Xarxes: pràctica 1

Programació d'aplicació de xarxa

Versió: v1

Data 02/05/2025

Gerard Safont Catena

48050853N

gsc23@alumnes.udl.cat

Índex

Introducció

Aquest document descriu el desenvolupament i implementació d'una aplicació de xarxa per a l'streaming de vídeo, elaborada en el marc de la pràctica 1 de l'assignatura de Xarxes. L'objectiu principal del treball és programar un client i un servidor que permetin la transmissió i recepció de contingut de vídeo mitjançant els protocols RTSP i RTP, gestionant correctament els estats de connexió i control de flux.

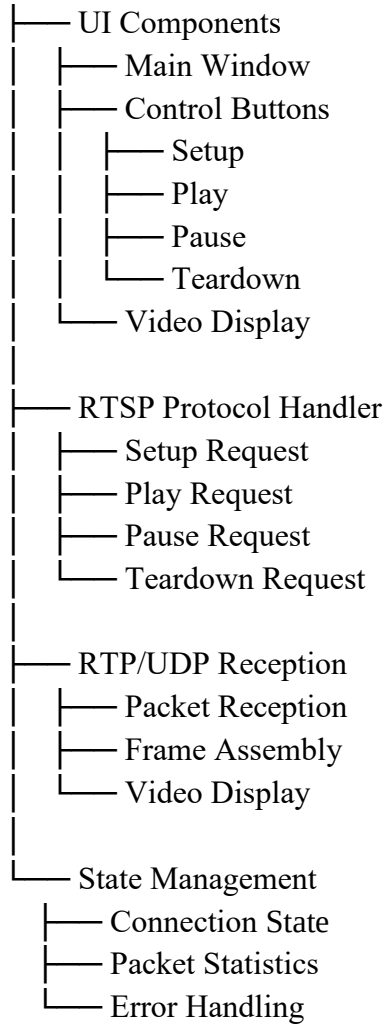
L'informe presenta, en primer lloc, l'estructura i el funcionament del client; tot seguit, es detalla l'arquitectura del servidor i les funcionalitats associades. Finalment, es fa una valoració dels resultats obtinguts i de les dificultats trobades durant el desenvolupament. L'apèndix conté informació complementària i es fa referència als documents adjunts que complementen la pràctica (codi font, vídeos de prova i manual d'ús).

Estructura del client

L'estructura de la classe **Client** és senzilla i es basa principalment en una sola classe que integra diverses funcions. Aquesta classe gestiona la creació de la interfície gràfica, així com l'actualització i reproducció del vídeo. A més, inclou funcions per a la gestió de les operacions bàsiques, com **SETUP**, **PLAY**, **PAUSE** i **TEARDOWN**.

La classe **Client** també importa la classe **StateMachine**, que implementa una màquina d'estats per controlar els estats en què es pot trobar el client. Gràcies a aquesta màquina d'estats, es gestionen les operacions que el client pot executar en funció de l'estat actual en què es troba.

Client (client.py)



Estructura dels servidor

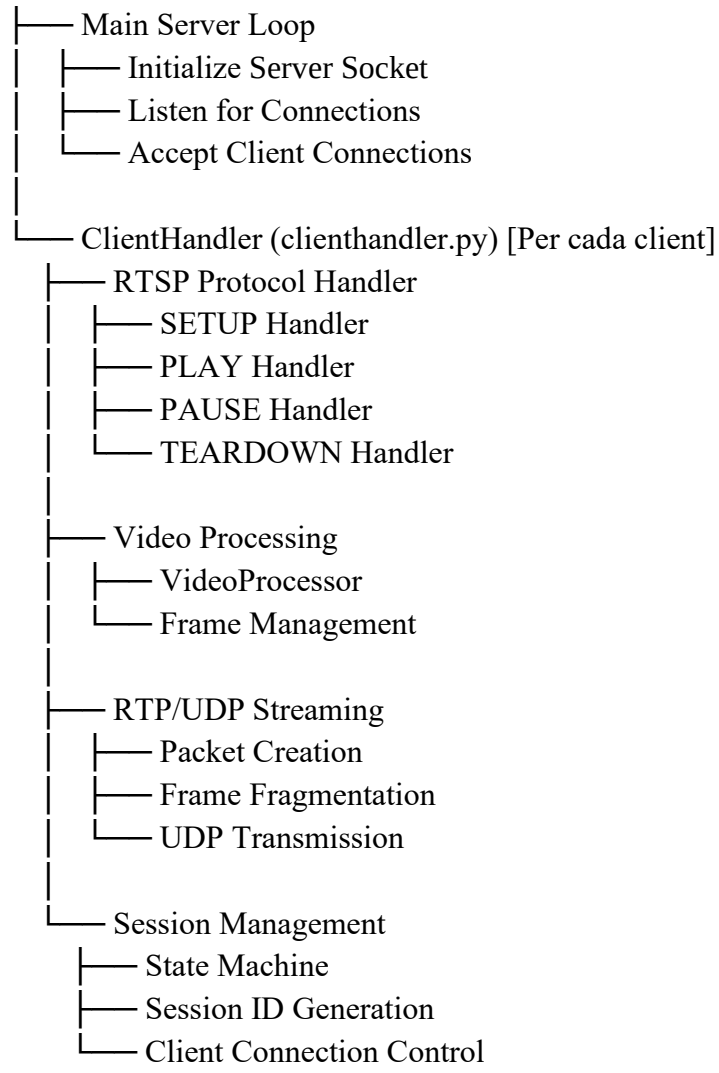
La classe **Server**, en comparació, presenta una estructura més complexa, ja que no es troba tot en un únic fitxer. Està composta per dos fitxers: el primer s'encarrega de gestionar les connexions i desconnexions dels clients, mentre que el segon fitxer defineix la classe **ClientHandler**, que és responsable d'executar les operacions del servidor per a cada client de manera independent. Aquesta separació s'ha implementat per garantir que cada client tingui les seves pròpies variables d'instància (per exemple, self.nom_variable), evitant així conflictes entre les variables de clients diferents que es puguin executar simultàniament.

A més, la classe **ClientHandler** hereta de la classe **threading.Thread**, permetent així que cada client s'executi en un fil (thread) independent. Això permet una gestió més eficient i paral·lela de múltiples clients.

Les funcions principals de la classe **ClientHandler** inclouen l'execució de les operacions **SETUP**, **PLAY**, **PAUSE** i **TEARDOWN**, que depenen del missatge RTSP rebut del client. A més, la classe gestiona el processament del vídeo, que posteriorment es transmet utilitzant el protocol **RTP**. Els paquets d'informació amb els frames de vídeo sol·licitats pel client es transmeten mitjançant un socket UDP, en línia amb l'ús del protocol RTP.

Aquesta classe també importa la classe **StateMachine** per garantir que les operacions es realitzin només quan el servidor es trobi en el mateix estat que el client, assegurant la coherència entre ambdós.

Server (server.py)



Consideracions de desenvolupament

En aquesta pràctica s'ha desenvolupat un client i un servidor per a la transmissió de vídeo en temps real. S'han utilitzat diverses classes per organitzar el codi, com ara la classe **CLI**, que permet seleccionar si es vol executar el client o el servidor, passant-hi paràmetres com el port o l'amfitrió (host) a utilitzar. A més, es poden passar opcions de depuració per mostrar els missatges del codi mitjançant logger.debug. La classe **Client** conté tot el codi relacionat amb el client, mentre que el codi del servidor està dividit entre la classe **Server** i la classe **ClientHandler**. També s'inclou la classe **VideoProcessor**, que s'encarrega de transformar el vídeo en frames per a poder enviar-los mitjançant datagrames UDP, els quals es creen gràcies a la classe **UDPDgram**. Finalment, la classe **StateMachine** implementa una màquina d'estats per verificar si una funció pot executar-se segons l'estat actual del sistema.

Durant la implementació del servidor, va ser necessari fragmentar els frames de vídeo en alguns casos, ja que alguns eren massa grans. Per exemple, en el vídeo de prova **artificial.webm**, es va produir un error de fragmentació, mentre que amb **rick.webm** no es va observar cap error de fragmentació o pèrdua de paquets. Aquesta va ser una de les dificultats més grans que vam trobar en aquesta pràctica, juntament amb la capacitat de gestionar múltiples clients rebent vídeos simultàniament, ja que la resta de funcionalitats estan bastant ben definides en l'enunciat de la pràctica.

El codi està pensat per ser executat mitjançant **Poetry**, tal com s'indica al fitxer **readme.md** que s'adjunta a la pràctica. Tot el codi està comentat per facilitar-ne el seguiment, i s'ha aplicat un enfocament de disseny descendent per aconseguir un codi més net i llegible. Hem intentat evitar la repetició de codi, implementant funcions auxiliars per a mantenir la claredat del codi.

Com es menciona anteriorment, s'adjunta un fitxer **readme.md** que explica les bases de la pràctica, així com les instruccions per executar-la mitjançant **Poetry**, amb exemples i els requisits necessaris. També s'hi descriuen totes les opcions que es poden passar tant al client com al servidor durant la seva execució.