SAPIENZA
UNIVERSITÀ DI ROMA

UNIVERSITÀ "SAPIENZA" DI ROMA
FACOLTÀ DI INFORMATICA

Reti di Elaboratori

Appunti integrati con il libro "Computer Networking: A Top-Down Approach", J. F. Kurose, K. W. Ross

Author
Simone Bianco

23 marzo 2023

Indice

0 Introduzione **1**

1 Introduzione alle reti **2**
1.1 Rete, Host e Collegamenti 2
1.2 Struttura di Internet 4
1.3 Pacchetti, Commutazione e Instradamento 7
1.4 Misura delle prestazioni 9
1.5 Stack protocollare TCP/IP 14

2 Livello di Applicazione **18**
2.1 Principi delle applicazioni di rete 18
2.2 Web e Protocollo HTTP 21
2.2.1 Messaggi di richiesta e risposta 23
2.2.2 Versioni di HTTP 26
2.2.3 Cookies e Web Caching 27
2.3 Posta elettronica 28
2.3.1 Protocolli SMTP e MIME 29

Capitolo 0

Introduzione

Capitolo 1

Introduzione alle reti

1.1 Rete, Host e Collegamenti

Definition 1. Rete e Link

Una **rete** è un'infrastruttura composta da dispositivi detti **nodi della rete** in grado di scambiarsi informazioni tramite dei mezzi di comunicazione, wireless o cablati, detti **link (o collegamenti)**

Definition 2. Nodi di una rete

I **nodi** costituenti una rete vengono differenziati in **due macrocategorie**:

- **Sistemi terminali**, differenziati a loro volta in
  - **Host**, ossia un dispositivo di proprietà dell'utente dedicato ad eseguire applicazioni utente
  - **Server**, ossia un dispositivo di elevate prestazioni destinato ad eseguire programmi che forniscono un servizio a diverse applicazioni utente
- **Dispositivi di interconnessione**, ossia dei dispositivi atti a modificare o prolungare il segnale ricevuto, differenziati a loro volta in:
  - **Router**, ossia dispositivi che collegano una rete ad una o più reti
  - **Switch**, ossia dispositivi che collegano più sistemi terminali all'interno di una rete
  - **Modem**, ossia dispositivi in grado di trasformare la codifica dei dati in segnale e viceversa

In particolare, classifichiamo le varie tipologie di rete in:

- **Personal Area Network (PAN)**, avente scala ridotta, solitamente equivalente a pochi metri (es: una rete Bluethoot)

- **Local Area Network (LAN)**, solitamente corrispondente ad una rete privata che collega i sistemi terminali di un appartamento (es: una rete wi-fi o Ethernet). Ogni sistema terminale possiede un indirizzo che lo identifica univocamente all'interno della LAN.

  Si differenziano in **LAN con cavo condiviso**, ossia dove tutti i dispositivi sono connessi al router tramite un cavo comune, e **LAN con switch**, ossia dove tutti i dispositivi sono connessi ad uno o più switch, i quali a loro volta sono connessi al router

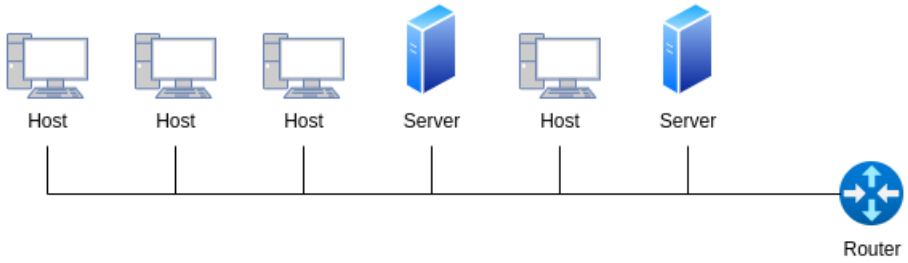
- **Metropolitan Area Network (MAN)**, avente scala pari ad una città
- **Wide Area Network (WAN)**, avente scala pari ad un paese o una nazione, solitamente gestita da un **Internet Service Provider (ISP)**.

  Si differenziano in **WAN punto-punto**, ossia collegante due reti tramite un singolo mezzo mezzo di trasmissione, e **WAN a commutazione**, ossia collegante più reti tramite più mezzi e dispositivi di collegamento

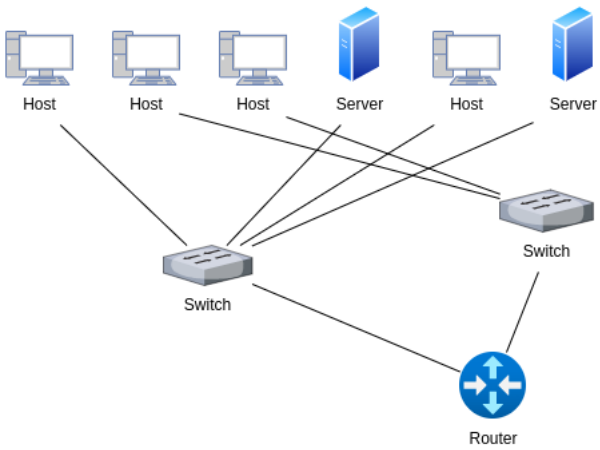
- L'**Internet**, avente scala globale

Esempi:

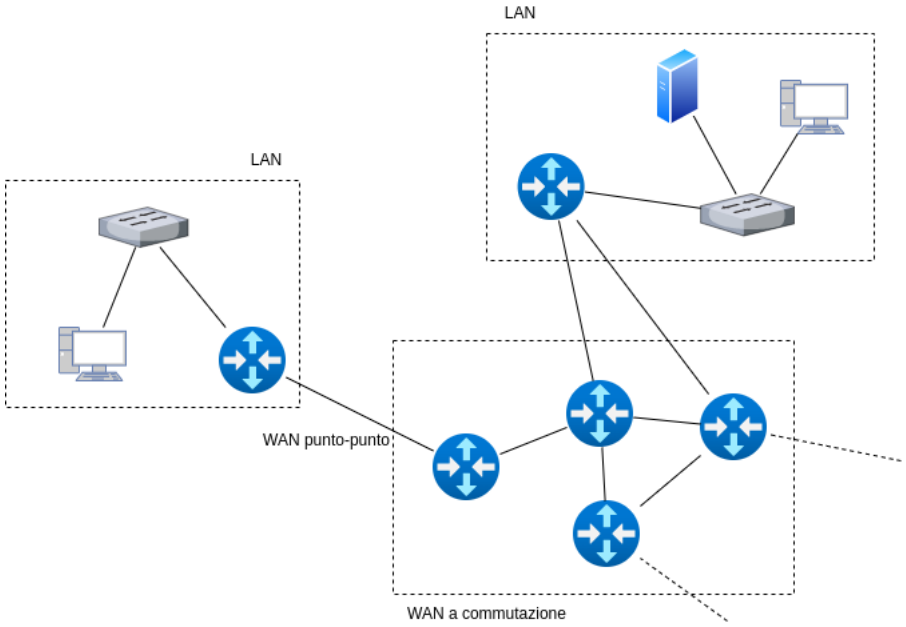
- **LAN a cavo condiviso**



- **LAN con switch**



- **Rete composta**



I supporti fisici utilizzabili per una trasmissione si differenziano in:

- **Doppino intrecciato** (ad esempio un cavo Ethernet), composto da due fili di rame isolati, uno utilizzato per inviare i dati ed uno per riceverli
- **Cavo coassiale**, composto da due conduttori di rame concentrici, entrambi bidirezionali, avente una larghezza di banda maggiore
- **Cavo in fibra ottica**, composto da una fibra di vetro che trasporta impulsi luminosi (dunque alla velocità della luce) al suo interno, ognuno rappresentante un singolo bit
- **Trasmissione wireless**, realizzata tramite l'invio di un segnale radio propagato nell'aria (es: rete cellulare, satellitare o wi-fi)

1.2 Struttura di Internet

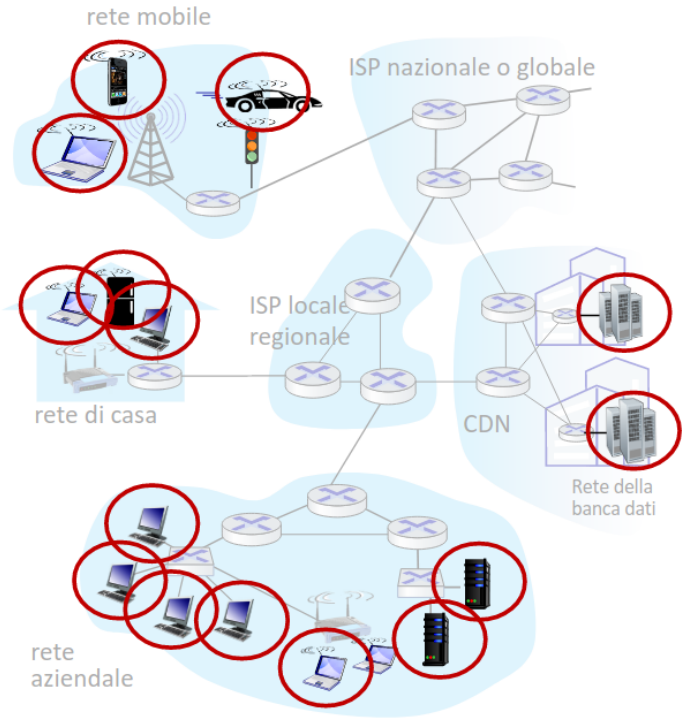
Definition 3. Rete internet

Definiamo come **internet** (abbreviativo di internetwork) una **rete di reti**, ossia una rete che mette in comunicazione due o più reti tra di loro.

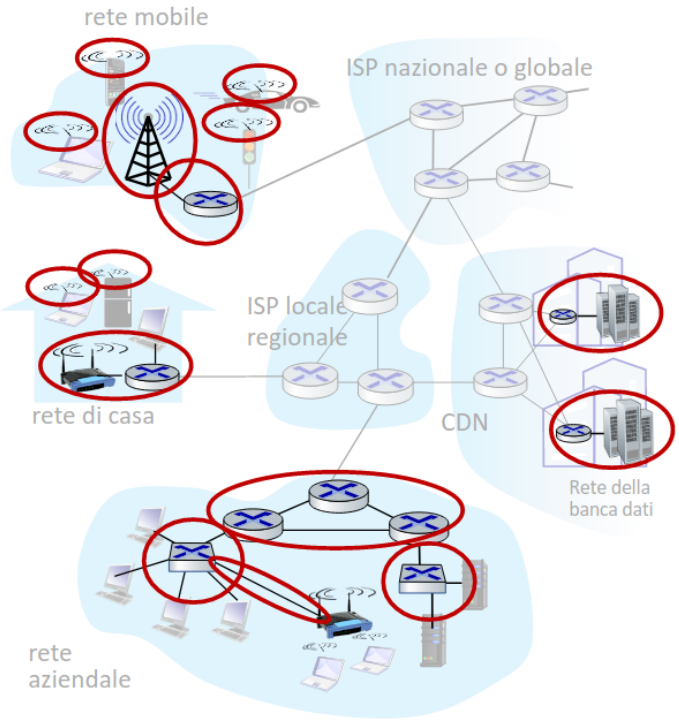
Attenzione: nonostante ciò che viene comunemente chiamato l'**Internet** sia una internet, è necessario puntalizzare che con tale termine comune viene indicata **la rete di tutte le reti**.

Al suo interno, la struttura di Internet risulta esssere composta da:

- **Periferia della rete (network edge)**, corrispondente all'insieme di tutti i sistemi terminali connessi.



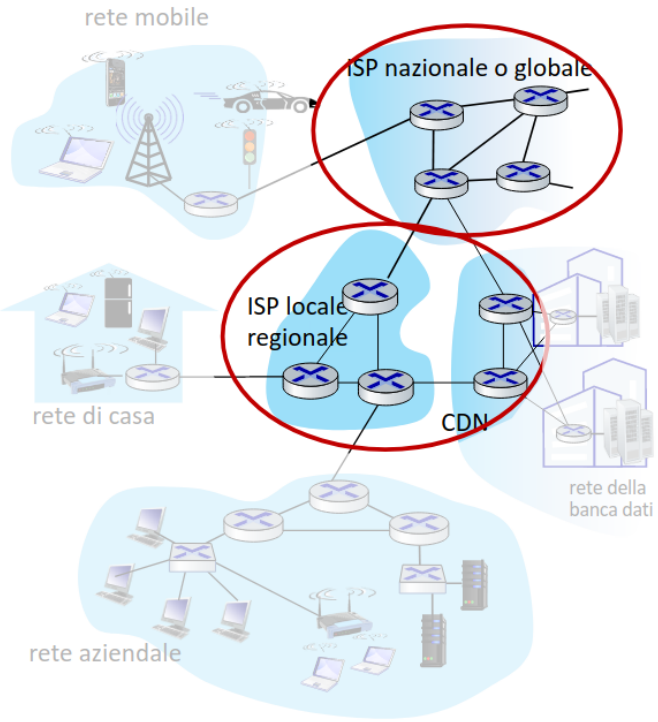
- **Reti di accesso (access network)**, corrispondente ai collegamenti fisici che connettono un sistema terminale al primo **edge router**, ossia il primo router presente nel percorso dal sistema terminale di origine ad un qualsiasi altro sistema terminale di destinazione.



In particolare, l'accesso all'Internet può essere effettuato in più modi:

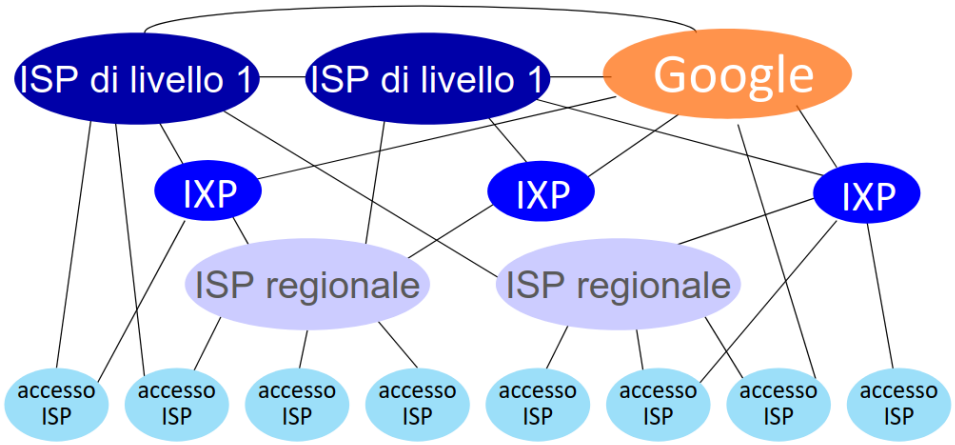
- **Accesso via cavo**, tramite supporti fisici connessi direttamente ad una rete di distribuzione, detta **cable headend** (es: il centralino di un ISP).
- **Accesso via Digital Subscriber Line (DSL)**, dove viene utilizzata la linea telefonica esistente per collegarsi alla rete dell'ISP
- **Accesso via Wireless LAN (WLAN)**, tramite un collegamento wireless ad una stazione base detta **access point** connessa con il router, a sua volta connesso con un cable headend
- **Accesso via rete cellulare**, dove viene utilizzata la rete cellulare esistente per collegarsi alla rete dell'ISP
- **Accesso via rete aziendale**, tramite una rete aziendale (o universitaria, privata, ...) direttamente connessa ad Internet

- **Nucleo di rete (core o backbone)**, ossia un sistema di router interconnessi tra di loro, corrispondente all'insieme di nodi tramite cui viene realizzata la vera interconnessione tra tutte le reti.



In particolare, all'interno del backbone di Internet sono presenti **più livelli di reti ISP** (es: regionali, nazionali, aziendali, ...), le quali devono essere interconnesse tra di loro tramite degli **Internet Exchange Point (IXP)**.

Inoltre, nel recente periodo, nel backbone di Internet sono state integrate anche delle grandi reti private aziendali, ossia le **reti dei content provider** (es: Google, Netflix, ...), le quali, ormai, funzionano come vere e proprie ISP.



1.3 Pacchetti, Commutazione e Instradamento

Definition 4. Pacchetto e Velocità di trasmissione

Dato un messaggio m da trasferire tra due terminali, definiamo come **pacchetti** l'insieme di blocchi di L bit tali che $m = \{p_1, \ldots, p_k\}$.

Ogni pacchetto viene trasmesso nella rete ad una **velocità di trasmissione** R (anche detta larghezza di banda o capacità del collegamento).

Definition 5. Forwarding e Routing

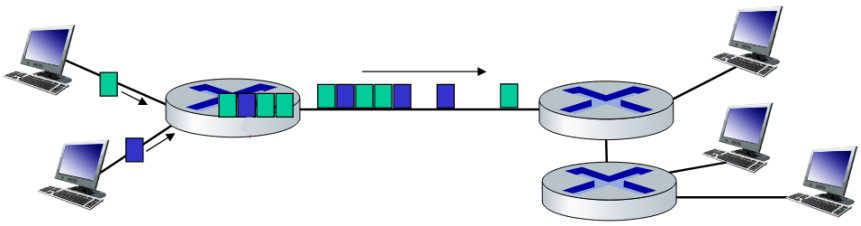
Le funzioni fondamentali di una rete si dividono in:

- **Forwarding o Switching (commutazione)**, ossia un'azione locale tramite cui vengono spostati i pacchetti in arrivo dal collegamento di ingresso del router al collegamento appropiato di uscita. Viene effettuato attraverso una **local forwarding table**, contenente gli indirizzi dei nodi locali
- **Routing (instradamento)**, ossia un'azione globare tramite cui vengono determinati i percorsi origine-destinazione seguiti dai pacchetti. Viene effettuato tramite **algoritmi di instradamento**

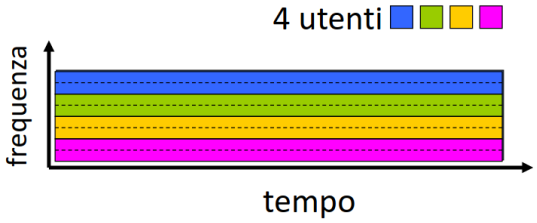
In particolare, la commutazione può avvenire in due modi:

- **Commutazione di pacchetto**:
  - La rete inoltra i pacchetti da un router all'altro attarverso i collegamenti presenti nell'instradamento dall'origine alla destinazione.
  - Una volta inviato, un pacchetto deve completamente raggiungere il nodo a cui sta attualmente venendo inviato prima di poter essere trasmesso al collegamento successivo (**store & forward**)

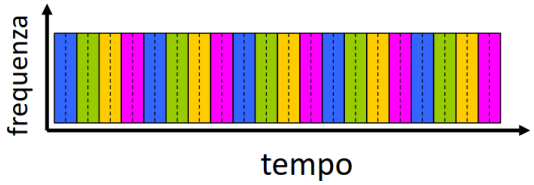
  - Se la velocità di trasmissione sul link di entrata supera la velocità di trasmissione di quello in uscita, i pacchetti verranno messi all'interno di una coda, in attesa di essere trasmessi sul link di uscita
  - Se il buffer della coda raggiunge capienza massima, i pacchetti verranno scartati (**perdita di pacchetti**), per poi, se necessario, essere rinviati



- **Commutazione di circuito**:
  - La banda dei mezzi di trasmissione viene **suddivisa** in parti, riservando ognuna di essere ad una comunicazione tra un'origine ed una destinazione.
  - Per via di tale suddivisione, il numero di utenti massimo della rete risulta essere **limitato dal numero di suddivisioni**
  - La suddivisione può essere effettuata in due modalità:
    * **Frequency Division Multiplexing (FDM)**, dove le frequenze del mezzo di trasmissione vengono suddivise in bande di freqeunza, ognuna di esse riservata ad una singola comunicazione, la quale può utilizzare al massimo la banda ad essa riservata



    * **Time Division Multiplexing (TDM)**, dove il tempo viene suddiviso in slot, ognuno di essi riservato ad una singola comunicazione, la quale può utilizzare l'intera banda del mezzo per il breve lasso di tempo dedicato.



Nonostante la **commutazione di pacchetto** permetta l'accesso di un numero maggiore di utenti e non necessiti di stabilitre una configurazione del collegamento, la presenza di una possibile perdita di pacchetti rende tale tipo di commutazione prettamente ottimo per **trasmissioni "bursty"**, ossia intermittenti e con lunghi periodi di inattività.

1.4 Misura delle prestazioni

Definition 6. Larghezza di banda e Transmission rate

Con il termine **larghezza di banda (bandwidth)** indichiamo due concetti strettamente legati tra loro:

- La quantità (espressa in Hz) rappresentante la **larghezza dell'intervallo di frequenze** utilizzato dal sistema trasmissivo, ossia l'intervallo di frequenze utilizzato dal sistema trasmissivo. Maggiore è tale quantità, maggiore è la quantità di informazioni veicolabili tramite il mezzo di trasmissione.
- La quantità (espressa in b/s) detta anche **transmission rate (o bit rate)** rappresentante la **quantità di bit al secondo** che un link **garantisce di trasmettere**. Tale quantità è proporzionale alla larghezza di banda (in Hz)

Definition 7. Throughput

Con il termine **throughput** indichiamo la **quantità di bit** al secondo che **passano attraverso un nodo** della rete.

Observation 1

A differenza del **transmission rate**, il quale fornisce una misura della **potenziale velocità di un link**, il **throughput** fornisce una misura dell'**effettiva velocità di un link**.

In generale, dunque, si ha che

$$T < R$$

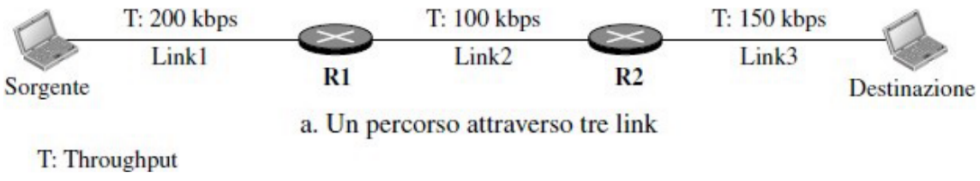
dove T è il throughput e R è il transmission rate

Definition 8. Collo di bottiglia

Dato un percorso end-to-end, ossia tra un dispositivo e un altro, definiamo come **collo di bottiglia** il link limitante il throughput dei link presenti su tale percorso

Esempio:

- Consideriamo il seguente percorso



a. Un percorso attraverso tre link

T: Throughput

- Il link $L2$ risulta essere il collo di bottiglia di tale percorso, limitando il throughput del percorso a 100 kb/s

Definition 9. Delay di trasmissione

Definiamo come **delay (o latenza) di trasmissione** il tempo necessario ad un nodo per immettere un pacchetto su un link, corrispondente a:

$$D_t = \frac{L}{R}$$

dove L è la dimensione del pacchetto e R è il transmission rate del link

Definition 10. Delay di propagazione

Definiamo come **delay (o latenza) di propagazione** il tempo impiegato dall'**ultimo bit di blocco di dati** posto su un link ad essere propagato fino al nodo di destinazione, corrispondente a:

$$D_p = \frac{k}{v}$$

dove k è la lunghezza del link e v è la velocità di propagazione del link

Definition 11. Delay di un pacchetto

Definiamo come **delay (o latenza) di un pacchetto** il tempo totale necessario ad un pacchetto per essere inviato completamente da un nodo origine ad un nodo destinatario

$$D_n = D_e + D_q + D_t + D_p$$

dove:

- D_e è il **delay di elaborazione del nodo**, dipendente dalle operazioni di controllo svolte dal nodo
- D_q è il **delay di queueing**, ossia l'attesa del pacchetto all'interno della coda del nodo prima di essere trasmesso, dipendente dalla quantità di pacchetti presenti nella coda
- D_t è il delay di trasmissione del link
- D_p è il delay di propagazione del link

Proposition 1. Prodotto rate per delay di propagazione

Dato un link con transmission rate R e delay di propagazione D_p, il prodotto

$$B_{max} = R \cdot D_p = \frac{L \cdot k}{D_t \cdot v}$$

rappresenta il **massimo numero di bit distribuiti tutto sul cavo** contemporaneamente

Esempi:

1. Si consideri un router A che trasmette pacchetti, ognuno di lunghezza $L = 4000$ bit, su un canale di trasmissione con rate $R = 10$ Mb/s verso un router B all'altro estremo del link. Si supponga che il delay di propagazione sia pari a 0.2 ms.

   - Quanto impiega il router A a trasmettere un pacchetto al router B?

   $$D_t = \frac{L}{R} = \frac{4 \cdot 10^3\ b}{10^7\ b/s} = 4 \cdot 10^{-4}\ s = 0.4\ ms$$

   - Quanto impiega il router A a trasmettere un bit al router B?

   $$D_{1b} = \frac{1}{R} = \frac{1\ b}{10^7\ b/s} = 10^{-7}\ s = 0.1\ \mu s$$

   - Qual è il massimo numero di pacchetti al secondo che possono essere trasmessi sul link?

   $$1\ P = 4000\ b \implies 1\ b = \frac{1}{4000}\ P \implies$$
   $$\implies R = 10^7\ b/s = \frac{10^7}{4000}\ P/s = \frac{1}{4} \cdot 10^3\ P/s = 2500\ P/s$$

   - Supponendo che il router A invii i pacchetti uno dopo l'altro senza introdurre ritardi tra la trasmissione di un pacchetto e il successivo, quanto tempo impiega il router B a ricevere 4 pacchetti?

     Poiché i pacchetti vengono inviati senza alcun delay tra di essi, possiamo considerare tali pacchetti come un unico grande pacchetto di dimensione $4 \cdot L$, implicando che

     $$D_{4t} = \frac{4 \cdot L}{R} = \frac{16 \cdot 10^3\ b}{10^7\ b/s} = 16 \cdot 10^{-4}\ s = 1.6\ ms$$

     Inoltre, per lo stesso motivo, il tempo di propagazione rimarrà inalterato, poiché esso non dipende dalla lunghezza del pacchetto, ma solo dalla lunghezza e della velocità di propagazione del link. Di conseguenza, il tempo totale impiegato sarà $1.6\ ms + 0.2\ ms = 1.8\ ms$

   - Qual è il massimo numero di bit e il numero di pacchetti che possono essere presenti sul canale?

   $$P_{max} = R \cdot D_p = 10^7\ b/s \cdot 0.2\ ms = 2000\ b = \frac{1}{2}\ P$$

2. Si consideri un host A che vuole inviare un file molto grande, 4 milioni di byte, a un host B. Il percorso tra A e B ha 3 link L_1, L_2, L_3, ognuno di lunghezza 300 km, ciascuno con rate rispettivo $R_1 = 500$ kb/s, $R_2 = 2$ Mb/s e $R_3 = 1$ Mb/s.

   - Assumendo l'assenza di ulteriore traffico nella rete, qual è il throughput per il file transfer?

     Poiché il link L_1 risulta essere il collo di bottiglia del percorso, il throughput risulta essere $R_1 = 500$ kb/s

- Qual è il tempo totale impiegato per trasferire il file all'host B assumento che i link siano cavi in fibra ottica?

  Poiché non vi è specificata la lunghezza di ogni pacchetto, assumiamo che il file venga inviato come un unico grande pacchetto, implicando che $L = 4 \cdot 10^6\ B = 32 \cdot 10^6\ b$.

  Di conseguenza, si ha che:

$$D_t(L_1) = \frac{32 \cdot 10^6\ b}{5 \cdot 10^5\ b/s} = 64\ s$$

$$D_t(L_2) = \frac{32 \cdot 10^6\ b}{2 \cdot 10^6\ b/s} = 16\ s$$

$$D_t(L_2) = \frac{32 \cdot 10^6\ b}{1 \cdot 10^6\ b/s} = 32\ s$$

  Poiché L_1, L_2, L_3 sono cavi in fibra ottica, la velocità di propagazione su di essi corrisponde alla velocità della luce, pari a $\sim 3 \cdot 10^8\ m/s$. Dunque, il delay di propagazione di ogni link corrisponderà a:

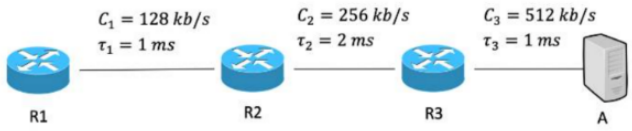
$$D_p = \frac{3 \cdot 10^5\ m}{3 \cdot 10^8\ m/s} = 1\ ms$$

  Infine, concludiamo che il tempo totale impiegato corrisponda a:

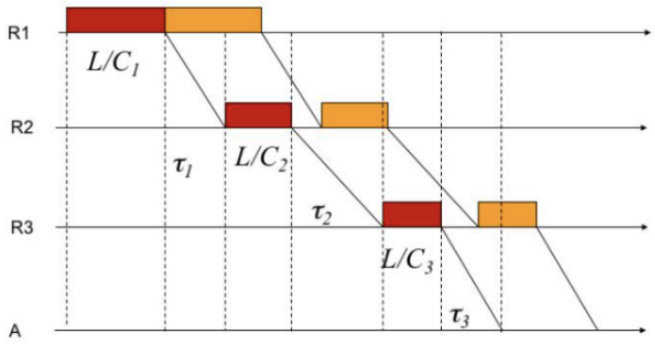
$$D_{tot} = D_t(L_1) + D_t(L_2) + D_t(L_3) + 3 \cdot D_p = 64\ s + 16\ s + 32\ s + 3 \cdot 1\ ms = 112,003\ s$$

3. Si consideri la rete nella seguente figura, dove C_1, C_2, C_3 e τ_1, τ_2, τ_3 sono rispettivamente i transmission rate e i delay di propagazione dei tre link.

   Al tempo $t = 0$, la coda di uscita di R_1 contiene 2 pacchetti diretti ad A. Assumento che la lunghezza dei pacchetti sia $L = 512$ b, si indichi per ciascun pacchetto l'istante in cui esso viene completamente ricevuto da A.



   Per aiutarci durante il calcolo, grafichiamo il contenuto di ogni coda al passare del tempo:



Dunque, il tempo totale impiegato dal primo pacchetto corrisponderà a:

$$T_1 = \frac{L}{C_1} + \tau_1 + \frac{L}{C_2} + \tau_2 + \frac{L}{C_3} + \tau_3 = 4\ ms + 1\ ms + 2\ ms + 2\ ms + 1\ ms + 1\ ms = 11\ ms$$

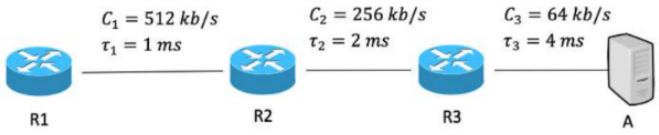
Analogamente, il tempo totale impiegato dal secondo pacchetto corrisponderà a:

$$T_2 = 2 \cdot \frac{L}{C_1} + \tau_1 + \frac{L}{C_2} + \tau_2 + \frac{L}{C_3} + \tau_3 = 8\ ms + 1\ ms + 2\ ms + 2\ ms + 1\ ms + 1\ ms = 15\ ms$$

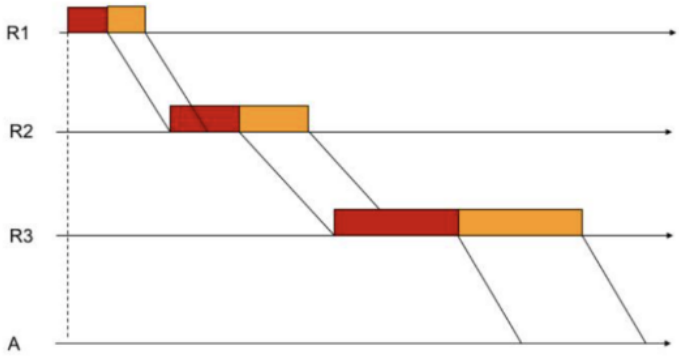
4. Si consideri la rete nella seguente figura, dove C_1, C_2, C_3 e τ_1, τ_2, τ_3 sono rispettivamente i transmission rate e i delay di propagazione dei tre link.

   Al tempo $t = 0$, la coda di uscita di R_1 contiene 2 pacchetti diretti ad A. Assumendo che la lunghezza dei pacchetti sia $L = 512$ b, si indichi per ciascun pacchetto l'istante in cui esso viene completamente ricevuto da A.

   Inoltre, supponendo che vi siano n pacchetti, si indichi una formula generica descrivente per ciascun pacchetto l'istante in cui esso viene completamente ricevuto da A



   Come nel caso precedente, grafichiamo il contenuto di ogni coda al passare del tempo:



   Il tempo totale impiegato dal primo pacchetto corrisponderà a:

$$T_1 = \frac{L}{C_1} + \tau_1 + \frac{L}{C_2} + \tau_2 + \frac{L}{C_3} + \tau_3 = 1\ ms + 1\ ms + 2\ ms + 2\ ms + 8\ ms + 4\ ms = 18\ ms$$

   Notiamo come, a differenza del caso precedente, il secondo pacchetto giunge nelle code successive mentre il primo pacchetto deve essere ancora completamente spedito, implicando che esso debba essere inserito nella coda di attesa.

   Dunque, una volta raggiunta la coda finale, il secondo pacchetto potrà essere inviato solo una volta completato il primo pacchetto.

Di conseguenza, il suo tempo totale di ricezione corrisponde a:

$$T_2 = T_1 + \frac{L}{C_3} = 18\ ms + 8\ ms = 26\ ms$$

Applicando lo stesso ragionamento nel caso di n pacchetti, la formula generica descrivente l'istante di ricezione dell'n-esimo pacchetto corrisponde a:

$$T_n = T_{n-1} + \frac{L}{C_3} = T_{n-2} + 2 \cdot \frac{L}{C_3} = \ldots = T_1 + (n-1)\frac{L}{C_3} = 18\ ms + 8(n-1)\ ms$$

1.5 Stack protocollare TCP/IP

Definition 12. Protocollo

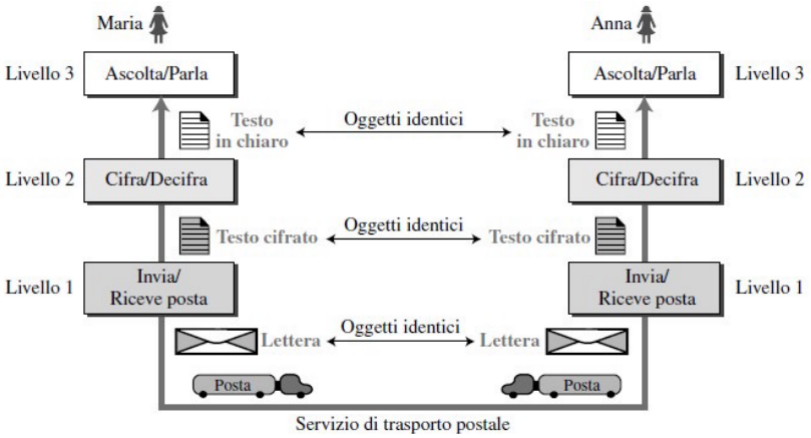
Un **protocollo** definisce l'insieme di **regole** che il dispositivo mittente e il dispositivo destinatario, così come tutti i sistemi intermedi coinvolti, devono rispettare per essere in grado di comunicare.

In situazioni più complesse, potrebbe essere opportuno suddividere i compiti necessari alla comunicazione fra **più livelli (layer)**, nel qual caso è richiesto **un protocollo per ciascun livello**. Tramite un layering dei protocolli, dunque, è possibile suddividere un compito complesso in compiti più semplici, ognuno gestibile da un singolo protocollo.

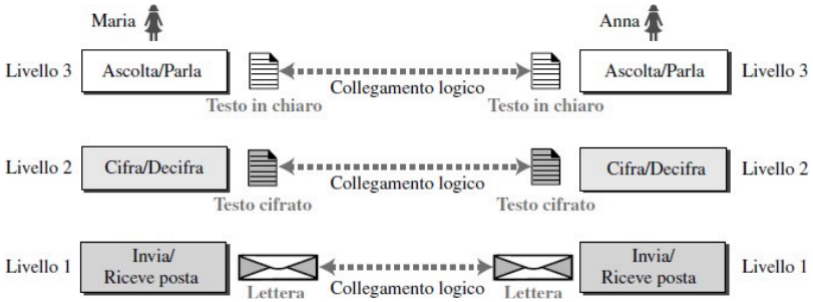
In particolare, ogni layer è **indipendente dagli altri** (modularizzazione), utilizzando i servizi forniti dal layer inferiore e offrendo servizi al layer superiore.

Ogni layer, dunque, può essere considerato come una **black box** con opportuni ingressi ed uscite, senza necessità di essere a conoscenza delle modalità con cui i dati in ingresso vengano trasformati in quelli di uscita.

Quando è richiesta una **comunicazione bidirezionale**, ciascun layer deve essere in grado di effettuare entrambi i compiti richiesti, ossia manipolare i dati in input per inviarli al livello superiore o manipolarli per inviarli al livello inferiore.



In particolare, l'effetto ottenuto tramite una suddivisione in uno stack di layer equivalenti permette l'istaurazione di un **collegamento logico** tra ogni livello dello stack: il protocollo implementato in ciascun livello specifica una comunicazione diretta tra i pari livelli delle due parti: il layer N di un dispositivo comunica solo ed esclusivamente con il layer N di tutti i dispositivi.



Inoltre, per via dell'estrema modularizzazione ottenuta, viene facilitata la manutenzione e l'aggiornamento del sistema, poiché il cambiando dell'implementazione del servizio di un layer rimane trasparente al resto del sistema.

Definition 13. Stack protocollare TCP/IP

La principale forma di stack protocollare utilizzata corrisponde allo **stack protocollare TCP/IP**, la cui struttura a layer corrisponde a:

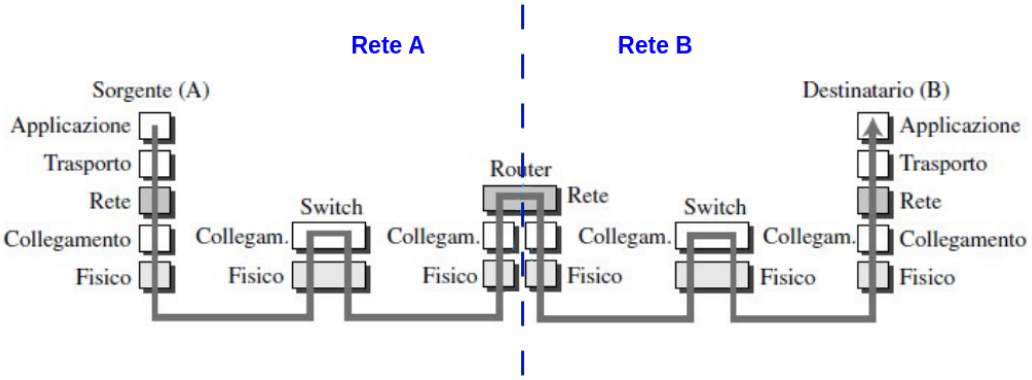
- **Livello di Applicazione**, il quale fornisce supporto alle applicazioni facente uso della rete (protocolli HTTP, SMTP, FTP, DNS, ...).
- **Livello di Trasporto**, il quale gestisce il trasferimento dei pacchetti dal processo del dispositivo mittente a quello del dispositivo destinatario (protocolli TCP, UDP, ...).
- **Livello di Rete**, il quale gestisce l'instradamento dei pacchetti dall'origine alla destinazione (protocolli IP, ...).
- **Livello di Collegamento (o Link)**, il quale gestisce la trasmissione dei pacchetti da un nodo a quello successivo sul percorso (protocolli Ethernet, Wi-fi, PPP, ...). Lungo il percorso, un pacchetto può essere gestito da protocolli discersi.
- **Livello Fisico**, dove avviene il vero e proprio trasferimento dei singoli bit

I livelli di Applicazione e di Trasporto sono gestiti tramite **software**, mentre i livelli di Collegamento e Fisico tramite **hardware**.

Durante l'invio di un pacchetto, quest'ultimo, partendo dal livello applicazione del dispositivo sorgente, **percorre tutti i layer dello stack protocollare**, fino a giungere al livello fisico, dove viene effettivamente inviato al nodo successivo.

Tutti i nodi intermedi presenti sul percorso lavoreranno utilizzando solo i livelli necessari. In particolare, ogni dispositivo utilizzerà il livello di collegamento, in modo da poter spedire il pacchetto stesso verso il nodo successivo del percorso.

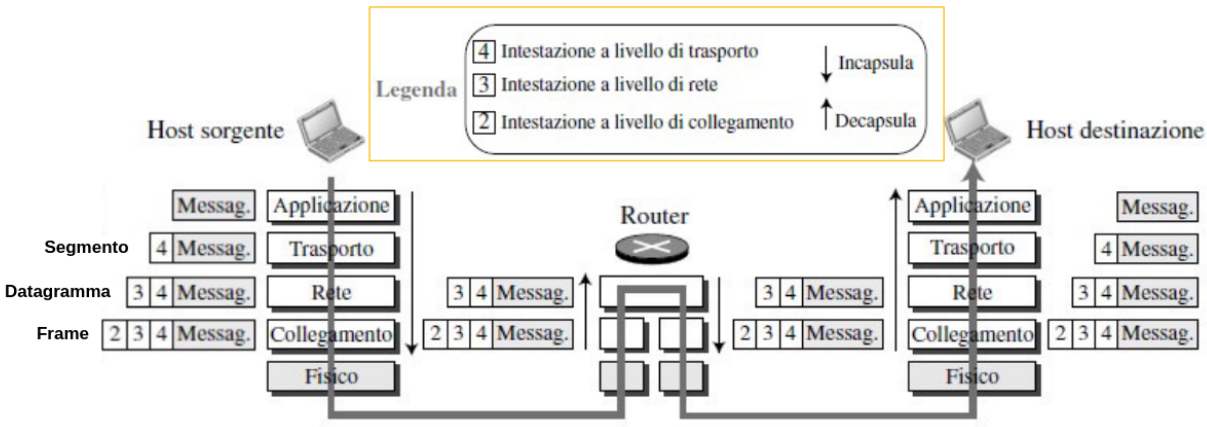
Nel caso in cui si raggiunga il **punto di scambio tra due reti**, solitamente un edge router, verrà utilizzato anche il livello di rete.



Prima di essere spedito al livello inferiore, ogni pacchetto viene **incapsulato**: una volta ricevuto il pacchetto dal layer superiore, il layer attuale applica un proprio **header (o intestazione)**, aggiungendo informazioni necessarie al layer del dispositivo di destinazione corrispondente a quello attuale.

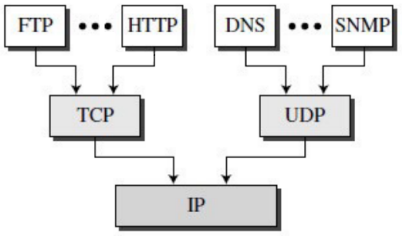
In particolare, ad ogni livello un pacchetto assume il nome di:

- **Messaggio (al livello di applicazione)**, corrispondente al pacchetto originale, senza alcuna intestazione
- **Segmento (al livello di trasporto)**, corrispondente al messaggio ricevuto dal layer superiore a cui viene aggiunto un header di trasporto
- **Datagramma (al livello di rete)**, corrispondente al segmento ricevuto dal layer superiore a cui viene aggiunto un header di rete
- **Frame (al livello di collegamento)**, corrispondente al datagramma ricevuto dal layer superiore a cui viene aggiunto un header di collegamento

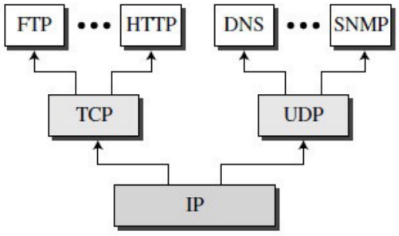


Poiché lo stack protocollare TCP/IP prevede la presenza di **più protocolli nello stesso livello**, ogni livello deve essere in grado di effettuare operazioni di:

- **Multiplexing**, dove ogni protocollo deve essere in grado di incapsulare (uno alla volta) i pacchetti ricevuti da più protocolli presenti al livello superiore



- **Demultiplexing**, dove ogni protocollo deve essere in grado di decapsulare i pacchetti ricevuti ed inviarli a più protocolli presenti nel livello superiore



Per realizzare ciò, nell'header di ogni layer viene inserito un **campo speciale** in grado di identificare quale sia il protocollo di appartenenza di tale pacchetto.

Capitolo 2

Livello di Applicazione

2.1 Principi delle applicazioni di rete

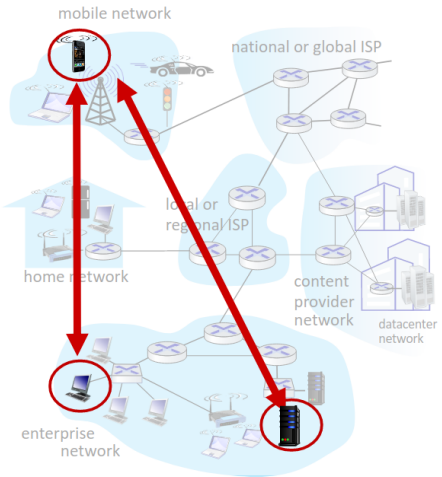
Definition 14. Paradigma di comunicazione

Un **paradigma di comunicazione** è una metodologia di scambio informazioni e gestione delle connessioni all'interno di una rete, principalmente all'interno di Internet.

In particolare, i due principali paradigmi utilizzati sono:

- **Paradigma Client-Server**, dove i sistemi terminali vegnono divisi in due categorie:
  - **Client**, il quale comunica solo ed esclusivamente con un server, **richiedendo dei servizi** a quest'ultimo, e può rimanere anche inattivo se non necessario, implicando che esso possa avere indirizzi IP dinamici nel tempo. In particolare, per tali caratteristiche, non vi è una comunicazione client-client, ma solo una comunicazione client-server-client
  - **Server**, il quale possiede un indirizzo IP permanente, rimanendo sempre attivo in attesa di **fornire servizi** ai vari client richiedenti

  Ad esempio, i protocolli HTTP, FTP e IMAP sono basati su tale paradigma

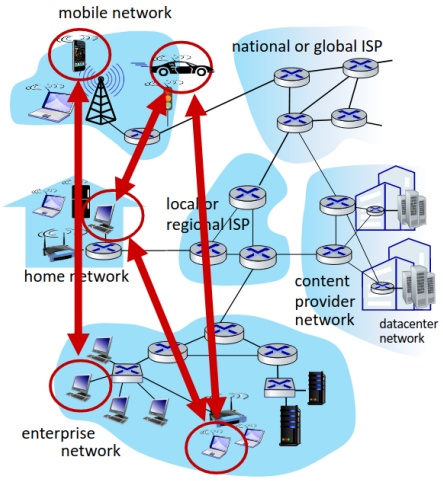


- **Paradigma Peer-to-Peer (P2P)**, dove i sistemi terminali vengono detti **peer** (tradotto: *pari, di egual importanza*) ed ognuno di essi è in grado di comunicare direttamente con ogni altro peer (assumendo quindi sia il compito di client che di server).

  In particolare, ogni peer **richiede e fornisce servizi ad altri peer**, rendendo il sistema **estremamente scalabile**: ogni nuovo peer incrementa le capacità di servizio e le richieste di servizio.

  Inoltre, come nel caso dei client, ogni peer può connettersi alla rete in modo intermittente utilizzando IP dinamici, diminuendo temporaneamente la quantità di servizi fornibili nella rete. Di conseguenza, la loro gestione risulta estremamente più complessa, ma anche più performante nel caso di un numero elevato di peer.

  Un classico esempio di utilizzo del paradigma P2P risultanno essere i vari protocolli legati al torrenting e alla condivisione di file di grandi dimensioni.



Definition 15. Processo

Un **processo** è un programma in esecuzione all'interno di un sistema terminale.

In particolare, un **processo client** è un processo che avvia una comunicazione, mentre un **processo server** è un processo che attente di essere contattato da un processo client.

All'interno dello stesso sistema, due processi comunicano tra di loro utilizzando una comunicazione **inter-process**, definita dal sistema operativo. I processi situati su sistemi diversi, invece, comunicano tra di loro tramite **scambio di messaggi**

Definition 16. Socket

Un **socket** è un'**astrazione software** tramite cui un processo può inviare e ricevere messaggi tramite il socket di un altro processo. Per poter comunicare, dunque, due processi devono connettersi tramite due socket (uno ciascuno), identificati da una coppia `<Indirizzo_IP, Numero_Porta>`

Ogni protocollo a livello di applicazione definisce:

- Le tipologie di messaggi scambiati (es: richiesta e risposta)
- La sintassi del messaggio
- La semantica del messaggio
- Le regole per come e quando i processi inviano e rispondono ai messaggi

In particolare, i protocolli a tale livello si differenziano in **protocolli aperti**, ossia definiti secondo uno standard pubblico ed adottato comunemente da ogni applicazione (es: HTTP, FTP, ...), e **protocolli proprietari**, ossia non pubblici e fini all'applicazione stessa (es: Skype, ...).

Per poter funzionare correttamente, ogni applicazione di rete necessita di alcuni **servizi di trasporto**. In particolare, esse possono necessitare di:

- **Integrità dei dati**, ossia un trasferimento dei dati affidabile al 100%, senza alcuna perdita di pacchetto o corruzione dei dati
- **Garanzie temporali**, ossia un basso ritardo per la ricezione dei dati
- **Garanzie di throughput**, ossia una quantità minima di throughput dati
- **Sicurezza**, ad esempio crittografia o integrità dei dati a seguito di manomissioni

Definition 17. Transmission Control Protocol (TCP)

Il **Transmission Control Protocol (TCP)** è un protocollo risiedente sul **layer di trasporto** in grado di fornire **trasporto affidabile**, ossia senza perdita di alcun pacchetto, e controllo del flusso e della congestione, in cambio di un'assenza di garanzie temporali, di throughput e di sicurezza.

Inoltre, il protocollo TCP è **orientato alla connessione**, ossia richiedente una configurazione **(handshaking)** tra il processo client e il processo server

Definition 18. User Datagram Protocol (UDP)

L'**User Datagram Protocol (UDP)** è un protocollo risiedente sul **layer di trasporto** in grado di fornire **trasporto veloce** poiché **non orientato alla connessione** ed **estremamente scarno**, ossia sprovvisto di: trasporto affidabile, controllo del flusso e della congestione e garanzie temporali, di throughput e di sicurezza

Poiché per loro natura i protocolli TCP ed UDP sono privi di garanzie di sicurezza, i messaggi scambiati tra socket TCP e UDP risultano sprovvisti di crittografia, attraversando il percorso instradato completamente in chiaro ed essendo quindi leggibili e manipolabili da qualsiasi dispositivo intermedio.

Per ovviare tale problema, viene implementato a livello di applicazione il protocollo **Transport Security Layer (TLS)** tramite socket realizzati con librerie software specifiche, fornendo connessioni crittografate, integrità dei dati ed autenticaazione dell'endpoint.

2.2 Web e Protocollo HTTP

Una pagina web è composta da **oggetti**, ognuno dei quali può essere archiviato su un diverso web server. In particolare, una pagina web consiste in un **file HTML** il quale include diversi oggetti referenziati tramite vari URL

$$\underbrace{\text{www.someschool.edu}}_{\text{host name}}\,/\,\underbrace{\text{someDept/image.gif}}_{\text{path name}}$$

Definition 19. Protocollo HTTP

Il **protocollo HTTP (Hypertext Transfer Protocol)** è un protocollo a livello di applicazione utilizzato per la realizzazione di servizi web. La sua porta di riferimento comune all'interno dei socket è la **porta 80**.

Il protocollo HTTP è **stateless**, ossia non conservante alcuna informazione sulle richieste passate, e basato sul **paradigma client-server**, dove il client invia messaggi detti **richieste** e il server invia messaggi detti **risposte**.

Inoltre, il protocollo HTTP fa uso del **protocollo TCP**:

1. Il client avvia una connessione TCP con il server utilizzando la porta 80, rimanendo in attesa che il server accetti la connessione (TCP handshaking)
2. Vengono scambiati messaggi HTTP tra client e server
3. La connessione TCP viene chiusa

Le **connessioni HTTP** si differenziano in due tipologie:

- **Connessione non persistente**, dove viene aperta la connessione TCP e viene inviato massimo un oggetto prima di chiudere la connessione TCP
- **Connessione persistente (HTTP 1.1)**, dove viene aperta la connessione TCP e vengono inviati multipli oggetti in successione prima di chiudere la connessione TCP

Esempio:

1. Supponiamo che un utente inserisca l'URL dell'oggetto "www.someSchool.edu/ someDepartment/home.index", contenente del testo e 10 riferimenti ad immagini.
2. Il client HTTP dell'utente (browser, cURL, ...) avvia la connessione TCP con il server HTTP tramite la porta 80
3. Il server HTTP sull'host "www.someSchool.edu" riceve la richiesta di connessione, accettandola e notificando il client
4. Il client HTTP invia un messaggio di richiesta HTTP, contenente il path dell'oggetto desiderato, ossia "/someDept/index.html"
5. Il server HTTP riceve il messaggio di richiesta e invia il messaggio di risposta contenente l'oggetto desiderato, il quale a sua volta contiene i riferimenti alle 10 immagini.

6. A questo punto, si creano due scenari:
   - Se la connessione non è persistente, il server chiude immediatamente la connessione TCP, implicando che l'intero processo debba essere ripetuto per tutti e 10 i riferimenti necessari
   - Se la connessione è persistente, il client invierà in successione altre 10 richieste al server, richiedendo quindi solo la ripetizione dei passaggi 4 e 5 (per 10 volte), per poi chiudere la connessione TCP

Definition 20. Round Trip Time (RTT)

Definiamo come **Round Trip Time (RTT)** il tempo impiegato da un pacchetto di piccole dimensioni per compiere il percorso clinet-server-client

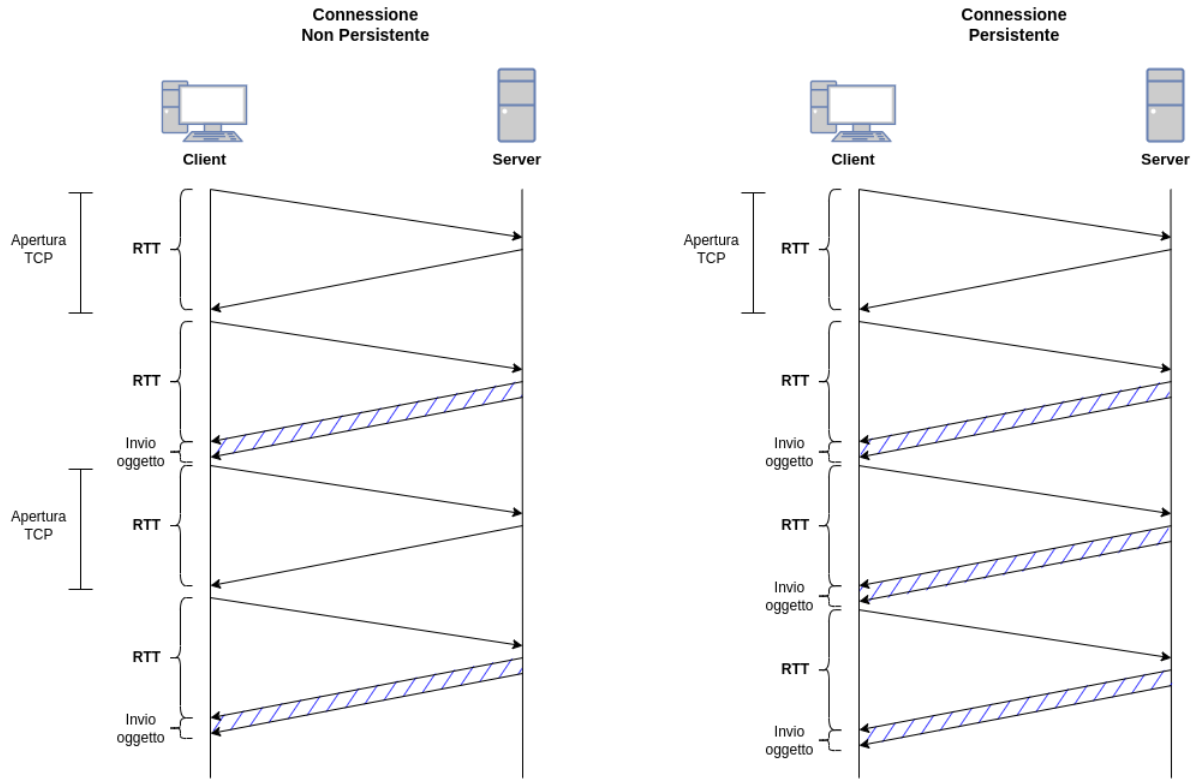
Observation 2. Tempo di risposta HTTP

Se la **connessione non è persistente**, per ogni oggetto sono necessari due RTT, uno per avviare la connessione TCP ed uno per l'invio di richiesta e risposta, seguiti dal tempo necessario ad inviare l'oggetto

$$T_{tot} = (2 \text{ RTT} + \text{ Tempo invio ogg.}) \cdot \text{Num. Oggetti}$$

Se la **connessione è persistente**, invece, saranno necessari un RTT per poter stabilire la connessione TCP, seguiti da un solo RTT per oggetto (con annesso tempo di invio)

$$T_{tot} = 1 \text{ RTT} + (1 \text{ RTT} + \text{ Tempo invio ogg.}) \cdot \text{Num. Oggetti}$$



2.2.1 Messaggi di richiesta e risposta

I messaggi HTTP di **richiesta** e **risposta** vengono formattati un formato leggibile dall'uomo (in particolare, in codice ASCII).

Ogni messaggio di **richiesta HTTP** viene strutturato nel seguente modo:

- Una **riga di richiesta**, composta dal **metodo** utilizzato, il path richiesto e la versione di HTTP utilizzata, seguiti da un carattere di ritorno a capo, ossia `\r`, ed un carattere di avanzamento di riga, ossia `\n`

  I **metodi** principali inseribili all'interno della riga di richiesta sono:

  - **Metodo GET**, utilizzato per l'invio di dati al server, i quali vengono inseriti all'interno dell'URL a seguito di un carattere '?'

    (es: `www.mysite.com/search?user=myuser`)

  - **Metodo POST**, utilizzato per l'invio di dati al server, i quali vengono aggiunti all'interno del body del messaggio (rimanendo quindi parzialmente offuscati all'utente)

  - **Metodo HEAD**, utilizzato per richiedere solo l'header di risposta che verrebbe restituito dalla destinazione a seguito di una richiesta GET

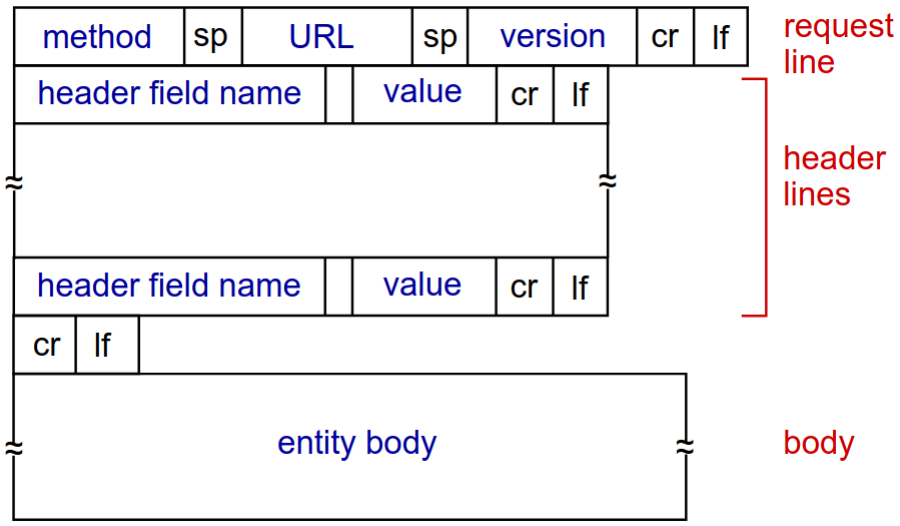
  - **Metodo PUT**, utilizzato per caricare un nuovo file o sostituirne uno esistente all'interno della destinazione (non più utilizzato poiché estremamente insicuro)

- Un **header (o intestazione)**, composto da varie linee contenenti informazioni utili alla connessione

  Alcuni esempi di campi inseribili all'interno di un header di richiesta sono:

Campo Header	Descrizione
`User-agent`	Indica il programma client utilizzato
`Accept`	Indica il formato dei contenuti che il client è in grado di accettare
`Accept-charset`	Famiglia di caratteri che il client è in grado di gestire
`Accept-encoding`	Schema di codifica supportato dal client
`Accept-language`	Linguaggio preferito dal client
`Authorization`	Indica le credenziali possedute dal client
`Host`	Host e numero di porta del client
`Date`	Data e ora del messaggio
`Upgrade`	Specifica il protocollo di comunicazione preferito
`Cookie`	Comunica un cookie al server
`If-Modified-Since`	Invia il documento solo se è più recedente della data specificata

- Un **body (o contenuto)**, ossia il vero contenuto del messaggio da inviare (solitamente vuoto a meno dell'uso del metodo POST)



Esempio:

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Analogamente, ogni messaggio di **risposta HTTP** viene strutturato in modo simile, ma con alcune differenze:

- Una **riga di stato**, composta dalla versione di HTTP utilizzata, un **codice di status** e una **frase di status** descrivente in breve il codice di status

  I **codici di status** si dividono in 5 categorie:

  - **Codici 1xx**, indicanti che la risposta ricevuta contiene solamente informazioni

    (es: `100 Continue` indica che il server è pronto a ricevere la richiesta del client)

  - **Codici 2xx**, indicanti che la richiesta effettuata è andata a buon fine

    (es: `200 OK` indica che la richiesta ha avuto successo e l'oggetto richiesto è stato trovato, `204 No Content` indica che la richiesta ha avuto successo ma l'oggetto richiesto non contiene nulla al suo interno)

  - **Codici 3xx**, indicanti che è stato effettuato un reindirizzamento a seguito della richiesta effettuata

    (es: `301 Moved Permanently` indica che l'oggetto richiesto possiede un path diverso da quello richiesto, rendirizzando automaticamente tutte le richieste successive del client)

  - **Codici 4xx**, indicanti un errore nella richiesta del client

    (es: `403 Forbidden` indica che il client non possiede i requisiti per accedere all'oggetto richiesto, `404 Not Found` indica che l'oggetto richiesto non esiste)

  - **Codici 5xx**, indicanti un errore per cui il server non è riuscito a completare la richiesta

    (es: `500 Internal Server Error` indica un errore sconosciuto all'interno del server, `503 Service Unavailable` indica che il server è attualmente non disponibile)

- Un **header (o intestazione)**, composto da varie linee contenenti informazioni utili alla risposta

  Alcuni esempi di campi inseribili all'interno di un header di risposta sono:

Campo Header	Descrizione
`Date`	Data e ora attuale
`Upgrade`	Specifica il protocollo di comunicazione preferito
`Server`	Indica il programma server utilizzato
`Set-Cookie`	Il server richiede al client di memorizzare un cookie
`Content-Encoding`	Specifica lo schema di codifica
`Content-Language`	Specifica la lingua utilizzata nel documento
`Content-Length`	Indica la lunghezza del documento
`Content-Type`	Specifica la tipologia del documento
`Location`	Chiede al client di inviare la richiesta ad un altro sito
`Last-modified`	Fornisce data e ora dell'ultima modifica del documento

- Un **body (o contenuto)**, ossia il vero contenuto del messaggio da restituire (in particolare, l'oggetto richiesto)

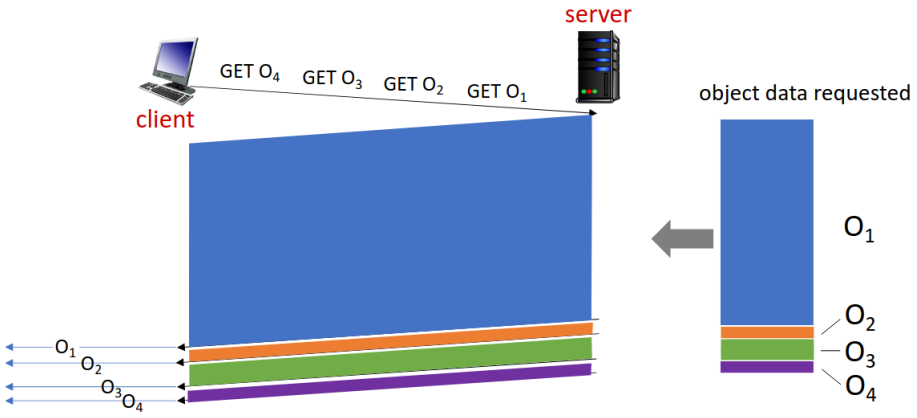
Esempio:

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
[document content...]
[...]
[document content...]
```

2.2.2 Versioni di HTTP

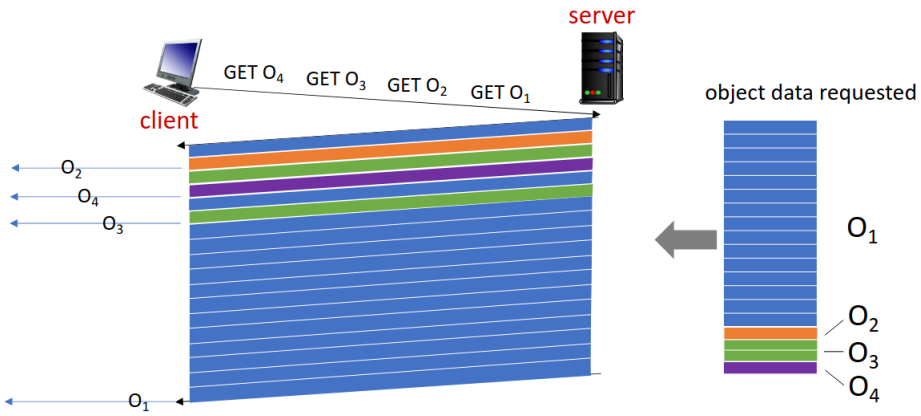
Come già discusso, il **protocollo HTTP 1.1** ha introdotto la possibilità di poter effettuare più richieste GET in successione tramite una singola connessione TCP. Tuttavia, tale modifica ha introdotto ulteriori problematiche:

- Il server risponde alle richieste GET nell'ordine in cui vengono effettuate **(First Come First Served (FCFS))**
- Un oggetto di piccole dimensioni potrebbe dover attendere la trasmissione di oggetti di grandi dimensioni richiesti prima di esso **(blocco head-of-line (HOL))**
- La perdita di un segmento TCP causa lo stallo del trasferimento di un oggetto



Per risolvere tali problematiche, il **protocollo HTTP/2** introduce una maggiore flessibilità al server nell'invio di oggetti al client:

- L'ordine di trasmissione degli oggetti richiesti viene stabilito in base alla priorità dell'oggetto specificata dal client
- Gli oggetti vengono **divisi in frame**, schedulati in modo da mitigare il blocco HOL



Il **protocollo HTTP/3**, invece, risolve le ultime problematiche rimanenti all'interno del protocollo HTTP/2, tramite l'aggiunta di controlli sulla sicurezza, sugli errori e sulla congestione per oggetto, utilizzando il **protocollo UDP** (con alcune migliorie) al posto del protocollo TCP.

2.2.3 Cookies e Web Caching

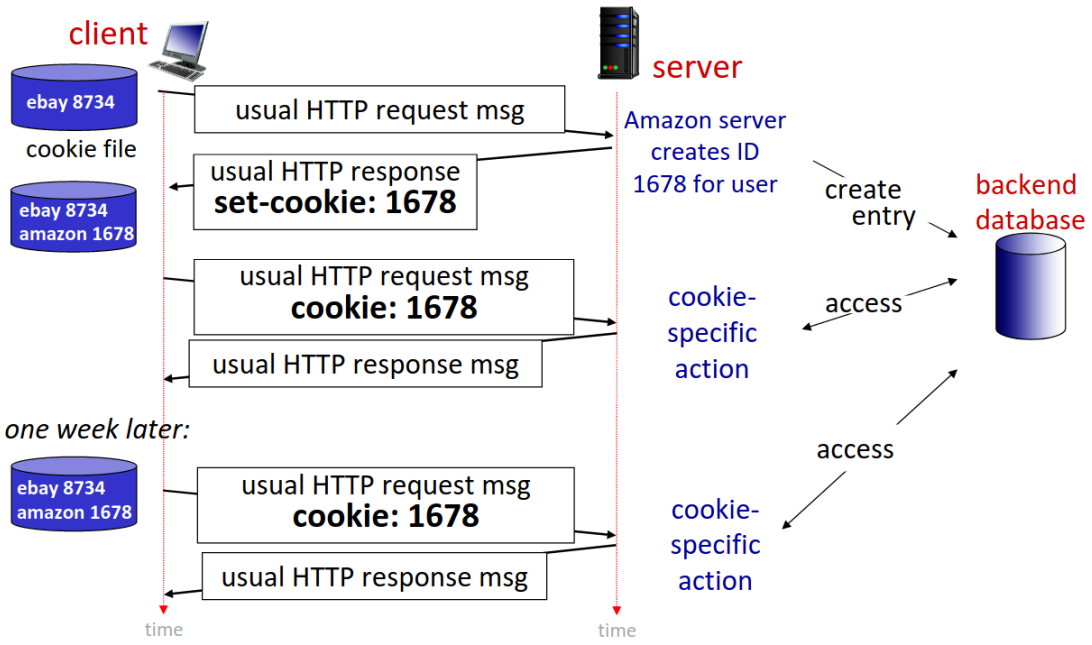
Definition 21. Cookie

Un **cookie** è un **piccolo file di testo** contenente brevi informazioni (preferenze sull'utilizzo, parametri preferiti, token di autorizzazione, ...) salvato all'interno di un client da parte di un server web

Poiché il protocollo HTTP è un protocollo **stateless**, i cookie vengono utilizzati all'interno delle applicazioni web per conservare indirettamente alcune informazioni sulle varie comunicazioni client-server effettuate, rendendo ogni richiesta HTTP indipendente dall'altra.

A seguito di un messaggio di riposta da un web server contenente il campo header `Set-Cookie`, il client salva il contenuto del cookie all'interno di un file. Durante le **successive richieste** effettuate dallo stesso client allo stesso server, tutti i cookie impostati da tale server vengono **allegati ad ogni richiesta HTTP**.

Solitamente, il cookie fornito dal server contiene un ID univoco, in modo da legare una voce nel suo database interno a quel client specifico.



La **durata di un cookie** inviato viene specificata tramite un campo header `Max-Age`, tramite il quale viene specificatoa il tempo di vita di tale cookie in **secondi**. Allo scadere di tali secondi, il client eliminerà automaticamente tale cookie. Inoltre, non c'è limite alla quantità di secondi specificabili, implicando che sia possibile specificare anche una quantità di secondi pari a mesi o anni

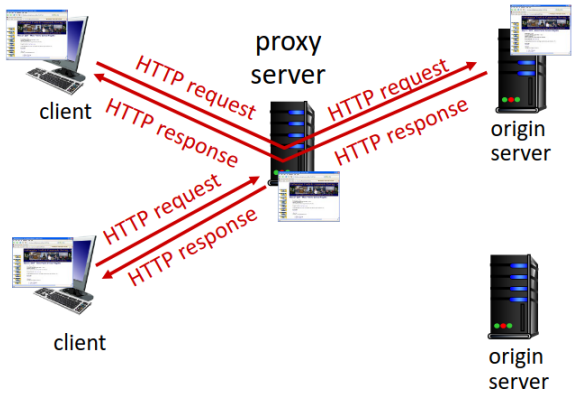
Definition 22. Proxy Server

Un **proxy server** è un server utilizzato come **intermediario** tra un client e il vero server destinatario.

Solitamente, tale tipologia server viene utilizzato per il **web caching**:

- Se il documento richiesto **è presente** nella cache del proxy server, esso viene restituito al client senza dover raggiungere il server originale
- Se il documento richiesto **non è presente** nella cache, il proxy server inoltra la richiesta del client al server di origine, memorizzando nella sua cache il documento ricevuto nella risposta, restituendolo al client

Tramite il web caching è possibile ridurre notevolmante i tempi di risposta e il traffico nei link di accesso alla rete del server di origine, consentendo ai fornitori di contenuti di essere più efficienti.



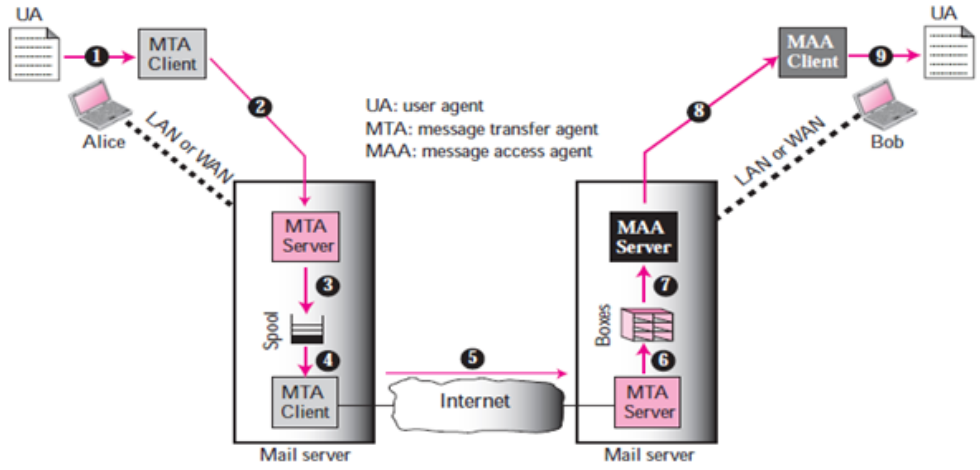
2.3 Posta elettronica

Il servizio di **posta elettronica** è costituito da tre entità fondamentali:

- Uno **User agent (UA)**, detto anche *mail reader*, è un processo attivo sul client utente attivato dall'utente stesso o da un timer. Si occupa di informare l'utente nel caso in cui sia disponibile una nuova email da leggere nella sua casella di posta.

  Inoltre, lo user agent permette la composizione, l'editing, l'invio e la lettura di messaggi di posta elettronica. Ogni messaggio di posta inviato da un UA viene passato ad un MTA
- **Mail Transfer Agent (MTA)**, è un processo attivo su un mail server utilizzato per il trasferimento tramite Internet di un messaggio ricevuto da un UA o da un altro MTA
- **Mail Access Agent (MAA)**, è un processo attivo su un mail server utilizzato per leggere i messaggi di posta in arrivo

Ogni **mail server** è dotato di una **casella di posta (mailbox)**, contenente i messaggi in arrivo per l'utente, ed una **coda di messaggi**, contenente i messaggi dell'utente ancora da inviare.



2.3.1 Protocolli SMTP e MIME

Definition 23. Protocollo SMTP

Il **protocollo SMTP (Simple Mail Transfer Protocol)** è un protocollo a livello di applicazione utilizzato per l'invio di messaggi di posta elettronica in formato ASCII. La sua porta di riferimento comune all'interno dei socket è la **porta 25**.

Il protocollo SMTP effettuta un **trasferimento diretto**, ossia dal mail server mittente a quello destinatario (dunque senza mail server intermedi), basato su un'**interazione comando/risposta**: viene inviato un comando in testo ASCII e viene ricevuta una risposta equivalente ad un codice di stato.

Inoltre, il protocollo SMTP fa uso del **protocollo TCP**:

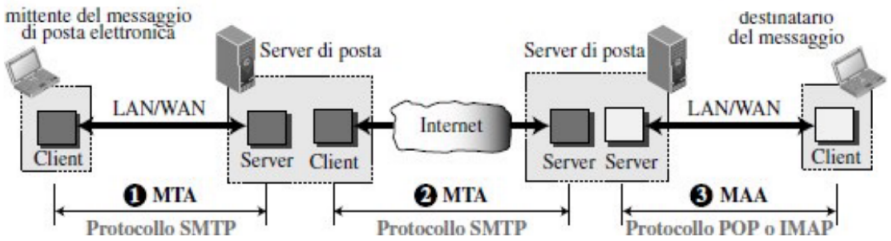
1. Il client avvia una connessione TCP con il server utilizzando la porta 25, rimanendo in attesa che il server accetti la connessione (TCP handshaking)
2. Vengono scambiati messaggi di posta tra client e server **(connessione persistente)**
3. La connessione TCP viene chiusa

Esempio:

1. Alice usa il suo UA per comporre il messaggio da inviare all'indirizzo di posta elettronica `bob@someschool.edu`
2. L'UA di Alice invia il messaggio al mail server di Alice, il quale porrà tale messaggio nella sua coda di messaggi. Successivamente, il client SMTP presente sul mail server di Alice apre una connessione TCP con il mail server di Bob
3. Il client SMTP invia il messaggio di Alice sulla connessione TCP tramite il suo MTA

4. Il mail server di Bob riceve il messaggio e lo pone nella casella di posta di Bob
5. Bob invoca il suo UA per leggere il messaggio, il quale preleverà il messaggio tramite l'MAA presente sul suo mail server

   (NB: tale operazione *non* è svolta dal protocollo SMTP, bensì dal protocollo POP3 o dal protocollo IMAP che vedremo in seguito)



In particolare, lo scambio di messaggi viene gestito dal protocollo SMTP nel seguente modo:

1. Il client SMTP **tenta di stabilire** una connessione TCP sulla porta 25 con il server STMP. Se il server è attivo, la connessione TCP viene stabilita. Altrimenti, il client riproverà dopo un determinato lasso di tempo.
2. Una volta stabilita la connessione, il client e il server effettuano una **forma aggiuntiva di handshaking**, dove il client indica al server l'indirizzo email del mittente e del destinatario
3. Il client invia il messaggio sulla connessione TCP. Una volta ricevuto il messaggio, se ci sono altri messaggi da inviare viene utilizzata la stessa connessione TCP (**connessione persistente**). Altrimenti, il client invia al server una richiesta di chiusura della connessione.

Esempio:

- Di seguito, vediamo un esempio di interazione tra un server SMTP, indicato con S, e un client SMTP, indicato con C.

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```