INDICE

INTRODUZIONE ED OBIETTIVO

INTRODUZIONE: l'esercizio facoltativo presente in questo lavoratorio consiste nella **simulazione di un traffico UDP** controllato in ambiente locale, con l'obiettivo di comprendere la logica di invio e ricezione dei pacchetti attraverso il protocollo UDP e di osservare il comportamento del sistema in assenza o presenza di un listener attivo.

OBIETTIVO: implementare e testare uno script capace di inviare pacchetti UDP da 1 KB verso l'indirizzo di loopback, introducendo un ritardo casuale compreso tra 0 e 0.1 secondi tra un pacchetto e l'altro, al fine di simulare un traffico realistico e verificare la corretta gestione del protocollo tramite l'analisi in Wireshark.

IMPORTAZIONE LIBRERIE

```
#============================================================
# GENOVESE VIKI SUSANNA ES FACOLTATIVO M2W7D4
#============================================================

#IMPORTO LIBRERIE
import socket       # PER CREARE IL SOCKET UDP
import random       # PER IL RITARDO CASUALE
import time         # PER time.sleep()
```

Le tre librerie importate in questa sezione rappresentano gli elementi fondamentali per la gestione del flusso di rete e del controllo temporale all'interno dello script.
La **libreria socket** è il componente principale che consente la creazione e l'utilizzo di **connessioni UDP**.
Attraverso di essa è possibile definire un endpoint di comunicazione che invia pacchetti verso un indirizzo e una porta specifici, senza stabilire una connessione stabile come avviene con il protocollo TCP.
Questo approccio è essenziale nei test di simulazione del traffico, in cui l'obiettivo non è instaurare una sessione, ma misurare la frequenza, la latenza e la risposta del sistema all'invio rapido di pacchetti.
La **libreria random** viene utilizzata per introdurre una componente di **variabilità** nel comportamento del programma.
Generando numeri casuali compresi in un intervallo definito (in questo caso tra 0 e 0.1 secondi), lo script evita che i pacchetti vengano inviati a intervalli costanti e prevedibili, ottenendo così un modello di traffico più realistico, simile a quello che si verifica quando molteplici utenti indipendenti comunicano con un server; questa casualità controllata è cruciale per rendere i test più vicini a scenari operativi reali.
Infine, la **libreria time** ha la funzione di applicare concretamente i **ritardi calcolati**.
Attraverso la chiamata *time.sleep()*, il programma sospende l'esecuzione per la durata specificata, permettendo di sincronizzare la generazione dei pacchetti con il ritardo casuale definito in precedenza.
Questa combinazione tra random e time consente di riprodurre pattern di traffico con maggiore accuratezza, utili per osservare le reazioni del sistema, analizzare la capacità di gestione dei buffer e studiare la resilienza dei servizi di rete in condizioni simulate di carico variabile.

DEFINIZIONE PARAMETRI DI INGRESSO PAYLOAD

```
# RACCOLTA PARAMETRI D'INGRESSO
target_ip = input("IP di destinazione (usa 127.0.0.1 per il lab): ").strip()
target_port = int(input("Porta UDP di destinazione: "))
num_packets = int(input("Numero di pacchetti da inviare: "))

# CREAZIONE PAYLOAD
payload = b"0" * 1024
```

In questa sezione del codice vengono definite le variabili che determinano i parametri di esecuzione del programma, ovvero gli elementi che consentono all'utente di personalizzare il comportamento dello script in base al tipo di test che si intende effettuare.
La prima istruzione raccoglie l'**indirizzo IP** di destinazione attraverso la funzione input(), con un suggerimento esplicito all'uso dell'indirizzo **127.0.0.1**, che corrisponde al **loopback locale**.
La seconda variabile, **target_port**, richiede all'utente di specificare la **porta UDP** su cui inviare i pacchetti, questo valore identifica il punto logico di accesso del destinatario, ovvero la porta sulla quale un'applicazione o un listener potrà ricevere il traffico generato.
La **conversione** esplicita dell'**input in intero** *(int())* garantisce che il dato venga trattato come valore numerico, evitando errori di tipo in fase di invio.
La terza variabile, **num_packets**, determina il **numero totale di pacchetti** che lo script dovrà trasmettere.
Anche qui l'uso di *int()* serve a validare l'input come numero intero, assicurando che il ciclo di invio lavori correttamente con una quantità definita di iterazioni.
Infine, nella sezione dedicata alla **creazione del payload**, la variabile payload viene impostata con una stringa di byte equivalente a **1 kilobyte di dati**.
Questo blocco rappresenta il contenuto effettivo del pacchetto UDP, ovvero ciò che viene inviato al destinatario in ogni iterazione.
La dimensione fissa del payload consente di mantenere costante il carico generato e di valutare con precisione il comportamento della rete e del listener di destinazione sotto condizioni controllate.

ESECUZIONE

```python
# ESECUZIONE
if target_ip != "127.0.0.1":
    print("\n[DRY-RUN] Simulo l'invio senza mandare pacchetti reali.\n")
    for i in range(1, num_packets + 1):
        delay = random.uniform(0, 0.1)          # RITARDO CASUALE TRA 0 E 0.1
        time.sleep(delay)                       # APPLICO IL RITARDO
        print(f"[SIMULATO] Pacchetto #{i} -> {target_ip}:{target_port} (ritardo {delay:.4f}s)")
else:
    print("\n[LAB] Invio reale verso 127.0.0.1.\n")
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # SOCKET UDP

    for i in range(1, num_packets + 1):
        sock.sendto(payload, (target_ip, target_port))        # INVIO PACCHETTO
        delay = random.uniform(0, 0.1)                        # RITARDO CASUALE
        time.sleep(delay)                                     # ASPETTO
        print(f"[INVIATO] Pacchetto #{i} -> {target_ip}:{target_port} (ritardo {delay:.4f}s)")

    sock.close()
    print("\n[OK] Invio completato.")
```

In questa parte del codice viene gestita la **fase di esecuzione vera e propria** dello script, dove avviene l'invio o la simulazione dei pacchetti UDP in base all'indirizzo IP specificato dall'utente.
Il programma distingue due modalità operative: una di **simulazione (Dry-Run)** e una di **invio reale (Lab)**.
Nel primo caso, se l'indirizzo di destinazione non corrisponde a 127.0.0.1, lo script non invia pacchetti reali ma esegue soltanto una simulazione, mostrando a schermo i dettagli di ogni invio (numero progressivo, destinazione e ritardo applicato) senza generare traffico di rete effettivo; questa opzione garantisce la sicurezza dell'esecuzione, evitando di trasmettere dati verso host esterni e permettendo comunque di analizzare il comportamento logico del ciclo di invio.
Nella seconda modalità, attiva quando la destinazione è il loopback (127.0.0.1), lo script apre un **socket UDP** tramite la libreria socket e procede con l'invio reale dei pacchetti.
All'interno di un **ciclo for**, ogni pacchetto da 1 KB viene trasmesso alla porta indicata dall'utente; subito dopo, viene calcolato un ritardo casuale compreso tra 0 e 0.1 secondi, durante il quale il programma sospende temporaneamente l'esecuzione grazie a **time.sleep(delay**).
Questo meccanismo introduce un intervallo variabile tra i pacchetti, simulando una situazione di traffico non costante, più vicina al comportamento reale di più client indipendenti.
Al termine del ciclo, il socket viene chiuso con **sock.close()** per rilasciare correttamente le risorse di rete, e il programma conferma l'avvenuto completamento dell'invio. Nel complesso, questa sezione traduce i parametri definiti dall'utente in

un processo di trasmissione controllata, consentendo di testare la comunicazione UDP locale e di osservare il comportamento del sistema in risposta a pacchetti inviati con frequenza casuale.

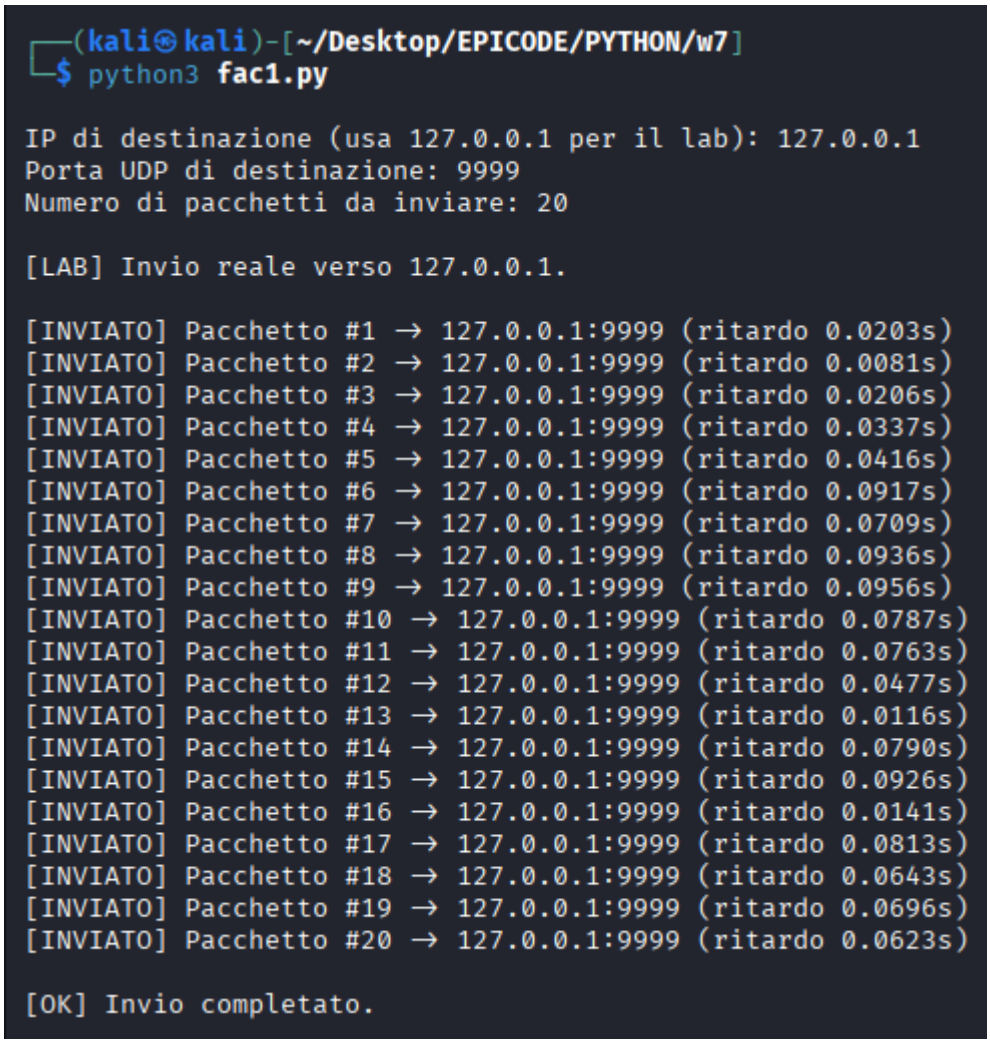
INVIO PACCHETTI UDP

```
┌──(kali㉿kali)-[~/Desktop/EPICODE/PYTHON/w7]
└─$ python3 fac1.py

IP di destinazione (usa 127.0.0.1 per il lab): 127.0.0.1
Porta UDP di destinazione: 9999
Numero di pacchetti da inviare: 20

[LAB] Invio reale verso 127.0.0.1.

[INVIATO] Pacchetto #1 → 127.0.0.1:9999 (ritardo 0.0203s)
[INVIATO] Pacchetto #2 → 127.0.0.1:9999 (ritardo 0.0081s)
[INVIATO] Pacchetto #3 → 127.0.0.1:9999 (ritardo 0.0206s)
[INVIATO] Pacchetto #4 → 127.0.0.1:9999 (ritardo 0.0337s)
[INVIATO] Pacchetto #5 → 127.0.0.1:9999 (ritardo 0.0416s)
[INVIATO] Pacchetto #6 → 127.0.0.1:9999 (ritardo 0.0917s)
[INVIATO] Pacchetto #7 → 127.0.0.1:9999 (ritardo 0.0709s)
[INVIATO] Pacchetto #8 → 127.0.0.1:9999 (ritardo 0.0936s)
[INVIATO] Pacchetto #9 → 127.0.0.1:9999 (ritardo 0.0956s)
[INVIATO] Pacchetto #10 → 127.0.0.1:9999 (ritardo 0.0787s)
[INVIATO] Pacchetto #11 → 127.0.0.1:9999 (ritardo 0.0763s)
[INVIATO] Pacchetto #12 → 127.0.0.1:9999 (ritardo 0.0477s)
[INVIATO] Pacchetto #13 → 127.0.0.1:9999 (ritardo 0.0116s)
[INVIATO] Pacchetto #14 → 127.0.0.1:9999 (ritardo 0.0790s)
[INVIATO] Pacchetto #15 → 127.0.0.1:9999 (ritardo 0.0926s)
[INVIATO] Pacchetto #16 → 127.0.0.1:9999 (ritardo 0.0141s)
[INVIATO] Pacchetto #17 → 127.0.0.1:9999 (ritardo 0.0813s)
[INVIATO] Pacchetto #18 → 127.0.0.1:9999 (ritardo 0.0643s)
[INVIATO] Pacchetto #19 → 127.0.0.1:9999 (ritardo 0.0696s)
[INVIATO] Pacchetto #20 → 127.0.0.1:9999 (ritardo 0.0623s)

[OK] Invio completato.
```

Nel terminale è visibile l'output generato durante l'esecuzione dello script, che conferma il corretto funzionamento della modalità di invio reale.
Dopo aver inserito l'indirizzo di loopback 127.0.0.1, la porta di destinazione 9999 e il numero totale di pacchetti da inviare (in questo caso 20), il programma ha avviato la trasmissione dei pacchetti UDP da 1 KB ciascuno.
Ogni riga del risultato mostra il numero progressivo del pacchetto, l'indirizzo e la porta di destinazione, seguiti dal valore del ritardo casuale applicato prima dell'invio successivo.
Tali ritardi, compresi tra 0 e 0.1 secondi, sono stati generati dinamicamente dalla libreria random e applicati tramite time.sleep(), producendo un flusso di traffico irregolare e realistico.
La sezione finale del messaggio, [OK] Invio completato., indica che il ciclo di invio si è concluso correttamente senza errori.

Questo output conferma che lo script è in grado di gestire in modo stabile l'invio di pacchetti multipli con temporizzazione variabile, simulando efficacemente un traffico UDP distribuito in ambiente di test.

RICEZIONE PACCHETTI SUL LISTENER



Nel terminale è mostrata la ricezione effettiva dei pacchetti inviati dallo script.
Il comando **nc -lu -p 9999** ha attivato un listener UDP sulla porta 9999, permettendo di intercettare e visualizzare in tempo reale il contenuto dei pacchetti trasmessi.
Ogni pacchetto ricevuto contiene un payload di **1 KB di dati**, rappresentato da una sequenza continua di caratteri "0", generati artificialmente dallo script tramite la variabile payload = b"0" * 1024.
La visualizzazione di queste stringhe conferma che la comunicazione tra il mittente e il listener (netcat) è avvenuta correttamente, il traffico UDP è stato instradato attraverso il loopback locale (127.0.0.1), ed infine ciascun pacchetto ha mantenuto intatto il suo contenuto durante la trasmissione.
Questo output rappresenta la prova diretta della riuscita dell'invio e della ricezione dei dati, validando il corretto funzionamento sia della parte di generazione che di gestione del traffico UDP.

ANALISI DEL TRAFFICO UDP IN WIRESHARK

udp.port == 9999

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
2	0.021148371	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
3	0.033409957	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
4	0.054412267	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
5	0.103441634	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
6	0.147831240	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
7	0.243251376	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
8	0.316219155	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
9	0.410341013	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
10	0.506649971	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
11	0.618149279	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
12	0.734159766	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
13	0.832700083	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
14	0.851475274	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
15	0.932787334	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
16	1.074021308	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
17	1.089236075	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
18	1.171603546	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
19	1.236865000	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024
20	1.321278615	127.0.0.1	127.0.0.1	UDP	1066	40155 → 9999 Len=1024

L'immagine mostra la cattura del traffico generato dallo script, analizzato con **Wireshark** utilizzando il filtro udp.port == 9999, che isola esclusivamente i pacchetti diretti o provenienti dalla porta corrispondente.
Ogni riga rappresenta un singolo pacchetto inviato attraverso l'interfaccia di loopback confermando che la comunicazione si è svolta interamente in ambito locale, senza alcun transito verso la rete esterna.
Nelle colonne principali sono riportate informazioni fondamentali per l'analisi: il **timestamp** di invio, l'indirizzo sorgente e di destinazione (identici, trattandosi del loopback), il **protocollo UDP**, la **lunghezza complessiva del pacchetto** (1066 byte, corrispondenti ai 1024 byte di payload più gli header di livello rete e trasporto), e il riepilogo della porta di destinazione e della dimensione del contenuto.
L'intervallo temporale irregolare tra un pacchetto e l'altro, visibile nella colonna "Time", riflette l'effetto del ritardo casuale introdotto dallo script (compreso tra 0 e 0.1 secondi), dimostrando che il meccanismo di temporizzazione funziona correttamente.
Questa acquisizione fornisce dunque la conferma visiva e tecnica che l'invio dei pacchetti UDP è avvenuto con la frequenza prevista e che ogni datagramma ha mantenuto la dimensione e la struttura stabilite in fase di configurazione.

CONCLUSIONE

L’attività ha permesso di comprendere in modo pratico il funzionamento del protocollo UDP e il comportamento del traffico generato artificialmente in un ambiente controllato.
Lo script realizzato ha eseguito correttamente la trasmissione di pacchetti da 1 KB verso l’indirizzo di loopback, integrando un ritardo casuale tra un invio e l’altro, simulando così un flusso di richieste asincrone e realistiche.
L’analisi con **Wireshark** ha confermato la coerenza dei pacchetti inviati, mostrando la corrispondenza tra dimensione del payload, indirizzi di origine e destinazione, e intervalli temporali variabili.
L’esecuzione con **Netcat** ha inoltre validato la ricezione dei pacchetti, evidenziando l’integrità del contenuto trasmesso e la stabilità del socket UDP nel gestire traffico consecutivo.

GLOSSARIO

UDP (User Datagram Protocol): protocollo di trasporto non orientato alla connessione, utilizzato per l'invio rapido di pacchetti ("datagrammi") senza richiedere una conferma di ricezione. È più veloce ma meno affidabile rispetto a TCP.
Socket: punto di comunicazione logico tra due dispositivi o processi che consente l'invio e la ricezione di dati attraverso una rete. Nel contesto dello script è stato utilizzato un socket UDP locale.
Loopback (127.0.0.1): indirizzo IP speciale che consente a un computer di comunicare con sé stesso. Serve per testare applicazioni e protocolli di rete senza utilizzare una connessione esterna.
Payload: parte del pacchetto che contiene i dati effettivi trasmessi.
Nel test realizzato corrisponde a 1 KB di byte "0", generati dallo script come contenuto dei pacchetti UDP.
Porta (Port): numero identificativo associato a un servizio o processo di rete. La porta 9999 è stata usata per l'ascolto e la ricezione dei pacchetti.
Netcat (nc): strumento da riga di comando che permette di aprire connessioni TCP o UDP, inviare e ricevere dati, e testare servizi di rete in modo semplice e diretto.
Wireshark: software di analisi del traffico di rete che consente di catturare, filtrare e ispezionare i pacchetti scambiati tra dispositivi o processi, utile per verificare la correttezza e la struttura delle comunicazioni.
Ritardo casuale (random delay): intervallo di tempo variabile introdotto intenzionalmente tra l'invio di due pacchetti consecutivi, generato in modo casuale per simulare un comportamento più realistico e non deterministico del traffico.
time.sleep(): funzione della libreria time che sospende temporaneamente l'esecuzione del programma per un numero di secondi specificato, utilizzata qui per applicare il ritardo casuale tra un invio e l'altro.
Dry-Run: modalità di simulazione che esegue tutte le istruzioni del programma senza inviare realmente pacchetti in rete, utile per testare il funzionamento logico dello script in sicurezza.