Nombre: Luis Alberto Vargas González.

Fecha: 25/02/2025

Actividad y unidad: Unidad 8 Actividad 1.

Clase: Sistemas Distribuidos.

Maestría en Ciberseguridad.

Descripción de programas.

En esta práctica se ejercita la programación de un reproductor de video en streaming, que; como bien sabemos, es la modalidad de consumo de contenido digital más ampliamente usada desde hace 10 años; esta modalidad nos permite que un usuario no necesariamente necesite descargar y guardar el contenido que estará visualizando, sin necesidad de utilizar sus recursos de almacenamiento en sus dispositivos de preferencia, por lo que es ideal este tipo de transmisiones de flujo constante en televisores , pantallas planas, computadores. Además, como hemos estado usando en los demás programas, se utiliza un sistema peer to peer para el envío y recepción de paquetes entre cliente y servidor. Adicional a esto, se crea la interfaz de usuario para poder manejar con botones, la reproducción, pausa, adelanto o retraso del video según sean las necesidades del cliente.

Código Nodo 1.

```
import socket
```

```
import os
import time

# Configuración del nodo
NODOS = [("127.0.0.1", 56433)] # Nodo 2
TAMANO_PAQUETE = 4096
DESPLAZAMIENTO_CESAR = 7

# Función de cifrado César para archivos binarios
def cifrar_cesar(datos, desplazamiento):
return bytes((byte + desplazamiento) % 256 for byte in datos)

# Función para enviar un archivo cifrado
def enviar_archivo(ruta_archivo):
if not os.path.exists(ruta_archivo):
print("El archivo no existe.")
return

nombre_archivo = os.path.basename(ruta_archivo)
tamano_archivo = os.path.getsize(ruta_archivo)
print(f"Enviando archivo cifrado: {nombre_archivo}, Tamaño:
{tamano_archivo} bytes")

for ip, puerto in NODOS:
try:
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
client_socket.connect((ip, puerto))
print(f"Conectado a {ip}:{puerto}, enviando archivo...")

# Enviar metadatos del archivo (nombre y tamaño)
```

```
metadatos = f"{nombre_archivo}|{tamano_archivo}\n"
client_socket.sendall(metadatos.encode())

# Enviar archivo en fragmentos cifrados
with open(ruta_archivo, "rb") as archivo:
while fragmento := archivo.read(TAMANO_PAQUETE):
fragmento_cifrado = cifrar_cesar(fragmento, DESPLAZAMIENTO_CESAR)
client_socket.sendall(fragmento_cifrado)

print(f"Archivo {nombre_archivo} cifrado y enviado con éxito a {ip}:{puerto}")

except ConnectionRefusedError:
print(f"No se pudo conectar a {ip}:{puerto}. Reintentando en 2s...")
time.sleep(2)

# Configuración y ejecución del nodo 1
if __name__ == "__main__":
time.sleep(3) # Esperar para iniciar el servidor en el otro nodo
enviar_archivo("Gatos-Graciosos .mp4")
```

En este programa se hace el envío del archivo fragmentado y cifrado al nodo 2 que es el que lo reúne y adicionará paa su posterior reproducción; este programa es prácticamente igual al de las prácticas anteriores, la fragmentación del video se hace en partes de paquetes consistentes de 4096 bytes como podemos notar en el código, esto nos permite un mejor manejo de archivos de videos que regularmente suelen pesar más de 50 MB.

Código Nodo 2.

```
import socket
import os
import cv2
import tkinter as tk
from PIL import Image, ImageTk
import threading

# Configuración del nodo receptor
MI_IP = "127.0.0.1"
MI_PUERTO = 56433 # Nodo 2
CARPETA_DESTINO = "./videos_recibidos"
os.makedirs(CARPETA_DESTINO, exist_ok=True)

TAMANO_PAQUETE = 4096
```

```
DESPLAZAMIENTO_CESAR = 7

# Función de descifrado César para archivos binarios
def descifrar_cesar(datos, desplazamiento):
return bytes ((byte - desplazamiento) % 256 for byte in datos)

# Función para recibir archivos y descifrarlos
def iniciar_servidor_tcp():
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((MI_IP, MI_PUERTO))
server_socket.listen()
print(f"Servidor TCP escuchando en {MI_IP}:{MI_PUERTO}")

while True:
conn, addr = server_socket.accept()
print(f"Conexión establecida con {addr}")

try:
metadata = conn.recv(1024).decode().strip()
if "|" not in metadata:
raise ValueError(f"Formato de metadatos incorrecto: {metadata}")

nombre_archivo, tamano_archivo = metadata.split("|", 1)
tamano_archivo = int(tamano_archivo.strip())

print(f"Recibiendo archivo cifrado: {nombre_archivo}, Tamaño:
{tamano_archivo} bytes")

ruta_completa = os.path.join(CARPETA_DESTINO, nombre_archivo)
```

```
with open(ruta_completa, "wb") as archivo:
bytes_recibidos = 0
while bytes_recibidos < tamano_archivo:
fragmento_cifrado = conn.recv(TAMANO_PAQUETE)
if not fragmento_cifrado:
break
fragmento_descifrado = descifrar_cesar(fragmento_cifrado,
DESPLAZAMIENTO_CESAR)
archivo.write(fragmento_descifrado)
bytes_recibidos += len(fragmento_cifrado)

print(f"Archivo {nombre_archivo} recibido y descifrado con éxito. Guardado
en {ruta_completa}")

# Iniciar la reproducción del video
reproductor = VideoPlayer(ruta_completa)
reproductor.run()

except Exception as e:
print(f"Error al recibir archivo: {e}")

# Clase para la GUI del reproductor de video
class VideoPlayer:
def __init__(self, video_path):
self.video_path = video_path
self.paused = False
self.cap = cv2.VideoCapture(video_path)

self.root = tk.Tk()
self.root.title("Reproductor de Video")
```

```
self.root.geometry("800x600")

self.panel = tk.Label(self.root)
self.panel.pack()

self.btn_play_pause = tk.Button(self.root, text="Play/Pause",
command=self.toggle_play)
self.btn_play_pause.pack(side=tk.LEFT)

self.btn_rewind = tk.Button(self.root, text="<<", command=self.rewind)
self.btn_rewind.pack(side=tk.LEFT)

self.btn_forward = tk.Button(self.root, text=">>", command=self.forward)
self.btn_forward.pack(side=tk.LEFT)

self.update_video()

def toggle_play(self):
self.paused = not self.paused

def rewind(self):
self.cap.set(cv2.CAP_PROP_POS_FRAMES, max(0,
self.cap.get(cv2.CAP_PROP_POS_FRAMES) - 30))

def forward(self):
total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
new_pos = min(total_frames, self.cap.get(cv2.CAP_PROP_POS_FRAMES) + 30)
self.cap.set(cv2.CAP_PROP_POS_FRAMES, new_pos)

def update_video(self):
```

```
if not self.paused:
ret, frame = self.cap.read()
if ret:
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame = Image.fromarray(frame)
frame = ImageTk.PhotoImage(frame)
self.panel.config(image=frame)
self.panel.image = frame
else:
self.cap.set(cv2.CAP_PROP_POS_FRAMES, 0) # Reiniciar el video
self.root.after(30, self.update_video)

def run(self):
self.root.mainloop()

# Iniciar el servidor en un hilo
if __name__ == "__main__":
servidor = threading.Thread(target=iniciar_servidor_tcp)
servidor.start()
```

Este es el nodo de recepción de videos, en donde podemos encontrar que el archivo al ser recibido se desfragmenta y se guarda en la carpeta correspondiente de archivos recibidos, adicionalmente encontramos la clase VideoPlayer la cual nos aporta la GUI del servicio de video, se crea el panel de reproducción en el cual se incluyen los botones de play/pause, forward y rewind. Así mismo, se incluye la función de actualizar video, la cual permite al usuario poder mantener en sesión los cambios que haya hecho a su video, ya

sea que haya adelantado n cantidad de segundos, atrasado n cantidad de segundos y el minuto y segundo exacto donde dejó de reproducir su video.

Conclusiones.

Podemos argumentar que; se realizó de manera satisfactoria la práctica aquí contenida, se logró hacer él envió de videos de un nodo a otro, junto con su cifrado (cesar, el cual cifra los bytes del video), así como la implementación de la GUI que nos permite adelantar, atrasar, poner en pausa y reproducir el video. Siendo esto último lo más complicado debido a que se necesita manipular los paquetes del archivo de video con una librería especial llamada CV2 que es la que además se usa para manipulación de video en IA en el campo de Visión Artificial.