CITE

Journal of Analytical and Applied Computer Sciences
Volume: 2, Issue: 1, 2026 (January - June), Pages: 12-20

Research Article

An Offline Classroom Presentation System *Via* Local Area Networks for Real-Time Screen Sharing

Awoyale Mercy Ayomidimeji, Alimi Olasunkanmi Maruf, Oluwaseyi Ezekiel Olorunshola, Adeniyi Usman Adedayo, Enem A. Theophilus, Adamu-Fika Fatimah

Department of Computer Science, Faculty of Computing, Air Force Institute of Technology, Kaduna, Nigeria.

*Correspondence
Awoyale Mercy Ayomidimeji
mercyawoyale6@gmail.com

Received: 2 January 2026 / Accepted: 25 April 2026 / Published: 30 June 2026

Traditional multimedia projectors in Nigerian classrooms are frequently affected by technical failures, poor maintenance, and outright unavailability, disrupting lessons and reducing student engagement. This study developed an offline classroom presentation system that uses WebRTC technology over a local wireless network as a practical alternative to traditional projectors. The system comprises a presenter device, a Node.js signalling server built with Express.js and Socket.io, a Mediasoup Selective Forwarding Unit (SFU), and multiple viewer devices, all connected over a classroom local area network (LAN) without internet access. The presenter uploads a single media stream to the SFU, which forwards it independently to each connected viewer, eliminating the bandwidth and CPU scaling constraints associated with a peer-to-peer topology. The frontend was built with HTML, CSS, and vanilla JavaScript, requiring only a modern browser for access, while the server enforced HTTPS using pre-generated self-signed certificates to satisfy the browser security requirements of the screen-capture API. System performance was evaluated using latency, CPU and RAM usage, scalability, and cross-browser compatibility metrics. Results showed stable, low-latency screen and audio sharing to up to twenty simultaneous viewers, with average latency remaining below 100 ms and consistent performance across Chrome, Edge, and Firefox. The findings confirm the technical feasibility of the system and its ability to replicate the core functionality of a projector without dependence on internet connectivity or costly hardware. The study concludes that WebRTC-based LAN systems represent a viable, scalable, and cost-effective alternative to traditional classroom projectors, with meaningful implications for educational technology deployment in resource-constrained environments.

Keywords: *Offline Classroom, Presentation System, Local Area, Networks, Real-Time Screen Sharing*

Introduction

Modern classrooms increasingly rely on multimedia technologies to enhance teaching and learning, with projectors serving as the primary tool for delivering visual content from a lecturer's screen to students. However, the reliability and availability of projectors in Nigerian educational institutions remain significant challenges. A study of senior secondary schools in Nnewi Metropolis found frequent projector unavailability, damaged bulbs, and connectivity issues, all of which negatively affected lesson continuity and student engagement (Umeozor & Onuh, 2023). Beyond the immediate disruption to lessons, maintaining and replacing projectors is costly and impractical for institutions operating under tight

budgets, while research shows that clear and reliable visual presentation of course materials significantly enhances students' understanding and retention (Egboka et al., 2022). Existing WebRTC-based screen-sharing solutions, though technically robust, are largely designed for internet-connected environments and have not been adapted for offline, LAN-only classroom deployment (Kasetwar et al., 2022; Mahmoud & Abozariba, 2024), leaving educators in under-resourced settings without a practical digital alternative. This combination of unreliable hardware and an unaddressed gap in the WebRTC literature creates a clear and pressing need for a reliable, accessible, and low-cost alternative to traditional classroom projection systems. This study addresses this gap by designing, implementing, and evaluating an offline classroom presentation system that uses WebRTC (Web Real-Time Communication) technology over a local wireless network to enable real-time screen and audio sharing. WebRTC is an open-source, browser-native technology that supports low-latency, encrypted, real-time communication such as screen sharing, audio, and video directly between web browsers without plugins or external software (Mahmoud & Abozariba, 2024). Because WebRTC can operate entirely over a local network, it functions without reliance on external servers, making it well suited to offline classroom environments. The system follows a client-server architecture centred on a Selective Forwarding Unit (SFU): a Node.js server, built with Express.js and Socket.io, acts as the signalling hub responsible for session management and the exchange of WebRTC transport metadata, while a Mediasoup SFU running in-process with the server handles all media routing. The presenter uploads a single stream to the SFU, which forwards it independently to every connected viewer, eliminating the need for the presenter's browser to maintain a separate connection per viewer (Andre et al., 2018). By deploying the system over a local area network (LAN), lecturers can share their screens and audio directly to students' laptops, tablets, and smartphones, ensuring that teaching continues uninterrupted even where projectors are unreliable or absent (De Silva, 2021; Diallo et al., 2023).

Aim and Objectives

The aim of this study is to develop an offline classroom presentation system, delivered over a local wireless network, that enables real-time screen and audio sharing across multiple devices as a cost-effective and practical alternative to traditional multimedia projectors, without requiring internet connectivity. The specific objectives are to:

i. design a functional local wireless network architecture that supports real-time screen and audio sharing within a classroom environment, ensuring stable connectivity for at least ten concurrent devices;

ii. develop and implement a WebRTC-based presentation system using Node.js, Socket.io, Mediasoup (SFU), and a vanilla JavaScript frontend, capable of enabling simultaneous screen sharing across multiple browser-enabled devices on a local network; and

iii. measure and evaluate system performance by testing latency (target: ≤ 300 ms), device compatibility (across at least three major browsers), and resource usage (CPU and memory consumption) under controlled classroom conditions.

Scope of the Study

This study is concerned with the design, development, and performance evaluation of an offline classroom presentation system supporting one-to-many screen and audio sharing from a single presenter to multiple viewer devices connected to the same LAN, without internet access. The target deployment environment is a classroom in which the presenter's machine hosts the Node.js server while browser-based clients join over HTTPS via the local IP address. Performance evaluation was conducted with up to twenty simultaneous viewer devices, a scale identified as representative of a typical classroom. The frontend was scoped to screen-sharing and session-management functionality only, without annotation, file-transfer, or interactive whiteboard features; the system also supports only one active presenter at a time, and session access is controlled solely by possession of a randomly generated four-digit session code.

Umeozor and Onuh (2023) investigated the availability and utilisation of multimedia projectors in senior secondary schools in Nnewi Metropolis, Nigeria, and found that projectors were frequently non-functional owing to damaged cables, faulty bulbs, and poor maintenance, regularly disrupting lessons and reducing student engagement. This study provided the primary local evidence establishing the infrastructure problem addressed by the present research. De Silva (2021) explored the adoption of technology in smart classrooms, highlighting institutional and technical barriers to implementing LAN and Wi-Fi-based teaching tools and underscoring the need for reliable, low-cost, and easily deployable systems, though without proposing a specific real-time screen-sharing solution. Egboka et al. (2022) evaluated the effects of multimedia video projection on undergraduate students' academic achievement and found that multimedia projection significantly improved engagement and retention when the delivery medium functioned as intended, but that flickering images, bulb failures, and poor connectivity frequently undermined these benefits. Together, these studies establish both the educational value of reliable visual content delivery and the practical cost that projector unreliability imposes on classroom instruction. Mahmoud and Abozariba (2024) conducted a systematic review of WebRTC applications, identifying the technology's core strengths in low-latency, encrypted, peer-to-peer communication and highlighting a limited body of literature on offline and LAN-specific deployments. Diallo et al.

(2023) evaluated WebRTC video streaming on resource-constrained devices and identified resolution and frame-rate settings that maintain performance on low-powered hardware, findings directly applicable to the classroom laptops targeted by the present study. Kasetwar et al. (2022) and Edan (2022) each demonstrated the technical feasibility of one-to-many WebRTC screen sharing using Node.js-based signalling, but neither incorporated the session governance, offline LAN constraints, or classroom-specific features required for formal educational deployment. Rathee et al. (2022) documented WebRTC signalling and screen-sharing workflows for audio and video conferencing, offering insights applicable to LAN-based deployment but without addressing offline classroom scenarios. Andre et al. (2018) compared several open-source SFU implementations, including Mediasoup, Janus, and Jitsi, and their analysis directly informed the selection of Mediasoup for the present system on account of its performance, zero-transcoding design, and native Node.js integration. Sunder Saini and Sen Sharma (2023) conducted empirical performance evaluations of WebRTC-based conferencing, measuring latency, bandwidth, and CPU usage under varying numbers of simultaneous users, and their benchmarks and evaluation methodology informed the performance metrics adopted in this study. The reviewed literature reveals a well-established but contextually narrow body of work on WebRTC-based screen sharing: the majority of prior implementations target internet-dependent, general-purpose conferencing rather than offline, locally-deployed classroom systems. While Kasetwar et al. (2022), Edan (2022), and Rathee et al. (2022) demonstrate the technical feasibility of WebRTC screen sharing, none of these systems operate in an offline, LAN-only environment or incorporate the session governance and HTTPS delivery mechanisms required for classroom deployment without internet access. Umeozor and Onuh (2023) established the scale of the infrastructure problem in Nigerian schools but proposed no digital alternative, and although Mahmoud and Abozariba (2024) explicitly called for context-specific WebRTC implementations in offline and resource-constrained environments, no study in the reviewed literature answered that call with a working classroom system. This study addresses that gap by designing, implementing, and evaluating an offline, LAN-based, browser-native screen-sharing platform built specifically for the classroom context identified in the literature.

Methodology

1. Analysis of the Existing System

Visual content delivery in Nigerian classrooms currently relies primarily on traditional multimedia projectors, connected to a lecturer's laptop via HDMI, VGA, or a wireless adaptor and projected onto a screen or wall. Visibility depends heavily on seating position, room lighting, and projector brightness, and in larger halls a single projected image often cannot serve every student adequately. The current system exhibits several well-documented problems: equipment unreliability caused by damaged bulbs, faulty cables, and connector incompatibility (Umeozor & Onuh, 2023); high maintenance cost arising from limited bulb lifespans; visibility limitations for students seated far from or at an angle to the screen; an absence of interactivity or personalisation, since content cannot be zoomed, annotated, or retained; setup dependency on physical cabling or wireless pairing; and a single point of failure, whereby a projector fault disrupts or cancels the lesson entirely with no fallback mechanism.

2. Proposed System Overview

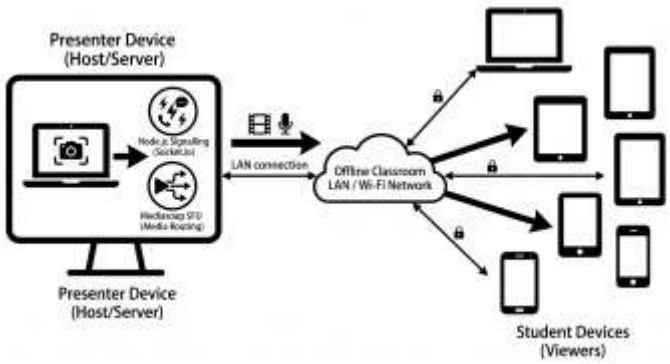

Figure 1. System Architecture Diagram

The proposed system replaces the projector with a browser-based, LAN-hosted screen-sharing application. Rather than projecting to a single shared screen, the presenter's display is streamed in real time to each student's personal device, which functions as an individual screen, over the classroom's existing Wi-Fi infrastructure without requiring internet connectivity. The architecture comprises four components: the presenter's device, a Node.js signalling server, a Mediasoup SFU running on the same machine, and the viewer devices. The server performs three roles: serving the

HTML/CSS/JavaScript frontend via Express.js; managing WebRTC signalling via Socket.io; and operating the Mediasoup SFU, which receives the presenter's single outgoing stream and forwards it independently to every connected viewer. All media passes through the SFU; no direct browser-to-browser connections exist. Session access is governed by a randomly generated four-digit code, which viewers use to join a session directly. Figure 1 presents the system architecture.

3. System Requirements

The system was designed against a set of functional and non-functional requirements. Functionally, the system must support session creation with a unique four-digit code, code-based session joining, browser-based screen and audio capture via the getDisplayMedia API, media streaming through the Mediasoup SFU to all connected viewers in real time, support for late joiners without requiring the presenter to restart sharing, and graceful session and per-viewer teardown that releases server-side resources without disrupting other viewers. Non-functionally, the system was required to keep end-to-end latency below 300 ms, support at least twenty simultaneous viewers, operate entirely offline using only local ICE candidates with no STUN or TURN queries, remain compatible with at least three major browsers without plugins, forward media without transcoding to keep CPU usage predictable, enforce DTLS and HTTPS encryption throughout, automatically recover the Mediasoup Worker process within two seconds of a crash, and require no software installation beyond a modern web browser on viewer devices.

4. Research Design and Development Approach

This study adopted a Design and Development Research (DDR) methodology, suited to projects whose primary output is a functional technical artefact, involving iterative cycles of design, implementation, and evaluation. The research proceeded through three phases. The first was system analysis and design, in which requirements were defined, the architecture was planned, and technology choices were justified against the literature. The second was implementation, in which the signalling server, Mediasoup SFU integration, WebRTC transport lifecycle, HTTPS delivery mechanism, session-management logic, and browser-based frontend were developed and integrated. The third was evaluation, in which the system was tested under controlled conditions measuring latency, CPU and RAM usage, scalability to twenty simultaneous viewers, and cross-browser compatibility. Quantitative metrics were summarised descriptively and compared against benchmarks identified in the literature review, and a qualitative comparison was drawn between the proposed system and the traditional projector setup based on the failure modes documented in the reviewed studies. Evaluation conditions mirrored a realistic classroom deployment, with multiple devices on the same wireless LAN accessing the system via a local HTTPS URL, and the presenter's machine serving as both the server host and the stream source.

5. Tools and Technologies

The system was implemented in JavaScript across both server and client layers, a choice motivated by WebRTC's browser-native APIs and Node.js's extension of JavaScript to the server, which unifies the technology stack and reduces context-switching during development. Node.js was selected for the server owing to its event-driven, non-blocking I/O model, well suited to managing many concurrent WebSocket connections with low latency; Express.js served static frontend files; and Socket.io provided a high-level abstraction over WebSocket connections for signalling, including reconnection handling and room-based session management. Mediasoup (v3) was selected as the SFU library for its native C++ Worker process, which routes RTP media with zero transcoding overhead and integrates directly with Node.js. The client used vanilla JavaScript, HTML5, and CSS3 without a frontend framework, keeping the interface lightweight for the low-specification devices common among students. Pre-generated self-signed SSL certificates, stored locally and loaded at server startup, enabled HTTPS delivery on local IP addresses without requiring an external certificate authority, satisfying the secure-context requirement that modern browsers impose on the getDisplayMedia API.

6. System Design

The system follows a client-server architecture with the Mediasoup SFU at its media core, as illustrated in Figure 1: the Node.js server sits at the centre of the signalling layer, connected to all browser clients via WebSocket, while the SFU independently forwards RTP packets from the presenter to every viewer with no direct browser-to-browser traffic. Two actors interact with the system: the Presenter, who starts a session, generates a session code, shares a screen, and stops the session; and the Viewer, who joins a session using the code, views the shared screen, and leaves the session. A SignallingServer class manages a map of active Session objects, each holding a reference to its own Mediasoup Router. The core session-management algorithm generates a random four-digit session identifier and a dedicated Router on

session creation, admits any viewer presenting a valid code directly to the corresponding Socket.io room, and removes the session and notifies all viewers when the presenter disconnects. The WebRTC signalling algorithm begins with the presenter capturing a MediaStream via getDisplayMedia() and producing it on a SendTransport created through Mediasoup's transport-based negotiation model; the server then notifies each connected viewer of the new producer, and each viewer creates a RecvTransport and issues a consume request, which the server validates for codec compatibility using router.canConsume() before creating a Consumer. Each transport completes a DTLS handshake on first use, using strictly local ICE candidates with no STUN or TURN servers, after which the viewer's video element renders the presenter's screen in real time.

Implementation

1. Server Initialisation

The server is started with a single command, node server.js, and performs four sequential operations on startup. It reads the pre-generated SSL certificate and private key and binds them to an HTTPS server instance; it enumerates all available network interfaces to determine the host machine's LAN IPv4 address, which is printed to the console as the join URL for students; it spawns a Mediasoup Worker process to manage the native RTP routing engine, registering a died event listener that automatically restarts the Worker after a two-second delay in the event of a crash; and it attaches a Socket.io server to the HTTPS instance, ready to accept persistent WebSocket connections from presenter and viewer clients.

2. Session and Approval Logic

When the presenter clicks Start, the client emits a create-session event; the server creates a Mediasoup Router configured with three codecs (Opus for audio, and VP8 and H.264 for video), returns the Router's RTP Capabilities to the presenter, and stores the session in an in-memory map keyed by the four-digit session code, alongside references to the presenter's socket ID, the Router, active transports, producers, and connected viewers. When a student enters the code and clicks Join, a join-session event is validated against this map; if valid, the viewer is immediately admitted to the corresponding Socket.io room and sent the Router's RTP Capabilities, with no presenter intervention required.

3. Transport Creation and DTLS Handshake

Rather than exchanging SDP offer/answer messages as in direct peer-to-peer WebRTC, the system uses Mediasoup's transport-based negotiation model. The presenter emits create-send-transport; the server calls router.createWebRtcTransport(), binding the transport to listen on all available LAN interfaces across both UDP and TCP, and returns the ICE and DTLS parameters needed to construct a SendTransport on the client. On the first produce() call, the transport fires a connect event, and the client's DTLS parameters are sent to the server to complete the handshake. The same exchange is repeated for each viewer via a create-recv-transport event, with a dedicated WebRtcTransport created per viewer so that each connection can be independently managed and closed without affecting others.

4. Media Production and Consumption

When the presenter clicks Share Screen, the browser invokes getDisplayMedia() to obtain a MediaStream containing up to one video track and one audio track. For each track, the client calls sendTransport.produce(), which creates a Producer on the Router and triggers a new-producer notification to every connected viewer. Each viewer then issues a consume request containing the producer ID and its own RTP capabilities; the server verifies codec compatibility with router.canConsume() before creating a Consumer and returning its parameters. The viewer's client calls recvTransport.consume() with these parameters, obtaining a Consumer whose track is assigned to the srcObject of an HTML video element, rendering the presenter's screen in real time.

5. Session Teardown

When the presenter ends the session, a cleanupSession() routine closes all viewer transports and consumers, closes the presenter's send transport and producers, closes the session's Mediasoup Router to free all allocated RTP ports, and removes the session from the sessions map, while a session-ended event is broadcast to all viewers. When an individual viewer disconnects, a symmetric cleanupViewer() function closes only that viewer's transport and consumers and notifies the presenter, without affecting any other viewer.

Results and Discussion

The system was tested under controlled conditions on a local classroom wireless network, with all devices connected to the same Wi-Fi access point and no internet connection present. Testing was conducted in three stages, latency measurement, resource-usage profiling, and scalability evaluation, followed by cross-browser compatibility testing.

1. Latency

End-to-end latency was measured as the elapsed time between a visible change on the presenter's screen and its appearance on a viewer's rendered video element, using screen recordings on both devices with timestamp accuracy of approximately one frame (33 ms at 30 fps); ten measurements were taken at each viewer count. Table 1 presents the results.

Viewer Count	Average Latency (ms)	Maximum Latency (ms)
1	55	81
5	66	104
10	77	128
15	102	155
20	85	179

Table 1. Latency Measurements at Varying Viewer Counts

All measured latency values fell well within the 300 ms target defined in the study's objectives. Latency increased modestly as viewer count grew, reflecting the additional routing load on the Mediasoup Worker, but remained stable and well below the perceptible-delay threshold for screen-sharing applications.

2. CPU and RAM Usage

Server-side CPU and RAM usage were monitored using Node.js's process.cpuUsage() and process.memoryUsage() APIs, sampled every five seconds during active sessions, while client-side resource usage was observed via each browser's built-in task manager. Table 2 presents the results.

Viewer Count	Server CPU Usage (%)	Server RAM (MB)	Viewer RAM per Device (MB)
1	7.8	150	119
5	16.6	202	120
10	28.4	260	123
15	40.6	321	125
20	53.0	379	127

Table 2. Server CPU and RAM Usage at Varying Viewer Counts

Server CPU usage scaled predictably with viewer count, reflecting Mediasoup's per-packet forwarding workload, while RAM consumption increased incrementally as each viewer's transport, consumer, and associated routing state were allocated in memory. Critically, CPU growth was sub-linear: the absence of transcoding meant that the server's per-packet processing cost remained largely independent of the resolution or codec configuration of the presenter's stream. Viewer-side memory consumption was minimal and consistent across device counts, confirming that the SFU architecture successfully offloaded the distribution burden from individual devices.

3. Scalability

Viewer Count	Successful Connections	Stream Interruptions
1	1 of 1	None
5	5 of 5	None
10	10 of 10	None
15	15 of 15	None
20	20 of 20	None

Table 3. Scalability Test Results

Scalability was evaluated by incrementally adding viewer connections to an active session and observing whether each new viewer successfully received the media stream without disrupting the existing viewers, from one to twenty simultaneous viewers. As shown in Table 3, all twenty viewers successfully received the presenter's stream throughout the session, with no dropped connections or stream interruptions observed.

4. Browser Compatibility

The system was tested on Google Chrome (v120), Microsoft Edge (v119), and Mozilla Firefox (v121). As summarised in Table 4, all three browsers successfully loaded the presenter and viewer interfaces over HTTPS, completed the Mediasoup transport handshake, and rendered the shared screen without modification. Chrome and Edge, both Chromium-based, produced identical behaviour, while Firefox exhibited marginally higher initial connection time owing to differences in its ICE agent implementation.

Browser	Session Join	Screen Share (Presenter)	Stream Playback (Viewer)
Chrome 120	Pass	Pass	Pass
Edge 119	Pass	Pass	Pass
Firefox 121	Pass	Pass	Pass

Table 4. Browser Compatibility Results

The results confirm that the system achieves its objectives across all four evaluation dimensions. Measured latency ranged from an average of 55 ms at one viewer to 85 ms at twenty viewers, well below the 300 ms threshold specified in Objective (iii) and within the sub-100 ms range that the WebRTC literature identifies as imperceptible in screen-sharing contexts; the gradual increase with viewer count reflects the routing workload of the Mediasoup Worker rather than any congestion on the presenter's connection, and is enabled directly by the SFU architecture, since the presenter's browser uploads only a single stream regardless of viewer count. Server resource usage scaled predictably with viewer count, rising from 7.8% CPU and 150 MB of RAM at one viewer to 53.0% CPU and 379 MB of RAM at twenty, remaining within the operating range of a standard laptop and leaving headroom for the presenter to run other applications concurrently; the absence of transcoding in the Mediasoup SFU was the critical factor here, since the server forwards RTP packets without re-encoding, keeping the CPU cost dominated by packet inspection and routing rather than computationally expensive codec operations.

The scalability test demonstrated that all twenty simultaneous viewers received the stream without interruption, exceeding the objective of stable connectivity for at least ten concurrent devices; this outcome was made possible by the SFU model, which keeps the presenter's upload cost constant regardless of viewer count, unlike a peer-to-peer design in which CPU and upstream bandwidth demand grow linearly with each additional viewer and saturate at approximately five to ten viewers. Browser-compatibility results confirmed that the system is accessible to students on any of the three major browsers without configuration changes, aided by the Mediasoup Router's dual codec support for VP8 and H.264, which allows router.canConsume() to match each viewer's Consumer to the codec best suited to their browser.

Compared with the traditional projector system, the implemented system addresses every problem identified in Section 3.1: equipment unreliability is eliminated because the system runs entirely in software on hardware the lecturer already owns; maintenance cost is limited to the initial installation of Node.js and its dependencies; visibility limitations are resolved because every student receives an individually rendered, full-resolution copy of the presenter's screen; the setup dependency on physical cabling is replaced by a single command; and the single point of failure characteristic of projector-based delivery is mitigated by the system's software-only stack and its automatic Worker-restart mechanism. One limitation observed during testing was that browser warning dialogues for untrusted, self-signed SSL certificates must be manually dismissed by each student on first access; this is a known trade-off of offline deployment, anticipated in the study's design, and in practice took fewer than ten seconds per device.

Conclusion

This study set out to determine whether WebRTC technology, delivered over a local wireless network without internet connectivity, could provide a reliable, accessible, and cost-effective alternative to traditional multimedia projectors in Nigerian classrooms. The evidence gathered through implementation and performance testing confirms that it can: the system met or exceeded all of its performance objectives, with latency consistently below the 300 ms threshold, resource usage that was moderate and predictable, no hardware requirement beyond a standard laptop and a classroom Wi-Fi router, uninterrupted service to twenty simultaneous viewers, and confirmed functionality across three major browsers. The architectural decision to use a Mediasoup SFU was vindicated by the scalability and resource data: the single-upload model eliminated the presenter-side bottleneck that would have made an earlier peer-to-peer approach impractical beyond roughly ten viewers. This study makes three main contributions to the existing body of knowledge. First, it presents the design and implementation of a fully functional, offline-deployable, browser-native screen-sharing system built for the specific constraints of Nigerian classroom environments, operating without internet connectivity,

enforcing HTTPS via self-signed certificates on a LAN, and incorporating a session-governance mechanism not present in comparable prior work. Second, it provides empirical evidence that a Mediasoup SFU architecture can be deployed on a single laptop within a Node.js process and support twenty simultaneous viewer connections with sub-100 ms latency and zero transcoding overhead, establishing a concrete performance baseline for SFU-based offline classroom deployments that did not previously exist in the literature. Third, it documents a replicable technical blueprint, including server initialisation, transport lifecycle management, multi-interface binding, automatic Worker recovery, and per-session resource cleanup, that can be adopted or extended by other researchers and developers targeting similar offline educational-technology deployments in resource-constrained environments. Beyond these technical results, the system addresses a real and documented educational infrastructure problem. By removing the dependency on projector hardware, eliminating the single point of failure, and delivering individually rendered, full-resolution screen content to every student's personal device, the system offers a materially better classroom experience than the projector it replaces, at lower cost and with greater reliability. It is therefore concluded that WebRTC-based LAN systems of the kind implemented in this study represent a viable, scalable, and educationally sound alternative to traditional classroom projectors, with significant implications for educational technology deployment in resource-constrained environments.

Recommendations

Based on the findings and limitations identified during this study, the following recommendations are made for future research and development.

i. Certificate management: future work should investigate the use of mkcert or a similar local certificate authority tool to install a trusted root certificate on the access point or classroom devices, eliminating the browser security warning without requiring an internet-based certificate authority.

ii. Larger-scale testing: future work should evaluate the system's performance at fifty, one hundred, and more simultaneous viewers, potentially distributing load across multiple Mediasoup Workers or server instances to establish the practical upper limit of the architecture.

iii. Adaptive bitrate streaming: future implementations could configure Mediasoup to support simulcast encoding on the presenter's stream, allowing the SFU to forward different quality layers to viewers based on available bandwidth.

iv. Persistent session recording: future work could integrate a server-side recording mechanism, using Mediasoup's PlainTransport to route the media stream to a recording process, enabling sessions to be saved and replayed.

v. Bidirectional interaction: future extensions could enable viewers to request the floor and temporarily share their own screens, supporting collaborative presentations and student demonstrations.

vi. Integration with learning management systems: deploying the system as a module within existing platforms such as Moodle would reduce the adoption barrier and allow session management to integrate with attendance and participation records.

vii. Mobile application packaging: wrapping the browser-based client in a Progressive Web App or native mobile application would improve the user experience on smartphones, particularly by simplifying the HTTPS certificate acceptance workflow.

Author contributions

Awoyale Mercy Ayomidimeji, Alimi Olasunkanmi Maruf, Oluwaseyi Ezekiel Olorunshola, Adeniyi Usman Adedayo, Enem A. Theophilus, and Adamu-Fika Fatimah are planned executed the tasks.

Funding

No funding

Conflict of interest

The author declares no conflict of interest. The manuscript has not been submitted for publication in other journal.

Ethics approval

Not applicable

AI tool usage declaration

The authors not used any AI and related tools to write this manuscript.

References

Andre, E., Le Breton, N., Lemesle, A., Roux, L., & Gouaillard, A. (2018). Comparative Study of WebRTC Open Source SFUs for Video Conferencing. *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, 1–8. https://doi.org/10.1109/IPTCOMM.2018.8567642

De Silva, J. U. (2021). *UTILIZING TECHNOLOGY IN THE CLASSROOM* [Selinus University of Science and Literature]. https://uniselinus.education/sites/default/files/2021-07/Tesi%20De%20Silva.pdf

Diallo, B., Ouamri, A., & Keche, M. (2023). A Hybrid Approach for WebRTC Video Streaming on Resource-Constrained Devices. *Electronics*, *12*(18), 3775. https://doi.org/10.3390/electronics12183775

Edan, N. M. (2022). *Design And Implementation Of Webrtc Screen Sharing*. *19*(3).

Egboka, P., Aliyu, H. D., Ahmed, D., & Muntaka, T. (2022). Effects of Multimedia Video Projection on Undergraduate Students' Achievement and Retention in Quantitative Economics in the Universities in Gombe State, Nigeria. *Kashere Journal of Education*, *2*(2), 259–266. https://doi.org/10.4314/kje.v2i2.32

Kasetwar, A., Balani, N., Damwani, D., Pandey, A., Sheikh, A., & Khadse, A. (2022). A WebRTC Based Video Conferencing System with Screen Sharing. *International Journal for Research in Applied Science and Engineering Technology*, *10*(5), 5061–5066. https://doi.org/10.22214/ijraset.2022.43595

Mahmoud, H., & Abozariba, R. (2024). A systematic review on WebRTC for potential applications and challenges beyond audio video streaming. *Multimedia Tools and Applications*, *84*(6), 2909–2946. https://doi.org/10.1007/s11042-024-20448-9

Rathee, P., Bhatla, D., Khan, S., Chowdhary, S., Diwan, V., & Maharaja Surajmal Institute of Technology. (2022). VIDEO/AUDIO CONFERENCING USING WEBRTC. *International Journal of Engineering Applied Sciences and Technology*, *7*(4), 276–280. https://doi.org/10.33564/IJEAST.2022.v07i04.043

Sunder Saini, S., & Sen Sharma, L. (2023). Performance Evaluation of WebRTC-Based Video Conferencing: A Comprehensive Analysis. *Journal of Advanced Zoology*, *44*(S7), 322–330. https://doi.org/10.17762/jaz.v44iS7.2740

Umeozor, U. J., & Onuh, U. B. (2023). Availability and utilization of multimedia projectors in teaching of computer science in senior secondary schools in nnewi metropolis. *Unizik Journal of Educational Management and Policy*, *5*(2), 73–85.