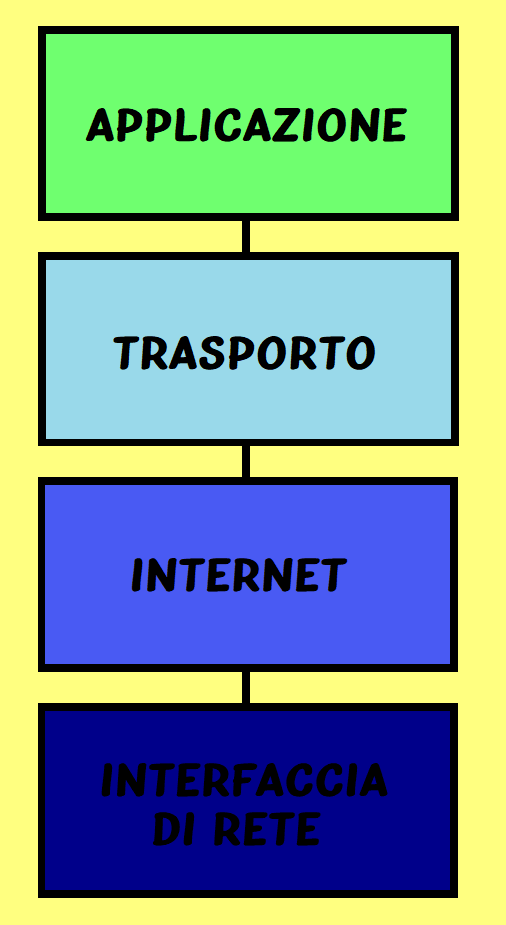
Verifica di TPSIT, classe 5^A ROB.

**Nota: le domande sono tutte a risposta aperta, per cui ogni risposta deve essere motivata con COMPLETEZZA. Disegna schemi o diagrammi quando necessari.**

1. Disegnare la pila TCP/IP indicando il nome di ciascun livello e il nome dei messaggi per i protocolli di ciascun livello. Descrivere brevemente le funzionalità di ciascuno dei livelli TCP/IP.   
   Tracciare e spiegare le due interfacce di separazione presenti nella pila TCP/IP.
2. TCP Port Scanner: uno scanner di porte TCP è un programma che prende di mira un host provando ad effettuare connessioni TCP su tutte le porte possibili per individuare quelle aperte. Una porta è considerata aperta qualora la connessione vada a buon fine.

**Scrivere uno scanner di porte TCP in Python 3.** Utilizzare il metodo *connect\_ex* della classe *socket*: questo metodo è identico al metodo connect con la differenza che ritorna 0 in caso di connessione andata a buon fine, mentre se la connessione fallisce ritorna un numero diverso da 0 e NON genera alcun errore.

1. Spiegare il concetto di multiplazione/demultiplazione.
2. Disegnare la pila TCP/IP indicando il nome di ciascun livello e il nome dei messaggi per i protocolli di ciascun livello. Descrivere brevemente le funzionalità di ciascuno dei livelli TCP/IP.   
   Tracciare e spiegare le due interfacce di separazione presenti nella pila TCP/IP.





In questa sezione troviamo i programmi che accedono alla rete. Ogni programma interagisce con i livelli del modello TCP/IP per inviare o ricevere i dati tramite i messaggi o i flussi (stream).



Comunicazione *end to end*, tra un host e l’altro tramite i processi. Il livello trasporto permette la comunicazione tra il programma e i server o tra programmi.

Può eseguire simultaneamente più processi perché riceve messaggi o flussi contemporaneamente da tutti i processi e allora a ognuno di loro viene assegnato un codice identificativo per riconoscerli.

Il numero massimo di processi che può eseguire simultaneamente sono 2^16-1 cioè circa 65000.

È capace di regolare la velocità dei dati e di determinarne la loro tipologia, divide i messaggi in pacchetti da inviare al livello successivo.

Per la comunicazione utilizza essenzialmente 2 protocolli, il protocollo TCP (Trasmission Control Protocol) e il protocollo UDP (User Datagram Protocol).

Il protocollo **TCP** è affidabile perchè utilizza un acknowledge system, cioè un messaggio di confermata ricezione che conferma la consegna del pacchetto.

Il protocollo **UDP** non è affidabile, è utilizzato nello streaming perché è più veloce e anche se non arriva un frame durante lo streaming l'utente non se ne accorge.



Ha la funzione di realizzare la comunicazione tra computer diversi e capisce se i datagrammi devono essere inviati o se servono sulla macchina locale, nel primo caso controlla anche se i datagrammi ricevuti sono validi. Per la comunicazione utilizza i datagrammi IP del quale esistono 2 versioni: la versione quattro, utilizzata da più tempo, e la versione sei, non ancora totalmente utilizzata.

Oltre ai messaggi implementa anche il viaggio che essi devono fare attraverso la rete ma non è affidabile.

Per comunicare utilizza l’overloading NAT, un meccanismo che permette di cambiare l'indirizzo IP nella rete locale con uno pubblico abbinato a una porta e che permette di risparmiare indirizzi IP se utilizzato correttamente.



Livello che è a cavallo tra il SO e l'hardware (scheda di rete), accetta (in entrata) o trasmette(in uscita) i Frame. Il livello Host-to-network o interfaccia di rete è diviso in due sottolivelli:

* LLC: Logical Link Control (connesso con il livello internet)

Il protocollo che definisce LLC è IEEE 802.2 ed è uguale per tutte le macchine.

Fornisce principalmente tre servizi senza affidabilità e senza sicurezza sull’ordine di arrivo al livello superiore:

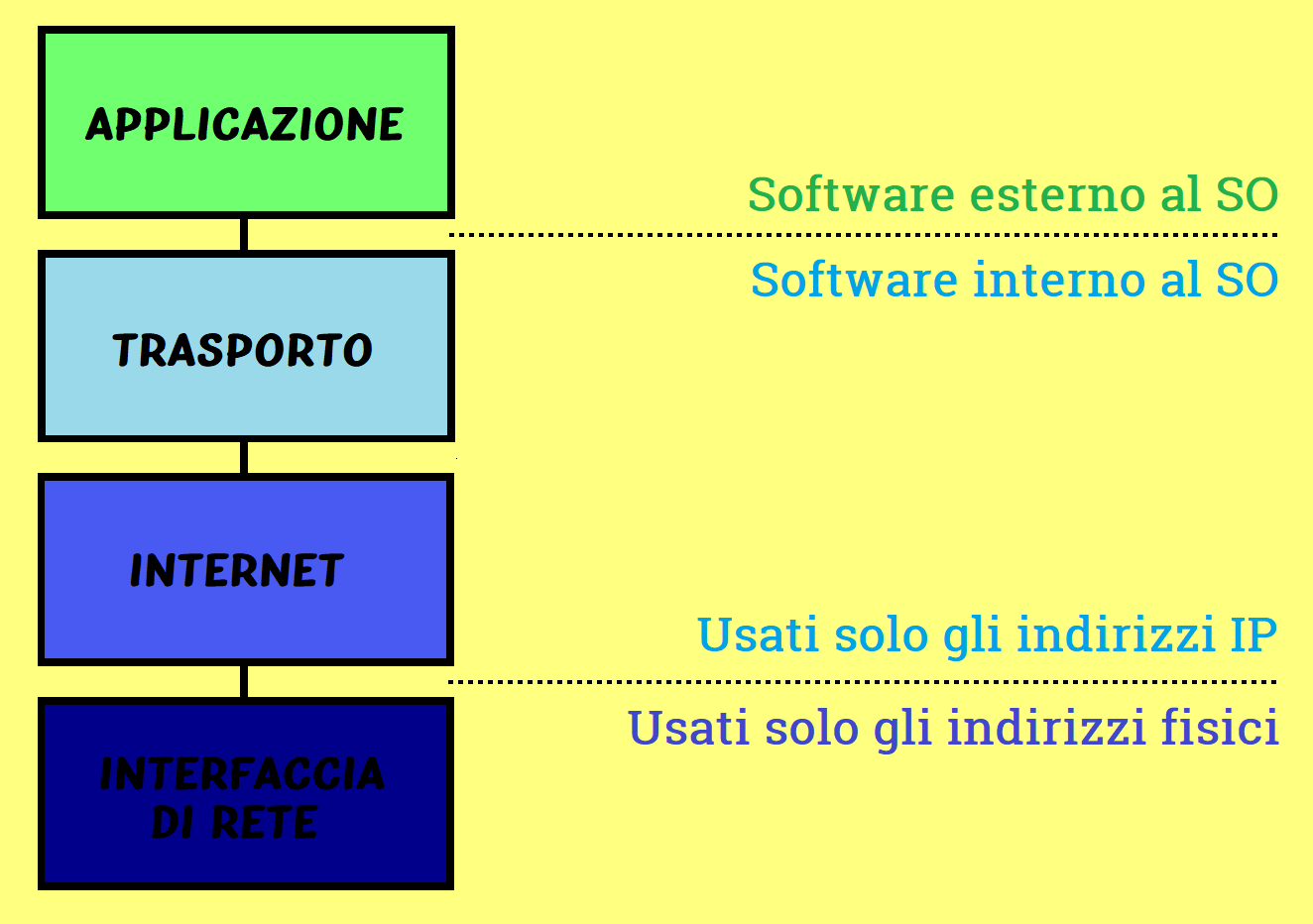
* **UNICAST:** Permette di spedire i frame a una destinazione
* **BROADCAST:** Permette di spedire i frame a più dispositivi
* **MULTICAST:** Permette di spedire i frame a tutti
* MAC: Medium Access Control (connesso con l'hardware)

Questo livello può avere protocolli diversi in base al mezzo fisico utilizzato.

Si occupa di generare i frame in base al protocollo utilizzato, controllare i frame in entrata e accedere al mezzo fisico.

Per comunicare tra di loro ai dispositivi viene assegnato un indirizzo MAC in modo da renderli unici sulla rete a livello mondiale, questo è salvato in una rom sulla scheda di rete ed è formato da 6 byte cioè da 6 coppie di caratteri esadecimali. Di queste sei coppie le prime tre indicano il produttore mentre le seconde tre sono riservate alla macchina.

La comunicazione avviene quando i messaggi partono da un dispositivo con un certo indirizzo MAC per arrivare a un dispositivo con un indirizzo MAC diverso. Per farlo inviano il loro segnale tramite il mezzo fisico, nel caso più host comunicano attraverso lo stesso mezzo fisico è necessario evitare di mischiare le informazioni e per questo è necessario un protocollo che permetta di evitare le collisioni come il CSMA/CD in ethernet (IEEE 802.3)



La prima interfaccia di separazione indica la divisione tra il livello applicazione e il livello trasporto poiché nel livello applicazione agiamo a livello del sistema operativo (varia in base alla macchina) mentre dal livello trasporto in giù siamo in uno stack uguale per tutti i dispositivi che non varia al variare di sistema operativo.

La seconda invece divide il metodo di indirizzamento dei pacchetti, dal livello internet in su si usa l’indirizzo ip e la relativa porta mentre dal livello HOST-TO-NETWORK si usano gli indirizzi MAC (qui implementati) per inviare e ricevere i messaggi.

1. TCP Port Scanner: uno scanner di porte TCP è un programma che prende di mira un host provando ad effettuare connessioni TCP su tutte le porte possibili per individuare quelle aperte. Una porta è considerata aperta qualora la connessione vada a buon fine.

**Scrivere uno scanner di porte TCP in Python 3.** Utilizzare il metodo *connect\_ex* della classe *socket*: questo metodo è identico al metodo connect con la differenza che ritorna 0 in caso di connessione andata a buon fine, mentre se la connessione fallisce ritorna un numero diverso da 0 e NON genera alcun errore.

|  |
| --- |
| """  Scansione porte TCP  """  import socket  *#parametri dell'attacco*  ip = '127.0.0.127' *#IP DELL'HOST DA ATTACCARE*  porta\_min = 0  porta\_max = 1000  *#lista per memorizzare le porte disponibili*  connessioni\_riuscite = []  *#variabile che indica la porta che sto attaccando*  porta= porta\_min  while (porta<porta\_max):  *#stampo dati del tentativo*  print(f"\nprova a connettermi alla porta {porta}")  *#creazione del socket client TCP IPv4*  client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  *#tentativo di connessione alla porta selezionata*  riuscita = client.connect\_ex((ip,porta))  *#salvataggio delle porte funzionanti*  if (riuscita==0):  connessioni\_riuscite.append(porta)  print("\n\tconnessione riuscita")  else:  print("\n\tconnessione fallita")  *#chiusura del socket*  client.close()  *#aggiornamento - passo alla porta successiva*  porta += 1  print ("porte su cui è riuscita la connessione:")  print (connessioni\_riuscite) |

1. Spiegare il concetto di multiplazione/demultiplazione.

La multiplazione e la demultiplazione sono due processi opposti che permettono allo stack TCP/IP di gestire contemporaneamente più flussi di dati, sia di tipo TCP sia di tipo UDP.

Quando dal livello applicazione si passa al livello trasporto si parla di demultiplazione poiché da più programmi si passa a uno stream unico. Al contrario quando arrivano al livello applicazione si parla di demultiplazione poiché da uno stream unico i flussi dati vengono destinati ad applicazioni diverse.

