appunto

- Numero di riscontro (32 bit): utilizzato per informare che è stato ricevuto correttamente un datagramma andando a chiedere il successivo (ha logica quindi quando flag $\mathrm{ACK}=1$)

  Numeri di riscontro
  - Dato che il TCP è full-duplex, due host **A** e **B** che comunicano possono sia ricevere che trasmettere i dati nello stesso momento.
  - Ogni segmento che l'host A invia all'host B, ha un numero di sequenza relativo ai dati che A invia a B. Il **numero di riscontro (RIS)** che l'host B inserisce nel suo segmento è pari a:

    $$RIS_B = SEQ_A + num\_dati_A$$

  - Supponiamo che l'host B riceva da A un segmento contenente nel campo dati 536 byte, numerati da 0 a 535, e supponiamo che B risponda ad A inviando un segmento. L'host B riscontra i byte ricevuti da A inserendo il valore 536 nel campo *numero di riscontro* del segmento che invia ad A.

- Lunghezza header (4 bit): campo necessario perché la testata *può avere campi opzionali*. La lunghezza minima è comunque fissata a 20 byte (la massima è 60 byte)
- Riservati (6 bit): informazioni riservate utilizzate per altri scopi
- Flags (6×1 bit):

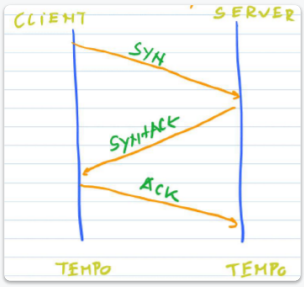
  > - PSH: (push) si richiede un invio immediato dei dati
  > - URG: indica se effettivamente il datagramma è urgente
  > - ACK: se vale 1, allora il datagramma è stato ricevuto correttamente. Si procederà quindi a inserire il numero di riscontro
  > - RST: interruzione/reset trasferimento dati (si attiva causa evento imprevisto: non è stato inviato/ricevuto ancora tutto il flusso ma si deve interrompere)
  > - SYN: richiesta di *apertura di un collegamento*
  > - FIN: richiesta di *chiusura di un collegamento*

- Finestra (16 bit): Dimensione sliding window - massima dimensione che il destinatario (processo) può accettare senza causare congestioni
- Checksum (16 bit): controllare integrità dei dati ricevuti, identificando eventuali errori (ha una grande importanza, essendo TCP affidabile)
- Puntatore urgente (16 bit): Indica il numero di byte che devono essere trasferiti immediatamente (ha senso quando $\mathrm{URG}=1$)
- Options (16 bit): funzionalità aggiuntive
- Padding (16 bit): dati "fittizi" di completamento (cfr. UDP)

APERTURA E CHIUSURA COLLEGAMENTO

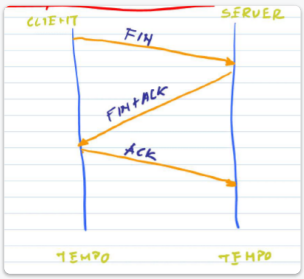
Apertura: THREEWAY HANDSHAKE

1 - Client invia un pacchetto di richiesta ponendo $\mathrm{SYN} = 1$ e un numero di sequenza casuale tra i $2^{32} - 1$ possibili
2 - Server riceve e nel caso risponde positivamente con un pacchetto $\mathrm{SYN} + \mathrm{ACK}$, in cui appunto $\mathrm{SYN}{=}1, \mathrm{ACK} = 1$. Il numero di sequenza è scelto in modo casuale e il numero di riscontro è uguale al numero di sequenza del pacchetto SYN aumentato di uno
3 - Client risponde con un messaggio di ACK. Questo pacchetto può includere dati (piggyback)



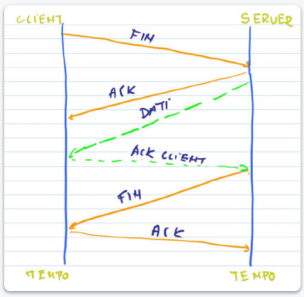
Chiusura: THREE-WAY oppure FOUR-WAY HANDSHAKE

THREE-WAY: server chiude subito collegamento



FOUR-WAY: server non può chiudere immediatamente il collegamento

Il server nonostante la richiesta di FIN del client continua a mantenere aperto il collegamento (perché magari deve concludere delle faccende), notificando però (perlomento) che ha ricevuto la richiesta inviando ACK (se deve ancora mandare dati, continua anche a inviarli). Continua lo scambio di ACK tra i due finché finalmente il server è disponibile alla chiusura e quindi manda FIN, procedendo poi in modo analogo al three-way



CONFRONTO ARCHITETTURE PROTOCOLLARI: TCP-IP vs ISO-OSI

- Applicazione, Trasporto, Rete: match perfetto tra i due
- Sessione, Presentazione: non previsti in TCP-IP (motivi storici, cfr. primi appunti TCP-IP)
- Collegamento, Fisico: non ben specificato nel TCP-IP, è più generico → si adattano alle interfacce (diverse, possibili) tra un host e la rete [nota: se fossero inadatti, il livello IP procede con la frammentazione a rendere adatto]

ISO-OSI: quella di riferimento mondiale
TCP-IP: quella utilizzata