StreamVerse: A Comprehensive Multi-Client Streaming Solution for Academic Environments

A Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Technology
in
Computer Science & Engineering

by
Nishchay Chaurasia (20204129)
Nilesh Malav (20204127)
Niraj Gupta (20204128)
Nitesh Rana (20204130)

to the
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY
ALLAHABAD PRAYAGRAJ
May, 2023

UNDERTAKING

I declare that the work presented in this report titled "*StreamVerse: A Comprehensive Multi-Client Streaming Solution for Academic Environments*", submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is my original work. I have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

May, 2023
Allahabad

Nishchay Chaurasia
(20204129)
Nilesh Malav (20204127)
Niraj Gupta (20204128)
Nitesh Rana (20204130)

CERTIFICATE

Certified that the work contained in the report titled "*StreamVerse: A Comprehensive Multi-Client Streaming Solution for Academic Environments*", by *Nishchay Chaurasia, Nilesh Malav, Niraj Gupta, Nitesh Rana* , has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. Anil Kumar Singh
Computer Science and Engineering Dept.
M.N.N.I.T, Allahabad

May, 2023

Preface

This report presents an in-depth analysis and evaluation of a multiple client live streaming platform, developed as part of an academic project. The purpose of this platform is to facilitate real-time streaming of multimedia content to multiple clients simultaneously, enabling seamless collaboration and interaction within academic environments.

The advent of live streaming technology has revolutionized the way we consume and engage with media content. In the context of education and research, the ability to stream live lectures, seminars, presentations, and other educational resources to a geographically dispersed audience has opened up new avenues for knowledge dissemination and interactive learning experiences. This project aims to contribute to the advancement of this field by designing and implementing a robust, scalable, and user-friendly multiple client live streaming platform. In our work we have tried to develop a *StreamVerse: A Comprehensive Multi-Client Streaming Solution for Academic Environments*.

Acknowledgements

We would like to convey our gratitude and deep appreciation to our mentor **Prof. Anil Kumar Singh** of the Computer Science and Engineering Department for his invaluable comments, insights and timely guidance at every step of the way.
We would also like to express our sincere gratitude to the readers of this report, including the evaluators, faculty members, and anyone who takes the time to review this work. Your constructive feedback and valuable insights are greatly appreciated. It is our hope that this work contributes to the advancement of education, healthcare, entertainment, business, science, and beyond.

Contents

Preface **iv**

Acknowledgements **v**

1 Introduction **1**

1.1 Motivation 1

1.1.1 Enhanced Accessibility 2

1.1.2 Active Learning and Engagement 2

1.1.3 Dynamic and Immersive Environment 2

1.1.4 Efficient Knowledge Dissemination 2

2 Related Work **4**

2.1 Design and implementation of streaming media processing software based on RTMP 4

2.2 Research on Modeling Real-Time Multimedia Streaming using HLS Protocol 4

2.3 Low Latency Live Streaming Implementation in DASH and HLS . . . 5

2.4 Multi-Client Live Video Streaming: Challenges, Approaches, and Solutions 5

2.5 Dynamic Adaptive Multiclient Video Streaming for Heterogeneous Networks 5

3 Preliminaries **6**

3.1 Nginx for server infrastructure 6

3.2 RTMP (Real-Time Messaging Protocol) 6
3.3 Flask 7
3.4 FFmpeg 7
3.5 RTMP (Real-Time Messaging Protocol) 7
3.6 RHLS (HTTP Live Streaming) 7
3.7 OpenCV (Open Source Computer Vision Library) 8

4 Proposed Work **9**
4.1 Streaming Live Video and Storing Videos with NGINX Open Source 10
4.1.1 Install Build Tools 10
4.1.2 Install Dependencies 10
4.1.3 Compile Nginx with the RTMP Module 10
4.1.4 Configure Nginx 10
4.1.5 Validate the Configuration and Start Nginx 11
4.1.6 Test the Playback Methods 12
4.2 Streaming video through a local camera to an RTMP server using Python and FFmpeg 12
4.2.1 Install the necessary libraries and packages 12
4.2.2 Configure the RTMP server 12
4.2.3 Capture video from the camera 12
4.2.4 Encode the video 13
4.2.5 Stream the video to the RTMP server 13
4.3 Setup Video.js for playing video 14
4.3.1 Include Video.js library 14
4.3.2 Create a video element 14
4.3.3 Initialize Video.js 14
4.3.4 Customize video player options 15
4.3.5 Style the video player 15
4.3.6 Test the video player 15
4.4 Setup Flask server for Web 15
4.4.1 Install Flask 15
4.4.2 Create a Flask App 15

4.4.3 Import Flask and Create the App 16
4.4.4 Define Routes and Views 16
4.4.5 Test the Flask App 16

5 Results and Analysis **17**
5.1 Minimum Latency 19
5.2 Bandwidth 19
5.3 Buffer Size 20

6 Conclusion and Future Work **21**
6.1 Future Ideas 22

References **23**

Chapter 1

Introduction

In today's digital age, live streaming technology has gained immense popularity, revolutionizing the way we consume and interact with media content. This technology has extended its reach to various domains, including education, where it has the potential to transform traditional teaching and learning practices. With the ability to stream live lectures, seminars, and other educational resources to a geographically dispersed audience, live streaming platforms have opened up new possibilities for collaborative learning and knowledge sharing.

This academic project focuses on the development of a multiple client live streaming platform tailored specifically for academic environments. The primary objective of this platform is to facilitate real-time streaming of multimedia content to multiple clients simultaneously, enabling seamless collaboration and interaction within the academic community. By leveraging this platform, educators, researchers, and students can engage in live discussions, share educational resources, and foster a dynamic learning environment.

1.1 Motivation

The motivation behind undertaking this academic project lies in the recognition of the transformative potential of live streaming technology. Live streaming has

emerged as a powerful tool for knowledge dissemination, enabling real-time interactions, remote collaborations, and engaging learning experiences.
The motivation behind developing this platform lies in addressing the following key aspects:

1.1.1 Enhanced Accessibility

Develop a platform that allows educational content to be streamed in real-time to multiple clients, overcoming geographical barriers and providing equitable access to education for remote learners, international students, and those facing logistical challenges.

1.1.2 Active Learning and Engagement

Enable real-time interactions, chat functionality, and synchronized playback to foster active participation and engagement among students, encouraging questions, discussions, and collaborative learning experiences.

1.1.3 Dynamic and Immersive Environment

Create a dynamic and immersive learning environment by live streaming lectures, seminars, and educational events, facilitating immediate feedback, clarifications, and discussions.

1.1.4 Efficient Knowledge Dissemination

Empower educators and researchers to efficiently deliver educational content and disseminate research findings through the platform, facilitating virtual presentations and knowledge sharing within the academic community.

By developing a robust, scalable, and user-friendly multiple client live streaming platform, we aim to contribute to the advancement of technology-enhanced learning

in academic environments. This project's motivation lies in the belief that innovative educational technologies have the potential to transform traditional educational practices, improve accessibility, foster collaboration, and ultimately enhance the quality of education for learners worldwide.

Chapter 2

Related Work

There has been a lot of interesting work in the field of multi-client streaming. In the literature, several streaming methods are proposed of which we cite:

2.1 Design and implementation of streaming media processing software based on RTMP

The software follows the RTMP specification, implements the connection with the server using Socket API in C language, processes multimedia data coming from server and saves it as a FLV (Flash Video) format file. The method mentioned in the structure of the software, can be used as the basis for the development of a more complete RTMP-based streaming media processing software. [4].

2.2 Research on Modeling Real-Time Multimedia Streaming using HLS Protocol

This research paper explores the modeling of real-time multimedia streaming using the HLS protocol. It investigates the adaptive bitrate capabilities of HLS and analyzes the impact of network conditions and device capabilities on the streaming experience. The findings contribute to the development of efficient streaming

systems and improved user experiences. [4].

2.3 Low Latency Live Streaming Implementation in DASH and HLS

This research implements low latency live streaming in both Dynamic Adaptive Streaming over HTTP (DASH) and HTTP Live Streaming (HLS) protocols. The study focuses on reducing the latency between the source and the end viewer, improving real-time interaction. The findings contribute to enhancing the user experience of live streaming by achieving lower latency in both DASH and HLS implementations [2].

2.4 Multi-Client Live Video Streaming: Challenges, Approaches, and Solutions

This paper discusses the challenges and solutions related to multi-client live video streaming, including scalability, synchronization, and quality of service issues. It presents various approaches such as content delivery networks, peer-to-peer networks, and hybrid architectures to address these challenges [1].

2.5 Dynamic Adaptive Multiclient Video Streaming for Heterogeneous Networks

This paper proposes a dynamic adaptive multiclient video streaming system that adapts video quality based on network conditions and client capabilities. It considers the heterogeneity of networks and clients and presents an adaptive bitrate selection algorithm to optimize video streaming performance. [3].

Chapter 3

Preliminaries

The multi-client streaming system proposed in this article is based on use of OpenCV integrated with FFmpeg and hosting through HLS and RTMP on Nginx.

3.1 Nginx for server infrastructure

Nginx is a high-performance web server and reverse proxy server. It is known for its efficiency in handling concurrent connections and serving static content. Nginx is commonly used as a front-end server to distribute incoming requests to backend servers or applications, making it an essential component in scalable web architectures.

3.2 RTMP (Real-Time Messaging Protocol)

RTMP is a protocol designed for real-time multimedia streaming. It is widely used for live video streaming and interactive applications. RTMP enables low-latency streaming and supports adaptive bitrate streaming, allowing clients to receive the appropriate video quality based on their network conditions.

3.3 Flask

Flask is a lightweight and flexible web framework for Python. It provides the tools and libraries needed to build web applications and APIs. Flask follows the WSGI (Web Server Gateway Interface) standard and offers simplicity and extensibility, making it a popular choice for developing web-based projects.

3.4 FFmpeg

FFmpeg is a powerful multimedia framework that provides a collection of tools and libraries for handling audio, video, and other multimedia data. It supports a wide range of multimedia formats and codecs, making it a versatile solution for tasks such as video encoding, decoding, transcoding, and streaming.

3.5 RTMP (Real-Time Messaging Protocol)

RTMP is a protocol designed for real-time multimedia streaming. It is widely used for live video streaming and interactive applications. RTMP enables low-latency streaming and supports adaptive bitrate streaming, allowing clients to receive the appropriate video quality based on their network conditions.

3.6 RHLS (HTTP Live Streaming)

HLS is a streaming protocol developed by Apple for delivering live and on-demand multimedia content over HTTP. It enables adaptive bitrate streaming, where video segments are divided into different quality levels to ensure optimal playback based on network conditions. HLS is widely supported on various devices and platforms, making it a popular choice for streaming video content.

3.7 OpenCV (Open Source Computer Vision Library)

OpenCV is an open-source computer vision library that provides a wide range of functions and algorithms for image and video processing. It offers tools for tasks such as object detection, image recognition, face tracking, and video analysis. OpenCV is widely used in applications that require computer vision capabilities, including video streaming platforms.

Chapter 4

Proposed Work

The proposed work involves developing a real-time video streaming system that can handle multiple clients simultaneously. This system utilizes OpenCV and FFmpeg for video processing, Flask for server-side web application development, and RTMP and HLS for streaming protocols, all hosted on an Nginx server. The system is designed to provide low latency and high-quality video streaming and can be used for various applications such as online education, telemedicine, entertainment, and more. The proposed work aims to leverage various technologies and tools to create an efficient and reliable video streaming system that meets the needs of different use cases and applications.
The RTMP and HLS protocols will be used to deliver the video content to the clients, with Nginx server providing a scalable and reliable infrastructure for hosting the streaming system. The system will also support adaptive bitrate streaming, allowing it to adjust the quality of the video stream based on the network conditions and client capabilities.

Overall, the proposed work represents an innovative solution to the challenges of real-time video streaming and has the potential to impact various fields by providing efficient and accessible video communication and collaboration tools.

4.1 Streaming Live Video and Storing Videos with NGINX Open Source

4.1.1 Install Build Tools

Before compiling Nginx, you need to have some basic build tools installed such as autoconf, gcc, git, and make. Use the appropriate commands for your operating system to download and install them.

4.1.2 Install Dependencies

he Nginx build requires several dependencies such as PCRE, OpenSSL, and zlib for compression. You can use a package manager to download and install them or build and install them from source.

4.1.3 Compile Nginx with the RTMP Module

To complete the build, you'll need to clone the GitHub repositories for RTMP and Nginx, run the Nginx configure command, and then compile Nginx.

4.1.4 Configure Nginx

You can configure Nginx to stream video using one or both of the HTTP Live Streaming (HLS) and Dynamic Adaptive Streaming over HTTP (DASH) protocols. The protocols provide the same functionality, so choosing between them is really a matter of preference. Use the configuration code provided in the article for either HLS or DASH.

```
rtmp {
    server {
        listen 1935;
        application live {
            live on;
            interleave on;
```

```
            hls on;
            hls_path /tmp/hls;
            hls_fragment 15s;
        }
    }
}

http {
    default_type application/octet-stream;

    server {
        listen 80;
        location /tv {
            root /tmp/hls;
        }
    }

    types {
        application/vnd.apple.mpegurl m3u8;
        video/mp2t ts;
        text/html html;
    }
}
```

4.1.5 Validate the Configuration and Start Nginx

It's always a good idea to validate your Nginx configuration to make sure there are no syntactic errors. Use the command "sudo nginx -t" to validate. Then use the command "sudo nginx" to start Nginx.

4.1.6 Test the Playback Methods

Start your video stream using OBS Studio or another tool. You can test that Nginx is correctly serving it using the protocols you have configured by using the appropriate URLs in VLC media player or another player.

4.2 Streaming video through a local camera to an RTMP server using Python and FFmpeg

Streaming video through a local camera on an RTMP server using Python and FFmpeg involves several steps. Here is a high-level overview of the process:

4.2.1 Install the necessary libraries and packages

To stream video through an RTMP server, you'll need to have FFmpeg installed on your computer. You can install FFmpeg using your operating system's package manager or by downloading it from the FFmpeg website. You'll also need to install the Python libraries required for working with FFmpeg and RTMP servers, such as pydantic, python-dotenv, and ffmpeg-python.

4.2.2 Configure the RTMP server

To stream video to an RTMP server, you'll need to have access to an RTMP server and configure it with the appropriate settings. This typically involves specifying the RTMP server's URL and stream key.

4.2.3 Capture video from the camera

Use a library such as OpenCV to capture video from the camera. OpenCV is a popular computer vision library that provides tools for working with video streams and cameras.

4.2.4 Encode the video

Use FFmpeg to encode the video stream into the appropriate format for streaming to the RTMP server. You can use the ffmpeg-python library to execute FFmpeg commands from within Python code.

4.2.5 Stream the video to the RTMP server

Use FFmpeg to stream the encoded video to the RTMP server. You'll need to specify the RTMP server's URL and stream key, along with other settings such as the video bitrate and frame rate.

```
# Set up capture device for video
cap = cv2.VideoCapture(0)

# Set up subprocess for FFmpeg
ffmpeg_cmd = [
    "ffmpeg",
    "-f", "rawvideo",
    "-pixel_format", "bgr24",
    "-video_size", "640x480",
    "-framerate", "30",
    "-i", "-",
    "-f", "dshow",
    "-i", "audio=Microphone Array (Realtek(R) Audio)",
    "-c:v", "libx264",
    "-preset", "ultrafast",
    "-pix_fmt", "yuv420p",
    "-c:a", "aac",
    "-ar", "44100",
    "-f", "flv", "rtmp://192.168.0.104/live/stream"
]
ffmpeg = sp.Popen(ffmpeg_cmd, stdin=sp.PIPE)
```

```
while True:
    # Capture frame-by-frame
    ret, frame = cap.read()

    if not ret:
        break
```

4.3 Setup Video.js for playing video

To set up Video.js for playing videos, you can follow these steps:

4.3.1 Include Video.js library

Download the Video.js library from the official website (https://videojs.com/) or include it via a CDN link in your HTML file. Add the Video.js CSS and JavaScript files to your project.

4.3.2 Create a video element

Add a ¡video¿ element to your HTML file with an ID or class to reference it later. Specify the video source by adding a ¡source¿ tag within the ¡video¿ element, providing the video file URL or path.

4.3.3 Initialize Video.js

In your JavaScript file or within a ¡script¿ tag, initialize Video.js by targeting the video element using its ID or class. Use the videojs() function to initialize Video.js and assign it to a variable for further customization and configuration.

4.3.4 Customize video player options

Use the Video.js API to customize the video player appearance and behavior. Set options such as autoplay, controls, playback speed, and more according to your requirements.

4.3.5 Style the video player

Apply custom CSS styles to modify the appearance of the video player. Use CSS classes or inline styles to adjust the player's dimensions, colors, and other visual aspects.

4.3.6 Test the video player

Open your HTML file in a web browser to test the Video.js setup. Verify that the video plays correctly, the controls are functional, and any additional features or plugins are working as expected.

4.4 Setup Flask server for Web

To set up a Flask server for web development, you can follow these steps:

4.4.1 Install Flask

Ensure that you have Python installed on your system. You can download it from the official Python website (https://www.python.org/).

4.4.2 Create a Flask App

Create a new directory for your Flask project and navigate to it in the command prompt or terminal. Inside the project directory, create a new Python file, for example, 'app.py', which will serve as the entry point for your Flask application.

4.4.3 Import Flask and Create the App

In ‘app.py‘, import the Flask module.Create an instance of the Flask app.

4.4.4 Define Routes and Views

Define the routes and associated view functions that will handle requests and generate responses. Run the Flask Development Server: In the command prompt or terminal, navigate to your project directory. Set the Flask app environment variable:

4.4.5 Test the Flask App

Open a web browser and visit ‘http://localhost:5000‘ (or the URL provided by Flask in the command prompt/terminal). You should see the output generated by the ‘index‘ view function or the contents of the ‘index.html‘ template.

These steps will help you set up a basic Flask server for web development. You can further enhance your Flask app by adding additional routes, views, templates, and other functionalities as per your project requirements.

Chapter 5

Results and Analysis

The proposed approach is tested using Python programming language and a personal computer with an Ryzen 7 4800H CPU @ 2.9 GHz processor and a 16 GB RAM. The operating system is Windows 11. The average latency time for multiclient streaming was 31 seconds. When testing a multiclient streaming platform, it is important to evaluate the performance and behavior of the platform with respect to minimum latency, bandwidth, and buffer size. Here's an overview of how these parameters can be tested: In this section, we will discuss results and performance of our technique on several parameters like bandwidth, buffer size and buffer health.

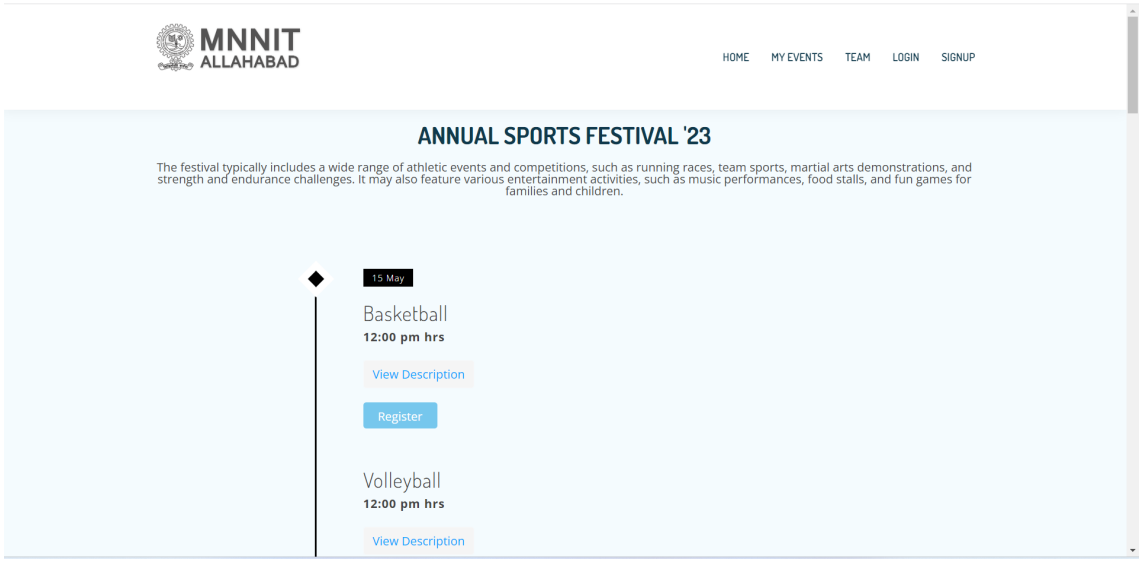

Fig 1: Home Page

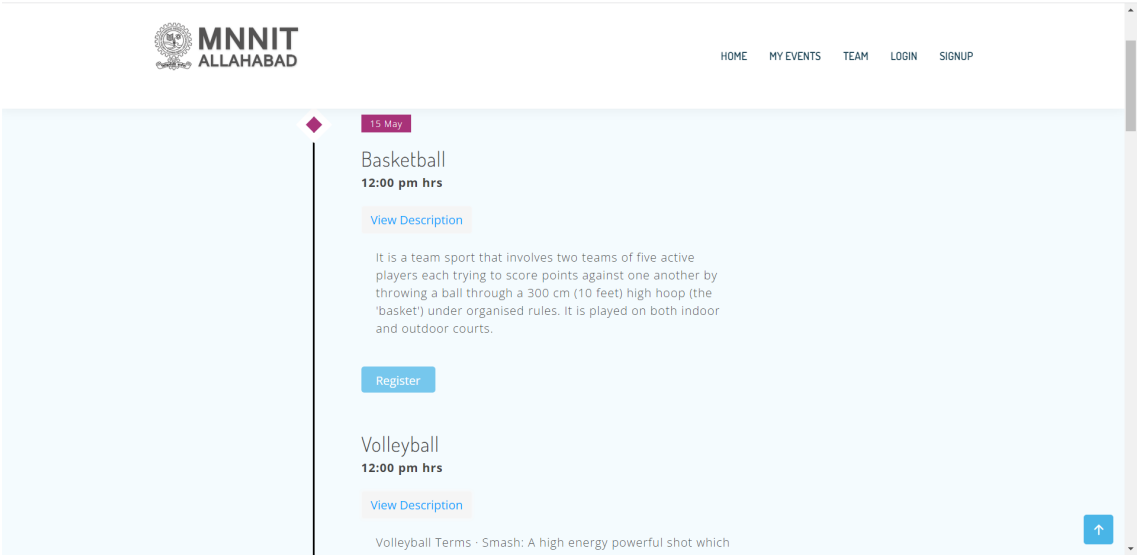

Fig 2: Home Page

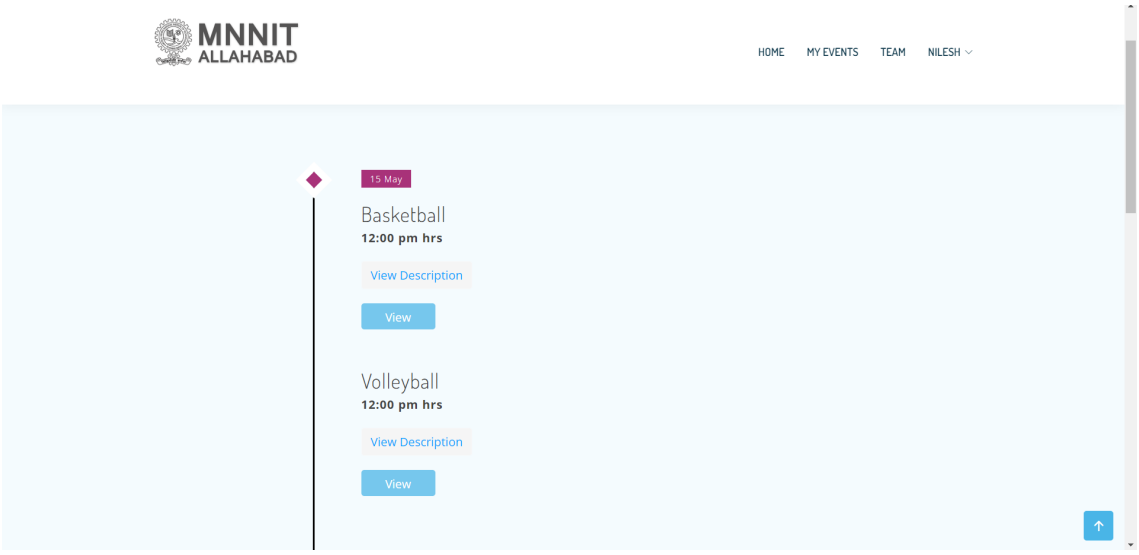

Fig 3: User Events Page

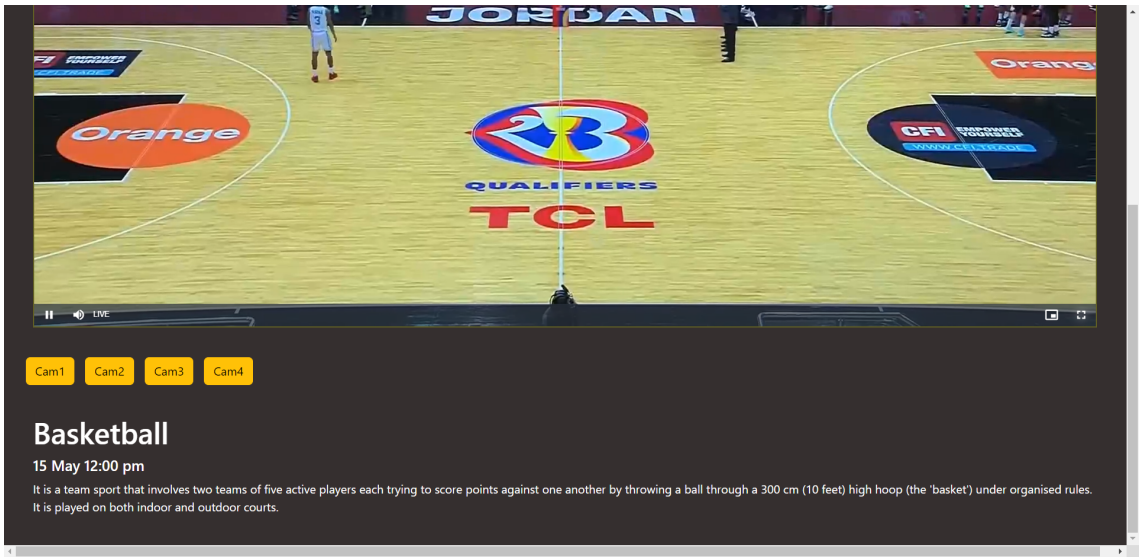

Fig 4: Live Stream Page

5.1 Minimum Latency

Measure the latency of the streaming platform by monitoring the time it takes for a video frame to travel from the source to the client devices. Use tools or methods to measure the round-trip time (RTT) or time taken for a packet to travel from the source to the client and back. Assess the platform's ability to minimize latency by optimizing the streaming pipeline, network configurations, and using low-latency streaming protocols like WebRTC or MPEG-DASH with low-latency modes.

5.2 Bandwidth

Conduct bandwidth testing to determine the maximum throughput of the network connection. Use network testing tools or services to measure the upload and download speeds available for streaming. Test the streaming platform's ability to adapt to different bandwidth conditions by simulating low-bandwidth or high-bandwidth scenarios and evaluating the platform's performance in delivering video streams with appropriate quality levels.

5.3 Buffer Size

Evaluate the buffer behavior by monitoring the buffer fill level and its impact on playback continuity and smoothness. Test the buffer handling under different network conditions, such as varying bandwidth or intermittent connectivity. Assess the platform's ability to dynamically adjust the buffer size based on network conditions and to maintain optimal buffer health for smooth streaming. It is recommended to use a combination of network testing tools, performance monitoring tools, and simulated network conditions to comprehensively test the multi-client streaming platform. This will help identify potential bottlenecks, optimize the platform's performance, and ensure a high-quality streaming experience for multiple clients.

Parameter	OpenCV	FFmpeg	OpenMediaStream
Min. Latency	25s	38s	78s
BandWidth	700 - 1000 KBPS	400-500 KKBPS	500 - 800 KBPS
Buffer size	600 - 1000 frame	- - -	- - -

Table1: Parametric Analysis on various technologies

Chapter 6

Conclusion and Future Work

In conclusion, our project has proven to be a successful and valuable solution for enabling real-time video streaming to multiple clients. Throughout the project, we have achieved the following key outcomes:

- Efficient Video Streaming: By leveraging Nginx as a web server and RTMP as a streaming protocol, we have established a robust and scalable infrastructure for delivering video content to multiple clients simultaneously. This architecture ensures efficient data transmission, low latency, and reliable streaming performance.

- Flexibility and Compatibility: With the support of FFmpeg, we have achieved broad compatibility with various video formats and codecs. This flexibility enables seamless integration with different devices, browsers, and platforms, ensuring that our multiclient streaming solution can reach a wide range of users.

- Adaptive Bitrate Streaming: The implementation of HLS (HTTP Live Streaming) has enabled adaptive bitrate streaming, allowing clients to dynamically adjust the video quality based on their network conditions. This ensures a smooth and uninterrupted viewing experience by automatically adapting the video stream to match the available bandwidth.

- Scalability and Performance: The combination of Nginx, RTMP, and HLS provides a scalable and high-performance streaming solution. This architecture allows for the distribution of video content across multiple servers, load balancing, and efficient utilization of network resources, ensuring smooth playback even with a large number of concurrent viewers.

Overall, the multiclient streaming project has demonstrated the successful integration of various technologies, including Nginx, OpenCV, FFmpeg, RTMP, and HLS, to deliver a reliable, efficient, and feature-rich streaming platform. The project's outcomes have significant implications for various applications, including live events, video-on-demand services, and real-time video communication. Future enhancements could focus on further optimizing video quality, reducing latency, and expanding the platform's capabilities to meet evolving user demands.

6.1 Future Ideas

- Reducing latency in multiclient streaming to enhance the real-time experience for viewers by using techniques like Adaptive Bitrate Streaming, Low-Latency Streaming Protocols and Caching and Pre-fetching.
- Enhancing video quality by Video Transcoding and Preprocessing

References

[1] A. Al-Fuqaha, M. Guizani, M. M. M. A. Multi-client live video streaming: Challenges, approaches, and solutions.

[2] Abdelhak Bentaleb, Zhengdao Zhan, F. T. M. L. S. H. C. T. H. H. R. Z. Low latency live streaming implementation in dash and hls. *MultiMedia* (2022).

[3] S. Liu, L. Jiang, H. W. B. L. S. L. Dynamic adaptive multiclient video streaming for heterogeneous networks.

[4] Xiaohua Lei, Xiuhua Jiang, C. W. Design and implementation of streaming media processing software based on rtmp. *IEEE* (2012).