Redes de Computadoras

Curso 2023

Grupo 27

Informe - Obligatorio 2

Autores:

Joaquin Campo

Mateo Daneri

Santiago Rogriguez

Supervisor:

Leonardo Vidal

15 de Octubre, 2023

Introducción

El objetivo de este informe es el de documentar el trabajo llevado a cabo sobre el obligatorio número dos, el cual se centra en fomentar el acercamiento del grupo a la implementación de sockets a nivel de código, a través de la realización de una plataforma de streaming entre diferentes hosts. Se abordarán conceptos como los detalles de la implementación de la plataforma, así como el proceso de desarrollo de la misma, junto con los obstáculos encontrados, finalizando con una conclusión y mejoras a futuro que no llegamos a abarcar en el proyecto.

Parte central

Servidor

Para el caso del código servidor, para resolver la independización de conexión de cada cliente, utilizamos un map "clientes" con claves "ipCliente" y "puertoUDPCliente", junto con un campo "estado" que toma valores "True" o "False" para gestionar a quién se le debe redirigir el video VLC en cada momento. Este diccionario será recorrido por la función encargada de redireccionar el video por cada datagrama del mismo. El acceso a este map está multiplexado por medio de la librería "threading" con la variable "clientes_mutex" y las funciones "clientes_mutex.acquire()" y "clientes_mutex.release()" para pedir su uso y avisar que se deja de usarlo respectivamente. Más allá, decidimos utilizar un socket UDP "udpServer" para recibir los datagramas de la fuente VLC del servidor y uno "udpSender" para reenviar los datagramas a cada cliente activo. Luego usamos un socket TCP que recibe las nuevas conexiones de parte de los clientes y a raíz de ello crea un socket también TCP para cada uno de ellos, encargado de escuchar los comandos.

Clientes

Hablando ahora de los clientes, utilizamos un socket TCP "sktTCP" el cual se conecta al servidor y a través del cual se enviarán los comandos de conexión, interrupción, etc. Luego, se define además un socket UDP "sktUDP" en el cual se recibirán los datagramas enviados por el socket "udpSender" correspondiente en el servidor. Por último, el socket UDP "sktVLC" se encarga de redirigir el video al reproductor VLC del cliente, ubicado en la ip '127.0.0.1' y puerto "vlcPort" ingresado por parámetro.

Soluciones API

Servidor

```
def CONTROLSTREAM(sktClient, ipClient, tcpClientPort):
    client_id = none

    bool conectado = False

    ipCliente, _ = sktClient.getpeer()


    # Recibir comandos del client y enviarle los datagramas
    while True:
        comando = ''
        repeat
            data, err = sktClient.receive()
            comando += data
        until find(comando, '\n') or err == 'closed' or err == 'timeout'

        if (err == timeout):
            # Comunicamos el error
            repeat
                remain, _ = client.send('No se recibio el
                comando correctamente, intente denuevo\n')
            until remain == ''

        if (err == closed):
            print('El client se desconecto')
            break

        if comando == 'CONECTAR' + puertoUDPCliente:
            if (!conectado):
                conectado = true
                # Asumimos que obtuvimos el puerto UDP del cliente
                client_id = ipCliente, puertoUDPCliente

                # Mandamos OK
                repeat
                    remain, _ = client.send('OK\n')
                until remain == ''

                mutex.acquire()
                    clients.add(ipCliente, puertoUDPCliente)
                    clients[client_id].estado = True
                mutex.release()

        elif comando == 'INTERRUMPIR':
            # Dejar de retransmitir al cliente
            mutex.acquire()
                clients[client_id].estado = False
            mutex.release()

            # Mandamos OK
```

```
            repeat
                remain, _ = client.send('OK\n')
            until remain == ''

        elif comando == 'CONTINUAR':
            # Comnezar a retransmitir al client
            mutex.acquire()
                clients[client_id].estado = True
            mutex.release()

            # Mandamos OK
            repeat
                remain, _ = client.send('OK\n')
            until remain == ''

        elif comando == 'DESCONECTAR':
            # Dejar de retransmitir al cliente y sacarlo
            mutex.acquire()
                clients.remove(client_id)
            mutex.release()

            # Mandamos OK
            repeat
                remain, _ = client.send('OK\n')
            until remain == ''

            sktClient.close()
            break

        else print('Comando invalido')

def enviarData():
    #recibir stream de VLC y redirigir
    while True:
        event.wait()
        # mientras que haya por lo menos un cliente conectado

        datagrama, ip, port = sktVLC.receive()
        mutex.acquire()
            for(client in Clients):
                sktVLC.sendto(datagrama, client.ipClient,
                client.udpClientPort)
        mutex.release()

# Configuraci n del socket TCP para aceptar la conexion de un client
sktMaster = socket.tcp()
sktMaster.bind(ServerIP, ServerPort)
sktServer = sktMaster.listen()

# Socket VLC (UDP)
sktVLC = socket.udp()
sktVLC.bind(127.0.0.1, 65534)

event = threading.Event()

# Crear thread para enviar los datagramas a los clients
thread.new(enviarData, sktVLC)

# Crear hash para clients conectados
```

```
clients = {}

# Semaforo para mutua exclusion de la lista de clients
mutex = Semaphore(1)

while True:
    # Esperamos por la conexion del client
    sktClient, err = sktServer.accept()
    ipclient, puertoTCP = sktClient.getpeer()

    thread.new(CONTROLSTREAM, sktClient, ipClient, tcpClientPort)

sktServer.close()
sktVLC.close()
```

Clientes

```
# Recibimos los datagramas desde el servidor
def recibirDatos:
    while True:
        datagrama, ip_server, port_server = sktData.receive()
        # Procesar datagrama
        sktVLC.sendto(127.0.0.1, vlcport)

# Recepcion de CONECTAR, INTERRUMPIR, DESCONECTAR, CONTINUAR
def CONTROLSTREAM():
    bool conectado = false
    while True:
        entrada = sys.input(command:)

        if (entrada == 'CONECTAR'):
            if(!conectado):
                conectado = true
                remain = 'CONECTAR <' + chosen_port + '>\n'
                repeat
                    remain, err = sktCommand.send(remain)
                until remain = '' or err == 'timeout' or err == 'closed'

                if (err == 'closed'):
                    print('ERROR - Se perdio la conceccion con el servidor')
                    break

                if ( err == 'timeout'):
                    print('ERROR - No se pudo conectar con el servidor')
                    conectado = false
                else:
                    # Recibimos el OK
                    ok = ''
                    repeat
                        data, err = sktClient.receive()
                        ok += data
                    until ok = 'OK\n'
                    print(ok)

        elif (entrada = 'INTERRUMPIR'):
            remain = 'INTERRUMPIR'
            repeat
                remain, err = sktCommand.send(remain)
            until remain = ''

            # Recibimos el OK
            ok = ''
            repeat
                data, err = sktClient.receive()
                ok += data
            until ok = 'OK\n'

            sys.out('OK')

        elif (entrada = 'CONTINUAR'):
            remain = 'CONTINUAR'
            repeat
                remain, err = sktCommand.send(remain)
```

```
            until remain = ''

            # Recibimos el OK
            ok = ''
            repeat
                data, err = sktClient.receive()
                ok += data
            until ok = 'OK\n'

            sys.out('OK')

        elif (entrada = 'DESCONECTAR'):
            remain = 'DESCONECTAR'
            repeat
                remain, err = sktCommand.send(remain)
            until remain = ''

            # Recibimos el OK
            ok = ''
            repeat
                data, err = sktClient.receive()
                ok += data
            until ok = 'OK\n'

            sys.out('OK')

        default:
            sys.out('ERROR - Comando no reconocido')

# Recibimos los parametros
host = sys.argv[1]
port = sys.argv[2]
vlcPort = int(sys.argv[3])

# Creamos el socket TCP para conectarnos con el servidor
sktMaster = socket.tcp()
sktMaster.connect((host, port))

# Creamos un socket UDP para recibir los datagramas del servidor
sktData = socket.udp()
sktData.bind(*, 0)

# Creamos un socket UDP para redirigir los datagramas al reproductor VLC
sktVLC = socket.udp()

# Creamos un hilo para recibir los datagramas
ip, chosen_port = sktData.gethost()
thread.new(recibirDatos, sktData, sktVLC)


# Nos conectamos al servidor
sktCommand, err = sktMaster.connect(ServerIP, ServerPort)
```

Experimentación

En una primera aproximación, tuvimos la idea de ir generando el código final en etapas, implementando primero en la API del curso un servidor que mande el video a un único cliente local, sin contemplar aún el pasaje de comandos entre ellos. Luego, al tener esto resuelto, comenzamos a indagar en el pasaje de mensajes (Conectar, interrumpir, continuar y desconectar) entre ellos. Una vez implementada dicha versión, pasamos ya a la generación de código Python de esta última versión, para luego llegar a la versión final con los n clientes locales. A partir de esta versión decidimos realizar las pruebas en localhost, dando a luz errores que tenía nuestra implementación que, al haber sido solo testeada en un ambiente cerrado, no habíamos encontrado. Otro problema que surgió durante el desarrollo de la plataforma se basa en el impedimento de establecer la conexión por medio de los sockets TCP. Luego de mucha investigación y experimentaciones, pudimos observar que el error se daba por el sistema de funcionamiento que lleva a cabo el firewall de Windows Defender, el cual veía la solicitud de conexión desde otro host como una solicitud peligrosa, por lo que la denegaba activamente.

Conclusiones y trabajo futuro

Algunas mejoras a tener en cuenta pueden ser:

1. Mejorar el rendimiento de videos con resoluciones superiores:

   Nuestra solución resultó ineficiente para resoluciones mayores a 1080p, no siendo capaz de reproducir el video con una fluidez aceptable. A pesar de indagar en tamaños de buffer y métodos de redirección de los datagramas más eficientes, no llegamos a una solución eficiente.

2. Mejorar el uso de los recursos:

   Nuestra implementación permite una situación en la cual, cuando no hay usuarios conectados o ninguna tupla en el diccionario *clientes* con estado en *True*, el servidor permanece recibiendo los datagramas que le envía VLC y recorre todo diccionario simplemente para no reenviar ningún datagrama, lo cual consume recursos innecesariamente. Una manera de solucionar esta problemática sería utilizando métodos de sincronización, con los cuales se bloquearía el thread que recibe los datagramas de vlc hasta que no haya por lo menos un cliente al que hay que reenviárselos.

3. Cambiar la manera en la que se reenvían los datagramas:

   En nuestra solución, para reenviar los datagramas a cada cliente, recorremos todo el diccionario *clientes* y se le envía el datagrama si tiene estado en *True*. Esto tiene problemas de escalabilidad, ya que cuando la cantidad de clientes que quieren recibir el video es elevada, habrá una diferencia importante entre los tiempos de recepción entre un datagrama y el siguiente, ya que para volver a recibir otro datagrama, el servidor tiene antes haberle enviado los datagramas a todos los otros clientes. Una manera de solucionar esto sería enviando simultáneamente los datagramas a cada cliente. Por ejemplo, teniendo un thread por cliente, el cual se encargue de enviarle los datagramas.