Redes de Computadoras

Obligatorio 2 - 2023

Facultad de Ingeniería
Instituto de Computación
Departamento de Arquitectura de Sistemas

Nota previa - IMPORTANTE

Se debe cumplir íntegramente el "Reglamento del Instituto de Computación ante Instancias de No Individualidad en los Laboratorios", disponible en el EVA.

En particular está prohibido utilizar documentación de otros estudiantes, de otros años, de cualquier índole, o hacer público código a través de cualquier medio (EVA, news, correo, papeles sobre la mesa, etc.).

Introducción

Control Intermedio

El control intermedio se realizará en el monitoreo de la semana del 02/10, donde se deberá presentar una solución en alto nivel utilizando la API de sockets del curso (ver sección 4.a), y una versión básica del servidor a definir con el docente de monitoreo. Para probar funcionalidades básicas del servidor se utilzará la utilidad `telnet`. Además, se espera que para este control se maneje adecuadamente la aplicación VLC.

Forma de entrega

Una clara, concisa y descriptiva documentación es clave para comprender el trabajo realizado. La entrega de la tarea consiste en un único archivo obligatorio2GrupoGG.tar.gz que deberá contener los siguientes archivos:

- Un documento llamado Obligatorio2GrupoGG.pdf donde se documente todo lo solicitado en la tarea. GG es el número del grupo. La documentación deberá describir claramente su solución, las decisiones tomadas, los problemas encontrados y posibles mejoras, y las pruebas realizadas.
- El código fuente del programa (**en lenguaje Python**) e instrucciones claras de cómo ejecutar el sistema.

La entrega se realizará en el sitio del curso, en la plataforma EVA.

Fecha de entrega

Los trabajos deberán ser entregados **antes del 15/10/2023 a las 23:30 horas**. No se aceptará ningún trabajo pasada la citada fecha y hora. En particular, no se aceptarán trabajos enviados por e-mail a los docentes del curso.

Objetivo del Trabajo

Aplicar los conceptos teóricos de capas de aplicación y transporte, la utilización de la API de sockets TCP y UDP, y la arquitectura de aplicaciones cliente-servidor.

Descripción general del problema

Se desea implementar un servidor de *streaming* de video y un cliente que consume el video con determinadas acciones de control.

Problema a resolver

Se desea implementar un servidor de *streaming* que realiza las siguientes tareas.

- Consumir un *stream* "fuente" usando el protocolo RTP sobre UDP.
- Reenviar dicho *streaming* de video a múltiples clientes.
- Aceptar conexiones de control sobre TCP de los clientes, que permitan gestionar la reproducción del *streaming*.

El servidor (Stream-server) estará esperando por conexiones TCP al puerto `ServerPort`, a la espera de clientes que quieran recibir el *stream*. En este puerto TCP el servidor implementa el protocolo CONTROLSTREAM (ver sección 3.1) para controlar el *streaming*.

El Stream-server consumirá la señal de video fuente recibida por UDP en una IP y puertos determinado (127.0.0.1:65534), y lo reenviará a todos los clientes que lo estén solicitando en ese momento.

Asociado a este servidor, se implementará el cliente de control TCP, que será usado por los usuarios finales para iniciar, interrumpir, continuar y cerrar su sesión de *streaming*.

Tanto la fuente del *stream* de video como su visualización se realizarán con una herramienta externa, VLC [1]. La instancia de VLC encargada de reproducir el *stream* del lado del cliente se ejecutará en el mismo *host* que el cliente de control correspondiente.

Para la solución del problema se debe instalar el software VLC media Player, herramienta que será utilizada para generar la fuente de transmisión del *stream*, consumida por el servidor y por el cliente para visualizar el *stream* retransmitido.

En la Figura 1 se muestra la generación del *stream* fuente que se entrega al *Stream-server* mediante el proceso identificado como *VLC media player (source-stream)*, ejecutando el siguiente comando desde una terminal del host donde se ejecuta el server:

```
cvlc -vvv videoplayback.mp4 –sout "#transcode{vcodec=mp4v,acodec=mpga}:rtp{proto=udp,
mux=ts,
dst=127.0.0.1, port=65534}" --loop --ttl 1
```

Éste comando genera, a partir del video mp4 pasado por parámetro (videoplayback.mp4), el *stream* transmitido por protocolo RTP (Real-time Transport Protocol) con destino IP 127.0.0.1 y puerto 65534.

El *Stream-client* solicita el video y es retransmitido por el servidor a la dirección IP del cliente y un puerto elegido. Para visualizar dicho video también se utiliza el proceso *VLC media player dest-stream*, ejecutando en una terminal del cliente con el siguiente comando:

vlc rtp://<ip_cliente>:<puerto_elegido>

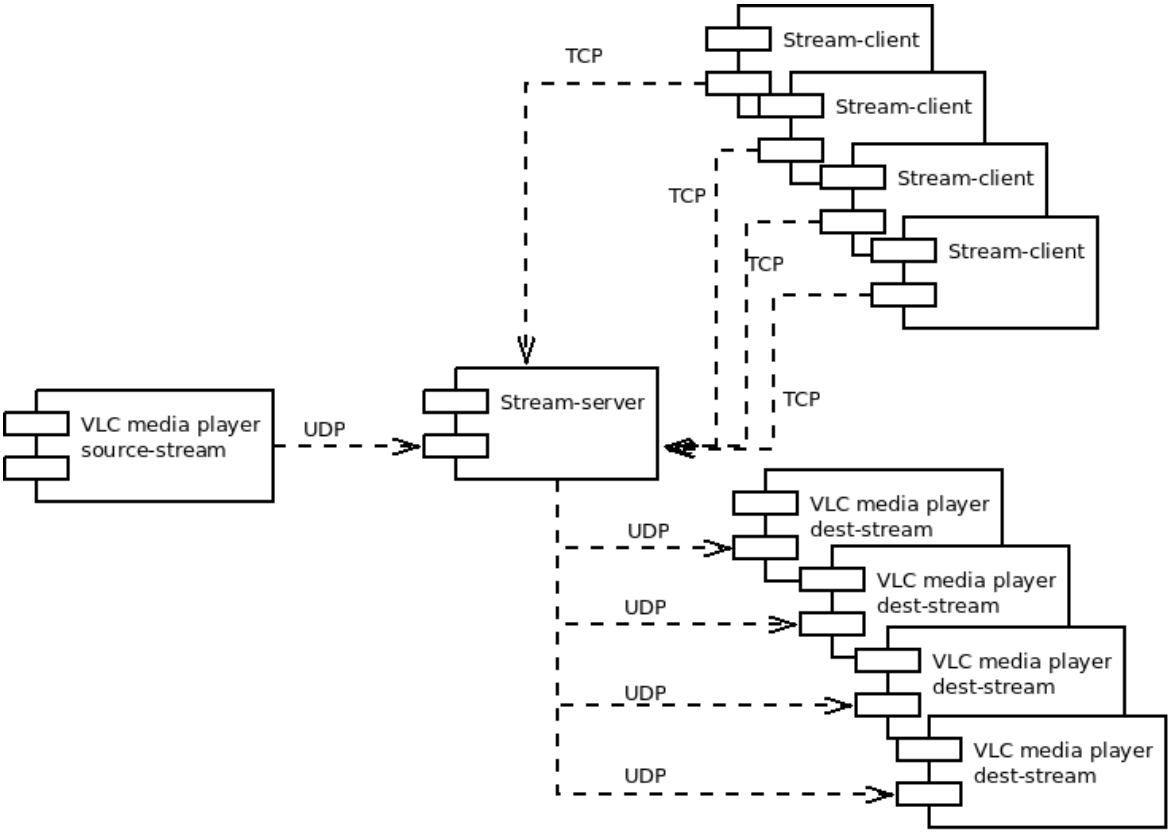

Figura 1: Arquitectura general del sistema de streaming propuesto

1. Servidor

El servidor al iniciar comienza a escuchar el puerto destino del VLC media player (en el ejemplo se utiliza el puerto 65534), que recibe el *stream*. Posteriormente levanta el puerto `ServerPort` en la IP `ServerIP`, donde recibe solicitudes de conexión de los clientes.

Se sugiere la siguiente forma de invocación:

python server.py <ServerIP> <ServerPort>

donde:

- <ServerIP>
  Dirección IP donde el servidor aceptará conexiones.
- <ServerPort>
  Puerto TCP donde el servidor aceptará conexiones.

1.1 Distribución del stream

El funcionamiento básico del servidor retrasmite por UDP cada datagrama del stream fuente recibido a todos los clientes conectados.

2. Cliente

El cliente es capaz de conectarse a un servidor que se le indique, y realizar detrminados operaciones que se indican en el protocolo CONTROLSTREAM (ver sección 3.1.

Se sugiere la siguiente forma de invocación:

python cliente.py <ServerIP> <ServerPort> <PuertoVLC>

donde:

- <ServerIP>
  Dirección IP del servidor al que se desea conectar, por ejemplo 127.0.0.1
- <ServerPort>
  Puerto del servidor al que se desea conectar, por ejemplo 2023
- <PuertoVLC>
  Puerto de la instancia de VLC del cliente usada para reproducir el *stream*.

El cliente debe implementar una consola de texto para leer comandos especificados por el usuario, que permiten controlar la reproducción. Se deben implementar los comandos CONECTAR, INTERRUMPIR, CONTINUAR y DESCONECTAR según se especifica a continuación:

- CONECTAR: solicita al servidor la reproducción del *stream*.
- INTERRUMPIR: solicita al servidor que suspenda momentáneamente la transmisión.
- CONTINUAR: solicita al servidor que retome la transmisión luego de haber ejecutado el comando INTERRUMPIR.
- DESCONECTAR: solicita al servidor la finalización de la transmisión.

Después de ejecutar el comando DESCONECTAR, se debe cerrar del cliente.

3. Protocolo

Se define el protocolo CONTROLSTREAM de comunicación entre el cliente y el servidor TCP.

El protocolo se maneja a través de mensajes de texto plano. Los mensajes generados desde el cliente son:

```
CONECTAR <PUERTO_UDP_CLIENTE>\n
INTERRUMPIR\n
CONTINUAR\n
DESCONECTAR\n
```

Los mensajes terminan con un \n (fin de línea).
La recepción de un mensaje, genera por parte del servidor una respuesta

```
OK\n
```

3.1. CONECTAR

CONECTAR <puerto-udp-cliente>**\n**

Solicita el envío del *stream* a la dirección IP del cliente y el puerto <*puerto-udp-*

cliente>.

3.1.2 INTERRUMPIR

`INTERRUMPIR\n`

Solicita interrumpir el envío del *stream*.

3.1.3 CONTINUAR

`CONTINUAR\n`

Solicita retomar el envío del *stream*. La continuación se realiza a partir del próximo datagrama del *stream* recibido por el server luego de recibir el comando `CONTINUAR.`

3.1.4 DESCONECTAR

`DESCONECTAR\n`

Solicita cancelar el envío del *stream.*

4. Se pide

a) Diseñe y documente el servidor y el cliente utilizando las primitivas de la API de sockets del curso [2].
b) Implemente en lenguaje Python, el servidor solicitado.
c) Implemente en lenguaje Python, el cliente solicitado.

5. Observaciones:

- Se aceptará el uso de bibliotecas de estructuras de datos, *parsing*, hilos, *hashing*, etc.

Referencias y Bibliografía Recomendada

[1] VideoLAN Organization. https://www.videolan.org/vlc/
[2] API de sockets para un lenguaje
de alto nivel https://eva.fing.edu.uy/pluginfile.php/254645/mod_resource/content/0/cartilla_sockets.pdf