Project Title

StreamSocket: A Real-Time Video Streaming Solution

Submitted by

Ayesha Islam Qadri 2020-CE-36

Ayesha Shafique 2020-CE-39

Submitted to

Ma'am Darkhshan Abdul Ghaffar

Course

CMPE333L Computer Networks

Semester

6th

Date

28th March 2023

Department of Computer of Engineering

University of Engineering and Technology, Lahore

Table of Contents

Abstract

Streaming video over the Internet has drawn a lot of attention from both academia and business due to the Internet's rapid expansion and the rising demand for multimedia content on the online. Live video streaming is being used by more and more businesses as a dependable means of reaching audiences, getting over physical obstacles, and generally enhancing operations. That's because live video streaming is not only a fantastic substitute for in-person events and other media, but it's also a superior solution and more practical one in some circumstances. The proposed RTVD System (RTVDS) makes the Admin to focus on the received frames sent by IP cameras to/from the server within signals broadcasting. In addition, RTVDS has the capability of tracking and storing the frames stream and encode them during the broadcasting process.[1]

Problem Statement

Live video streaming has become a ubiquitous means of communication in various contexts, including remote work, education, entertainment, and social media. However, developing a live video streaming application with audio using socket programming with Python can be a daunting task. In this project, we aim to address this challenge by developing a live video streaming application with audio using socket programming with Python. Our application will leverage the client-server architecture to enable users to stream live videos with audio in real time.

Scope

The intention of the suggested project is to create a live video streaming application with audio that may be used in a variety of fields, such as distance learning, training, live events, conferences, and the entertainment sector. Users will be able to adjust the quality settings, and the programme will be able to stream music and video in real time. The project's scope is very broad, and it may be modified to satisfy the unique requirements of many sectors. For businesses and people that demand live video streaming with audio, it will offer a reliable, affordable, and simple alternative.

Explanation

Python socket programming will be used in the proposed project to create a live video streaming application with audio. A multithreaded server with several client connections will be part of the project. For the purpose of connecting the server and client, the project will make use of Python's socket programming language.

Live video streaming through socket programming involves the use of network sockets to establish a connection between a client and server, allowing for real-time data transfer. Socket programming allows for a direct, low-level interface with network protocols, making it a popular choice for live video streaming applications.To enable live video streaming through socket programming, the server and client applications must be developed to support this functionality. The server application acts as a "host" for the live video stream, while the client application establishes a connection to the server and receives the video data in real-time. The video data is broken down into packets and sent through the socket connection to the client, which reassembles them to display the video on the user's screen.

One of the primary benefits of live video streaming through socket programming is the ability to transmit video data in real-time, without significant delay or buffering. This is particularly important for applications such as video conferencing, live events, and gaming, where delays or interruptions can have a significant impact on the user experience.

However, live video streaming through socket programming can also present challenges, particularly in terms of bandwidth and network latency. Streaming high-quality video requires significant bandwidth, and network latency can impact the speed and reliability of the data transfer. Additionally, security considerations must be taken into account to protect the video data from unauthorized access.

1. Client -Server Archiecture

The proposed live video streaming application with audio will be developed using the client-server architecture. The client-server architecture is a distributed architecture that consists of two components: a server and multiple clients. The server provides services to multiple clients, and the clients request services from the server. The client-server architecture is commonly used in network-based applications and is well-suited for the proposed project.

- **Server**

The server component of the application will be responsible for handling multiple client requests simultaneously. It will be a multithreaded server, which means that it can handle multiple client requests concurrently. The server will receive live video and audio streams from the clients and then broadcast them to all connected clients. The server will also handle audio streams in real time, ensuring that the audio is synchronized with the video.

- **Client**

The client component of the application will be responsible for streaming a live video and audio from the server. Multiple clients can connect to the server simultaneously, and each client will receive the same video and audio stream from the server. The client will also be able to customize the video and audio quality settings to suit their preferences.

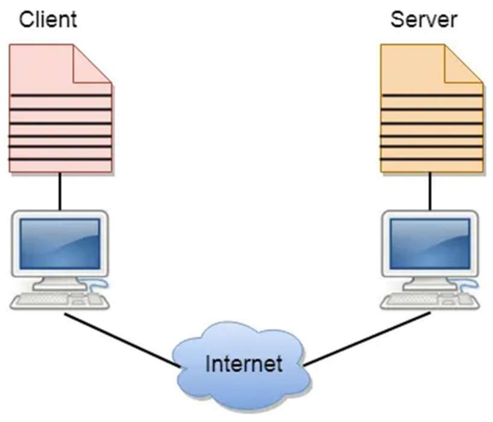


Figure 1 Client-Server Model in networks

2. Socket Programming

Socket programming is a type of network programming that enables real-time communication between two applications over a network. In the context of live video streaming, socket programming is used to establish a connection between the server and client applications, allowing for the transmission of video data in real-time.

When a live video stream is initiated, the server application breaks down the video data into smaller packets and sends them through a socket connection to the client application. The client application receives the packets and reassembles them to display the video on the user's screen. The use of socket programming allows for direct, low-level communication between the server and client applications, which is essential for real-time video streaming. The socket connection ensures that the video data is transmitted without significant delay or buffering, providing a smooth and seamless viewing experience for the user.However, there are several challenges associated with live video streaming through socket programming. High-quality video requires significant bandwidth, and network latency can impact the speed and reliability of the data transfer. Additionally, security considerations must be taken into account to protect the video data from unauthorized access.

In summary, socket programming is a powerful tool for enabling live video streaming by establishing a direct, real-time connection between the server and client applications. However, it requires careful consideration of bandwidth, latency, and security considerations to ensure a seamless and reliable user experience.

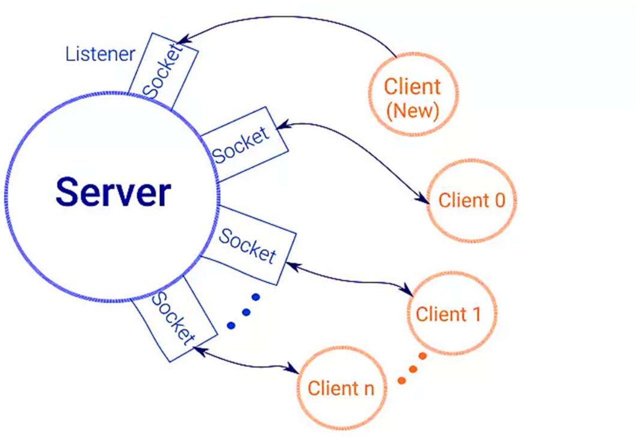


Figure 2 Sockets with Client & Server

3. Video Streaming Architecture

The architecture shown in the following diagram includes a trans-coding server, origin server, cache servers, load balancers, and DNS load balancers to guarantee seamless and effective video streaming to the end users. Two other servers make up the trans-coding server: the segmentation server and the encoding server. The transcoder is provided with the raw video as input. De-multiplexing the raw or pre-encoded video footage into individual audio or video files is followed by encoding it with an encoder and an AAC audio encoder into various adaptive bitrate video files. Only three representations high bitrate, medium bitrate, and low bitrate are examined in this study.[2] These three representations are then multiplexed and delivered to a segmenter, which divides each one into various video chunks or segments. The outcomes are then kept on the content provider's origin server or video streaming server. Afterwards, numerous web servers placed in the suitable locations receive the accessible video content in the adaptive bitrate pattern.

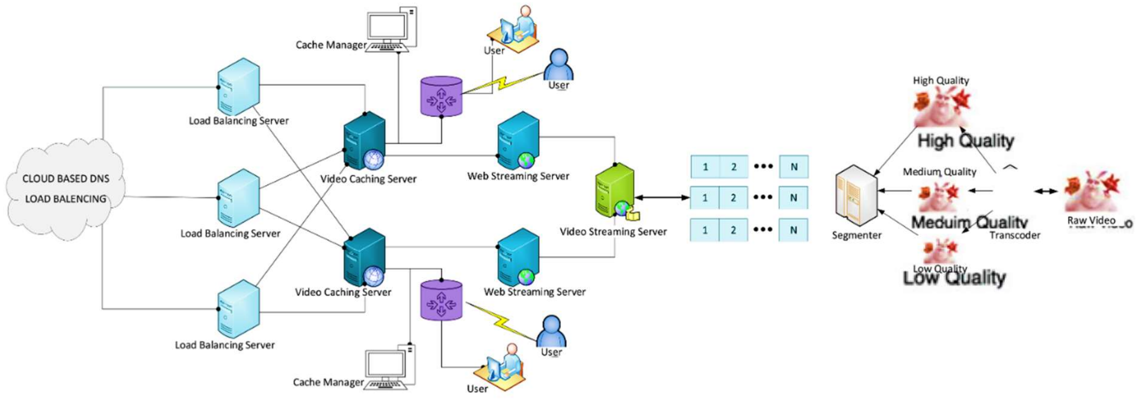


Figure 3 Architectural Diagram

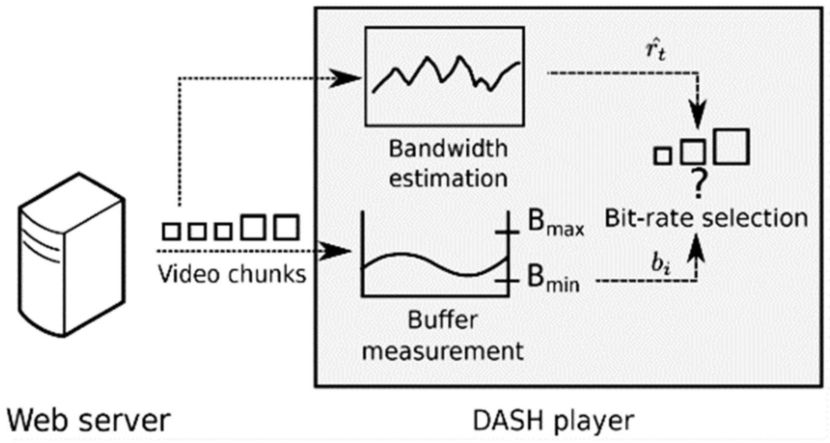


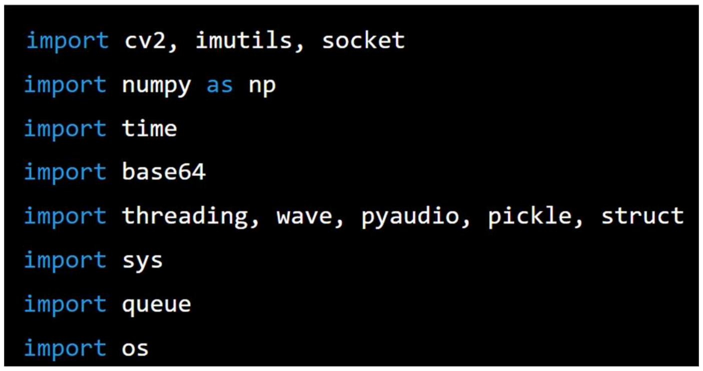
Figure 4 Video Streaming Model

Explanation of code

1. Server Side

In the server code, we first convert a given video file into an audio file using FFmpeg. Then we create a socket object and bind it to the host IP address and port number. We then set the receive buffer size of the socket to a large value. After this, we initialize the OpenCV VideoCapture object with the video file and get some information about the video such as the frame rate and total number of frames. We also initialize some global variables such as the time interval between each frame transmission and a flag to indicate whether the video player has been closed or not.

First we have to describe the imports of the code for what purpose they are use for:



```python
import cv2, imutils, socket
import numpy as np
import time
import base64
import threading, wave, pyaudio, pickle, struct
import sys
import queue
import os
```

- **cv2**: This is the OpenCV library, used for computer vision tasks such as image and video processing, object detection, and tracking.
- **Python opencv**: Use of the highly efficient library for numerical operations known as numpy is made possible by OpenCV-Python. Developed to address computer vision issues, OpenCV-Python is a collection of Python bindings. OpenCV-Python is a Python wrapper and a numpy array is used to convert each and every OpenCV array structure.
- **imutils**: A package built on top of OpenCV, providing a set of utility functions for performing common tasks in computer vision, such as resizing and rotating images, working with video streams, and more.
- **socket**: A module for working with network sockets, used in this case for sending and receiving video and audio frames over a network.
- **numpy**: A powerful library for numerical computing in Python, used extensively in scientific computing, machine learning, and other fields.
- **time**: A module for working with time, used in this code to calculate the frame rate of the video being streamed.
- **base64**: A module for encoding and decoding data using Base64, a method for representing binary data as ASCII text.
- **threading**: A module for working with threads, used in this code to run multiple functions concurrently.
- **wave**: A module for working with WAV audio files.
- **pyaudio**: A library for working with audio in Python, used in this code to play back audio.
- **pickle**: A module for serializing and deserializing Python objects.
- **struct**: A module for working with binary data in Python.
- **sys**: A module providing access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
- **queue**: A module providing a queue data structure, used in this code for buffering video frames.

- **os**: A module for working with the operating system, used in this code to run a command to convert a video file to an audio file.

```
q = queue.Queue(maxsize=10)
```

This line creates a queue object with a maximum size of 10.

```
filename = 'count1.mp4'
command = "ffmpeg -i {} -ab 160k -ac 2 -ar 44100 -vn {}".format(filename, 'temp.wav')
os.system(command)
```

These lines set the filename of the video file and convert it to a temporary WAV file using ffmpeg.

- **FFmpeg**

FFmpeg is a free and open-source software project that produces libraries and programs for handling multimedia data. It is used to record, convert and stream audio and video in various formats. FFmpeg stands for "Fast Forward MPEG" where MPEG refers to the Moving Picture Experts Group.

FFmpeg is used for a variety of multimedia applications, including:

- **Transcoding**: converting one video/audio format to another
- **Streaming:** live streaming of audio and video over the internet
- **Recording:** capturing audio and video from a variety of sources
- **Editing:** basic video and audio editing, such as cutting and splicing
- **Post-production:** adding effects, transitions, and other enhancements to multimedia content.

```
BUFF_SIZE = 65536
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, BUFF_SIZE)
host_name = socket.gethostname()
host_ip = ''  # socket.gethostbyname(host_name)
print(host_ip)
port = 9688
socket_address = (host_ip, port)
server_socket.bind(socket_address)
print('Listening at:', socket_address)
```

This block of code sets the buffer size, creates a UDP socket, gets the hostname, gets the IP address, sets the port number, sets the socket address, binds the socket to the socket address, and prints the IP address and port number of the socket.

```
vid = cv2.VideoCapture(filename)
FPS = vid.get(cv2.CAP_PROP_FPS)
global TS
if (FPS > 0):
    TS = (0.5 / FPS)
BREAK = False
print('FPS:', FPS, TS)
totalNoFrames = int(vid.get(cv2.CAP_PROP_FRAME_COUNT))
durationInSeconds = float(totalNoFrames) / float(FPS)
d = vid.get(cv2.CAP_PROP_POS_MSEC)
print(durationInSeconds, d)
```

This block of code initializes the video capture object with the specified filename, gets the frame rate of the video, calculates the time step based on the frame rate, sets the total number of frames, calculates the duration of the video, and gets the position of the video in milliseconds.

```
def video_stream_gen():
    WIDTH = 500
    while (vid.isOpened()):
        try:
            _, frame = vid.read()
            frame = imutils.resize(frame, width=WIDTH)
            q.put(frame)
        except:
            os._exit(1)
    print('Player closed')
    BREAK = True
    vid.release()
```

This function reads frames from the video object, resizes them, and puts them in the queue object. If the video object is closed, the function exits and prints a message.

```
def video_stream():
    global TS
    fps, st, frames_to_count, cnt = (0, 0, 1, 0)
    cv2.namedWindow('TRANSMITTING VIDEO')
    cv2.moveWindow('TRANSMITTING VIDEO', 10, 30)
    while True:
        msg, client_addr = server_socket.recvfrom(BUFF_SIZE)
        print('GOT connection from ', client_addr)
        WIDTH = 500
        while(True):
            frame = q.get()
            encoded, buffer = cv2.imencode('.jpeg', frame, [cv2.IMWRITE_JPEG_QUALITY,
            message = base64.b64encode(buffer)
            server_socket.sendto(message, client_addr)
```

This function receives data from the client and sends the video frames to the client. It is called as a separate thread and waits for the connection from the client. Once the connection is established, it sends the video frames in the queue to the client. It also calculates the frames per second and adjusts the time interval between frames accordingly. It also shows the transmitting video in a window.Once a connection is established, it encodes each frame of the video into JPEG format with a quality level of 80, and then sends the encoded image to the client over the network using UDP protocol. The video stream continues until the program is stopped.

2. Client Side

This line imports the necessary Python libraries and modules for the program and these all are explained above.

```
import cv2, imutils, socket
import numpy as np
import time, os
import base64
import threading, wave, pyaudio, pickle, struct
```

This line sets the buffer size for the socket.

```
BUFF_SIZE = 65536
```

This line creates a socket for UDP communication and sets the socket options.

```
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, BUFF_SIZE)
```

```
def video_stream():
```

This line defines a function **video_stream()** that receives and displays video frames. This function receives data from the server and displays the video frames in a window. It is called as a separate thread and waits for data from the server. Once the data is received, it decodes the base64-encoded message, converts it to a NumPy array, and displays it in a window.

```
def audio_stream():
```

This line defines a function **audio_stream()** that receives and plays audio frames. This function receives data from the server and plays the audio. It is called as a separate thread and waits for data from the server. Once the data is received, it converts it to bytes, unpacks the bytes, and plays the audio.

```
from concurrent.futures import ThreadPoolExecutor

with ThreadPoolExecutor(max_workers=2) as executor:
    executor.submit(audio_stream)
    executor.submit(video_stream)
```

This line uses the **ThreadPoolExecutor** to run the **audio_stream()** and **video_stream()** functions concurrently using two threads.

- **ThreadPoolExecutor**

ThreadPoolExecutor is a class in the Python concurrent.futures module which provides a high-level interface for asynchronously executing functions using threads. In the context of networking and audio/video transmission in socket programming, ThreadPoolExecutor can be used to handle multiple client connections concurrently. When a client connection is accepted by the server, a new thread is spawned to handle that client's requests. With ThreadPoolExecutor, instead of spawning a new thread for each client connection, a pool of threads is created and the incoming requests are assigned to an available thread from the pool. This can help to improve performance by avoiding the overhead of creating and destroying threads for each connection. Using ThreadPoolExecutor can be especially useful in scenarios where there are a large number of simultaneous client connections, such as in audio/video transmission where multiple clients may be streaming data to the server simultaneously. By using a thread pool, the server can handle these connections efficiently and with minimal latency.

3. Some Methods

Following are the methods used in the code are:

- **bind() method**: A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that IP and port.
- **listen() method**: A server has a listen() method which puts the server into listen mode. [3]
- **imutils.resize()**: The resize function of imutils maintains the aspect ratio and provides the keyword arguments width and height so the image can be resized to the intended width/height while maintaining aspect ratio and ensuring the dimensions of the image do not have to be explicitly computed by the developer.
- **pickle.dump()**: The dump () method of the pickle module in Python, converts a Python object hierarchy into a byte stream.
- **struct.pack ()**: This is used to pack elements into a Python byte-string (byte object).

Features

The following are the features of the live video streaming application with audio:

1. The multithreaded server can handle multiple client requests at the same time.
2. Support for audio and video streams in real-time.
3. Customizable video and audio quality settings.
4. Start and stop the server and client streams with ease.
5. Support for multiple clients simultaneously.
6. Low latency streaming for real-time communication.
7. Minimal system requirements.
8. User-friendly interface

Applications

There are many applications of live video streaming with audio, some of which are:

1. **Video Conferencing:** Live video streaming with audio is widely used in video conferencing for remote communication and collaboration between people. It allows people to communicate with each other in real-time, irrespective of their location.

2. **Online Education:** Live video streaming with audio is also used in online education for conducting live lectures and interactive sessions. It allows teachers and students to communicate with each other in real-time, providing an immersive learning experience.

3. **Gaming:** Live video streaming with audio is used extensively in the gaming industry for live streaming of gaming sessions. It allows gamers to interact with their audience in real-time, creating an engaging and interactive experience.

4. **Social Media:** Live video streaming with audio is also used in social media platforms like Facebook, Instagram, and Twitter for live streaming of events and activities. It allows people to share their experiences with their followers in real-time.

5. **Security and Surveillance:** Live video streaming with audio is used for security and surveillance purposes, enabling real-time monitoring and surveillance of sensitive areas.

Usage

The code can be used to stream videos and audio files over a network, for example, for video conferencing or remote surveillance. It can also be used as a starting point to develop more complex streaming applications that require real-time processing of frames or audio.

Future Work

- The code can be extended to support live video and audio streaming from a camera or microphone.
- The quality of stream can be improved by using more efficient video and audio codecs.
- The code can be modified to handle multiple clients concurrently.
- The code can be optimized for better performance and efficiency.

References

[1] https://ieeexplore.ieee.org/document/9019347/figures#figures

[2] https://www.mdpi.com/2076-3417/12/20/10274

[3] https://sourcenet.hashnode.dev/video-streaming-using-socket-programming-in-python