Collaborative 360-Degree Video Streaming: A Multi-User Synchronization Solution with Socket.IO

Anderson Augusto Simiscuka
Dublin City University
Dublin, Ireland
andersonaugusto.simiscuka@dcu.ie

Gianluca Fadda
Università degli Studi di Cagliari
Cagliari, Italy
gianluca.fadda@unica.it

Maurizio Murroni
Università degli Studi di Cagliari
Cagliari, Italy
maurizio.murroni@unica.it

Gabriel-Miro Muntean
Dublin City University
Dublin, Ireland
gabriel.muntean@dcu.ie

***Abstract*—The rapid development of 360-degree video technology has transformed the way users engage with video content. Despite these advancements, synchronizing 360-degree video playback across multiple users remains a significant challenge. While libraries and plugins such as videojs-xr support playback of 360-degree videos on both browsers and headsets, they do not offer a unified, synchronized viewing experience for multiple users viewing simultaneously. This article presents a novel solution for real-time synchronized playback of 360-degree videos across various client devices. The proposed system employs a server architecture built on Node.js and Express.js, along with WebSocket communication managed by Socket.IO, to ensure low-latency state synchronization across all connected clients. By incorporating ngrok for seamless internet access, the solution facilitates real-time interaction without the need for complex network configurations. This approach not only addresses the limitations of existing technologies but also greatly enhances the viewing experience in applications such as education, entertainment, and beyond. Performance evaluation results demonstrate the system's ability to maintain suitable throughput levels and efficient CPU consumption under different scenarios, including setups with 2, 8, and 16 users.**

***Index Terms*—VR, 360-degree video, Node.js, ngrok, Socket.IO, videojs-xr**

I. Introduction

In the rapidly evolving field of immersive media, 360-degree video content represents a significant technological innovation for entertainment, training, and education. Indeed, a growing number of video-sharing platforms (i.e. Youtube, Facebook, Vimeo, etc.) now have their dedicated section to support publishing and viewing immersive videos, with playback on their website, mobile apps or VR devices [1].

Despite these advancements, several critical factors must still be considered when addressing the synchronization of 360-degree video playback across multiple users. First, achieving accurate real-time synchronization through heterogeneous networks remains a significant challenge [2], [3]. Moreover, effectively handling jitter and packet loss in group streaming scenarios is crucial to maintaining smooth playback and preventing disruptions [4]. Another important aspect is the absence of standardized Quality of Experience (QoE) metrics tailored specifically for synchronized multi-user virtual reality (VR) environments [5]. Furthermore, aligning playback timelines for users who join the session asynchronously presents a unique challenge that requires careful coordination [6]. Finally, dynamic real-time coordination of multiple viewports is essential for delivering an immersive and synchronized experience to all users [7]. Addressing these challenges is crucial for the seamless integration of 360-degree video in multi-user VR applications.

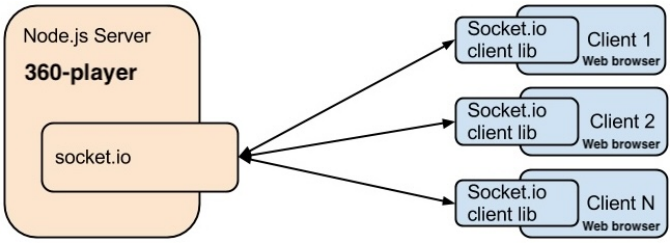

Fig. 1. Solution Architecture

Existing research has predominantly relied on live streaming technologies to synchronize the playback of 360-degree videos. Therefore, inspired by recent advancements in WebSocket-based communication, this work explores an alternative approach for synchronizing 360-degree video playback using Socket.IO. Unlike traditional streaming methods, Socket.IO facilitates low-latency bidirectional communication between clients and servers, making it an ideal candidate for real-time state synchronization [8].

This paper also aims to enhance the videojs-xr JavaScript library by incorporating broadcasting capabilities for synchronized 360-degree video playback. The proposed solution employs Node.js to host the video streaming server; Socket.IO for real-time communication and synchronization among multiple clients; and Ngrok to expose the locally hosted server to the internet without requiring complex network configurations.

This rest of this paper is organized as follows: section II reviews related works while section III discusses the technical design of the proposed solution. Section IV presents test

results under different network conditions and performance analysis. Section V presents conclusions and directions for future research.

II. Related Works

A valuable technology that enables the rendering of 360-degree videos on web pages is the videojs-xr Javascript library [9]. This plugin extends the functionality of the Video.js player to support 360-degree videos and it is compatible with VR devices. By integrating 360-degree video rendering into web pages, videojs-xr allows users to interactively explore the full 360-degree field of view by clicking and dragging, offering them a high degree of control over their viewing experience. This interactivity is an essential feature for creating immersive experiences, as it empowers viewers to adjust their perspectives and fully engage with the content. Moreover, videojs-xr supports playback on various VR headsets, such as Meta Quest devices and HTC Vive, by detecting when these devices are connected to a PC either via USB cables or wirelessly. Once connected, content from the web browser is processed through WebXR and played directly in the VR headset, offering an immersive experience.

For video streaming and playback, a common approach involves using tools such as FFmpeg for the compression, encoding, and packaging of 360-degree videos into streaming media formats. These encoded video streams are then transmitted using RTMP (Real-Time Messaging Protocol) and served through Nginx-based servers, which enables synchronized playback between multiple users [10]. Although this method is effective in many cases, it involves several complex processes, including video encoding, network configuration, and server setup, all of which can introduce performance bottlenecks. The substantial size of 360-degree video streams further complicates these tasks, contributing to issues such as increased latency and reduced scalability. These challenges are particularly detrimental in real-time, interactive use cases such as virtual tours, live entertainment events, and collaborative training environments, where low-latency seamless video delivery is crucial for maintaining an engaging user experience [11]–[13].

In mobile streaming scenarios, additional challenges arise, including the need to scale synchronization across multiple base stations in mobile networks [14]. Variations in the propagation delay during cellular streaming [15] can also introduce synchronization issues, negatively impacting the quality of the user experience. Furthermore, the challenge of synchronizing personalized viewports for individual users adds complexity to maintaining an immersive and cohesive experience across devices [16]. Another concern is the storage overhead required for maintaining multiple pre-rendered video variants, which can strain system resources and hinder efficient content delivery [17]. Moreover, the absence of standardized control protocols for real-time synchronization presents a significant hurdle for ensuring consistent playback quality across diverse network conditions and devices [18]. These challenges collectively emphasize the need for innovative solutions to enhance the scalability, efficiency, and synchronization of 360-degree video streaming in dynamic, real-time applications.

III. Technical Description

This work focuses on the development of a real-time video synchronization solution utilizing a Node.js server enhanced with Express.js and Socket.IO, as seen on Fig. 1. Node.js, a robust open-source, cross-platform runtime environment built on the Chrome V8 JavaScript engine, supports the execution of JavaScript on the server side. Its event-driven, non-blocking I/O model offers superior efficiency and performance, particularly in data-intensive, real-time applications. Express.js serves as the foundational framework for managing HTTP requests. By streamlining routing and middleware functionality, it facilitates efficient responses to client requests and the delivery of static files essential for initializing the video player and user interface. The integration of Socket.IO extends the server's capabilities by enabling real-time, bidirectional communication between the server and connected clients. This capability is pivotal to achieving the project's real-time synchronization objectives. Socket.IO ensures that any changes to the video playback state—such as play, pause, or seek operations—are instantly propagated to all clients. As a result, all connected clients view synchronized video content. The tightly coupled system, comprising Node.js, Express.js, and Socket.IO, leverages WebSocket connections to support this synchronization functionality seamlessly.

Socket.IO, a versatile JavaScript library, is specifically designed to enable real-time, bidirectional communication in web applications. Its ability to dynamically transmit data between connected clients makes it ideal for interactive applications requiring immediate state updates. In this project, Socket.IO is integral for synchronizing video playback across multiple clients. The server manages the playback state by broadcasting updates to all connected clients whenever a change occurs. For instance, if the playback position is modified or the video switches between play and pause states, these changes are relayed in real time using the Socket.IO infrastructure. This ensures that all viewers experience synchronized content playback.

The system workflow involves importing the Express framework, HTTP module, and Socket.IO library. An HTTP server is created using the Express application and enhanced with Socket.IO for handling WebSocket connections. Key state variables are declared to manage video synchronization: 'currentTime' tracks the current playback time, 'videoState' indicates whether playback is paused or playing, and 'hostID' identifies the host client. When a new client connects, the server immediately emits an event to share the current playback state, including 'currentTime' and 'videoState'. The first client to connect is designated as the host, responsible for controlling the playback state. The host's video provides updates for 'currentTime' and 'videoState', which are then broadcast to all connected clients to ensure synchronized playback across devices. Fig. 2 illustrates the client view, while Fig. 3 displays the host/administrator view of the player. In this setup, the

host controls the video playback, while clients are limited to controlling the volume and player size.

The snipped below demonstrates how the server assigns a “host” client to control video playback in the real-time synchronization system and ensures that the host’s playback state is shared with other clients.

```
// Assign a host to control playback :
if (!hostId) {
    hostId = socket.id;
    socket.emit('host', true);
} else {
    socket.emit('host', false);
}
socket.on('sync', (data) => {
    if (socket.id === hostId) {
currentTime = data.currentTime;
videoState = data.state;
socket.broadcast.emit('sync', data);
}
});
```

The next step involves creating an HTML page that loads the necessary libraries, including video.js, videojs-xr, and socket.io. The page then establishes a WebSocket connection with the server using the following line: const socket = io(‘https://f02c-54-217-106-36.ngrok-free.app’);. This URL represents the ngrok address of the local server, allowing it to be accessed over the internet. The client listens for the ’host’ event from the server. If the event indicates that the client is the host, video playback controls, such as play/pause buttons and a progress bar, will be displayed. If the client is not the host, these controls will remain hidden. The client also listens for the ’sync’ event, which contains the current playback time and state from the host. Upon receiving this event, the client will synchronize its video playback time and state to match the host’s, ensuring a consistent viewing experience across all clients. The specific code is as follows:

```
socket.on('sync', (data) => {
    player.currentTime(data.currentTime);
    if (data.state === 'playing') {
        player.play();
    } else {
        player.pause(); }
});
```

As outlined in the design, when a new client connects, their video playback will synchronize with the host client’s current playback time. If the host client clicks play or pause, all other clients will also play or pause the video accordingly. Similarly, when the host client adjusts the progress bar, the playback time on the other clients will update to match the host’s, ensuring synchronized playback of the 360-degree video.

To facilitate testing, the local server needs to be exposed to the public internet using ngrok. This can be done by running the command ‘ngrok http http://localhost:3001’.

Fig. 2. Web 360-player - Client

Fig. 3. Web 360-player - Host

Once the ngrok-provided link is clicked, all clients will immediately skip to the host’s video playback time. If the host pauses the video, all clients will pause as well. This demonstrates the feasibility of the design.

IV. Results

In the testing phase, this study uses Wireshark to measure throughput and CPU consumption under varying client loads.

The server used in the tests is equipped with an AMD Ryzen 7 7840H processor with Radeon 780M Graphics, operating at 3.80 GHz, and has 16.0 GB of RAM. It runs a 64-bit operating system with an x64-based processor, specifically Windows 11. The tests were conducted in a Wi-Fi network environment with the SSID CMCC-JRW4-5G, using a Xiaomi router with Wi-Fi 6 (802.11ax) protocol. The client device is equipped with a MediaTek Inc. RZ616 Wi-Fi 6E 160MHz wireless network module, operating on the 5 GHz band. The actual link speed is 306/216 Mbps (download/upload).

In Figs. 4, 5, and 6, the Y-axis of each graph represents throughput in bits, while the X-axis represents time in seconds. These charts visually illustrate the system’s throughput performance under different client loads.

In Fig. 4, when two clients are connected, the throughput starts to increase around the 30-second mazrk, peaking at approximately 2.5 Mbits. A decrease around the 40-second mark indicates that the data transmission phase has concluded. Despite some fluctuations, the throughput remains relatively

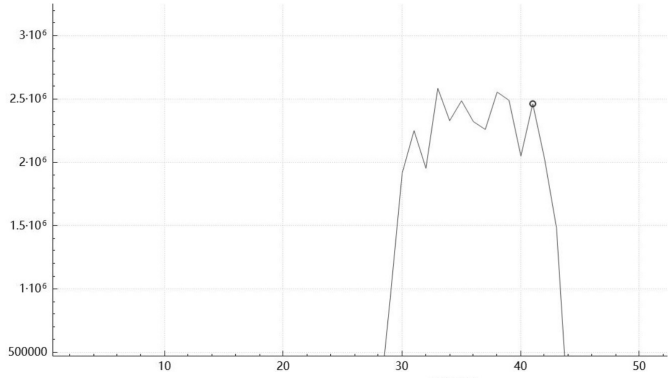

Fig. 4. 2 clients - Throughput

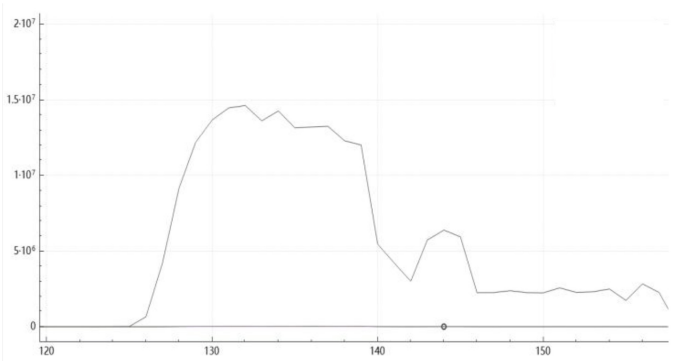

Fig. 5. 8 clients - Throughput

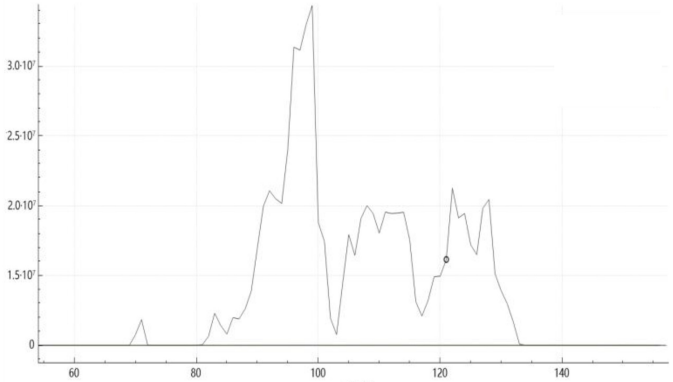

Fig. 6. 16 clients - Throughput

stable with minimal variation. This demonstrates that the server can handle two clients without placing undue stress on its resources.

In Fig. 5, with eight clients connected, throughput increases again, reaching approximately 15 Mbits around the 130-second mark. A significant drop in throughput occurs around 140 seconds, followed by increased fluctuations, although the throughput stabilizes shortly after.

In Fig. 6, when 16 clients are connected, throughput peaks around the 100-second mark, exceeding 30 Mbits. However, significant volatility and instability are observed, with increased fluctuations. This suggests that the current server may

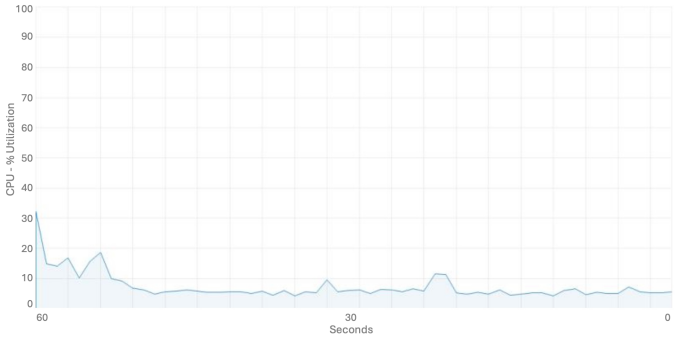

Fig. 7. 2 clients - CPU Consumption

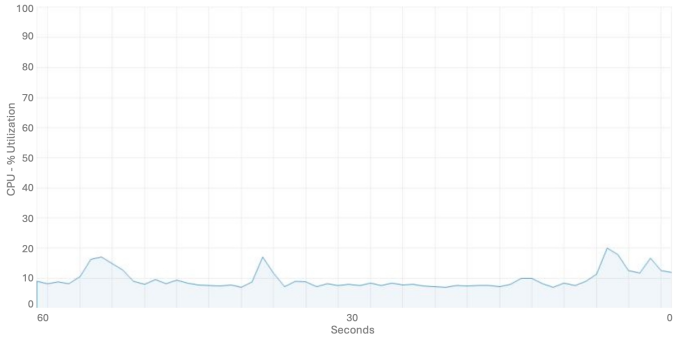

Fig. 8. 8 clients - CPU Consumption

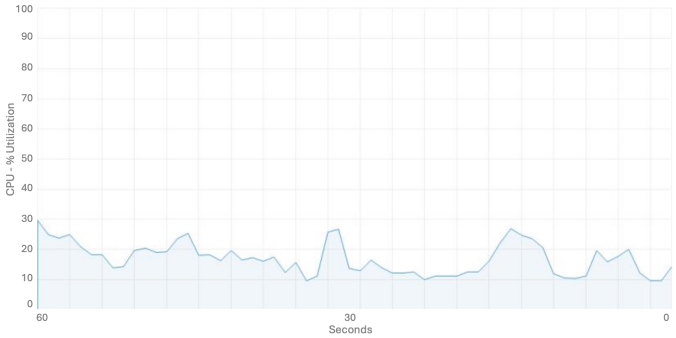

Fig. 9. 16 clients - CPU Consumption

have difficulty supporting more than 16 clients simultaneously, highlighting the need for enhanced network scalability and upgraded equipment with greater resources to accommodate a larger number of concurrent users, if necessary.

In addition to throughput, CPU consumption is analyzed to assess overall system performance and identify potential bottlenecks.

Fig. 7 shows the CPU consumption when two clients are connected, which remains consistently low, averaging around 5%. This low utilization indicates that the server has ample processing capacity to handle the minimal load imposed by two clients. The system operates efficiently, with most of its processing power remaining unused.

Fig. 8 demonstrates that as the number of clients increases to eight, CPU consumption rises slightly, averaging about

11%. This moderate increase reflects the additional processing required to manage the increased number of client connections. However, CPU consumption remains relatively low, suggesting that the server still has considerable processing headroom.

Fig. 9 displays the CPU consumption when 16 clients are connected, with usage averaging around 18%. Although this is a noticeable increase, it is still far below the server's maximum CPU capacity. Despite the increase in client numbers and the corresponding rise in throughput, CPU utilization remains low across all scenarios, indicating that processing power is not a bottleneck for the server. To handle a larger number of connected clients, optimizing network conditions may be necessary.

V. Conclusions and Future Work

This paper described and evaluated a solution for synchronizing 360-degree video playback across multiple clients. The aim was to address the challenge of real-time synchronization of video playback on different devices using the videojs-xr player. By integrating a Node.js server with Express.js and Socket.IO, effective real-time bidirectional communication between the server and clients was implemented. To assess the scalability and stability of this solution, throughput and CPU consumption were analyzed under varying numbers of connected devices.

Tests were conducted in a controlled environment with limited network conditions and a narrow range of client devices. The study involved a maximum of 16 clients, which, while sufficient for preliminary evaluations, may not fully capture the challenges associated with scaling the solution to a larger number of clients, which is the main focus of future research.

VI. Acknowledgement

This work was supported by Research Ireland via the Frontiers Projects grant 21/FFP-P/10244 (FRADIS) and Research Centres grant 12/RC/2289_P2 (INSIGHT), and by the European Union (EU) Horizon Europe grant 101135637 (HEAT Project). The code provided by Zhongyu Yuan is also gratefully acknowledged.

References

[1] Y.-C. Su and K. Grauman, "Learning Compressible 360° Video Isomers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 8, pp. 2697–2709, 2021.

[2] A. A. Simiscuka, M. A. Togou, R. Verma, M. Zorrilla, N. E. O'Connor, and G.-M. Muntean, "An Evaluation of 360° Video and Audio Quality in an Artistic-Oriented Platform," in *Proc. IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2022, pp. 1–5.

[3] M. Wang, X. Chen, X. Yang, and S. Peng, "CoLive: Edge-Assisted Clustered Learning Framework for Viewport Prediction in 360 Live Streaming," *IEEE Transactions on Multimedia*, 2023.

[4] M. K. Sharma, I. Farhat, C. F. Liu, and N. Sehad, "Real-time immersive aerial video streaming: A comprehensive survey, benchmarking, and open challenges," *IEEE Open Journal of the Communications Society*, 2024.

[5] A. Yaqoob, T. Bi, and G.-M. Muntean, "A Survey on Adaptive 360 Video Streaming: Solutions, Challenges and Opportunities," *IEEE Communications Surveys Tutorials*, vol. 22, no. 4, pp. 2801–2839, 2020.

[6] R. Verma, A. A. Simiscuka, M. A. Togou, M. Zorrilla, and G.-M. Muntean, "A Live Adaptive Streaming Solution for Enhancing Quality of Experience in Co-Created Opera," *IEEE Transactions on Broadcasting*, pp. 1–12, 2025.

[7] H. Sami, A. Hammoud, M. Arafeh, M. Wazzeh, S. Arisdakessian, M. Chahoud, O. Wehbi, M. Ajaj, A. Mourad, H. Otrok, O. Abdel Wahab, R. Mizouni, J. Bentahar, C. Talhi, Z. Dziong, E. Damiani, and M. Guizani, "The Metaverse: Survey, Trends, Novel Pipeline Ecosystem Future Directions," *IEEE Communications Surveys Tutorials*, vol. 26, no. 4, pp. 2914–2960, 2024.

[8] T. Kondengar, M. Ba, S. E. Zabolo, A. M. Kachallah, S. Ouya, I. Dioum, and E. G. Gbetie, "Secure platform for remote medical learning using WebRTC and facial recognition authentication," in *Proc. IEEE International Conference and Expo on Real Time Communications at IIT (RTC)*, 2024, pp. 31–38.

[9] T. Deppisch, "videojs-xr: Webxr plugin for video.js," https://github.com/thomasdeppisch/videojs-xr, 2023, accessed: 2025-03-23.

[10] C. Liu, Q. Mao, Y. Wang, J. Liu, and Y. Wang, "Analysis and research of campus live broadcasting system based on srs and ffmpeg," in *Proc. International Conference on Electronic Information Technology and Computer Engineering*. Association for Computing Machinery, 2024, p. 1306–1310.

[11] D. Wuebben, J. L. Rubio-Tamayo, M. Gertrudix Barrio, and J. Romero-Luis, "360° Video for Research Communication and Dissemination: A Case Study and Guidelines," *IEEE Transactions on Professional Communication*, vol. 66, no. 1, pp. 59–77, 2023.

[12] A. A. Simiscuka, M. A. Togou, M. Zorrilla, and G.-M. Muntean, "360-ADAPT: An Open-RAN-Based Adaptive Scheme for Quality Enhancement of Opera 360° Content Distribution," *IEEE Transactions on Green Communications and Networking*, vol. 8, no. 3, pp. 924–938, 2024.

[13] A. A. Simiscuka, D. A. Ghadge, and G.-M. Muntean, "OmniScent: An Omnidirectional Olfaction-Enhanced Virtual Reality 360° Video Delivery Solution for Increasing Viewer Quality of Experience," *IEEE Transactions on Broadcasting*, vol. 69, no. 4, pp. 941–950, 2023.

[14] O. Eltobgy, O. Arafa, and M. Hefeeda, "Mobile streaming of live 360-degree videos," *IEEE Transactions on Multimedia*, vol. 22, no. 12, pp. 3139–3152, 2020.

[15] K. Khan, "Advances and Challenges in 360 Mixed Reality Video Streaming: A Comprehensive Review," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, vol. 6, no. 6, 2023.

[16] A. Yaqoob and G. M. Muntean, "A Combined Field-of-View Prediction-Assisted Viewport Adaptive Delivery Scheme for 360° Videos," *IEEE Transactions on Broadcasting*, vol. 67, no. 3, pp. 701–713, 2021.

[17] E. Stafidas and F. Foukalas, "A Survey on Enabling XR Services in Beyond 5G Mobile Networks," *IEEE Access*, vol. 12, pp. 42 856–42 885, 2024.

[18] K. Khan, "Advancements and Challenges in 360-Degree Virtual Reality Video Streaming at the Edge: A Comprehensive Review," *International Journal of Multidisciplinary Research and Publications (IJMRAP)*, vol. 6, no. 6, pp. 182–192, 2023, accessed March 23, 2025.