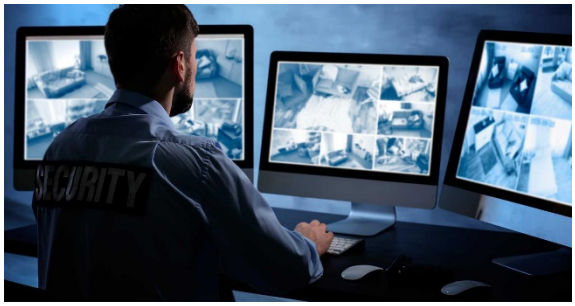
Enviar video a través de sockets UDP

El protocolo UDP (User Datagram Protocol) es ampliamente utilizado para la transmisión de video debido a sus características que ofrecen ventajas significativas en comparación con TCP (Transmission Control Protocol). Uno de los principales beneficios de UDP es su bajo overhead en comparación con TCP. UDP no realiza una verificación exhaustiva de errores ni asegura la entrega de los paquetes, lo que reduce la latencia y mejora la eficiencia en la transmisión de video en tiempo real. Esta característica es crucial para aplicaciones como la transmisión en vivo o videoconferencias, donde la sincronización y la velocidad son más importantes que la corrección de errores, ya que un pequeño retraso en la visualización puede impactar negativamente en la experiencia del usuario.

El proceso de envío de video usando UDP implica la apertura de un socket UDP que permite la comunicación sin conexión. A diferencia de TCP, que establece una conexión estable entre el emisor y el receptor antes de la transferencia de datos, UDP envía los datos en forma de datagramas individuales sin necesidad de establecer una conexión previa. Esto se traduce en una configuración más simple y una menor sobrecarga en la transmisión de datos. El servidor de video abre un socket UDP, configura la dirección IP y el puerto del destinatario, y comienza a enviar frames de video en paquetes a través de este socket. La falta de confirmaciones de recepción y retransmisiones reduce el tiempo requerido para el envío y la recepción de datos.

En el contexto de la transmisión de video, UDP permite una comunicación continua y rápida entre el servidor y el cliente. El proceso típicamente implica capturar video en el servidor, codificar los frames en un formato adecuado (como JPEG o H.264), y luego enviar estos frames codificados a través de UDP a la dirección IP del cliente y el puerto especificado. El cliente recibe los datagramas UDP, los decodifica y los muestra en tiempo real. La capacidad de UDP para manejar grandes cantidades de datos rápidamente es fundamental para aplicaciones de video, donde la calidad de la experiencia del usuario depende en gran medida de la fluidez y la sincronización del video.

La monitorización y transmisión de video a través de UDP también tiene aplicaciones prácticas en diversas áreas. En la vigilancia de seguridad, por ejemplo, las cámaras de vigilancia transmiten video en tiempo real a estaciones de monitoreo utilizando UDP para minimizar el retraso en la visualización de las imágenes. En el ámbito de las videoconferencias y las transmisiones en vivo, UDP se utiliza para enviar el video de manera eficiente y sin interrupciones, asegurando que la comunicación sea fluida y sin problemas. La capacidad de UDP para enviar datos de manera continua sin el overhead de las confirmaciones y la corrección de errores hace que sea ideal para estos usos.



Desarrollo:

El motivo de esta práctica es poder enviar video a través de un servidor a un cliente, ahora se dará una breve explicación de lo que puede realizar el cliente y servidor, ya que cada uno está utilizando el protocolo UDP.

Este código implementa un cliente en Python que recibe y muestra frames de video a través del protocolo UDP. Utiliza varias bibliotecas para el procesamiento y visualización de video, así como para la comunicación en red. Primero, se importan las bibliotecas necesarias: cv2 para el procesamiento de video con OpenCV, imutils para utilidades adicionales en imágenes, socket para la comunicación en red, numpy para manipulación de arrays, time para el cálculo de FPS, y base64 para la codificación y decodificación de datos en base64.

El cliente establece un socket UDP con un tamaño de buffer de 65,536 bytes, lo que es adecuado para recibir grandes cantidades de datos como frames de video. Se crea un socket UDP mediante socket.socket(socket.AF_INET, socket.SOCK_DGRAM), se configura el tamaño del buffer de recepción con client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, BUFF_SIZE), y se obtiene la IP del servidor a la que se conectará (en este caso, 'TU_IP') y el puerto (8080). A continuación, el cliente envía un mensaje de conexión al servidor utilizando client_socket.sendto(message, (host_ip, port)), lo que indica al servidor que está listo para recibir datos.

En el bucle principal, el cliente entra en un ciclo continuo en el que recibe paquetes de datos del servidor. Utiliza client_socket.recvfrom(BUFF_SIZE) para recibir datos del socket, que se decodifican desde el formato base64 a bytes y luego se convierten a un array de numpy usando np.fromstring(data, dtype=np.uint8). Este array se decodifica en una imagen con cv2.imdecode(npdata, 1), y se le agrega un texto que muestra el FPS (frames por segundo) utilizando cv2.putText(). El frame de video se muestra en una ventana titulada "Recibiendo video" con cv2.imshow().

Para calcular el FPS, el cliente mantiene un conteo de frames y el tiempo transcurrido. Cada 20 frames, se calcula el FPS actual y se actualiza el contador. Si se presiona la tecla 'q', el bucle se rompe y el socket se cierra con client_socket.close(). Este proceso asegura que el video se muestre en tiempo real con una actualización continua y que el rendimiento del cliente pueda ser monitoreado en términos de FPS. En resumen, el código proporciona una manera eficiente de recibir y visualizar video en tiempo real a través de una conexión UDP, utilizando técnicas de procesamiento y visualización de imágenes con OpenCV y numpy.

Ahora vamos por pasos, se deben de instalar las bilbiotecas necesarias, en caso de no tener las bilbiotecas mencionadas se pueden instalar de la siguiente manera:

```
pip install opencv-python
```

También otra biblioteca que se solicita instalar comúnmente es la siguiente:

```
pip install imutils
```

Ahora una vez comprendiendo lo que realiza el cliente se agregará lo siguiente al archivo cliente.py:

```
# This is client code to receive video frames over UDP
import cv2, imutils, socket
import numpy as np
import time
import base64

BUFF_SIZE = 65536
client_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
client_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
host_name = socket.gethostname()
host_ip = 'INSERTA_TU_IP'
print(host_ip)
port = 8080
message = b'Conectado'

client_socket.sendto(message,(host_ip,port))
fps,st,frames_to_count,cnt = (0,0,20,0)
while True:
	packet,_ = client_socket.recvfrom(BUFF_SIZE)
	data = base64.b64decode(packet,' /')
	npdata = np.fromstring(data,dtype=np.uint8)
	frame = cv2.imdecode(npdata,1)
	frame = cv2.putText(frame,'FPS:
'+str(fps),(10,40),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
	cv2.imshow("Recibiendo video",frame)
	key = cv2.waitKey(1) & 0xFF
	if key == ord('q'):
		client_socket.close()
		break
	if cnt == frames_to_count:
		try:
			fps = round(frames_to_count/(time.time()-st))
			st=time.time()
			cnt=0
		except:
			pass
	cnt+=1
```

Ahora se debe de comprender que es lo que realiza el servidor para el envío del vídeo que se va a transmitir,.

Este código implementa un servidor en Python que transmite frames de video a través del protocolo UDP. Utiliza varias bibliotecas para capturar, procesar, codificar y enviar los frames de video. El proceso comienza con la configuración del socket UDP, que se crea utilizando socket.socket(socket.AF_INET, socket.SOCK_DGRAM). Este socket se configura para recibir datos con un buffer de tamaño 65,536 bytes, lo suficientemente grande para manejar los paquetes de video. La dirección IP del servidor se establece como 'TU_IP' y el puerto como 8080. Luego, el servidor se vincula a esta dirección y puerto con server_socket.bind(socket_address), lo que permite al servidor escuchar las conexiones entrantes.

Para la captura de video, el servidor utiliza cv2.VideoCapture(0), que abre la webcam conectada al sistema. Si se desea usar un archivo de video en lugar de la webcam, se puede reemplazar el 0 con la ruta del archivo. Una vez que el servidor está listo y capturando video, entra en un bucle donde espera recibir un mensaje de conexión de un cliente usando server_socket.recvfrom(BUFF_SIZE). Cuando se recibe una solicitud de conexión, el servidor imprime la dirección del cliente y comienza a transmitir los frames de video a esa dirección.

Dentro del bucle de transmisión, el servidor lee frames del video con vid.read() y redimensiona cada frame a un ancho de 400 píxeles usando imutils.resize(). Cada frame se codifica en formato JPEG utilizando cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 80]), lo que convierte el frame en un buffer de bytes. Este buffer se codifica en base64 para garantizar que los datos puedan ser transmitidos de manera segura a través de UDP con base64.b64encode(buffer). El frame codificado en base64 se envía al cliente usando server_socket.sendto(message, client_addr).

Para proporcionar retroalimentación visual sobre la transmisión, el servidor agrega un texto al frame que muestra el FPS (frames por segundo) actual utilizando cv2.putText(). Luego, el frame se muestra en una ventana titulada 'Transmitiendo video' con cv2.imshow(). El cálculo de FPS se realiza cada 20 frames para evaluar el rendimiento del servidor. Si se presiona la tecla 'q', el servidor cierra el socket con server_socket.close() y termina la transmisión. Durante la ejecución, el servidor calcula el FPS y actualiza el contador de frames para mantener la tasa de cuadros por segundo en la ventana de visualización.

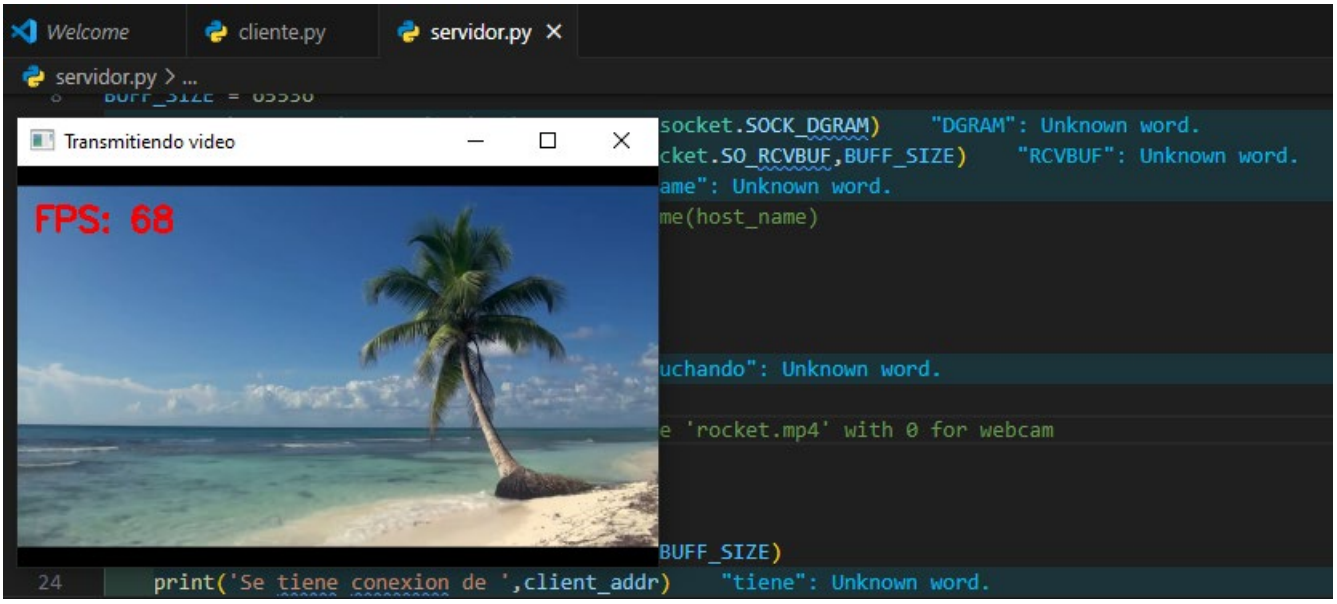
Una vez comprendido el flujo que sigue el servidor se debe agregar el siguiente fragmento de código al archivo servidor.py:

```
import cv2, imutils, socket
import numpy as np
import time
import base64

BUFF_SIZE = 65536
server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_socket.setsockopt(socket.SOL_SOCKET,socket.SO_RCVBUF,BUFF_SIZE)
host_name = socket.gethostname()
host_ip = 'TU_IP'#  socket.gethostbyname(host_name)
print(host_ip)
port = 8080
socket_address = (host_ip,port)
server_socket.bind(socket_address)
print('Escuchando a:',socket_address)

vid = cv2.VideoCapture(0) #  utilizer 0 para usar webcam
fps,st,frames_to_count,cnt = (0,0,20,0)

while True:
      msg,client_addr = server_socket.recvfrom(BUFF_SIZE)
      print('Se tiene conexion de ',client_addr)
      WIDTH=400
      while(vid.isOpened()):
            _,frame = vid.read()
            frame = imutils.resize(frame,width=WIDTH)
```

```
            encoded,buffer =
cv2.imencode('.jpg',frame,[cv2.IMWRITE_JPEG_QUALITY,80])
            message = base64.b64encode(buffer)
            server_socket.sendto(message,client_addr)
            frame = cv2.putText(frame,'FPS:
'+str(fps),(10,40),cv2.FONT_HERSHEY_SIMPLEX,0.7,(0,0,255),2)
            cv2.imshow('Transmitiendo video',frame)
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'):
                  server_socket.close()
                  break
            if cnt == frames_to_count:
                  try:
                        fps = round(frames_to_count/(time.time()-st))
                        st=time.time()
                        cnt=0
                  except:
                        pass
            cnt+=1
```

Ahora procedemos a ejecutar la práctica, en este caso se está ejecutando el servidor en un sistema operativo Windows y el cliente en un sistema operativo Linux, pertenecientes a la misma red.

Del lado del sistema operativo de Linux se ejecuta el cliente:



```
[redacted]
[redacted]DeprecationWarning: The binary mode of
 fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer inste
ad
  npdata = np.fromstring(data,dtype=np.uint8)
qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "[redacted]i
b/python3.10/site-packages/cv2/qt/plugins"
```