Reti di Calcolatori e Laboratorio

Federico Matteoni

Indice

1 Introduzione

Appunti del corso di **Reti di Calcolatori** presi a lezione da **Federico Matteoni**.

Prof.: **Federica Paganelli**, federica.paganelli@unipi.it
Riferimenti web:

- *elearning.di.unipi.it/enrol/index.php?id=169*
  Password: **RETI2019**

Esame: scritto (o compitini), discussione orale facoltativa + progetto con discussione (progetto + teoria di laboratorio, progetto da consegnare 7gg prima della discussione)
Libri e materiale didattico:

- Slide su eLearning
- IETF RFC
  tools.ietf.org/rfc
  www.ietf.org/rfc.html
- "Computer Networks: A Top-Down Approach" B. A. Forouzan, F. Mosharraf, McGraw Hill

Ricevimento: stanza 355 DO, II piano

2 Rete

Definizione di rete Interconnessione di dispositivi in grado di scambiarsi informazioni, come end system, router, switch e modem.
Gli end system possono essere di due tipi:

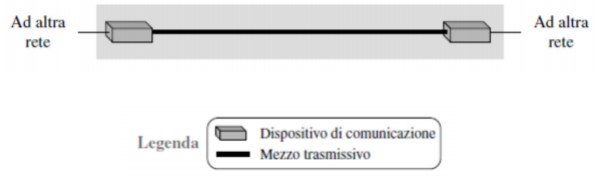
> **Host**: una macchina, in genere di proprietà degli utenti, **dedicata ad eseguire applicazioni**. Esempi: desktop, portatile, smartphone, tablet...
>
> **Server**: una macchina, tipicamente con elevate prestazioni, destinata ad eseguire programmi che **forniscono servizi** a diverse applicazioni utente. Esempi: posta elettronica, web, ...

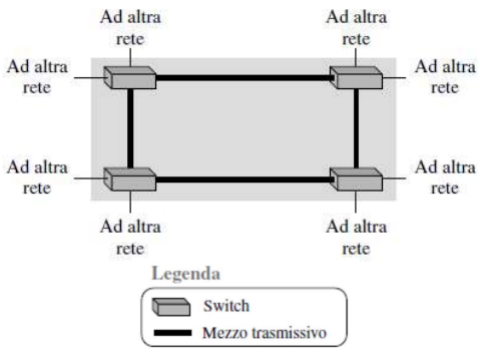
Con il termine host si può anche indicare un server.

2.1 Tipi di Rete

Local Area Network Una **LAN è una rete di area geografica limitata**: un ufficio, una casa ecc.. I dispositivi comunicano attraverso una determinata tecnologica: switch, BUS, HUB ecc..
In una rete locale tipicamente una serie di host comunicano tra loro attraverso, ad esempio, uno switch centrale.

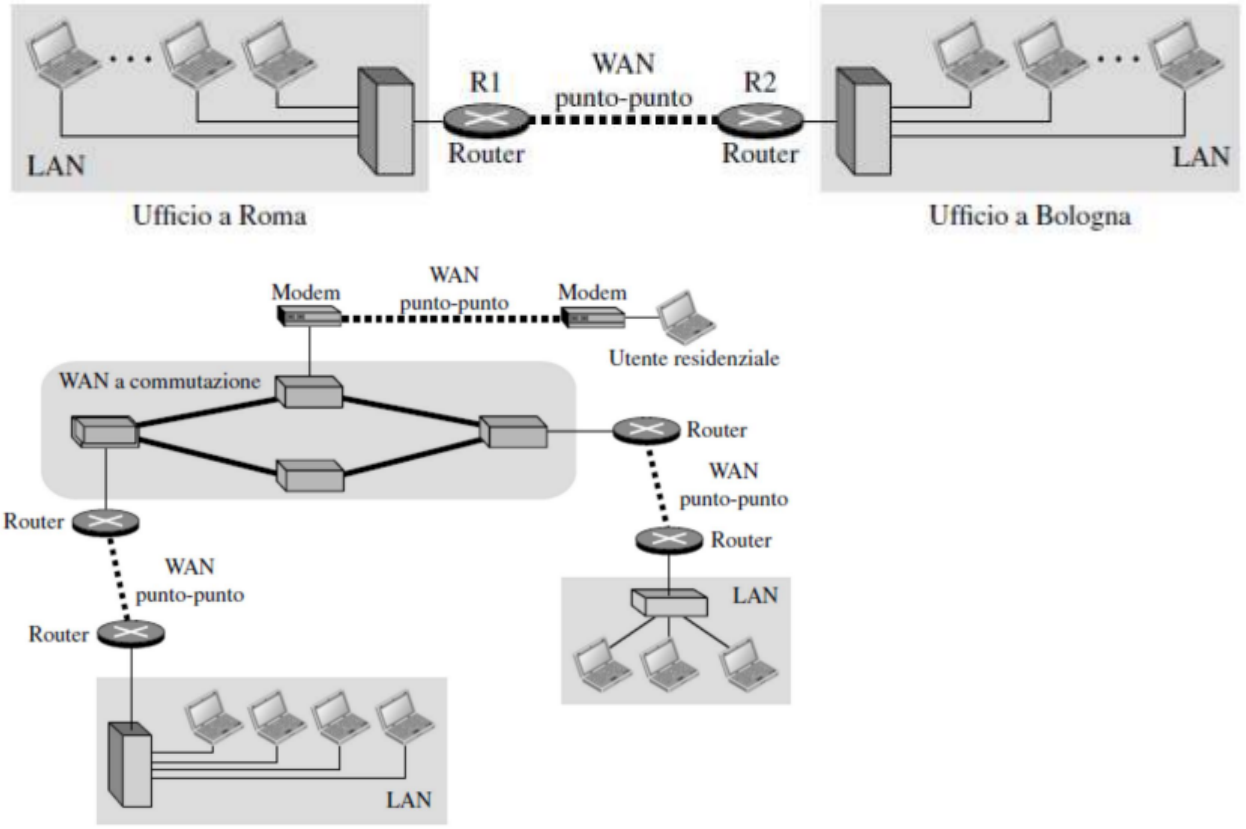
Wide Area Network Alcuni esempi:





2.2 Internetwork

Una **internetwork** si crea quando si **interconnettono diverse reti**. Alcuni esempi:



2.3 Switching

Una rete internet è formata dall'interconnesione di reti composte da link e dispositivi capaci di scambiarsi informazioni. In particolare, i sistemi terminali comunicano tra di loro per mezzo di dispositivi come switch, router ecc. che si trovano nel percorso tra i sistemi sorgente e destinazione.

Switched Network Reti a commutazione di circuito, tipico delle vecchie reti telefoniche
Le risorse sono riservate end-to-end per una connessione. Le risorse di rete (es. bandwidth) vengono suddivise in pezzi, e ciascun pezzo è allocato ai vari collegamenti. Le risorse rimangono inattive se non vengono utilizzate, cioè **non c'è condivisione**. L'allocazione della rete rende necessario un setup della comunicazione.
A tutti gli effetti vi è un circuito dedicato per tutta la durata della connessione. Ciò è rende poco flessibile l'utilizzo delle risorse (**overprovisioning**).

Packet-Switched Network Reti a commutazione di pacchetto, più moderno
Flusso di dati punto-punto suddiviso in pacchetti. I pacchetti degli utenti condividono le risorse di rete. Ciascun pacchetto utilizza completamente il canale.
Store and Forward: il commutatore deve ricevere l'intero pacchetto prima di ritrasmetterlo in uscita.
Le risorse vengono usate **a seconda delle necessità**. Vi è **contesa per le risorse**: la richiesta di risorse può eccedere la disponibilità e si può verificare **congestione** quando i pacchetti vengono accodati in attesa di utilizzare il collegamento. Si possono anche verificare perdite.

3 Internet

L'internetwork più famosa ed utilizzata è **internet**, ed è composta da migliaia di reti interconnesse. **Ogni rete** connessa ad internet **deve utilizzare il protocollo IP** e rispettare certe convenzioni su nomi ed indirizzi. Si possono aggiungere nuove reti ad internet molto facilmente.

Dispositivi in internet I **dispositivi** connessi ad internet possono essere host, end systems come PC, workstations, servers, pda, smartphones ecc...
I **link di comunicazione** possono essere fibre ottiche, doppini telefonici, cavi coassiali, onde radio... Le **entità software** in internet possono essere:

Applicazioni e processi

Protocolli: regolamentano la trasmissione e la ricezione di messaggi (TCP, IP, HTTP, FTP, PPP...)

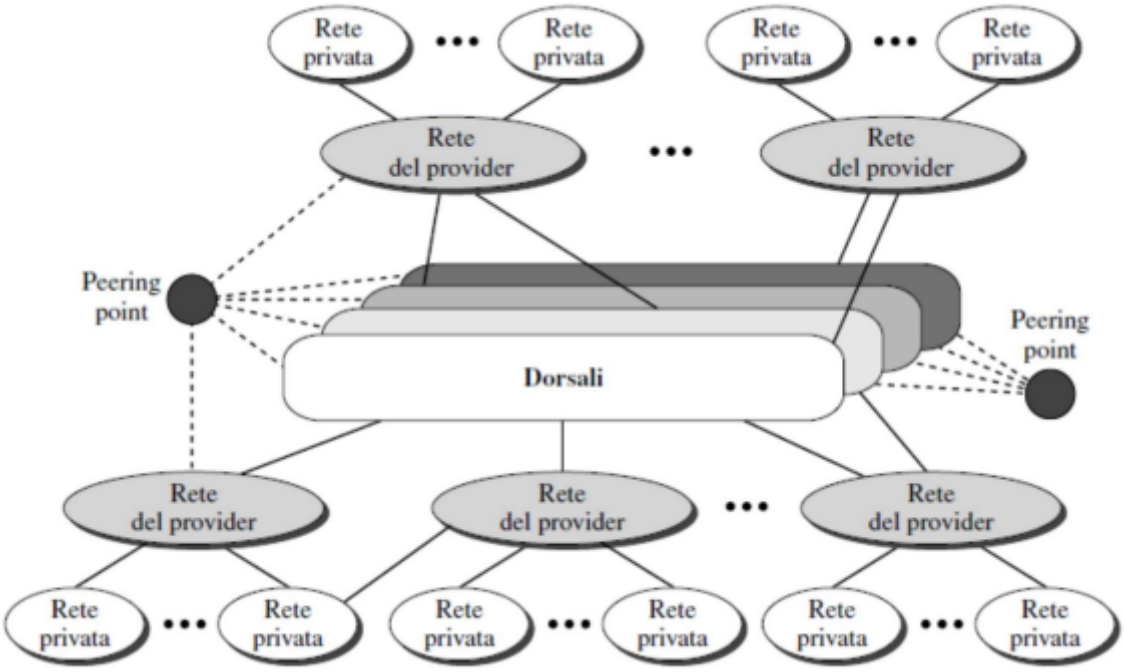
Interfacce

Standard di internet e del web: RFC (Request for Comments) e W3C.

Internet è una visione dei servizi. L'infrastruttura di comunicazione permette alle applicazioni distribuite di scambiare informazioni (WWW, e-mail, giochi, e-commerce, controllo remoto...) e fornisce loro **servizi di comunicazione connectionless** (senza garanzia di consegna) o **connection-oriented** (dati garantiti in integrità, completezza ed ordine).

3.1 Enti Ufficiali

L'**Internet Engineering Task Force** (IETF) è l'organismo che studia e sviluppa i protocolli in uso su internet. Si basa su gruppi di lavoro a cui chiunque può accedere. I documenti ufficiali che pubblica, dove descrivono i protocolli usati in internet, sono gli RFC/STD (Request for Comments/STanDards).
L'**Internet Corporation for Assigned Names and Numbers** (ICANN) si occupa di coordinare il sistema dei **nomi di dominio** (DNS) e assegna i gruppi di indirizzi, gli identificativi di protocollo.



Peering point: interconnessione tra due sistemi autonomi

3.2 Reti di accesso

Il collegamento tra l'utente ed il primo router di internet è detto **rete di accesso**. Può avvenire in 3 modi:

Tramite rete telefonica: servizio dial-up, ADSL...

Tramite reti wireless

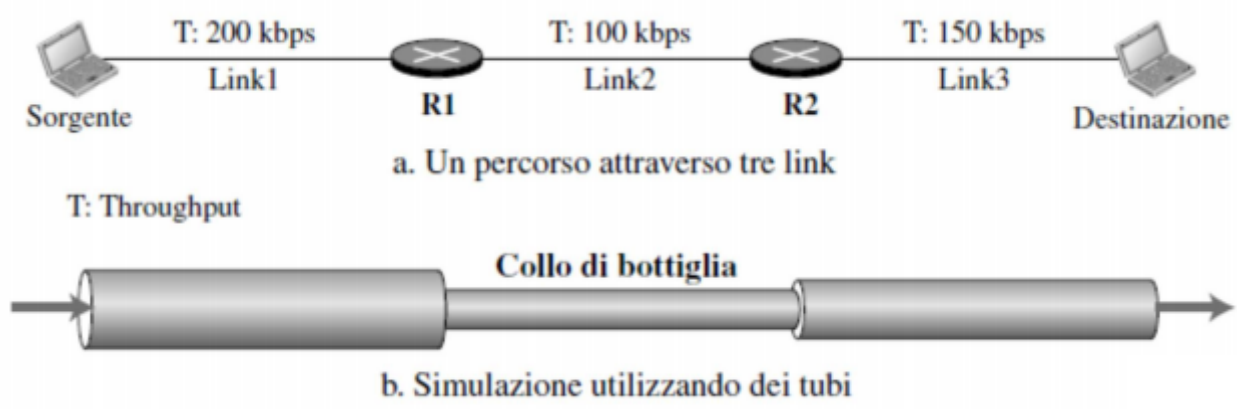
Collegamento diretto, come collegamenti WAN dedicati ad alta velocità (aziende e università)

4 Metriche di Riferimento

Come misurare le prestazioni della rete? Tramite una serie di metriche:

Bandwith o ampiezza di banda: è la larghezza dell'intervallo di frequenze utilizzato dal sistema trasmissivo (Hz).
Bitrate o **transmission rate**: quantità di bit che possono essere trasmessi o ricevuti nell'unità di tempo (bit/secondo, bps)
Il bitrate dipende dalla bandwidth e dalla tecnica trasmissiva utilizzata.

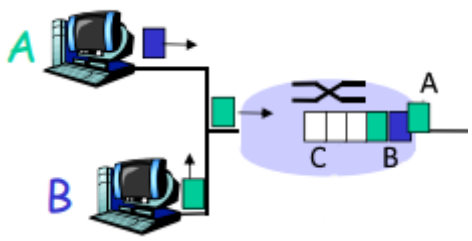
Throughput: la quantità di traffico che arriva realmente a destinazione nell'unità di tempo (al netto di perdite sulla rete, funzionamento dei protocolli ecc...).



Non è detto che corrisponda alla bandwidth perché ci potrebbe essere un collo di bottiglia.

Latenza o ritardo: il tempo richiesto affinché un messaggio arrivi a destinazione dal momento in cui il primo bit parte dalla sorgente.
`latenza = ritardo di propagazione + ritardo di trasmissione + ritardo di accodamento + ritardo di elaborazione`

Perdita di pacchetti. Come si può verificare?



A → pacchetti **in attesa** di essere trasmessi (*ritardo*)
B → pacchetti **accodati** (*ritardo*)
C → buffer **liberi** (se non ci sono buffer liberi, i pacchetti in arrivo vengono scartati, *perdita*)

I pacchetti da spedire vengono accodati nei buffer dei router. Di solito, il tasso di arrivo dei pacchetti sul router eccede le capacità del router di evaderli, quindi **i pacchetti si accodano in attesa del proprio turno**.

Il **ritardo di elaborazione** è dato dal controllo sui bit e dalla determinazione del canale di uscita (trascurabile)
Il **ritardo di accodamento** è dato dall'attesa di un pacchetto di essere trasmesso (B)
Il **ritardo di trasmissione** è il tempo impiegato per trasmettere un pacchetto sul link.
$R_{trasmissione}$ `= R/L`
`R` = rate di trasmissione del collegamento, in bps

L = lunghezza del pacchetto in bit
Il **ritardo di propagazione** è il tempo impiegato da 1 bit per essere propagato da un nodo all'altro.
$R_{propagazione}$ = d/s
d = lunghezza del collegamento
s = velocità di propagazione del collegamento (si usa la velocità della luce, circa 3 x 10^8 m/s)

d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}
d_{proc} = **ritardo di elaborazione**, pochi microsecondi
d_{queue} = **ritardo di accodamento**, dipende dalla congestione
d_{trans} = **ritardo di trasmissione**, L/R e significativo a lunga distanza
d_{prop} = **ritardo di propagazione**, d/s, da pochi microsecondi a centinaia di millisecondi

5 Modelli Stratificati

https : *//elearning.di.unipi.it/pluginfile.php/*27387*/mod*$_r$*esource/content/*1*/L*02$_i$*ntroduzione*$_p$*rotocolli.pdf*

Perché usare un modello a strati Per mandare dei dati da un host ad un altro comunicando su una rete, si devono eseguire una **serie di operazioni**: **trovare il percorso** di rete da attraversare, **decidere in che modo spedire e codificare** i dati, **risolvere eventuali problemi** di comunicazione e altro ancora.
Programmare ogni volta tutto il procedimento è un **lavoro estremamente complesso** e ripetitivo. Un modello a strati **astrae su più livelli il problema della trasmissione dati** in modo da fornire di volta in volta strumenti utili al programmatore per poter evitare di "*reinventare la ruota*".

Definizioni generali Nelle architetture di comunicazione a strati sono importanti una serie di definizioni:

- Stratificazione
- Information hiding
- Separation of concern
- Modello ISO/OSI
- Stack TCP/IP

Tali definizioni verranno viste durante il corso.

Lo Strato Uno **strato** è un **modulo interamente definito** attraverso i servizi, le interfacce e i protocolli che lo caratterizzano. Si indica anche col nome di livello.
Uno strato n **comunica direttamente** con lo strato n di un'altra unità tramite un **protocollo assegnato**. Lo stesso strato n può richiedere servizi allo strato n-1 attraverso la **loro interfaccia**, e fornisce servizi allo strato n+1 attraverso la **rispettiva interfaccia**.

Es. modello stratificato: sistema postale Vedi slide
https : *//elearning.di.unipi.it/pluginfile.php/*27387*/mod*$_r$*esource/content/*1*/L*02$_i$*ntroduzione*$_p$*rotocolli.pdf* , 48
Dal livello più alto al livello più basso per la spedizione, viceversa per la ricezione. Un problema importante che si incontra quando si manda una lettera, ad esempio, dall'Italia al Giappone è la traduzione. In una **serie di passi**, in cui in ognuno viene **eseguito un particolare compito su un messaggio**, che viene poi **trasferito ad un altro livello**. Nell'esempio, la segretaria prepara lettera (traduce in giapponese e imbusta) affinché il postino la possa prendere. Però il "messaggio" della segretaria è "scritto" per essere interpretato dalla segretaria giapponese, il direttore italiano scrive per il direttore giapponese. **Messaggi di un livello del sistema che spedisce sono scritti per essere interpretati dal medesimo livello del sistema ricevente**.

5.1 Perché stratificare

La stratificazione è molto utile per **scomporre il sistema complesso della gestione della comunicazione**. Prendo un sistema estremamente costoso da costruire per una singola coppia di aziende, quindi lo trasformo in strati così che il costo della singola lettera sia irrisorio.
Definisco funzioni di base per effettuare trasferimento e agenti che le svolgono. Principi di base:

Separation of Concern
Far fare ad un determinato strato **solo ciò che gli compete**, **delegando agli altri tutto ciò che è delegabile**

Information Hiding
Nascondo ad un determinato strato **le informazioni non indispensabili allo svolgere della sua operazione**.

Esempio Se traduco il modello postale nel modello a strati ho, ad esempio:

Utente		Utente
Segretaria		Segretaria
Postino		Postino
Smistamento		Smistamento
Stazione		Stazione
	—Viaggio→	

5.2 Smistamento Intermedio

Spedire un pacchetto che è destinato ad determinato livello intermedio. Quindi **arrivo fino al corrispondente livello intermedio per evitare che si possano esporre info sensibili**.

5.3 Elementi fondamentali

Gli **elementi fondamentali** del modello stratificato sono:

Flusso dati

Servizio: una **funzione** che uno strato offre allo strato superiore, attraverso un'interfaccia.

Protocollo

Interfaccia: **insieme di regole** che governano il formato e il significato dei frame, pacchetti o messaggi che vengono **scambiati tra strati adiacenti della stessa entità**.

I servizi indicano *cosa* si può fare, le interfaccie regolano *come* si può fare.

5.4 Modalità di Servizio

I **dati** possono essere scambiati in due modalità diverse:

Connection-Oriented: il livello di trasferimento stabilisce una **connessione logica** tra due sistemi. La connessione è quindi **gestita**:

- **Instaurazione** della connessione
- **Trasferimento** dei dati
- **Chiusura** della connessione

Connectionless: i dati vengono **trasferiti senza stabilire una connessione**.

5.5 Vantaggi

Il vantaggio più grosso è che **sviluppare il singolo strato è più semplice ed economico rispetto a sviluppare tutto il sistema complesso**. Questo perché **i servizi degli strati inferiori vengono usati da più entità che implementano gli strati superiori**.

6 Protocolli

Cos'è un protocollo Un **protocollo** è un **insieme di regole** che dice come comunicare ed esporre dati verso l'esterno. I protocolli **definiscono il formato e l'ordine dei messaggi inviati e ricevuti, con le azioni per trasmette e ricevere tali messaggi**.

6.1 Incapsulamento

Processo in cui **aggiungo strati, "involucri" al messaggio originale che vengono man mano tolti alla destinazione**.

7 OSI RM (Open Systems Interconnction Reference Model)

Le prime reti erano chiuse, composte da tecnologie e protocolli proprietari. Alcuni esempi sono ARPANET, SNA (IBM), DNA (Digital). Non potevano intercomunicare tra loro, perché usavano **protocolli diversi**, erano costruite per servizi specifici (TELCO). Insorse quindi un obiettivo: creare un **modello di riferimento per sistemi aperti**, per permettere a qualsiasi terminale di poter comunicare mediante qualsiasi rete. C'era quindi necessità di **accordarsi sulle regole**.

OSI L'OSI è una **collezione di protocolli aperti**: questo significa che i loro **dettagli sono pubblici** e i **cambiamenti vengono gestiti da un'organizzazione con partecipazione aperta al pubblico**. **Un sistema che implementa i protocolli aperti è un sistema aperto**.

7.0.1 Pila di protocolli

L'**OSI** prevede **sette strati di protocolli**:

7. Applicativo: elaborazione dei dati

6. Presentazione: unificazione dei dati, preparazione del **pacchetto** da trasmettere/ricevere

5. Sessione: controllo del dialogo tra gli host sorgente e destinazione.

4. Trasporto: offre il vero e proprio trasferimento dati tra gli host terminali, cioè **astrae la logica** con la quale si scambiano i dati tra host e gestisce gli errori. Realizza il **dialogo end-to-end**.

3. Rete: instradamento del traffico (principalmente router, offre il servizio di consegna attraverso il sistema distribuito dei nodi intermedi).

2. Datalink: consegna il frame tra le interfacce, interpretato dalla scheda di rete.

1. Fisico: modulazione del segnale elettrico per trasmettere correttamente il flusso di bit sul mezzo fisico.

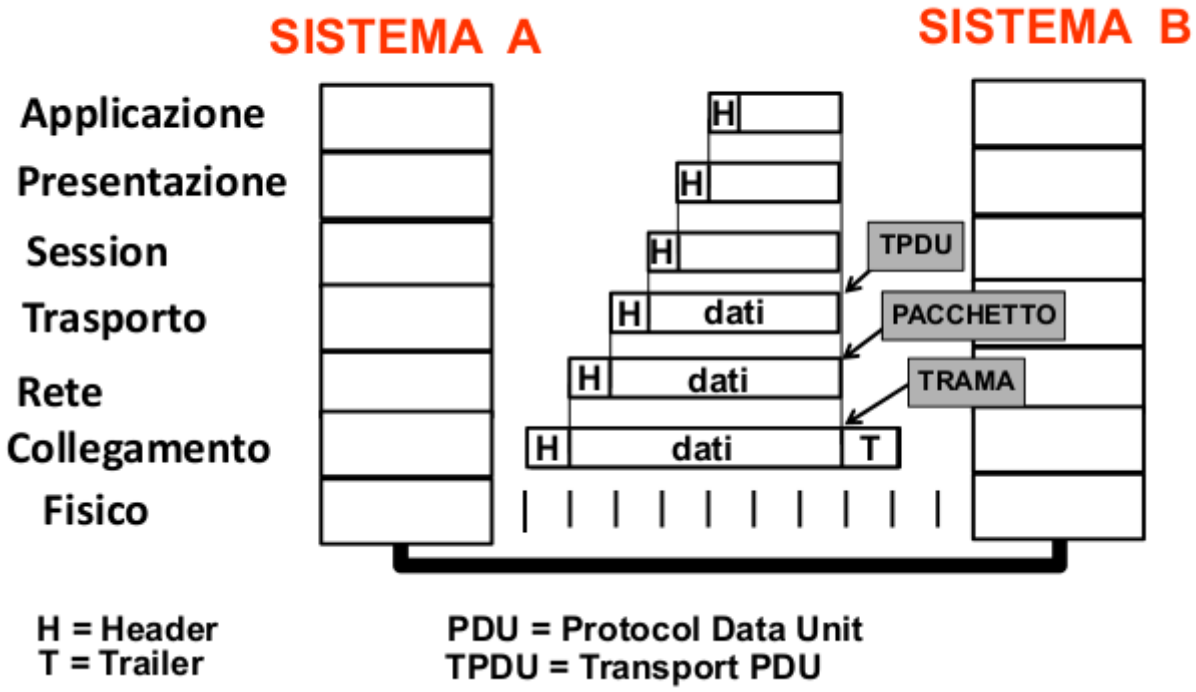
I livelli **7-5** si possono raggruppare in più modi, principalmente sono gli strati di supporto all'elaborazione e all'interazione con l'utente. Sono livelli **software**
I livelli **4-1** sono **software e hardware**. Sono strati di supporto alla rete e all'infrastruttura trasmissiva, cioè gestiscono la vera e propria trasmissione dei dati.

8 Flusso dell'Informazione

Per le reti, l'informazione ha origine al **livello applicativo**, che la genera per mandarla in remoto. Una volta generata, essa discende i vari livelli fino al canale fisico e ogni livello **aggiunge** all'informazione ricevuta dal livello superiore **una – o più – propria sezione informativa** sotto forma di **header**, contenente informazioni esclusive di quel livello. Per l'informazione ricevuta, si segue il **cammino inverso**, quindi dal basso verso il livello applicativo, e ogni livello "spacchetta" l'header del livello corrispondente, ne legge le informazioni esclusive e lo gestisce appositamente.
Il **processo di incapsulamento** è quindi **reversibile**: ogni livello esegue una operazione di incapsulamento su dati già incapsulati dal livello precedente, in modo tale da garantire la possibilità di estrarre i dati precedentemente incapsulati.

HEADER ——— DATA ——— TRAILER

Header: qualificazione del pacchetto per questo livello
Data: payload proveniente dal livello superiore
Trailer: generalmente usato in funzione del trattamento dell'errore



9 Stack protocollare TCP/IP

Il **TCP/IP** è una famiglia di protocolli attualmente utilizzati in internet. Si tratta di una **gerarchia di protocolli** costituita da **moduli interagenti**, ciascuno con funzioni specifiche.

Gerarchia Con il termine **gerarchia** s'intende che ciascun protocollo di livello superiore è **supportato dai servizi forniti dai protocolli di livello inferiore**. Cioè un protocollo a livello `n` realizza le sue funzionalità grazie ai protocolli a livello `n-1`.

9.1 I Livelli

Lo stack TCP/IP in origine era intesa come **quattro livelli software** sovrastanti **un livello hardware**. Oggi è intesa come semplicemente **composta da 5 livelli**

Livello Applicativo Il livello più alto, con il quale interagisce l'utente
Identificativi risorse: URL, URI, URN
Il web: user agents, http: request, response, connessioni persistenti, GET, POST, PUT, DELETE, status code, proxy server, caching
FTP: connessioni dati e di controllo, rappresentazione
TELNET
Posta elettronica: SMTP, POP3, IMAP
DNS e risoluzioni nomi: gerarchia nomi, risoluzione iterativa e ricorsiva, formato pessaggi, nslookup...

Livello Trasporto Livello al quale si definisce la codifica e il protocollo di trasporto
Servizi: mux demux, controllo errore, connectionless
TCP: formato segmenti, gestione connessione, controllo flusso e congestione
UDP: formato segmenti

Livello Rete Dove si gestisce l'indirizzamento dei vari host
Strato di rete e funzioni
Indirizzamentoi IP: classful IPv4, NAT, sottoreti e maschere, classless, CIDR
Risoluzione IP e MAC, ARP
IPv4: formato datagramma ip, frammentazione
Routing IP e istradamento
Introduzione IPv6

Livello Link Trasferimento dati tra elementi di rete vicini
Ethernet

Livello Fisico Bit sul filo

10 Livello Applicativo

Applicazioni e processi Le applicazioni possono essere composte da **vari processi distribuiti** comunicanti fra loro. Un **processo** sono programmi eseguiti degli host di una rete. Due processi possono anche comunicare all'interno dello stesso host attraverso la **comunicazione inter-processo** (definita dal S.O.).
Nella **comunicazione a livello applicativo fra due host di una rete**, due o più processi vengono eseguiti da ciascuno degli host comunicanti e **si scambiano messaggi**.

I livelli applicazione dei due host **si comportano come se esistesse un collegamento diretto** attraverso cui mandare e ricevere messaggi.

10.1 Protocollo a Livello Applicativo

Un protocollo di livello applicativo **definisce**:

Tipi di messaggi scambiati a quel livello, ad esempio di richiesta o di risposta

Sintassi dei vari tipi di messaggi, cioè i campi

Semantica dei campi

Regole per determinare quando e come un processo invia o risponde ai messaggi

10.2 Paradigmi

I due programmi applicativi devono essere entrambi in grado di richiedere e offrie servizi oppure ognuno deve occuparsi di uno dei due compiti?

Client-Server Un **numero limitato** di **processi server** che **offrono** un servizio, in esecuzione **in attesa di richieste** dai **processi client**, che **richiedono** servizi.

Client *Parla per primo*, cioè inizia il contatto con il server. Tipicamente richiede un servizio al server, ad esempio: per il web il client è implementato nel browser, per l'e-mail è implementato nel mail reader.

Server Fornisce al client il servizio richiesto e **rimane sempre attivo**. Ad esempio: un web server invia le pagine richieste, un mail server smista ed invia le mail.

Peer-to-Peer Host **peer** che possono **offrire servizi e inviare richieste**.

Misto Un misto tra i due paradigmi sopra.

10.3 Componenti di un'Applicazione di Rete

Due esempi

Web Composto da:

- Web Browser, sul **client**
- Web **Server**
- **Standard per il formato** delle risorse (pagine ecc.)
- **Protocollo** HTTP

Posta Elettronica Composta da:

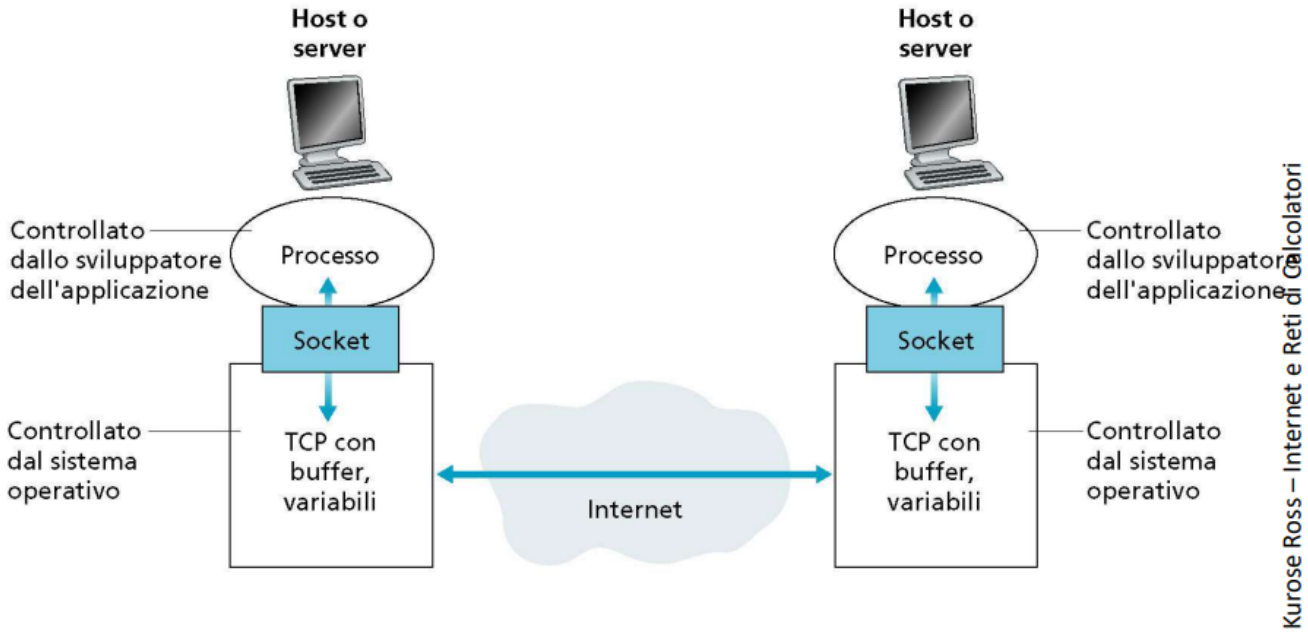
- Programmi di lettura e scrittura sul **client**
- **Server** di posta in internet
- **Standard per il formato** dei messaggi
- **Protocolli** SMTP, POP3 ecc.

10.4 Terminologia

API **Application Programming Interface**: si tratta di un **insieme di regole** che un programmatore deve rispettare per utilizzare delle risorse.

Socket Una **API** che funge da **interfaccia** tra gli strati di applicazione e di trasporto. A tutti gli effetti è la **API di internet per eccellenza**, due processi comunicano mandando dati sul socket e leggendoli da esso. Forma una **connessione logica**, l'invio e ricezione dei dati sono responsabilità del S.O. e del TCP/IP.

10.5 Identificazione di un Processo



I servizi di trasporto sono offerti al livello applicativo tramite le API. Ogni servizio di transport è **usato simultaneamente** da più processi application.
Come identifico i processi di livello application **di host diversi**? Serve un identificativo che identifichi sia l'host che il processo.
→ Coppie `<Indirizzo IP, Numero di porta>`

Indirizzo IP [32bit]	Numero di porta [16 bit]

10.6 Esempio di API: TCP

```
Connection TCPopen(IPAddress, int)      //per aprire una connessione
void TCPsend(Connection, Data)          //per spedire dati su una connessione
Data TCPreceive(Connection)             //per ricevere dati da una connessione
void TCPclose(Connection)               //per chiudere una connessione
int TCPbind(int)                        //per richiedere l'assegnazione della porta su
                                        //cui attendere le richieste di connessione

void TCPunbind(int)                     //per liberare la porta assegnata
Connection TCPaccept(int)               //per attendere le richieste di connessione

//Connection: identificata da una quadrupla
//Astraggo dalle possibili eccezioni sollevate e dal loro trattamento
```

10.7 Uso dei Servizi di Trasporto

Una coppia di processi fornisce servizi agli utenti di Internet, siano questi persone o applicazioni. La coppia di processi, tuttavia, **deve utilizzare i servizi offerti dal livello di trasporto** per la comunicazione, poiché non vi è una comunicazione fisica a livello applicativo. Le applicazioni di retesono quindi **realizzate *sopra* ai servizi di trasporto dati**.
Nel livello trasporto dello stack protocollare TCP/IP sono previsti **due protocolli di trasporto principali**:

TCP Transfer Control Protocol

Connection-Oriented: è richiesto un setup tra client e server

Trasporto **affidabile** tra mittente e destinatario

Controllo del flusso: il mittente non *inonderà* di dati il destinatario

Controllo di congestione: *limita* il mittente quando la rete è satura

Non offre garanzie di timing né di ampiezza minima di banda

UDP User Datagram Protocol

Connectionless

Trasporto **non affidabile**

NO controllo del flusso

NO controllo di congestione

NO garanzie di timing o di ampiezza minima di banda

Quindi quali applicazioni usano UDP, e perché?

Che tipo di trasporto richiede un'applicazione?

Throughput Anche detta **banda**, è la frequenza alla quale il processo mittente può inviare i bit al processo ricevente. Alcune applicazioni (es. multimedia) richiedono una **banda minima** per essere efficaci, altri (**elastic apps**) usano la banda che trovano a disposizione.
Velocità di trasferimento $\neq$ velocità di propagazione

Perdita di dati Alcune applicazioni (es. audio) possono tollerare alcune perdite, altre (es. telnet, trasferimento file) richiedono un trasferimenti dati **affidabile** al 100

Timing Alcune applicazioni (es. teleconferenze, videogame) richiedono un basso ritardo per essere efficaci

Applicazione	Tolleranza alla perdita dati	Throughput	Sensibilità al tempo
Trasferimento file	No	Variabile	No
Posta Elettronica	No	Variabile	No
Documenti Web	No	Variabile	No
Audio/Video in tempo reale	Si	Audio: 5Kbps – 1 Mbps Video: 10Kbps – 5MKbps	Si, centinaia di millisecondi
Audio/Video memorizzati	Si	Audio: 5Kbps – 1 Mbps Video: 10Kbps – 5MKbps	Si, pochi secondi
Videogame	Si	Fino a pochi Kbps	Si, centinaia di millisecondi
Messaggistica istantanea	No	Variabile	Si e no

Applicazione	Protocollo a Livello Applicativo	Protocollo di Trasporto
Posta Elettronica	SMTP (RFC 2821)	TCP
Accesso a terminali remoti	Telnet (RFC 854)	TCP
Web	HTTP (RFC 2616)	TCP
Trasferimento file	FTP (RFC 959)	TCP
Streaming multimediale	HTTP (es. YouTube), RTP (RFC 1889)	TCP o UDP
Telefonia internet	SIP, RTP, proprietario (es. Skype)	Tipicamente UDP

11 Applicazioni Web e HTTP

11.1 Terminologia

WEB Consiste di *oggetti* indirizzati da un **URL** (Uniform Resource Locator)

Pagine Web Solitamente formate da: *pagine WEB* (HTML, Javascript...) e diversi **oggetti referenziati** (altre pagine, immagini, script...)

Browser Lo user agent per il web, ad esempio: Chrome, Firefox, Netscape, Lynx

Web server Il server per il web, ad esempio: Apache, MS Internet Information Server

11.2 Uniform Resource Identifier

Una **URI** è una **forma generale per identificare una risorsa presente sulla rete** (IETF RFC 2396: *una Uniform Resource Identifier è una stringa compatta di caratteri usata per identificare una risorsa astratta o fisica*).
La sintassi di uno URI è stata progettata ponendo la **trascrivibilità globale** come uno degli obiettivi principali: utilizza caratteri da un **alfabeto molto limitato** (es. le lettere dell'alfabeto latino base, numeri e qualche carattere speciale).
Uno URI può essere **rappresentato in molti modi**, ad esempio: inchiostro su carta, pixel su schermo, sequenza di ottetti... L'**interpretazione di uno URI dipende soltanto dai caratteri utilizzati e non da come essi vengono rappresentati** nel protocollo di rete.

Uniform **Uniformità della sintassi** dell'identificatore, anche se i meccanismi per accedere alle risorse possono variare.

Resource Qualsiasi cosa abbia un'identità: documento, servizio, immagine, collezione di risorse...

Identifier Oggetto che può agire da riferimento verso qualcosa che ha identità

Esistono due tipi di URI:

URL Uniform Resource Locator: sottotipo di URI che identifica una risorsa attraverso il suo **meccanismo di accesso primario**, ad esempio la *posizione* nella rete.
Esempi:

URL `https://doi.org/10.1109/LCN.1988.10239`

URL `ftp://ftp.is.co.za/rfc/rfc1808.txt`

URL `https://www.apple.com/index.html`

URN Uniform Resource Name: sottotipo di URI che devono essere **globalmente univoci e persistenti**, anche quando la risorse cessa di esistere o di essere disponibile.
Esempi:

URN `urn:oasis:names:specification:docbook:dtd:xml:4.1.2:`

URN `urn:doi:10.1109/LCN.1988.10239`

11.2.1 Sintassi

La sintassi di un URI è **organizzata gerarchicamente**, con le componenti **elencate in ordine decrescente** di importanza da sinistra a destra.
Una **URI assoluta** può essere formata da **quattro** componenti

```
<scheme>://<authority><path>?<query>
```

`<scheme>` **Obbligatorio**, schema per identificare la risorsa.
Lo URI scheme **definisce il namespace** dello URI, quindi potrebbe porre ulteriori vincoli su sintassi e semantica degli identificatori che usano quello schema. Nonostante molti URL scheme prendono il nome da protocolli, **questo non implica che l'unico modo di accedere** la risorsa dello URL **sia attraverso il protocollo specificato**.

`<authority>` Elemento gerarchico per richiamare un'authority così che la gestione del namespace definito sia delegato a quella authority. Il **nome di dominio** di un host o il suo **indirizzo IP** in notazione puntata decimale.
`authority = [userinfo@]host[:port]`

`<path>` Contiene dati specifici per l'authority (o lo scheme) e **identifica la risorsa nel contesto** di quello schema e di quella autorità. Può consistere in una sequenza di segmenti.

`<query>` L'interrogazione o i dati da passare alla risorsa richiesta

Esempi:

`foo://example.com:8042/over/there?name=ferret#nose`

- `scheme = foo`
- `authority = example.com:8042`
- `path = /over/there`
- `query = name=ferret`
- `fragment = nose`

`urn:example:animal:ferret:nose`

- `scheme = urn`
- `path = example:animal:ferret:nose`

`http://maps.google.it/maps/place?q=largo+bruno+pontecorvo+pisa&hl=it`

- `scheme = http`
- `authority = maps.google.it`
- `path = /maps/place`
- `query = q=largo+bruno+pontecorvo+pisa&hl=it`

11.2.2 Assolute e Relative

Le URI possono essere assolute o relative.

URI Assoluta Identifica una risorsa **indipendentemente dal contesto** in cui è usata.

URI Relativa Informazioni per **identificare una risorsa in relazione ad un'altra URL** (è priva di `scheme` e `authority`). **Non viaggiano sulla rete**, sono interpretate dal browser in relazione al documento di partenza.

Esempio di URI relativa Sia `http://a/b/c/d;p?q` il documento di partenza, allora

`g` = `http://a/b/c/g`

`/g` = `http://a/g`

`//g` = `http://g`

`?y` = `http://a/b/c/?y`

`#s` = *documento corrente*`#s`

`g;x?y#s` = `http://a/b/c/g;x?y#s`

`..` = `http://a/b/`

`../../g` = `http://a/g`

12 HyperText Transfer Protocol

Lo HTTP è usato dal 1990 come **protocollo di trasferimento per il World Wide Web**. Definito nel seguente modo (RFC 2068, RFC 2616): ***procollo di livello applicazione*** *per sistemi di informazione distribuiti, collaborativi ed impermediali.*
Protocollo **generico**, **stateless** e **object-oriented** che può essere usato per molte attività, come name server e sistemi distribuiti di gestione oggetti, attraverso l'estensione dei suoi **request methods** (comandi). Una funzionalità dell'HTTP è la rappresentazione del tipo di dati, consentendo al sistema di essere **costruito indipendentemente dai dati che vengono trasferiti**.

12.1 HTTP URL

Lo schema `http` è usato per accedere alla risorsa attraverso il protocollo HTTP.

Sintassi La sintassi per un URL http è:

`http_URL = http://host[:port][path]`

Host: un dominio, hostname o indirizzo IP in forma decimale puntata di Internet.

Porta: un numero, se omessa viene usata la porta 80.

La risorsa è localizzata nel server in ascolto per connessioni TCP su quella porta di quell'host. Il path specifica la **Request-URI**.

12.2 Caratteristiche

Il protocollo HTTP è un protocollo **request/response**: la **connessione** viene **iniziata dal client**, che invia un **messaggio di request** al quale il server risponde con una **response**.
In quanto **generico e stateless** le coppie **richiesta/risposta sono indipendenti**.

12.2.1 Modello

Il modello del protocollo HTTP è **client-server**:

Client: browser che richiede, riceve e visualizza oggetti web.
Stabilisce una connessione con il server e invia una **richiesta sotto forma di request-method, URI e versione di protocollo**, seguito da un messaggio.

Server: web server che invia oggetti in risposta ad una richiesta.
Accetta le connessioni e serve le richieste rispondendo con i dati richiesti.

12.2.2 Connessioni

Una **connessione** è un **circuito logico di livello trasporto** stabilito tra due programmi applicativi per comunicare tra loro.

Non-Persistent Connection `http1.0: RFC 1945`
Viene stabilita una **connessione TCP separata** per raggiungere ogni URL. **Aumenta il carico** sui server HTTP e **può causare congestioni su Internet** (questo perché, ad esempio, se vengono usate tante immagini allora si creano richieste multiple del solito server in un breve lasso di tempo).

Persistent Connection `http1.1: RFC 2616`
Se non è indicato altrimenti, il client può **assumere che il server manterrà una connessione persistente**.
Lo standard specifica un **meccanismo con il quale** un client o un server **può segnalare la chiusura di una connessione TCP** (il campo `connection` nell'header). Una volta che la chiusura viene segnalata, il cliente **non deve più mandare richieste** su quella connessione.

12.3 Esempio HTTP

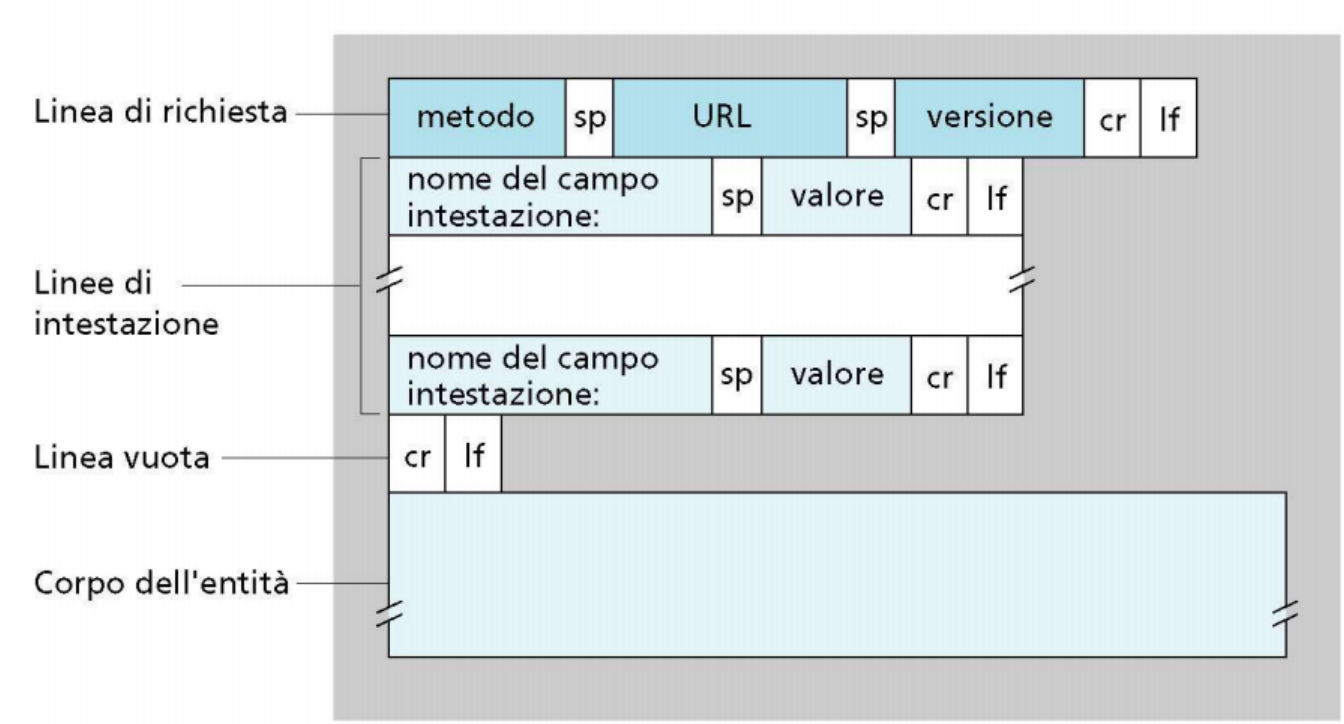
`Vedi slide`[1]

12.4 Messaggi HTTP

```
generic-message = start-line *message-header CRLF [message-body]
            start-line = Request-Line | Status-Line
```

La `start-line` distingue **request** da **response**.

HTTP Request Message



HTTP Response Message

[1] https://elearning.di.unipi.it/pluginfile.php/27477/mod_resource/content/2/L03_Applicativo_HTTP.pdf, slide 34

HTTP Request `Request-Line *( general-header | request-header | entity-header ) CRLF [message-body]`
Ad esempio:

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host:  www.w3.org
Connection:  close
User Agent:  Mozilla/4.0
Accept-language:  it

(Body)
```

HTTP Request Line `Request-Line = Method SP Request-URI SP HTTP-Version CRLF`

```
GET http://www.w3.org/pub/WW/TheProject.html HTTP/1.1
```

`Method = GET`

`Request-URI = http://www.w3.org/pub/WW/TheProject.html`

`HTTP-Version = HTTP/1.1`

```
Method = "OPTIONS" | "GET" | "HEAD" | "POST" | "PUT" | "DELETE" | "TRACE" | extension-method
```

Method: indica il **metodo che deve essere eseguito** sulla risorsa identificata dal `Request-URI`. **Case sensitive**.

HTTP-Version: indicare la versione del protocollo è pensato per consentire al mittente di indicare il formato di un messaggio e la sua capacità di capire il resto della comunicazione HTTP.

Le **URI** sono stringhe formattate in modo semplice che identificano una risorsa di rete.

12.5 Header

Gli **header** sono insiemi di coppie ¡nome:valore¿ che **specificano alcuni parametri** del messaggio trasmesso o ricevuto.

General header Relativi alla trasmissione

Data: data e ora di generazione del messaggio

Connection: consente al mittente di specificare le opzioni desiderate per quella particolare connessione. L'opzione "close" segnala che la connessione verrà chiusa al completamento della response

Transfer-encoding: indica quale (se presente) tipo di trasformazione è stata applicata al message body per trasferirlo correttamente dal mittente al destinatario (chunked, gzip...)

Cache Control

Public Indica che la response è cachable in qualsiasi cache

Private Indica che tutte o parti della response sono destinate ad un singolo utente e **non devono essere** memorizzate in una cache condivisa (shared cache). Una cache privata (non-shared) potrebbe memorizzare la response

no-cache Indica che tutte o parti della response **non devono essere memorizzate in nessuna cache**

```
general-header = Cache-Control | Connection | Date | Pragma | Transfer-Encoding | Upgrade
| Via
```

Questi header si applicano **a tutto il messaggio**. Esempi:

```
Date:  Tue, 15 Nov 1994 08:12:31 GMT
Connection:  close
Transfer-Encoding:  chunked
```

Entity header Relativi all'entità trasmessa
Content-type, Content-length, data di scadenza...

12.5.1 Request header

Relativi alla richiesta
Chi fa la richiesta, a chi viene fatta, che tipo di caratteristiche è in grado di accettare il client, autorizzazione... consente al client di passare **informazioni aggiuntive** a proposito della richiesta o del client stesso al server. Questi campi agiscono come **modificatori di richiesta**, con **semantica equivalente a quella dei parametri** di un metodo.
`request-header = Accept | Accept-Charset | Accept-Encoding | Accept-Language | Authorization | Proxy-Authorization | From | Host | If-Modified-Since | If-Unmodified-Since | If-Match | If-None-Match | If-Range | Max-Forwards | Range | Referer | User-Agent`

Accept Specifica che tipi di media sono accettabili nella response. Parametro `q` per indicare un fattore di qualità relativo, default a 1.

Accept-Charset Indica il set di caratteri accettato per la risposta

Accept-Encoding Tipi di trasformazioni accettate (es. compressione)

```
Accept:  text/plain;q=0.5, text/html, text/x-dvi;q=0.8, text/x-c
Accept-Charset:  iso-8859-5, unicode-1-1;q=0.8
Accept-Encoding:  compress, gzip
```

(Alcuni) metodi request:

OPTIONS Richiede **solo le opzioni di comunicazione** associate ad un URL o al server stesso (capacità, metodi esposti ecc.). Un esempio:

```
OPTIONS http://192.168.11.66/manual/index.html HTTP/1.1
host:  192.168.11.66
Connection:  close

HTTP/1.1 200 OK
Date:  Sun, 14 May 2000 19:52:12 GMT
Server:  Apache/1.3.9 (Unix) (Red Hat/Linux)
Content-Length:  0
Allow:  GET, HEAD, OPTIONS, TRACE
Connection:  close
```

GET Richiede il trasferimento di una risorsa identificata da un URL o le operazioni associate all'URL stessa. Un esempio:

```
GET http://192.168.11.66 HTTP/1.1
host:  192.168.11.66
Connection:  close
```

Response

```
HTTP/1.1 200 OK
Date:  Sun, 14 May 2000 19:57:13 GMT
Server:  Apache/1.3.9 (Unix) (Red Hat/Linux)
Last-Modified:  Tue, 21 Sep 1999 14:46:36 GMT
ETag:  "f2fc-799-37e79a4c"
Accept-Ranges:  bytes
Content-Length:  1945
Connection:  close
Content-Type:  text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2
Final//EN">
<HTML>...
```

Sono possibili **GET condizionali e parziali**. Esempio di GET condizionale:

```
GET http://192.168.11.66 HTTP/1.1
Host:  192.168.11.66
If-Modified-Since:  Tue, 21 Sep 1999
14:46:36 GMT
```

Response:
```
HTTP/1.1 304 Not Modified
Date:  Wed, 22 Sep 1999 15:06:36 GMT
Server:  Apache/1.3.9 (Unix) (RedHat/Linux)
```

HEAD Simile al GET, ma il server **non trasferisce il body** nella response. Utile per controllare lo stato dei documenti (validità, modifiche...). Un esempio:
```
HEAD http://192.168.11.66 HTTP/1.1
host:  192.168.11.66
Connection:  close
```

Response (notare la somiglianza con GET, esclusa la mancanza qua del body):
```
HTTP/1.1 200 OK
Date:  Sun, 14 May 2000 20:02:41 GMT
Server:  Apache/1.3.9 (Unix) (Red Hat/Linux)
Last-Modified:  Tue, 21 Sep 1999 14:46:36 GMT
ETag:  "f2fc-799-37e79a4c"
Accept-Ranges:  bytes
Content-Length:  1945
Connection:  close
Content-Type:  text/html
```

POST Serve per **inviare dal client al server** informazioni inserite nel body del messaggio.
In teoria lo standard dice che il metodo POST è usato per richiedere che il server **accetti l'entità racchiusa nella richiesta come nuovo suboordinato della risorsa identificata** dallo Request-URI nel Request-Line.
Nella pratica, **la funzionalità effettiva del metodo POST** è **determinata dal server** e solitamente dipende dalla Request-URI.

DELETE Il client **chiede di cancellare una risorsa identificata** dalla Request-URI.
Solitamente non attivo su server pubblici.

PUT Il client **chiede di creare/modificare una risorsa identificata** dalla Request-URI. Dopo posso usare una GET per recuperarla.
Solitamente non attivo su server pubblici.

Response header Nel messaggio di risposta
Server, autorizzazione richiesta...

Safe Methods Metodi che **non hanno effetti collaterali** (es. non modificano la risorsa): GET, HEAD, OPTIONS, TRACE

Idempotent Methods Metodi che **non hanno effetti ulteriori se vengono fatti N ¿ 0 richieste identiche**: GET, HEAD, PUT, DELETE, OPTIONS, TRACE

12.6 HTTP Response

```
Response = Status-Line *( general-header | response-header | entity-header ) CRLF [message-body]
```

Un esempio:
```
HTTP/1.1 200 OK
Date:  Sun, 14 May 2000 23:49:39 GMT
Server:  Apache/1.3.9 (Unix) (Red Hat/Linux)
Last-Modified:  Tue, 21 Sep 1999 14:46:36 GMT
```

Status-Line La prima linea del messaggio di risposta.
`Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF`
Esempio: `HTTP/1.1 200 OK`

Status-Code Intero a 3 cifre, risultato del tentativo di comprendere e soddisfare la richiesta.

1xx: Informational - Request received,
continuing process
2xx: Success - The action was successfully
received, understood, and accepted
3xx: Redirection - Further action must be
taken in order to complete the request
4xx: Client Error - The request contains bad
syntax or cannot be fulfilled
5xx: Server Error - The server failed to
fulfill an apparently valid request

`http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

Reason-Phrase Ha l'obiettivo di fornire una breve descrizione testuale dello Status-Code. Lo Status-Code è indirizzato ai computer mentre la Reason-Phrase è per gli umani.

12.6.1 Response Headers

Il campo response-header consente al server di passare ulteriori informazioni sulla response. Questi campi dell'header forniscono informazioni sul server e sull'accesso alla risorsa identificata dallo Request-URI.
`response-header = Age | Location | Proxy-Authenticate | Public | Retry-After | Server | Vary | Warning | WWW-Authenticate`

Esempio:
```
Age:  150 // età del doc.  se tramite Proxy
Location:  http://www.w3.org/pub/WWW/People.html
Server:  CERN/3.0 libwww/2.17
```

Age Una stima in secondi del tempo passato dalla generazione della risposta dal server di origine

Location Usato per reindirizzare il ricevente verso una destinazione diversa dalla Request-URI per il completamento della richiesta o l'identificazione di una nuova risorsa

Server Informazioni sul software usato dal server d'origine per gestire la richiesta

12.7 Negoziazione del contenuto

Le risorse possono essere **disponibili in multiple rappresentazioni**, ad esempio più lingue, formati, dimensioni e risoluzioni, o variare in altri modi ancora. La **content negotiation** è il meccanismo usato per **selezionare l'appropriata rappresentazione** quando si serve una richiesta. Ogni entità è costituita da un **entity body** e da una serie di **entity headers** che ne definiscono il contenuto e le proprietà. Gli entity header sono **informazioni sulle informazioni**, cioè **metadati**.

12.7.1 Entity Headers

`entity-header = Allow | Content-Base | Content-Encoding | Content-Language | Content-Length | Content-Location | Content-MD5 | Content-Range | Content-Type | ETag | Expires | Last-Modified | extension-header`

Content-Base URI assoluta da usare per risolvere le URL relative contenute nell'entity-body

Content-Encoding Codifica dell'entity-body (es. gzip)

Content-Language Lingua dell'entity-body (es. en, it)

Content-Type Tipo dell'entity-body (es. text/html)

Expires Valore temporale dell'entity-body (utile nel caching)

Last-Modified Data dell'ultima modifica sul server (utile nel caching)

13 Web Caching

L'obiettivo è **soddisfare una richiesta** del cliente **senza contattare il server**. Si **memorizzano copie temporanee delle risorse web** (es. pagine HTML e immagini) e si servono al client per ridurre l'uso di risorse (es. banda e workload sul server), diminuendo anche il tempo di risposta.

User Agent Cache lo **user agent** (il browser) mantiene una **copia delle risorse visitate dall'utente**.

Proxy Cache Il proxy intercetta il traffico e **mette in cache le risposte**. Le successive richieste alla stessa Request-URI **possono essere servite dal proxy senza inoltrare la richiesta** al server.

Proxy Programma intermediario che agisce sia da server che da client, con l'obiettivo di fare richieste per conto di altri client. Le richieste sono servite internamente o passandole oltre, anche traducendole, ad altri server.

14 Cookies

L'HTTP è **stateless**, per cui non mantiene info sui client. Come posso riconoscere il cliente di un'applicazione web (es. Amazon)? Come posso realizzare applicazioni web con stato (es. carrello della spesa)? Ricordiamo che **tipicamente l'utente si connette ogni volta con un indirizzo IP e porta diversi**.
Soluzione: **numerare i client** e obbligarli a **farsi riconoscere ogni volta presentando un cookie**.

Funzionamento Il client C invia al server S una normale richiesta HTTP.
Il server invia la normale risposta + una linea **Set-Cookie: 1678453**
Il client memorizza il cookie in un file associato a S, e aggiunge una linea **Cookie: 1678453** a **tutte le successive richieste** verso quel sito.
Il server confronta il cookie presentato con l'informazione che ha associato a quel cookie.