**Le porte, le socket ed i servizi.**

**Le porte nel modello TCP/IP:**

Nei livelli inferiori dello stack TCP/IP (fisico, data link, network), la comunicazione avviene tra nodi e host, ma non è sufficiente per identificare l'applicazione destinataria del messaggio una volta che il pacchetto arriva all’host. Il protocollo IP, ad esempio, trasporta i datagrammi ma non specifica quale applicazione debba riceverli.

**Problema:**

Su un host possono essere attivi più processi contemporaneamente. Quando un pacchetto arriva, è necessario sapere a quale processo (e quindi a quale applicazione) consegnarlo. Tuttavia:

* I processi sono dinamici e non possono essere conosciuti in anticipo dai mittenti.
* Un processo può gestire più funzioni, ma solo una è quella corretta destinataria.

**Soluzione: Le porte**

Per risolvere questo problema, si è introdotto il concetto di **porte**:

* Ogni host definisce **punti di accesso** (porte) associati a specifiche funzioni e protocolli.
* I pacchetti vengono consegnati solo alla porta corretta.
* Ogni porta è identificata da un numero intero a 16 bit (range: **0–65.535**).

**Classificazione delle porte (IANA):**

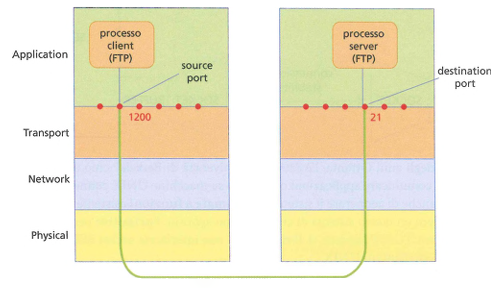
| **Tipo di porta** | **Intervallo numerico** | **Uso previsto** |
| --- | --- | --- |
| Well Known Ports | 0–1023 | Servizi standard (es. HTTP, FTP) |
| Registered Ports | 1024–49151 | Applicazioni registrate |
| Dynamic/Private Ports | 49152–65535 | Uso temporaneo o privato |

* Le porte sono assegnate da **IANA** e spesso condivise tra i protocolli **TCP** e **UDP**.
* La registrazione ufficiale è regolata da [RFC4340], sezione 19.9.

**Comunicazione client-server:**

Per stabilire una comunicazione, entrambi gli host devono conoscere:

* **Source port**: porta del client da cui parte la richiesta e su cui attende la risposta.
* **Destination port**: porta del server su cui è in ascolto il processo che fornisce il servizio.



**Esempio:**

Nel caso di **FTP** (File Transfer Protocol), il client usa una porta sorgente per inviare la richiesta, mentre il server riceve sulla porta destinazione associata al servizio FTP.

**Le socket nel modello TCP/IP:**

**Funzione delle socket:**

Le **porte** permettono di identificare un processo su un host, ma per stabilire una **connessione bidirezionale univoca** tra client e server è necessario un ulteriore livello: la **socket**.

Una socket è un punto terminale di comunicazione che collega:

* un'applicazione client (che fa la richiesta),
* a un'applicazione server (che fornisce il servizio).

**Identificazione della connessione:**

Ogni pacchetto che viaggia in rete deve contenere:

* Indirizzo IP del client
* Indirizzo IP del server
* Porta del client
* Porta del server

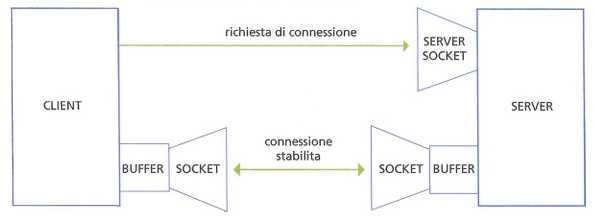
Aggiungendo il protocollo (es. TCP), si ottiene una **quintupla** detta *Association*:

(protocollo, IP locale, porta locale, IP remoto, porta remota)

esempio: (TCP, 200.18.6.12, 3000, 58.163.138.10, 4000)

Questa quintupla è composta da due socket:

* Socket locale: (protocollo, IP locale, porta locale)
* Socket remota: (protocollo, IP remoto, porta remota)



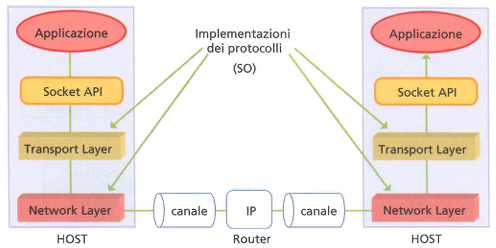
**Interfaccia BSD Socket:**

Negli anni ’80, TARPA e l’Università di Berkeley svilupparono l’interfaccia **BSD Socket** per UNIX, basata su chiamate di sistema. Le principali **primitive** sono:

| **Primitiva** | **Descrizione** | **Applicazione** |
| --- | --- | --- |
| SOCKET | Crea un nuovo punto di comunicazione. Restituisce un descrittore di file. | Server, Client |
| BIND | Associa un indirizzo locale alla socket. | Server |
| LISTEN | Prepara la socket ad accettare connessioni multiple. Non è bloccante. | Server |
| ACCEPT | Blocca il server finché non riceve una richiesta. Crea una nuova socket e processo/thread. | Server |
| CONNECT | Usata dal client per iniziare la connessione. Blocca finché il server risponde. | Client |
| SEND | Invia dati sulla connessione. | Server, Client |
| RECEIVE | Riceve dati dalla connessione. | Server, Client |
| CLOSE | Chiude la connessione in modo simmetrico. | Server, Client |

**API e programmazione socket:**

Nel 1991, Microsoft definì una **API standard per TCP/IP** in ambiente Windows, semplificando la comunicazione tra host.



Oggi, linguaggi come **Java** e **C#** offrono API object-oriented per la programmazione socket:

* Java: esercizi con TCP e UDP per client-server.
* C#: esercizi analoghi in ambiente .NET Framework.

**Cos’è un’API?**

Un’**API (Application Programming Interface)** è un insieme di procedure o librerie che permettono agli sviluppatori di accedere a funzioni di sistema o servizi web. Sono fondamentali per:

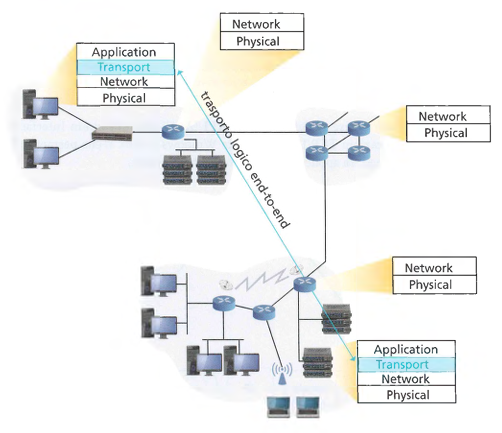
* creare nuove funzionalità,
* integrare servizi esterni (es. Facebook, Google Maps, Twitter).

**I servizi del livello Transport:**

**Funzioni principali:**

Il **livello Transport** svolge funzioni simili al livello Data Link (controllo errori, sequenza, flusso), ma opera in uno scenario diverso:

* **Data Link**: comunicazione diretta tra due router tramite canale fisico.
* **Transport**: comunicazione **end-to-end** tra host remoti tramite canale logico.



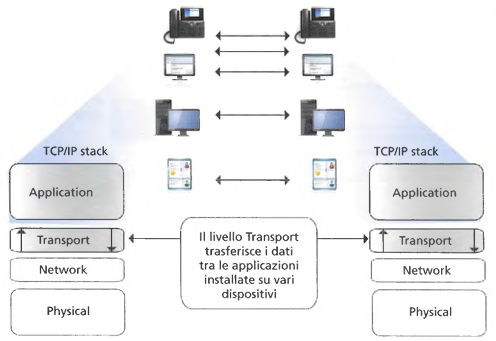
La comunicazione avviene tra le **socket** predisposte sui due host.

**Ruolo nel modello TCP/IP:**

Il livello Transport agisce come **interfaccia cuscinetto** tra:

* le **applicazioni** (che non devono conoscere i dettagli della rete),
* e i **livelli inferiori** (che non sanno quante applicazioni stanno comunicando).

Questo consente il trasferimento dati tra applicazioni su dispositivi diversi, senza preoccuparsi del tipo di rete, percorso o mezzo trasmissivo.



**Protocolli di trasporto:**

I principali protocolli standardizzati sono:

| **Protocollo** | **Caratteristiche** |
| --- | --- |
| **TCP** (Transmission Control Protocol) | Affidabile, connessione end-to-end, controllo congestione e flusso, consegna ordinata dei dati |
| **UDP** (User Datagram Protocol) | Non affidabile, senza connessione, adatto a trasmissioni rapide e leggere |
| **SCTP** (Stream Control Transmission Protocol) | Introdotto nel 2000, supporta multistreaming e multihoming, tolleranza ai guasti, controllo congestione |

Nei documenti RFC:

* L’unità dati del livello Transport è chiamata **segment** in TCP,
* e **datagram** in UDP.

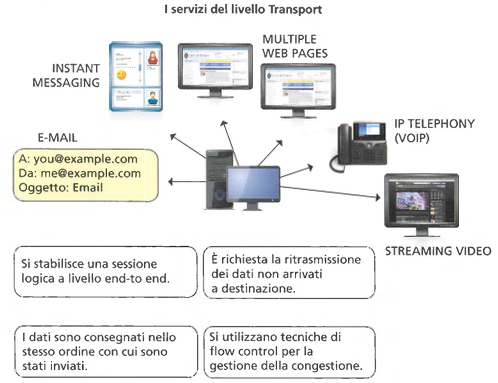
**Servizi offerti dal livello Transport:**

Il livello Transport fornisce:

* **Multiplexing/Demultiplexing**: instradamento dei dati verso i processi corretti.
* **Controllo integrità**: verifica che i dati arrivino correttamente.
* **Garanzia di consegna** (TCP): ritrasmissione dei dati persi.
* **Controllo di flusso**: gestione della velocità di trasmissione (es. sliding window).
* **Controllo congestione**: prevenzione del sovraccarico della rete.

**Applicazioni supportate:**

| **Tipo di servizio** | **Supporto del livello Transport** |
| --- | --- |
| Web (HTTP) | ✔ TCP garantisce ordine e affidabilità |
| Email (SMTP) | ✔ TCP garantisce consegna |
| Messaggistica istantanea | ✔ TCP o UDP a seconda del servizio |
| VoIP (telefonia IP) | ✔ SCTP o UDP |
| Streaming video | ❌ UDP, senza garanzia di ordine o ritrasmissione |



Non tutti i protocolli offrono tutti i servizi: ad esempio, **UDP** non garantisce né ordine né affidabilità.