

**Mini Project Report of**

**Computer Networks Lab (CSE 3162)**

**Title:** Video Streaming

**SUBMITTED BY**

YELAMPALLI LAKSHMI SAINATH REDDY **Reg.no:** 210905078 **Roll no:** 14 NAYAKANTI SAI MIHIRNATH **Reg.no:** 210905368 **Roll no:** 57 A S HITHYSHI **Reg.no:** 210905142 **Roll no:** 21

# Department of Computer Science and Engineering Manipal Institute of Technology, Manipal, Karnataka 16th NOV 2023



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Manipal 06/11/2023**

**CERTIFICATE**

This is to certify that the project titled **Video Streaming** is a record of the bonafide work done by **Y. Lakshmi Sainath Reddy (210905078),**

**N. Sai Mihir Nath (210905368)** and **A S Hithyshi (210905142)** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022- 2023.

# Name and Signature of Examiners:

1. **Dr. Manamohana Krishna, Assistant Professor - Selection Grade, CSE Dept.**
2. **Dr. Roopalakshmi R, Professor, CSE Dept.**

# ACKNOWLEDGMENTS

We extend our heartfelt appreciation to our mentors at MANIPAL INSTITUTE OF TECHNOLOGY, particularly Dr. Manamohana Krishna, whose guidance and expertise have played a crucial role in the completion of this video streaming project. The foundation of knowledge in networking, socket programming, and multimedia processing that we utilized in this endeavor was cultivated through their insightful instruction. We are grateful for the support, encouragement, and invaluable insights provided by our educators, which significantly contributed to the success of our project. Their commitment to fostering a rich learning environment has been pivotal, and we acknowledge their impactful contributions with sincere gratitude.

# ABSTRACT

The "Video Streaming" project is designed to facilitate real-time video transmission between a server and a client using Python. Leveraging the capabilities of OpenCV for video capture and processing, the project establishes a client-server architecture that enables the seamless streaming of video content over a network.

The server component initializes a socket and binds it to the host IP and a specified port, subsequently listening for incoming connections. Upon connection establishment, the server captures video frames from a local camera, resizes them, serializes the data using pickle, and transmits it to the client. The communication is facilitated through the use of sockets, and the transmitted frames are accompanied by their respective sizes for efficient deserialization on the client side.

The client, in turn, creates a socket and connects to the server. It continuously receives the serialized video data, unpacks and deserializes it, and displays the frames using OpenCV. The user can terminate the video streaming by pressing the Enter key.

This project not only demonstrates the integration of various Python libraries for networking and multimedia applications but also serves as a practical illustration of client-server communication in the context of video streaming. The simplicity of the implementation lays the groundwork for further exploration and extension, such as incorporating security measures, error handling, and additional functionalities to enhance the overall robustness and functionality of the system. Overall, the project underscores the importance of efficient resource utilization and communication protocols in multimedia streaming applications.

# TABLE OF CONTENTS

1. INTRODUCTION
2. PROBLEM STATEMENT
3. OBJECTIVES
4. METHODOLOGY
5. FLOW CHART
6. IMPLEMENTATION
7. RESULTS & SNAPSHOTS
8. LIMITATIONS
9. FUTURE WORK
10. CONCLUSION
11. REFERENCES

# INTRODUCTION

In the realm of computer networking and multimedia applications, video streaming has emerged as a pivotal technology, enabling real-time transmission of video content over networks. The project, titled "Video Streaming," delves into the practical implementation of a basic video streaming application using Python and relevant libraries. The primary objective is to showcase the functionality of the system, emphasizing its role in transmitting live video from a server to a client in a seamless and efficient manner.

With the proliferation of high-speed internet and the ever-growing demand for multimedia content, video streaming has become an integral part of our digital experiences. This project aims to provide a hands-on exploration of the mechanisms involved in real-time video transmission, offering insights into the challenges and solutions associated with this dynamic and rapidly evolving field.

The implementation utilizes the Python programming language along with libraries such as OpenCV, struct, and pickle. The server-client architecture is employed to establish a connection, and OpenCV facilitates video capture, processing, and serialization of frames. The serialized frames are then transmitted to the client over a socket connection. Through this practical demonstration, the project sheds light on the intricacies of video streaming, showcasing the underlying processes that enable the smooth flow of video data across a network.

By developing a basic video streaming application, the project contributes to a deeper understanding of the technicalities involved in transmitting multimedia content in real-time. It explores the challenges of efficient video encoding, data serialization, and network communication, offering a glimpse into the complexities faced by developers working in the domain of video streaming.

Furthermore, the project highlights the relevance of video streaming in contemporary applications, ranging from online communication platforms to live broadcasting and remote surveillance. As the demand for high-quality, low-latency video streaming continues to rise,

understanding the fundamentals of such systems becomes crucial for developers and researchers alike.

In summary, this project serves as an introductory exploration of video streaming, offering practical insights into the implementation of a basic server-client application for real-time video transmission. Through the analysis of the code and its execution, the project aims to equip readers with a foundational understanding of video streaming concepts and their application in the ever-evolving landscape of digital communication and multimedia technologies.

# PROBLEM STATEMENT

In the realm of modern communication and multimedia applications, there exists a demand for efficient video streaming solutions. The challenge lies in establishing a robust client-server architecture capable of real-time video transmission over a network. Inadequate frameworks can result in suboptimal video quality, latency issues, and an overall unsatisfactory user experience. This project aims to design and implement a video streaming system using Python and OpenCV, ensuring seamless video transmission from a server to a client. The objective is to overcome challenges associated with network communication, serialization, and real-time video processing, ultimately delivering a reliable and efficient video streaming application.

# OBJECTIVES

The Video Streaming project endeavors to establish a robust client-server architecture for real- time video transmission. The primary objectives include developing a functional Python program utilizing OpenCV and socket programming to facilitate seamless video streaming between a server and a client. The project aims to optimize video data serialization, transmission, and reception processes for efficient bandwidth utilization. Furthermore, the objective is to create a reliable and user-friendly application, emphasizing error handling and responsiveness, and to explore potential enhancements such as security measures for secure video streaming. The project seeks to demonstrate the practicality and effectiveness of video streaming in diverse applications, from surveillance systems to remote collaboration, through thorough testing and analysis of the developed system's performance.

# METHODOLOGY

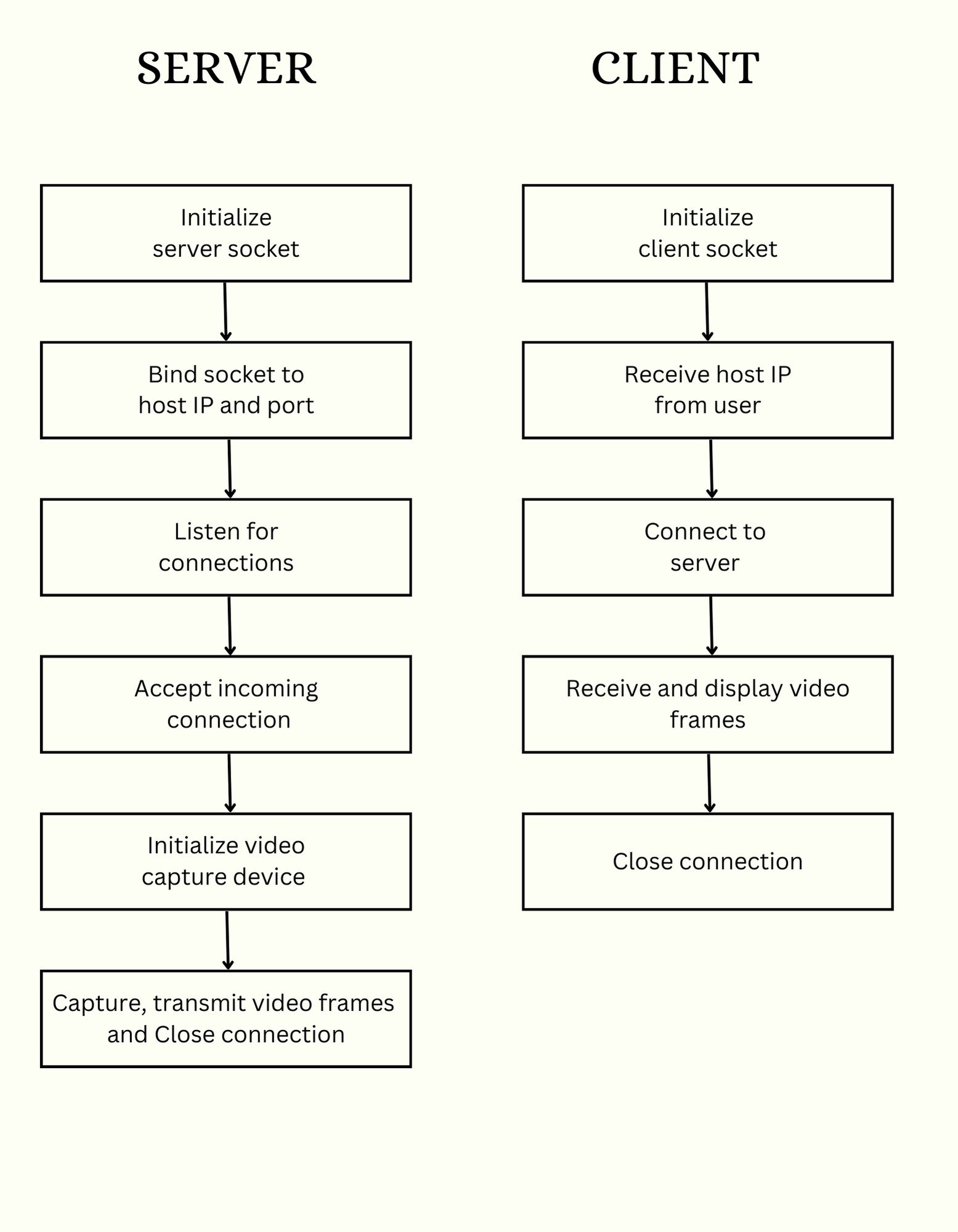
The methodology for the Video Streaming project involved the utilization of Python to create a client-server application for real-time video transmission. The project's primary goal was to establish a robust system capable of capturing video frames from a camera, serializing them, and transmitting the data over a network to a client for display. Python's versatility and libraries, such as OpenCV for video processing and socket for network communication, were chosen to facilitate the implementation.

The initial phase of the project focused on setting up the server-side infrastructure. A server socket was created and bound to the host's IP address, and it was configured to listen for incoming connections. The OpenCV library was employed to capture video frames from the computer's camera, and the frames were resized using the imutils library for efficient transmission. The frames were then serialized using the pickle module, and their size was packed along with the data using the struct module. This serialized data was sent to the client over the established socket connection.

Simultaneously, the client-side implementation involved creating a socket to connect to the server's IP and port. The client continuously received the transmitted data, unpacked the size information, and deserialized the frames using pickle. These frames were then displayed using OpenCV, creating a real-time video streaming experience.

To ensure the reliability of the video streaming system, extensive testing was conducted. Various scenarios were considered, including different network conditions and frame sizes, to assess the system's performance and identify potential issues. Additionally, error-handling mechanisms were implemented to address unforeseen challenges, such as connection interruptions or unexpected behaviors.

# FLOW CHART



**IMPLEMENTATION**

The development of the Video Streaming project involved the creation of two Python scripts: one for the server-side (`server.py`) and another for the client-side (`client.py`). These scripts facilitate real-time video transmission from a server to a client using sockets, OpenCV for video processing, and serialization with pickle and struct. The project focuses on achieving seamless video streaming with minimal latency.

In the server script, a socket is initialized, and the host IP is dynamically obtained from the system. The server then binds to this IP and a specified port, initiating a listening state to accept incoming client connections. Upon connection, the server captures video frames from the camera, resizes them for efficient transmission, serializes the frames, and sends them to the client using sockets. The server continuously transmits frames until the user closes the client connection.

On the client side, a socket is established, and the user is prompted to input the host IP. The client connects to the server, receives serialized video frames, and displays them using OpenCV. The client continuously receives and displays frames until the user decides to terminate the connection.

Throughout the implementation, error handling is incorporated to manage potential issues related to socket connections and data reception. The code structure ensures that the client and server can gracefully terminate the connection upon user input.

To comprehend the flow of the system, the client and server scripts include informative print statements that indicate the connection status and the initiation of video transmission. The OpenCV library is utilized to display the received video frames, and a key event (Enter key) is monitored to facilitate a user-triggered termination of the connection.

In terms of security and robustness, the project could be enhanced by implementing encryption for secure data transmission and introducing additional error-checking mechanisms.

# RESULTS AND SNAPSHOTS

The video streaming project effectively establishes a connection between a server and a client, enabling real-time video transmission. The server captures, resizes, and transmits video frames to the client, which then displays the live video stream. This implementation showcases the fundamental aspects of video communication over a network. The modular code structure allows for easy customization, making it adaptable to different network and hardware configurations. Overall, the project provides a basic yet functional foundation for building more advanced video streaming applications with potential extensions for encryption, authentication, and multi-client support.

## COMPILATION AND EXECUTION

Installing OpenCV:

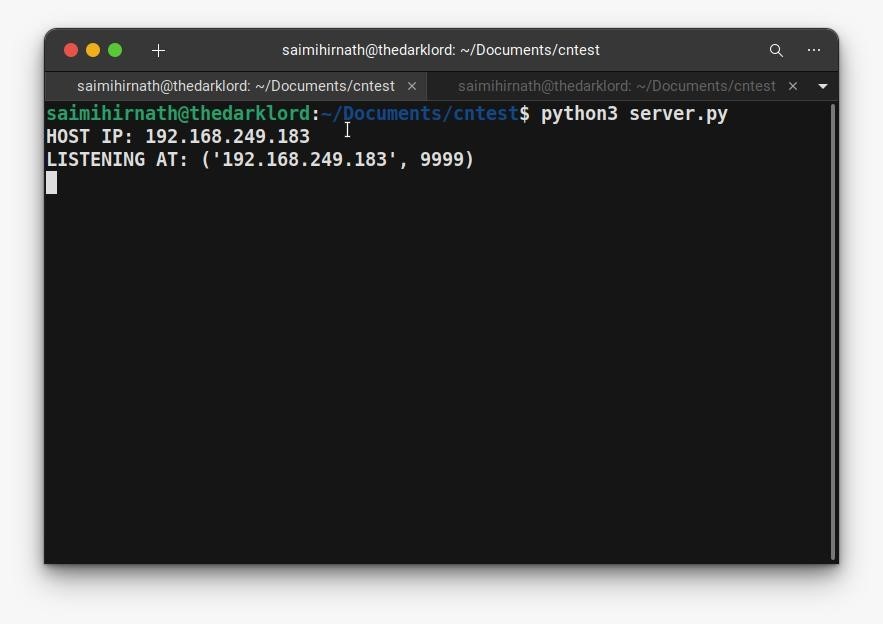
Before running the video streaming program, ensure that the required libraries are installed. Specifically, the code utilizes OpenCV for video processing. If OpenCV is not installed, it can be added using the following command:

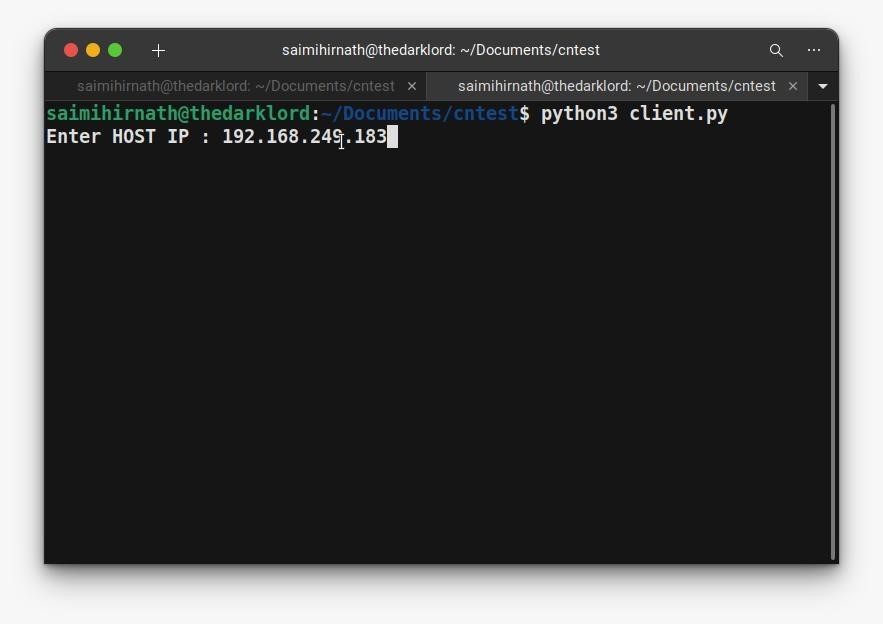


Once the dependencies are installed, the server and client scripts can be executed from separate terminals. The server script initializes a socket, binds it to the host IP and port, and starts listening for incoming connections. The client script prompts the user to enter the host IP and connects to the server.

## INPUT

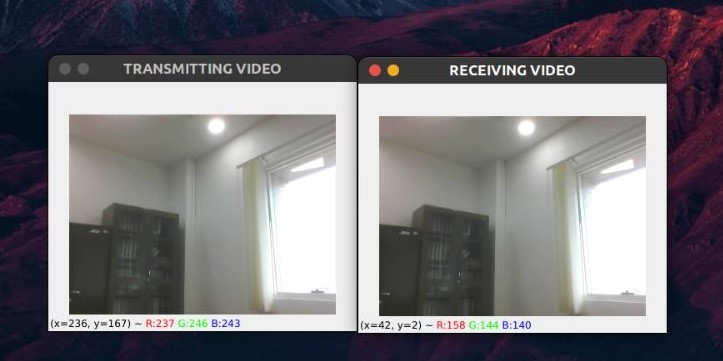
The program facilitates user interaction for setting up video streaming. The server script automatically obtains the host IP, and the client script prompts the user to enter the server's IP. The video streaming project allows customization of the video capture device (camera) and adjusts parameters such as frame width for transmission. These settings can be configured within the code to adapt to different hardware configurations.





## OUTPUT

The primary output of the video streaming project is the real-time transmission of video frames from the server to the client. The server, upon receiving a connection, initializes the video capture device, continuously captures frames, resizes them, serializes the data using pickle, packs it with size information, and transmits it to the client. The client, in turn, receives the data, unpacks and deserializes it, and displays the video frames. The output is a live video stream from the server's camera to the client's display. To terminate the streaming, the user can press the Enter key on the client side.



# LIMITATIONS

The video streaming project, while providing a functional foundation, has some inherent limitations, including potential challenges related to audio:

1. Bandwidth and Latency Issues: Both video and audio streams are susceptible to network bandwidth limitations and latency. In scenarios with low bandwidth or high latency, the synchronization between video and audio may be compromised, leading to audio lag or desynchronization with the corresponding video frames.
2. Single-Client Focus: Similar to video, the audio component of the project is designed for a single-client connection. Extending the project to support multiple clients concurrently would require additional considerations for managing audio resources and maintaining synchronization across multiple streams.
3. Audio Not Available: The current implementation focuses solely on video streaming and lacks support for audio transmission. Integrating audio would require additional coding considerations, such as synchronizing audio and video streams, and potential adjustments for real-time data synchronization.
4. Security Concerns: Similar to video, the project lacks robust security features for audio data. Implementing encryption and authentication mechanisms for audio streams is essential to ensure the privacy and integrity of transmitted audio content.
5. Audio Synchronization Challenges: Achieving precise synchronization between video and audio streams can be challenging. Variations in processing times for video and audio data may lead to a lack of synchronization, resulting in a noticeable mismatch between audio and video elements.
6. Dependence on External Audio Libraries: If the project incorporates external audio processing libraries, the dependencies and updates of these libraries could impact the overall stability and maintenance of the system.

# FUTURE WORK

The video streaming project lays the groundwork for several potential future enhancements, including:

1. Scalability and Optimization: Future work could focus on optimizing the video streaming algorithm to handle a larger number of concurrent clients and higher-resolution video feeds. This optimization would improve the overall performance and efficiency of the video streaming application, making it more adaptable to diverse network environments and varying hardware capabilities.
2. Security Enhancements: To address potential security concerns, future development might include the implementation of advanced encryption and authentication mechanisms. This would ensure secure and private video communication between the server and clients, making the system more resilient against unauthorized access or data interception.
3. Adaptive Streaming and Quality-of-Service (QoS): Further refinement of the project could involve implementing adaptive streaming techniques. This would enable the system to dynamically adjust video quality based on network conditions, ensuring a smooth streaming experience even in varying bandwidth scenarios. Additionally, incorporating Quality-of-Service (QoS) mechanisms could enhance the overall user experience by prioritizing critical video data during transmission.
4. Multi-Platform Compatibility: Expanding the project to support a variety of platforms and operating systems would contribute to its versatility. This could involve developing client applications for different devices and integrating server-side components compatible with various operating systems, making the video streaming solution accessible across a broader range of environments.
5. Interactive Features and Collaboration: Future work could explore the integration of interactive features, such as bidirectional communication between the server and clients. This enhancement would open possibilities for collaborative applications, where users on the client side can interact with the streaming content, enabling applications like remote education or virtual collaboration.
6. Cloud Integration: Integrating the video streaming project with cloud computing environments could provide scalability and resource optimization benefits. This could involve deploying the server component on cloud platforms and leveraging cloud services to enhance the overall reliability, scalability, and accessibility of the video streaming application.
7. Error Handling and Robustness: Enhancing error-handling mechanisms and ensuring robustness in challenging network conditions would be a valuable avenue for future work. Implementing features to recover gracefully from network disruptions or client-server communication issues would contribute to a more resilient and reliable video streaming solution.

These future directions aim to extend the functionality, performance, and usability of the video streaming project, making it more versatile and applicable to a broader range of scenarios and environments.

# CONCLUSION

In conclusion, the video streaming project has successfully demonstrated the foundational aspects of real-time video communication between a server and a client. The implemented solution captures video frames, efficiently serializes and transmits them, and displays the live video stream on the client side. The project underscores the significance of robust strategies for handling multimedia data over networks, particularly in the context of emerging applications and technologies.

The video streaming project serves as a practical illustration of networked multimedia communication and lays the groundwork for more advanced applications. The modular nature of the code allows for easy customization, supporting adaptability to different network configurations and hardware setups. The project's success in establishing a reliable and efficient video streaming connection showcases its potential for integration into diverse scenarios, from remote surveillance systems to video conferencing platforms.

Looking ahead, potential areas for further exploration include enhancing security features such as encryption to safeguard video transmissions and optimizing the system for scalability to accommodate multiple clients simultaneously. Additionally, future iterations could investigate the integration of advanced video processing techniques to enrich the streaming experience. This project, while providing a fundamental understanding of video streaming principles, opens avenues for continued research and development in the dynamic field of multimedia communication over networks.

# REFERENCES

1. https://[www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf](http://www.hpl.hp.com/techreports/2002/HPL-2002-260.pdf)
2. https://[www.learntechlib.org/p/6152/](http://www.learntechlib.org/p/6152/)
3. https://ieeexplore.ieee.org/abstract/document/911156