TECNOLOGIE E PROTOCOLLI DELLE RETI DI COMPUTER

Il web e internet sono due cose diverse. Internet è un sistema di comunicazione, il web è un insieme di pagine che forniscono dei servizi agli utenti. Il web è solo una delle funzioni messe a disposizione da internet. L'evoluzione del web nel tempo è stata la seguente:

1. Prima che nascesse il web, i predecessori di internet erano delle reti che collegavano università e centri di ricerca utilizzata da studiosi che si scambiavano email e file (fine anni '70, anni '80)
2. Il WWW (World Wide Web) Insieme di ipertesti (documenti navigabili con link) visualizzabili con un browser (anni '90, il WWW è nato nel 1991 grazie al linguaggio HTML e al protocollo http di Tim Berners Lee). La comunicazione era uno a molti: significa che i contenuti (le pagine web statiche) sono realizzate da POCHI UTENTI mentre gli UTENTI che usufruiscono dei contenuti sono molti (più utenti leggono le pagine web). Ma la maggior parte degli utenti non contribuisce alla creazione di contenuti.
3. WEB 2.0 (anni '2000): piattaforme di condivisione di contenuti multimediali (testi, audio, video, immagini). La comunicazione è diventata molti a molti (blog, social network, wiki), ossia TUTTI GLI UTENTI non solo sono UTILIZZATORI dei contenuti ma CONTRIBUISCONO ALLA CREAZIONE DI CONTENUTI (nei loro blog, nei social network, nei Wiki).
4. Il WEB 3.0: con questo termine si indica l'evoluzione attualmente in atto del WEB verso una rete più "intelligente" e autonoma. Le caratteristiche del WEB 3.0 sono le seguenti:

    a. **Decentralizzazione e proprietà dei dati**: i **dati** e gli **asset digitali (sono la versione digitale di immagini, video, audio, testi, contenuti dei social network, comportamenti, come ad esempio i like, che possono essere digitalizzati e memorizzati)** non sono conservati in database centralizzati, ma su una rete distribuita. Ciascun utente dovrebbe poter interagire con il web mantenendo il controllo dei propri dati. Le applicazioni sono realizzate utilizzando servizi web distribuiti (ad esempio le API di Google Maps o di OpenStreet Maps per integrare le mappe di tutto il mondo sulla propria applicazione).
    b. **WEB semantico e Intelligenza Artificiale**: utilizzo di standard e tecnologie che consentono di rendere le informazioni sul WEB non solo "leggibili dagli utenti" ma anche "comprensibili dalle macchine", in modo che esse possano, attraverso anche l'intelligenza Artificiale, "comprendere" il significato delle informazioni presenti sul WEB in modo da fornire agli utenti esperienze di navigazione personalizzate. Ad esempio, se stai cercando informazioni su un "libro" in un motore di ricerca, un sistema semantico non si limiterà a cercare parole chiave come "libro", ma cercherà di comprendere che cosa intendi per "libro" (ad esempio, se stai cercando un libro fisico, un e-book, un autore, o una recensione), e ti restituirà risultati pertinenti in base a queste informazioni contestuali.

5. WEB 4.0: con questo termine si indica un'evoluzione futura del web le cui caratteristiche non sono ancora ben delineate anche perché variano in base all'evoluzione tecnologica. Certamente alcune di queste caratteristiche sono:
    a. Utilizzo di funzionalità avanzate di Intelligenza Artificiale

b. IOT (Internet of Things: Internet delle cose): grazie a sensori e attuatori, molti oggetti (telecamere, elettrodomestici, automobili..) saranno connessi alla rete ed utilizzabili, consultabili, comandabili da remoto. Ad esempio: la scatola di pastiglie avviserà la nonna se si è scordata di prende la pastiglia, la sveglia suonerà prima se ci sarà traffico.
c. Esperienze immersive e realtà aumentata/virtuale: attraverso opportuni attuatori si vuole rendere l'esperienza di navigazione tridimensionale, ad esempio con la possibilità di realizzare riunioni virtuali con i propri colleghi o svolgere simulazioni di volo realistiche.

Visto l'incremento delle funzionalità presenti nel web sono nati dei servizi software chiamati web service in cui utenti non sono gli esseri umani ma le **applicazioni**, le quali accedono a tali servizi per ottenere dati che poi esse forniranno in un formato personalizzato, ai loro utenti "umani". I WEB service sono dei servizi per le applicazioni messi a disposizione da appositi server. Ai WEB SERVICE dunque non accedono direttamente gli utenti ma vi accedono le applicazioni per usufruirne. Si può dire che "mentre gli utenti utilizzano internet per accedere alle pagine web, le applicazioni vi accedono per ottenere risorse dai web service", sarà compito delle applicazioni rappresentare queste risorse agli esseri umani nel formato idoneo. Vi sono web service gratuiti e a pagamento.

Un esempio è il web service geocoding che consente, specificando un indirizzo, di conoscerne le coordinate geografiche (latitudine e longitudine)

Digitando in un Browser la seguente stringa osserviamo che si ottiene un documento JSON che contiene diverse informazioni sul luogo indicato (coordinate, provincia, regione ecc..):

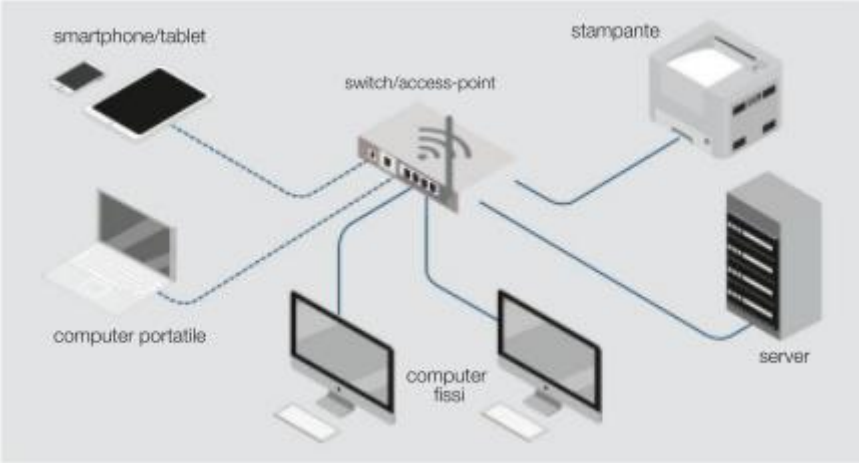
https://nominatim.openstreetmap.org/search.php?q=via+ubertosa+1+darfo+boario+terme+25047+Italy&format=json

Un po' di Storia di Internet

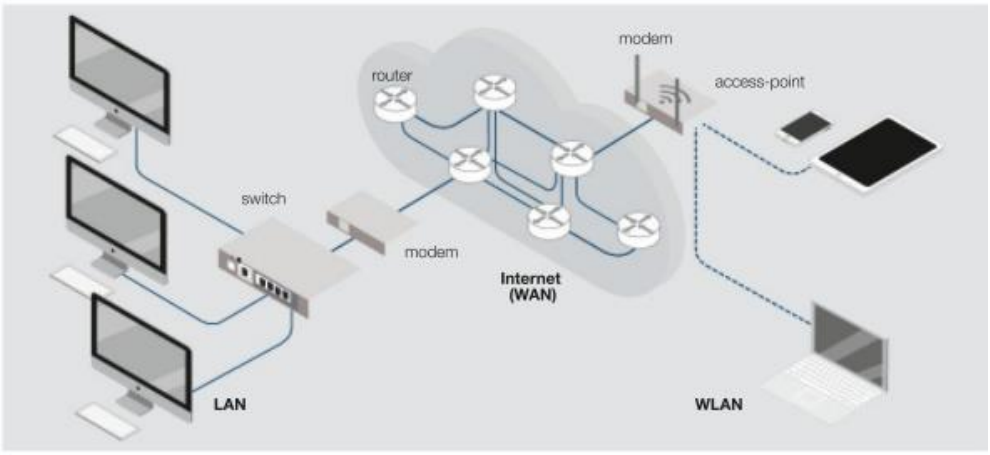
Internet è l'evoluzione di un progetto per un sistema di comunicazione di calcolatori noto con il nome di ARPANET sviluppato nel 1969 da un'agenzia del dipartimento di difesa degli USA chiamato ARPA (Advanced Research Project Agency). Il progetto era sviluppato da università e centri di ricerca e lo scopo iniziale era quello di realizzare sistemi di comunicazione sicuri e robusti per collegare, inizialmente, università americane. Secondo alcune fonti non confermate, ARPAnet è stato ideato al fine di ottenere un sistema di comunicazione in grado di resistere ad eventuali attacchi militari nucleari (si era nel periodo della guerra fredda fra USA e URSS). Inizialmente, come detto, la rete collegava fra loro le LAN di Università e centri di ricerca. Nel tempo il sistema di comunicazione ha connesso fra loro un numero sempre crescente di LAN sparse per il pianeta fino a diventare Internet, negli anni '80 del secolo scorso.

Ma come funziona questo sistema di connessione?

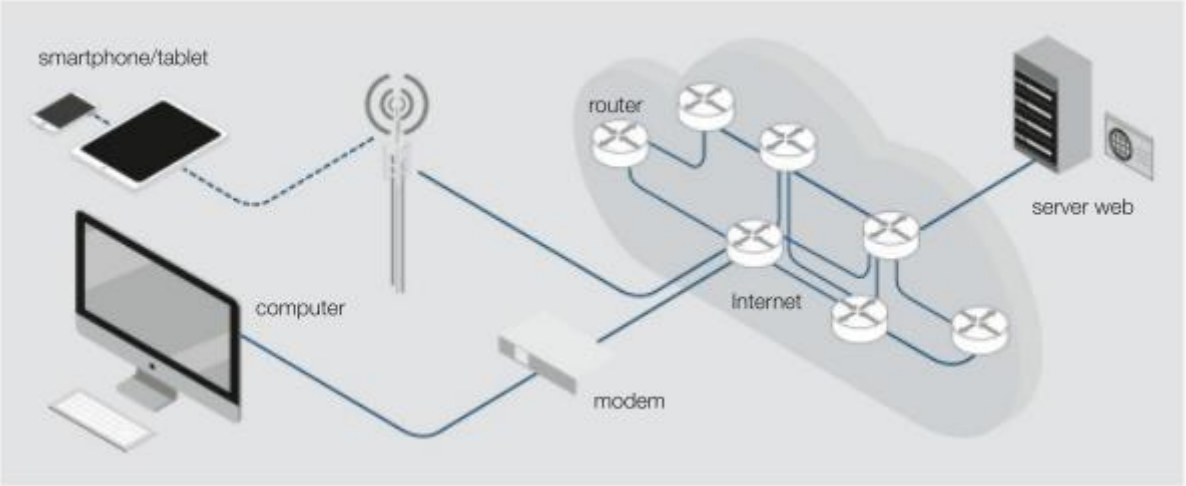
Una LAN (Local Area Network) è costituita da una serie di dispositivi (computer, stampanti, smartphone.. chiamati generalmente HOST) che sono connessi ad un dispositivo centrale chiamato SWITCH. Nel caso in cui la connessione fra gli Host e il dispositivo centrale avvenga attraverso onde radio generate da apposite antenne il dispositivo centrale viene chiamato ACCESS POINT e la LAN prende il nome di WLAN (WirelessLAN)



Come avviene la connessione fra le varie LAN? La WAN (Wide Area Network) Internet è formata da milioni di LAN (magari anche fatte da un solo HOST) CONNESSE FRA DI LORO. Ogni LAN è connessa ad un ROUTER, i router sono connessi fra di loro e hanno lo scopo di smistare il traffico nelle rete consentendo la comunicazione fra HOST che si trovano su LAN diverse.



Grazie ad Internet gli Host collegati possono scambiarsi dei dati. Il modello di riferimento per lo scambio dei dati è il modello **client/server** nel quale un host svolge il ruolo di server, ossia di dispositivo che fornisce dei servizi ad altri host (chiamati client). Un client effettua le richieste dei servizi al server e quest'ultimo fornisce il servizio. Con servizi si intende generalmente pagine web e servizi web (web service). In questo modello quindi **un server non prende mai l'iniziativa**. Esso rimane in attesa di una richiesta da parte di un client che gli chiede i dati e, quando arriva la richiesta, la soddisfa.



1. LA TECNOLOGIA PACKET-SWITCHING E LA RETE INTERNET

Come detto, l'origine della rete Internet è quella di un progetto per un sistema di comunicazione. Uno dei requisiti fondamentali del progetto era la robustezza del sistema di comunicazione, ossia che il sistema potesse funzionare anche in seguito ad eventuali default di alcuni componenti. Per questo motivo si è sviluppata la tecnologia della **commutazione di pacchetto** (**packet switching)** che è tutt'ora alla base di internet.

Che significa packet switching e perché tale tecnologia rende la rete robusta?

La tecnologia packet switching prevede che ogni insieme di dati da inviare (testo, immagini…comunque sempre byte alla fine), venga suddiviso in *pacchetti di dati*. Quando un Host deve inviare dei dati ad un altro host, tali dati vengono suddivisi in pacchetti, i pacchetti vengono "*numerati*" e spediti nella rete, i router si *passano i pacchetti* e si occupano di individuare la strada migliore (più veloce) per raggiungere l'host di destinazione. Può darsi che pacchetti diversi raggiungano l'host di destinazione seguendo strade diverse perché durante la propagazione un router ha un malfunzionamento e smette di funzionare. Questo non preclude la trasmissione di tutto il messaggio ma solo di alcuni pacchetti perché i router *vicini* a quello malfunzionante si accorgono del malfunzionamento, lo comunicano agli altri router, i quali instradano i rimanenti pacchetti del messaggio per altri cammini, per altre strade (router significa infatti instradatore). Questa tecnologia quindi garantisce il funzionamento della rete di comunicazione anche in presenza di numerosi componenti guasti garantendo la **robustezza** della rete.

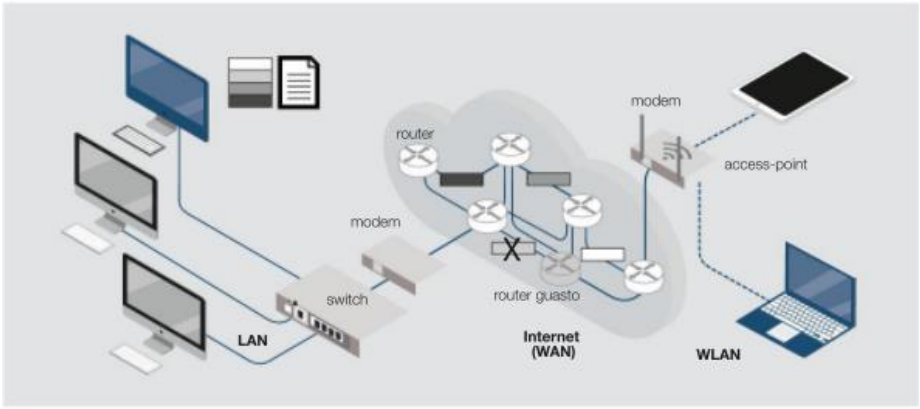
La tecnologia a commutazione di pacchetto ha altri due vantaggi: **l'affidabilità** e **l'efficienza**.

Primo vantaggio, **l'affidabilità**. Nel caso in cui un pacchetto andasse perso nel tragitto, ad esempio perché il router che lo sta elaborando si guasta proprio in quel momento oppure un cavo in fibra ottica si rompe, la trasmissione del messaggio non viene compromessa. Ecco come ciò avviene: poiché i pacchetti sono *numerati* e l'host destinatario conosce *l'indirizzo* (chiamato indirizzo IP) dell'host mittente, nel caso di smarrimento, l'host destinatario, accorgendosi della mancanza di un pacchetto mentre ricostruisce il messaggio unendo i pacchetti ricevuti, invia una richiesta al mittente di inviare di nuovo solamente il pacchetto mancante!

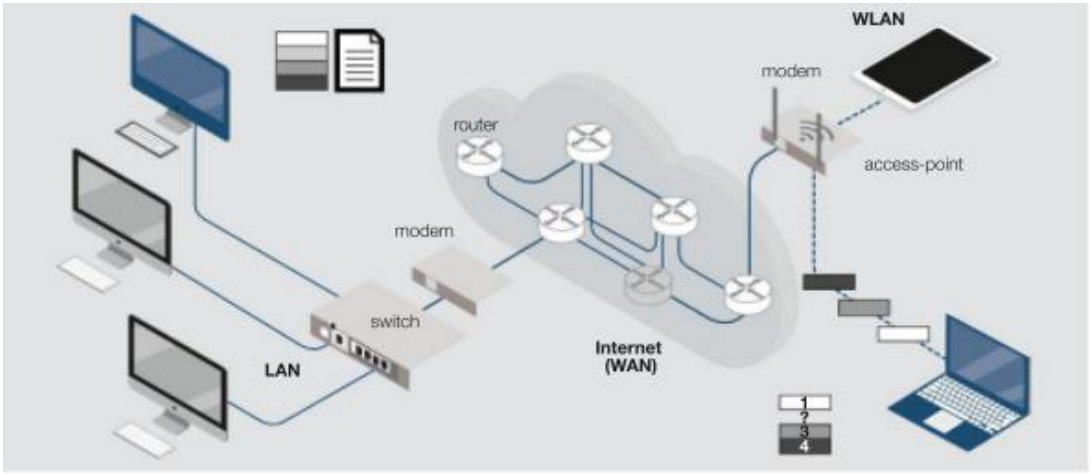
Secondo vantaggio **l'efficienza**. Nel caso in cui, a causa dei numerosi pacchetti da elaborare, un router si intasasse, rallentando di conseguenza la trasmissione, gli altri router se ne accorgerebbero e cercherebbero strade più veloci per l'instradamento dei pacchetti, garantendo comunque sempre la migliore velocità di trasmissione possibile.

Ecco le figure che mostrano lo smarrimento di un pacchetto e la sua ritrasmissione:

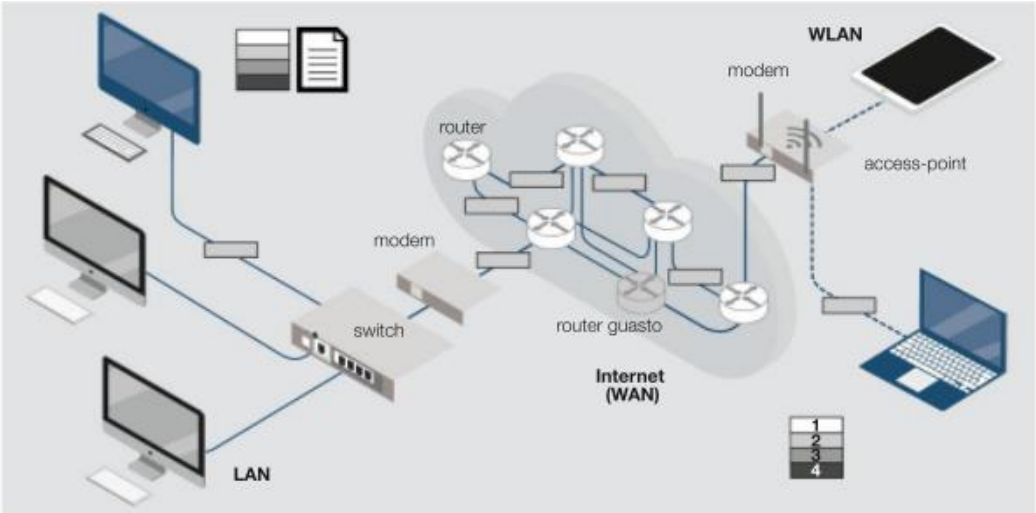
Il mittente invia un messaggio composto da 4 pacchetti ma un pacchetto viene perso per un malfunzionamento su un tratto di rete:



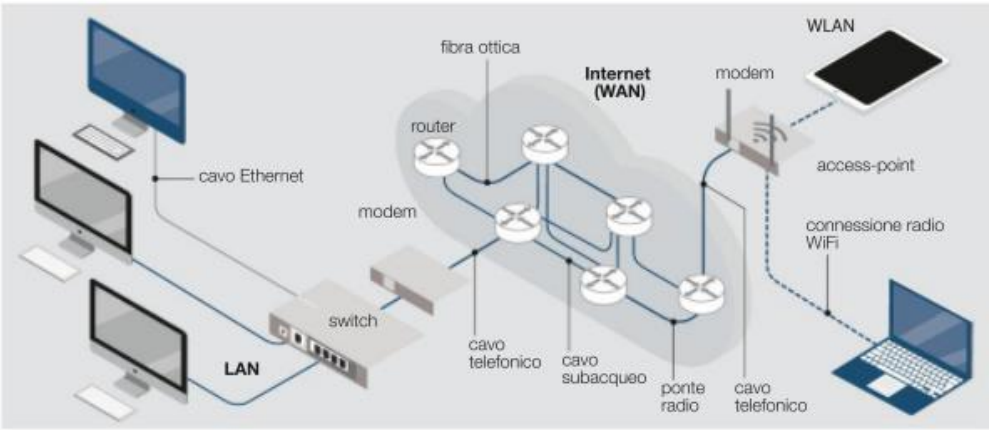
Il destinatario rileva la mancanza del pacchetto 2 e chiede al mittente di trasmetterlo nuovamente:



Il pacchetto 2 viene ritrasmesso seguendo un percorso alternativo a quello malfunzionante:



L’infrastruttura di comunicazione di internet è molto varia, è costituita da cavi interrati, fibre ottiche, cavi sottomarini, ponti radio, trasmissioni satellitari….



2. IL MODELLO OSI DELL' ISO

Non so se ci rendiamo conto che la rete Internet è una delle cose più complesse realizzate dall'uomo. In particolare si osservi che non esiste un proprietario di Internet. Non esiste qualcuno che "ha inventato Internet" dall'inizio alla fine. Il funzionamento di Internet si basa su tecnologie hardware e software di diversi produttori che devono comunicare fra di loro. Esistono diversi produttori di Switch, diversi produttori di schede di rete, esistono i proprietari di una linea in fibra ottica, o di un ponte radio, esistono diversi browser, ecc…., e tutti questi elementi devono collaborare fra loro e consentire la trasmissione di dati. Affinché questo sia possibile sono necessarie delle regole tecniche di comunicazione che tutti i produttori di dispositivi devono rispettare per far sì che tutti gli elementi possano comunicare fra loro. Tali regole sono dette **protocolli di rete**. Al plurale perché non ce n'è solo uno, ce ne sono diversi.

Un protocollo di rete è quindi un insieme di regole tecniche (o meglio "specifiche" tecniche) che devono essere rispettate nella realizzazione dei vari dispositivi hardware e software per consentire la comunicazione fra computer (o dispositivi) connessi attraverso una rete LAN o WAN.

Il modello ISO/OSI

Cosa è il modello ISO/OSI? E' un **modello standardizzato di un sistema di comunicazione**.

Negli anni '70 diverse aziende iniziarono a lavorare per cercare il modo di scambiare dati fra computer e creare così delle reti di computer. Ogni rete aveva la propria modalità per far comunicare i computer. Alla fine degli anni '70 l'**ISO (*International Organization for Standardization***) avviò un progetto per definire delle regole standard per la realizzazione delle reti di comunicazione in modo tale che tutti i dispositivi potessero comunicare fra loro. Questo progetto, è stato poi definito nel 1984 e ha preso il nome di **OSI** (***Open System Interconnection***).

La parola chiave è **modello** (standardizzato). Lo standard ISO/OSI è un modello, ossia **una rappresentazione astratta** di un sistema di comunicazione.

In questo modello, chiamato appunto ISO/OSI, la rappresentazione di un sistema di comunicazione, essendo molto complesso, è stata suddivisa in 7 strati disposti uno sopra l'altro, per questo è chiamato anche ***stack ISO/OSI*****,** in modo da poter definire *"chi deve fare cosa"* per realizzare il sistema di comunicazione fra due dispositivi, o meglio, fra due applicazioni in esecuzione su due distinti dispositivi. Per stabilire il *"come"* devono essere svolti i compiti di ciascun livello, per ogni livello si devono definire dei protocolli. L'ISO/OSI però non è un protocollo, è semplicemente un modello che rappresenta un generico sistema di comunicazione.

Gli strati dello stack ISO/OSI sono i seguenti (questi bisogna saperli a memoria):

7 Applicazione

6 Presentazione

5 Sessione

4 Trasporto

3 Rete

2 Data link

1 Fisico

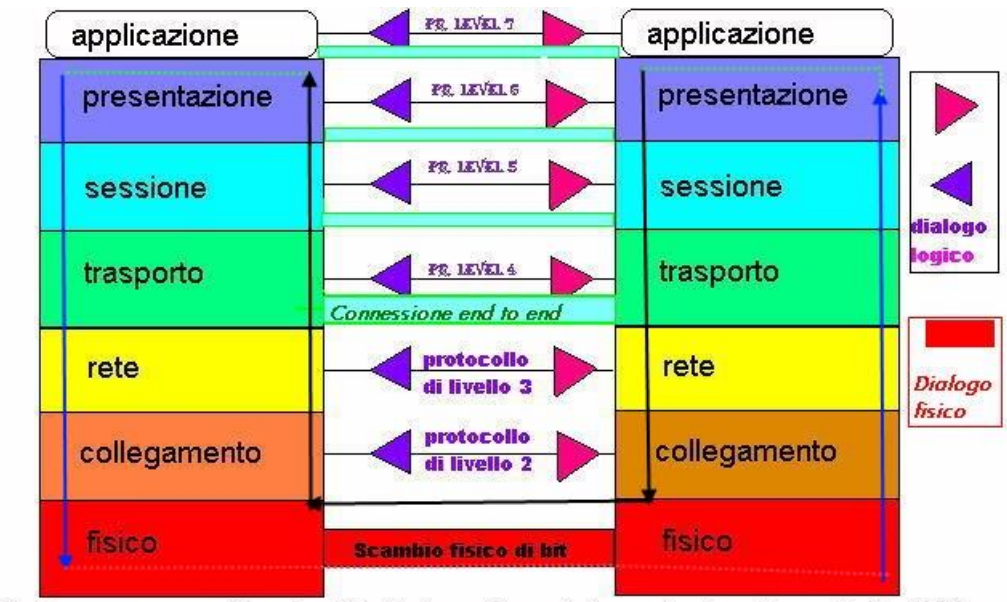
Ma cosa sono questi strati?

Ogni strato viene realizzato con appositi elementi hardware e software chiamati **entità**. Un esempio di entità hardware è lo switch (agisce a livello 2), oppure il router (agisce a livello 3). Un esempio di entità software è la coppia browser - web server (a livello di applicazione, livello 7).

I dati che devono essere trasmessi da un computer all'altro (ad esempio del testo da inviare via email), parte dal livello 7 del processo in esecuzione sull'host di partenza, attraversa tutto lo stack, raggiunge l'host di destinazione attraverso il canale fisico (i cavi, la fibra ottica, le onde radio…), ed arriva allo strato 7 dell'host di destinazione risalendo lo stack.

Ogni strato fornisce un **servizio** allo strato superiore, nel senso che gli fornisce delle funzioni da utilizzare per la trasmissione dei propri dati verso la destinazione.

Nella trasmissione dei dati, ogni strato non sa come funziona o come è realizzato lo strato inferiore (trasparenza), semplicemente gli passa dei dati (in trasmissione) o riceve dei dati (in ricezione) che poi questo a sua volta passerà allo strato inferiore (in trasmissione) o superiore (in ricezione) e così via… Questo concetto si esprime dicendo che ogni strato inferiore fornisce i propri servizi allo strato immediatamente superiore in maniera **trasparente**.



Schema concettuale di dialogo tra elaboratori nel modello OSI

Mentre i dati attraversano in verticale lo stack (in trasmissione lo stack è percorso dall'alto verso il basso, in ricezione dal basso verso l'alto), dal punto di vista **logico** ogni entità di un determinato strato (o livello) dell'host di partenza comunica con l'entità di equivalente livello dell'host di destinazione attraverso uno specifico protocollo.

Il protocollo stabilisce un insieme di regole che consente la comunicazione fra **entità** dello stesso livello.

Ma cosa stabiliscono queste famose regole che formano il protocollo? ogni protocollo stabilisce delle cose leggermente diverse, ma in generale ogni protocollo si occupa di definire:

- il formato dei messaggi
- l'ordine di invio e di ricezione degli stessi
- le azioni che le **entità** devono eseguire in seguito alla loro ricezione.

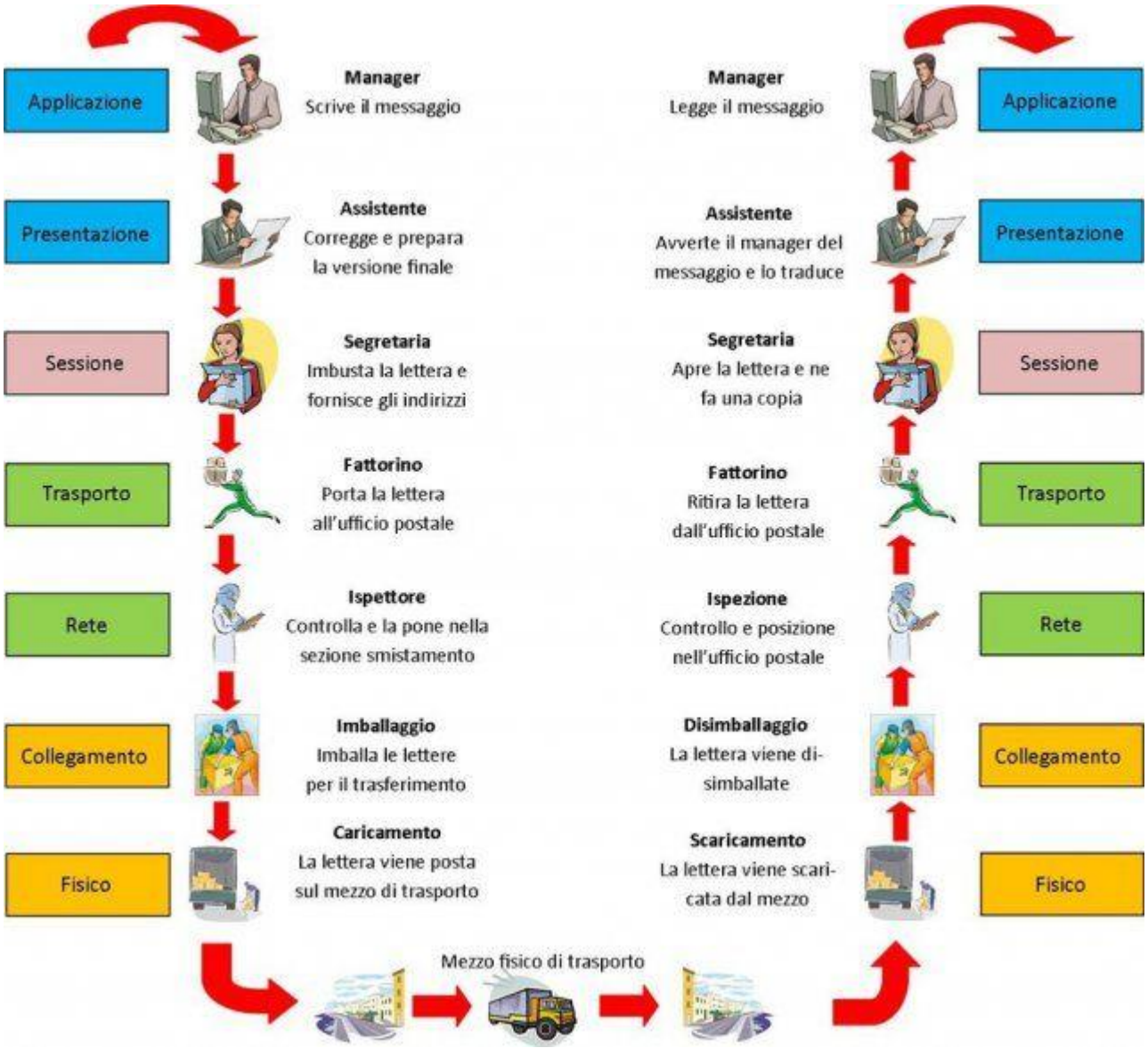
Quindi ad ogni livello dello stack sono definiti dei protocolli di comunicazione. Ci saranno dei protocolli a livello di applicazione (HTTP, SMTP, FTP…), dei protocolli a livello di trasporto (TCP, UDP), dei protocolli a livello di rete (IP) , dei protocolli a livelli inferiori (IEEE 802.3, IEEE 802.11)

Infatti, un router, dal punto di vista logico, comunica con un altro router inviandogli dei messaggi, utilizzando un apposito protocollo (il protocollo IP), ma dal punto di vista fisico i dati che un router invia ad un altro router sono dei segnali elettrici che devono necessariamente percorrere lo stack fino al livello fisico (livello 1), raggiungere il router di destinazione e risalire lo stack fino al livello 3 (rete).

La seguente figura rappresenta una classica metafora per comprendere la suddivisione dei compiti in livelli di un sistema di comunicazione. La metafora è quella del sistema postale.

Nella metafora si evidenziano alcune cose:

- che ogni livello si limita a svolgere una attività e poi passa "dei dati" al livello successivo che si occupa di un'altra attività. Ad esempio il manager scrive la lettera e poi non si interessa di quello che accadrà successivamente, egli sa che poi la lettera arriverà al manager destinatario.
- gli stessi livelli applicano delle regole condivise nello svolgimento delle operazioni (i protocolli). Ad esempio le segretarie sanno che la lettera sarà sempre all'interno di una busta, che il mittente sarà indicato sul fronte della busta e il destinatario sul retro.
- Il livello piu' basso prevede il trasposto fisico dei dati.



Tornando al modello ISO/OSI quindi: ogni livello ha un compito. Il modo con cui realizza tale compito è definito dal protocollo (in alcuni casi **da più** protocolli) utilizzato da quel livello.

Vediamo, in maniera molto riassuntiva, quale è il principale compito svolto dalle entità a ciascun livello dello stack ISO/OSI

LIVELLO1 FISICO

Un protocollo a questo livello si occupa di stabilire la forma, la durata, i valori di tensione dei segnali elettrici che rappresentano i bit da trasmettere sul mezzo di comunicazione, e di determinare le caratteristiche fisiche dei componenti (cavi, dei connettori, antenne ecc..).

LIVELLO 2 DATA LINK

Un protocollo a questo livello si occupa di rendere affidabile **la linea di trasmissione** che connette fra loro due dispositivi (entità hardware) direttamente connessi.

Fra due host il percorso è costituito da molti dispositivi collegati fisicamente fra loro, si dice "**punto punto**" (host – switch – router – router – router - ------ - switch – host). Le entità a livello 2 si occupano di far sì che un pacchetto di dati, da un dispositivo, raggiunga il dispositivo successivo ad esso fisicamente collegato.

I pacchetti, a questo livello, si chiamano **frame**.

Fra i dispositivi che operano a questo livello ci sono gli **switch** e gli **access point**.

LIVELLO 3 RETE

Le principali funzioni svolte dalle entità hardware e software a livello di rete sono:

1. **instradamento**
2. **controllo della congestione**.

L'instradamento (routing): consiste nello scegliere il percorso migliore che consente ad un pacchetto che parte dall' host di arrivare all'host di destinazione.

Controllo della congestione: consiste nel trovare un *percorso alternativo* ai pacchetti quando un router deve gestire un numero eccessivo di pacchetti e quindi rallenta la trasmissione.

I dispositivi hardware che operano a livello 3 sono dunque i **router**. I pacchetti, a questo livello sono chiamati **datagrammi**.

I protocolli a questo livello stabiliscono, fra le altre cose, come devono essere fatti i pacchetti in modo tale che il software contenuto nei router sia in grado di estrarre l'indirizzo di destinazione del pacchetto e in base a quello determini a quale router inviare il pacchetto fino a giungere all'host di destinazione.

LIVELLO 4 TRASPORTO

Questo livello viene implementato solamente sugli **end system** della comunicazione, ossia solo sull' host nel quale è in esecuzione l'applicazione che trasmette i dati e sull' host nel quale è in esecuzione l'applicazione che riceve i messaggi, quindi non è implementato sui router ma solamente su *computer* di *partenza* e di arrivo dei *pacchetti*. Per questo si dice che i protocolli a livello 4 sono protocolli **end to end.**

Le funzioni principali di questo livello (i protocolli stabiliscono come realizzare tali funzionalità) sono due:

1. **stabilire una connessione fa i due host che devono scambiarsi i dati**
2. **svolgere la multiplazione e la demultiplazione dei pacchetti**.

Cosa significa **stabilire una connessione fra due host**? Significa realizzare un canale affidabile di comunicazione, vale a dire realizzare un software che consenta di inviare pacchetti dall'host 1 all' host2, e se i pacchetti non arrivano, l'host 2 chiede di inviarli nuovamente. A livello di trasporto, una volta stabilita una connessione fra due host, ci si può disinteressare di tutto quello che avviene ai livelli sottostanti (i router che si passano i pacchetti, la trasmissione fisica….). Si può immaginare che, una volta stabilita la connessione, vi sia un canale virtuale che collega direttamente i due host di partenza e destinazione.

I sistemi operativi dei PC mettono a disposizione delle API che costituiscono l'interfaccia del livello di trasporto (e che noi impareremo ad usare). Interfaccia nel senso di "funzioni" alle quali basterà indicare come parametri un identificativo del'host di destinazione (che sarà il suo indirizzo IP), l'applicazione di destinazione (che sarà il numero di porta logica dell'applicazione), i dati da inviare, e la API si occuperà di trasmettere il pacchetto a destinazione.

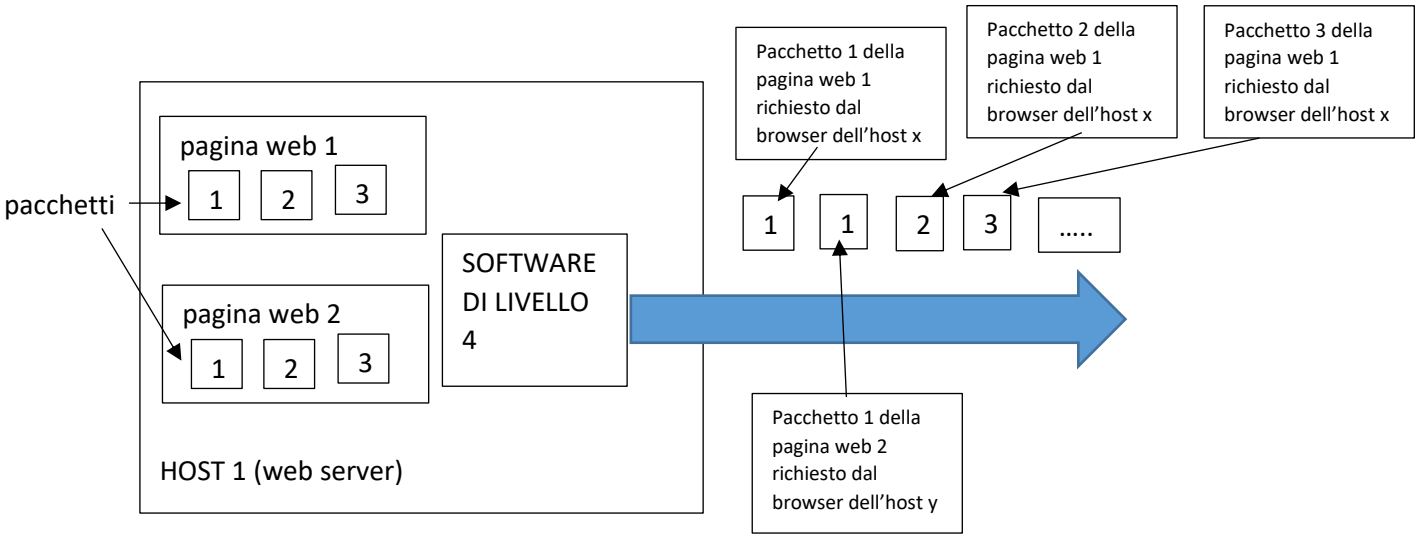
Le entità di livello 4 sono queste API.

A livello 4 i pacchetti sono chiamati in alcuni casi **segmenti** e in altri **datagrammi**.

Cosa significa svolgere la multiplazione/demultiplazione? Ogni scambio di dati fra host si riferisce sempre ad una specifica applicazione (ad esempio la visualizzazione di una pagina web, oppure una comunicazione su Teams ecc..). Ad esempio, quando un host invia pacchetti verso uno o più altri host, è sempre necessario identificare non solo l'host di destinazione ma anche l'applicazione alla quale vanno inviati i pacchetti. Analogamente, quando un host riceve pacchetti da diversi altri host è necessario che l'host ricevente "sappia" non solo da quale host arrivano i dati ma anche a quale delle proprie applicazioni sono destinati. Con il termine multiplazione/demultiplazione si intende l'identificazione della corretta coppia host/applicazione dei pacchetti. Di questo si occupano le entità a livello 4. Il seguente esempio dovrebbe chiarire:

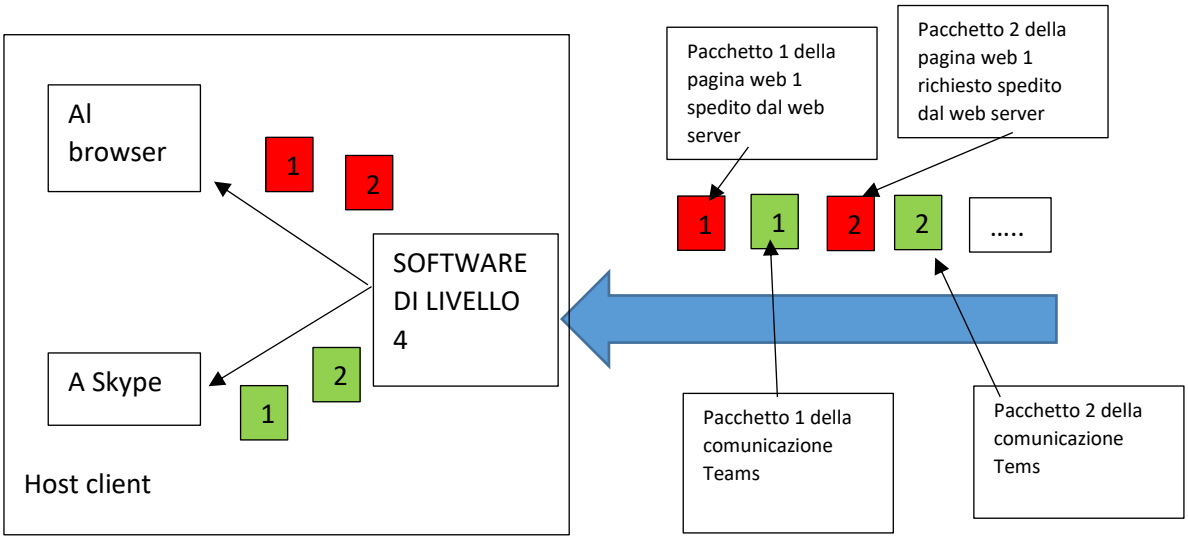
Un web server che ospita diversi siti web dovrà trasmettere pagine web diverse a diversi host client che le hanno richieste, i software di livello 4 del web server si occuperanno di **inserire nei pacchetti le informazioni** per identificare a quale coppia host-applicazione devono essere inviati i singoli pacchetti:

Esempio di multiplazione:



Allo stesso modo, quando un Host che riceve pacchetti da due diverse applicazioni, ad esempio pacchetti di una pagina web richiesta a un web server, e pacchetti di una comunicazione su Teams con un altro Host, i software di livello 4 identificano i pacchetti e li inviano alle relative applicazioni

Esempio demultiplazione:



LIVELLO 5 SESSIONE

Gestisce la sessione di comunicazione fra processi in esecuzione fra due host della rete. Ad esempio attraverso opportuni punti di sincronizzazione si occupa di ripristinare una comunicazione interrotta per un malfunzionamento dallo stesso punto in cui si era interrotta.

LIVELLO 6 PRESENTAZIONE

A questo livello i protocolli si occupano di stabilire un **formato comune** per la rappresentazione dei dati fra i due host in comunicazione, ad esempio indicare quale codifica dovrà essere utilizzare per trasmettere dei caratteri (UTF-8, UTF-16….) o le immagini (JPEG, BMP..). Inoltre i protocolli a questo livello stabiliscono eventuali compressioni e crittografia dei dati trasmessi. Ad esempio il protocollo TLS (Transport Layer Security) è un protocollo di sicurezza utilizzato nelle trasmissioni sicure (http+tls→https)

LIVELLO 7 APPLICAZIONE

I protocolli a questo livello stabiliscono le regole per la comunicazione fra due processi in esecuzione sulla rete. Ad esempio, stabiliscono **come deve essere scritto un messaggio** (la sintassi, i comandi…) che un browser deve spedire ad un web server affinché esso risponda con l'invio di una determinata pagina web, e stabiliscono il come deve essere il messaggio di risposta.

3. LA INTERNET PROTOCOL SUITE (CHIAMATA ANCHE SUITE TCP/IP)

Abbiamo visto che il modello ISO/OSI prevede che per ogni livello siano definiti opportuni protocolli che svolgono funzioni diverse. Vediamo ora alcuni di questi protocolli.

Quali sono i protocolli più utilizzati per implementare i 7 livelli dello stack ISO/OSI? Sono quelli previsti dalla **Internet Protocol Suite**, chiamata anche, imprecisamente, **protocollo TCP/IP** ("imprecisamente" perché in realtà tale denominazione non indica **UN** protocollo ma un **INSIEME** di protocolli. Per questo è detta suite TCP/IP).

La suite TCP/IP è un insieme di protocolli che si è imposto nella realizzazione delle reti di comunicazione come standard *de facto*.

I documenti con le specifiche tecniche dei vari protocolli della suite TCP/IP sono documenti pubblici chiamati **RFCs** (*Request For Comments*) e pubblicati dalla **IETF** (Internet Engeneering Task Force), un'associazione internazionale di tecnici che si occupa della ricerca e dell'evoluzione di Internet. Qui si può trovare un esempio di RFC, è un tutorial che spiega i passaggi per l'inoltro di un datagramma IP attraverso un router: https://datatracker.ietf.org/doc/html/rfc1180

La suite TCP/IP prevede l'impiego di diversi protocolli ai diversi livelli dello stack ISO/OSI ma non a tutti i livelli, infatti nel TCP/IP i livelli 5,6,7 sono visti come un solo livello, per il quale lo standard TCP/IP non definisce alcun protocollo. I protocolli utilizzati a questo macro livello applicativo dipendono dal tipo di operazione che l'utente vuole eseguire sulla rete (se vuole ottenere una pagina web userà il protocollo HTTP, se vuole trasferire un file userà FTP, e così via..).

Inoltre la suite TCP/IP aggrega anche i livelli 1 e 2 e non definisce alcuno specifico protocollo per tale livello (livello data link). Generalmente i protocolli utilizzati sono IEEE 802.3 per le reti LAN cablate e 802.11 per le reti LAN wireless, PPP per le WAN.

La seguente tabella mostra lo Stack del protocollo TCP/IP con l'elenco dei protocolli più utilizzati

TABELLA 3

Livello	Protocollo
7. Applicazione 6. Presentazione 5. Sessione	**HTTP** (*Hyper-Text Transfer Protocol*) **SMTP** (*Simple Mail Transfer Protocol*) **POP** (*Post Office Protocol*) **FTP** (*File Transfer Protocol*) ...
4. Trasporto	**TCP** (*Transmission Control Protocol*) **UDP** (*User Datagram Protocol*)
3. Rete	**IP** (*Internet Protocol*)
2. *Data-link*	**IEEE-802.3** (*wired Ethernet*) **IEEE-802.11** (*wireless Ethernet*) **PPP** (*Point-to-Point Protocol*) **HDLC** (*High-level Data-Link Control*) ...

Per quanto riguarda i livelli 3 e 4 i protocolli utilizzati per tutte le reti sono sempre gli stessi:

- protocollo **IP** (*Internet Protocol*) per il livello 3

- protocolli **TCP** (*Tramsmission Control protocol*) oppure **UDP** (*User Data Protocol*) per il livello 4

Nella suiteTCP/IP abbiamo dunque solamente 4 livelli rispetto ai 7 del modello ISO/OSI.

Ma cerchiamo di capire come funzionano questi protocolli.

Il concetto principale da comprendere è quello dell'**imbustamento** o **incapsulamento** dei pacchetti.

Come detto, a livello di applicazione i dati da trasmettere vengono scomposti in pacchetti.

Ogni pacchetto di dati viene "*passato*" al livello sottostante nel quale il protocollo prevede che vengano aggiunte al pacchetto ricevuto delle informazioni necessarie allo svolgimento del compito previsto da quel livello.

Il pacchetto inoltre viene indicato con nomi diversi nei vari livelli. Un generico pacchetto viene indicato con il termine PDU (Protocol Data Unit) ma viene chiamato:

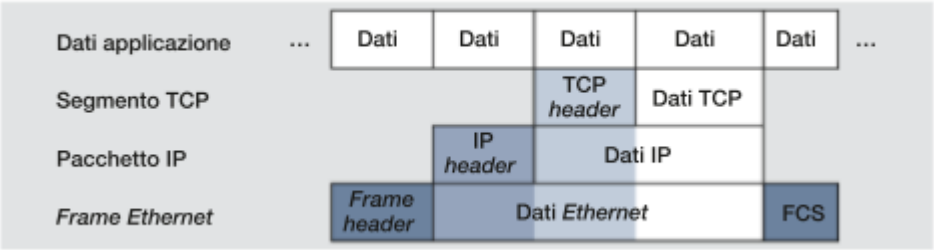
- a livello 4: **segmento** nel protocollo TCP – **datagramma** nel protocollo UDP
- a livello 3**: datagramma** nel protocollo IP
- a livello 1 e 2**: frame** nel protocollo Ethernet (è il protocollo più utilizzato a livello 1,2)

Il pacchetto contenente i dati da trasmettere aumenta di dimensioni (di byte) man mano che scende di livello, quando raggiunge il livello più basso, il pacchetto viene trasmesso sul canale fisico e raggiunge l'host di destinazione. Qui il pacchetto risale lo stack TCP/IP e ogni entità estrae dal pacchetto ricevuto solo le informazioni ad essa necessarie. Questo passaggio di pacchetti avviene dal livello più basso verso l'alto e si ripete fino al livello applicativo, dove i dati costituiscono il messaggio originale che era stato spedito.

Ma cosa sono le informazioni che ogni entità aggiunge al pacchetto che riceve dal livello superiore? Tale informazioni aggiuntive sono dette **header** (intestazione) e, ad esempio, per il livello 3 sono gli indirizzi IP del mittente e del destinatario del pacchetto. La parte di dati che viene incapsulata da ciascun protocollo viene anche indicata con il termine di **payload**. Il PDU di un livello diventa il payload del livello sottostante.

Nella seguente figura viene mostrata la tecnica dell'incapsulamento:

- I dati dell'applicazione vengono suddivisi in pacchetti
- Ogni pacchetto viene passato al livello di trasporto dove il protocollo TCP, oppure UDP, prevede che venga aggiunto il proprio header che contiene, fra le alte cose, un numero che identifica se il protocollo di trasporto è TCP o UDP. Il pacchetto così ottenuto (**segmento o datagramma**) viene passato al livello di rete.
- A livello di rete il protocollo IP aggiunge il proprio header che contiene gli indirizzi IP di chi trasmette e del destinatario. Il pacchetto così ottenuto (**datagramma**) viene passato al livello di data link
- A livello di data link il protocollo (solitamente Ethernet oppure PPP) aggiunge il proprio header che contiene, ad esempio, il FCS (Frame checksum), i 2 indirizzi fisici (MAC Address) e altre informazioni. Questo pacchetto, il **frame**, è una sequenza di bit che vengono trasmessi sul canale fisico.



Prima di raggiungere l'host di destinazione, ogni volta che raggiunge un router, il frame viene elaborato in modo che venga estratto il datagramma IP. Dal pacchetto IP il router può estrarre l'indirizzo IP di destinazione e sapere a quale altro router inviare il datagramma sulla base delle proprie tabelle di instradamento, tale datagramma sarà dunque incapsulato in un altro frame e spedito al router successivo.

4. IL LIVELLO DI TRASPORTO E I PROTOCOLLI TCP E UDP

Quando uno sviluppatore realizza delle applicazioni software che devono comunicare in rete può scegliere, in base alle proprie esigenze uno fra questi due protocolli a livello di trasporto:

UDP (User Datagram Protocol)

TCP (Trasmission Control Protocol)

Per capire quando è opportuno usare uno e quando l'altro analizziamoli uno alla volta. Prima di fare questo è però necessario introdurre due concetti, il concetto di **connessione** e il concetto di **porta logica**.

COSA E' UNA CONNESSIONE

La fase più delicata dei protocolli di trasporto è quella che porta a stabilire un connessione. Una connessione è un canale affidabile per la comunicazione. Quando è stabilita una connessione, se l'host di partenza invia un pacchetto, è certo che esso arriverà a destinazione (l'host destinazione deve rispondere "Ok ricevuto!"). E se il pacchetto non arriva? L'host di partenza lo invia nuovamente! Vedremo che solo TCP stabilisce una connessione fra i due host, UDP non lo fa.

COSA E' UNA PORTA LOGICA

Supponiamo che un Host A stia scambiando dati con due host diversi, ad esempio sta ricevendo dei pacchetti da un web server relativi ad una pagina web e nello stesso tempo sta ricevendo dei pacchetti relativi ad una comunicazione con Teams da un altro host.

Quando i pacchetti che contengono i dati arrivano all' Host A, è necessario che i pacchetti relativi alla pagina web vengano inviati al processo in esecuzione "*Browser*" e i pacchetti relativi alla comunicazione vocale vengano inviati al processo in esecuzione dell'applicazione "*Teams"*. Per "riconoscere" il processo a cui sono destinati i pacchetti (e analogamente il processo che ha generato i pacchetti negli host che li hanno inviati), il protocollo di trasporto dell'host trasmettitore, aggiunge ai pacchetti che riceve dal livello di applicazione un header, che contiene, fra altri dati, due numeri a 16 bit (da 0 a 65535) che identificano il processo sorgente e il processo destinazione.

Ecco una delle cose più importanti che fa il protocollo di trasporto! Tali due numeri sono chiamati **porte logiche** (o spesso anche semplicemente **porte**).

Possiamo quindi dire che una porta è un numero a 16 bit che indentifica un processo in esecuzione.

Analizziamo ora le differenze fra i due protocolli di trasporto TCP e UDP.

UDP

Avevamo detto che il livello di trasporto ha il compito di realizzare una connessione fra due host. Bene, il protocollo UDP in realtà non lo fa, è un protocollo cosiddetto **connectionless** (senza connessione), infatti **non è affidabile** perché non garantisce che i suoi pacchetti arrivino a destinazione e non garantisce che i pacchetti arrivino in ordine. Se il programmatore vuole garantire l'arrivo dei pacchetti e l'ordine, dovrà implementare tali caratteristiche a livello di applicazione. Un protocollo non affidabile (come sono IP e UDP) si dice che implementa un comportamento ***best-effort***, ossia, fa del suo meglio ma non garantisce nulla.

I suoi pacchetti sono chiamati infatti datagrammi, come quelli del protocollo IP, anche lui non affidabile. Allora quale è il vantaggio?

Il vantaggio è che è semplice e veloce, adatto per trasferimento di dati altrettanto semplici.

Infatti è utilizzato per applicazioni in cui la velocità di trasmissione è "più importante" rispetto all'affidabilità, oppure in cui i dati scambiati sono pochi (es. **DHCP**, audio e video real time, **DNS** per la semplicità dei dati scambiati).

Se ci pensiamo, durante una videoconferenza è più importante che non ci siano ritardi nella comunicazione piuttosto che la qualità perfetta dell'immagine.

Esempio di comunicazione UDP:

quando un host A deve comunicare con un host B, A deve indicare, nel pacchetto la porta sulla quale intende ricevere i datagrammi e la porta dell'host destinazione verso la quale intende trasmettere i datagrammi.

L' Host B a sua volta trasmetterà verso la porta di ricezione dell'Host A e riceverà verso la propria porta di ricezione.

In questo modo A e B si scambiano dati ma non è certo che i dati arrivino, dovrà essere l'applicazione a decidere cosa fare quando non arrivano i datagrammi o arrivano sbagliati (per esempio, aspettare il datagramma successivo).

Per la sua semplicità ed assenza di connessione UDP è detto anche protocollo *fire and forget* (spara e dimentica).

E' un protocollo peer to peer (P2P) dove, quindi, entrambi i lati della comunicazione possono inviare e ricevere dati semplicemente conoscendo Indirizzo IP e Porta del destinatario, i dispositivi che partecipano alla comunicazione hanno entrambi lo stesso ruolo (trasmettitori e ricevitori entrambi, quindi non client server).

Esempio:

l'host A (IP 201.33.33.22) trasmette, indicando come **propria porta di ricezione** la porta 2000, verso la porta 3000 dell'host B (IP 211.77.5.44)

L'host B riceve sulla propria porta 3000 e trasmette sulla porta 2000 dell' host A.

La comunicazione fra i due processi in esecuzione sui due host può avvenire perché sono noti i due indirizzi IP (gestiti dal livello 3, rete, ISO/OSI) che identificano gli host e i due numeri di porta che identificano i processi sugli host (gestiti dal livello 4, trasporto, del modello ISO/OSI). Ogni datagramma deve quini contenere:

- **IP e porta di ricezione del destinatario**
- **IP e porta di ricezione di chi spedisce il pacchetto.**

TCP

E' un protocollo affidabile, infatti **garantisce la trasmissione corretta e ordinata dei suoi pacchetti** (chiamati **segmenti**), liberando da questo compito il programmatore che scrive l'applicazione. L'affidabilità va a discapito della velocità, infatti il protocollo viene utilizzato nelle applicazioni in cui l'affidabilità è più importante della velocità. Ad esempio nel trasferimento delle pagine web (protocollo HTTP a livello di applicazione), se non arriva un tag di chiusura il browser potrebbe non visualizzare correttamente la pagina, oppure nel trasferimento di file (protocollo FTP), se mancano dei byte il file ricevuto è corrotto.

Gli host che partecipano alla trasmissione hanno ruoli completamente diversi, uno è il client che (richiede la connessione) e l'altro è il server (che accetta la connessione)

Esempio di connessione TCP

1. host 1 (client) invia una richiesta di connessione ad host 2 (server)
2. host 2 risponde ad host 1 che accetta la richiesta
3. host 1 comunica a host 2 che la connessione è stabilita. (e se questa comunicazione non arriva all' host 2?....la connessione non è stabilita!). Questa ultima comunicazione serve all'host B per stimare il timeout della connessione, ossia per sapere *"dopo quanto tempo*" un pacchetto da esso inviato può considerarsi eventualmente "*perso*".

Ora host1 e host2 hanno una connessione affidabile attraverso cui scambiarsi i pacchetti, diversamente da UDP, il protocollo TCP, ogni volta che un pacchetto viene inviato (dal server), l'host che lo riceve (client) risponde con un acknowlegment (che significa "OK ricevuto!"). Se il server non riceve l' acknowledgment sia perché si è perso nella rete il pacchetto (e quindi il client non l'ha mai ricevuto), sia perché si è perso nella rete l'acknowledgement, il server provvederà, dopo un certo tempo a ritrasmettere il pacchetto. Se il client ha già ricevuto tale pacchetto, lo scarterà e invierà l'acknowledgment.

ANCORA QUALCOSA SULLE PORTE

Come abbiamo visto, per iniziare una comunicazione UDP o per richiedere una connessione TCP, un host deve inviare un pacchetto in cui indicare il numero di porta del processo che invierà la risposta.

Per applicazioni che utilizzano alcuni protocolli a livello applicativo molto diffusi, il numero di porta è stato standardizzato e quindi riservato a tali protocolli. Tali numeri di porta sono noti con il nome di *well know* port:

Protocollo applicativo	Numero di porta
FTP (*File Transfer Protocol*)	20, 21
SMTP (*Simple Mail Transfer Protocol*)	25
DNS (*Domain Name Server*)	53
HTTP (*Hyper-text Transfer Protocol*)	80
POP (*Post Office Protocol*)	110
NTP (*Network Time Protocol*)	123
HTTPS (*Hyper-text Transfer Protocol Secure*)	443
Doom (*videogame*)	666
FTPS (*File Transfer Protocol Secure*)	989, 990

I protocolli sicuri HTTPS e FTPS sono sempre gli HTTP e FTP ma implementano un ulteriore protocollo (TLS) a livello di presentazione che si occupa di crittografare i pacchetti.

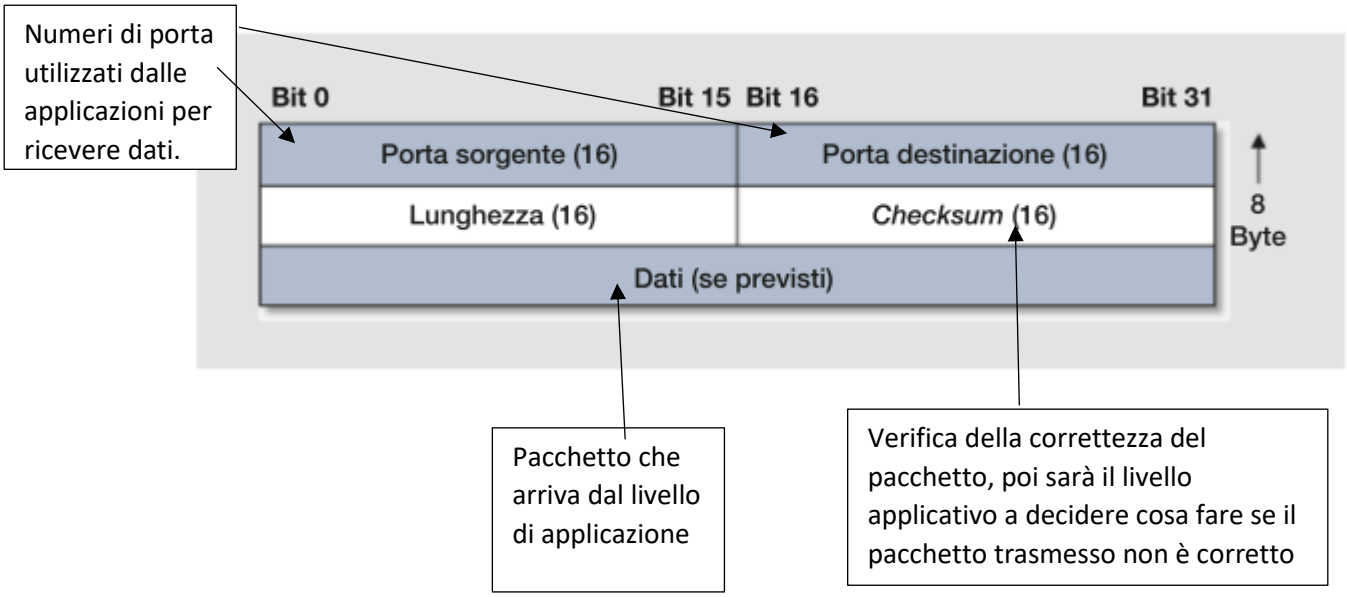
Molti numeri di porta compresi fra 1024 e 49151 sono stati registrati dai produttori software per le proprie applicazioni. Se una applicazione sta utilizzando un numero di porta, tale numero non può essere utilizzato da altre applicazioni.

Il protocollo TCP svolge anche l'attività di **controllo della congestione** anche se tale attività, secondo quanto stabilito dallo stack ISO/OSI dovrebbe essere svolta a livello di rete. Ogni volta che un pacchetto TCP è ricevuto dall'host di destinazione, quest'ultimo invia un pacchetto di ACK (Acknowlege). Se il tempo che passa fra quando l'host sorgente trasmette il pacchetto e quando riceve l' ACK supera un certo valore, l'host sorgente rallenta la trasmissione.

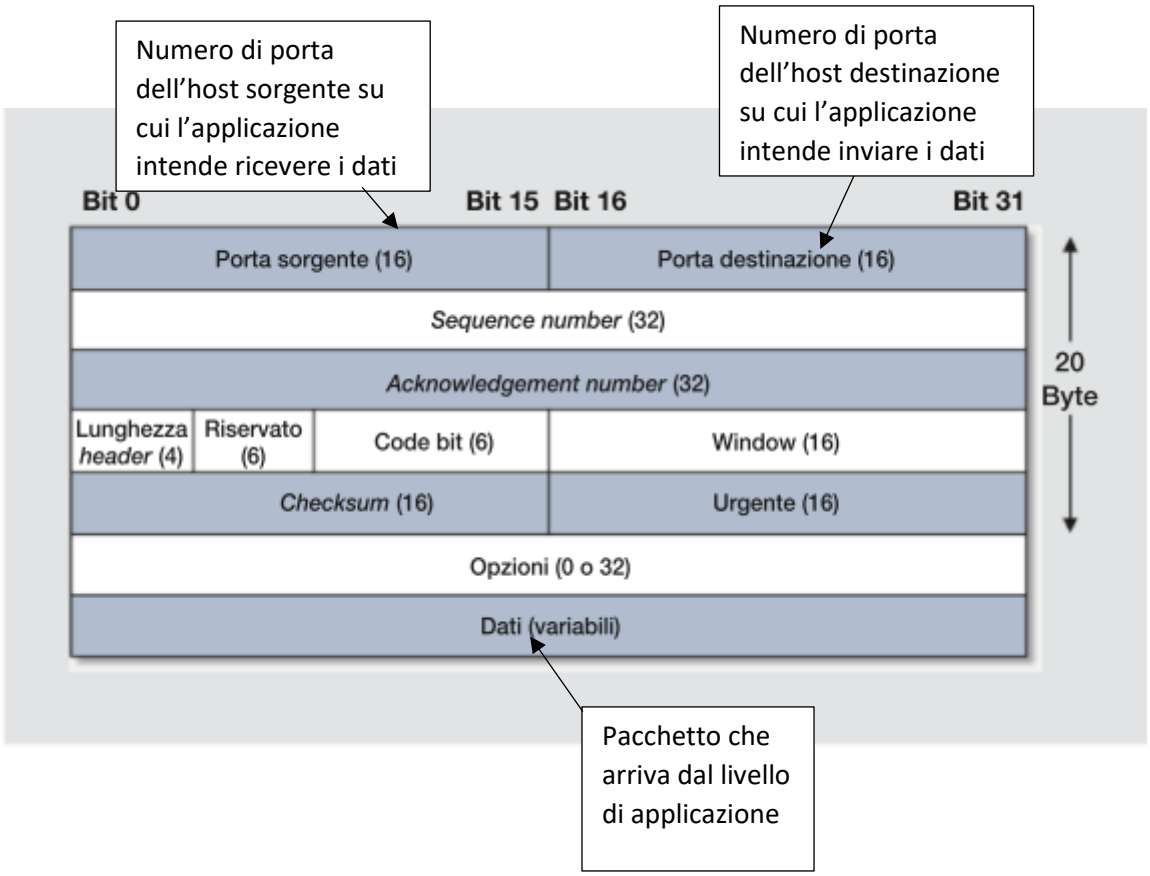
I PACCHETTI (PDU) DEI DUE PROTOCOLLI

Osservando la composizione dei PDU dei due protocolli si può immediatamente rilevare una maggior semplicità nei datagrammi UDP rispetto ai segmenti TCP.

Datagramma UDP



Segmento TCP



Sequence number indica il numero di byte inviati dall'inizio della connessione

Acknowlege number indica il numero di byte ricevuti dall' inizio della connessione

Questi campi vengono utilizzati dal protocollo per verificare che tutti i pacchetti siano stati trasmessi e ricevuti.

5. IL MODELLO CLIENT SERVER E IL PROTOCOLLO APPPLICATIVO HTTP

CLIENT-SERVER

Molti protocolli di livello applicativo che utilizzano TCP come protocollo a livello trasporto implementano una modalità di comunicazione chiamata **client-server**.

Secondo questa modalità di comunicazione, i due host che si scambiano dati svolgono ciascuno un proprio ruolo preciso. Per questo la modalità di comunicazione client/server è definita come asimmetrica (i due host fanno cose diverse, i loro ruoli non sono interscambiabili).

Il **server è un processo** in esecuzione su un host, tale processo è chiamato **servizio**, ad esso è associato un numero di porta noto, tale processo rimane in ascolto di eventuali richieste.

Il **client è un processo** in esecuzione su un host che effettua una **richiesta** di una **risorsa** al processo in esecuzione sul server. In questa richiesta verrà specificato l'indirizzo IP del server e il numero di porta (noto) relativo al servizio in esecuzione sul server. Nella richiesta del client verrà indicato (al server) anche il numero di porta che il processo client utilizzerà per ricevere i dati (quindi il numero di porta sul quale il server dovrà inviare i dati). Non è necessario che questo numero di porta sia sempre lo stesso per una determinata applicazione, anzi, spesso cambia, per questo è indicato anche con il termine di ***porta effimera***. Ad esempio, se un browser volesse visualizzare su due finestre la stessa pagina web richiesta allo stesso web server, dovrebbe effettuare due richieste. Ciascuna richiesta andrebbe inviata alla porta 80 (o 443) del web server (*destination port*=80) ma la *source port effimera*(la porta su cui il client riceve i dati), sarebbe diversa per le due richieste.

Ecco l'esempio di una comunicazione client server relativa alla richiesta di una pagina web da parte di un browser

ESEMPIO I siti web sono ospitati su computer server che, sulla porta 80 o 443, ricevono dai *browser* in esecuzione su computer o dispositivi client le richieste per singole pagine web che costituiscono il sito e alle quali rispondono inviando il contenuto delle pagine richieste. Come illustrato in FIGURA **18**, il protocollo applicativo HTTP stabilisce le regole di comunicazione per la richiesta e l'invio delle pagine web.

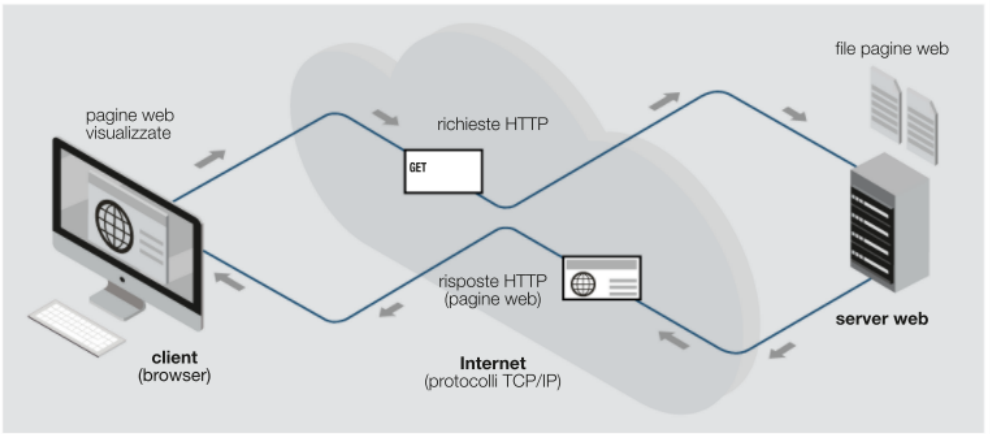


FIGURA **18**

LA NASCITA DEL PROTOCOLO HTTP

Come già detto, fino agli anni ’80 la WAN che è poi è diventata internet, connetteva fra di loro i computer di università e centri di ricerca. La rete veniva utilizzata da scienziati per trasmettersi email e documenti.

All’epoca le pagine web non esistevano.

All’inizio degli anni ’90 fu lo scienziato **Tim Berners-Lee**, fisico e informatico inglese presso il CERN di Ginevra, a ideare un protocollo a livello applicativo (**HTTP**) e un linguaggio di markup (**HTML**) per facilitare la lettura e la condivisione degli articoli scientifici. L’idea fu quella di realizzare un protocollo applicativo che consentisse facilmente la visualizzazione di articoli scientifici sotto forma di ipertesti, in modo che quando un articolo facesse riferimento ad un altro articolo, fosse semplice (con un click) visualizzare questo altro articolo.

Per questo motivo il protocollo fu chiamato **HTTP** (*Hyper Text Transfer Protocol*) e il linguaggio per la scrittura delle pagine web **HTML** (*Hyper Text Markup Language*).

L’insieme delle pagine web diffuse nei web server di tutto il mondo è stata denominata sin dalla sua nascita **World Wide Web** (indicata con **WWW**).

LE CARATTERISTICHE DEL PROTOCOLLO HTTP

Il protocollo prevede che il client effettui delle richieste al client, la richiesta è un messaggio chiamato **request.** Il server rimane in ascolto di eventuali request e, quando la riceve e può soddisfare la richiesta, risponde con un messaggio chiamato **response**. Le richieste specificano delle operazioni da svolgere su una risorsa, la risorsa può essere di qualsiasi tipo, ad esempio il codice di una pagina web, un’immagine, un file, i dati di un database…. ed è identificata da identificata da una stringa chiamata **URL** (Uniform Resource Locator), ad esempio: www.olivelliputelli.edu.it.

L’operazione che il client chiede di svolgere è generalmente una di queste:

- creare
- richiedere (leggere)
- aggiornare
- eliminare

la risorsa. Queste quattro operazioni sono chiamate operazioni **CRUD**: Create, Read, Update, Delete.

Il protocollo HTTP è un protocollo **stateless** (senza stato), il che significa che non mantiene memoria delle richieste effettuate durante la comunicazione fra client e server, quindi ogni richiesta è indipendente dalle richieste che l’hanno preceduta. Esistono dei meccanismi, che vedremo successivamente , per mantenere lo stato di una comunicazione, chiamate sessioni e cookies.

HTTP utilizza come protocollo a livello di trasporto il protocollo TCP.

Il protocollo http utilizza la well know port TCP 80. Per rendere più sicuro il trasferimento di dati fra client e server è stato aggiunto al protocollo http il protocollo **TLS** (Transport Layer Security). L’unione di http +TLS viene indicata come protocollo HTTPS e utilizza la well know port **TCP 443**.

L'EVOLUZIONE DEL PROTOCOLLO HTTP

Nel tempo vi sono state diverse versioni del protocollo HTTP:

1. **HTTP 0.9**
    a. il client poteva solamente effettuare delle richieste (GET) di risorse
    b. le risorse erano solamente pagine html
    c. la connessione TCP utilizzata era **non persistente**. Questo significa che, dopo aver richiesto una risorsa ad un server (ad esempio il codice html di una pagina web) la connessione TCP veniva chiusa.

2. **HTTP 1.0 (1991)**
    a. E' stata introdotta la possibilità di svolgere altre operazioni sulle risorse (aggiungere risorse, modificarle, eliminarle)
    b. Il protocollo è stato modificato per consentire di trasmettere qualunque tipo di risorsa (immagini, video, audio file ecc..)
    Il fatto che la connessione rimanga **non persistente** crea un problema poiché le nuove pagine web contengono immagini ecc. La non persistenza della connessione TCP fa sì che, dopo aver richiesto il codice html di una pagina web la connessione TCP viene chiusa. Il client per ottenere ulteriori risorse dallo stesso server (ad esempio le immagini della pagine web richiesta prima) deve stabilire una nuova connessione TCP per ogni risorsa.

3. **HTTP 1.1 (1997)**
    a. Vengono introdotti meccanismi di caching che memorizzano le risorse sul client in modo da evitare la richiesta di risorse già precedentemente richieste
    b. La connessione viene resa persistente a tempo, quindi la connessione TCP viene interrotta dopo un certo tempo e non dopo ogni request.
    c. Vengono introdotti i cookies
    d. Viene introdotto HTTPS (unione di HTTP con TLS) per aumentare la sicurezza della comunicazione.

4. **HTTP 2.0 (2015)**
    a. Le risorse vengono trasferite tutte in binario (anche il codice html o altri file di testo) e poi interpretate opportunamente dal browser
    b. Viene aggiunto il **multiplexing**, ossia la possibilità di trasmettere più stream di dati binari sulla stessa connessione TCP, come se ci fossero più canali di comunicazione (bidirezionali). Questo velocizza la trasmissione dei dati.
    c. Viene aggiunta la modalità **push** da parte del server. Grazie a questa modalità il server è in grado di "capire" quando diverse risorse fanno "probabilmente" parte di una stessa richiesta e inviarle senza attendere ulteriori request del client (il server prende l'iniziativa). Questo avviene ad esempio con le immagini, video, file CSS ecc.. che fanno parte di una stessa pagina web. Il client può limitare questa funzionalità o consentirla solo per determinate risorse. Anche questo velocizza la trasmissione dei dati.

d. E’ attualmente il piu’ utilizzato

5. **HTTP 3.0 (2022)**

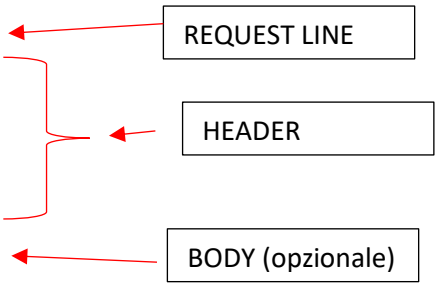
a. Si basa sul protocollo QUIC inventato da Google
b. Utilizza UDP anziché TCP come protocollo di trasporto (più veloce)
c. Rende obbligatorio l’utilizzo di TLS per la cifratura dei dati.

I MESSAGGI DEL PROTOCOLLO HTTP

Analizziamo come sono fatti la **request** e la **response** del protocollo HTTP

REQUEST

METODO	**URL**	**VERSIONE**
PARAMETRO	:	VALORE
PARAMETRO	:	VALORE
PARAMETRO	:	VALORE
….	:	….
BODY		



La prima riga si chiama **request line** e contiene:

- METODO: il metodo indica COSA vuole fare il client con la risorsa (ottenerla, modificarla, ecc.)
- URL: indica dove si trova la risorsa
- VERSIONE: è la versione del protocollo che il client utilizza per effettuare la request

Dopo la request line si trova **l’HEADER** (intestazione) costituito da una sequenza di coppie “parametro:valore”. Il numero di tali parametri non è fisso, dipende da come è implementata l’applicazione che effettua la request, vedremo successivamente alcuni dei parametri principali.

Il **BODY** contiene i dati che il client vuole inviare al server all’interno della request. Tale campo è opzionale, infatti verrà valorizzato solamente in presenza di alcuni metodi.

I **METODI** della **request line** principali sono i seguenti:

- GET: per chiedere di ottenere la risorsa (leggerla). Se tutto va bene, il server invierà la risorsa richiesta nel body della response.
- POST: per inviare dati a una risorsa (ad esempio per aggiungere dati ad una risorsa)

in realtà anche con GET è possibile inviare dati a una risorsa ma in tal caso i dati sono inseriti nell'URL quindi non è sicuro, l'URL non viene crittografato**. I dati inviati con POST, ad esempio i dati compilati di un form oppure i valori da aggiungere ad una tabella di un database nel server, verranno trasmessi all'interno del body della request.**

- PUT: per chiedere di modificare la risorsa (ad esempio modificare i valori di una tabella di un database). **I dati da modificare saranno indicati nel body**.
- DELETE: per chiedere di eliminare la risorsa (ad esempio eliminare i valori di una tabella di un database)
- HEAD: richiede il header di una risorsa. E' simile alla GET ma il server nella response (la cui struttura vedremo successivamente) invierà solamente il HEADER che conterrà informazioni sulla risorsa. non la risorsa stessa. Il body della response non conterrà la risorsa. Viene utilizzato per avere informazioni sulla risorsa, ad esempio per verificare se è stata modificata dall'ultima lettura, per sapere il tipo di dati della risorsa (ad esempio image/jpg o image/PNG…)
- OPTION: lo usa il client per richiedere al server quali metodi possono essere utilizzati nella request.

Vediamo alcuni esempi di parametri del **header della request** e a cosa servono:

- **Host**: specifica il nome dell'host a cui è destinata la richiesta, quindi può essere un nome di dominio oppure un indirizzo IP. Questo parametro è obbligatorio dalla versione 1.1 di HTTP
- **User agent:** contiene informazioni sul client che esegue la request, ad esempio la versione del browser utilizzata dal client.
- **Referer:** indica al server l'URL da dove arriva la request e serve per mantenere la tracciabilità delle navigazioni. Ad esempio quando da una pagina web A si clicca un link per aprire una pagina web B, il browser effettuerà una nuova request HTTP alla pagina B indicando nel parametro referer l'URL della pagina A. Tale parametro viene effettuato dalle piattaforme di analisi per verificare da dove arrivano le richieste di una pagina, per valutare l'efficacia di campagne di marketing, per consentire l'accesso a determinati contenuti solo agli utenti che provengono da determinati siti.
- **Accept**: il client specifica in quali formati è in grado di ricevere la risorsa richiesta. I possibili formati sono specificati da uno standard chiamato **Internet Media Type** che specifica il formato dei dati che possono essere scambiati su internet**.** Questo standard originariamente era nato per specificare il tipo di allegati che si potevano inviare con la posta elettronica ed era chiamato **MIME**)**. Esempi di Internet Media Type:**

1. Tipo di applicazione:

```
application/java-archive
application/EDI-X12
application/EDIFACT
application/javascript (obsolete)
application/octet-stream
application/ogg
application/pdf
application/xhtml+xml
application/x-shockwave-flash
application/json
application/ld+json
application/xml
application/zip
application/x-www-form-urlencoded
```

I media type sono sempre specificati da una coppia:

tipo/sottotipo

2. Tipo audio:

```
audio/mpeg
audio/x-ms-wma
audio/vnd.rn-realaudio
audio/x-wav
```

3. Digita immagine:

```
image/gif
image/jpeg
image/png
image/tiff
image/vnd.microsoft.icon
image/x-icon
image/vnd.djvu
image/svg+xml
```

Ad esempio, inviando il seguente parametro

Accept: image/gif image/jpeg

il client sta comunicando al server in che formato vuole le immagini. Se il server contiene nel proprio database le immagini in diversi formati, le invierà in formato gif o jpeg, e non tiff o png

- **Content type**: il client specifica quel è il formato **Internet Media Type** dei dati inviati nel body.
- **Connection**: specifica se il client vuole mantenere una connessione persistente, e quindi utilizzare l'attuale connessione TCP per effettuare ulteriori request, oppure no. Nel caso di connessioni persistente il valore del parametro è:

  Connection: keep – alive

  altrimenti è

  Connection: close

- **ETag**: contiene un codice che viene utilizzato nel meccanismo di cahing del client. Spieghiamo con un esempio. La prima volta che un browser richiede una risorsa ad un server, ad esempio un pagina web, il server invia la risorsa al browser. Tale risorsa è identificata da un codice ETag. Tale risorsa (la pagina web) viene memorizzata nella cache del browser. Quando il browser necessita di nuovo della stessa risorsa, è opportuno che carichi tale risorsa dalla cache (più veloce rispetto ad una nuova richiesta al server). Potrebbe però succedere che nel frattempo la risorsa sul server sia stata modificata, in tal caso la risorsa presente nella cache è "vecchia". Se il server modifica tale risorsa, modifica il suo codice ETag. Ciò che accade è che quando il browser invia una richiesta di una risorsa già precedentemente richiesta è che, nella request il browser inserisce il codice ETag della versione di tale risorsa presente nella cache. Il server verifica dal codice ETag se la risorsa "già posseduta" dl browser

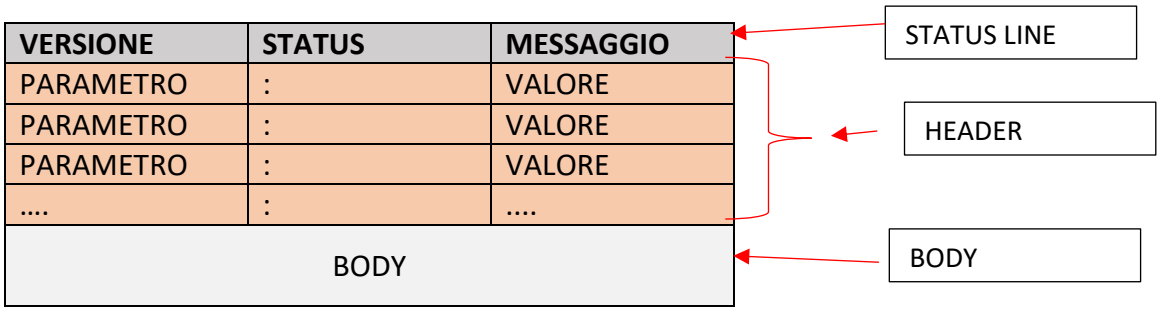
è l'ultima versione di tale risorsa confrontando il codice Etag inviato dal browser con il codice Etag della propria risorsa, e invierà la risorsa al browser solo se tale risorsa è stata modificata, altrimenti il server invia un codice che comunica al browser di caricare la risorsa dalla cache. (Si può verificarne il funzionamento facendo una GET con telnet con una semplice pagina html sul web server apache locale)

Ecco una tabella riassuntiva dei principali parametri del header sia della request sia della response

Header	Richiesta	Risposta	Descrizione
`Host`	×		Nome di dominio o indirizzo IP del server che ospita la risorsa.
`Server`		×	Nome del server HTTP che fornisce la risposta.
`Date`	×	×	Data/ora di invio della richiesta o della risposta.
`Content-Length`	×	×	Dimensione in byte del *body* della risposta o della richiesta.
`Content-Type`	×	×	Tipo MIME (*Internet Media Type*) del *body* (esempio: `text/html`).
`Last-Modified`		×	Data/ora di ultima modifica del *body* della risposta.
`Accept`	×		Elenco di tipi MIME accettati dal client (esempi: `text/xml`, `text/json`, ...).
`Accept-Encoding`	×		Tipo di codifica/compressione accettati dal client.
`Accept-Charset`	×		Tipi di codifica dei caratteri accettati dal client (esempi: `utf-8`, `iso-8859-1`, ...).
`User-Agent`	×		Informazioni relative al client che esegue la richiesta.
`Authorization`	×		Informazioni di autenticazione del client.
`Content-Encoding`		×	Tipo di codifica/compressione del *body* della risposta.
`Upgrade`	×		Richiesta di aggiornamento a un diverso protocollo come HTTPS o HTTP/2.

Analizziamo ora il messaggio response.

RESPONSE



La prima riga si chiama **status line** e contiene:

- VERSIONE: è la versione del protocollo che utilizzata dal server
- STATUS: contiene un codice a tre cifre che indica l'esito della request indicando se è andata a buon fine o no
- MESSAGGIO: è la trascrizione a parole del codice indicato nello STATUS

Dopo la request line si trova **l'HEADER** (intestazione) costituito da una sequenza di coppie "parametro:valore" analoghe a quelle viste per la request

Il **BODY** contiene i dati che il server invia al client. Se la request precedentemente inviata dal client è stata una GET ed è stata soddisfatta, il body della response contiene la risorsa richiesta, ad esempio il codice HTML della pagina web, oppure i byte che costituiscono l'immagine richiesta ecc. Nel caso in cui la request sia stata diversa dalla GET il body contiene comunque dei dati inviati dal client. Nella response il body contiene sempre dei dati, non è opzionale.

Vediamo nel dettaglio i possibili codici di stato. I codici sono sempre di 3 cifre, la prima cifra indica la categoria della risposta (successo, errore ecc.) mentre le altre cifre specificano il dettaglio dell'esito.

Formato del codice di stato	Categoria
1XX (100, 101, ...)	Informativa (codici di negoziazione tra client e server).
2XX (200, 201, ...)	Successo (codici che confermano al client che la richiesta avanzata al server è stata eseguita correttamente).
3XX (300, 301, ...)	Ridirezione (codici che comunicano al client che la richiesta avanzata al server NON è stata eseguita a causa di un'errata specificazione della risorsa: nella risposta può essere fornita una nuova valida specificazione).
4XX (400, 401, ...)	Errore del client (codici che comunicano al client che la richiesta avanzata al server NON è stata eseguita perché errata).
5XX (500, 501, ...)	Errore del server (codici che comunicano al client che la richiesta avanzata al server NON è stata eseguita a causa di un errore del server).

- 1XX: poco utilizzata, indica che il server ha ricevuto la request e che tale request è ancora in fase di elaborazione da parte del server.

- 2XX: successo. Indica che la richiesta ha avuto successo. Nel caso di una GET la risorsa richiesta sarà contenuta nel body.
  Esempio: 200 OK

- 3XX: indica che il client deve effettuare qualche altra azione affincchè la richiesta sia soddisfatta.
  Esempio: 303 See Other. Questo status indica che un'operazione è stata effettuata con successo e il client deve essere reindirizzato su un'altra risorsa. Ad esempio quando si compila un form di login, la response ha il codice 303 e il browser viene reindirizzato alla pagina di conferma. Nel Header vi sarà un parametro Location che contiene l'URL a cui il client deve essere reindirizzato:

  ```
  HTTP/1.1 303 See Other
  Location: https://esempio.com/contatto-completato
  ```

  Un altro esempio in cui è utilizzato il codice 303 è quando, in seguito alla richiesta di una risorsa già presente nella cache del client, il server comunica che l' ETag della risorsa nella cahce coincide con quello della risorsa sul server e quindi il client può caricare tale risorsa dalla cache.

- 4XX: indica un errore nella request (errore lato client) ad esempio errori di sintassi.
  Esempi:
  400 BAD REQUEST. La request non è stata compresa
  403 FORBIDDEN. Il client non ha l'autorizzazione per accedere alla risorsa
  404 NOT FOUND. La risorsa non è stata trovata

- 5XX: indica un errore del server. La request è sintatticamente corretta ma il server non è in grado di soddisfarla
  Esempi:
  500 INTERNAL SERVER ERROR.
  505 HTTP VERSION NOT SUPPORTED. Qundo la request utilizza una versione del protocollo non supportata dal server

Un esempio di richiesta e risposta HTTP è la seguente, in seguito vedremo altri esempi:

Accedendo con un client *Telnet* a un web server in esecuzione sullo stesso computer e che ha nella propria directory di lavoro il file `hello.html` contenente il codice HTML di una semplice pagina web, si possono inviare richieste e visualizzare le relative risposte.
Inviando la seguente richiesta[1]

Request

```
GET /hello.html HTTP/1.1
Host: 127.0.0.1
```

Request line

Header

si ottiene una risposta simile a questa:

Response

```
HTTP/1.1 200 OK
Date: Sun, 15 Aug 2021 06:10:40 GMT
Server: Apache/2.4.48 (Win64) OpenSSL/1.1.1k PHP/8.0.9
Last-Modified: Sun, 15 Aug 2021 05:58:59 GMT
Content-Length: 107
Content-Type: text/html

<html>
  <head>
    <title>Hello world</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

Status line

Header

Body

ESEMPI DI MESSAGGI HTTP

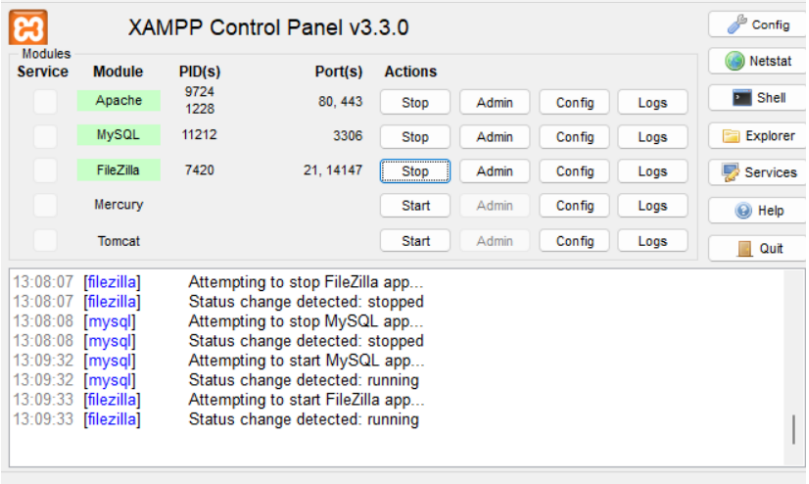
I metodi si possono testare accedendo con un client Telnet (avviabile dal prompt dei comadi) ad un web server (eventualmente anche un web server attivato sul proprio pc con il software XAMPP, utilizzando l'indirizzo IP di loopback 127.0.0.1)

Facciamo alcuni test sul metodo GET.

- Per abilitare Telnet sul prompt dei comandi windows: cercare FUNZIONALITA' WINDOWS, attivare la voce CLIENT TELNET.
- Per eseguire Telnet digitare nel prompt dei comandi: telnet
- Per uscire da telnet: digitare quit

ESEMPIO 1: Richiesta corretta

Avviare un web server sul PC, ad esempio con il software Xampp avviare il servizio Apache.



Creare una semplice pagina html nella cartella principale del web server (la cartella **C:\xampp\htdocs**), ad esempio la pagina giroitalia.html contenente del semplice testo e un'immagine.

1. Attivo una connessione con il web server

```
C:\Users\User>telnet 127.0.0.1 80
```

2. Effettuo la seguente GET con telnet (il codice non viene visualizzato)

   GET /giroitalia.html HTTP/1.1 <invio>

   Host: 127.0.0.1 <invio>

   <invio>

Nome della risorsa. Indicare l'URL senza il nome di dominio iniziando con /. Il nome di dominio va specificato nel parametro Host dalla versione 1.1 di HTTP.

E' necessario altrimenti la risorsa viene cercata nella cache del browser

3. Si ottiene la seguente risposta:

Stringa di stato

Header

Body. Contiene il codice HTML della risorsa

```
Prompt dei comandi

HTTP/1.1 200 OK
Date: Wed, 24 Jul 2024 11:22:06 GMT
Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/8.0.19
Last-Modified: Wed, 24 Jul 2024 11:18:48 GMT
ETag: "1d6-61dfc71321de8"
Accept-Ranges: bytes
Content-Length: 470
Content-Type: text/html

<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Il Giro d'italia</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script>
</head>
<body>
    <h1>IL GIRO D'ITALIA</h1>
    <p>Il giro d'Italia ecc...</p>
    <img src="giro.jpg">
</body>
</html>

Connessione all'host perduta.

C:\Users\Gian>
```

ESEMPIO 2: Richiesta risorsa non esistente (codice risposta 404)

GET /miro.html HTTP/1.1 <invio>

Host: 127.0.0.1 <invio>

<invio>

Risultato:

la stringa di stato risponde con un codice 404 che indica che la risorsa non è stata trovata

```
HTTP/1.1 404 Not Found
```

La risposta ha comunque un body. Copiando e incollandolo su un file di testo e aprendolo come pagina web si vede che è la pagina di errore che risulta nel browser quando si cerca una risorsa e non la si trova

ESEMPIO 3: Richiesta utilizzando un metodo non esistente (codice risposta 501)

XXX /giroitalia.html HTTP/1.1 <invio>

Host: 127.0.0.1 <invio>

<invio>

Risultato:

```
HTTP/1.1 501 Not Implemented
```

LE SESSIONI HTTP

Il protocollo HTTP è per sua natura stateless (senza stato) e le sessioni sono un modo per mantenere lo "stato" di una comunicazione fra client e server.

Spieghiamo cosa significa che HTTP è stateless. Ogni volta che il Browser effettua una HTTP request, è come se contattasse il server per la prima volta. Il protocollo infatti non tiene memoria (lo stato) degli scambi di dati avvenuti precedentemente fra client e server. Per questo si dice che HTTP è un protocollo **stateless** (senza stato, ossia senza memoria). Anche il protocollo HTTP2, che utilizza connessioni TCP persistenti, è comunque stateless a livello applicativo poiché le request sono sempre indipendenti una dall'altra.

Generalmente è utile per chi progetta un sito web tenere memoria delle visite precedentemente effettuate da uno stesso utente sul sito, per conoscere le preferenze di colui che visita il sito oppure

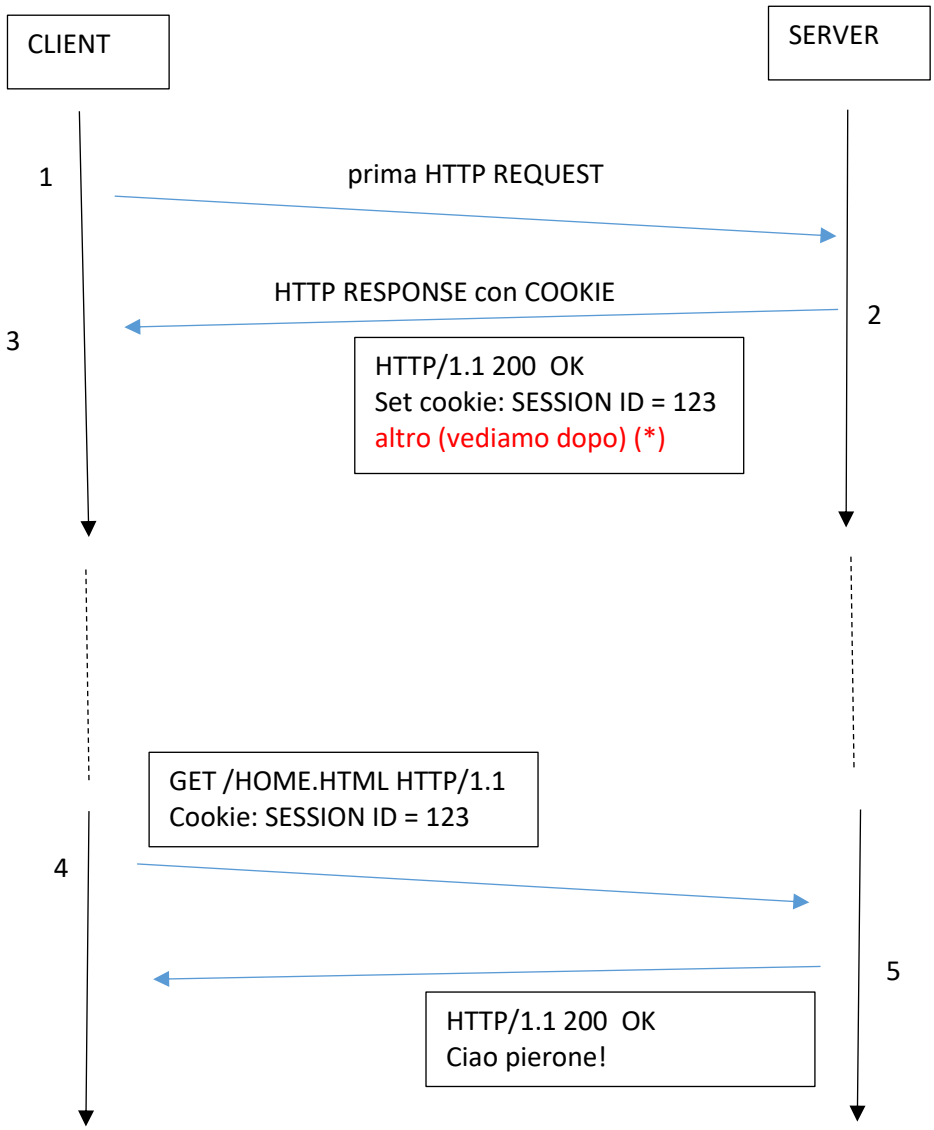
per adattare il contenuto delle pagine alle preferenze del visitatore. Ad esempio, nel caso di un sito di ecommerce, man mano che l'utente aggiunge prodotti al carrello, le pagine web successive dovranno mostrare il carrello degli acquisti con i prodotti aggiunti. Poiché il protocollo http è **stateless**, per mantenere la memoria di una sequenza di comunicazioni fra client server si utilizza il concetto di **sessione HTTP**.

La sessione è definita come una serie di scambi di informazioni legati fra di loro fra due punti di un sistema di comunicazione.

La sessione consente dunque di mantenere memoria dello scambio di dati fra i due host.

Possiamo pensare alla sessione come ad un semplice **codice identificativo**. Grazie a questo codice di sessione il Browser ed il Server "capiscono" che stanno comunicando nella stessa sessione.

Come avviene:



1 Il browser si connette per la prima volta al server,

2 Il server genera ed invia al browser il codice di sessione. Tale codice, è un codice identificativo chiamato “Session ID” che viene invito al client nel header della response nel parametro Set-Cookie. Come ha fatto il server a sapere che era la prima request HTTP da parte del client? Lo capisce dal fatto che la request non contiene alcun cookie.

3 Il Browser riceve questo parametro “cookie” (che contiene il session ID) ed ha la responsabilità di gestirlo. L'accordo è il seguente:

> il client si impegna a rimandare al server, assieme ad ogni richiesta, il cookie di sessione ricevuto con la prima risposta, senza modificarlo
>
> il server si impegna a tenere traccia dei cookies prodotti, usandoli come codice per riconoscere la sessione dell'utente al fine di identificarlo

4 Successivamente il client effettua un’altra HTTP request allo stesso server, ad esempio cliccando un pulsante per aprire un’altra pagina oppure cliccando un pulsante per aggiungere un prodotto al carrello ecc.. In queste successive request HTTP che il server invia al client, il client verifica se nel proprio file system vi sono cookie relativi a quel server e, in tal caso, glieli invia. Per inviarli si avvale del parametro cookie del header della request:

Esempio della request:

```
GET /HOME.HTML HTTP/1.1
Cookie: SESSION ID = 123
```

5 Il server, grazie al Session ID “riconosce” che la request si riferisce alla stessa sessione e quindi memorizza all’interno del proprio database, dei dati relativi al client, ad esempio: il nome, la pagina che sta visitando, i prodotti che ha messo nel carrello, ecc.. Grazie a queste informazioni il server potrà costruire (ad esempio in PHP) una pagina personalizzata con le preferenze dell’utente, ad esempio mostrando i prodotti che sono nel carrello, suggerendo prodotti simili ecc..

Si sottolinea che i dati della sessione (ad esempio l’elenco dei prodotti nel carrello…) potrebbero essere memorizzati sul client sotto forma di altri cookie, ma poiché tale memorizzazione non sarebbe controllata dal server, è ritenuta poco sicura, pertanto i dati sono di solito memorizzati sul server in un database, non sul client. Un altro motivo che rende più conveniente memorizzare i dati sul database del server è che i dati che si possono memorizzare sui cookie sono pochi (max 4 kB) rispetto a quelli che si possono memorizzare nel database. Il client quindi memorizza solamente l’id della sessione nell’apposito cookie.

(*) Altri parametri del header che possono essere inviati alla prima HTTP response oltre al campo Set-cookie sono i seguenti:

- **DOMAIN**: contiene il dominio al quale si riferisce il cookie, e quindi il dominio al quale inviare il cookie
- **PATH**: indica uno specifico pathname del server in cui il cookie è valido (più preciso del DOMAIN)

- **EXPIRES**: indica quando scade il cookie (numero di minuti, o giorni o data ora di scadenza). Se questo campo non è presente la sua scadenza avviene alla chiusura del browser (**cookie di sessione**). Se il valore è presente il cookie è persistente, viene memorizzato nella memoria di massa del client e viene eliminato dal client quando è scaduto (**cookie persistente**). si
- **SECURE**: booleano. Se è true il cookie verrà trasmesso solamente se il protocollo è HTTPS quindi garantendo la cifratura dei tati trasmessi.

Classificazione dei cookies.
I cookies si possono classificare in base a tre caratteristiche nei seguenti modi:

- cookies persistenti o di sessione.
  I coockies di sessione sono quelli che non sono memorizzati nella memoria di massa del'Host client e quindi vengono eliminati quando il browser viene chiuso. I cookies permamenti sono memorizzati nella memoria di massa del Client.

- cookies anonimi o specifici.
  Quando un utente ad un sito (ad esempio Amazon) senza effettuare il login, il cookie che viene creato consente al server di raccogliere informazioni sulla navigazione ma non di ricondurre tali informazioni ad un utente specifico. Amazon potrà sapere quante volte gli utenti hanno visitato una particolare pagina ma non quante volte uno specifico utente ha visitato una pagina. Questi sono cookies **anonimi**.
  Quando un utente naviga dopo aver effettuato il login, tutte le informazioni sulla navigazione potranno essere ricondotte all'account dell'utente. Quindi il server potrà memorizzare nel proprio database quali pagine sono state visitate da uno specifico utente e per quante volte. Questi sono cookies **specifici**.

- cookies di prime parti o di terze parti.
  Quando un utente visita un sito A , il server del sito A invia, alla prima response, il cookie di sito A. Se il sito A contiene al suo interno un oggetto, ad esempio un'immagine minuscola delle dimensioni di un pixel, che si trova su un altro dominio (ad esempio un server di una piattaforma pubblicitaria come Google Ads o Facebook Ads) il browser, per ottenere tale oggetto e visualizzarlo, invierà una request HTTP al server del sito pubblicitario. Insieme all'immagine minuscola, nella response, il sito pubblicitario invierà anche un proprio cookie. In questo modo la piattaforma pubblicitaria potrà conoscere informazioni sulla navigazione dell'utente (quali pagine ha visitato, quanto tempo si è sono fermato su una pagina, su un'immagine, quali prodotti ha acquistato, ecc.). Il cookie della piattaforma pubblicitaria è un **cookie di terze parti**, nel senso che non è un cookie che arriva direttamente dalla pagina che l'utente sta visitando. Questo cookie consente alla piattaforma pubblicitaria di acquisire informazioni sulla navigazione dell'utente.

  Quali informazioni può memorizzare la piattaforma pubblicitaria (ad esempio Google ads)? Dipende da quali cookie l'utente ha accettato di condividere. Comunque, anche se l'utente

non accetta di trasmettere i propri dati personali (ad esempio nome ed email) la piattaforma pubblicitaria potrà creare un profilo anonimo delle attività di navigazione svolte dall'utente. Se l'utente sta navigando in forma anonima, la piattaforma pubblicitaria assocerà all'utente un ID utente =1 e potrà memorizzare le preferenze di navigazione di ID utente=1. Saprà inoltre che ID utente =1 si è connesso da una certa zona geografica, che ha utilizzato un certo dispositivo, ed altre informazioni che consentiranno di creare un profilo di ID Utente=1. Addirittura, se l'utente ha fatto il login utilizzando l'account già in possesso sella piattaforma pubblicitaria (ad esempio Google o Facebook), tutte le attività svolte saranno memorizzate e associate al profilo dell'utente.

Cosa accade quando l'utente accede ad un altro sito, chiamiamolo sito B, il quale contiene anche lui l'oggetto minuscolo della piattaforma pubblicitaria?
Anche in questo caso il browser effettua la prima HTTP request al server del sito B, ricevendo nella response i cookie di prima parte del sito B. A questo punto il browser effettuerà la request al sito della piattaforma pubblicitaria per ricevere l'oggetto minuscolo, ma in questo caso, avendo già memorizzato il cookie di terze parti, invierà al server della piattaforma tale cookie. Poiché tale cookie contiene ID Utente=1, la piattaforma pubblicitaria raccoglierà ulteriori informazioni di navigazione al profilo di ID Utente=1, inoltre potrà inviare dei messaggi pubblicitari specifici a quell'utente.

In questo modo la piattaforma pubblicitaria può' conoscere una grande quantità di informazioni sull'utente ID=1 arrivando a mettere a repentaglio la riservatezza dei dati.

DOMANDE GUIDA

- Cosa si indica con web 1.0, web 2.0, web 3.0 e web 4.0?
- In cosa consiste la tecnologia packet switching? Descrivi le principali caratteristiche di questa tecnologia
- Cosa è lo stack ISO/OSI? Descrivi i compiti principali dei suoi 7 livelli
- Cosa è la suita TCP/IP? Quali sono i protocolli più utilizzati dalla suite TCP IP? Spiega il concetto di incapsulamento dei PDU. Qual è il nome dei PDU nei protocolli fi livello 2, 3 e 4? Cosa è il payload?
- Protocollo HTTP:
    - A che livello dello stack ISO OSI si trova?
    - Come è cambiato nel tempo?
    - Come sono fatte la request e la response?
    - Cosa significa che è stateless?
    - Cosa sono le sessioni e icookies?
    - Spiega le diverse tipologie di cookies.