

Applicazioni di Rete, Socket e Architetture

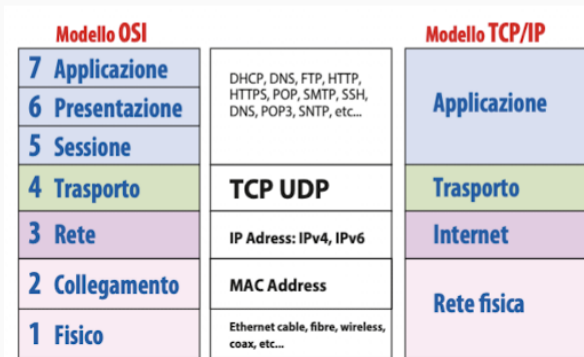
DAL LIVELLO APPLICAZIONE AI MODELLI CLIENT-SERVER E P2P

Francesco Gobbi

10 novembre 2025

Livello applicazione nei modelli ISO/OSI e TCP/IP

- **Scopo:** fornire i **protocolli** usati dalle **applicazioni di rete** (email, web, VoIP, streaming, autenticazione, ...).
- **ISO/OSI:** livelli 5–7 (Sessione, Presentazione, Applicazione).
- **TCP/IP:** il livello *Application* accorpa le funzionalità dei livelli alti OSI.
- Il programmatore usa **primitive** e **API** senza gestire i dettagli degli strati inferiori.



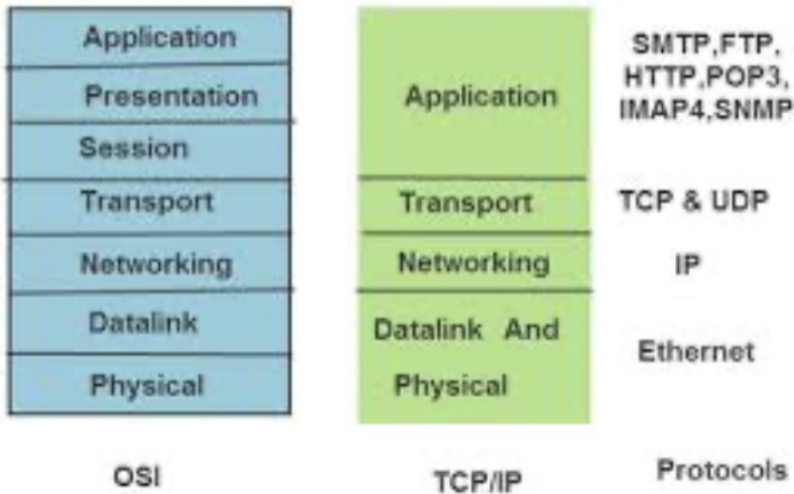
Protocolli applicativi più comuni

- **HTTP/HTTPS**: trasferimento ipertesti e API (80/443).
- **SMTP** invio email (25, 587), **POP3** (110) e **IMAP** (143) per ricezione.
- **DNS**: risoluzione nomi → IP (53 UDP/TCP).
- **FTP/SFTP**: trasferimento file (20–21 / 22).
- **SNMP**: gestione apparati (161/162).

Nota

Il livello applicazione *non coincide* con l'applicazione: espone **protocolli** che le app usano per dialogare con host remoti.

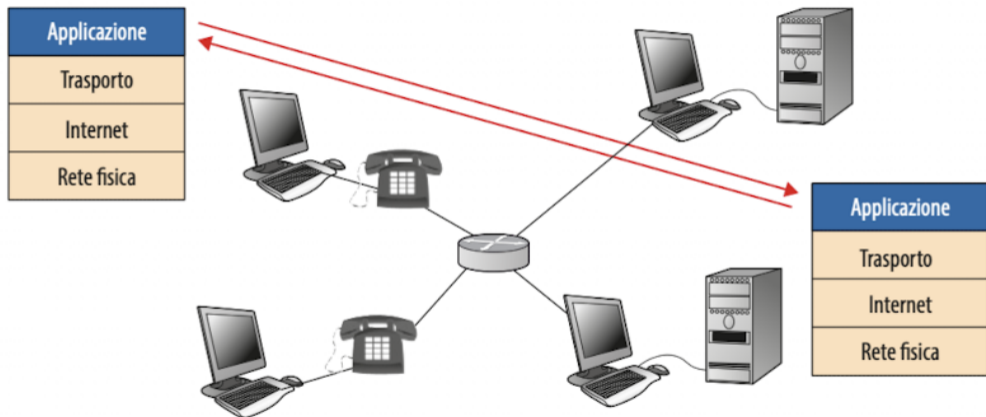
OSI & TCP/IP Protocol-Stacks and Protocols



Applicazioni di rete: componenti

- Un'**applicazione di rete** è composta da più programmi che operano su computer diversi, collegati da una rete.
- Questi programmi collaborano condividendo risorse comuni (come database o file) e lavorano **concorentemente**.
- Poiché i componenti si trovano su più macchine, si parla di **applicazione distribuita**.
- I **processi** che appartengono alla stessa applicazione devono scambiarsi messaggi per cooperare.
- La comunicazione può avvenire:
 - nella **stessa rete locale** (LAN);
 - oppure tra host **remoti**, anche molto distanti.
- Per comunicare, i processi devono “mettersi in contatto” usando gli **indirizzi** e i **servizi** del livello di applicazione.

Applicazioni di rete

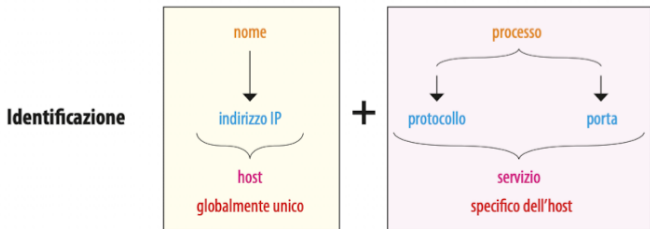


Identificazione di un servizio: socket = IP:porta

- Un **processo destinatario** deve essere identificato in modo univoco su un host.
- **Indirizzo IP** individua l'host; **numero di porta** individua il *servizio* su quell'host.
- Il server esegue un **bind** su una porta; il client contatta IP:porta.
- **Porte well-known** (0–1023) per servizi standard; **porte effimere** per i client.

Vincolo

Una porta non può essere condivisa da più processi *con lo stesso protocollo di trasporto* sullo stesso host.



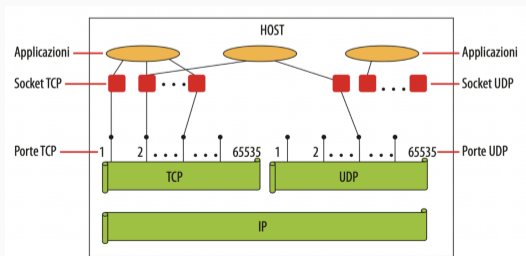
Trasporto e socket: TCP vs UDP

TCP

- Orientato alla connessione (3-way handshake).
- Affidabile, ordinato, controllo di flusso/congestione.
- Adatto a Web, email, file transfer, login remoti.

UDP

- Senza connessione, best-effort.
- Ritardi minori, no ritrasmissioni.
- Adatto a DNS, streaming/VoIP, giochi online (con logica applicativa).



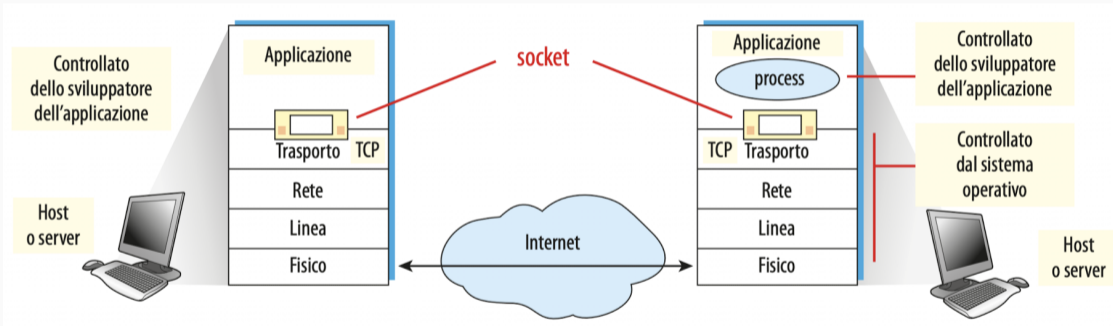
Socket, API e componenti di un'applicazione di rete

- **Socket** = <indirizzo IP:porta>: punto di accesso/uscita di un processo. Il mittente invia dal proprio socket; la pila **TCP/IP** trasporta i dati fino al socket del processo destinatario.
- **API di rete** (es. BSD sockets) offrono le primitive per: *creare* un socket, *bind* su una porta, *connettersi/accettare*, *inviare/ricevere* e *chiudere* la comunicazione.
- Un'applicazione di rete ha due parti:
 1. **User agent** — interfaccia e logica lato utente, da visualizzare.
 2. **Implementazione dei protocolli** — librerie/stack che parlano con la rete, per permettere la connessione.

Esempio: Browser Web

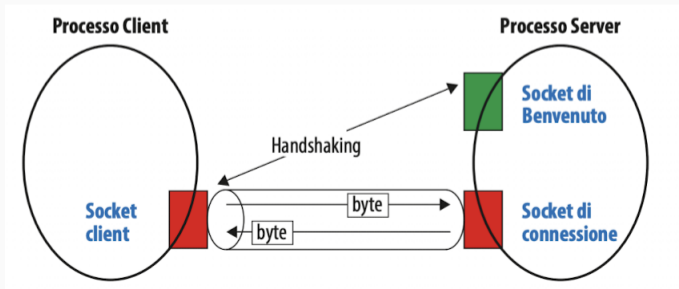
1. **Interfaccia utente**: mostra le pagine, permette la navigazione e l'inserimento di URL.
2. **Motore del browser**: risolve DNS, apre una connessione **TCP/TLS** verso IP:443, invia richieste **HTTP(S)** e riceve le risposte.

Trasporto e socket TCP



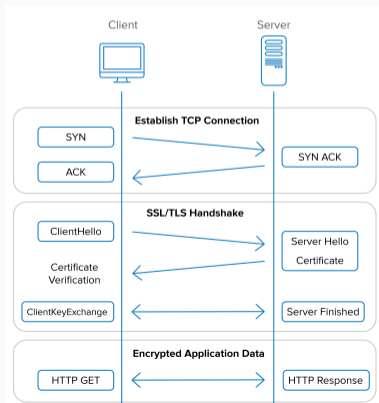
Connessione tra client e server

- Il **server** può servire più **client** contemporaneamente.
- Utilizza un **socket di benvenuto** per attendere nuove richieste.
- Quando un client si connette, il server:
 1. accetta la richiesta (**handshaking**);
 2. crea un nuovo **socket di connessione** dedicato al client;
 3. avvia un **thread concorrente** per gestire la comunicazione.
- Nel frattempo, il socket di benvenuto resta libero per altri client.



Esempio percorso di una richiesta Web

- L'utente inserisce un **URL** → il browser interroga **DNS** per l'IP.
- Apertura **TCP 3-way handshake** verso IP:443 (HTTPS).
- Scambio **HTTP** (request/response), eventuale **TLS** per la cifratura.
- Concorrenza: più richieste in parallelo, connessioni persistenti.



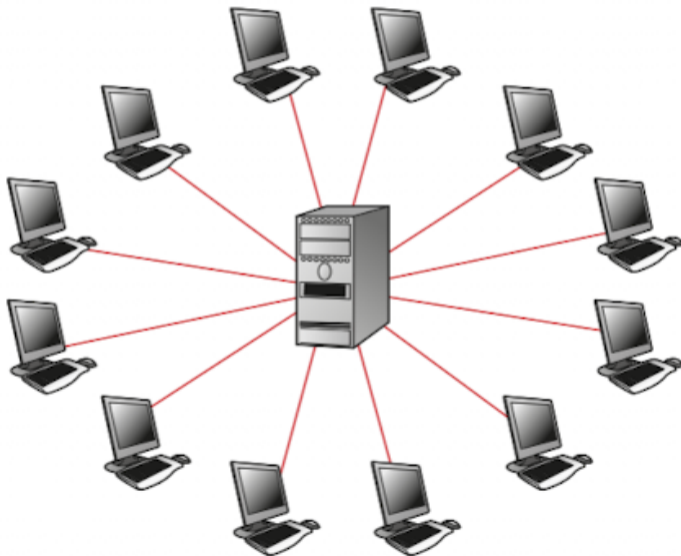
È importante per lo sviluppo e la progettazione dell'applicazione di rete definire il tipo di architettura da utilizzare:

- Client-server
- Peer-to-peer(P2P)
- Architetture ibride (unione tra le due precedenti)

La scelta chiaramente è centrale e lo sviluppatore o il team devono valutare attentamente le potenzialità di ognuna per ottenere un buon prodotto finito.

- In questa architettura esiste sempre un **server attivo** che offre un servizio ai **client**.
- I **client** si connettono al server per richiedere risorse o informazioni, ma **non comunicano tra loro**.
- Il **server** deve avere:
 - un **indirizzo IP statico**, per essere sempre raggiungibile;
 - la capacità di servire più client **contemporaneamente**.
- Un esempio tipico è il **WWW**: i browser (client) inviano richieste ai web server, che rispondono con le pagine web.
- Se il numero di richieste è elevato, si può creare una **server farm**: più server che condividono il carico.

Architettura *client-server*



- Nelle reti **P2P** non c'è un server centrale: tutti i computer possono comportarsi sia da **client** che da **server**.
- Gli host si chiamano **peer** ("pari"): ognuno può condividere risorse e richiederle agli altri.
- Il sistema P2P si basa sulla **collaborazione**: ogni peer offre e riceve servizi o dati.
- Esempi: condivisione di file (*eMule*, *BitTorrent*), comunicazioni dirette, reti di calcolo distribuito.

Modello *P2P decentralizzato*

- Tutti i peer sono sullo stesso piano: ognuno è sia **client** che **server**.
- Non esiste alcun nodo di controllo centrale.
- Le risorse (file, dati, memoria, banda) sono distribuite tra i peer.
- Ogni peer trova gli altri usando meccanismi interni di **ricerca e indirizzamento**.
- Il sistema si adatta facilmente ai cambiamenti (peer che si aggiungono o si scollegano).
- Esempio: uTorrent e simili

Vantaggi

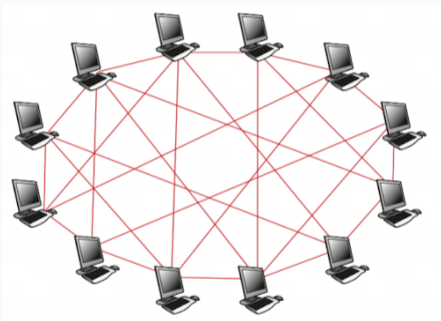
- Alta **scalabilità** e **robustezza**.
- Nessun punto unico di guasto (SPOF = Single Point Of Failure).

- È un compromesso tra P2P puro e modello client-server.
- Esiste un **server centrale** (directory server) che tiene l'elenco dei peer e delle risorse condivise.
- Il server non conserva i file, ma aiuta i peer a trovarsi tra loro (**indice risorse–peer**).
- Dopo la ricerca, i peer si scambiano i dati direttamente.

Esempio

Napster: gli utenti si collegavano a un server centrale per cercare le canzoni, ma il trasferimento avveniva direttamente tra peer.

Confronto tra due architetture P2P



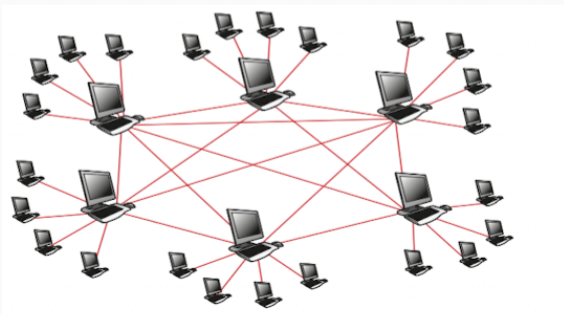
P2P Decentralizzato



P2P Centralizzato

P2P ibrido (parzialmente centralizzato)

- In una rete **P2P ibrida** alcuni nodi diventano **super-peer** (o *supernodi/ultra-peer*).
- I super-peer **indicizzano** risorse e collegano molti **leaf peer** (nodi foglia).
- La ricerca passa dai leaf al super-peer, poi ai peer che possiedono la risorsa.
- **Vantaggi:** più *scalabile* del client-server; più *efficiente* del P2P puro nelle ricerche.
- **Limiti:** dipendenza dai super-peer (possibili colli di bottiglia / punti critici).



- Le **applicazioni mobile** sono progettate per dispositivi mobili (smartphone, tablet) e funzionano sia **online** che **offline**.
- Sono pensate per l'uso in movimento e hanno interfacce ottimizzate:
 - **Touch screen**, icone, gesture (swipe);
 - funzioni come **riconoscimento vocale**, **GPS**, **fotocamera**, **NFC**.
- A differenza delle applicazioni web, non richiedono sempre una connessione attiva: possono usare anche dati salvati localmente.
- Tipologie principali:
 - **App native**: create per un solo sistema operativo (es. Android, iOS), con accesso completo all'hardware.
 - **App ibride**: usano tecnologie web ma sfruttano anche funzioni del dispositivo.

Esempi:

- **Offline:** mappe salvate localmente; note/todolist che si sincronizzano quando torna la rete; musica scaricata.
- **Uso sensori:** navigazione con **GPS**, QR scanner con **fotocamera**, pagamenti contactless con **NFC**.
- **Native:** giochi 3D, app fotocamera, navigatori turn-by-turn, wallet (Apple/Google Pay).
- **Ibribe/PWA:** portali scolastici/e-commerce installabili da browser; esempi noti di PWA: *Twitter Lite*, *Starbucks*, *Pinterest*.

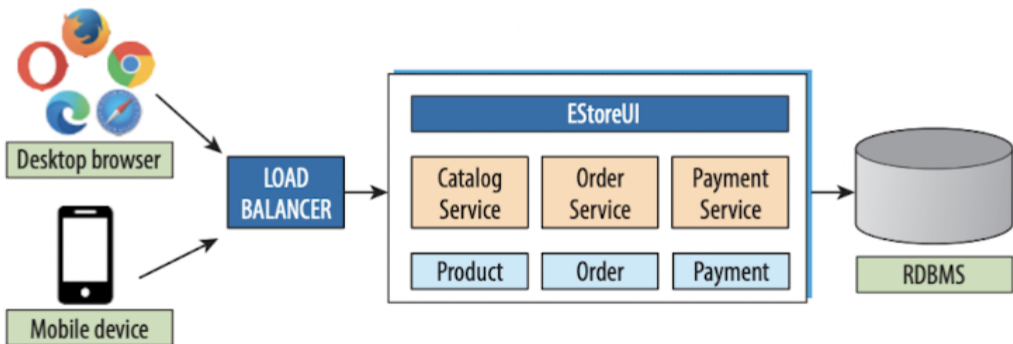
Applicazioni web monolitiche

- Un'applicazione **monolitica** è un unico programma che contiene tutte le funzioni:
 - interfaccia utente;
 - logica applicativa;
 - connessione al database.
- Tutte le parti dipendono fortemente tra loro → **poca separazione e bassa flessibilità**.
- Funziona bene per progetti piccoli o poco complessi, ma:
 - è **poco scalabile**;
 - difficile da aggiornare o modificare senza impatti sull'intero sistema.

Esempio

Un sito e-commerce dove un solo programma gestisce catalogo, ordini e pagamenti in un unico database.

Applicazioni web monolitiche



- L'**architettura a microservizi** divide l'applicazione in **piccoli servizi indipendenti**, ognuno con una funzione specifica.
- Ogni microservizio:
 - è sviluppato e aggiornato autonomamente;
 - comunica con gli altri tramite **API** o **protocolli standard** (HTTP, REST, AMQP...);
 - può avere il proprio **database** e linguaggio di programmazione.
- Offre **scalabilità**, **flessibilità** e **manutenibilità** molto superiori.

Esempio

Un e-commerce in cui ogni servizio (catalogo, carrello, ordini, pagamenti) è un modulo indipendente con il suo database.

Applicazioni web a microservizi

