Reti di Calcolatori a.a. 2019/2020

Francesco Iannelli

September 16, 2019

Contents

1 Introduzione al corso

email: federica.paganelli@unipi.it
modalità d'esame: prova scritta (compitini) e orale (facoltativo) più laboratorio che prevede progetto e orale.
N.B. Si accede alla prova orale di laboratorio solo dopo aver passato lo scritto.

2 Cenni di modelli stratificati

Sono architetture di comunicazione a strati.
Concetti generali:

- Stratificazione
- Information hiding
- Separation of concerns

Vantaggi della stratificazione:

- **Facilità di progettazione.**
- **Facilità di manutenzione.**
- **Possibilità di riciclo.**

Due modelli: **ISO/OSI** (approccio top-down) e **Stack TCP/IP** (approccio bottom-up), quest ultimo vincente.
Idea chiave: suddivisone in sottoproblemi.

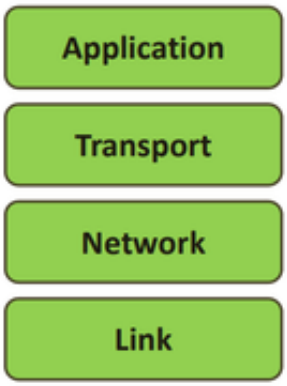


Figure 1: Modello stratificato

2.1 Livello applicativo

Fanno parte del livello applicativo:

- Identificativi delle risorse: URL, URI e URN.
- Il web: user agents, protocollo http.
- Protocollo FTP.
- TELNET: servizio di terminale virtuale.
- Posta elettronica.
- Sistema dei nomi DNS a dominio e la risoluzione dei nomi: iterativa e ricorsiva.
- molto altro ancora...

2.2 Livello di trasporto

Due tecnologie degne di nota:

1. Protocollo **TCP**: **connection-oriented**, *orientato alla connessione.*
2. Protocollo **UDP**: **connection-less**, molto più leggero, prende dati applicativi e li affida allo strato IP, NON da garanzie di consegna nè di ordine.

2.3 Livello rete

Nel livello rete si ricava un percorso dall'host sorgente all'host destinatario usando le informazioni che si trovano nell'IP.
Verrà trattato il protocollo Ipv4 e introdotto il protocollo Ipv6.

2.4 Livello link

Si occupa di gestire il collegamento tra due nodi **adiacenti**. La tecnologia principale è l'**ethernet**.

3 Introduzione alle reti

Cos'è una rete? Quante tipologie di reti ci sono? Cos'è internet?

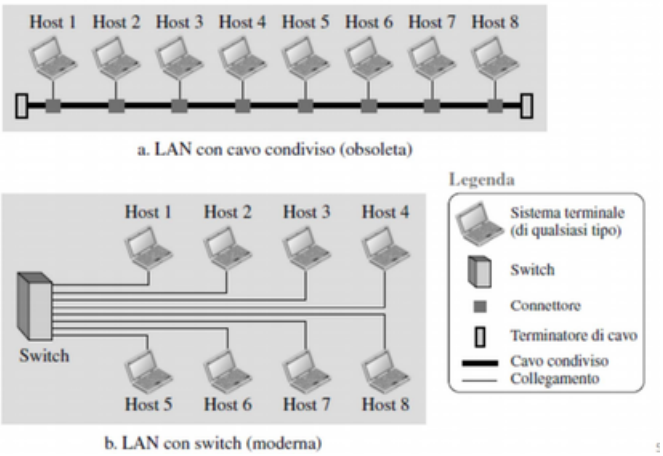
Definizione 3.1. Una **rete** è un'interconnessione di dispositivi in grado di scambiarsi e interpretare le informazioni. Comprende sistemi terminali e intermedi: *e.g. router, switch e modem.*

I sistemi terminali si possono dividere in due tipi: **host** e **server**, sull'host girano le applicazioni utente mentre il server esegue programmi che forniscono servizi applicativi ad applicazioni.
N.B. il termine host può essere usato per indicare anche un server.
L'host, infatti, può essere sia un server sia il terminale di un utente che esegue un'applicazione client, più generalmente ***l'host è una macchina****.*

Definizione 3.2. Una rete è formata da **dispositivi** e da **tecnologie**.

3.1 LAN

Acronimo di **L**ocal **A**rea **N**etwork, è una rete di area geografica limitata collegata attraverso una tecnologia ethernet **bus** o ethernet **switch**: Ciascun



host ha un cavo che lo collega allo switch e a ogni porta dello switch corrisponde un host. Lo **switch** possiede una tecnologia di autoapprendimento ed è una componente del **livello link**.

3.2 WAN

Acronimo di **W**ide **A**rea **N**etwork, è una rete di area geografica estesa: è composta da due o più reti collegate tramite un mezzo di trasmissione. Le reti coinvolte potrebbero anche essere reti LAN. *(e il link potrebbe essere affittato a un'azienda da un operatore di telefonia).*

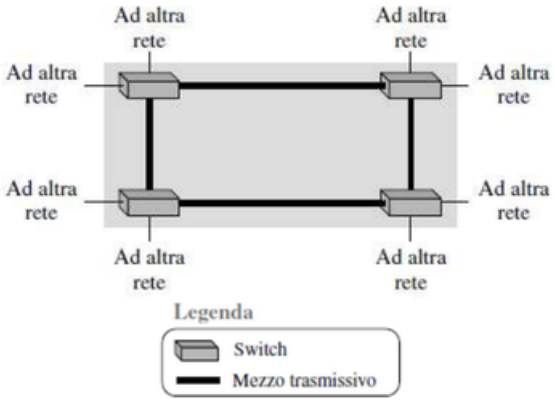


Figure 2: Un esempio di WAN a cavo condiviso e a commutazione.
Queste WAN permettono l'esistenza di **percorsi alternativi** e la **divisione del traffico**. Una **WAN punto punto** invece ha **solo** 2 punti di terminazione.

3.3 Internetwork

L'internetwork è un sistema in cui ci sono più reti composte, capaci di scambiarsi informazioni e collegate. Concettualmente è una WAN ma è più complicata.
I dispositivi che la compongono si distinguono in **sistemi terminali** e dispositivi come gli **switch** e i **routers** che si trovano nel percorso tra i sistemi sorgente e i sistemi destinazione.

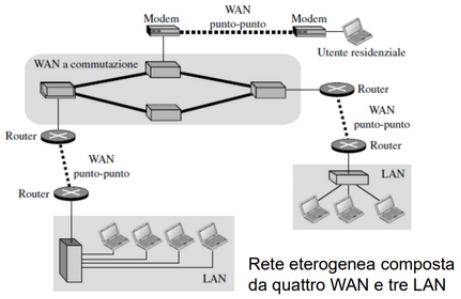


Figure 3: Una internetwork.

Problema: come mandare informazioni da un host a un altro?

3.3.1 Reti a commutazione di circuito

Nelle **reti a commutazione di circuito** le **risorse** sono **riservate end to end** per ogni connessione. Risulta quindi necessario il **setup della comunicazione** per instaurare la connessione ed elargire le risorse.
La risorsa non è tutto il link, bensì si considerano come risorse la capacità o la larghezza di banda porzionate per ogni connessione. Le risorse assegnate rimangono inattive se non utilizzate (*e.g. telefonata*). I dispositivi mantengono lo stato della connessione. **Le performance sono garantite**. La capacità delle linee (*link*) cambia a seconda della loro funzione all'interno della rete. Il **punto debole** delle reti a commutazione di circuito è la **poca flessibilità nel dispiegamento delle risorse**.

3.3.2 Reti a commutazione di pacchetto

Nelle **reti a commutazione di pacchetto** gli utenti inviano pacchetti che condividono le risorse del canale di comunicazione. **Non c'è** quindi **un canale dedicato** ai pacchetti di un singolo utente. La principale differenza rispetto alla commutazione di circuito risiede nell'implementazione della logica dei dispositivi di interconnessione, ovvero:

- **Commutazione di Circuito**: avviene il **setup** della connessione dove si prealloca l'utilizzo del collegamento trasmissivo con collegamenti garantiti.
- **Commutazione di Pacchetto**: non viene instaurata una connessione bensì le informazioni necessarie si trovano all'interno dei pacchetti stessi, non ci sono informazioni di connessione memorizzate nei dispositivi coinvolti.

Nelle reti a commutazione di pacchetto quindi le risorse vengono usate a seconda della necessità. Possono quindi verificarsi situazioni di contesa delle risorse e sussiste il pericolo di congestione o di perdita dei pacchetti nel caso in cui la dimensione della coda del router non fosse sufficiente a contenere il flusso dei pacchetti entranti: il commutatore (*router*) deve infatti ricevere l'intero pacchetto prima di poter cominciare a trasmetterlo sul collegamento in uscita (***store and forward***). **Non sono** quindi **garantite le prestazioni**.

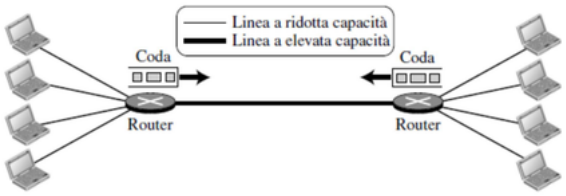


Figure 4: Rete a commutazione di pacchetto. Da notare le diverse capacità dei link.

3.3.3 Packet Switch e Circuit Switch a confronto

Vi sono 35 utenti su una rete con 100 Kbit/s di connessione e un link da 1 Mbit/s. Ogni utente è attivo solo il 10% del tempo.
Con una **rete a commutazione di circuito** si riescono a gestire **solo** 10 utenti.
Con una **rete a commutazione di pacchetto** si hanno i seguenti casi:

1. 10 o meno utenti attivi: nessun problema.
2. Più di 10 utenti attivi: ritardo.

Tuttavia che gli utenti siano tutti e 35 attivi contemporaneamente è poco probabile (infatti $P(35) = 0.0004$). Se ne deduce che la rete a commutazione di pacchetto riesce a gestire tutti gli utenti contemporaneamente nella maggior parte dei casi.

Nonostante il risultato ottenuto non si deve pensare che la rete a commutazione di circuito sia obsoleta. Nel corso degli anni infatti le due tecnologie sono state **integrate** in vari modi.
La commutazione di circuito infatti è usata nella telefonia fissa (*PSTN: public switch telephone network*) per i servizi voce, la commutazione di pacchetto invece per i dati.
Nelle reti ottiche di prima e seconda generazione si usano entrambe le tecnologie.

3.3.4 Circuiti virtuali

I circuiti virtuali funzionano nel seguente modo: viene stabilito un path tra host sorgente e host destinazione e tutti i pacchetti di un certo flusso seguono lo **stesso** path.

3.3.5 Datagram Network

Con **datagram** si indica un'entità informativa autocontenuta che contiene le informazioni sufficienti per essere indirizzata alla destinazione senza comunicazioni aggiuntive tra sorgente e destinazione: **non è quindi detto** che pacchetti di uno stesso flusso seguano lo stesso path sulla rete.

3.4 Internet

Come interconnettere reti già esistenti?

Definizione 3.3. Una internet (con i minuscola) è una rete costituita da due o più reti interconnesse.

La internet più famosa è chiamata **Internet** (con i maiuscola) ed è composta da migliaia di reti interconnesse. Ogni rete connessa ad Internet deve usare il protocollo IP e rispettare certe convenzioni su come vengono assegnati nomi e indirizzi. Si possono facilmente aggiungere nuove reti.
Tuttavia è impensabile avere un link fisico tra ogni host, si hanno invece numerosi dispositivi di interconnessione che permettono la comunicazione da un host all'altro e da un router all'altro.
Uno scorcio delle **componenti di Internet**:

- Miliardi di dispositivi interconnessi (e.g. hosts, end systems).
- Link di comunicazione (e.g. fibre ottiche, doppini telefonici, cavi coassiali, onde radio).
- Routers: instradano pacchetti *(sequenze)* di dati attraverso la rete.

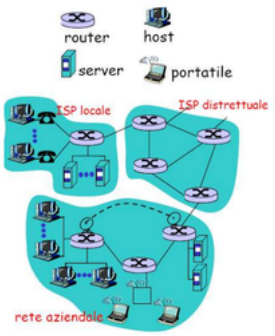


Figure 5: Una porzione di Internet

Uno scorcio delle **entità software** di Internet:

- Applicazioni e processi che elaborano le informazioni.
- **Protocolli** che regolamentano la trasmissione e la ricezione di informazioni e.g. TCP, IP, HTTP, FTP, PPP.
- Interfacce: verranno definite in seguito, sono le *"membrane"* che separano gli *"strati"*.

3.4.1 Servizi

L'**infrastruttura di comunicazione** consente il funzionamento delle applicazioni distribuite per scambio di informazioni *(e.g. WWW, email, giochi, e-commerce, database, controllo remoto, ecc).*
Lo **stack protocollare** offre il servizio di connessione. Vi sono due approcci:

1. **Connection-less**: I dati vengono trasferiti **senza** stabilire una connessione, non c'è nessuna garanzia di ordine e consegna. *Ogni pacchetto ha una vita a sè.*

2. **Connection-oriented**: Prevede l'**instaurazione della connessione**, il trasferimento dei dati e, in seguito, la chiusura della connessione. Garantisce integrità, completezza e ordine.

3.4.2 IETF/RFC/ICANN

Definizione 3.4. L'IETF (Internet Engineering Task Force) è l'organismo che studia e sviluppa i protocolli in uso su Internet. Si basa su gruppi di lavoro a cui chiunque può accedere.

Definizione 3.5. RFC/STD (Request For Comments & STanDards) sono i documenti *"ufficiali"* che descrivono i protocolli usati su Internet. Sono pubblicamente accessibili in rete.

Definizione 3.6. ICANN (Internet Corporation for Assigned Names and Numbers) È un ente internazionale che coordina il sistema dei nomi di dominio (DNS), assegna i gruppi di indirizzi di rete, identificativi di protocollo e ha funzioni di controllo (blando) dello sviluppo di Internet.

3.4.3 Rete di accesso

Internet è una internetwork che consente a qualsiasi utente di farne parte. L'utente, tuttavia, deve essere fisicamente collegato a un ISP (*internet service provider*).

Definizione 3.7. Il collegamento che connette l'utente al primo router di internet è detto **rete di accesso**, suddetto collegamento può essere effettuato tramite rete telefonica, rete wireless o tramite accesso diretto.

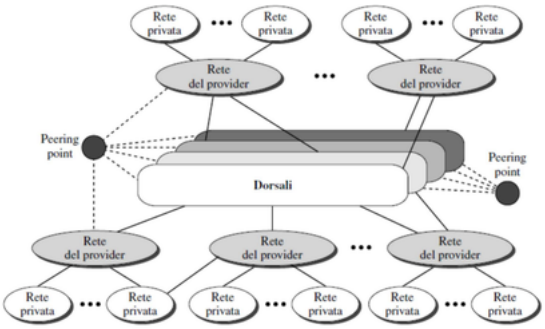


Figure 6: Un modello concettuale di Internet

4 Metriche di riferimento

Come misurare le prestazioni di una rete?

Definizione 4.1. La **larghezza di banda** o **bandwidth** è la larghezza dell'intervallo di frequenze utilizzato dal sistema trasmissivo.

Definizione 4.2. Il **bit rate** o **trasmission rate** è la quantità di dati che possono essere trasmessi o ricevuti nell'unità di tempo. [e.g. bps = bit/s]
Il bitrate dipende dalla tecnica trasmissiva ed è proporzionale alla larghezza di banda.

Definizione 4.3. Il **throughput** è la quantità di traffico che arriva realmente a destinazione nell'unità di tempo, al netto di perdite sulla rete, del funzionamento dei protocolli etc.

Definizione 4.4. La **latenza** o **latency** è il tempo che passa dal momento in cui il primo bit parte dalla sorgente al momento in cui l'intero messaggio arriva a destinazione.

$$\boxed{L = r_{propagazione} + r_{trasmissione} + r_{accomodamento} + r_{elaborazione}}$$

4.1 Ritardi

Il ritardo introdotto da un nodo è la somma di questi 4 ritardi:

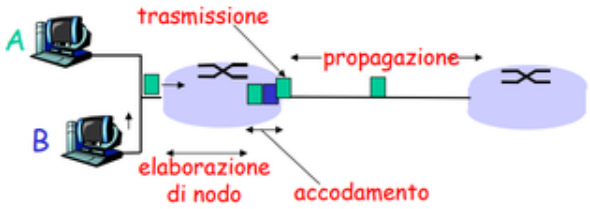


Figure 7: Una visione di contesto

4.1.1 Ritardo di elaborazione del nodo

Il ritardo di elaborazione **è causato dall'elaborazione del percorso** (ovvero dove inoltrare il pacchetto scegliendo il percorso *"migliore"*) **e dal controllo di errori sui bit**, è **tipicamente piccolo e trascurabile**.

4.1.2 Ritardo di accodamento

Il ritardo di accodamento **è il tempo che un pacchetto passa nella coda del router**, dipende dall'intensità e dal tipo di traffico. I pacchetti si accodano nei buffer dei router se il tasso di arrivo dei pacchetti eccede la capacità del collegamento di inoltrarli. Se non ci sono spazi liberi i pacchetti in arrivo vengono scartati.

4.1.3 Rtardo di trasmissione

Il ritardo di trasmissione **è il tempo impiegato a trasmettere un pacchetto intero** sul link.
R = rate di trasmissione del collegamento.
L = lunghezza del pacchetto.

$$r_{trasmissione} = \frac{L}{R}$$

4.1.4 Ritardo di propagazione

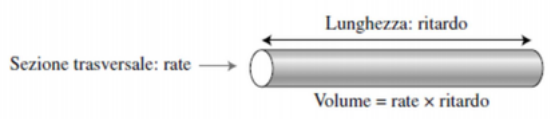
Il ritardo di propagazione **è il tempo impiegato da un bit per essere propagato da un nodo (router) all'altro**.
d = lunghezza del collegamento fisico
s = velocità di propagazione del collegamento fisico

$$r_{propagazione} = \frac{d}{s}$$

4.2 Esempio

Si consideri l'invio di un file di 1 MBit su un datalink di lunghezza 4800km:
$d = 4800 \times 10^3 m$
$s = 3 \times 10^3 m/s$
Si calcoli il ritardo di propagazione.
Soluzione:
$r_{propagazione} = \frac{d}{s} = \frac{4800 \times 10^3 m}{3 \times 10^3 m/s} = 0.016$ secondi.
Sia il trasmission rate pari a 64 kbps, si calcoli il ritardo di trasmissione.
$r_{trasmissione} = \frac{L}{R} = \frac{10^6 bit}{64 \times 10^3 bps} = 15.625$ secondi.
Se il trasmission rate fosse invece di 1 Gbps?
$r_{trasmissione} = \frac{L}{R} = \frac{10^6 bit}{10^9 bps} = 0.001$ secondi.

4.3 Volume di un link



Definizione 4.5. Il volume di un link è il numero massimo di bit che il link può contenere.

$$\boxed{Volume = bitrate \times ritardo}$$

4.4 Esempio

Si calcoli il ritardo end-to-end di un pacchetto su un percorso con due router. Sia trascurabile il ritardo di congestione e si suppongano uguali su tutti link il propagation delay, il transmission delay e il processing delay.

Soluzione:
Essendoci due router intermedi bisongna attraversare tre link, quindi:

$$r_{totale} = 3 \times r_{propagation} + 3 \times r_{trasmission} + 3 \times r_{processing}$$

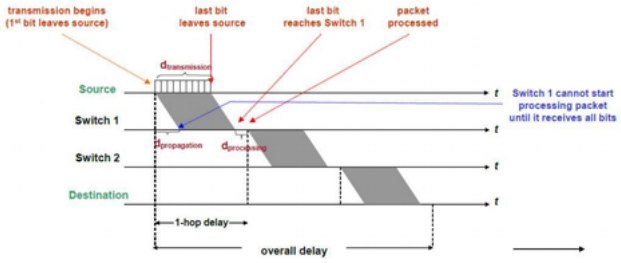


Figure 8: Quello che succede in dettaglio nell'esempio precedente.

5 Modelli Stratificati e Protocolli

Cos'è un protocollo? Cos'è uno strato?

Figure 9: Esempio di stratificazione. Si nota come i vari strati del modello interagiscono tra di loro, dal basso verso l'alto e viceversa, per consentire al sig. Rossi e al sig. Takamura di scambiarsi lettere.

Vademecum per la lettura del seguente contenuto: N.B.:

1. Gli strati comunicano *(di solito)* **solo** con gli altri strati a loro adiacenti.
2. Uno strato **fornisce** servizi allo strato superiore e **riceve** servizi da quello inferiore.
3. La comunicazione tra strati adiacenti avviene attraverso un'**interfaccia**.
4. Tra due entità diverse comunicano fra di loro solo gli strati dello **stesso livello** e secondo un **protocollo assegnato**, queste due entità sono dette **peer**.

5.1 OSI: Open Systems Interconnection

Le **prime** reti di calcolatori nacquero come **sistemi chiusi** in cui tutti i componenti dovevano essere dello stesso costruttore. Erano quindi tecnologie chiuse e **non interoperabili** l'una con l'altra a causa di drastiche differenze *(e.g. differenza di linguaggio,* ***modelli di stratificazione diversi*** *e impossibilità per i programmi applicativi di riuscire ad operare in ambiente distribuito).* Alla fine degli anni '60 esistevano: ARPANET, SNA (IBM), DNA (Digital).
I **Sistemi Aperti** nascono dall'obbiettivo di alcune aziende di realizzare una rete di calcolatori in cui qualsiasi terminale potesse comunicare con un qualsiasi fornitore di servizi mediante qualsiasi rete.

Per realizzare un sistema aperto è necessario stabilire delle regole comuni: **gli standards**.

Definizione 5.1. Un sistema che implementa **protocolli aperti** è un **sistema aperto** (open system).

Definizione 5.2. Un set di protocolli è **aperto** se:

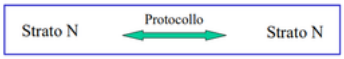
1. I dettagli (**specifiche**) dei protocolli sono disponibili pubblicamente.
2. I cambiamenti al set sono gestiti da un'organizzazione la cui partecipazione è aperta al pubblico

5.1.1 Modello ISO/OSI

L'International Organization for Standards (ISO) ha specificato uno standard per l'interconnessione di sistemi aperti: l' **Open System Interconnection Reference Model** (OSI-RM) poi diventato standard internazionale nel 1983 (ISO 7498). **Si basa sul concetto di architettura a strati i cui criteri sono:**.

- **Divisione delle funzionalità**: il protocollo di telecomunicazione è diviso in strati o layers, ognuno dei quali svolge un compito piccolo e indipendente dagli altri.
  Si cerca quindi di mantenere un minor numero di strati possibile e di far svolgere a ognuno di essi il minor numero di compiti possibile.
- **Comunicazione mediante interfacce**: i livelli comunicano mediante **chiamate standard**. Ogni livello è tenuto a rispondere alle **sole** chiamate che gli competono e che verranno invocate dai due livelli ad esso adiacenti.
- **Information hiding**: le modalità con cui le funzioni competenti ad un livello vengono svolte non è visibile dall'esterno che ne è così svincolato.

5.2 Protocollo



I protocolli definiscono il **formato** e l'**ordine** dei messaggi inviati e ricevuti tra entità della rete al livello n-esimo e le **azioni** che vengono fatte per la loro **trasmissione** e **ricezione**.

Definizione 5.3. Un **protocollo** è un insieme di regole che permettono a due entità uno scambio **efficace** ed **efficiente** delle informazioni. **Definisce** il **formato** e il **significato** dei frame (campi del messaggio), dei pacchetti o dei messaggi che vengono scambiati tra gli **strati paritari** di due entità diverse.

Un protocollo specifica quindi:

- La **sintassi** di un messaggio (e.g. i campi).
- La **semantica**.
- **Le azioni da compiere** (e.g. per l'invio, alla ricezione, alla trasmissione etc...).

Definizione 5.4. Uno **strato** o livello è un modulo interamente definito attraverso i servizi, protocolli e le interfacce che lo caratterizzano.

Definizione 5.5. Un' **interfaccia** è il set di regole governanti sintassi e semantica della comunicazione tra due **strati successivi** della **stessa entità**.

Definizione 5.6. Un **servizio** è insieme di **primitive** (operazioni) che uno strato fornisce ad uno strato soprastante. (*vedi sez. 3.4.1*).

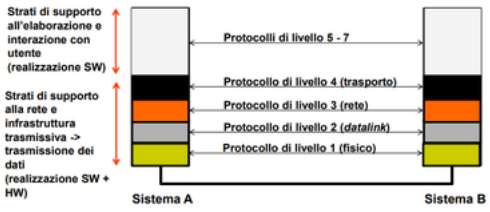


Figure 10: Esempio di stack protocollare.

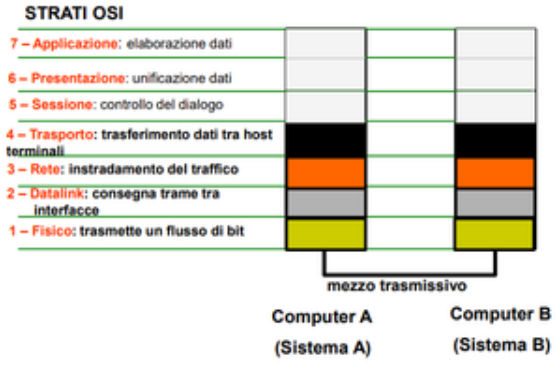


Figure 11: Gli strati di OSI.

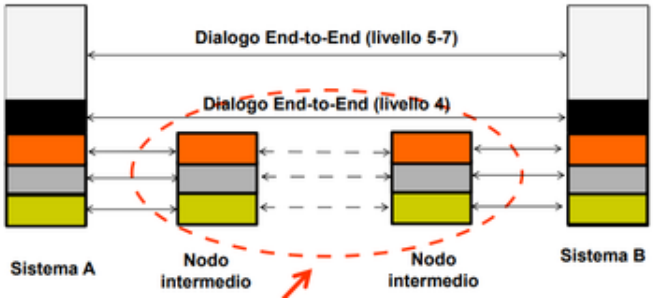


Figure 12: Esempio di collegamento tra end systems.

5.3 Incapsulamento dell'informazione

All'interno della rete l'informazione ha origine **al livello applicativo** (*livello 7 in figura*), discende quindi i vari livelli fino alla **trasmissione**, che avviene mediante il **canale fisico**. Da ogni livello attraversato viene aggiunta all'informazione una sezione informativa (o più di una) chiamata **header** che contiene informazioni pertinenti esclusivamente al livello stesso. Per i dati ricevuti invece si segue il cammino inverso. Si tratta infatti di un **processo di incapsulamento reversibile**.

- **Header**: è qualificazione del pacchetto dati per questo livello.
- **DATA**: è il payload proveniente dal livello superiore.
- **Trailer**: è usato per rilevare e correggere gli errori.

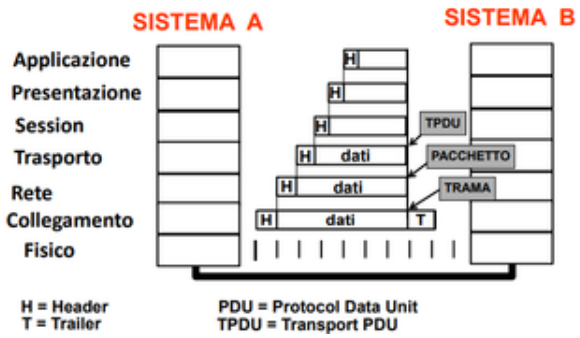


Figure 13: Il processo di incapsulamento. Da notare in particolare il payload.

6 Stack protocollare TCP/IP

TCP/IP è una **famiglia di protocolli** attualmente in uso su Internet. Si tratta di una **gerarchia di protocolli** costituita da **moduli interagenti**, ciascuno dei quali fornisce funzionalità specifiche. Il termine **gerarchia** significa che ciascun protocollo di **livello superiore** è supportato dai servizi **forniti** dai protocolli di **livello inferiore**. Definita in origine in termini di quattro livelli software soprastanti a un livello hardware, **la pila TCP/IP** è **oggi** intesa come **composta di cinque livelli**.

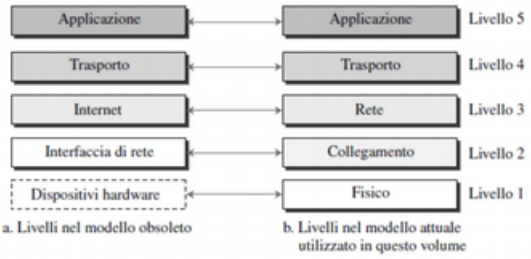


Figure 14: Livelli dello stack protocollare TCP/IP, si notino le differenze tra il modello originario e il modello attuale.

- Il **livello applicativo** supporta le applicazioni di rete.
- Il **livello di trasporto** supporta il trasferimento di dati da un host all'altro.
- Il **livello di rete** instrada i datagrammi dalla sorgente alla destinazione.
- Il **livello link** trasferisce dati tra elementi adiacenti della rete.
- Al **livello fisico** troviamo i bit sul link.

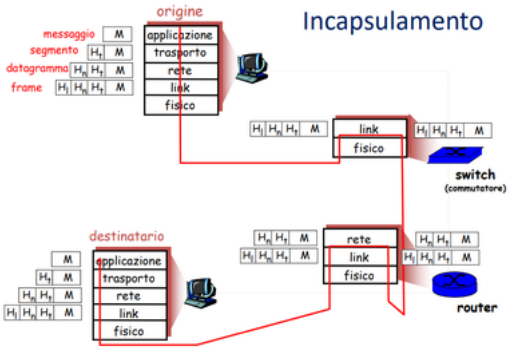


Figure 15: Esemplificazione del processo di incapsulamento/decapsulamento dell'informazione.

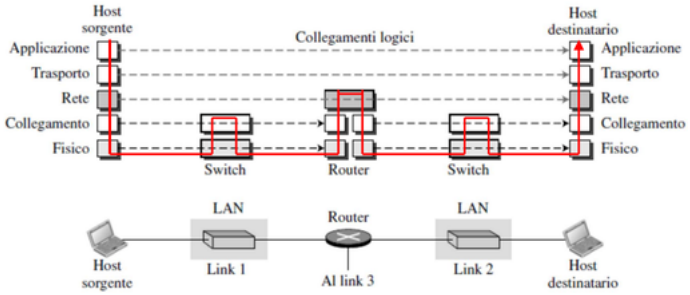


Figure 16: I vari collegamenti logici di comunicazione, in **rosso** invece, **il collegamento fisico**. La modularità del sistema fa in modo che gli strati paritari dei due host abbiano l'**illusione** di comunicare direttamente tra di loro. Si ricorda che queste entità situate a livelli corrispondenti su macchine (host) diverse sono dette **peer**.

6.1 Lo strato applicativo

Dello **strato applicativo** fanno parte le **applicazioni di rete**. Le applicazioni di rete sono composte da **processi distribuiti** e **comunicanti**, ovvero programmi eseguiti dai dispositivi terminali (host o end system) di una rete. Nella comunicazione **a livello applicativo** fra due dispositivi terminali interconnessi, due o più processi sono in esecuzione su ciascuno degli host comunicanti e si scambiano messaggi. Il protocollo dello strato applicativo definisce: (*repetita iuvant*)

- I **tipi** di messaggi scambiati al livello applicativo (e.g. richiesta e risposta).
- La **sintassi** e la **semantica** dei campi dei messaggi.
- Le **regole** di comunicazione **interprocessuale**.

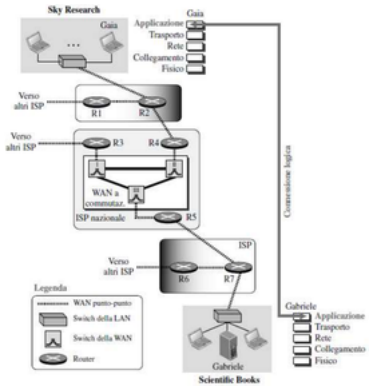


Figure 17: Esempio di comunicazione tra applicazioni di rete. Ancora una volta è bene ricordare che il protocollo crea l'*illusione* che i processi siano in comunicazione diretta.

Presentiamo ora i **paradigmi di comunicazione** dello strato applicativo:

- **Client-Server**: prevede un numero *limitato* di processi **server** che offrono servizi e sono **sempre** in esecuzione in attesa di ricevere richieste dai client. Un **client** è un programma che richiede un servizio. Tipicamente il client inizia il contatto con il server **inviando una richiesta** e il server risponde **offrendo il servizio** richiesto.
- **Peer-to-Peer**: comunicazione tra **peer** *(pari)* che possono sia **offrire** servizi che **inviare** richieste.
- **Misto**.

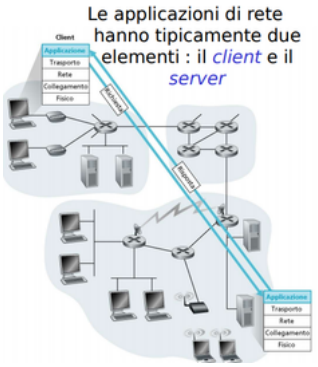


Figure 18: Esempio di paradigma client-server.

Abbiamo, seppur assai brevemente, esaminato la comunicazione a livello applicativo tra macchine diverse. Scendiamo ora più nel dettaglio iniziando a tracciare il percorso **reale** dei dati dallo **strato applicativo** allo **strato di trasporto**. Partiamo dalla seguente definizione:

Definizione 6.1. API: acronimo di Application Programming Interface, è un insieme di **procedure** e **regole** che un programmatore deve seguire per accedere a delle risorse. Facilita molto la programmazione del software client-side.

L' **API** che funge da **interfaccia** tra gli **strati di applicazione e** di **trasporto** è chiamata **socket** ed è usata dai processi dello strato applicativo per inviare e ricevere dati dallo strato di trasporto. Si tratta, ancora una volta, di una connessione **logica** poichè in realtà l'invio e la ricezione dei dati sono, nel **concreto**, responsabilità del sistema operativo e del protocollo TCP/IP.
Riportiamo un estratto dell'**API** di **TCP:**

```
connection TCPopen(IPaddress, int) //to open a conn.
void TCPsend(connection, data) //to send data
data TCPreceive(connection) //to receive data
void TCPclose(connection) //to close a conn.
```

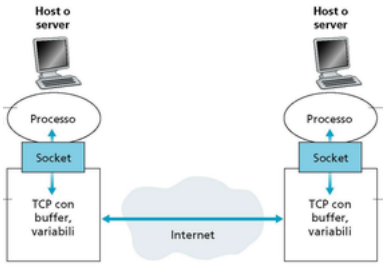


Figure 19: In figura due processi che comunicano tramite socket. Il processo è controllato dallo sviluppatore dell'applicazione, tutto ciò che è presente sotto la socket è controllato dal sistema operativo.

Sorge ora spontanea una domanda: **se ci fossero più processi su ogni host**?
Servirebbe un modo per **identificarli**... ci viene in aiuto il **socket address**.

Figure 20: Il socket address identifica sia il **processo** con il numero di porta che l'**host** con l'indirizzo IP, a ogni porta corrisponde un processo.

Riassumendo: dello strato applicativo fanno parte le **applicazioni di rete**, composte da **processi distribuiti** su macchine diverse che comunicano tra di loro mediante il protocollo proprio dello strato applicativo. Sebbene il protocollo fornisca alle applicazioni l'illusione di comunicare direttamente, la comunicazione è in realtà garantita da tutto lo stack protocollare sottostante. In particolare, lo strato applicativo dipende dai servizi offerti dallo **strato di trasporto** che gli è immediatamente sottostante. I due strati comunicano tramite una **API** che si chiama **socket**.

A seconda del **servizio di trasporto richiesto** dall'applicazione è possibile che si trovino in uso **protocolli di trasporto diversi** tra i quali: **TCP** e **UDP**. *(vedi sez. 2.2)*

Applicazione	Tolleranza alla perdita di 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	Sì	Audio: da 5 Kbps a 1 Mbps Video: da 10 Kbps a 5 Mbps	Sì, centinaia di ms
Audio/video memorizzati	Sì	Come sopra	Sì, pochi secondi
Giochi interattivi	Sì	Fino a pochi Kbps	Sì, centinaia di ms
Messaggistica istantanea	No	Variabile	Sì e no

Figure 21: In figura le caratteristiche di alcune delle principali applicazioni di rete. Non tutte le applicazioni sono uguali. Tipicamente la telefonia di internet usa il protocollo UDP.

6.1.1 Web

Il **web** è formato da risorse indirizzate da **URL**, acronimo di **uniform resource locator**. Generalmente queste risorse sono **pagine web**, formate da altri oggetti referenziati *(e.g. altre pagine web, immagini ecc...)*. Lo **user agent** o **client** per il web è chiamato **browser** ed il **server** è chiamato **web server**.

Definizione 6.2. Una **URI** o Uniform Resource Identifier è una stringa compatta di caratteri che identifica una risorsa fisica o astratta.

Le **URL** sono un **sottoinsieme** delle URI e identificano le risorse tramite la loro posizione all'interno della rete. Le **URN** acronimo di **uniform resource name** sono un altro sottoinsieme delle URI la cui funzione è di rimanere **globalmente uniche** e **persistenti** anche quando le risorse da loro puntate cessano di esistere o non sono più disponibili. La **sintassi** delle URI è organizzata **gerarchicamente** e i componenti sono disposti in ordine di importanza da sinistra verso destra.

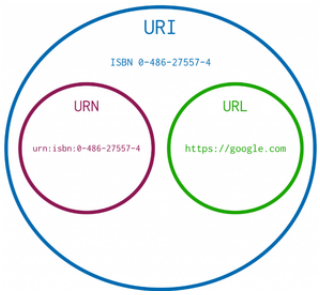


Figure 22: URN e URL non sono altro che specializzazioni delle URI.

Le **sintassi** della URI:

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

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

- **scheme**: è **obbligatorio**, definisce lo **spazio dei nomi** della risorsa.
- **authority**: indica il **nome di dominio** di un host *(reg_name)* o il suo **indirizzo IP** in notazione decimale puntata. Tipicamente identifica un computer sulla rete.
- **path**: contiene **dati** specifici per l'authority (o per lo scheme) e **identifica** la risorsa **nel contesto** di quel namespace e di quell'authority.

Le URI possono essere **assolute** o **relative**, una **URI assoluta** identifica una risorsa **indipendentemente** dal contesto in cui è usata. Una **URI relativa** identifica una risorsa in relazione ad un'altra URL, **è priva di schema e di authority**, non viaggia sulla rete ed è interpretata dal browser in relazione al documento di partenza.
(e.g. http://www.w3.org/pub/WWW/TheProject.html oppure /pub/WWW/TheProject.html sotto l'host: www.w3.org).

6.1.2 HTTP Protocol

Il protocollo **HTTP** è un protocollo di tipo **request/response**: il client inizia la connessione inviando un messaggio di **request** al server che a sua volta invia una **response**. Si tratta di un **protocollo generico**, poichè non dipende dal formato delle risorse, e **stateless** poichè **le coppie richiesta/risposta sono indipendenti l'una dall'altra**. In **HTTP 1.0** dopo la prima coppia di richiesta/risposta la connessione viene **terminata** mentre in **HTTP 1.1** si **procede** con un'altra coppia.
Lo **schema http** è usato per accedere alle risorse attraverso il protocollo HTTP.

Si riporta di seguito la **sintassi** per le URL http:

$$http : // < host > [:< port >][< path >]$$

- **host**: è un **host domain di Internet** valido oppure un indirizzo IP in forma decimale puntata.
- **port**: **è un intero**, se la porta è vuota o non è indicata si usa automaticamente la porta 80.
- **path**: specifica la **request URI** *(vedi seguito)*.

N.B. Il protocollo HTTP utilizza il protocollo TCP tramite la sua API.

```
//esempio client
c = TCPopen("131.115.7.24", 80);
TCPsend(c,"GET /index.html");
d = TCPreceive(c);

//esempio server
p = TCPbind(80); //where to wait for connections
d = TCPaccept(p); //waiting for connections
r = TCPreceive(d);
...
TCPsend(d,pag);
TCPclose(d);
```

Definizione 6.3. Una **connessione http** è un circuito logico di livello di trasporto stabilito tra due programmi applicativi per comunicare tra loro.

Una **connessione** può essere:

- **Non persistente***(http1.0: RFC 1945)*: si parla di connessione non persistente quando **per ogni richiesta** del **client** viene instaurata **una nuova connessione** con il **server**. Ciò aumenta il **carico** su quest ultimo e potrebbero verificarsi fenomeni di **congestione**. *Infatti per visualizzare le n immagini di un sito il client invia di seguito n richieste al server.*
- **Persistente***(http1.1: RFC 2616)*: la connessione è appunto **persistente**. Nello **standard** è specificato un meccanismo che consente al server di **chiudere** la connessione TCP su richiesta del client. *(CONNECTION = CLOSE, in GENERAL HEADER, vedi seguito.)*
  N.B. Una volta che la chiusura della connessione è stata segnalata, il client **non deve** inviare altre richieste.

Esempio:
Supponiamo che l'utente digiti la seguente URL:

www.someSchool.edu/someDepartment/home.index

Con **una connessione non persistente**, in ordine temporale succedono le seguenti cose:

1. Il **client http** invia una richiesta di connessione **TCP** verso il server http al **www.someSchool.edu** *(La porta 80 è usata di default per il server http.)*

2. Il **server http** dell'host **www.someSchool.edu**, che aspetta le richieste di connessione **TCP** alla **porta 80**, **accetta la richiesta di connessione** e notifica il client.

3. Il **client http** invia quindi un **messaggio di richiesta**, contenente la URL.

4. Il **server http** riceve il messaggio di richiesta, compila un **messaggio di risposta** con l'oggetto richiesto indicato dalla URL: **someDepartment/home.index**, invia il messaggio e in seguito **chiude** la connessione.

5. Il **client http** riceve il messaggio di risposta che contiene il file html e lo visualizza.

Si ricorda che la ricezione e la trasmissione di tutti i messaggi elencati poco sopra avviene tramite socket.
Supponiamo ora che la URL contenga dei riferimenti a 10 immagini, in tal caso per ogni riferimento si devono ripetere tutti i passaggi definiti sopra. Salta immediatamente all'occhio la **scarsa efficienza** della procedura.

Con **una connessione persistente** invece il server lascerebbe aperta la connessione TCP dopo aver spedito la prima risposta e vi riceverebbe quindi le richieste successive. La connessione verrebbe chiusa dal server quando specificato nell'header di un messaggio **inviatogli dal client**, oppure alla mancata ricezione di richieste per un certo intervallo di tempo *(time out)*.

Un ulteriore miglioramento delle prestazioni, si otterrebbe con una tecnica di **pipelining**, consistente nell'invio da parte del client di molteplici richieste senza aspettarne la ricezione da parte del server.
Il **server deve** tuttavia inviare le risposte nello stesso ordine in cui sono state ricevute le richieste e il **client non** può inviare **in pipeline** richieste che usano metodi HTTP **non idempotenti**. Ma cos'è un **metodo idempotente**?

Definizione 6.4. Un **metodo idempotente** è un metodo tale che il suo effetto collaterale sulla risorsa è lo **stesso** per N o 1 richieste identiche che ne fanno uso. *(e.g. GET, HEAD, PUT, DELETE, OPTIONS, TRACE)*

Definizione 6.5. Un **metodo safe** è un metodo che non produce effetti collaterali sulle risorse. *(e.g. non le modificano: GET, HEAD, OPTIONS, TRACE...)*

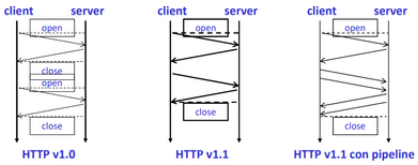


Figure 23: In figura sono rappresentate le differenze viste prima.

Come sono strutturati i messaggi HTTP?

Riportiamo di seguito la **struttura** di un generico messaggio http:
`Request = Request-Line` *o* `Response = Status-Line` *per i messaggi di risposta.*

```
*( general-header
| request-header o response-header
| entity-header )
CRLF
[ message-body ]
```

La prima riga o **start line** distingue i messaggi di request dai messaggi di response.

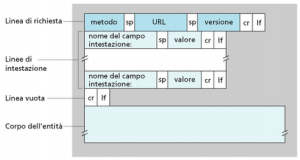


Figure 24: In figura un messaggio di richiesta.

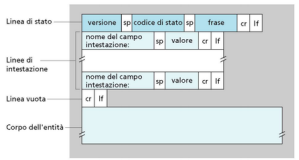


Figure 25: In figura un messaggio di risposta.

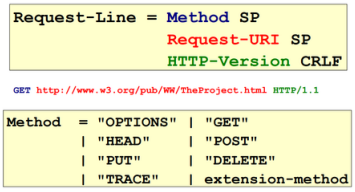


Figure 26: La struttura della **request line**. Il campo **method** è case sensitive ed indica l'operazioe da eseguire sulla risorsa identificata dalla URI. Il metodo **POST** serve per inviare dal client al server le informazioni inserite nel body del messaggio, **PUT** è usato dal client per chiedere al server di creare o modificare una risorsa, **DELETE** per cancellarla.

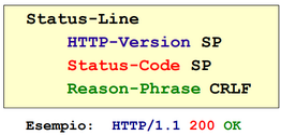


Figure 27: La **status-Line** è la prima riga del messaggio di risposta. Lo **status code** è un numero di 3 cifre, indica il risultato del tentativo di soddisfare la richiesta del client. La **reason-phrase** dà una breve descrizione testuale dello status-code. Lo status-code è rivolto alla macchina mentre la reason-phrase all'utente umano.

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

- **General Headers**: sono relativi alla **trasmissione** e si applicano a **tutto il messaggio**. *(e.g.* ***Date, Connection:*** *usato dal mittente per specificare delle opzioni desiderate per la connessione, ad esempio close.* ***Transfer-encoding****: specifica se e quali trasformazioni sono state applicate al corpo del messaggio ad esempio gzip, chunked ecc.* ***Cache control****: indica quale tipologia di cache può memorizzare il messaggio, può essere* ***public****,* ***private*** *o* ***no-cache****.)*
- **Entity Headers**: sono **metadati** relativi all'**entità trasmessa**. Ogni entity è costituita da un **entity body** e da una serie di **entity headers** che ne definiscono contenuto e proprietà. *(e.g. Content Lenght)*
- **Request Headers**: sono relativi alla **richiesta**. Supportano il **content negotiation**, il processo tramite cui il server sceglie la forma "giusta" del contenuto richiesto dal client. *(Specificano da chi è fatta la richiesta, a chi viene fatta, che tipo di risposte il client è in grado di accettare, l'autorizzazione, ecc.)*
- **Response Headers**: sono relativi al **messaggio di risposta**.

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

Figure 28: L'elenco dei general headers.

```
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
```

Figure 29: L'elenco dei request headers. **Accept** specifica quali tipi di media sono accettati in risposta dal client, **Accept-Charset**, quali caratteri e **Accept-Encoding** quali formati.

```
"100" - Continue                "101" - Switching Protocols

"200" - OK                      "201" - Created
"202" - Accepted                "203" - Non-Authoritative Information
"204" - No Content              "205" - Reset Content
"206" - Partial Content

"300" - Multiple Choices        "301" - Moved Permanently
"302" - Moved Temporarily       "303" - See Other
"304" - Not Modified            "305" - Use Proxy

"400" - Bad Request             "401" - Unauthorized
"402" - Payment Required        "403" - Forbidden
"404" - Not Found               "405" - Method Not Allowed
"406" - Not Acceptable          "407" - Proxy Authentication Required
"408" - Request Time-out        "409" - Conflict
"410" - Gone                    "411" - Length Required
"412" - Precondition Failed     "413" - Request Entity Too Large
"414" - Request-URI Too Large   "415" - Unsupported Media Type

"500" - Internal Server Error   "501" - Not Implemented
"502" - Bad Gateway             "503" - Service Unavailable
"504" - Gateway Time-out        "505" - HTTP Version not supported
```

Figure 30: In figura l'elenco degli status-codes e il loro significato.

```
response-header = Age
                | Location
                | Proxy-Authenticate
                | Public
                | Retry-After
                | Server
                | Vary
                | Warning
                | WWW-Authenticate
```

Figure 31: I campi del **response-header** consentono al server di inviare informazioni aggiuntive che non possono essere poste nella status-line. Ad esempio: **Age**: indica quanto tempo è trascorso, in secondi, tra l'invio della risposta da parte del mittente e la generazione della stessa all'origine. **Location** é usato per reindirizzare il client verso un altra URI per completare la sua richiesta o per fornire una nuova risorsa. Il campo **Server** contiene informazioni sul software usato dal server di origine per portare a compimento la richiesta del client.

Fin'ora abbiamo mostrato come il client accede alle risorse del web inviando richieste al server. Cosa succederebbe se, ad esempio, un client richiedesse continuamente la **stessa** risorsa? Potremmo evitare di inoltrare richieste tutte uguali al server?
Sì, ci viene in aiuto il **web caching**. L'obbiettivo del web caching è di soddisfare le richieste del client **senza** contattare il server. Funziona memorizzando **copie temporanee** di risorse web, servendole poi al client così da **ridurre** l'uso di **banda**, limitare il **workload** sul server e di conseguenza **diminuire tempo di risposta** verso gli altri clients.
Consideriamo due modelli possibili di web caching:

- **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. Successive richieste alla stessa URI possono essere servite direttamente dal proxy senza inoltrare la richiesta al server. Sta poi all' utente configurare il browser per consentire gli accessi Web via proxy.

Cos'è un proxy?

Definizione 6.6. Un **proxy** è un **programma intermediario** che agisce sia da **server** che da **client**, **inviando** e **servendo richieste** per altri clients. Le richieste sono gestite internamente oppure sono girate a server terzi.

Resta ora da chiarire come è possibile che, sebbene abbiamo definito il protocollo HTTP come **stateless**, alcuni siti, *riconoscano* gli utenti.
Sebbene infatti il protocollo HTTP sia a tutti gli effetti stateless, le applicazioni web **non** lo sono. Come fare quindi a conciliare queste due realtà?

Con i coockies!

Definizione 6.7. I **coockies** sono stringhe di testo contenenti informazioni relative all'utente.

I **coockies** funzionano nel seguente modo:

1. Un client invia al server una richiesta HTTP.
2. Il server invia al client la risposta HTTP e in più una linea **set-cookie: 1678453**. *(esempio fittizio)*
3. Il client **memorizza** il cookie in un file e lo associa al server. Lo aggiungerà con la seguente linea: **cookie: 1678453** a tutte le sue successive richieste.
4. Alla successiva richiesta da parte del client, il server **risalirà** tramite il cookie alle informazioni ad esso **associate**.

6.1.3 TELNET

TELNET, acronimo per **TE**rmina**L NET**work, è un **protocollo client-server** che fornisce una comunicazione **interattiva ed orientata al testo** tra due macchine, è basato sul trasporto **connection-oriented** ed usa il protocollo **TCP**. Consente all'utente di effettuare una sessione di **login** in una macchina remota e quindi di utilizzarne il terminale. L'utente che si autentica tramite login remoto ha accesso infatti a tutti i **comandi** e ai **programmi** disponibili su di essa. **I comandi vengono eseguiti come se l'utente li digitasse dalla tastiera stessa della macchina.** *Per estensione,* `telnet` *è anche il nome di un programma usato per avviare una sessione TELNET verso un host remoto.* **TELNET** infatti include due programmi: un programma **TELNET client** e un programma **TELNET server**. **TELNET client** (`telnet`) interagisce con l'utente sulla macchina locale e scambia messaggi con **TELNET server**. Riportiamo di seguito una semplificazione del funzionamento:

1. L'utente, dalla **propria** macchina **locale** (**TELNET client**), stabilisce una connessione **TCP**, persistente per tutta la durata della sessione, con una **macchina remota (TELNET server)** alla **porta 23** e vi si **autentifica**.

2. Tutte le **battute** dei **tasti** della macchina locale vengono **trasmesse** dal client alla **macchina remota**.

3. La **macchina remota** accetta la connessione TCP e il **TELNET server** trasmette i dati al sistema operativo locale.

4. L'**output** della **macchina remota** viene quindi **ricevuto** e **trasmesso** sul terminale dell'utente.

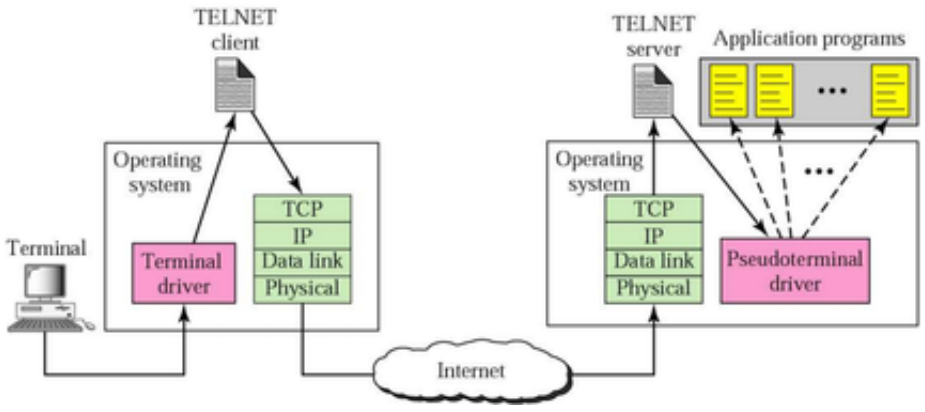


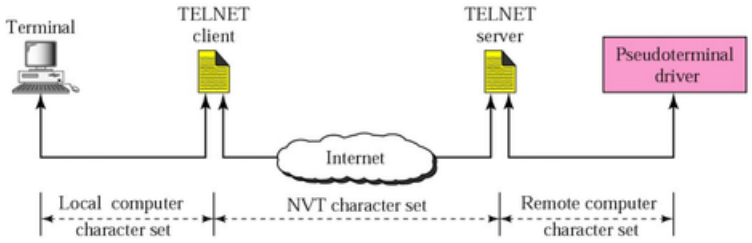
Figure 32: In figura il funzionamento di TELNET.

N.B. TELNET è un protocollo STATEFUL.

Problema: come operare con sistemi operativi diversi?

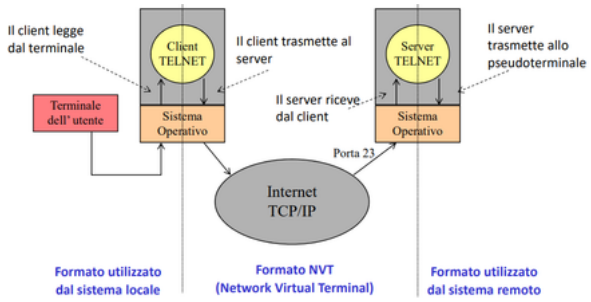
I terminali infatti non sono tutti uguali, possono differire gli uni dagli altri per il **set** e la **codifica** dei caratteri, per la **lunghezza** della **linea** e della **pagina** e per i **tasti funzione** individuati da diverse sequenze di caratteri.

Una **soluzione** è stata trovata mediante la definizione di un **Network Virtual Terminal (NVT) che definisce un set di caratteri universali**. L'NVT è quindi un dispositivo "immaginario" che fornisce una rappresentazione astratta di un terminale. Gli host, sia client che server, **traducono** le loro caratteristiche **locali tramite il set universale** così da apparire **in rete** come un NVT e assumono che l'host remoto sia anch'esso un NVT. Questa operazione di traduzione è ovviamente **reversibile**.



Il funzionamento di TELNET, con l'aggiunta della trasformazioni intermedie sopra descritte diventa:

1. ...
2. Tutte le **battute** dei **tasti** della macchina locale vengono **trasformate in NVT e successivamente trasmesse dal TELNET client** alla **macchina remota**.
3. La **macchina remota** accetta la connessione TCP, il **TELNET server traduce** da NVT allo standard del sistema operativo remoto e infine gli trasmette i dati ricevuti.
4. ...



N.B. NVT invia i caratteri di controllo prioritariamente con TCP URGENT.

Concludiamo con qualche tecnicismo:

I terminali NVT si scambiano dati in formato **7-bit US-ASCII** e adottano inoltre un approccio **in-band signaling**, ovvero dati e comandi viaggiano sullo **stesso canale**. Per distinguere i due tipi di informazione si usa la seguente convenzione: ogni carattere è inviato come un **ottetto di bit** con **il primo bit settato a zero**. I caratteri, in notazione decimale, vanno dal numero 0 al 127. I comandi invece sono identificati tramite ottetti speciali di 1, in n. d. vanno dal numero 240 al numero 254, per distinguerli dai dati sono sempre preceduti da un carattere speciale: **IAC o Interpret As Command** identificato, sempre in n. d., dal numero 255. Essenzialmente quindi si usa un canale di 8 bit per scambiare dati di tipo 7 bit ASCII. I messaggi scambiati durante la fase iniziale della comunicazione, ovvero prima del login, sono **messaggi di controllo** e costituiscono la **Telnet Option Negotiation**, in sostanza sono usati per scambiare **informazioni** sulle **caratteristiche** degli **host**.

Comando	**Codifica decimale**	**Significato**
IAC	255	Interpret as command
EL	248	Erase line
EC	247	Erase character
IP	244	Interrupt process
EOR	239	End of record

Ma quindi NVT conviene?

Rispondiamo alla domanda mostrando dei semplici calcoli:

Supponiamo che N sia il numero di sistemi distinti che si vogliono far interoperare:

- **Senza** l'uso di NVT, si necessita la scrittura di **N-1** client per ogni sistema (N-1 TELNET-clients che traducano negli N-1 standards dei sistemi), e 1 TELNET-server per ogni sistema:
  In totale si avranno $N \times (N-1) + N$ applicativi.
- **Usando** NVT, bisogna scrivere 1 TELNET-server per ogni sistema e N TELNET-client (1 per ogni sistema che traduca dallo standard di sistema allo standard di NVT):
  Avremo quindi $N + N = 2N$ applicativi.

Quindi per $N > 2$ **conviene** usare NVT!

6.1.4 SSH

TELNET non possiede **alcuna** misura di sicurezza poichè è stato progettato per l'uso su **reti private**, trasmette tutto in chiaro, anche le **password**! Con l'avvento delle reti pubbliche però si è reso necessario prendere delle contromisure.
SSH o **S**ecure **SH**ell è un'applicazione nata per sostituire TELNET.

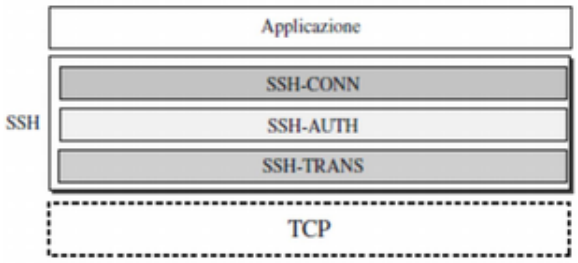


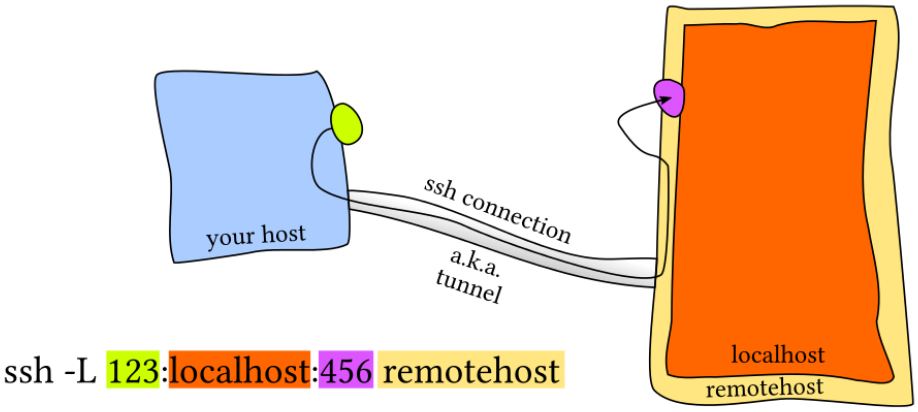
Figure 33: Le componenti di SSH

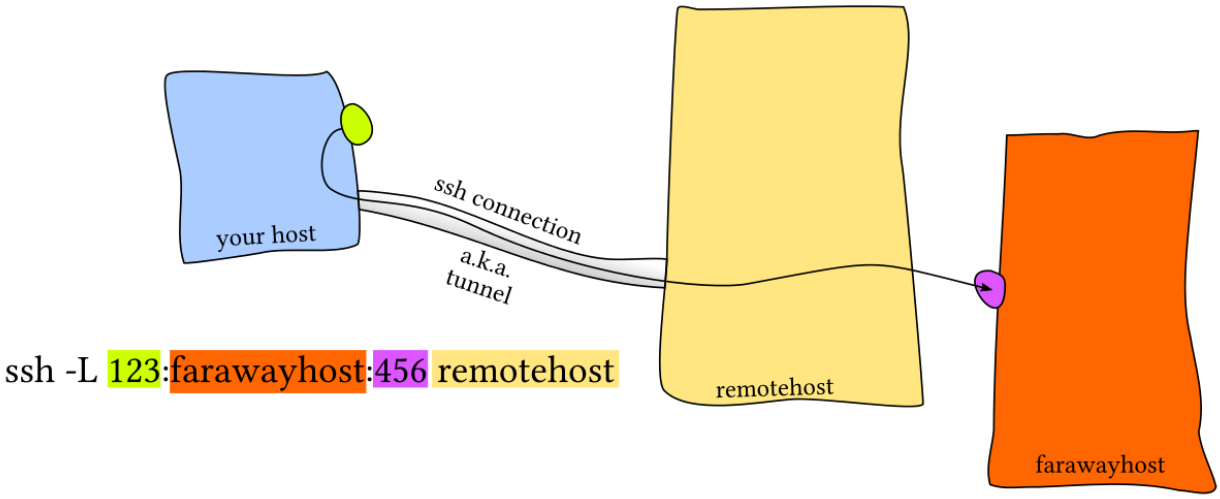
Il **protocollo** di livello applicazione **SSH** è composto da **tre** diverse componenti: **SSH-TRANS, SSH-AUTH e SSH-CON**. La componente **SSH-TRANS** o **SSH-Transport Layer Protocol** costruisce un canale di comunicazione sicuro, tramite tecniche crittografiche, sfruttando la connessione offerta da TCP. Il **protocollo TCP** trasmette tutte le informazioni in chiaro, **non** è quindi in grado **da solo** di garantire **privacy** e **confidenza**. *SSH-TRANS è anche in grado di riconoscere se il server a cui ci stiamo connettendo è quello autentico o meno.* La componente **SSH-AUTH** si occupa di autenticare il client. *Sono disponibili altre tecniche di autenticazione oltre alla classica con username e password tra le quali l'accesso basato su coppie di chiavi crittografiche.* Infine la componente **SSH-CONN** sfruttando i servizi offerti dalle altre due componenti offre i servizi di **terminale**, **trasferimento file**, **creazione di tunnel** ecc...
SSH offre quindi **molti più servizi di TELNET**.

TCP Port Forwarding:

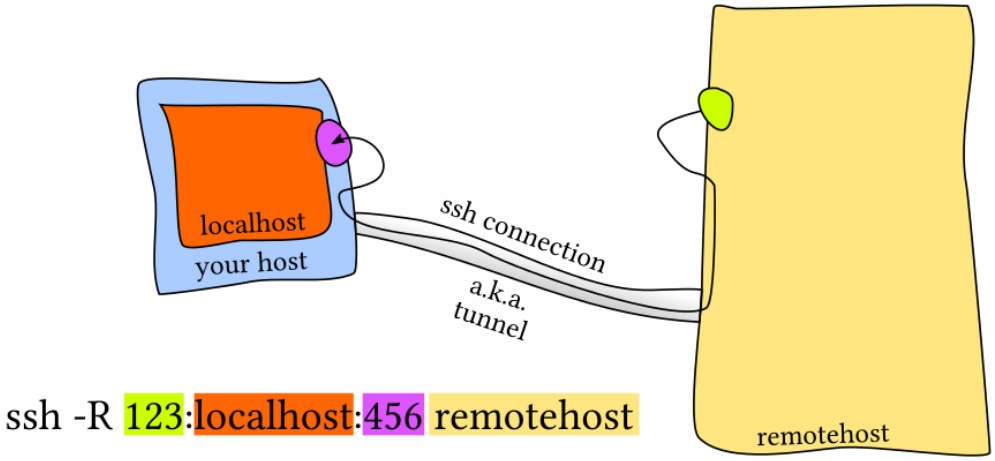
TCP Port Forwarding è un meccanismo che permette di **creare** un **canale di comunicazione sicuro** attraverso il quale **veicolare qualsiasi** tipo di **connessione TCP**. Opera creando un canale di comunicazione **cifrato** tra la **porta** all'**indirizzo remoto** a cui ci si vuole collegare e una **porta locale** libera. le applicazioni punteranno il collegamento alla porta locale e la connessione verrà inoltrata automaticamente all'host remoto tramite un canale sicuro.
Segue una spiegazione delle differenze tra **local port forwarding** e **remote port forwarding**.

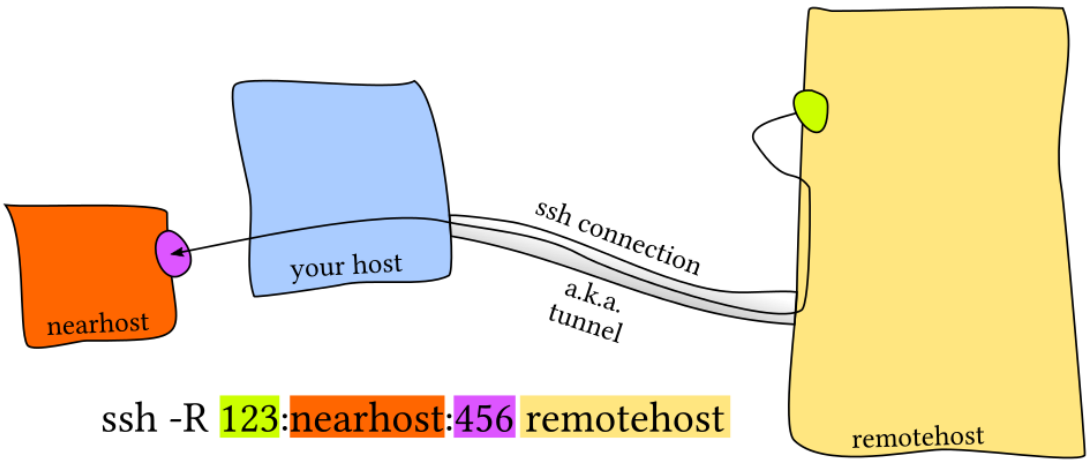
- **Local**: `ssh -L` specifica che il traffico sulla porta indicata della macchina locale deve essere reindirizzato verso la porta indicata della macchina remota.
  e.g. `ssh -L sourcePort:forwardToHost:onPort connectToHost` significa: connettiti via ssh a **connectToHost**, e inoltra tutti i tentativi di connessione che arrivano alla porta locale **sourcePort** verso la porta **onPort** della macchina chiamata **forwardToHost**, che si raggiunge tramite la macchina chiamata a sua volta **connectToHost**.





- **Remote**: `ssh -R` specifica che il traffico sulla porta indicata della macchina remota deve essere reindirizzato verso la porta indicata della macchina locale.
  e.g. `ssh -R sourcePort:forwardToHost:onPort connectToHost` significa: connettiti via ssh a **connectToHost**, e inoltra tutti i tentativi di connessione che arrivanao alla porta remota **sourcePort** verso la porta **onPort** della macchina **forwardToHost**, che si raggiunge tramite la tua macchina locale.

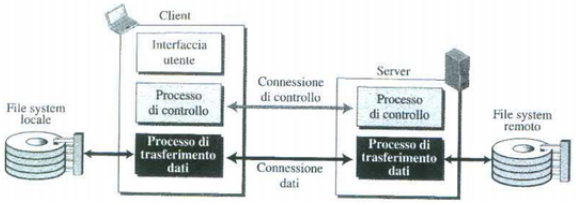




N.B. SSH è un protocollo STATEFUL.

6.1.5 FTP

FTP acronimo per **F**ile **T**ransfer **P**rotocol, è un **protocollo** per il **trasferimento** di **dati** tra due host di una rete, è lo **standard** per il **trasferimento** di file **offerto da TCP/IP**. Adotta il modello **client-server**: il **client** richiede il trasferimento di un file che può consistere sia nell'**acquisizione** di una copia locale modificabile sia nell'eventuale **trasferimento** della copia modificata sull'host remoto (**server**).



Il **client** ha **tre** componenti: **interfaccia utente**, **processo di controllo** e **processo di trasferimento dati** mentre il **server** remoto ne ha solo **due**: **processo di controllo** e **processo di trasferimento dati**. La **separazione** del trasferimento dei dati da quello dei comandi rende il protocollo FTP **efficiente**. Infatti la **connessione di controllo** usa **regole semplici** così da ridurre lo scambio delle informazioni a **una riga di comando** e **una di risposta per ogni interazione**. Mentre la **connessione dati usa regole più complicate** a causa della varietà delle informazioni che vi transitano. **FTP** offre funzionalità **aggiuntive** oltre al semplice trasferimento di dati, infatti **mette a disposizione**:

- **Accesso interattivo**: l'utente può **navigare**, **cambiare** e **modificare** l'albero di directory nel file system dell'host remoto.
- **Specificazione del formato dei dati da trasferire** (e.g. file di testo o file binari)
- **Autenticazione**: il client può **autenticarsi** con username e password.

Poco sopra abbiamo accennato al fatto che **FTP** prevede l'instaurazione di **due tipi di connessione tra il client e il server** ovvero:

1. **Control connection**: prevede uno scambio di comandi e risposte tra client e server. Segue il protocollo TELNET e **rimane aperta per l'intera durata della sessione interattiva**. Si usa la **porta 21** del **server** e la codifica standard **NVT ASCII**.
2. **Data connection**: prevede il trasferimento di dati mediante procedure e la **specifica dei tipi**. I dati trasferiti possono essere parte di un file, un file o un set di file. **Viene aperta e chiusa per ogni singolo scambio**. Per lo scambio di dati il **server** usa la **porta 20**.

La **Data Connection non segue il protocollo TELNET** e la sua apertura avviene secondo uno schema completamente diverso, ovvero:

1. Il **client**, non il server, effettua un'**apertura passiva** usando una **porta effimera** e resta in attesa di connessione, viene fatto dal client poichè è tale processo che invierà i comandi, tramite la connessione di controllo, per il trasferimento dei file.

2. Il client **invia** questo **numero** di **porta** al **server** per mezzo del comando PORT.

3. Il **server**, ricevuto il numero di porta, **effettua** un'**apertura attiva**, ovvero apre la connessione, usando la propria **porta** nota **20** e quella effimera offerta dal client.

Per effettuare il trasferimento dei file inoltre, il client deve **definire** il **tipo** di file, la **struttura dati** e la **modalità di trasmissione** al fine di risolvere i problemi di eterogeneità tra client e server, va infatti ricordato che programma client e programma server sono su macchine **diverse**. Questo scambio di informazioni avviene mediante la **connessione di controllo**.

FTP è quindi un protocollo **STATEFUL** poichè il **server deve tener traccia dello stato dell'utente**: bisogna tenere conto infatti, tra le altre cose, della directory del file system remoto in cui si trova l'utente!

Concludiamo menzionando il fatto che esistono server che supportano connessioni FTP **senza autenticazione** (**Anonymous FTP**). Tipicamente consentono di accedere **solo** ad una parte del file system e permettono **solo** un subset di operazioni (e.g. la PUT **non** è permessa). Di solito **si usa** un **username comune** (solitamente "ftp" or "anonymous") e una **password qualsiasi** (e.g. l'indirizzo email dell'utente). *Erano usate per distribuire a un pubblico dei file senza dover generare numerosi username e password.*

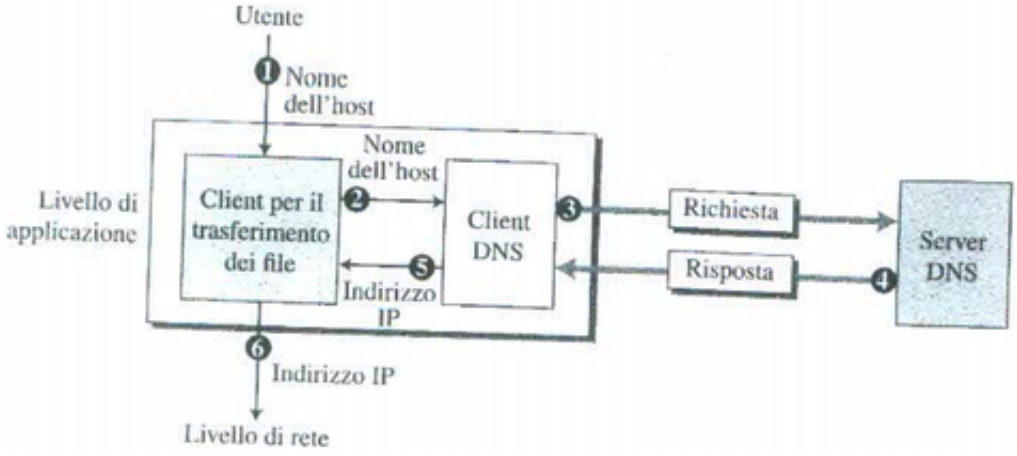
Comandi di controllo	**Significato**
USER	username
PASS	password
LIST	elenca i file della directory corrente
NLST	richiede elenco file e directory (ls)
RETR filename	recupera (get) un file dalla directory corrente
STOR filename	memorizza (put) un file nell'host remoto
ABOR	interrompe l'ultimo comando ed i trasferimenti in corso
PORT	indirizzo e numero di porta del client
SYST	il server restituisce il tipo di sistema
QUIT	(quit) chiude la connessione

6.1.6 DNS

Sappiamo oramai che i dispositivi connessi in rete vengono individuati dai protocolli TCP/IP mediante i loro **indirizzi IP**, gli utenti, d'altro canto, preferiscono usare dei **nomi** invece che degli indirizzi numerici. Un **nome** identifica un **oggetto** mentre un **indirizzo** specifica **dove** l'oggetto è situato.

Come fare per associare i nomi agli indirizzi?

Agli albori di Internet l'associazione tra nomi logici e indirizzi IP era statica! Tutti i nomi *logici* e i relativi indirizzi IP erano contenuti in un file chiamato **host file** e periodicamente tutti gli host ne prelevavano una versione aggiornata, chiamata a sua volta **master host file**, da un server ufficiale.
A noi utenti moderni però dovrebbe immediatamente saltare all'occhio la seguente **problematica**: le **dimensioni attuali di Internet** rendono questo approccio **impraticabile**. Non sarebbe infatti possibile che ogni host possegga una **copia aggiornata** di un elenco del genere, in più la dimensione di questo sarebbe sterminata, per non parlare del **volume di traffico** sul server ufficiale, il pericolo dovuto alla presenza di un **unico punto di fallimento** e l'**impossibilità di scalare** di questo sistema.
Fu così che all'inizio degli anni '80 venne ideato il **DNS** o **Domain Name System**. L'**idea centrale** è di **suddividere** la sconfinata mole di associazioni nome-indirizzo e **distribuirne** le varie parti su **calcolatori sparsi per il mondo**. *Come funziona? Così:*



Supponiamo che un utente utilizzi un client di trasferimento file per accedere a un file su un server. L'utente conoscerà solo il **nome** del server e.g. *cheneso.com*, lo **stack TCP/IP** invece ha bisogno dell'**indirizzo IP** del server per stabilire una connessione. Alla pagina seguente troviamo i sei passi necessari per **associare** l'**indirizzo** IP al **nome** del server.

1. L'**utente** comunica il nome del server al client di trasferimento file.
2. **Il client di trasferimento file** trasmette il nome del server al **client DNS**.
3. **Qualsiasi computer una volta avviato conosce l'indirizzo IP di un server DNS**, il **client DNS** invia dunque al **server DNS** la richiesta di traduzione del nome simbolico del server.
4. Il **server DNS** risponde con l'indirizzo IP del server desiderato.
5. Il **client DNS** comunica al **client di trasferimento file** l'indirizzo IP del server.
6. Il **client di trasferimento file utilizza** così l'**indirizzo IP ricevuto** per accedere al server.

Per far sì che questo meccanismo funzioni bisogna **eliminare le ambiguità sui nomi** e per far ciò si è definito uno **spazio dei nomi**. Lo **spazio dei nomi** ha una **struttura gerarchica** per una ragione principale: se tutti i nomi fossero composti da una sola stringa alfanumerica servirebbe un'**autorità centrale** che controllasse l'unicità di ogni singolo nome. Una **struttura gerarchica** consente invece di avere nomi composti da diverse parti e di **delegare il controllo** su ciascuna parte a enti o società diverse, **decentralizzando** in tal modo il processo di controllo.

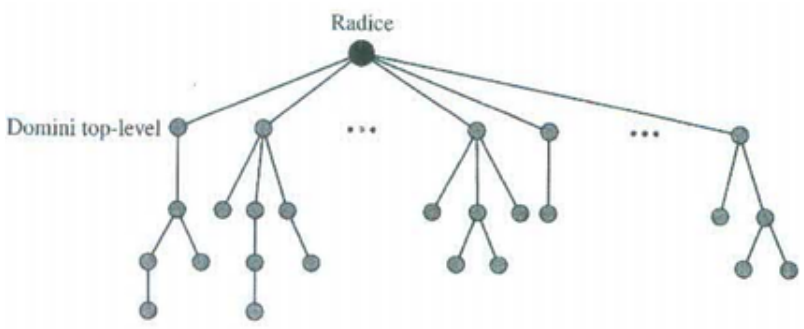


Figure 34: In figura lo spazio dei nomi di dominio.

Nello **spazio dei nomi di dominio** i nomi hanno una struttura ad **albero** con la radice in cima e un numero di livelli compreso tra 0 e 127. Ogni nodo è individuato da un'**etichetta** costituita da **massimo 63 caratteri** (*la radice ha l'etichetta vuota*), tutti i nodi collegati a uno stesso nodo da rami diversi hanno etichette **diverse**, ciò garantisce l'**univocità** dei nomi. Ogni nodo dell'albero ha inoltre un **nome di dominio**, che, se letto da sinistra verso destra, è costituito da tutte le etichette, separate da punti, di tutti i nodi a partire dal nodo stesso fino alla radice, la cui etichetta è la stringa **nulla**. Immediatamente sotto la radice si trovano i **domini top-level**.

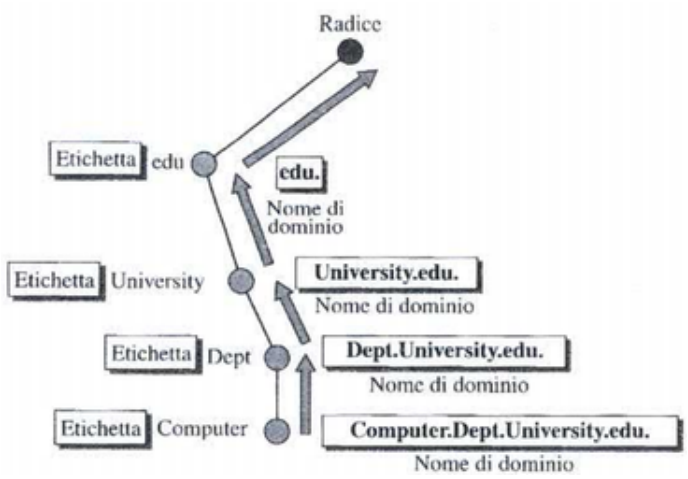


Figure 35: In figura le etichette e i nomi di dominio.

Definizione 6.8. Un **dominio** è un **sottoalbero** dello spazio dei nomi che viene identificato dal **nome di dominio** della sua radice.

Ma dove sono contenute le informazioni relative allo spazio dei nomi di dominio?

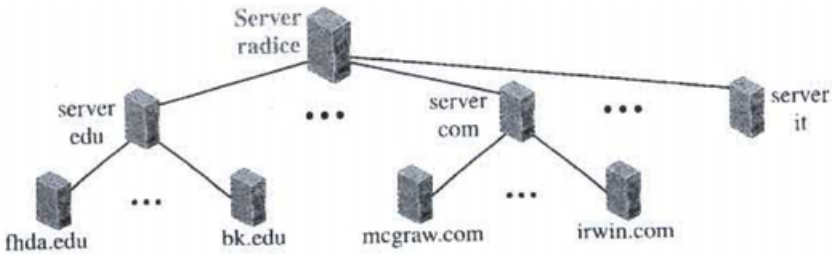


Figure 36: In figura la gerarchia dei name server.

All'interno dei **DNS-Servers** o **name servers**. L'intero spazio dei nomi è stato **diviso** in diversi domini, differenziati al **top-level**, e i domini così ottenuti sono stati divisi a loro volta ottenendo dei sotto domini. Ogni **name server** è responsabile di un dominio o di un sottodominio. Vi è quindi una **gerarchia di server**. Una **zona è tutto ciò di cui è responsabile un server**, se un server è responsabile di un dominio e non effettua suddivisioni in sottodomini, allora la sua zona e il suo dominio **coincidono**. Se invece il server suddivide il proprio dominio in sottodomini la sua zona e il suo dominio **differiscono**. Ogni name-server ha un database chiamato **file di zona** contenente le informazioni su tutti i nodi che ricadono nella **sua zona** di competenza.

Un **root server** è un server che ha per zona l'**intero albero dello spazio dei nomi**.
Solitamente i **root server** si limitano a immagazzinare riferimenti relativi ad altri server e **delegano** loro tutte le responsabilità.
Esempio di top-level domains:

Dominio	**Uso**
com	organizzazioni commerciali
edu	istituti di istruzione
mil	gruppi militari
gov	istituzioni governative americane
net	principali centri di supporto alla rete
org	organizzazioni diverse dalle precedenti
it, uk, us, fr, ecc.	codice geografico per nazioni

I **server DNS** possono essere **primari** o **secondari**.

- Un server **primario** possiede sul disco e aggiorna il **file di zona** relativo alla zona sotto la sua responsabilità.
- Un server **secondario** riceve le informazioni relative a una zona da un **server primario**.

I **server primari** e **secondari** hanno la medesima autorità sulla loro zona di competenza, è bene specificare che **un server può essere primario per una zona e secondario per un altra**, **primario** e **secondario** sono quindi aggettivi **relativi alla zona**. Si noti inoltre come l'introduzione di un server secondario in una zona porti a una duplicazione del **file di zona** che può risultare utile per eventuali guasti al **server primario**.

Come si associa l'indirizzo IP da un nome?

Il processo con cui si associa l'indirizzo IP da un nome è detto **processo di risoluzione**. Il **protocollo DNS** è progettato come protocollo **client-server**. Un host che voglia ricavare un indirizzo IP da un nome si rivolge al **programma client** detto anche **resolver** che invierà un'opportuna richiesta al **server DNS più vicino** il quale, se dispone della risposta, invierà l'indirizzo o il nome (*è infatti possibile fare il processo al contrario*), oppure inoltrerà la richiesta a un altro server o comunicherà al resolver l'indirizzo di un altro server a cui fare riferimento. Il **resolver**, ricevuta la risposta, la esaminerà per verificarne la presenza di errori e la trasmetterà al processo che l'ha effettuata. La **risoluzione** può essere **ricorsiva** o **iterativa**.

- Con la **risoluzione ricorsiva**, la **query DNS** viaggia dall'host su cui è in esecuzione il processo applicativo che ne ha fatto richiesta fino a un host che conosce l'indirizzo IP richiesto, eventualmente scalando, poi discendendo e infine percorrendo a all'indietro **l'intera gerarchia dei server DNS**. **All'host che ha richiesto la query viene inviato solo l'indirizzo IP risultante**. Il server locale ha richiesto quindi una **conversione completa**.

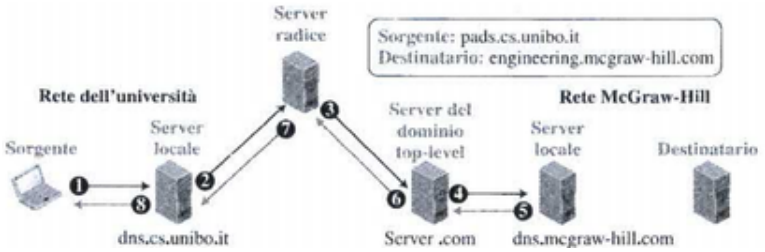


Figure 37: In figura un esempio di risoluzione ricorsiva.

- Con la **risoluzione iterativa** ogni **server-DNS** che non è in grado di risolvere la **query-DNS** dell'host risponde, **direttamente al client-DNS**, con **l'indirizzo di un altro server in grado di risolverla**. **All'host che ha richiesto la query non viene quindi inviato solo l'indirizzo IP risultante**. La **risoluzione iterativa** è supportata da **tutti** i **name server**, l'host può richiedere che venga usata la **risoluzione ricorsiva** ma potrebbe non essere disponibile.

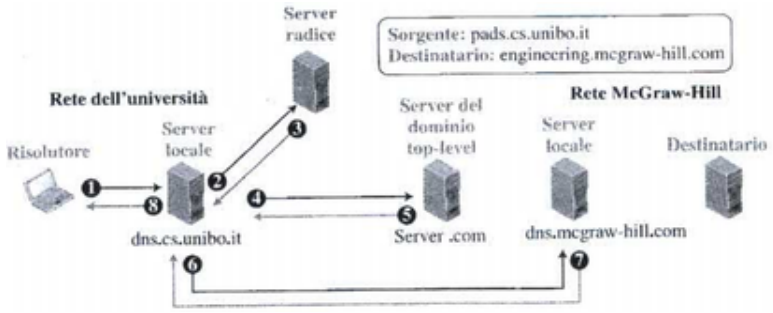


Figure 38: In figura un esempio di risoluzione iterativa.

Ogni volta che a un **server** arriva una richiesta di risoluzione di un nome, che non fa parte del suo dominio, deve cercare, all'interno del suo database, un altro server a cui inoltrare la richiesta. Ridurre questo tempo di ricerca significa ridurre il tempo di attesa della risposta e aumentare l'efficienza. Ancora una vola ci viene in aiuto il **caching**: una volta che un **server** ha appreso un' **associazione**, la inserisce in **cache**. Il server marca le risposte prese dalla propria cache come **unauthoritative** o non autorevole.

Dopo un certo lasso di tempo chiamato **TTL** o **time to live** il server cancella dalla cache l'associazione per evitare l'invio di risposte obsolete.

Ma come è fatto quindi il database di un server?

Il **database del server** non è altro che una collezione di **records** strutturati nel seguente modo:

(Nome di dominio, Tipo, Classe, TTL, Valore)

A ogni **nome di dominio**, quindi a ogni **nodo dell'albero dello spazio dei nomi di dominio**, è associato un record composto da 5 campi.

- **Nome di dominio** identifica il record della risorsa.
- **Tipo** definisce come interpretare il campo **Valore**.
- **Classe** definisce il tipo di rete, IN sta per Internet.
- **TTL** indica il numero di secondi per cui l'informazione deve essere ritenuta valida
- **Valore** contiene l'informazione memorizzata relativa al **nome di dominio**.

Di seguito i **tipi** dei **record**:

Tipo	**Interpretazione**
A	indirizzo Ipv4 a 32 bit
NS	identifica i server autoritativi di una zona
CNAME	il nome di dominio è un alias per quello ufficiale
SOA	informazioni autoritative riguardanti una zona
MX	server di posta del dominio corrente
AAAA	indirizzo Ipv6

Che protocollo di livello trasporto è usato?

Il sistema DNS può usare **sia** il **il protocollo TCP** che l'**UDP**. Il **protocollo UDP** viene usato **quando** la dimensione del messaggio di risposta è **inferiore** a **512 byte**, molto spesso infatti i **datagrammi utente UDP** non possono superare i 512 byte come dimensione massima. In caso contrario si usa il **protocollo TCP**.

Come si aggiungono nuovi domini al DNS?

Pagando i **registrar**, ovvero aziende commerciali accreditate dall'ICANN.

Come sono fatti i messaggi DNS?

I **messaggi DNS** posso essere di **due tipi**: di **interrogazione** o di **risposta** e i due tipi hanno lo **stesso** formato.

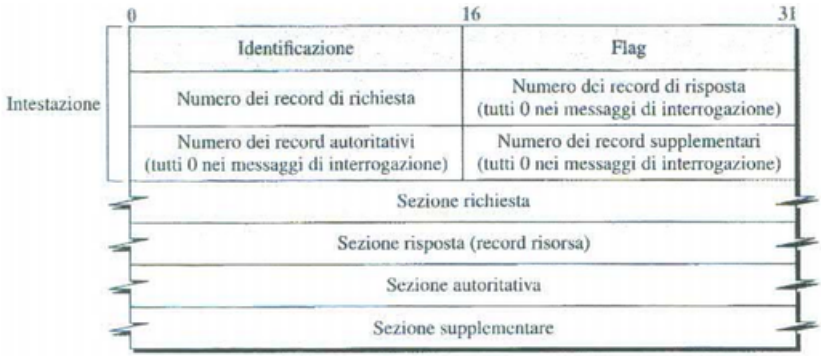


Figure 39: La struttura dei messaggi DNS, il messaggio interrogazione contiene solo la sezione richiesta mentre il messaggio di risposta contiene la sezione richiesta, la sezione risposta ed eventualmente le altre due.

Descriviamo ora brevemente i vari campi situati prima dell'**intestazione** lunga **12 bytes**:

- **Identificazione** è il campo usato dal **client** per associare la risposta all'interrogazione.
- Il campo **Flag** indica se si tratta di un messaggio di richiesta o di risposta e segnala inoltre la presenza di eventuali errori.
- I **quattro** campi successivi dell'**intestazione** specificano il numero di ciascun tipo di record presente nel messaggio.

Poi del **messaggio**:

- La **sezione di richiesta** che è inclusa nell'interrogazione e poi ripetuta anche nel messaggio di risposta è **formata** da **uno o più record di richiesta**.
- La **sezione di risposta** è presente **esclusivamente** nei messaggi di risposta consiste **in uno o più record di risorsa**
- La **sezione autoritativa** fornisce le informazioni di uno o più server autorevoli per l'interrogazione.
- La **sezione supplementare** contiene **informazioni aggiuntive** che potrebbero essere **utili** al **client DNS**.

In UNIX si può utilizzare il comando `nslookup` *per ottenere associazioni nome simbolico : indirizzo numerico.*

6.1.7 EMAIL - SMTP

La **posta elettronica** è uno dei primi servizi applicativi di Internet, la sua nascita risale infatti al 1971, quando un tale Ray Tomlinson installò su ARPANET un sistema in grado di scambiare messaggi fra le varie università. La **posta elettronica** consente agli utenti di scambiarsi **messaggi**, sebbene si siano trattate **altre applicazioni** fornenti questo servizio e.g. **HTTP** e **TCP**, il funzionamento della posta elettronica è tale da distinguersi da queste due applicazioni citate poco prima. Infatti in applicazioni come **FTP** e **HTTP** il programma **server** è **sempre attivo** e in attesa di richieste, che quando arrivano, vengono processate. Vi è quindi una richiesta e una risposta. Nella **posta elettronica** il **funzionamento** è **differente**: infatti l'invio di un messaggio è considerato una **transazione unidirezionale**, la risposta può anche non arrivare e se arriva è considerata un'altra transazione unidirezionale. Secondariamente **non avrebbe** molto **senso tenere in esecuzione continuata un programma server** in attesa che qualcuno ci invii un messaggio, potremmo ragionevolmente voler spegnere il computer nell'attesa. Ciò implica che l'idea di modello client-server debba essere realizzata in un altro modo, magari utilizzando dei **server intermedi** e **disaccoppiando** le funzionalità di **ricezione** da quelle di **invio**. Di seguito illustriamo concisamente l'architettura della posta elettronica.

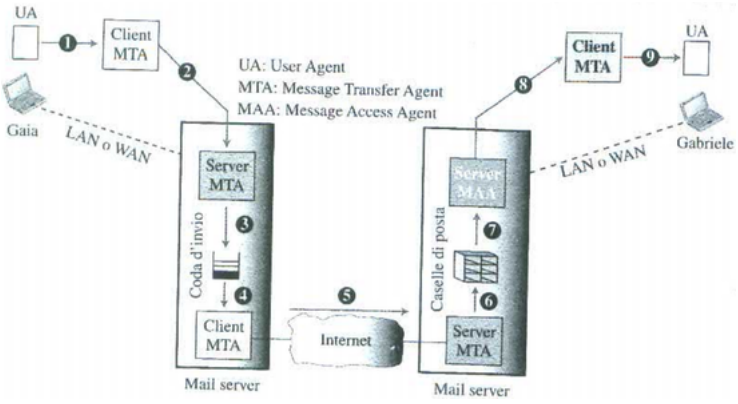


Figure 40: Un esempio di architettura di posta elettronica.

Tipicamente due utenti, mittente e destinatario, sono connessi a due **server di posta**. Ogni utente possiede una **mailbox** sul server, ovvero una porzione sulla memoria del server a cui solo lui può accedere e ogni server ha una **coda di invio**, o **spool**, dove memorizza i messaggi in attesa di essere inviati. Più:

- **UA** o **user agent**, prepara il messaggio e lo invia al server.
- **MTA** o **message trasfer agent**, è un'applicazione **push** .
- **MAA** o **message access agent**, è un'applicazione **pull**.

Illustriamo ora **come avviene l'invio di un messaggio** da parte di *Gaia* a *Gabriele*.

1. Gaia, utilizzando un programma **UA**, prepara il messaggio e lo invia al proprio server di posta.

2. Il **server di posta memorizza** il messaggio in una coda, e lo **invia** tramite un **programma MTA** al server di posta di Gabriele. **N.B. sono necessari due programmi MTA per ogni server di posta, un MTA server sempre attivo, in attesa di messaggi, e un MTA client, attivo all'evenienza, che contatta il server di posta a cui deve essere inviato un messaggio ed effettua l'invio**.

3. Il server di posta **riceverà** il messaggio e lo **memorizzerà** nella **casella postale di Gabriele** che, a sua volta usando un programma **MAA client**, contatterà il programma **MAA server** del **server** e riceverà infine il messaggio.

Lo **user agent** è il primo componente di un sistema di posta elettronica, **facilita all'utente l'invio e la ricezione dei messaggi**. Può avere o un'interfaccia a **riga di comando**, ma è abbastanza desueto, o di tipo **grafico**. Se l'utente decide di leggere i messaggi nella sua casella di posta elettronica lo **user agent** mostra un elenco dei messaggi ricevuti. Ogni riga dell'elenco offre un breve resoconto del messaggio, solitamente: **indirizzo del mittente**, **ora** e **oggetto del messaggio**.

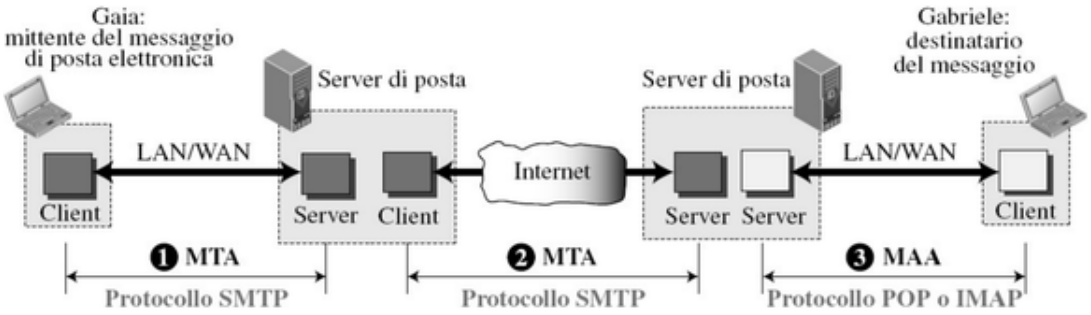
Come sono fatti gli indirizzi di posta elettronica?

Gli indirizzi di posta elettronica individuano gi utenti in modo **univoco**. Su **Internet** consistono di **due parti**:

La **parte locale** identifica la **casella di posta del destinatario** sul server e il nome di dominio identifica il **server**.

Come viaggiano i messaggi di posta elettronica?

Con il protocollo **SMTP** o **simple mail transfer protocol**. Il protocollo **SMTP** definisce in maniera formale l'**interazione tra il client MTA e il server MTA**. Nell'operazione di invio di un messaggio SMTP è utilizzato **due** volte: tra il **mittente** e il **suo server di post**a e tra il **server di posta del mittente** e **quello del destinatario**. Di seguito un'immagine per schiarire le idee.



Il **protocollo SMTP** definisce come deve avvenire l'interazione, per mezzo di **comandi** e **risposte**, tra **client e server MTA**. I comandi sono **inviati** dal **client MTA** al **server MTA** e viceversa le risposte. I **comandi** e **le risposte** terminano tutti con la medesima **coppia** di **caratteri**: **ritorno a capo e fine linea**.
La connessione tra client MTA e server MTA è una sola ed è bidirezionale.

- I **comandi** sono composti da una **keyword** e da **uno** o **più argomenti**.

Nome	**Argomenti**	**Significato**
HELO	nome host mittente	host mittente si identifica
MAIL FROM	mittente del messaggio	identifica mittente messaggio
RCPT TO	destinatario	identifica destinatario messaggio
DATA	corpo del messaggio	il messaggio
QUIT		termina sessione SMTP corrente
RSET		interrompe la transazione in atto
VRFY	nome destinatario	verifica validità nome destinatario

- Le **risposte** sono costituite da un **codice a tre cifre** seguite eventualmente da testo.

Codice	**Descrizione**
220	servizio pronto
221	servizio in chiusura canale trasmissione
250	comando richiesto completato
354	corpo del messaggio
421	servizio non disponibile
450	mailbox non disponibile
502	comando non disponibile

N.B. Il protocollo **SMTP usa** il protocollo di trasporto **FTP** per consegnare in maniera **affidabile** i messaggi.

La consegna di un messaggio tramite SMTP prevede **tre fasi**:

1. **Apertura della connessione** o **handshaking** con cui il client SMTP stabilisce una connessione FTP alla porta nota 25 con il server SMTP. Consiste di altre **tre sotto fasi**:

   1.1. Il **server** invia al **client** il codice **220** per indicare che è **pronto alla ricezione di messaggi** o il codice **421** in caso **contrario**.

   1.2. Il **client** si **identifica** con il comando **HELO** seguito dal suo **nome di dominio** in modo tale da informare il server del proprio nome di dominio.

   1.3. Il **server** invia il codice **250** o altri codici a seconda della situazione particolare.

2. **invio del messaggio**: se l'apertura della connessione tra il client SMTP e il server SMTP è avvenuta con **successo**, il client può inviare **un singolo messaggio** a uno o più destinatari. Riportiamo di seguito gli **otto** passi necessari per portare a termine questa operazione:

   2.1. Il **client** invia al server il comando MAIL FROM con argomento l'indirizzo mail del mittente, in modo tale che, nel caso in cui si verifichino degli errori, il server **sappia a chi inviare i messaggi di errore**.

   2.2. Il **server** risponde con il codice 250.

   2.3. Il **client** invia al server il comando RCPT TO con argomento l'indirizzo mail del destinatario

   2.4. Il server risponde con il codice 250.

   2.5. Il **client** invia il comando data per iniziare il trasferimento del messaggio.

   2.6. Il **server** risponde con il codice 354.

   2.7. Il **client** invia il messaggio come sequenza di righe, ognuna terminante con la coppia di caratteri ritorno a capo e fine linea. Il messaggio termina con una riga **contenente solo un punto**.

   2.8. Il server risponde con il codice 250.

   Se ci sono **più destinatari** i passi 2.3 e 2.4 sono **ripetuti**.

3. **Chiusura della connessione**: Il **client**, trasferito il messaggio, **chiude la connessione**. Questa operazione avviene in **due** fasi:

   3.1 Il **client** invia al server il comando QUIT.

   3.2 Il **server** risponde con il codice 221.

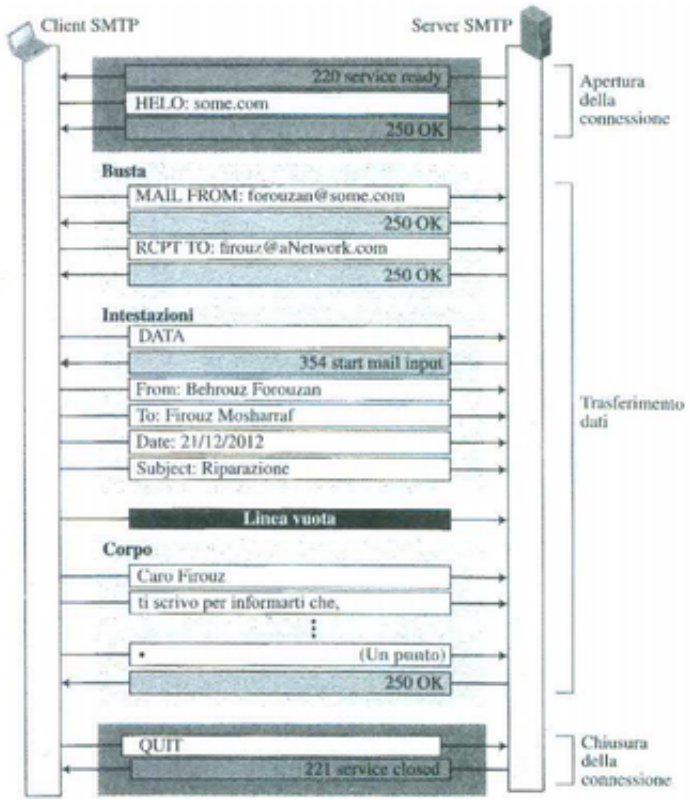


Figure 41: In figura un esempio di scambio di messaggi tramite SMTP. L'RFC 2822 definisce lo standard per il formato del messaggio. To: , From: , Subject: . **I campi sono diversi dai comandi SMTP. Il corpo del messaggio o body contiene solamente caratteri ASCII a 7 bit**.

Come vengono ricevuti i messaggi?

Poco sopra abbiamo descritto l'invio di un messaggio di posta elettronica come la **composizione di tre fasi**: una di **preparazione** del messaggio, una di **invio** e una di **ricezione**. Le prime due utilizzano i **protocollo SMTP**, che è un protocollo **push**, il messaggio viene *spinto* dal client mittente verso il server. L'ultima fase, quella di ricezione, usa invece un protocollo **pull**, il client destinatario *tira* i messaggi dal server. Attualmente sono in uso **due** protocolli di tipo **pull**: **POP3** o **Post Office Protocol v.3** e **IMAP4** o **Internet Mail Access Protocol v.4**. **POP3** è molto semplice ma ha funzionalità limitate. Il software **client POP3** è installato sul computer del destinatario mentre il software **server POP3** sul suo server di posta. Il **client POP3** apre una connessione TCP sulla porta **110** del **server POP3** e invia il proprio nome utente e la propria password per accedere alla casella postale. L'utente richiede poi la lista dei messaggi presenti e li preleva uno alla volta.

Il **protocollo POP3** prevede due modalità: **delete** e **keep**, con la modalità **delete** i messaggi vengono automaticamente eliminati dalla mailbox dopo il prelievo, con la modalità **keep** vengono tenuti per uso futuro.

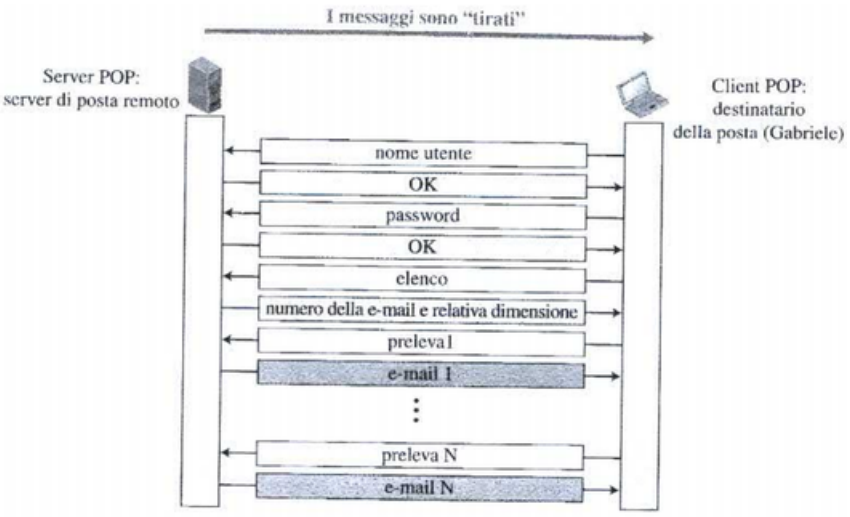


Figure 42: In figura il protocollo POP3.

Il **protocollo IMAP4** è simile al POP3 ma è più **potente** e **complesso**. POP3 **non** consente di **gestire più caselle** di posta sul server, di **organizzare** la posta e di **controllare una parte del messaggio prima di prelevarlo** nella sua interezza. **IMP4** invece fornisce varie funzionalità aggiuntive, tra le quali:

- **Controllare le intestazioni** dei messaggi prima di **prelevarli**.
- **Ricercare una stringa** specifica **nei messaggi prima di prelevarli**.
- **Prelevare i messaggi in modo parziale**. Utile per quando ci sono limitazioni di larghezza di banda.
- **Creare**, **cancellare** e **rinominare** le **mailbox** sul server di posta.
- **Creare una gerarchia di cartelle** all'interno della mailbox a scopo di archiviazione.

Abbiamo visto fin'ora come la posta elettronica abbia una struttura molto **elementare**, questa semplicità comporta però che i protocolli visti fin'ora supportino **soltanto** messaggi nel formato standard **NVT ASCII a 7 bit**.

Come fare quindi a scambiarsi messaggi con caratteri non supportati dal formato NVT ASCII a 7 bit, come ad esempio il ***formato binario****?*

Il **MIME** o **Multipurpose External Mail Extension** è un protocollo **supplementare** che permette l'invio di messaggi in formato **diverso** dall'**ASCII**. Il **MIME** agisce **sia** dal lato **mittente che** dal lato **destinatario**. Traduce tutti i dati in formato **non ASCII** in ASCII **prima** di inviare il messaggio alla MTA e opera poi a ritroso quest'operazione di traduzione **presso** il destinatario dopo che ha ricevuto il messaggio tramite la MAA.

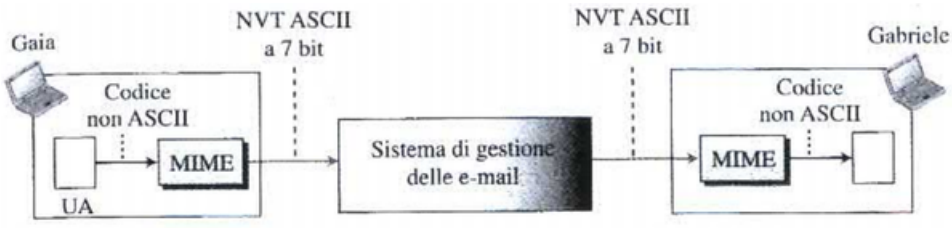


Figure 43: In figura il il funzionamento del protocollo MIME.

Il **protocollo MIME** definisce **cinque** tipi di intestazioni specifiche che si aggiungono a quelle originali previste dal protocollo di posta elettronica:

- **MIME - version**: vi è dichiarata la versione di MIME usata.
- **Content Type**: vi sono dichiarati i tipi di dato contenuti nel corpo del messaggio. Il tipo del contenuto e il sottotipo sono separati da una barra. Il protocollo MIME usa sette tipi di dato diversi:
  - **Text**: **plain** o **html**, formato del testo.
  - **Multipart**: **mixed**, **parallel** o **alternative**, da informazioni circa le parti da cui è composto il messaggio.
  - **Message**: **RFC 882**, **partial**, **external-body**. Da informazioni circa il messaggio, rispettivamente: se è un messaggio incapsulato, se è una parte di un altro messaggio o se è un riferimento ad un altro messaggio.
  - **Image**: **jpeg** o **gif**.
  - **Video**: **mpeg**.
  - **Audio**: **basic**.
  - **Application**: **PostScript** o **octet-stream**.
- **Content-Transfer-Encoding**: definisce la codifica utilizzata in trasporto per il messaggio.
- **Content-ID**: individua univocamente una parte del messaggio nei messaggi che sono composti da più parti.
- **Content-Description**: indica se il corpo del messaggio contiene un'immagine, un file audio o video.

Concludiamo questa sezione offrendo una breve panoramica del servizio chiamato **web mail**. Data la larga diffusione della posta elettronica molti siti web ne offrono il servizio. I **webmail servers** a seconda del **client** interessato, che può essere **SMTP** o **HTTP**, ricevono ed evadono i messaggi tramite o il **protocollo HTTP** o il **protocollo SMTP** come illustrato nelle seguenti figure:

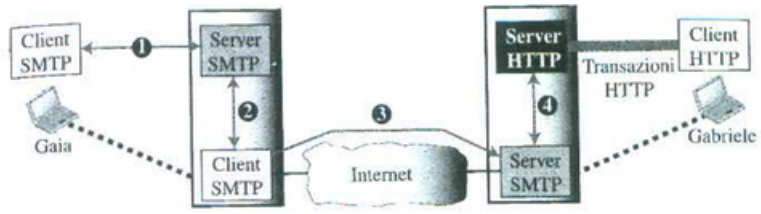


Figure 44: In figura uno scenario in cui **solo** il ricevente utilizza **HTTP**.

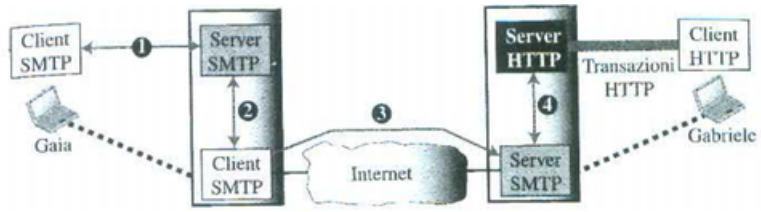


Figure 45: In figura uno scenario in cui **sia** che il ricevente che il destinatario utilizzano **HTTP**.