

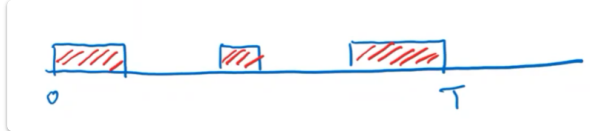
RETI PER TRASMISSIONI DATI (Data Network)

Sono reti che permettono il collegamento tra dispositivi (computer, oggetti, sensori...) effettuando uno scambio di bit. Nascono quindi per il traffico dati (a differenza della rete telefonica che "si adattava" con l'utilizzo di filtri talvolta analogici)

TRAFFICO DATI

Ha alcune caratteristiche che lo rappresentano:

- **Intermittenza** nel tempo: il flusso d'informazione non è continuo: \exists tempi in cui il collegamento non è attivo. Diventa vitale "giocarsi bene" i tempi di scollegamento e a tal proposito esistono tecniche apposite per non influenzare la qualità percepita, mantenendo comunque inevitabilmente l'intermittenza



- **Asimmetria**: le linee *up* (client-server) e *down* (server-client) possono essere impegnate in maniera diversa. Ad esempio, la query che inviamo a un server (verde) avrà un tempo di invio molto basso, mentre la risposta del server sarà eventualmente anche piuttosto lunga (rossa). Quindi le due linee (\longleftrightarrow) hanno un utilizzo diverso



- **Integrità**: è necessario garantire la ricezione corretta di un flusso dati (deve arrivare la giusta stringa di bit a destinazione per evitare che ci siano errori di rielaborazione)

SERVIZIO

Un servizio è un insieme di operazioni di base che un livello i offre al livello immediatamente superiore $i + 1$

- Il livello inferiore tra i due viene detto *provider* (procura il servizio)
- Il livello superiore tra i due viene detto *utente* (riceve il servizio)

PROTOCOLLO

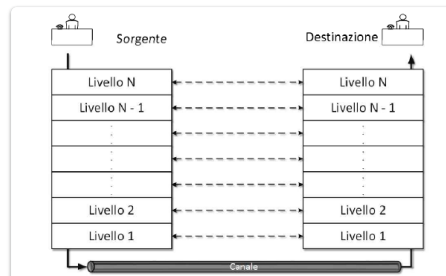
Insieme di regole da rispettare volte a far svolgere correttamente una serie di azioni/servizi a determinati dispositivi coinvolti

- Servono per poter implementare i servizi che risiedono su macchine diverse

ARCHITETTURA A LIVELLI

Per rendere più flessibile e ottimizzato il progetto di una rete le operazioni base sono state suddivise in gruppi omogenei e ordinate in una *struttura gerarchica*, detta a livelli.

- Tutti i dispositivi che colloquiano avranno la stessa struttura, perché la modalità d'interazione viene **standardizzata** seguendo un certo protocollo ben definito



- La pila è flessibile perché posso modificare un certo livello i senza modificare tutta la pila, ma agendo solo su esso (questo ha aiutato la nascita dell'open-source)
- Il livello j della *sorgente* può "apparentemente" comunicare in maniera diretta con il livello j della *destinazione* (in realtà i dati passano dai livelli più alti verso quelli più bassi, fino al *canale* di trasmissione effettivo - come un cablaggio di rame - che giunge fino al destinatario e poi l'informazione risale i livelli)

OPERAZIONI DI BASE TRA STATI DELLO STESSO LIVELLO

Le operazioni tra stati (*layer*) dello stesso livello sono attivate mediante alcune *operazioni primitive* (comandi).

- Possono essere di diverso tipo
- Note:
- XXX rappresenta il servizio a cui la primitiva si riferisce
- "." delimita la tipologia di primitiva

Ecco i comandi:

- `XXX.REQUEST` è una richiesta di *attivazione del servizio* XXX
- `XXX.INDICATION` è una *notifica* della richiesta di attivazione del servizio XXX (rende il terminale consapevole di cosa è chiamato a fare)
- `XXX.RESPONSE` è una autorizzazione a *iniziare* l'operazione relativa al servizio XXX
- `XXX.CONFIRM` è la *conferma* di utilizzo del servizio

☰ Esempio: Roberto chiama Anna

1. Roberto fa il numero di Anna → `CONNECT.REQUEST` (viene mandata alla rete)
2. Arriva la notifica ad Anna → `CONNECT.INDICATION` (la rete ha portato la richiesta fino ad ad Anna, e il suo telefono infatti glielo indica con lo squillo)
3. Anna risponde → `CONNECT.RESPONSE` (avverte la rete che ha accettato l'invito)
4. Arriva la conferma a Roberto dell'accettato invito → `CONNECT.CONFIRM` (Anna dice "pronto")
Da cui inizia il colloquio e poi ci sarà la chiusura del collegamento

TIPOLOGIE DI SERVIZI

CONNECTION ORIENTED

"Orientato Alla Connessione": è caratterizzato dal vincolo di avere in ricezione un flusso dati che preserva (mantiene) *lo stesso ordinamento* che ha alla sua origine (l'ordinamento dei pacchetti viene mantenuto dalla sorgente fino alla destinazione)

- Quando abbiamo tante reti non è garantito sempre tale servizio
- Si utilizza un *buffer* in ricezione se il servizio non viene garantito dal livello *trasporto* che permette di riordinare i pacchetti una volta (già) giunti a destinazione
 - Se si implementa a livello trasporto (cfr. TCP), non serve il buffer in ricezione

CONNECTIONLESS

Non è richiesto/garantito il rispetto dell'ordinamento del flusso di dati in ricezione (pacchetti)

- Utilizzato quando i pacchetti sono uno indipendenti tra loro o quando in generale non ci interessa troppo il preciso ordinamento
- Esempio: streaming

Nota (banale): l'unico caso in cui connection oriented e connectionless coincidono al 100%, è quando il flusso di dati è composto da un solo pacchetto (cfr. TCP-IP)

AFFIDABILE

Prevede notifica esplicita di ricezione (corretta) di un flusso informativo

- Permettono d'individuare eventuali errori/disturbi di rete o congestioni/overflow del router
- Esempio: ricevuta di ritorno

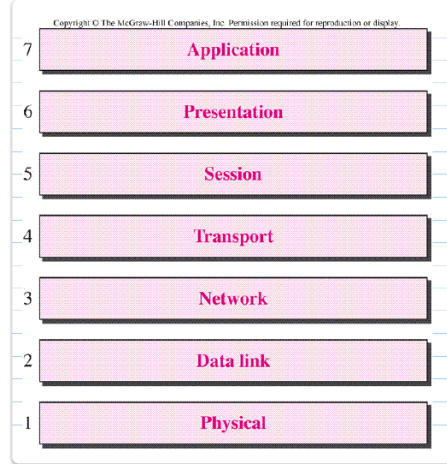
NON AFFIDABILE

Non prevede notifica di riscontro alcuna

- Una volta trasmesso il pacchetto, finisce lì
- Rischioso da implementare quando la qualità della rete è bassa (per reti invece buone e affidabili, conviene adottare questo approccio per sfruttare a pieno il vantaggio relativo alla *velocità* di trasferimento di risorse)

MODELLI ISO OSI

- Architettura a strati di tipo aperto (*Open Systems Interconnection*)
- Prevede 7 livelli



Il livello *utente* in generale è colui che richiede un servizio a livelli precedenti

- Anche il livello fisico, seppur nella pila sia il primo, è un *utente* per il livello ancora inferiore che è il mezzo fisico che permette l'effettivo trasporto tra due dispositivi
- Tutti i livelli tranne quello applicativo sono *provider* per il rispettivo livello successivo

LIVELLO 1 FISICO

Operazione di trasferimento/conversione di bit in segnali elettrici in fase di trasmissione e operazione di ricostruzione dei segnali elettrici in bit in fase di ricezione

LIVELLO 2 DATA LINK

Tra le varie funzionalità che vengono effettuate a questo livello c'è il *mantenimento dell'integrità dei dati*

- Prende come base di lavoro il flusso di bit che gli viene trasferito dal livello fisico e attiva le procedure rivolte a riconoscere eventuali errori in tale flusso (per garantire appunto se quanto ricevuto è stato corrotto dalla trasmissione o meno)

Un altro compito, soprattutto nelle reti locali, è relativo alla *condivisione dell'accesso*, ovvero in che modo uno stesso canale (supporto di trasmissione) può essere messo a comune tra più utenti ^{permette di} → aumentare l'utilizzo di un collegamento, cioè sfruttare in maniera più uniforme il collegamento, combinando appositamente le intermissioni dei vari flussi considerati

- ci sono tante tecniche per farlo (cfr. argomenti successivi): ordinata e casuale

Altre funzioni: framing, gestione indirizzi fisici (delle celle di memoria cioè), controllo errori...

LIVELLO 3: RETE

La sua funzionalità principale è quella di consentire a utenti remoti di costruire e usufruire dei collegamenti che la rete può offrire

- Permette quindi d'implementare il collegamento tra gli utenti

Una delle operazioni principali che deve svolgere è il **routing**, ovvero permettere il corretto instradamento dei pacchetti (basandosi su opportuni algoritmi)

LIVELLO 4: TRASPORTO

Gestisce le modalità di collegamento tra i due utenti: si può implementare a questo livello servizi connection oriented o connectionless a seconda del caso, permettendo il trasporto appunto dei dati.

Sono utilizzati a questo livello principalmente due protocolli, TCP e UDP

LIVELLO 5: SESSIONE

Permette di creare una sessione (anche temporanea) tra due utenti, eseguendo eventuali controllo preliminari per concedere l'autorizzazione o meno

- Ad esempio, quando compriamo un biglietto online: ci viene concesso ≈ 15 min. di tempo per completare l'acquisto → siamo in sessione con il sito per quell'intervallo, che ci ha autorizzato a procedere

LIVELLO 6: PRESENTAZIONE

Rende possibile l'interpretazione su dispositivi di varia natura di usufruire di formati diversi di generazione d'informazione

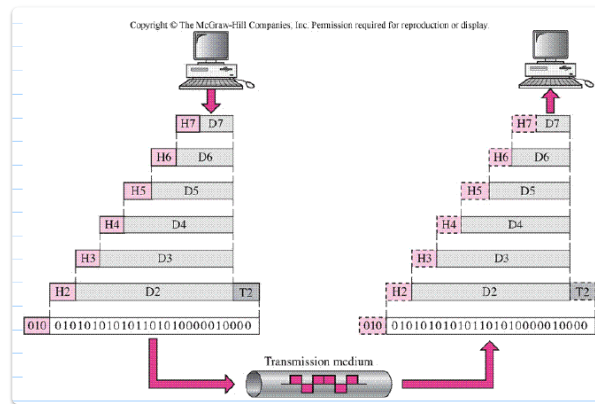
- Un esempio banale è la lingua (traduzione), ma anche l'adattamento di altre specificità

LIVELLO 7: APPLICAZIONE

Rappresenta le specifiche applicazioni installate/usufruite sui vari dispositivi

- Tecnologie di alto livello

GESTIONE DEI LIVELLI



Nota: *header* racchiude le regole che dovranno essere notificate al pari livello destinazione per interpretare il campo *data*

FASI

Invio:

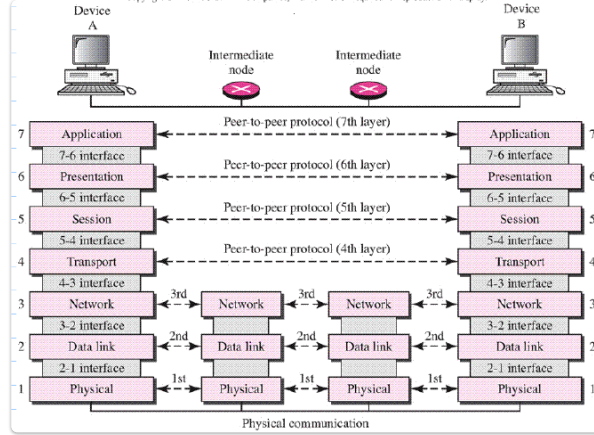
- Lo stato applicazione genera un pacchetto informativo composto da due parti: *header* e *data*
 - In particolare **H7** **D7**
 - Il pacchetto (blocco) viene trasferito al livello inferiore
- Lo stato presentazione interpreta le informazioni ricevute e aggiunge il proprio *header*
- ⋮
- *il procedimento continua...*
- ⋮
- Lo stato data-link (collegamento) si nota che è l'unico che può *introdurre nuove informazioni di supporto (istruzioni) anche in coda*: in particolare aggiunge anche lui un *header* in testa, ma (cosa nuova) aggiunge anche **T2**
 - Viene poi passato tutto a livello 1
- Lo stato fisico riceve le informazioni, converte il blocco in bit, aggiunge una ulteriore *testata* relativa a una serie di operazioni di sincronizzazione che saranno poi utili in destinazione per capire da dove iniziare per rileggere il flusso di bit (in questo caso contrassegnate con **010**) e passa tutto sul canale fisico di trasmissione, con le modalità specificate a seconda del mezzo fisico (ad esempio: impulso radio/luce *high* quando il bit vale 1, impulso radio/luce *low* quando il bit vale 0)
- Il mezzo fisico mette in connessione l'altra entità del collegamento

Ricezione (operazione opposta):

- Il livello fisico ricostruisce la sequenza di bit secondo le tipologie di segnali del mezzo fisico (utilizzando la *testata* per gestire correttamente il flusso)
 - Passa la *parte dati* al livello successivo
- Lo stato data link esegue la medesima operazione concettuale (variano le operazioni specifiche ovviamente)
 - Passa anch'esso la parte dati al livello successivo
- ⋮
- *il procedimento continua...*
- ⋮
- Lo stato applicazione è quello di arrivo, si conclude lo scambio

☹️: i collegamenti tra i vari livelli sono virtuali, solo il mezzo fisico è quello che effettivamente collega appunto fisicamente le due entità

ESEMPIO DI CONNESSIONE



Device A vuole comunicare con Device B attraverso due *router*

- Si suppone che la connessione tra A e B sia già stata definita (ovvero il livello rete ha già individuato il percorso di connessione A → B)

Note:

- I due Host A e B hanno disponibili per caratteristiche operative tutti e 7 i livelli a disposizione
- I due *router* operano solo fino al livello 3
 - Il livello *network* (rete) deve interpretare le richieste di connessione e individuare le porte di uscita per avere il collegamento migliore verso l'entità di arrivo. È la parte più intelligente del dispositivo di routing
 - Il livello *data link* (collegamento) deve essere attivo perché bisogna garantire quanto possibile l'integrità (e se lo metto su tutti dispositivi intermedi me ne accorgo prima se ci sono errori o meno)
 - Il livello *physical* (fisico) è necessario banalmente per lo scambio d'informazioni "atomiche", ovvero i bit che poi serviranno anche negli strati successivi

Tutti i livelli lavorano direttamente (peer-to-peer) con il livello medesimo del dispositivo di interfaccia
In particolare si nota come i livelli 4 – 7 sono virtualmente connessi tra loro (end-to-end)
I livelli 1 – 3 sono attivati concretamente tra tutti i vari dispositivi (link-to-link)

MODALITA' DI COMMUTAZIONE

Modalità attraverso le quali si trasferisce l'informazione attraverso la rete


- Inoltro e gestione del flusso informativo attraverso la rete
- 3 modalità (circuitto CC, messaggio CM, pacchetto CP)

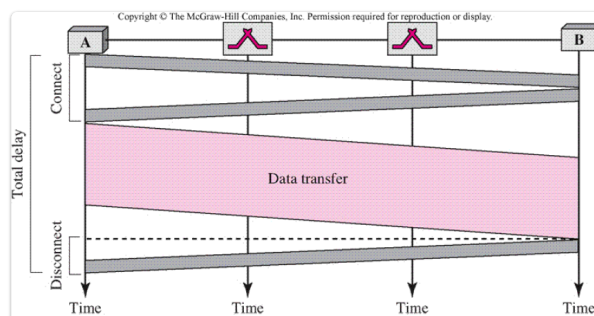
COMMUTAZIONE DI CIRCUITO (CC)

Associa a una coppia sorgente-destinazione un percorso *a uso esclusivo*

- La rete predispone cioè un percorso tra tutti quelli disponibili per comunicare tra A → B fino a che i due ne hanno necessità
 - Durante tale intervallo di tempo, il percorso suddetto è dedicato esclusivamente al collegamento tra A e B (non è condiviso con terzi)

- 😊: possibilità di iniziare la comunicazione ogni volta che se ne ha necessità senza ritardi o attese (che in genere sono dovute all'utilizzo di altri)
- 😞: se il traffico dati è sporadico, allora si ha uno spreco di risorse (ridotto utilizzo del collegamento → lato utente questo causa anche spese inutili)
- 😞: fase di setup → quando si richiede alla rete di riservare il percorso, essa deve predisporre tutto il necessario (quindi abbiamo un ritardo iniziale)
- 😞: fase di endup (chiusura) → tempo (ritardo) dovuto alla chiusura del collegamento predisposto

 opportuno quando il tempo di utilizzo è più grande della fase di setup ed endup



Come si vede in figura ci sono 3 fasi (asse verticale: tempo, asse orizzontale: percorso):

- Connessione (setup)
- Trasferimento (si spera più grande per un miglior sfruttamento del servizio)
- Disconnessione (endup)

▶ dobbiamo avere una continuità garantita (elettrica) una volta settato il percorso (fase rosa), per garantire la commutazione di servizio (con le altre modalità questo non sarà indispensabile)

🔗 impegno della rete su base *end-to-end*

COMMUTAZIONE DI MESSAGGIO

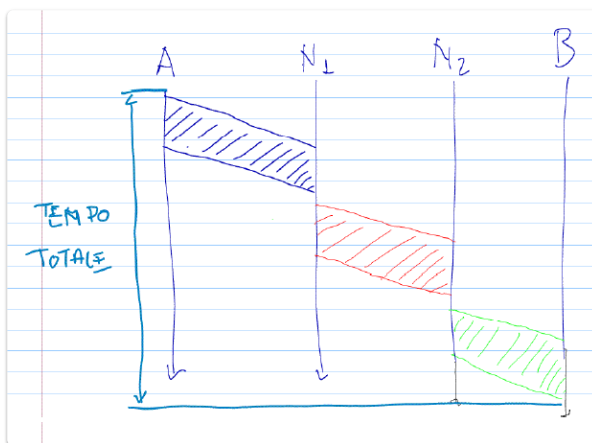
Creata per risolvere le carenze della precedente

Si lavora a livello di messaggio: si riserva l'utilizzo esclusivo di un collegamento *unicamente per il tempo impiegato per trasferire il messaggio*

- Impegno della rete **partizionato** (non esclusivo)

Vediamo schematicamente come funziona (due dispositivi A e B e due routers R_1 e R_2)

- Si suppone sempre che la fase di connessione sia già stata effettuata



🔗 impegno della rete su base *link-to-link*

😊: minor tempo di setup ed endup

😊: buono per collegamenti brevi

😞: richiesta modalità integrità flusso dati → si esegue il controllo di integrità unicamente *dopo aver ricevuto tutto il blocco* e i pacchetti che costituiscono il messaggio

- Vulnerabile quindi ai cambiamenti d'informazione/disturbi durante il percorso

- Causerebbe un tempo maggiore

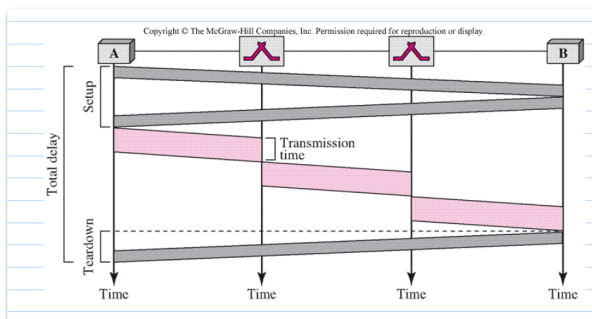
COMMUTAZIONE DI PACCHETTO

Nato per risolvere i problemi delle precedenti e offrire una alternativa ibrida

- L'unità di riferimento è il pacchetto (insieme di pochi bit)
- Ha due varianti di modalità

- Circuito virtuale
- Datagramma

CIRCUITO VIRTUALE



Fasi:

- Setup (invio e risposta)
- Trasmissione → su base *link-to-link* (per questo si chiama virtuale: apparentemente le due entità sembrano direttamente connesse, ma in realtà il collegamento non è esclusivo: viene condiviso con altri dispositivi)

😞 Questa condivisione può causare ritardi

😊 Si raggiunge però un utilizzo migliore: meno probabilità di errori, più controlli intermedi d'integrità...

😞 una sola fase di setup (per le fasi intermedie non serve)

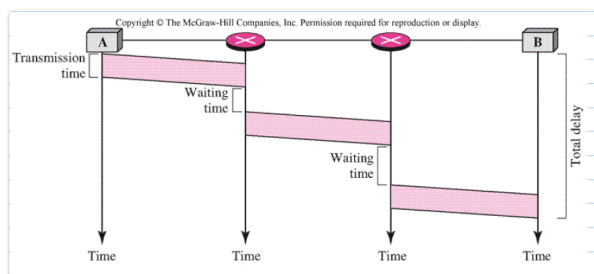
- Endup (tempo di terminazione)

👤 Quindi, vantaggio principale: *non esclusività (condivisione) dispositivi di rete* → collegamenti non esclusivi da due host

📌 si adatta bene alla modalità *connection oriented*, infatti i pacchetti sono inviati uno dietro l'altro → non si hanno disordini

DATAGRAMMA

Obiettivo: migliorare utilizzo della rete e contenere tempi di trasferimento



Non c'è fase di setup: ogni pacchetto può seguire una strada diversa, che si ritiene migliore per esso

😊: tempo di trasferimento (utilizzo rete) molto vicino al tempo di trasmissione della parte informativa

😞: non è garantita la *connection oriented*

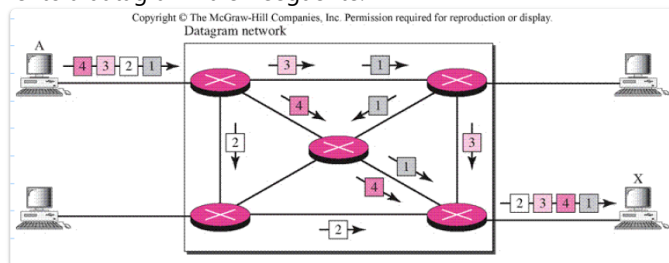
📌: adatto per la *connectionless*

😊: flessibilità errori → si gestiscono meglio eventuali guasti della rete: se un router comincia a essere congestionato (e quindi non riesce in tempi brevi a gestire le richieste), basta cambiarlo oppure indirizzare i pacchetti verso uno alternativo

😊: flessibilità scelta mezzo trasmissivo → si possono collegare ad esempio doppiini telefonici con fibre ottiche (un po' come succede con la fibra quando non arriva direttamente in casa: magari arriva all'armadio in strada e poi si collega da lì con il filo di rame fino in casa)

😊: è la modalità più utilizzata nel TCP-IP

Un esempio più dettagliato di instradamento a datagramma è il seguente:



Si nota come la sequenza 4321 → viene consegnata al *router di frontiera* (quello più vicino che gestisce l'utente).

- Esso decide le varie strade man mano che arrivano i pacchetti, a seconda di come si sta evolvendo la situazione (ogni volta si sceglie la migliore scelta tra quelle disponibili)
- In uscita sul terminale di arrivo X abbiamo un ordine dei pacchetti 2341 diverso da quello di invio
 - Se il servizio è *connection oriented*, si inserisce un buffer per riordinare

😊: lavoro in parallelo

- Quando viene inviato un pacchetto, si può procedere con il successivo. Lo stesso avviene in ricezione: appena viene ricevuto un pacchetto, si procede con la ricezione del successivo

- Non si aspetta cioè che venga mandato tutto il messaggio, ma si procede pacchetto per pacchetto

- In questo modo magari qualche pacchetto è "più difficile da leggere", però nel frattempo stiamo già procedendo col successivo, quindi mediamente i tempi sono più veloci

- Ci aiuta ad avere un tempo di utilizzo della rete quasi pari al tempo d'invio del pacchetto (più flessibilità)

