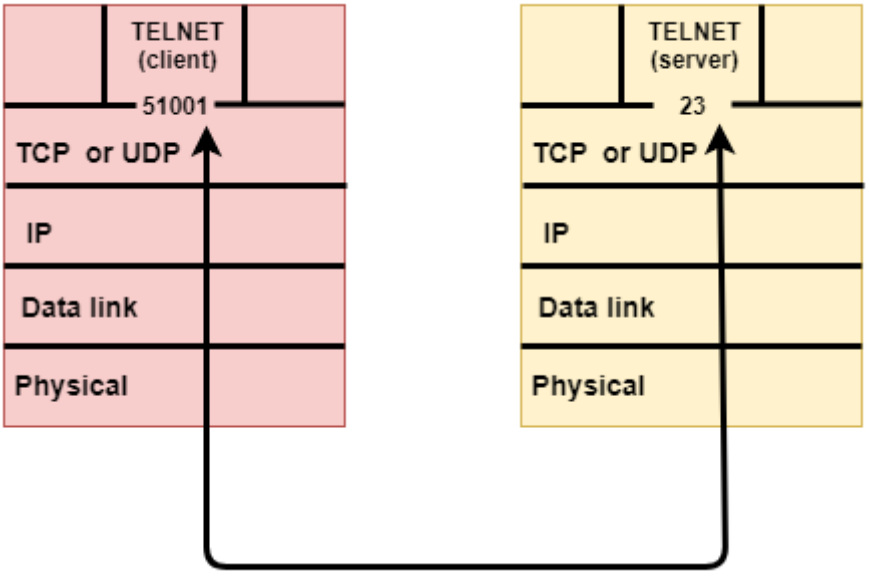
Livello di trasporto

Scritto da [Febogamedev]

Il compito del **livello di trasporto** è fornire servizi al soprastante livello applicativo e, per raggiungere tale scopo, sfrutta i servizi offerti dal sottostante livello 3, il livello di rete (Eddy, 2022).

Lo scopo del livello di trasporto è fornire un canale logico di comunicazione end-to-end per pacchetti. Nello specifico, si occupa di supplire alle carenze delle funzionalità di trasferimento in termini di affidabilità, implementando le suddette funzioni come garanzie sul trasporto stesso (Eddy, 2022).

Il livello di trasporto si trova solo negli end-systems (hosts) e consente il collegamento logico tra processi applicativi.



Nell'immagine si vedono i due livelli di trasporto di due host che dialogano. Per dialogare devono passare attraverso i livelli sottostanti.

Protocolli di trasporto

Il livello di trasporto fornisce al livello applicativo delle connessioni con specifiche qualità, in particolare sfruttando i protocolli *UDP, TCP e QUIC* (Eddy, 2022; Iyengar & Thomson, 2021; Postel, 1980).

TCP

***TCP* (Transmission Control Protocol)** è uno dei principali protocolli di comunicazione utilizzati nel livello di trasporto (Eddy, 2022).

Il TCP consente di avere un servizio orientato alla connessione e affidabile.
Tramite TCP, tra mittente e destinatario si stabilisce una connessione logica affidabile e bidirezionale.

Le caratteristiche principali del *TCP* includono:

- Affidabilità: *TCP* garantisce una consegna affidabile dei dati. Utilizza meccanismi di acknowledgment (abbreviato ack, cioè conferma di ricezione), controllo degli errori e ritrasmissione per assicurarsi che i pacchetti siano consegnati integri e nell'ordine corretto. Se un pacchetto viene perso o corrotto durante la trasmissione, il mittente lo ritrasmette finché il destinatario non conferma la sua ricezione (Eddy, 2022).
- Controllo di flusso: *TCP* gestisce il controllo del flusso, che serve a evitare che il mittente invii dati più velocemente di quanto il destinatario riesca a riceverli ed elaborarli (Eddy, 2022).
- Controllo della congestione: *TCP* regola la velocità di invio in base alle condizioni della rete, per evitare di sovraccaricare i collegamenti e i router intermedi (Allman et al., 2009).
- Orientamento alla connessione: *TCP* stabilisce una connessione tra il mittente e il destinatario prima di iniziare a trasmettere i dati.

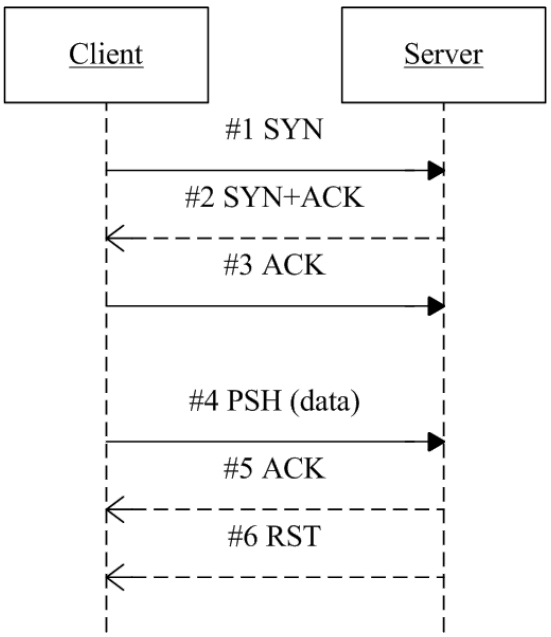
Per inviare dati tramite *TCP*, il protocollo suddivide il messaggio dato dal livello applicativo in pacchetti più piccoli chiamati **segmenti**. Questi segmenti contengono (Eddy, 2022):

- I dati originali (payload).
- L'header TCP che contiene informazioni come il numero di sequenza, il numero di conferma, le informazioni di controllo e altri parametri necessari per garantire la corretta consegna e il controllo del flusso.

I segmenti *TCP* vengono inseriti in pacchetti *IP* (pacchetti creati nel livello di rete) e trasmessi attraverso la rete. I pacchetti *IP* possono anche seguire percorsi diversi prima di raggiungere la destinazione.
Una volta arrivati alla destinazione, il ricevitore assembla i segmenti in ordine di sequenza e utilizza le informazioni di controllo per confermare la ricezione dei segmenti.
Se un pacchetto non viene ricevuto correttamente, il *TCP* richiede la sua ritrasmissione al mittente per garantire l'integrità e l'affidabilità dei dati.
TCP è stato ed è ancora molto usato in Internet, soprattutto per servizi che richiedono affidabilità, come il trasferimento di file, molte comunicazioni web, email e accesso remoto.

Una connessione TCP passa almeno attraverso queste fasi (Eddy, 2022):

- **SYN**: il client chiede di aprire una connessione (*SYN*) = "Inizio connessione".
- **SYN-ACK / ACK**: il server risponde (*SYN-ACK*), e il client conferma (*ACK*) = "Connessione stabilita".
- **Data Transfer / PSH**: i due scambiano dati (segmenti con conferma, cioè *ACK*) = "Connessione attiva".
- **FIN / RST**: uno dei due chiude la connessione (*FIN* o *RST*) = "Connessione terminata".



UDP

***UDP (*User Datagram Protocol)**, come *TCP*, è un protocollo del livello di trasporto, ma offre un servizio molto diverso (Postel, 1980).

A differenza del *TCP*, che offre una comunicazione affidabile e orientata alla connessione, l'*UDP* è un protocollo senza connessione e non garantisce l'affidabilità della consegna dei dati (Postel, 1980).
Questo significa che quando un'applicazione utilizza l'*UDP* per inviare dati a un'altra applicazione, i dati vengono semplicemente inviati senza alcuna garanzia che siano ricevuti o consegnati correttamente.
L'UDP consente quindi di avere un servizio non orientato alla connessione e non affidabile.

Caratteristiche principali di *UDP*:

- Senza connessione: non viene stabilita alcuna connessione tra mittente e destinatario prima dell'invio dei dati. I dati vengono inviati senza nessuna negoziazione preliminare.
- Comunicazione veloce e leggera: poiché *UDP* non ha la complessità della gestione delle connessioni o del controllo degli errori come *TCP*, è più veloce e leggero in termini di overhead.
- Nessun controllo del flusso e della congestione: *UDP* non gestisce il controllo del flusso né quello della congestione e non implementa meccanismi complessi di affidabilità come conferme, ritrasmissioni e riordinamento dei dati (Eggert et al., 2017).
- Utilizzo comune per applicazioni che richiedono bassa latenza: *UDP* è spesso utilizzato in applicazioni in cui la velocità e la bassa latenza (il tempo di arrivo di un pacchetto) sono più importanti dell'affidabilità, come streaming multimediale, giochi online e trasmissioni in tempo reale.

Poiché l'*UDP* non garantisce la consegna affidabile dei dati, le applicazioni che utilizzano questo protocollo devono gestire eventuali perdite o errori di dati a livello dell'applicazione stessa, se necessario (Eggert et al., 2017).
L'*UDP* è utilizzato in situazioni in cui la velocità e la semplicità sono più importanti dell'affidabilità nella consegna dei dati. Alcuni dei principali casi d'uso in cui l'*UDP* è preferito includono:

- Flussi audio/video: *UDP* è usato spesso per flussi audio/video in tempo reale, come videoconferenze, *VoIP* o trasmissioni live a bassa latenza.
- Giochi online: nei giochi online, l'*UDP* è spesso preferito per trasmettere informazioni sulle azioni dei giocatori in tempo reale. La bassa latenza è critica per garantire un'esperienza di gioco reattiva e fluida.
- Trasmissioni in tempo reale: nelle applicazioni di trasmissione in diretta, come trasmissioni sportive o eventi in tempo reale, l'*UDP* è preferito per fornire una visione in tempo reale senza buffering.
- Servizi di ricerca di dispositivi locali: nelle reti locali, l'*UDP* è utilizzato in alcuni protocolli di ricerca di dispositivi, dove è importante una rapida scoperta dei dispositivi nella rete.
- Applicazioni IoT (Internet of Things): in alcuni scenari IoT (come i sensori di temperatura, ecc.), in cui i dispositivi devono trasmettere piccole quantità di dati in modo rapido e senza molta complessità, l'UDP può essere adottato.

L'unità di dati inviata da UDP si chiama **datagramma**. Un datagramma *UDP* contiene un header con informazioni di controllo essenziali e un payload, cioè i dati ricevuti dall'applicazione (Postel, 1980).

QUIC

Dopo aver visto *TCP* e *UDP*, possiamo introdurre un protocollo più recente: ***QUIC*** **(Quick UDP Internet Connections)**.
QUIC è un protocollo di trasporto moderno progettato per rendere le comunicazioni su Internet più veloci, sicure ed efficienti (Iyengar & Thomson, 2021).
È particolarmente importante perché viene usato da *HTTP/3*, una delle versioni più recenti del protocollo *HTTP* utilizzato per il Web (Bishop, 2022).

La caratteristica interessante di *QUIC* è che funziona sopra *UDP*.
Questo può sembrare strano, perché *UDP* è un protocollo non orientato alla connessione e non affidabile.
Tuttavia *QUIC* utilizza *UDP* come base e aggiunge sopra di esso molte funzionalità che normalmente associamo a *TCP*, come l'affidabilità, il controllo della congestione e la gestione della connessione (Iyengar & Thomson, 2021).

Uno dei motivi per cui *QUIC* è stato sviluppato è ridurre il tempo necessario per iniziare una comunicazione. Con *TCP*, prima bisogna stabilire una connessione; se poi si usa *HTTPS* (HTTP più la sicurezza), bisogna aggiungere anche la negoziazione crittografica.
Questo può richiedere più passaggi prima che i dati veri e propri inizino a essere scambiati.

QUIC integra invece la sicurezza direttamente nel protocollo e usa meccanismi basati su *TLS*. Questo permette di ridurre il numero di passaggi necessari per stabilire una connessione sicura. Il risultato è che, in molti casi, una comunicazione può iniziare più rapidamente (Thomson & Turner, 2021).

Un'altra caratteristica importante di *QUIC* è la gestione migliore delle interruzioni e dei cambi di rete. Per esempio, uno smartphone può iniziare a navigare usando il *Wi-Fi* e poi passare alla rete mobile.
Con una connessione tradizionale basata su *TCP*, questo cambio può interrompere la connessione, perché cambia l'indirizzo di rete (l'indirizzo IP) usato dal dispositivo.
QUIC, invece, è progettato per gestire meglio questi cambiamenti e mantenere la comunicazione attiva quando possibile (Iyengar & Thomson, 2021).

QUIC migliora anche la gestione di più flussi di dati all'interno della stessa connessione. Nel Web moderno, una pagina non è formata da un solo file: contiene testo, immagini, fogli di stile, script, video e molte altre risorse. *QUIC* permette di gestire più flussi in parallelo, riducendo il rischio che il ritardo o la perdita di una parte dei dati blocchi inutilmente anche le altre (Bishop, 2022; Iyengar & Thomson, 2021).
Questa caratteristica è importante perché nelle comunicazioni di rete alcuni pacchetti possono arrivare in ritardo o andare persi. Con alcuni sistemi tradizionali, la perdita di un pacchetto può rallentare anche dati che sarebbero già disponibili. *QUIC* cerca di ridurre questo problema separando meglio i diversi flussi di comunicazione.

Possiamo quindi vedere QUIC come un protocollo che unisce alcune caratteristiche positive di *TCP* e *UDP*:

- Come *UDP*, è costruito su datagrammi e non richiede la stessa struttura rigida di *TCP*.
- Come *TCP*, può offrire affidabilità, controllo della congestione e gestione ordinata dei dati; però, a differenza di TCP tradizionale, integra la sicurezza e può stabilire comunicazioni protette più rapidamente.

QUIC è adatto al Web moderno, dove servono velocità, sicurezza e capacità di gestire molti flussi di dati contemporaneamente.
QUIC è particolarmente importante perché è alla base di HTTP/3, la versione più recente di *HTTP* (Bishop, 2022).

Porta

In un sistema di rete, **una porta** è il numero logico utilizzato per identificare un canale specifico di comunicazione all'interno di un host (Cotton et al., 2011).
È uno degli elementi fondamentali che permette a più applicazioni nello stesso host di usare contemporaneamente la rete senza interferire tra loro.

Immagina un centralino telefonico di una grande azienda.
Tutti i telefoni sono collegati a un unico numero (l'indirizzo reale), ma quando chiami quel numero puoi essere messo in contatto con diversi interni: vendite, assistenza,

amministrazione… Ecco: il numero di telefono è come l'indirizzo reale del computer, mentre gli interni sono le porte.

Quindi ricapitolando:
Le porte sono lo strumento utilizzato per permettere a un host di effettuare contemporaneamente più connessioni diversificate verso altri host, facendo in modo che i dati relativi a un servizio specifico vengano indirizzati al processo che li sta aspettando (Internet Assigned Numbers Authority [IANA], n.d.).

Nelle comunicazioni basate su *TCP* o *UDP*, le porte sono usate per associare i dati al processo applicativo corretto (Cotton et al., 2011).
In una comunicazione di rete non esiste solo una porta: di solito ci sono una porta sorgente e una porta di destinazione.

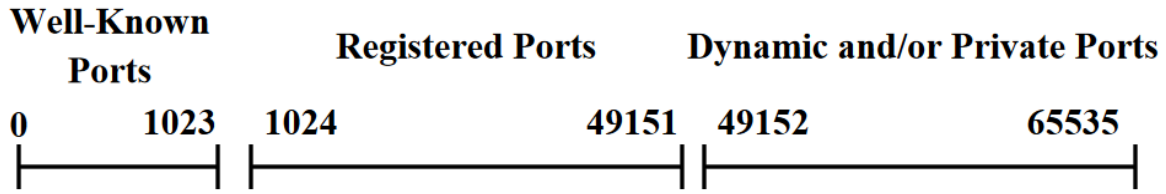
- La porta di destinazione indica il servizio a cui vogliamo collegarci. Per esempio, un server web HTTPS ascolta normalmente sulla porta 443.
- La porta sorgente, invece, è scelta dal dispositivo che avvia la comunicazione. Spesso è una porta temporanea assegnata automaticamente dal sistema operativo.

In questo modo, il computer può distinguere più comunicazioni attive nello stesso momento. Per esempio, se un browser apre più connessioni verso siti web, il sistema operativo può usare porte sorgente diverse per riconoscere ciascuna comunicazione.

Le porte sono $2^{16} = 65536$, classificabili in tre gruppi (IANA, n.d.):

- **Le porte conosciute** sono quelle inferiori a 1024 e sono generalmente utilizzate a livello di sistema operativo o di processi di sistema. In genere rimangono in ascolto su queste porte applicazioni con funzioni di server (del livello applicativo).
  Alcuni esempi di protocolli utilizzati dalle applicazioni sono (IANA, n.d.):
  - *FTP,* un protocollo di trasferimento di file usato da FileZilla => porta 21.
  - *SSH,* un protocollo che permette di stabilire una sessione remota cifrata => porta 22.
  - *SMTP,* un protocollo standard per la trasmissione di email => porta 25.
  - *HTTP,* un protocollo a livello applicativo usato come principale sistema per la trasmissione d'informazioni sul web => porta 80.
  - *HTTPS*, l'*HTTP* con l'aggiunta della sicurezza → porta 443.
- **Le porte registrate:** queste porte sono assegnate a specifici protocolli, applicazioni o servizi, ma non sono riservate in modo rigido come le porte conosciute.
  Le porte registrate sono spesso utilizzate da applicazioni e servizi meno comuni o da nuovi protocolli che richiedono un numero di porta standardizzato ma che non necessitano dell'ampia riconoscibilità associata alle porte ben note. Ad esempio, alcuni protocolli di comunicazione personalizzati o applicazioni meno diffuse potrebbero utilizzare porte registrate.
- **Le porte dinamiche/private**, invece, sono tutte le altre, liberamente utilizzabili da tutte le applicazioni utente, salvo l'occupazione contemporanea da parte di qualche altro processo. Le porte dinamiche sono generalmente assegnate dinamicamente dal sistema operativo e non sono riservate o assegnate in modo permanente a protocolli specifici. Ciò consente una flessibilità maggiore nell'allocazione delle risorse di

comunicazione all'interno di un sistema, evitando conflitti di porte quando più applicazioni comunicano contemporaneamente (Cotton et al., 2011).



Esempio
Vediamo un esempio di porta conosciuta.
Quando visiti un sito web come "www.example.com" nel tuo browser, stai implicitamente comunicando con il server attraverso il protocollo HTTP (Hypertext Transfer Protocol) sulla porta predefinita 80. Il protocollo HTTP è utilizzato per richiedere e trasferire pagine web e altri contenuti attraverso il web.

Vediamo un esempio di porta registrata.
Un'azienda vuole creare un servizio web che consente di sommare due numeri.
Tale servizio deve ricevere in input due numeri e restituire in output un numero che sarà la somma dei due input.
Il programma di calcolo viene associato alla porta 3000.
Tutti i client che vogliono usare il servizio devono collegarsi all'indirizzo del server specificando la porta 3000, inserire nel messaggio due numeri e ricevere come risultato un numero che sarà la somma dei due numeri precedenti (protocollo).

Vediamo un esempio di porta dinamica.
Quando apriamo un sito HTTPS, il browser si collega alla porta 443 del server web. Il computer dell'utente, però, usa anche una propria porta temporanea, scelta automaticamente dal sistema operativo, per identificare quella specifica comunicazione.
Per esempio, il browser potrebbe comunicare così: client: porta 52341 → server: porta 443.
Se l'utente apre più schede o più servizi contemporaneamente, il sistema operativo può assegnare porte dinamiche diverse, in modo da distinguere le varie comunicazioni attive.

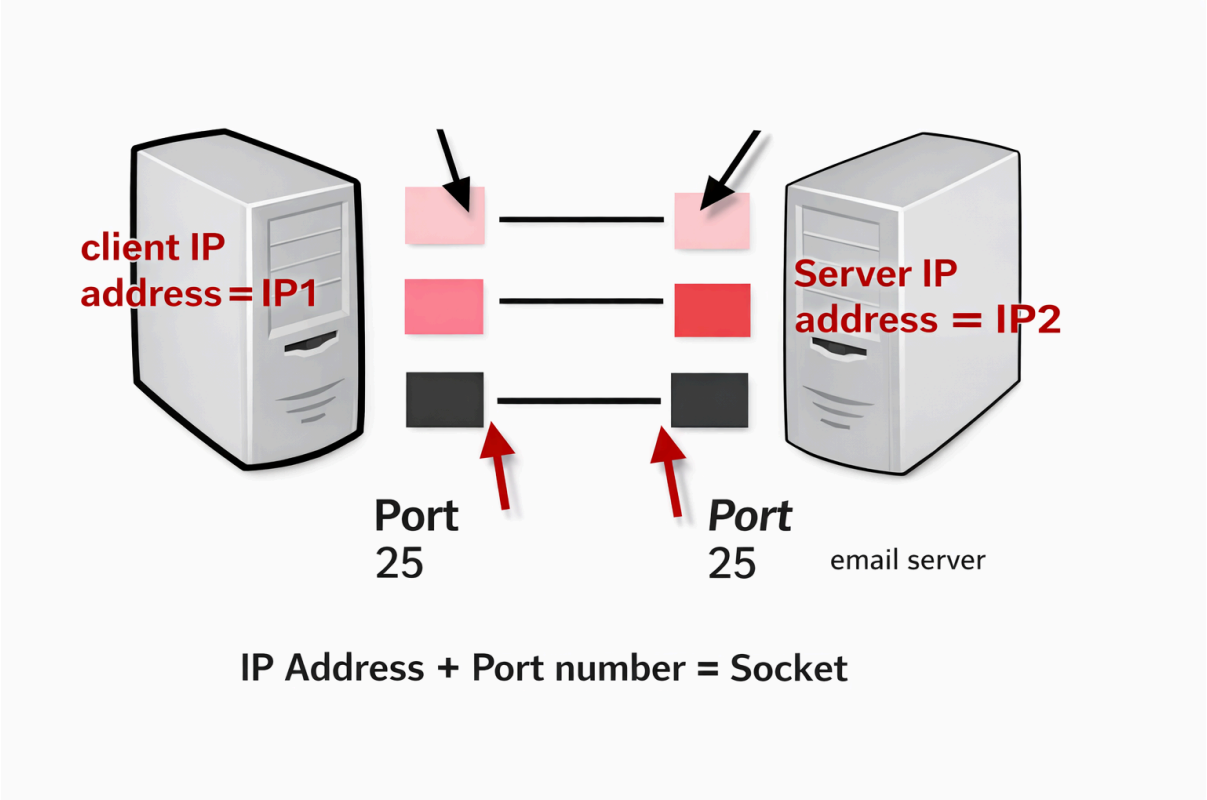
Socket

Quando due dispositivi comunicano attraverso una rete, non basta conoscere il loro indirizzo e la porta: serve anche un meccanismo che gestisca la connessione vera e propria tra i due programmi.
Questo meccanismo è la **socket**.

La socket rappresenta il punto software attraverso cui il programma comunica in rete (Gilligan et al., 2003).

- Nel caso *TCP* può rappresentare una connessione.
- Nel caso *UDP* rappresenta un punto di invio e ricezione di datagrammi.

La socket è l'oggetto software che permette a un programma di inviare e ricevere dati su una rete, sfruttando un indirizzo e una porta:
- L'indirizzo identifica il computer.
- La porta identifica il programma.
- La socket rappresenta il punto di comunicazione tra i due.



Quando un'applicazione vuole comunicare con un'altra applicazione remota, crea una socket:
- Se usa *TCP*, stabilisce una connessione con il server.
- Se usa *UDP*, invia datagrammi verso un indirizzo e una porta di destinazione.

La comunicazione tra le due applicazioni avviene scambiando dati attraverso le loro rispettive socket.

Una socket locale è identificata almeno da un indirizzo, una porta e un protocollo di trasporto, per esempio TCP o UDP (Gilligan et al., 2003).
Nel caso di una connessione *TCP*, la comunicazione viene identificata in modo più preciso dalla combinazione di indirizzo e porta del mittente, indirizzo e porta del destinatario, e protocollo utilizzato (Eddy, 2022).

Con questi elementi, una socket identifica in maniera puntuale una specifica comunicazione tra due host.

Le fasi di una connessione con la socket sono:

- Creazione: il programma crea una socket e la lega a un indirizzo e una porta.
- Connessione (per TCP):
    - Se l'host è un client, cerca il server e stabilisce una connessione.
    - Se l'host è un server, esso rimane in ascolto. Quando un client si connette, il server crea una nuova socket per quella connessione, con indirizzo e porta del client e del server.
- Trasmissione: i dati possono essere inviati e ricevuti attraverso la socket.
- Chiusura: una volta finita la comunicazione, la socket viene chiusa.

Le socket si classificano in due categorie, ciascuna caratterizzata da una propria modalità di comunicazione.

- **Datagram Socket (Socket *UDP*)**: questa tipologia di socket utilizza il protocollo *UDP*. Ciò significa che l'invio dei dati avviene mediante il trasferimento di piccoli datagrammi, senza garantire il corretto ordine d'arrivo e la correttezza dell'informazione. Il client e il server non instaurano una vera e propria connessione, ma il client comunica direttamente con il server quando vuole (Postel, 1980).
- **Stream Socket (Socket *TCP*)**: questa tipologia di socket utilizza una connessione basata sul protocollo *TCP*; è quindi connection-oriented, offre più controlli e affidabilità, ma introduce più overhead rispetto a una socket *UDP* (Eddy, 2022).

Esempio

Di seguito sono descritti i passaggi che avvengono quando un utente richiede tramite browser l'apertura della pagina www.wikipedia.org:

- L'utente scrive www.wikipedia.org nel browser.
- Il sistema usa il *DNS* per ottenere l'indirizzo del server associato a quel nome.
- Il browser apre una socket *TCP* sulla porta 443 per comunicare con il server *HTTPS* (il protocollo *HTTP* ma sicuro) dell'indirizzo appena trovato.
- Il server risponde; i dati passano attraverso la socket.
- Il browser mostra la pagina web.
- Finito lo scambio, la socket viene chiusa.

Si esamina un esempio molto semplice di socket.
In questo esempio il client manda un numero al server e il server calcola il quadrato del numero e restituisce il risultato al client.
Di seguito è riportato il codice del livello applicativo del server e del client, che sfrutta chiamate del livello di trasporto per mandare effettivamente i messaggi.
Le chiamate al livello di trasporto sono:

Lato client

- *CONNETTI(indirizzo, destPort):* richiede una connessione con un host specificato (solo per protocolli orientati alla connessione, es. *TCP*). La funzione restituisce la socket specifica.
- *INVIA (socket, dati)*: invia dati sulla connessione o sulla porta (*TCP* o *UDP*).
- R*ICEVI(socket)*: riceve dati dalla socket (sia *TCP* che *UDP*).
- *CHIUDI(socket)*: termina la connessione (*TCP*).

Lato server

- *ATTENDI(localPort)*: attende richieste di connessione sulla porta indicata (solo *TCP*). La funzione restituisce la socket specifica. Questa funzione rappresenta in modo semplificato l'insieme delle operazioni con cui il server si mette in ascolto e accetta una connessione.
- *INVIA (socket, dati)*: invia dati sulla connessione o sulla porta (*TCP* o *UDP*).
- *RICEVI(socket)*: riceve dati dalla socket (sia *TCP* che *UDP*).
- *CHIUDI(socket)*: chiude la connessione o la porta.

Di seguito si vedono le socket rispettivamente del client e del server.

Socket client

```
indirizzoServer = "127.0.0.1"
portaServer = 65432
numero = 5

socket = CONNETTI(indirizzoServer, portaServer)
INVIA(socket, numero)
risultato = RICEVI(socket)
CHIUDI(socket)
```

Socket server

```
portaServer = 65432

ITERA: # il server quando finisce un'operazione si rimette in ascolto
     socket = ATTENDI(portaServer)
     dato = RICEVI(socket)

     risultato = dato * dato
     INVIA(socket, risultato)
     CHIUDI(socket)
```

Materiale di supporto

Elementi chiave

- **Il livello di trasporto** collega logicamente i processi applicativi presenti su host diversi.
- I principali protocolli di trasporto sono *TCP*, *UDP* e *QUIC*.
- ***TCP*** è orientato alla connessione e affidabile: controlla errori, ordine, flusso e congestione.
- TCP divide i dati in segmenti e ritrasmette quelli persi o danneggiati.
- ***UDP*** è senza connessione e non affidabile, ma è più veloce e introduce meno overhead.
- UDP usa unità chiamate datagrammi ed è adatto a streaming, giochi online, VoIP e applicazioni in tempo reale.
- ***QUIC*** funziona sopra *UDP*, ma aggiunge affidabilità, controllo della congestione e sicurezza tramite *TLS*.
- *QUIC* riduce i tempi di connessione, gestisce meglio i cambi di rete e supporta più flussi indipendenti; è alla base di *HTTP/3*.
- **Una porta** è un numero logico che identifica un'applicazione o un servizio all'interno di un host.
- Le porte possono essere **conosciute**, **registrate** oppure **dinamiche/private**.
- **Una socket** è il punto software attraverso cui un'applicazione invia e riceve dati usando indirizzo, porta e protocollo.
- **Le stream socket** usano *TCP*, mentre **le datagram socket** usano *UDP*.

Esercizi

TCP o UDP?

Una software house sta sviluppando tre applicazioni:

- Un servizio per caricare file importanti su un server.
- Un videogioco multiplayer in cui le posizioni dei giocatori devono essere aggiornate rapidamente.
- Un sistema di videoconferenza in tempo reale.

Per ciascuna applicazione:

- Scegli se utilizzare principalmente TCP oppure UDP.
- Motiva la scelta considerando affidabilità, latenza e overhead.
- Spiega cosa potrebbe accadere se un pacchetto venisse perso.
- Indica in quali casi sarebbe importante ritrasmettere i dati persi e in quali potrebbe essere preferibile continuare la comunicazione.
- Spiega perché UDP può risultare vantaggioso anche se non garantisce la consegna.
- Spiega perché TCP può risultare più adatto quando l'integrità dei dati è più importante della velocità di consegna.

Un segmento TCP va perso

Un client sta trasferendo un documento attraverso TCP.

Il messaggio applicativo viene suddiviso in cinque segmenti:
1 – 2 – 3 – 4 – 5

Durante la trasmissione il segmento 3 viene perso, mentre gli altri raggiungono correttamente il destinatario.

Analizza cosa deve accadere.

- Spiega quale funzione svolgono i numeri di sequenza.
- Spiega come il destinatario può accorgersi che manca una parte dei dati.
- Indica quale ruolo svolgono gli acknowledgment.
- Descrivi cosa deve fare TCP per recuperare il segmento mancante.
- Spiega perché il destinatario deve ricostruire i segmenti nell'ordine corretto.
- Spiega quale differenza ci sarebbe se la stessa comunicazione utilizzasse UDP.
- Durante il trasferimento il destinatario inizia inoltre a elaborare i dati più lentamente.
- Spiega quale funzione di TCP serve a evitare che il mittente continui a inviare dati troppo velocemente.
- Se invece è la rete a risultare sovraccarica, indica quale altro meccanismo di TCP interviene.

Aprire e chiudere una connessione TCP

Un browser deve comunicare con un server tramite TCP.
La comunicazione attraversa queste fasi:
SYN → SYN-ACK → ACK → trasferimento dati → FIN

Rispondi alle seguenti domande:

- Spiega cosa sta chiedendo il client quando invia SYN.
- Spiega cosa comunica il server attraverso SYN-ACK.
- Indica quale funzione svolge l'ACK successivo del client.
- Stabilisci in quale momento può essere considerata stabilita la connessione.
- Spiega cosa avviene durante la fase di trasferimento dei dati.
- Spiega quale significato ha FIN.
- Indica in quale situazione potrebbe invece comparire un RST.
- Spiega perché questa procedura rende TCP un protocollo orientato alla connessione, mentre UDP non richiede una fase equivalente.

Perché esistono le porte?

Un computer sta svolgendo contemporaneamente queste attività:

- Un browser visita un sito HTTPS.
- Un programma invia una e-mail tramite SMTP.
- Un altro programma apre una sessione SSH.
- Un secondo browser apre contemporaneamente un altro sito HTTPS.

Tutte le comunicazioni utilizzano lo stesso computer e quindi lo stesso indirizzo di rete.
Rispondi alle seguenti domande:

- Spiega perché l'indirizzo del computer non è sufficiente per distinguere tutte le comunicazioni.
- Indica quale funzione svolgono le porte.

- Associa ai servizi indicati le porte conosciute citate nella dispensa: SSH, SMTP e HTTPS.
- Spiega la differenza tra porta sorgente e porta di destinazione.

Il primo browser comunica così:
Client: porta 52341 → Server: porta 443
- Spiega il significato dei due numeri.
- Un secondo browser apre un'altra connessione HTTPS. Spiega perché il sistema operativo può assegnargli una porta sorgente differente.
- Spiega in questo contesto la differenza tra porte conosciute, porte registrate e porte dinamiche/private.

Perché QUIC?
Uno smartphone sta caricando una pagina web moderna composta da testo, immagini, script e video.
Durante la navigazione accadono due cose:
- Una delle risorse subisce un ritardo.
- Lo smartphone passa dalla rete Wi-Fi alla rete mobile.

Confronta l'utilizzo di una comunicazione tradizionale basata su TCP con una basata su QUIC.
- Spiega perché QUIC utilizza UDP come base pur offrendo funzionalità di affidabilità.
- Indica quali caratteristiche normalmente associate a TCP vengono aggiunte da QUIC.
- Spiega perché QUIC può ridurre il tempo necessario per iniziare una comunicazione sicura.
- Spiega quale vantaggio offre l'integrazione della sicurezza tramite TLS.
- Descrivi perché il passaggio da Wi-Fi a rete mobile può creare problemi a una connessione TCP tradizionale.
- Spiega perché QUIC può gestire meglio questo cambiamento.
- Una risorsa della pagina subisce una perdita di dati. Spiega perché la gestione di più flussi indipendenti può evitare che il problema rallenti inutilmente anche le altre risorse.
- Spiega perché queste caratteristiche rendono QUIC particolarmente adatto a HTTP/3 e al Web moderno.

Progettare un servizio con le socket
Devi realizzare un semplice servizio client-server.
Il client invia al server un numero intero. Il server calcola il doppio del numero ricevuto e restituisce il risultato.
Utilizza le operazioni astratte presenti nella dispensa:
CONNETTI – ATTENDI – INVIA – RICEVI – CHIUDI

Completa concettualmente il funzionamento del sistema.
- Scrivi la sequenza di operazioni che dovrebbe eseguire il client.
- Scrivi la sequenza di operazioni che dovrebbe eseguire il server.
- Spiega quale funzione svolge ATTENDI(portaServer) sul server.
- Spiega cosa restituisce concettualmente CONNETTI(indirizzo, porta).

- Indica attraverso quale oggetto software client e server inviano e ricevono i dati.
- Spiega perché una socket non coincide semplicemente con una porta.
- Descrivi quali informazioni identificano almeno una socket locale.
- Se il sistema utilizza TCP, indica quale tipo di socket viene utilizzato.
- Spiega cosa cambierebbe concettualmente utilizzando una Datagram Socket UDP al posto di una Stream Socket TCP.
- Il server deve continuare a servire nuovi client dopo aver terminato una richiesta. Spiega perché, dopo CHIUDI(socket), deve tornare nuovamente in ascolto sulla propria porta.

Bibliografia

Allman, M., Paxson, V., & Blanton, E. (2009). *TCP congestion control* (RFC 5681). RFC Editor. https://doi.org/10.17487/RFC5681

Bishop, M. (2022). *HTTP/3* (RFC 9114). RFC Editor. https://doi.org/10.17487/RFC9114

Cotton, M., Eggert, L., Touch, J., Westerlund, M., & Cheshire, S. (2011). *Internet Assigned Numbers Authority (IANA) procedures for the management of the service name and transport protocol port number registry* (RFC 6335). RFC Editor. https://doi.org/10.17487/RFC6335

Eddy, W. (Ed.). (2022). *Transmission Control Protocol (TCP)* (RFC 9293). RFC Editor. https://doi.org/10.17487/RFC9293

Eggert, L., Fairhurst, G., & Shepherd, G. (2017). *UDP usage guidelines* (RFC 8085). RFC Editor. https://doi.org/10.17487/RFC8085

Gilligan, R., Thomson, S., Bound, J., McCann, J., & Stevens, W. (2003). *Basic socket interface extensions for IPv6* (RFC 3493). RFC Editor. https://doi.org/10.17487/RFC3493

Internet Assigned Numbers Authority. (n.d.). *Service name and transport protocol port number registry*. Retrieved August 17, 2026, from https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

Iyengar, J., & Thomson, M. (Eds.). (2021). *QUIC: A UDP-based multiplexed and secure transport* (RFC 9000). RFC Editor. https://doi.org/10.17487/RFC9000

Postel, J. (1980). *User Datagram Protocol* (RFC 768). RFC Editor. https://doi.org/10.17487/RFC768

Thomson, M., & Turner, S. (Eds.). (2021). *Using TLS to secure QUIC* (RFC 9001). RFC Editor. https://doi.org/10.17487/RFC9001

Approfondimenti

- James F. Kurose e Keith W. Ross, Reti di calcolatori e Internet. Un approccio top-down, 8ª ed., Pearson, 2022 — Approfondisce TCP e UDP, il trasferimento affidabile, il controllo del flusso e della congestione, il multiplexing e l'uso delle socket.
- Behrouz A. Forouzan e Firouz Mosharraf, Reti di calcolatori, 2ª ed., a cura di Gaia Maselli, McGraw-Hill Education, 2024 — Presenta in modo visuale i protocolli di trasporto, le porte, le socket e i principali meccanismi di controllo degli errori.

- Andrew S. Tanenbaum, Nick Feamster e David J. Wetherall, Reti di calcolatori, 6ª ed., Pearson, 2023 — Analizza i servizi del livello di trasporto e confronta TCP e UDP, dedicando attenzione all'affidabilità e alla gestione delle connessioni.
- Douglas E. Comer, Internetworking con TCP/IP. Vol. 1: principi, protocolli e architetture, 5ª ed., Pearson, 2006 — Approfondisce TCP, acknowledgment, ritrasmissioni, controllo della congestione, SACK ed ECN all'interno della suite TCP/IP.
- Marco Paganini e Luca Libanore, I nuovi protocolli delle reti e di Internet. Cosa bolle in pentola nelle reti degli anni 2000: QUIC e tutto il resto, In Riga Edizioni, 2022 — Introduce QUIC, UDP-Lite e altri protocolli recenti, chiarendo come velocità, sicurezza e affidabilità vengano integrate nelle comunicazioni moderne.

Nota editoriale

Per la realizzazione di questi materiali ho utilizzato strumenti di intelligenza artificiale generativa come supporto alla scrittura, in particolare per migliorare la forma del testo, riorganizzare i contenuti, correggere la formulazione e velocizzare alcune attività redazionali.

Lavorando autonomamente alla produzione di questi materiali, cerco infatti di automatizzare tutte le attività che possono esserlo, così da poter dedicare più tempo alla ricerca, alla progettazione e allo sviluppo dei contenuti.

L'intelligenza artificiale non determina però i contenuti dell'opera: la scelta degli argomenti, la struttura, le idee, le interpretazioni, gli esempi e l'impostazione didattica sono elaborati da me. L'IA viene quindi utilizzata principalmente come strumento di supporto alla produzione e alla revisione formale, mentre la responsabilità autoriale e progettuale dei contenuti rimane mia.