Instituto Politécnico Nacional

IPN

Escuela Superior de Cómputo

ESCOM

Academia de Redes Aplicaciones para comunicaciones en red

6CV2

Práctica 7

“Enviar video a través de sockets UDP”

Alumna:

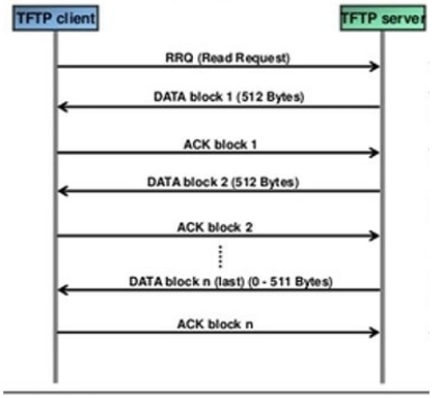
Navarrete Becerril Sharon Anette

Fecha de entrega: “11-Noviembre-2024”

Profesor: Ojeda Santillan Rodrigo

OBJETIVO

Demostrar el funcionamiento del protocolo TFTP, por lo que se va a realizar un cliente y un servidor que permita la transferencia de archivos, se sugiere realizarlo en una distribución de Linux. Se comenzará creando en tu entorno de desarrollo de preferencia dos archivos, uno será para el cliente y uno para el servidor: • Cliente.c • Servidor.c



Para replicar la comunicación como la que se muestra en la imagen anterior donde se realiza una petición de parte de un cliente a un servidor.

INTRODUCCIÓN

El Protocolo de Transferencia de Archivos Trivial (TFTP, por sus siglas en inglés) es un protocolo ligero y simple diseñado para la transferencia de archivos a través de redes, particularmente útil en situaciones donde los recursos son limitados y no se requieren funciones avanzadas de seguridad o control. Basado en el protocolo UDP (User Datagram Protocol), TFTP se destaca por su baja sobrecarga y su capacidad para operar de manera eficiente en redes que no necesitan la robustez que ofrecen otros protocolos de transferencia como FTP o HTTP.

TFTP es comúnmente utilizado en dispositivos de red como routers, switches y sistemas embebidos, donde su diseño minimalista permite implementaciones rápidas y sencillas. A diferencia de FTP, TFTP no ofrece características como autenticación, cifrado o gestión avanzada de conexiones, lo que lo hace menos seguro, pero al mismo tiempo ideal para tareas de mantenimiento, como la actualización de firmware o la configuración remota de dispositivos.

El protocolo opera transfiriendo archivos en bloques de tamaño fijo, donde cada bloque es confirmado por el receptor mediante un acuse de recibo (ACK). Este esquema sencillo asegura que los archivos se transfieran de manera secuencial y controlada, aunque sin las garantías de entrega que se esperarían en un protocolo basado en TCP. Por esta razón, TFTP es especialmente útil en redes confiables o controladas, donde la pérdida de datos es mínima.

DESARROLLO

El funcionamiento del código se basa en la creación de un socket UDP mediante `socket(AF_INET, SOCK_DGRAM, 0)`, seguido de la configuración inicial del socket tanto a nivel local como remoto. Esto incluye la vinculación con `bind` y la configuración de la dirección del servidor remoto utilizando `inet_addr`. Dependiendo de la opción seleccionada por el usuario, el programa puede realizar una solicitud de lectura o escritura de un archivo. En el caso de una lectura, el cliente envía la solicitud al servidor TFTP y, si es exitosa, recibe el archivo en bloques, los cuales se almacenan localmente hasta que se completa la transferencia o se alcanza un límite de 10 MB. Para la escritura, el archivo se divide en bloques de 512 bytes, que se envían al servidor TFTP, esperando un ACK (Acknowledgment) después de cada bloque antes de continuar. Además, se implementa un manejo básico de errores para situaciones como la falta de recepción de ACKs. El código también incluye funciones para crear solicitudes de lectura y escritura, enviar ACKs y gestionar la estructura de los datos transferidos.
Inicialmente se va a establecer la lógica para poder desarrollar el cliente, por lo que se agrega el siguiente fragmento de código en el archivo cliente.c:

Código del cliente:

```
# Cliente para recibir video a través de UDP
import cv2
import imutils
import socket
import numpy as np
import time
import base64

BUFF_SIZE = 65536

# Crear socket UDP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, BUFF_SIZE)

# Configuración de IP y puerto
host_ip = '172.100.75.142'  # IP del servidor
port = 8080
message = b'Conectado'
client_socket.sendto(message, (host_ip, port))

fps, st, frames_to_count, cnt = (0, 0, 20, 0)

while True:
    # Recibir paquete de datos
    packet, _ = client_socket.recvfrom(BUFF_SIZE)

    # Decodificar el paquete base64
    data = base64.b64decode(packet, ' /')

    # Convertir datos a un array de numpy
    npdata = np.frombuffer(data, dtype=np.uint8)

    # Decodificar la imagen
    frame = cv2.imdecode(npdata, 1)

    # Mostrar el FPS en el frame
```

```
    frame = cv2.putText(frame, f'FPS: {fps}', (10, 40),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Mostrar el frame en la ventana del cliente
    cv2.imshow("Recibiendo video", frame)

    # Salir si se presiona 'q'
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        client_socket.close()
        cv2.destroyAllWindows()
        print("Conexión cerrada.")
        break

    # Calcular FPS
    if cnt == frames_to_count:
        try:
            fps = round(frames_to_count / (time.time() - st))
            st = time.time()
            cnt = 0
        except ZeroDivisionError:
            pass

    cnt += 1
```

Para el servidor se implementa el funcionamiento de transferencia de archivos utilizando el protocolo TFTP. Utiliza varias bibliotecas estándar de C para manejar la creación y manipulación de sockets, estructuras de datos de red, y operaciones básicas de entrada y salida.

El servidor crea un socket UDP y lo enlaza a una dirección y puerto local (puerto 69, estándar de TFTP). En el bucle principal, espera solicitudes de lectura o escritura de archivos, identificadas por el segundo byte del mensaje recibido.

Para una solicitud de lectura, el servidor intenta abrir el archivo en modo binario y, si tiene éxito, lo envía en bloques de 512 bytes, esperando un ACK del cliente después de cada bloque. Si no recibe un ACK, reenvía el bloque.

En caso de una solicitud de escritura, el servidor crea un archivo nuevo, envía un ACK inicial al cliente y luego recibe los bloques de datos, escribiéndolos en el archivo. Envía un ACK por cada bloque recibido hasta que la transferencia termina. Por lo tanto se agrega el siguiente fragmento de código en el archivo servidor.c:

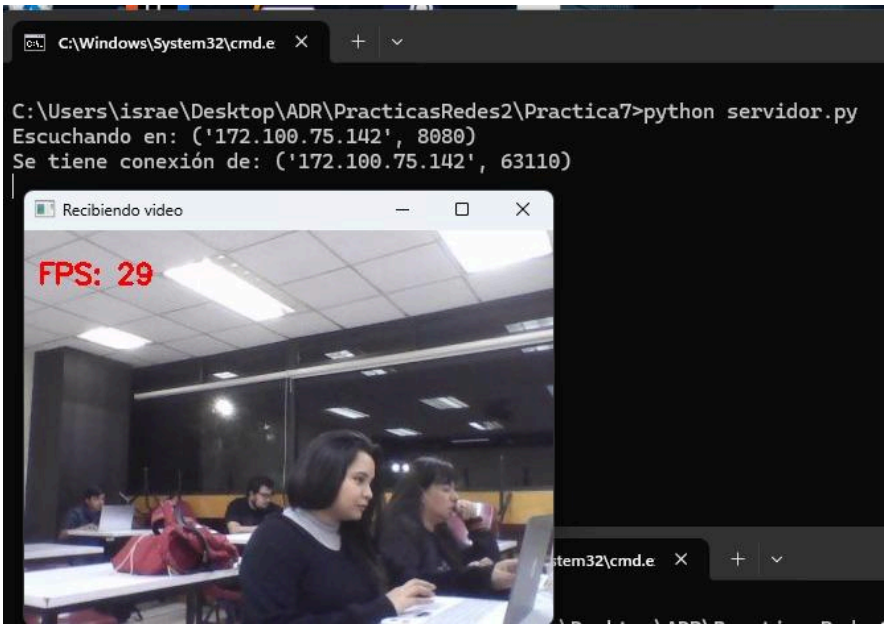
Código del servidor:

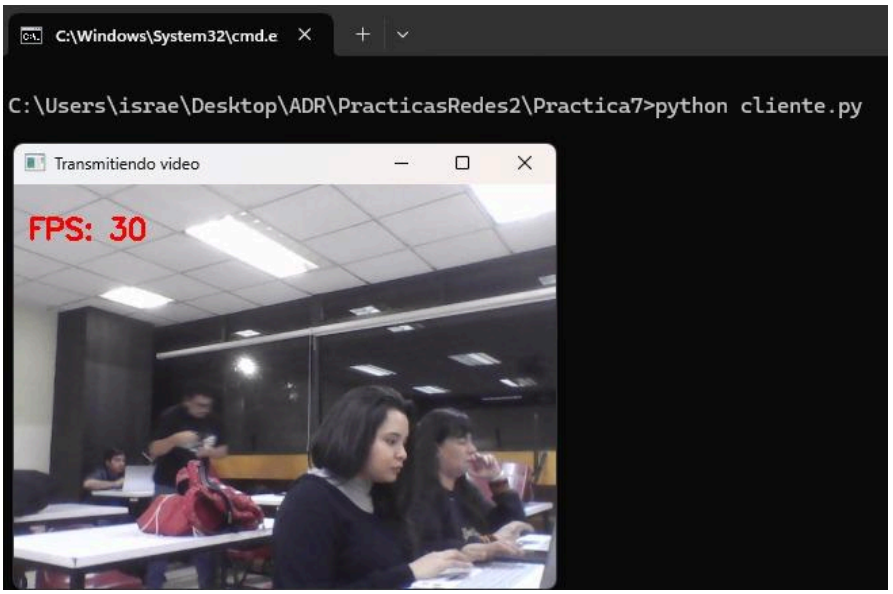
```
 # Servidor para enviar video a través de UDP
import cv2
import imutils
import socket
import numpy as np
import time
import base64
```

```
BUFF_SIZE = 65536

# Crear socket UDP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, BUFF_SIZE)

# Configuración de IP y puerto
host_ip = '172.100.75.142'  # IP del servidor
port = 8080
socket_address = (host_ip, port)
server_socket.bind(socket_address)

print(f'Escuchando en: {socket_address}')

# Capturar video desde la webcam (0 para webcam predeterminada)
vid = cv2.VideoCapture(0)

fps, st, frames_to_count, cnt = (0, 0, 20, 0)

while True:
    # Esperar mensaje de conexión del cliente
    msg, client_addr = server_socket.recvfrom(BUFF_SIZE)
    print(f'Se tiene conexión de: {client_addr}')

    # Parámetros del video
    WIDTH = 400

    while vid.isOpened():
        # Leer un frame de la webcam
        ret, frame = vid.read()

        if not ret:
            print("No se pudo capturar el video. Saliendo...")
            break

        # Redimensionar el frame
        frame = imutils.resize(frame, width=WIDTH)

        # Codificar el frame a formato JPEG
        encoded, buffer = cv2.imencode('.jpg', frame,
[cv2.IMWRITE_JPEG_QUALITY, 80])

        # Codificar en base64
        message = base64.b64encode(buffer)

        # Enviar el frame al cliente
        server_socket.sendto(message, client_addr)

        # Mostrar el FPS en el frame
        frame = cv2.putText(frame, f'FPS: {fps}', (10, 40),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        # Mostrar el frame en la ventana de transmisión
        cv2.imshow('Transmitiendo video', frame)

        # Salir si se presiona 'q'
        key = cv2.waitKey(1) & 0xFF
        if key == ord('q'):
            server_socket.close()
```

```
            vid.release()
            cv2.destroyAllWindows()
            print("Conexión cerrada.")
            break

        # Calcular FPS
        if cnt == frames_to_count:
            try:
                fps = round(frames_to_count / (time.time() - st))
                st = time.time()
                cnt = 0
            except ZeroDivisionError:
                pass

        cnt += 1
```

Capturas del funcionamiento:





CONCLUSIONES

Navarrete Becerril Sharon Anette:

Implementación práctica y mínima en redes controladas: La implementación de TFTP y la transmisión de video mediante sockets UDP es especialmente útil en redes controladas, donde la pérdida de paquetes es mínima. Este enfoque reduce la complejidad del sistema, lo que facilita su integración en dispositivos de red, aunque no es adecuado para entornos que requieran mecanismos de seguridad avanzados o transferencias confiables en redes inestables.

BIBLIOGRAFÍA

Stevens, W. Richard. Programación en red con Unix: La API de Sockets. Addison-Wesley Professional, 2003.

Forouzan, Behrouz A. Comunicaciones de Datos y Redes. McGraw-Hill, 2012. Tanenbaum, Andrew S., y David J. Wetherall. Redes de Computadoras. Pearson, 2010.

Comer, Douglas E. Internetworking con TCP/IP Volumen Uno. Prentice Hall, 2006. Kurose, James F., y Keith W. Ross. Redes de Computadoras: Un Enfoque Descendente. Pearson, 2017.

Beazley, David, y Brian K. Jones. Python Cookbook: Recetas para Dominar Python 3. O'Reilly Media, 2013. Begg, A. Sockets TCP/IP en C: Guía Práctica para Programadores. Morgan Kaufmann, 2000