UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

OpenParty Player

propusă de

Iulian Luca

Sesiunea: *iulie, 2021*

Coordonator științific

Conf. Dr. Anca Vitcu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

OpenParty Player

Iulian Luca

Sesiunea: *iulie, 2021*

Coordonator științific

Conf. Dr. Anca Vitcu

Cuprins

Motivație

La finalul secolului XIX, frații Lumière au introdus pentru prima dată în istorie societatea franceză la un nou mediu de cunoaștere, cinematografia. Fie că imaginile în mișcare deservesc un scop educațional sau de divertisment, acestea au devenit o importantă parte a societății. Încă de la o vârstă fragedă, individul este introdus la o serie de imagini care sunt redate atât de rapid încât ochiul uman percepe senzația de mișcare. Dea lungul secolului XX, cinematografia, care a început de la bazele unui sistem artistic, a căpătat un caracter informațional odată cu apariția televiziunii. Abilitatea ca orice persoană să aibă acces la o serie de imagini în mișcare transmise dintr-o locație mai multor consumatori a creat posibilitatea ca mediul video să fie valabil nu doar în cinematografe dar și în casele oamenilor. Astfel, filmele sau video-urile devin principalele surse de informație sau divertisment.

Odată cu apariția internetului, redarea imaginilor în mișcare nu a mai fost limitată doar la nivel de televiziune sau cinematografie. Internetul a adus odată cu el posibilitatea de a reda imagini în mișcare în mediul digital. Această posibilitate are definiția de streaming. Astfel, în loc să fie necesară stocarea completa a unui fișier, consumatorii pot primi în timp real informația, singura limitare fiind viteza conexiunii. Apar noțiuni cum ar fi comprimarea datelor pentru a folosi cât mai puțin trafic, verificarea și controlul calității transmiterii, live streaming unde conținutul este înregistrat și trimis în timp real.

La începutul anilor ’90 apar primele servicii de Video on demand (VOD). Utilizatorii au astfel acces în timp real la ce conținut doresc, atâta timp cât acesta se află pe serverele furnizorului de servicii. În 2005 apare primul site popular de streaming din lume: YouTube. La scurt timp apar alte servicii de VOD streaming dedicate filmelor și serialelor: Netflix, hulu și altele.

Combinând avantajele VOD de a selecta conținutul dorit oricând cu nevoia în contextul social de a experimenta conținutul împreună cu cel puțin încă o persoană, apar metode prin care grupurile pot realiza acest lucru pe Internet, într-un mod sincronizat. Teleparty (cunoscut și ca Netflix Party) și Disney Party Plus sunt câteva exemple de extensii care creează sesiuni cu un grup de participanți ce pot consuma conținut de tip VOD într-un mod sincron. Orice utilizator poate viziona, opri, reporni sau repoziționa conținutul, aceste acțiuni propagându-se la restul participanților din sesiune. Termenul pentru aceste acțiuni poate fi denumit *universal remote*. Librăria este însă limitată de capacitatea serverelor furnizorului acestor servicii. Watch2Gether este un website care permite sincronizarea clipurilor de pe YouTube, Twitch, vimeo într-o sesiune comună, dar este de asemenea limitat de conținutul stocat pe servere. Syncplay, o extensie pentru playere care permite sincronizarea redării între fișierele locale ale participanților, necesită ca fiecare participant sa dețină conținutul stocat pe disk. Opțiunile de share screen ale aplicațiilor ca Zoom, Skype, Discord pun la dispoziție comenzile de playback numai persoanei care inițiază ecranarea.

Prin tema propusă de mine, doresc să ofer o alternativă de a consuma conținutul multimedia cu un grup de participanți in timp real, conținutul fiind deținut de un singur utilizator pe mașina sa care funcționează ca un server. Acesta va transmite, prin anumite căi de comunicație, informația către restul participanților. Oricine va avea dreptul de a executa comenzile de redare specifice unui player obișnuit.

Introducere

Internetul a fost gândit de la început pentru a ajuta la evoluția în comunicarea informației. La început se folosea această cale pentru a transmite informație sub formă de fișiere text și e-mail. Până în acel moment, comunicarea prin mijloace audio sau video se realiza folosind exclusiv legătura telefonică sau TV-ul prin cablu.

În prezent, am ajuns să consumăm prin internet informație ce nu este doar sub formă de text, ci poate include sunet, imagini în mișcare sau ambele. Dispunem de resurse ce ne permit procesarea datelor rapid la nivel local. Prin urmare, modul de transmitere trebuie să evolueze pentru a ține pasul cu capabilitățile de procesare pe care le avem la dispoziție.

Odată cu evoluția internetului în materie de resurse și creșterea masivă a numărului de utilizatori, s-a ajuns la o nevoie mai mare de comunicare multimedia. Transmiterea audio și video prin Internet a devenit tot mai comună în zilele de astăzi. Telefonia și programele TV se pot transmite acum prin internet iar conferințele video sau audio la distanță lungă sunt ușor de realizat.

Este nevoie de transmiterea eficientă a informației cu scopul de a fi recepționate rapid și cu pierderi minime. Ne-am propus astfel să facem o prezentare de ansamblu asupra unor practici generale de transmitere a informației pe bucăți împreună cu o aplicație ce se folosește de aceste practici pentru a transmite informație multimedia către consumatori. Analizând cerințele unui utilizator obișnuit, aplicația a fost realizată pentru a acoperi majoritatea necesităților.

În capitolul 1 vor fi menționate metode de prelucrare a dimensiunii și reprezentării imaginilor video, împreună cu protocoale ce asigură o transmitere calitativă a datelor. În capitolul 2 se explică ce librării au ajutat la punerea în aplicare a logicii gândite pentru funcționalitățile aplicației și ce rol a avut fiecare librărie. Capitolul 3 va descrie cum a fost împărțită logica în module ce îndeplinesc fiecare câte un rol stabilit, atât pentru un server, cât și pentru clienți. În ultimul capitol facem o trecere în revistă a aplicației finale, problemele întâlnite în timpul dezvoltării, cât și eventuale îmbunătățiri.

1 Tehnici de video streaming

Există două modalități de transmitere a informației: descărcarea completă a unui fișier sau trimiterea pe bucăți a informației pentru a fi redat în timp real. Poate fi considerată o a 3-a abordare numita pseudo-streaming. Aceasta se bazează pe redarea informației la câteva secunde după ce se începe descărcarea iar, dacă viteza internetului ne permite, utilizatorii pot viziona conținutul în timp ce se face descărcarea completă a fișierului în spate. Streaming-ul este echivalent cu vizionarea conținutului pe televizor. Conținutul este consumat în timp real și este pierdut la final. Pseudo-streaming poate fi înțeles ca vizionarea unui conținut pe televizor și înregistrarea lui pentru a putea fi redat în viitor.

Trei factori au influențat dezvoltarea streaming-ului: algoritmii de compresie pentru audio și video, serverele pentru streaming și îmbunătățiri aduse în rețelele de comunicare.

1.1 Compresie

Video-ul digital reprezintă o serie de imagini numite cadre. Fiecare cadru este compus dintr-o matrice 2D de pixeli, fiecare pixel reprezentând valori R, G și B. Astfel, matricea 2D de pixeli care constituie o imagine este de fapt alcătuita din trei matrice bidimensionale, una pentru fiecare dintre componentele RGB. O rezoluție de 8 biți per componentă, adică 256 de valori, este de obicei suficientă pentru aplicațiile tipice pentru consumatori. [1]

Compresia se referă la procedeul prin care mărimea fișierului video se reduce considerabil prin modalități de codare și decodare înainte și respectiv după trimiterea lor. Există două metode de bază de comprimare: fără pierderi și cu pierderi. în timp ce compresia fără pierdere poate reproduce identic imaginile după decompresie, compresia cu pierderi redă imagini aproximate, dar nu echivalente. Compresia este aplicată folosind unul sau mai mulți algoritmi matematici pentru a elimina din date. Când videoclipul este vizionat, se aplica alți algoritmi de interpretare a datelor pentru a reconstrui imaginile și a le reda. Timpul în care se realizează acești algoritmi se numește latența de compresie. Latența de compresia devine mai mare cu cât algoritmul folosit este mai avansat. Algoritmii mai complecși folosesc metode de eliminare a informației inutile în cadrul unei serii de imagini. Atunci când se aplică compresia video și se compară cadrele adiacente în algoritm, se introduce o latență mai mare. În unele aplicări, cum ar fi filmele de studio, această latență este irelevanta, din moment ce video-ul nu este vizionat în direct. În schimb, în alte aplicații care necesită transmiterea informației cat mai rapid posibil, latența redusă este esențială.

Compresia video se poate realiza prin exploatarea a patru tipuri diferite de redundanțe:

- Redundanța perceptuală
- Redundanța temporală
- Redundanța spațială
- Redundanța statistică

Redundața perceptuală tratează strict detaliile imaginii care nu pot fi percepute de ochiul uman. Orice detaliu de acest gen poate fi eliminat fără sa afecteze calitatea imaginii. Sistemul vizual uman afectează modul în care sunt percepute atât detaliile spațiale cât și cele temporale într-o secvență video. [1]

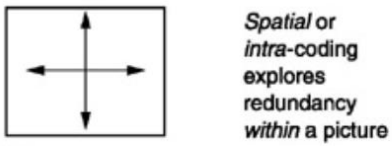


Figura 1-1 - Redundanța într-un cadru [2]

Redundața temporală. Folosind o frecvență mai redusă pentru afișarea cadrelor, putem exploata proprietatea de persistență. Astfel, reușim să reproducem percepția de mișcare continuă folosind mai puțină informație. Din moment ce un video este în esența o secvență de imagini redată la o rată discretă a cadrelor, doua cadre succesive vor arăta foarte asemănător. Extinderea similitudinii între două cadre succesive depinde de cât de apropiate sunt intervalele dintre cadre și de mișcarea obiectelor din scenă. Putem da ca exemplu un video cu un prezentator de știri. Presupunând că frecvența cadrelor ar fi de 30 de afișări pe secunda iar subiectul nu se mișcă prea des, prin urmare, este foarte probabil ca două cadre succesive să arate asemănător. Exploatarea redundanței temporale reprezintă majoritatea câștigurilor de compresie în codarea video. [1]

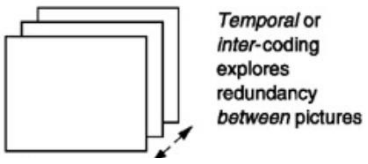


Figura 1-2 - Redundanța dintr-o secvență de cadre [2]

Redundața spațială. Frecvențele spațiale se referă la modificările nivelurilor dintr-o imagine. Sensibilitatea ochiului scade pe măsură ce frecvențele spațiale cresc. Pe măsură ce frecvențele spațiale cresc, abilitatea ochiului de a face diferența între nivelurile de schimbare scade. Orice detaliu care nu poate fi rezolvat este mediat. Această proprietate a ochiului se numește integrare spațială. Proprietatea ochiului poate fi exploatată pentru a elimina sau a reduce frecvențele mai mari fără a afecta calitatea percepută. Percepția vizuală umană permite astfel exploatarea redundanțelor spațiale, temporale și perceptuale. [1]

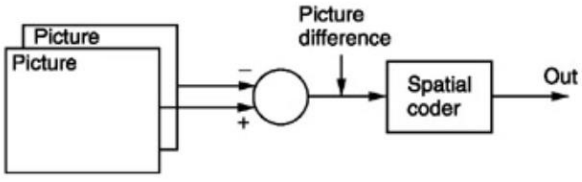


Figura 1-3 - Diferențierea în timp dintre două cadre [2]

Redundața statistică. Coeficienții de transformare, vectorii de mișcare și alte date trebuie codate folosind codurile binare în ultima etapă a compresiei video. Cea mai simplă modalitate de a codifica aceste valori este prin utilizarea unor coduri de lungime fixă; de exemplu, cuvinte de 16 biți. Totuși, aceste valori nu au o distribuție uniformă și utilizarea codurilor de lungime fixă este risipă. Lungimea medie a codului poate fi redusă prin alocarea cuvintelor de cod mai scurte la valori cu probabilitate mai mare. Codificarea lungimii variabile este utilizată pentru a exploata redundanța statistică și pentru a crește în continuare eficiența compresiei. [1]

Există două ramuri mari în standardizarea compresiei: JPEG și MPEG. în timp ce standardele de baza JPEG se folosesc de comprimarea fiecărui cadru, majoritatea standardelor MPEG se bazează pe principii care reduc redundanțele din cadrele secvențiale.

1.1.1 M-JPEG

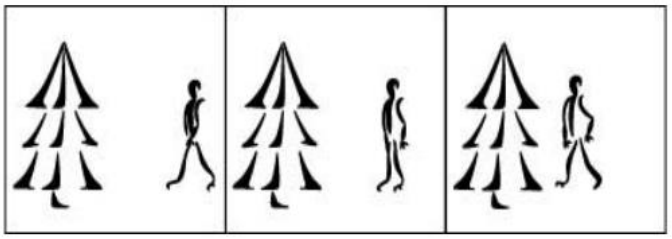
M-JPEG (Motion JPEG) - reprezintă o serie de imagini JPEG. Asigură o compresie de până la 15:1 deoarece fiecare cadru este compresat independent. Din această cauză, nu este necesară o putere de procesare foarte mare și nu ocupă multă memorie. Principalul dezavantaj este că raportul de compresie este foarte mic, din moment ce nu se folosește de tehnici reale de compresie video.

M-JPEG 2000 – are un raport de compresie ușor mai bun comparativ cu M-JPEG, însă este mai complex decât predecesorul sau. Conține același dezavantaj cu M-JPEG din moment ce se folosește tot de compresie pe cadre individuale.

1.1.2 MPEG

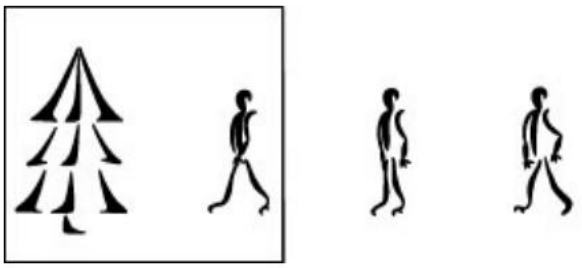
MPEG (Moving Picture Experts Group) – a fost conceput în 1988 pentru a dezvolta standarde de compresie audio și video. Există câteva standarde importante MPEG, având asemănări și diferențe notabile între ele. Un lucru pe care toate îl au în comun este faptul că acestea sunt standarde internaționale stabilite de ISO (Organizația Internațională de Standardizare) și IEC (Comisia Electrotehnica Internațională) cu contribuitori din SUA, Europa și Japonia.

- MPEG-1, dezvoltat în 1993 sub standardul ISO/IEC 11172, tratează codarea imaginilor în mișcare și audio asociate pentru medii de stocare digitala cu un bit-rate de 1.5 Mbps. Folosește aceeași tehnică de comprimare ca JPEG, adăugând câteva tehnici de codare eficientă bazata pe secvențele video.



În compresia M-JPEG fiecare cadru este comprimat și trimis individual (Figura 1-4)

Figura 1-4 - Secvența de trei cadre JPEG [1]



În compresia MPEG sunt incluse doar informații despre părțile diferite dintre cadre (Figura 1-5)

Figura 1-5 - Secvența de trei cadre MPEG [1]

În timpul transmiterii, MPEG limitează consumul lățimii de banda, dar redarea secvențelor din cele 2 figuri va fi aceeași.
MPEG-1 a fost conceput pentru comprimarea video-urilor de calitate VHS fără pierderi de calitate excesive. Formatul asigura o compresie de 52:1

- MPEG-2, dezvoltat în 1995 sub standardul ISO/IEC 13818, acesta o fost utilizat ca standard de bază în televiziunea digitala și DVD, fiind capabil sa realizeze compresia semnalului video de la 280 Mbps la 2-6 Mbps (rată de compresie aproximativa de 200:1). Semnalul audio digital este compresat de la 1.5 Mbps la 100-400 Kbps. Este compatibil cu MPEG-1, însemnând că poate decodifica secvențe de acest tip. Se poate obține o calitate mai înalta a imaginilor folosind mai multa lățime de banda.
  Un standard MPEG-3 a fost propus, destinat televiziunii de definiție înaltă (HDTV). Acesta a fost combinat cu MPEG-2 când s-a realizat că standardul MPEG-2 îndeplinea deja cerințele HDTV folosind modificări minime.

- MPEG-4. dezvoltat sub standardul ISO/IEC 14496, caută să ajute la suportul aplicațiilor care consumă lățime de bandă puțină, cum ar fi dispozitivele mobile, și în același timp, să ofere suport pentru aplicații care necesită calitate mare și dispun de lățime de bandă aproape nelimitată, ca de exemplu filmele de studio. Acesta asigura o rata de compresie mai mare de 200:1. În general, standardul MPEG-4 este mult mai amplu decât standardele anterioare. De asemenea, permite orice frecvență între cadre, în timp ce MPEG-2 a fost limitat la 25 de cadre pe secundă în PAL și 30 de cadre pe secundă în NTSC. Atunci când "MPEG-4" este menționat în aplicațiile de supraveghere, este de obicei menționată MPEG-4 partea 2. Acesta este standardul "clasic" de streaming video MPEG-4, cum ar fi MPEG-4 Visual. [3]

Majoritatea diferențelor dintre MPEG-2 și MPEG-4 sunt caracteristici care nu au legătură cu codarea video. MPEG implică numai codarea completă a cadrelor cheie prin algoritmul JPEG și estimarea modificărilor de mișcare între aceste cadre cheie. Deoarece sunt transmise informații minime la fiecare patru sau cinci cadre, s-a realizat o reducere semnificativă a biților necesari pentru a descrie rezultatele imaginii. În consecință, ratele de comprimare de peste 100: 1 sunt comune. Metoda aceasta de codare MPEG este foarte complexă și plasează o sarcină computațională foarte mare pentru estimarea mișcării. Decodarea este însă mult mai simplă [1]

- MPEG-7 este despre descrierea obiectelor multimedia și nu are nimic de-a face cu compresia. Acesta oferă o bibliotecă de instrumente de descriere de bază și un limbaj de definire a descrierii (DDL) bazat pe XML pentru extinderea bibliotecii cu obiecte multimedia suplimentare. Culoarea, textura, forma și mișcarea sunt exemple de caracteristici definite de MPEG-7.

H.264 este un standard de compresie video folosit pentru înregistrarea, comprimarea și distribuirea conținutului video online. Denumit ca și codecul cel mai utilizat pe scară largă din lume și, de asemenea, cunoscut sub numele de Advanced Video Coding (AVC) sau MPEG-4 Partea 10, H264 a fost dezvoltat în comun de ITU (Uniunea Internațională a Telecomunicațiilor). Scopul lui H.264/AVC a fost sa aducă un nou standard video digital, capabil să ofere o calitate video bună la un bitrate substanțial mai mic decât standardele anterioare, fără a complica prea mult designul, astfel încât punerea în aplicare a acestuia sa rămână practică și relativ ușor de implementat.

Obiective principale ale standardului: [1]

1) Implementări care oferă o reducere a bitrate-ului de 50%, rezultând o calitate video fixă în comparație cu orice alt standard video.

2) Toleranța la erori, astfel încât erorile de transmisie în diverse rețele de comunicare să nu fie deranjante.

3) Capabilități de latență scăzută și o calitate mai bună cu costul unei latențe mari.

4) Specificație simplă de sintaxă care simplifică implementările.

5) Decodarea exactă a potrivirii, care definește exact modul în care calculele numerice trebuie făcute de un codificator și un decodor, pentru a evita acumularea erorilor

Următoarea generație de compresie este cunoscută sub numele de H.265 sau HEVC, oferind un alt salt în eficiență.

1.2 Streaming media prin internet

Datorită existenței sale omniprezente, Internetul a devenit platforma majorității activităților de rețea, inclusiv a aplicațiilor media streaming, unde utilizatorii necesită integrarea serviciilor multimedia. Cu toate acestea, ca rețea de datagrame partajate, Internetul nu era potrivit în mod natural pentru traficul în timp real.

În general, următoarele probleme trebuie rezolvate pentru a reda un flux date multimedia prin Internet:

- Hardware-ul de bază trebuie să ofere o lățime de bandă suficientă pentru a gestiona dimensiunea mare a datelor multimedia [4]

- Aplicațiile care plănuiesc servirea consumatorilor pe o scală largă trebuie să ia în considerare difuzarea multicast pentru a reduce traficul folosit [4]

- Ar trebui să existe unele mecanisme pentru aplicațiile în timp real care rezervă resurse de-a lungul căii de transmisie pentru a putea garanta calitatea serviciilor (QoS) [4]

- Aplicațiile trebuie să se ocupe de problemele de sincronizare, astfel încât datele audio și video sa poată fi redate continuu cu sincronizarea corectă. Acestea trebuie, de asemenea, sa definească operațiuni standard pentru a gestiona livrarea și a afișa datele multimedia. [4]

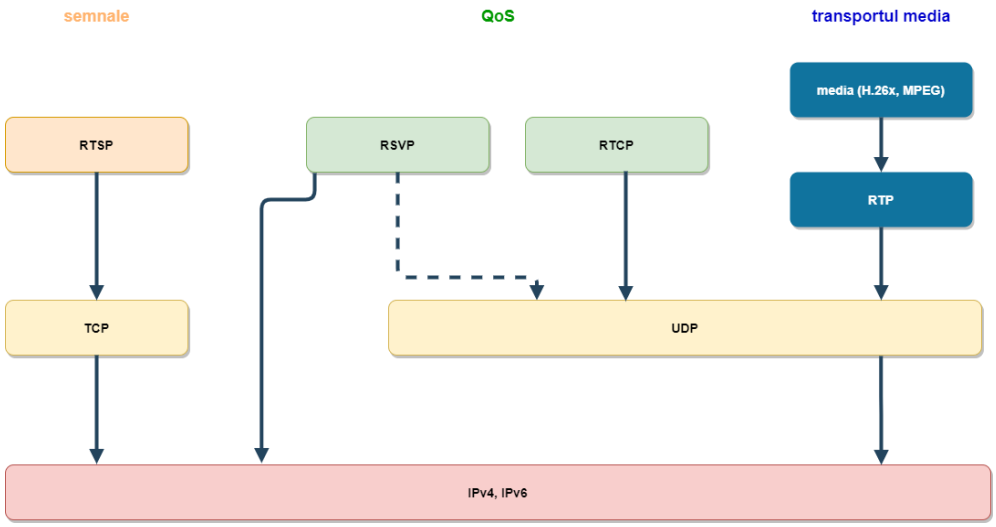


Figura 1-6 - Relația dintre protocoale

1.2.1 RTP

Transportul media în multe aplicații de streaming este implementat în principal cu RTP (Protocolul de transport în timp real, RFC 1889). Protocolul de transport este bazat pe IP pentru realizarea conferințelor audio/video și alte aplicații multiparticipante în timp real. Este un protocol ușor fără funcționalitate de corectare a erorilor sau de control al fluxului. Astfel, nu garantează nici calitatea serviciilor, nici rezervarea resurselor de-a lungul căii de rețea. RTP este, de asemenea, conceput pentru a lucra împreună cu protocolul de control auxiliar, RTCP, ce obține feedback cu privire la calitatea transmiterii datelor și informații despre participanții la sesiunea în curs. RTP este conceput în principal pentru difuzarea multiplă a datelor în timp real, dar poate fi utilizat și în unicast. Acesta poate fi utilizat pentru transportul într-un singur sens, cum ar fi video-on-demand, precum și servicii interactive, cum ar fi telefonia prin Internet. Aplicațiile multimedia necesită o sincronizare adecvată în transmiterea și redarea datelor. Pentru a accepta transmiterea în timp real a datelor, RTP oferă timestamping, numerotarea secvențelor și alte mecanisme. Prin intermediul acestor mecanisme, RTP oferă transport end-to-end pentru date în timp real prin rețeaua de datagrame. [4]

- Timestamping-ul este cea mai importantă informație pentru aplicațiile în timp real. Expeditorul setează marcajul temporal când pachetul a fost trimis. Receptorul utilizează marcajul temporal pentru a reconstrui sincronizarea originală. De asemenea, poate fi utilizat pentru a sincroniza diferite fluxuri cu proprietăți de sincronizare, cum ar fi date audio și video în MPEG. Cu toate acestea, RTP în sine nu este responsabil pentru sincronizare. Acest lucru trebuie făcut la nivel de aplicație.

- Numerele de secvență sunt utilizate pentru a determina ordinea corectă, deoarece protocolul UDP nu livrează pachetele în ordine neapărat. De asemenea, este utilizat pentru detectarea pierderilor de pachete. În unele formate video, atunci când un cadru video este împărțit în mai multe pachete RTP, toate pot avea același marcaj temporal. Prin urmare, timestamping-ul nu este suficient pentru a pune pachetele în ordine.

- Identificatorul payload specifică formatul, precum și schemele de codificare/compresie. Folosind acest identificator, aplicația de primire știe să interpreteze și să redea datele. Exemple de specificații includ JPEG, MPEG1/MPEG2 audio și video, și multe altele. Mai multe tipuri pot fi adăugate prin furnizarea unei specificații de profil și a formatului.

- Identificarea sursei permite aplicației de primire să știe de unde provin datele. De exemplu, într-o conferință audio, din identificatorul sursă, un utilizator poate să își dea seama cine vorbește

RTP este în general folosit împreună cu UDP. Este proiectat în principal pentru difuzare de tip multicast. Din moment ce protocolul TCP este orientat pentru conexiune, acesta nu se scalează bine. Pentru datele în timp real, fiabilitatea nu este la fel de importantă ca și livrarea în timp util. Chiar și dacă transmisia furnizată de retransmiterea datelor în TCP garantează faptul că toate pachetele ajung la destinație, acest lucru nu este ceva foarte necesar în cazul de față. Pachetele RTP și RTCP sunt de obicei transmise utilizând serviciul UDP/IP.

1.2.2 RTCP

RTCP (RTP Control Protocol) este protocolul de control proiectat să funcționeze împreună cu RTP. Într-o sesiune RTP, participanții trimit periodic pachete RTCP pentru a transmite feedback cu privire la calitatea livrării datelor și informațiile fiecărui membru. Acesta este reprezentat de următoarele tipuri de mesaje: [4]

- RR (raport receptor): Receptorul trimite un feedback de calitate a recepției datelor, cum ar fi cel mai mare număr de pachete primite, numărul de pachete pierdute, decalajul între sosire și timestamp-uri pentru a calcula întârzierea dus-întors dintre expeditor și destinatar

- SR (raport expeditor): Expeditorul transmite o secțiune de informații despre acesta, furnizând informații despre sincronizarea inter-media, contoarele cumulative de pachete și numărul de octeți trimiși

- SDES (elemente de descriere ale sursei): Informații cu scopul de a descrie sursele

- APP (funcții specifice aplicației): Este destinat utilizării experimentale pe măsură ce sunt dezvoltate noi aplicații și noi caracteristici

Folosind mesajele de mai sus, RTCP oferă următoarele servicii pentru a controla transmiterea datelor cu RTP: [4]

- Monitorizarea calității serviciilor și controlul congestiei. RTCP oferă feedback unei aplicații cu privire la calitatea distribuției datelor. Expeditorul își poate ajusta transmisia pe baza feedback-ului din raportul receptorului. Receptorii pot ști dacă o congestie este locală, regională sau globală. Managerii de rețea pot evalua performanța rețelei pentru distribuția multicast

- Identificarea sursei. Pachetele RTCP SDES (de descriere a sursei) conțin informații text, reprezentând identificatori unici la nivel global ai participanților la sesiune. Acestea pot include numele unui utilizator, numărul de telefon, adresa de e-mail și alte informații

- Sincronizare inter-media. Expeditorul RTCP raportează un indicator care specifică timpul real, împreună cu timestamp-ul pachetului RTP. Acesta este folosit pentru sincronizare inter-media

- Pachetele RTCP sunt trimise period către participanți. Pentru a putea fi scalat la grupuri mari de multicast, RTCP trebuie sa împiedice controlul traficului din a suprasolicita resursele rețelei. RTP limitează astfel controlul traficului la cel mult 5% din traficul total al sesiunii. Acesta este impus prin ajustarea ratei în funcție de numărul de participanți

1.2.3 RTSP

RTSP (Real Time Streaming Protocol, RFC 2326) este un protocol client-server de prezentare multimedia ce permite livrarea controlată a datelor multimedia transmise prin rețeaua IP. RTSP oferă metode de realizare a comenzilor (redare, derulare, pauză, oprire) similare cu funcționalitatea furnizată de CD playere sau VCR-uri. RTSP este un protocol la nivel de aplicație conceput pentru a funcționa cu protocoale de nivel inferior, cum ar fi RTP și RSVP, pentru a oferi un serviciu complet de streaming prin Internet. Poate acționa ca o telecomandă de rețea pentru servere multimedia și poate rula prin TCP sau UDP. RTSP poate controla fie un singur stream, sau mai multe stream-uri sincronizate. RTSP oferă următoarele operațiuni: [4]

- Cerere media de la server. Clientul poate solicita detalii ale prezentării și cere server-ului să seteze o sesiune separata pentru a trimite datele
- Invitația unui server media la o conferință. Serverul media poate fi invitat la conferință pentru a reda fișiere media sau pentru a înregistra o prezentare
- Adăugarea de fișiere media la o prezentare existentă. Serverul și clientul se pot notifica reciproc cu privire la orice suport suplimentar care devine disponibil

În RTSP, fiecare prezentare și stream media este identificat printr-un URL de tip RTSP. Prezentarea generală și proprietățile suportului sunt definite într-un fișier de descriere a prezentării, care poate include codificarea, limba, URL-urile RTSP, adresa de destinație, portul și alți parametri. Fișierul de descriere a prezentării poate fi obținut de către client utilizând HTTP, e-mail sau alte mijloace. RTSP își propune să ofere aceleași servicii pentru audio și video transmise așa cum HTTP oferă pentru text și grafică. Dar RTSP diferă de HTTP prin mai multe aspecte. În primul rând, în timp ce HTTP este un protocol stateless, un server RTSP trebuie să mențină "stări de sesiune" pentru a corela solicitările RTSP cu un stream. În al doilea rând, HTTP este practic un protocol asimetric, în cazul în care clientul emite cereri și serverul răspunde. Dar în RTSP, atât serverul media, cât și clientul pot emite solicitări. De exemplu, serverul poate emite o solicitare pentru a seta parametrii de redare ai unui stream. [4]

1.2.4 RSVP

RSVP (Resource reSerVation Protocol) este protocolul de control al rețelei care permite receptorului de date să solicite o calitate specială end-to-end a serviciului pentru fluxurile sale de date. Astfel, aplicațiile în timp real pot utiliza RSVP pentru a rezerva resursele necesare la routere de-a lungul căilor de transmisie, astfel încât lățimea de bandă solicitată să poată fi disponibilă atunci când transmisia are loc efectiv. RSVP este o componentă principală pentru IntServ[1], care poate oferi atât servicii de calitate bună, cât și servicii în timp real.

RSVP este utilizat pentru a seta parametrii de rezervare a resurselor rețelei. Atunci când o aplicație gazdă (receptorul stream-ului de date) solicită o anumită calitate a serviciului pentru stream-ul său de date, se utilizează RSVP pentru a livra solicitarea sa routerelor de-a lungul căilor fluxului de date. RSVP este responsabil pentru negocierea parametrilor de conectare cu aceste routere. [4]

[1] Integrated Services – arhitectura care specifica elementele ce garantează calitatea serviciilor (QoS)

2 Tehnologii utilizate

Dezvoltarea aplicației s-a realizat folosind limbajul Python 3.6.0 împreună cu librării ce au ajutat la implementarea diverselor funcționalități necesare.

PyQt5 [5] este o legătură către framework-ul Qt v5, un set de librării C++ și instrumente de dezvoltare care includ abstractizări independente de platformă pentru interfețe grafice cu utilizatorul (GUI), precum și rețele, fire de executie, expresii regulate, baze de date SQL, SVG, OpenGL, XML și multe alte caracteristici puternice. Dintre acestea au fost selectate funcționalitățile pentru generarea interfeței grafice (împreună cu tool-ul Qt Designer - Figura 2-1), stabilirea legăturii dintre elementele interfeței și codul rulat în spate, elemente care afișează informația video extrasa din fișierele multimedia folosite de către aplicație și inițializează diferite fire de execuție și asigura comunicarea între acestea.

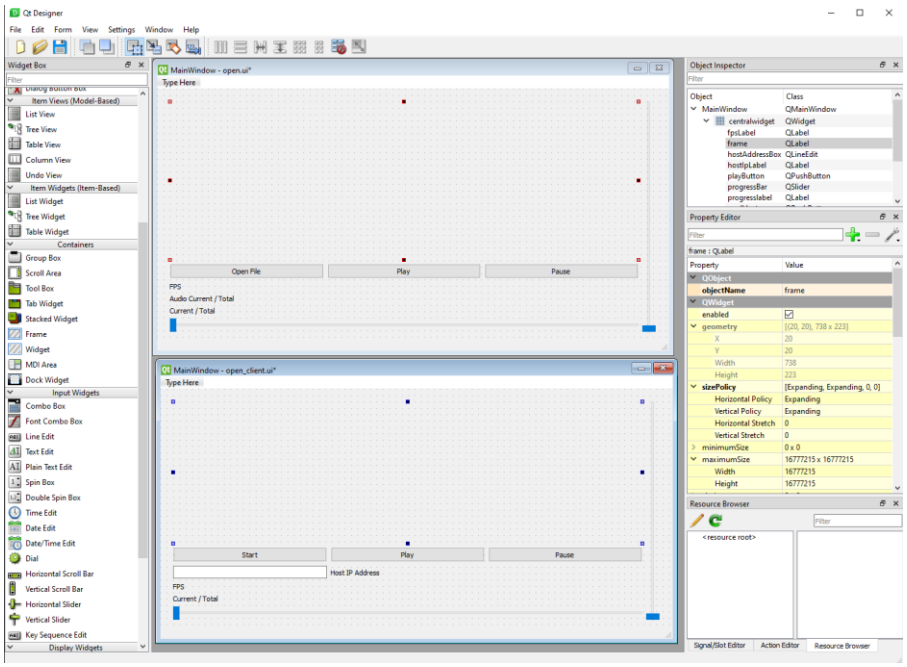


Figura 2-1- Qt Designer pentru generarea interfeței

OpenCV [6] (Open Source Computer Vision Library) este o librărie ce a fost folosită pentru extragerea și procesarea imaginilor din fișierele multimedia.

FFmpeg [7] este o colecție de librarii și tool-uri folosite la nivel de linie de comanda pentru manipularea informațiilor multimedia cum ar fi audio, video, subtitrărilor și a metadatelor. Acesta a fost utilizat în dezvoltarea aplicației cu scopul de a citi și extrage informația audio și generarea unui fișier de tip *wave* compatibil cu librăria de procesare audio folosit în aplicație.

PyAudio [8] este o legătură către PortAudio, o librărie open-source ce se ocupa cu redarea informațiilor audio primite de către fișierul generat de librăria descrisa mai sus.

Pycaw [9] (Python Core Audio Windows Library) este o librărie ce permite obținerea și setarea valorilor de volum ale aplicatiei la nivel de sistem de operare. Aceasta librărie ne limitează, din păcate, la un singur tip de sistem de operare: Microsoft Windows.

PyInstaller [10] este o librărie ce compilează codul aplicației, împreună cu toate dependentele sale într-un singur pachet. Acest pachet va putea fi rulat pe sisteme ce nu conțin un interpretor Python instalat sau module folosite în codul sursa. în momentul de fata, aceasta librărie este capabilă să compileze cod compatibil cu Windows, Mac OS X și Linux. Totuși, nu este cross-platform. O aplicație care este menită să ruleze pe Windows trebuie generată de asemenea din Windows.

Radmin VPN este o aplicație externă gratuită ce ajută la conectarea utilizatorilor aplicației OpenParty Player prin simularea unei rețele locale. Realizează o cale de comunicare criptată și e capabilă să ofere viteze de până la 100 Mbps, punând la dispoziție o interfață ușor de folosit. În schimb are două dezavantaje: codul este closed-source și aplicația este valabilă numai pentru sistemele de operare Windows.

Pentru realizarea comunicării între participanți a fost utilizată librăria nativă *socket* împreună cu aplicarea protocoalelor UDP pentru transmiterea datelor într-un mod unicast, respectiv TCP pentru funcționalitatea de chatroom și asigurarea primirii mesajelor de tip comenzi playback de la clienți către server. Metoda unicast a fost propusă deoarece aplicația a fost concepută pentru un număr relativ mic de consumatori și se dorește pe viitor controlul calității pachetelor trimise la nivel individual, în funcție de necesitățile clienților.

3 Implementarea aplicației

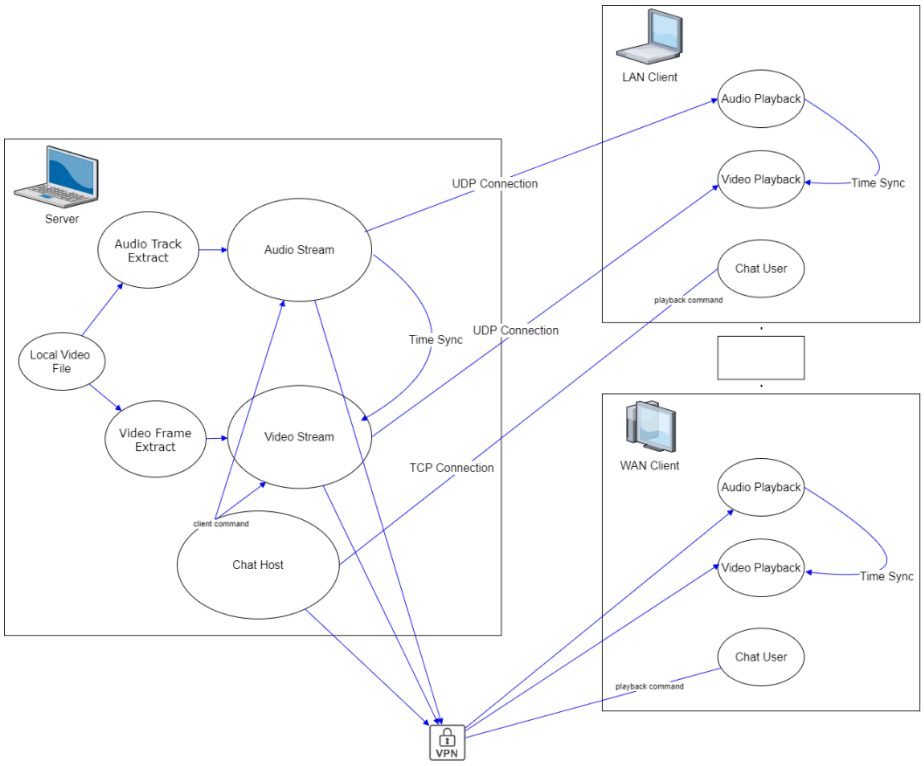


Figura 3-1 - Arhitectura aplicației

Aplicația este împărțită în 5 module pentru server, respectiv 4 module pentru client. Fiecare componenta are un rol și un scop bine definit, unele comunicând intre ele pentru a accesa informații necesare în scopuri precum sincronizări sau emiteri de comenzi și semnale. Modulele au fost împărțite in:

- Modulul principal (server /client)
- Modulul de extragere a cadrelor video din fișier (server)
- Modulul de redare și trimitere/primire a cadrelor video (server /client)
- Modulul de redare și trimitere/primire a segmentelor audio (server /client)
- Modulul de TCP chatroom (server /client)

În acest capitol vor fi abordate de asemenea niște metode de a sincroniza timpii de redare între modulele audio și video. Spre final, va fi menționată o metodă de împachetare a codului sursă pentru a permite rularea aplicației fără să fie necesară compilarea sa cu un interpretor și conectarea în afara rețelei locale folosind un VPN.

3.1 Modulul principal

Acesta, în primă fază, are rolul de a inițializa interfața grafică (Figura 3-2) dintr-un fișier .ui generat cu ajutorul lui Qt Designer (Figura 2-1) și face legătură dintre elementele sale cu codul de procesare ale celorlalte module.

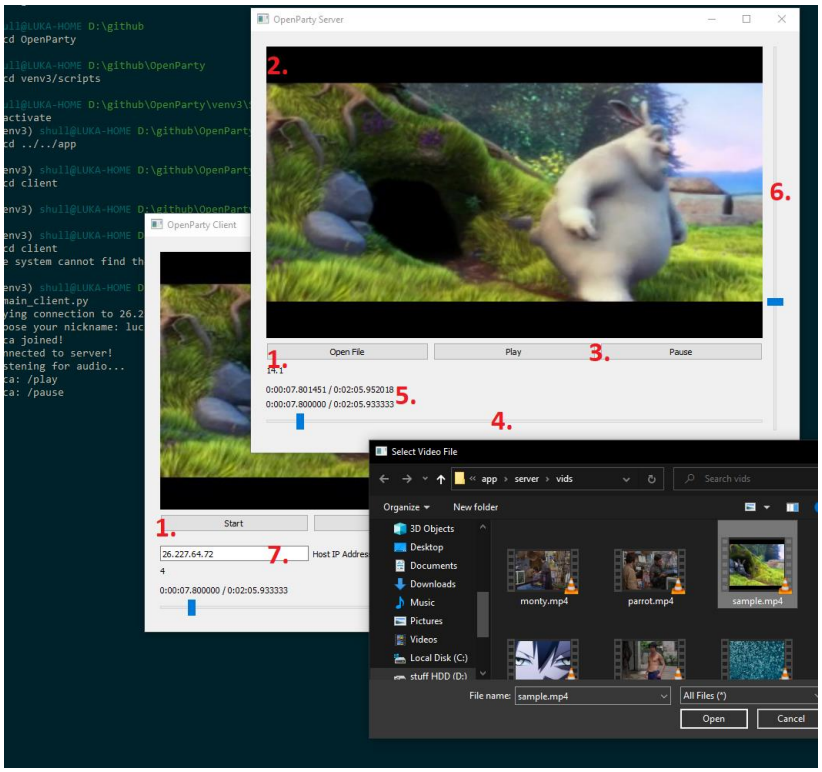


Figura 3-2 - Interfața grafică și elementele sale

1. Elementul de tip QPushButton conține următoarele funcționalități și caracteristici pentru server:

- Marcat cu textul „Open File”.
- După selectarea unui fișier, acesta se asociază unei instanțe de tip VideoCapture din *OpenCV* folosite în operațiile video și extragerea FPS[2]-ului luat din metadata fișierului.
- Se generează un nou fișier audio cu numele „temp.wav” ce va fi folosit în transmiterea și afișarea segmentelor audio. Această generare are loc cu o instrucțiune la nivelul liniei de comandă, folosind *FFmpeg*. Sunt incluși câțiva parametrii care setează fișierul generat cu bitstream/număr de canale/rata de redare prestabilite și opțiunea de suprascriere activă.
- Se inițializează o variabilă de tip coadă unde putem seta o mărime maximă. Această variabilă are rolul de a funcționa ca un intermediar între producător – consumator pentru extragerea și redarea cadrelor video în doua fire de execuție diferite.
- Se inițializează firul de execuție pentru extragerea cadrelor video.

[2] FPS – cadre pe secunda (frames per second)

- Se inițializează firele de execuție pentru trimiterea și afișarea cadrelor video și audio. Dacă au fost deja inițializate, firele de execuție sunt distruse și pornite altele, pentru a actualiza metadatele noului fișier selectat.
- Se inițializează firul de execuție pentru găzduirea chatroom-ului și așteptarea clienților. Daca a fost deja inițializat, atunci el va face legătura cu noile fire audio/video create și le transmite lista curentă cu adresele IP ale clienților
- Se inițializează o variabilă de tip pycaw care face legătura cu mixerul audio al sistemului de operare (valabil doar cu Windows)

Pentru client:

- Marcat cu textul „Start”
- Se inițializează firele de execuție pentru primirea și afișarea cadrelor video și audio.
- Se inițializează firul de execuție pentru conectarea la chatroom-ul ținut de server. Interfața clientului va conține un text box în care se introduce IP-ul server-ului
- Se inițializează o variabila de tip *pycaw* care face legătura cu mixerul audio al sistemului de operare (valabil doar cu Windows)

2. Element de tip *QLabel* este transmis către firul afișării cadrelor video.

3. Două elemente de tip *QPushButton* trimise în inițializarea firelor de redare audio/video pentru a opri/reporni derularea.

4. Element de tip *QSlider* trimis în inițializarea firului de redare video și audio pentru a afișa sau seta poziția în timp a conținutului.

5. Element de tip *QLabel* ce afișează timestamp-ul în format hh:mm:ss.ms în care se afiseaza timpul curent/timpul total. Pe partea de server vor fi doua etichete: una pentru timestamp-ul video și cealaltă pentru timestamp-ul audio, folosite pentru debug.

6. Element de tip *QSlider* ce este folosit pentru reglarea volumului aplicației. Acesta utilizeaza în spate un obiect *SimpleAudioVolume* ce va putea interacționa direct cu mixerul audio din Windows.

7. Element de tip *QLineEdit* folosit în interfața client pentru a introduce adresa IP a serverului

3.1 Modulul de redare și trimitere/primire a cadrelor audio

În inițializarea modulului se creează legăturile cu elementele necesare din interfața, se setează socketul UDP și se creează obiectul de tip *PyAudio* responsabil cu redarea sunetului.

Din modulul principal, fișierul deschis va fi folosit împreuna cu librăria *fmmpeg* pentru a genera un fișier *.wav* folosit în stream. Fișierul va fi reconstruit cu o frecvență de 44100 Hz, un bitrate de 160 kbps și 2 canale pentru redare stereo. Reconstruind fișierul cu aceste valori cunoscute, vom ști ce parametrii de intrare vor fi folosiți pentru obiectul redării audio pe partea de client (Tabel 1).

```
self.p = pyaudio.PyAudio()
self.CHUNK = 1024
self.stream = self.p.open(format=self.p.get_format_from_width(2),
                          channels=2,
                          rate=44100,
                          output=True,
                          frames_per_buffer=self.CHUNK)
```

Tabel 1 - Inițializarea obiectului de stream din client cu valorile folosite în generarea fișierului wave

Pe partea de server, fișierul generat se poate deschide folosind librăria nativă *wave* din python cu scopul transmiterii datelor către clienți. în funcția principală a modulului se citesc datele din obiectul *wave* pe bucăți de câte 1024 de octeți.

Actualizăm timestamp-ul audio în interfață, trimitem datele către adresele clienților prin socket-ul UDP și le scriem în obiectul *pyaudio* pentru redarea sunetului în aplicația server-ului. Această funcție va fi repetată la o anumita frecvență, folosind un obiect *QTimer* atașat de funcțiile legate de butoanele Play și Pause, similar cu timer-ul din Modulul de redare și trimitere/primire a cadrelor video. Încă o funcție va mai fi creată și legată de slider-ul din interfață. Folosind funcția wave.setpos() setăm poziția aleasă din slider (Tabel 2).

```
self.wf = wave.open("temp.wav")
self.data = self.wf.readframes(self.CHUNK)
current_position = self.wf.tell()
current_second = current_position / self.sample_rate
total_seconds = self.total_frames / self.sample_rate
progress = str(timedelta(seconds=current_second)) + ' / ' \
         + str(timedelta(seconds=total_seconds))
self.audioProgressLabel.setText(progress)

# send the audio data by chunks to the list of clients
for client în self.clients:
    self.audio_socket.sendto(self.data, (client, self.client_port))

# playback audio locally
self.stream.write(self.data)
```

Tabel 2 - Cod de citire, transmitere și redare audio pe bucăți

Pe partea de client deschidem un fir de execuție separat care așteaptă datele de la server prin socket-ul UDP și le introduce într-o coadă de așteptare. Funcția de redare audio citește aceste date din coadă și le scrie în obiectul de stream audio. Coada de așteptare nu trebuie sa aibă o mărime prea mare, deoarece redarea sunetului va fi continuata pana la golirea cozii în aplicația clienților, chiar și după ce aplicația server pune pauza. Serverul nu folosește o coadă, ci redă segmentele audio în același timp cu transmiterea lor.

3.2 Modulul de redare și trimitere/primire a cadrelor video

În primă fază, se conectează elementele 2, 3, 4 și 5 ale interfeței grafice (Figura 3-2) și se primesc informații din Modulul principal. Un socket de tip UDP va fi inițializat cu scopul de a trimite cadrele video către lista IP a clienților, respectiv primi cadrele de la server. Lista din server este actualizată de către Modulul de TCP la fiecare primă inițializare sau la fiecare conectare a unui nou participant în chat.

Funcționalitatea primară a modulului pe partea de server este descrisă astfel: se ia un cadru cu numărul sau de secvență din coada comuna cu Modulul de extragere a cadrelor video. Folosind apoi următoarea formulă, se calculează timestamp-ul curent, împreună cu timestamp-ul total al derulării video:

$timestamp\ video\ in\ secunde = \frac{\text{nr\_secventa}}{FPS\_metadata}$	(actualizat în eticheta de timestamp a video-ului la fiecare cadru cu formatarea hh:mm:ss.ms)

Tabel 3 - Calcul timestamp în secunde pentru video

Este folosită o metodă de compresie JPEG din librăria OpenCV la fiecare cadru înainte de a fi trimis clienților folosind funcția *imencode* cu o calitate pre-setată de Q=80. Astfel, reușim să reducem mărimea unui cadru cu o rată de compresie de până la 26:1. Se stochează cadrul împreună cu numărul său de secvență și timestamp-ul într-un dicționar, acestea urmând a fi serializate și trimise către lista ce conține IP-urile clienților, actualizată de firul de execuție a Modulul de TCP chatroom. Se va afișa cadrul video în elementul 2 din interfață (Figura 3-2). Modulul repetă aceste operații până când va fi semnalată oprirea sau mutarea poziției în timp, folosind bara de progres.

Bara de progres execută o funcție care mai întâi oprește actualizarea poziției cât timp este apăsată. Se eliberează apoi coada de așteptare a cadrelor, generată în Modulul de extragere a cadrelor video, și obiectul de tip *VideoCapture* va fi setat pe poziția aleasă (Tabel 4).

```
def when_slider_pressed(self):
    self.slider_pressed = True
…
def play_video(self):
    …
    if self.slider_pressed is False:
        self.progressBar.setValue(current_frame_no)
    …
def move_progress_bar(self):
        try:
            value = self.progressBar.value()

            # stop video playback timer
            self.timer.stop()
            self.threadVideoGen.stop_q = True

            # empty remaining frames stored în queue
            while not self.q.empty():
                self.q.get()

            self.cap.set(1, value)
            self.threadVideoGen.stop_q = False
            self.slider_pressed = False
            if not self.is_paused:
                self.timer.start(self.frame_freq * 1000)
        except Exception as e:
            logging.error(e)
```

Tabel 4 - Actualizarea poziției folosind bara de progres

Pe partea de client se inițializează firul de execuție cu legăturile interfeței și se pornește un fir separat care așteaptă mesaje de la server. Acesta vor fi introduse într-o coadă de așteptare. Funcția principală ia cadrele din acea coadă, deserializează structura și aplică o funcție de decodare pentru a afișa cadrul în aplicație. Pe când serverul folosește timer-ul pentru a opri sau reporni execuția (descris în Metoda de sincronizare audio-video mai jos), clientul, la apăsarea butoanelor din elementul 3 al interfeței (Figura 3-2), va trimite printr-un socket TCP mesaje de cerere către server. Serverul va citi comenzile din Modulul de TCP chatroom și le va executa ca și când ar fi fost citite din interfața sa locală.

3.2.1 Modulul de extragere a cadrelor video

Valabil doar pe partea de server. Acesta este inițializat în Modulul principal și are rolul de producător pentru redarea și trimiterea cadrelor video, împreună cu numărul lor de secvență, folosind o coada de stocare comuna cu Modulul de redare și trimitere/primire a cadrelor video. Funcționează ca un buffer pentru redarea parții video. Aici putem modifica cadrele. Singura modificare prezentă la momentul actual este micșorarea mărimii cadrelor pentru a ne permite transmiterea lor prin socket UDP. Tot în acest modul tratăm posibile erori, cum ar fi lipsa anumitor cadre din fișier.

3.3 Metoda de sincronizare audio-video

Din cauza faptului că viteza de procesare și afișare a fiecărui cadru video variază, trebuie să aplicăm o metodă care ajustează funcția de afișare la o anumită frecvență reprezentată de variabila FPS extrasă din metadata fișierului. Folosim următorul pseudocod [11]:

```
timp_de_asteptare = 1 / fps_metadata
tpf[3] = timpul_curent()
repeta cat timp avem cadre și status = redare:
      proceseaza_si_afiseaza_cadru()
      fps_actual = 1 / (timpul_curent() - tpf)
      tpf = timpul_curent()
      daca fps_actual > fps_metadata:
            mareste timp_de_asteptare
      altfel:
            micsoreaza timp_de_asteptare
      asteapta_in_secunde(timp_de_asteptare)
```

Tabel 5 - Pseudocod de sincronizare cu FPS-ul din metadata

Pentru a aplica timpul de așteptare, folosim o noțiune din librăria *PyQt* numită *QTimer*, care funcționează ca un cronometru cu un timp de așteptare stabilit. Se face legătura cu o funcție si, odată ce a fost pornit timer-ul, așteaptă cu timpul stabilit și aplică funcția. Timpul este apoi resetat la valoarea inițială și se repetă cât timp timer-ul rămâne în statusul activ. Se poate actualiza oricând în timpul execuției un timp de așteptare nou ce va fi aplicat la următoarea resetare. Statusul timer-ului se poate modifica în activ/inactiv în interiorul firului de execuție în care a fost creat.

Timer-ul nu poate fi oprit sau pornit de către alt fir de execuție în mod direct. Pentru a comunica acestuia oprirea/pornirea din exteriorul firului de execuție, se folosește o altă noțiune din librăria *PyQt* numită *pyqtSignal*. Se creează un atribut al clasei de acest tip și se face legătura acestuia cu o funcție definita în interior. Atâta timp cât avem acces la instanța acelei clase, putem folosi atributul din exterior pentru a semnala firului să execute funcția (în cazul de față, semnalăm firului de execuție sa oprească/pornească timer-ul din interior). Astfel, un modul exterior poate porni/opri acel timer de frecvență atunci când îndeplinește anumite condiții, cum ar fi o comandă de la un client conectat. Aplicând sincronizarea descrisă în Tabel 5, redarea cadrelor va da o senzație de mișcare normală. Totuși acesta nu asigură sincronizarea cu redarea audio.

Rulând Modulul de redare și trimitere/primire a cadrelor audio, s-a observat o redare naturală a sunetului, fără sa fie nevoie de reglări. Astfel, ne putem folosi de timestamp-ul audio pentru a aplica reglări în frecvența de afișare a cadrelor video. Putem include instanța firului de execuție audio pentru a accesa momentul curent în care se află redarea.

[3] tpf = momentul in care a fost afișat cadrul precedent (time previous frame)

Înlocuind în Tabel 5 variabilele de FPS cu timestamp-ul audio, respectiv video, obținem următorul cod:

```
timp_de_asteptare = 1 / fps_metadata
repeta cat timp avem cadre și status = redare:
      proceseaza_si_afiseaza_cadru()
      timestamp_video = nr_secventa_video / FPS_metadata
      timestamp_audio = (1024 * nr_bucati) / sample_rate
      daca timestamp_video > timestamp_audio:
            mareste timp_de_asteptare
      altfel:
            micsoreaza timp_de_asteptare
      asteapta_in_secunde(timp_de_asteptare)
```

Tabel 6 - Pseudocod de sincronizare cu timestamp audio

Aplicând numai pseudocodul descris în Tabel 6, timestamp-urile video și audio vor fi sincronizate, însă redarea video face schimbări prea bruște la frecvența de afișare.

Folosind împreună cele două metode (Tabel 7), reușim să sincronizăm redarea audio cu video, iar frecvența cadrelor video va fi stabilă:

```
timp_de_asteptare = 1 / fps_metadata
tpf = timpul_curent()
repeta cat timp avem cadre și status = redare:
      proceseaza_si_afiseaza_cadru()
      timestamp_video = nr_secventa_video / FPS_metadata
      timestamp_audio = nr_octeti_cititi / rata_redare
      daca timestamp_video > timestamp_audio:
            mareste timp_de_asteptare
      altfel:
            micsoreaza timp_de_asteptare

      fps_actual = 1 / (timpul_curent() - tpf)
      tpv = timpul_curent()
      daca fps_actual > fps_metadata:
            mareste timp_de_asteptare
      altfel:
            micsoreaza timp_de_asteptare
      asteapta_in_secunde(timp_de_asteptare)
```

Tabel 7 - Pseudocod de sincronizare cu timestamp audio și FPS-ul din metadata

Cu un fișier audio-video, putem analiza schimbările în FPS-ul actual în timpul rulării aplicației. Reamintim că FPS-ul actual a fost calculat cu formula:

FPS_actual = 1 / (timpul_curent() – timp_cadru_precedent())

Astfel, dintr-un set de date de 1500 de cadre (1 minut de redare cu 25 FPS) am extras FPS-ul actual calculat pentru fiecare cadru cu toate cele 3 metode de sincronizare:

	Sync cu FPS metadata	Sync cu audio timestamp	Sync cu ambele metode
Medie FPS	25.985	31.96343	27.18817
Deviație Standard	6.79591	35.19043	8.419155

Tabel 8 - Comparații intre metodele de sincronizare audio și FPS cu video

În concluzie, pseudocodul din Tabel 7 poate fi folosit ca să ținem redarea ambelor module sincronizată și, în același timp, redarea cadrelor video nu va fi deranjantă ochiului uman.

3.4 Modulul de TCP chatroom

Modulul va fi inițializat pe partea de server cu un socket TCP care așteaptă conexiuni și o listă ce va fi folosită în transmiterea adreselor clienților către firele de execuție audio și video. Va fi executată o funcție ce îndeplinește următoarele roluri: [12]

- așteaptă și acceptă conexiuni noi și confirmă către realizarea cu succes a conexiunii client printr-un mesaj
- trimite un mesaj prin care cere clientului conectat să introducă un username
- anunță toți participanții pe chat cine s-a conectat
- adaugă adresele IP ale conexiunilor noi într-o listă
- trimite lista adreselor IP firelor de redare audio și video
- pornește câte un fir de execuție pentru fiecare client care execută o funcție numită *handle*

Funcția *handle* așteaptă alte mesaje și verifica daca acestea sunt comenzi playback. Astfel, au fost tratate 3 posibile comenzi (Tabel 9):

- „/pause” trimite un semnal firelor de execuție pentru redare audio și video să oprească excuția
- „/play” trimite câte un semnal firelor de execuție pentru redare audio și video sa reînceapă execuția
- „/skipto <număr>” accesează firele de redare audio/video și le trimite numărul la care sa se repoziționeze player-ul. Acest număr reprezintă numărul de secvență al cadrelor video

În caz de eroare, clientul este considerat scos din sesiune. Se actualizează lista de adrese a clienților în firele de redare audio/video și serverul anunță restul participanților cine a ieșit.

```
if command == '/play':
    self.threadVideoPlay.playSignal.emit()
    self.threadAudioPlay.playSignal.emit()
    if self.threadVideoPlay.is_paused:
        self.broadcast('{} resumed playback'.format(user_msg["user"]))
elif command == '/pause':
    self.threadVideoPlay.stopSignal.emit()
    self.threadAudioPlay.stopSignal.emit()
    if not self.threadVideoPlay.is_paused:
        self.broadcast('{} stopped playback'.format(user_msg["user"]))
elif command[:7] == '/skipto':
    try:
        frame_nb = int(command[8:])
        if frame_nb > self.threadVideoPlay.totalFrames:
            raise Exception('invalid frame number selected')
        self.threadVideoPlay.stopSignal.emit()
        self.threadAudioPlay.stopSignal.emit()
        self.threadVideoPlay.move_progress_bar_client(frame_nb)
        self.threadAudioPlay.move_slider_client(frame_nb)
        if not self.threadVideoPlay.is_paused:
            self.threadVideoPlay.playSignal.emit()
            self.threadAudioPlay.playSignal.emit()
        self.broadcast('{} skipped playback'.format(user_msg["user"]))
    except Exception as e:
        logging.error('Error reading frame skip command\n')
```

Tabel 9 - Procesarea comenzilor primite de la clienți

Pe partea de client, la inițializare, se face legătura cu socket-ul creat în Modulul principal și se așteaptă un input de username de la tastatură. Se încercă apoi realizarea conexiunii cu serverul prin adresa introdusă în elementul 7 din interfață (Figura 3-2).

In funcția principală se deschid 2 fire de execuție:

- un fir așteaptă input-uri de la tastatură. La conectarea inițială, serverul trimite un mesaj prin care se cere introducerea unui nume. atașează numele și mesajul într-o structură de tipul {„user”: user, „msg”: message}, o serializează și o trimite prin socket-ul TCP către server

- un fir așteaptă mesaje de la server, acestea fiind de fapt mesaje trimise de alți utilizatori către server și transmise mai departe către toți participanții sesiunii

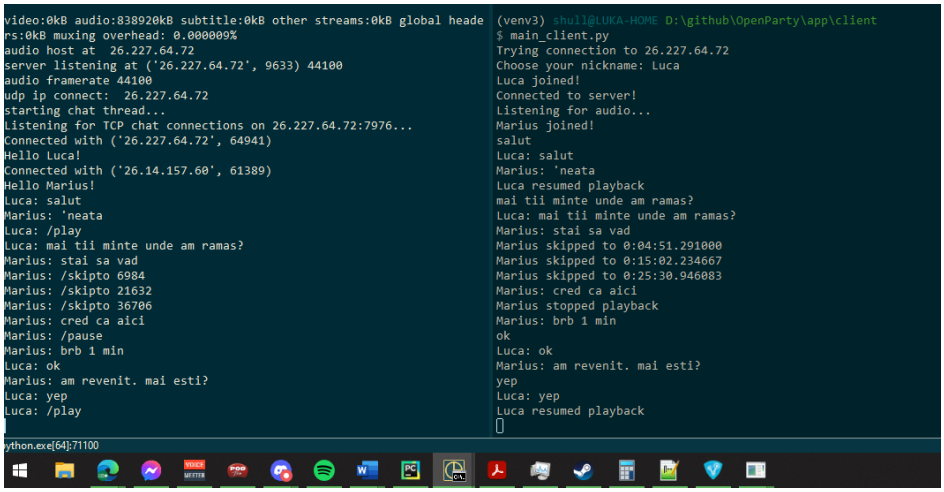


Figura 3-3 - Exemplu de chatroom în linia de comanda cu server și 2 clienți

3.5 Împachetare și rulare în afara LAN

Folosind librăria *pyinstaller*, putem împacheta codul sursă pentru a putea folosi aplicația fără să trebuiască folosit interpretorul python sau instalate librăriile folosite de acesta (**Error! R eference source not found.**). Singura dependență necesară este totuși ca serverul sa aibă setat în variabila de mediu PATH calea către librăria *ffmpeg*, din moment ce este o librărie externă și extragerea audio are loc cu o instrucțiune în linia de comandă.

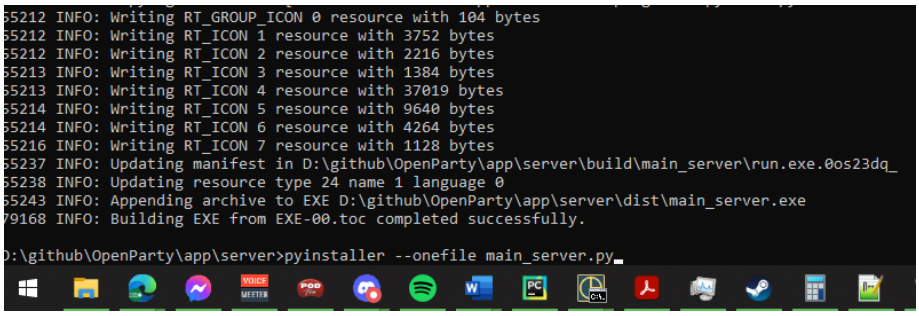


Figura 3-4 - Împachetare cod sursă în executabil Windows

Împreuna cu un VPN, putem stabili o cale care să permită comunicarea în afara rețelei locale, pentru a transmite datele către utilizatori. Exemplul folosit în testarea aplicației OpenParty Player a fost Radmin VPN.

Odată inițiată conexiunea pe o camera privată, clienții pot folosi adresa IP afișată în interfață pentru a se conecta către persoana care găzduiește sesiunea OpenParty Player, odată ce un fișier a fost încărcat în aplicație de către acesta.

Din moment ce aplicația de VPN instalează un adaptor de rețea separat (Figura 3-5), este necesara dezactivarea unui serviciu ce repornește adaptorul automat și închiderea manuală a adaptorului, in caz că se dorește găzduirea pe IP-ul rețelei locale (Figura 3-6 și Figura 3-7).

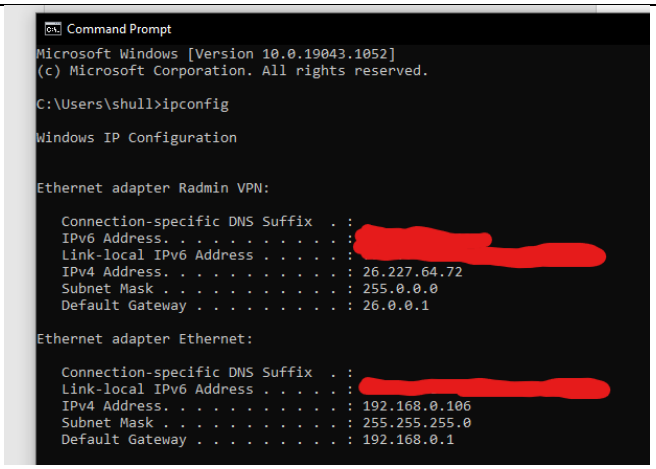


Figura 3-5 - adaptoarele pentru placa de rețea și aplicația VPN

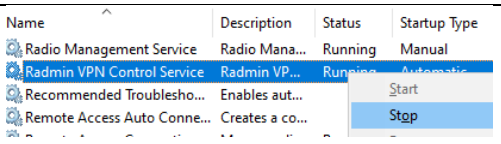


Figura 3-6 - serviciul de repornire adaptor Radmin

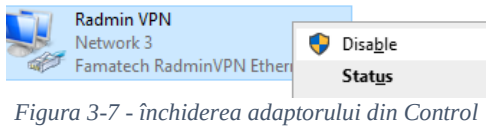


Figura 3-7 - închiderea adaptorului din Control Panel

4 Concluzii

Lucrarea de față a avut ca scop principal transmiterea multimedia în rețea cu o modalitate de sincronizare și control playback din partea oricărui participant. Transmiterea informațiilor s-a realizat prin utilizarea datagramelor, iar receptarea comenzilor pentru introducerea conceptului de *universal remote* printr-o sesiune de chat TCP. Extragerea și transmiterea informațiilor audio și video a fost realizată prin două fire de execuție, respectiv două căi de comunicare diferite, sincronizarea lor urmând a fi făcută folosind indicatorii de timp și o valoare extrasă din metadata fișierului ce indică frecvența teoretică corectă a cadrelor.

La începutul dezvoltării aplicației s-a încercat o abordare folosind numai librăria OpenCV pentru afișarea cadrelor video si crearea interfeței. S-a ajuns la concluzia că librăria nu a fost concepută pentru o interfață de media player după ce metode de actualizare a elementelor interfeței, cum ar fi bara de progres, duceau la performanțe foarte slabe în procesarea și afișarea cadrelor. Dezvoltarea unei interfețe folosind această librărie ar trebui abordată pentru o aplicație care nu actualizează automat elementele la o frecvență foarte scurtă de timp, de exemplu, setarea manuală a unor parametrii în vederea procesării imaginilor. Astfel, s-a ajuns la folosirea librăriei PyQt5 ce a ușurat considerabil efortul dezvoltării.

Pentru micșorarea datelor video a fost folosită compresia individuală a cadrelor in format JPEG cu o calitate constantă definită de Q=80. A fost studiat impactul calității asupra ratei de compresie [13], lucru care ne poate ajuta pe viitor în implementarea unui mod de control variabil al calității compresiei pentru a ne permite trimiterea de cadre la rezoluții mai mari. S-a observat spre finalul dezvoltării că folosirea unei cozi pentru receptarea cadrelor video la clienți stochează maxim un cadru. Aceasta se datorează faptului că trimiterea cadrelor se realizează in același timp cu afișarea lor pe partea de aplicație server. Pentru rezolvare, logica de transmitere ar trebui mutată în Modulul de extragere a cadrelor video, având grijă să trimitem și valori precum momentul în care s-a generat cadrul, pentru a realiza o sincronizare mai eficientă.

S-au folosit concepte similare cu descrierea părții de transport din lucrare. Cadrele se trimit împreună cu un număr de secvență și alte metadate ce ajută la actualizarea interfeței clientului și sincronizări. Comenzile de redare playback de la client sunt transmise către server într-un mod ce garantează receptarea lor. Aici mai putem dezvolta pe viitor un mecanism ce transmite feedback legat de calitatea livrării datelor. Transmiterea datelor se face într-un mod unicast, ceea ce ne permite pe viitor setarea calității datelor transportate la nivel individual, în funcție de necesitățile fiecărui client.

Un modul ce mai poate fi adăugat în viitorul apropiat ar fi un suport pentru afișarea subtitrărilor peste cadre. De asemenea, trebuie integrat modulul de TCP chat într-o bucată din interfață pentru a nu mai fi limitați la linia de comandă folosită în comunicarea participanților.

La final a fost realizată o aplicație ce își îndeplinește scopul propus inițial, urmând a mai fi introduse optimizări sau metode mai eficiente de a realiza aceste funcționalități.

Bibliografie

Referințe parte teoretică:

[1] Ponlatha S. și Sabeenian R. S. (2013) - „Comparison of Video Compression Standards" din *International Journal of Computer and Electrical Engineering*, vol. 5, nr. 6

URL: http://www.ijcee.org/papers/770-ET055.pdf

[2] Watkinson J. (2001) - "MPEG Handbook: MPEG-1, MPEG-2, MPEG-4", Focal Press

[3] Axis Communications (2008) - „An explanation of video compression techniques"

URL: https://www.accentalarms.com/specsheets/axis/_wp_videocompression.pdf

[4] Dashti A., Seon H. K. și Shahabi C. (2003) - "Streaming Media Server Design", Prentice Hall

Referințe aplicație:

[5] Riverbank Computing - „PyQt5 Components"

URL: https://doc.bccnsoft.com/docs/PyQt5/introduction.html#module-PyQt5.

[6] Intel; Willow Garage (1999) - „Open Source Computer Vision Library"

URL: https://docs.opencv.org/master/

[7] Bellard F. (2000) - „ffmpeg Documentation"

URL: https://www.ffmpeg.org/ffmpeg.html

[8] Pham H. (2006) - „PyAudio Documentation,"

URL: https://people.csail.mit.edu/hubert/pyaudio/docs/

[9] Miras A. (2020) - „Python Core Audio Windows Library"

URL: https://github.com/AndreMiras/pycaw

[10] „PyInstaller Docs"

URL: https://pyinstaller.readthedocs.io/en/stable/usage.html

[11] „How to send video using UDP socket in Python" PyShine, 2021

URL: https://pyshine.com/Send-video-over-UDP-socket-in-Python

[12] Chouhan Y. S. (2020) - „Creating Command-line Based Chat Room using Python"

URL: https://hackernoon.com/creating-command-line-based-chat-room-using-python-oxu3u33

[13] Graphics Mill (2014) - „Compression ratio for different JPEG quality values"

URL: https://www.graphicsmill.com/blog/2014/11/06/Compression-ratio-for-different-JPEG-quality-values