International Journal on Perceptive and Cognitive Computing (IJPCC) Vol 12, Issue 1 (2026)
https://doi.org/10.31436/ijpcc.v12i1.678

A Hybrid Overlay Architecture for Social Feature Integration in Browser-Based Cloud Gaming

Ahmad Nur Zafran Shah bin Ahmad Shahrizal, Danish Haikal bin Mohammad, Zainab S. Attarbashi*, Amal Abdulwahab Hasan Alamrami, Nur-Adib Maspo
Department of Computer Science, International Islamic University Malaysia,

*Corresponding author zainab_senan@iium.edu.my
(Received: 9th December 2025; Accepted: 2nd January 2026; Published on-line: 30th January 2026)

Abstract— Current cloud gaming platforms force a trade-off between streaming performance and integrated social features, typically requiring resource-intensive dedicated clients. This paper presents an architecture that eliminates this compromise through a Hybrid Overlay engine. Built with vanilla TypeScript/HTML5 and decoupled from the WebRTC video pipeline, the engine renders social overlays (chat, friend lists) directly onto the game canvas, avoiding DOM overhead. A Rust/Actix-web backend ensures low-latency streaming. The system was validated through comprehensive testing. Performance and security tests confirmed: automatic streamer binary compilation, successful WebRTC stream initiation, automated SSL generation, and strict HTTPS enforcement. Functional tests demonstrated robust authentication (registration, session persistence), real-time message synchronization (<200ms), and correct social workflows. Crucially, the input sandbox isolated chat keystrokes from the game stream, and Firebase RBAC rules blocked all unauthorized data writes. By unifying high-fidelity streaming with lightweight, native social integration, this work provides a performant, zero-install platform that makes social cloud gaming accessible on low-end devices, establishing a new model for architecting these services.

Keywords— Cloud Gaming, WebRTC, Low-Latency Streaming, TypeScript, Performance Optimization.

I. Introduction

The gaming industry has evolved into a dominant segment of the technology landscape, motivated by advancements in internet infrastructure. This evolution has given rise to cloud-based gaming platforms, which enable users to stream games remotely from high-performance servers, thereby offloading intensive computational tasks such as physics processing and graphical rendering [1]. As cloud gaming technology has matured offering improved accessibility and performance, its market is projected to exceed $21.4 billion by 2028 [2]. Yet, despite its potential, mainstream cloud gaming has yet to fully realize its promise as a unified social experience. Current platforms frequently lack deeply integrated social functionalities, rely on resource-heavy dedicated applications, and often employ subscription models or hardware constraints. These shortcomings fragment the user experience, forcing players to depend on third-party applications for communication and coordination, which introduces complexity and overhead particularly on low-end devices. Figure 1 shows an online cloud gaming structure as described by [3].

The social dimension of gaming is not peripheral; it is central. A 2023 global survey indicated that 46% of gamers play weekly to connect with friends [4], underscoring the role of games as platforms for community and collaboration. For such experiences to be viable in the cloud, minimizing end-to-end latency is paramount to preserve interactivity and the sense of shared presence. This work therefore addresses the need for a cloud gaming architecture designed from the ground up to integrate social features natively while employing technologies suited for low-latency delivery, all within an accessible, browser-based interface. This paper presents the design, implementation, and evaluation of a lightweight, browser-native cloud gaming platform built to address this gap. The core contribution is a Hybrid Overlay architecture that decouples social features including real-time chat, friend management, and community hubs from the game streaming pipeline. The frontend is implemented in vanilla TypeScript and HTML5, rendering interactive overlays directly onto the video canvas to avoid Document Object Model (DOM) overhead. The backend uses the Rust programming language with the Actix-web framework to manage signalling and WebRTC peer connections, ensuring low-latency media delivery. User state and real-time data are synchronized via Firebase Cloud Services, providing scalable authentication and data persistence.

https://doi.org/10.31436/ijpcc.v12i1.678

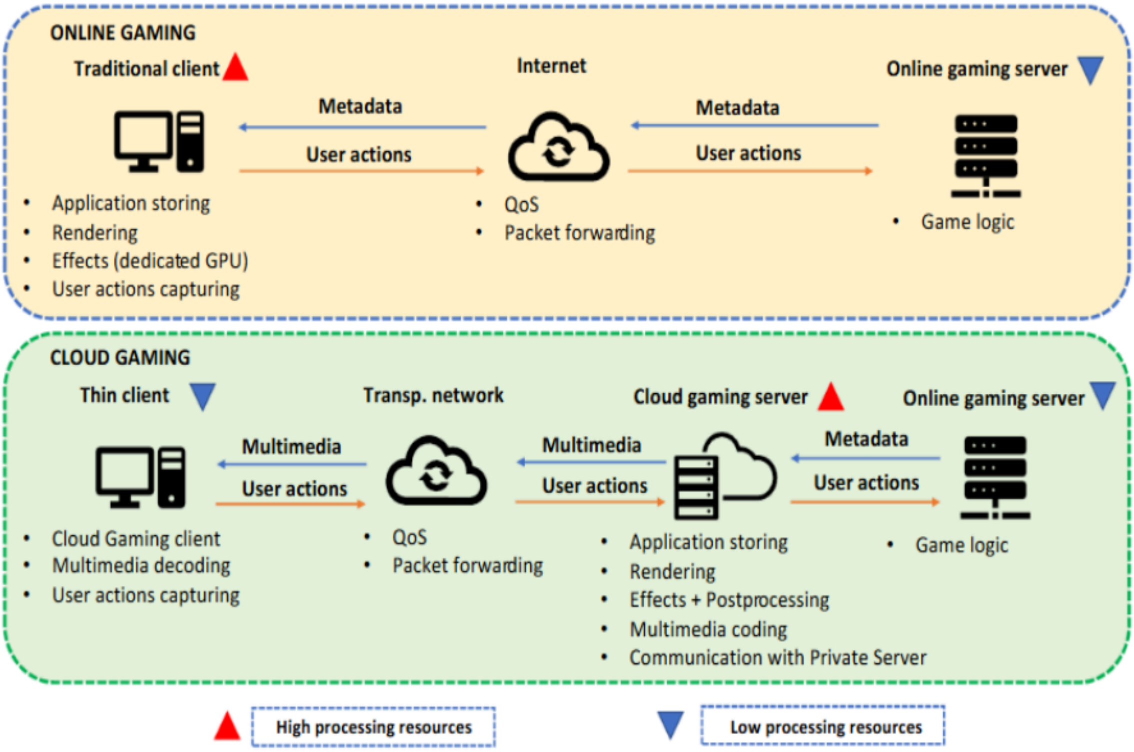

Fig. 1 Online and Cloud Gaming structures [3]

The remainder of this paper is organized as follows: Section 2 reviews related work in cloud gaming architectures and social feature integration. Section 3 details the system design and implementation, including the Hybrid Overlay architecture and the full technology stack. Section 4 presents the results of functional, performance, and security testing. Finally, Section 6 concludes the paper with a discussion of the findings, acknowledges limitations, and suggests directions for future work.

II. Related Works

This section reviews the architectural approaches and limitations of existing commercial platforms, followed by an analysis of key technical research that informs the design of efficient, socially integrated systems.

A. *Commercial Platforms and Their Limitations*

Leading services such as NVIDIA GeForce NOW, Xbox Cloud Gaming, Boosteroid, and Shadow PC demonstrate the current state of the industry, yet each exhibits significant trade-offs in accessibility, social integration, and cost (see Table 1). NVIDIA GeForce NOW provides broad game library support and basic voice chat but requires a desktop client for optimal performance and lacks built-in community features [5]. Xbox Cloud Gaming uses Microsoft's ecosystem but is restricted to a subscription model and depends on external Xbox Live services for social interaction [6]. Boosteroid offers browser-based access, improving platform agnosticism, but provides no free tier or native social capabilities. Shadow PC delivers full desktop virtualization, offering maximum flexibility at the cost of high pricing, a mandatory client application, and no integrated social tools [7]. A consistent pattern across these platforms is the reliance on third-party applications for communication and community functions, which fragments the user experience and introduces additional overhead, particularly on low-end devices.

TABLE I
FEATURE COMPARISON OF MAJOR CLOUD GAMING PLATFORMS

Feature	NVIDIA GeForce NOW	Xbox Cloud Gaming	Boosteroid	Shadow PC
Primary Access	Client & limited browser	Client & mobile browser	Web browser	Desktop client
Free Tier	Limited	No	No	No
Built-in Voice Chat	Yes	No	No	No
Integrated Community Hub	No	Via Xbox Live	No	No
Social Dependency	High (3rd party)	High (Xbox app)	High (3rd party)	High (3rd party)

B. Technical Foundations and Research

Early research established the fundamental shift from traditional online gaming to a cloud-based model, highlighting the critical role of network quality and server-side processing [8]. Studies have extensively analyzed the impact of latency, jitter, and packet loss on Quality of Experience (QoE), leading to adaptive streaming techniques that adjust video parameters in response to network conditions [9], [10]. Further research into traffic characterization of services like OnLive has informed protocol design and server optimization for real-time streaming [11].

The emergence of Web Real-Time Communication (WebRTC) has been essential for browser-based streaming. It provides a standardized API for peer-to-peer, low-latency media delivery directly within browsers, eliminating the need for plugins [12]. Projects like moonlight-web-stream demonstrate a practical bridge between high-performance streaming hosts (e.g., Sunshine) and web clients using a Rust-based WebRTC pipeline, proving the viability of a browser-native approach [13]. Concurrently, research into real-time communication via JavaScript has addressed synchronization challenges critical for implementing responsive social features like chat and notifications in a browser context.

C. Identified Gap and Contribution of this Work

Despite these advances, a significant gap remains: commercial platforms prioritize streaming performance but treat social features as an external, fragmented layer, while technical research often optimizes streaming protocols or social features in isolation. Current models that rely on separate DOM composition for social overlays or external applications incur inherent latency and resource overheads that are poorly quantified. This work directly addresses this gap by proposing and evaluating the Hybrid Overlay architecture. This architecture is designed to enable a principled comparison by co-rendering social interfaces directly onto the video stream and centralizing input management, with the explicit goal of minimizing the overhead that current decoupled approaches introduce.

III. Proposed System Design

This section details the architecture, development methodology, and core specifications of the proposed cloud gaming platform including the system's three primary modules: the frontend streaming client, the backend signalling server, and the integrated social layer.

A. Development Requirements

System requirements were derived from an analysis of existing platforms and core objectives. Functional requirements include secure user authentication (via Firebase), low-latency game streaming using WebRTC, a real-time community hub with chat and friend management, and robust session handshake handling. Non-functional requirements include minimal client resource consumption; scalable Firestore database performance, compatibility with modern WebRTC-supported browsers, and comprehensive security via HTTPS/WSS, DTLS/SRTP, and Firebase Role-Based Access Control (RBAC) rules.

B. System Architecture

The platform's logical flow and component interactions are modelled in Figure 2 (System Flowchart).

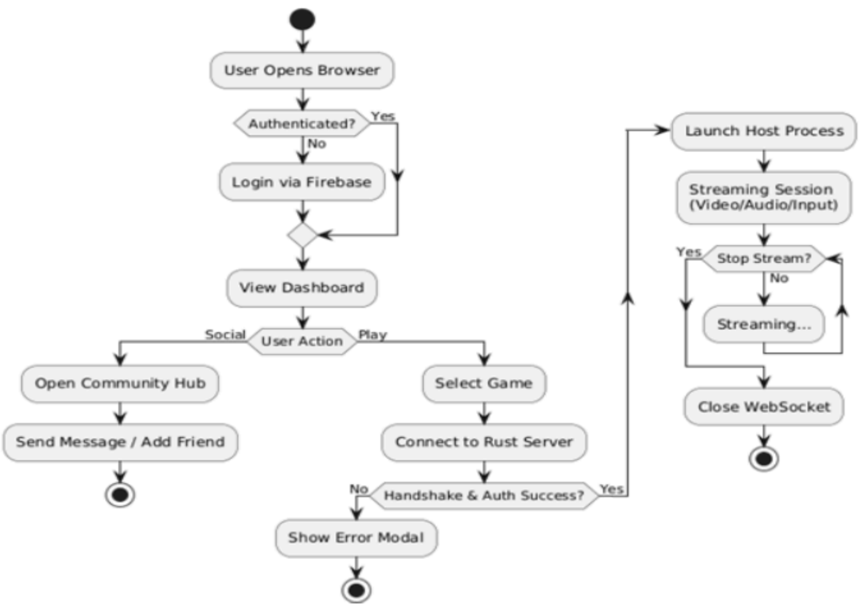

Fig. 2 System Flowchart

https://doi.org/10.31436/ijpcc.v12i1.678

A user authenticates via Firebase before accessing the main dashboard. From there, they may enter the Community Hub for social interaction or initiate a game stream. To start a stream, the client establishes a WebSocket connection to the Rust signalling server, which authorizes the session and orchestrates a WebRTC handshake with the remote Game Host. Upon successful negotiation, a peer-to-peer media channel is established for video/audio streaming and input relay.

The Rust server spawns a dedicated Streamer child process, which configures the Game Host's capture hardware. The server then mediates the exchange of Session Description Protocol (SDP) offers and answers between the Streamer Process and the Web Client, followed by Interactive Connectivity Establishment (ICE) candidate exchange to establish a direct UDP data channel.

D. *Data Model*

The system employs a NoSQL data model implemented in Firebase Firestore, structured to support real-time social features and session management (see Entity Relationship Diagram, Figure 3). Core collections include: USERS (authentication and profiles); FRIENDS and FRIEND_REQUESTS (social graph); MESSAGES and GROUPS (communication); and SESSIONS with GAME metadata (stream management).

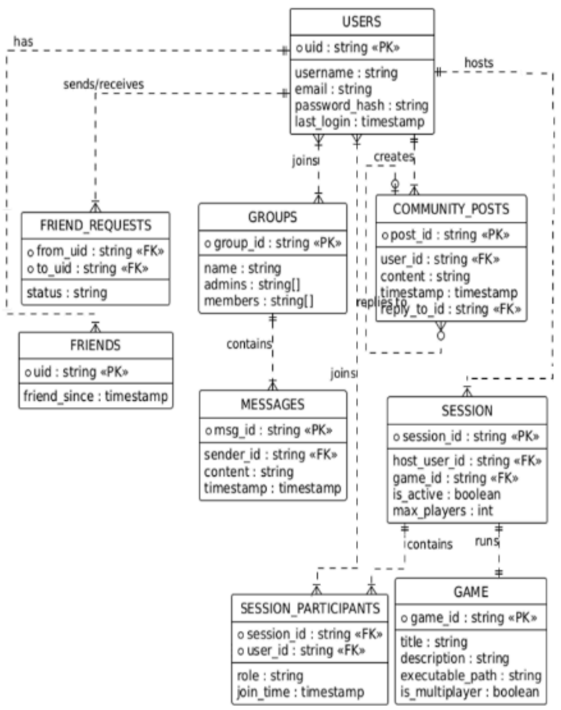

Fig. 3 System Flowchart

A multi-layered security design protects data and communications:

1) Transport Security: All signaling traffic (WebSocket, HTTP) is encrypted via TLS (HTTPS/WSS).
2) Media Security: The WebRTC peer-to-peer stream is secured using Datagram TLS (DTLS) for key exchange and the Secure Real-time Transport Protocol (SRTP) for audio/video encryption.
3) Data Access Control: Firebase Security Rules enforce RBAC, isolating user data and ensuring users can only modify their own profiles and authorized group content.

The system architecture is modular, comprising a signalling backend, a client-side hybrid overlay, and a real-time social data layer (see Figure 4).

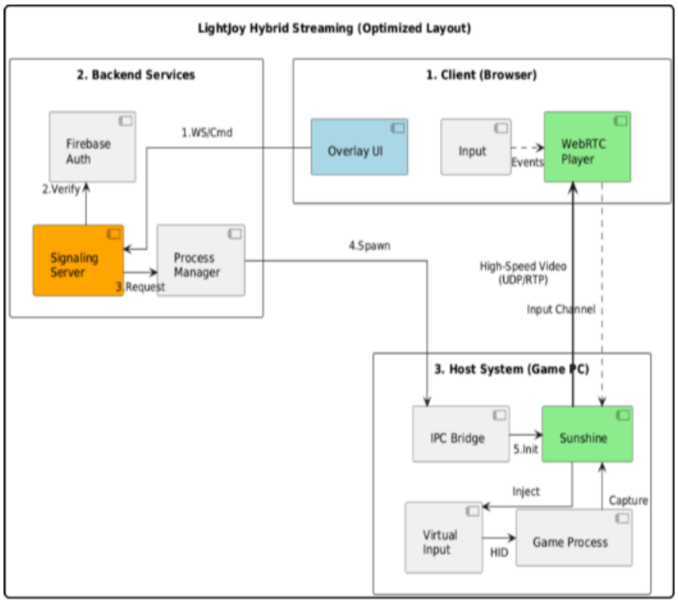

Fig. 4 System Flowchart

The implementation integrates three core technological layers: The Signalling & Streaming Layer uses the Sunshine encoder on the host side, with orchestration managed by a custom Rust signalling server (moonlight-server) responsible for WebRTC session negotiation, host selection, and spawning subprocesses to handle the raw UDP media stream. A key innovation is the client-side Hybrid Overlay Architecture (viewer.html, stream.ts), which renders HTML5 social interfaces directly atop a hardware-accelerated WebGL video stream. This architecture employs precise CSS z-index stacking to position a DOM-based "Community Sidebar" over the video canvas and incorporates an Input Isolation Engine that intercepts focus events; when the chat sidebar is active, event.stopPropagation() prevents keystrokes from being forwarded to the game host, enabling seamless simultaneous gameplay and

https://doi.org/10.31436/ijpcc.v12i1.678

communication. Finally, the Social Engine (Data Layer) is powered by a custom module (comm.js) interfacing with Firebase Firestore, enables instant messaging through onSnapshot listeners, and handles the complete lifecycle of friend requests including sending, acceptance, and rejection.

The backend signalling server was implemented in Rust, utilizing the Actix-web framework and Cargo package manager. This language was selected for its guaranteed memory safety and high-concurrency capabilities, which are critical for managing numerous simultaneous WebRTC handshakes. The client frontend was built with TypeScript and native HTML5 DOM APIs, deliberately avoiding heavier frameworks to minimize overhead and ensure responsive performance on low-end hardware, with development conducted primarily in Visual Studio Code. For data persistence and real-time synchronization, Firebase Firestore served as the NoSQL backend database.

During integration, key technical challenges were encountered and resolved. Overlay Input Interference, where keyboard inputs affected both the game stream and chat interface, was solved by implementing a client-side Input Isolation Engine that uses event.stopPropagation() to contain keystrokes within the focused social overlay. Additionally, Z-Index Layering conflicts between the HTML DOM and the WebGL video canvas were addressed by enforcing a strict CSS stacking context, assigning the canvas a z-index of 0 and the overlay a z-index of 10 to ensure correct visual compositing.

Security is enforced through a multi-layered approach. All signalling communication between the client and server is protected by Transport Layer Security (TLS) via HTTPS and WSS connections. The WebRTC media stream itself is secured using Datagram TLS (DTLS) for key exchange and the Secure Real-time Transport Protocol (SRTP) for encrypting the audio and video payloads. Data integrity and privacy within the social features are maintained through Firebase's Role-Based Access Control (RBAC) with security rules restrict database write operations, cryptographically scoping them to the authenticated user's session ID (request.auth.uid) to prevent unauthorized data modification. Finally, the client-side Input Isolation Engine functions as a critical security sandbox, ensuring that user interactions with the social overlay cannot inadvertently leak into the remote desktop control pipeline, thereby mitigating a core risk in browser-based remote access systems.

IV. System Evaluation

This section presents the operational outputs and performance evaluation of the deployed platform, covering administrative interfaces resource utilization, and the end-user experience.

A. *Administrative Outputs*

System administration is supported via a backend console and a cloud dashboard.

- *Rust Signaling Server Console:*

To provide real-time diagnostics. Logs confirm server initialization, TLS setup, successful user authentication via Firebase, WebSocket session establishment, and the lifecycle management of the streaming subprocess (see Figures 5-7).

```
[14:00:23] INFO  [main] [Server]: Loading Configuration from ./server/config.json
[14:00:23] INFO  [main] [Config]: Loaded SSL Keys: cert.pem, key.pem
[14:00:23] INFO  [main] [Server]: Running Https Server with ssl tls
[14:00:23] INFO  [actix_server::builder] starting 4 workers
[14:00:23] INFO  [actix_server::server] Actix Runtime is starting on 0.0.0.0:8080
```

Fig. 5 Server Startup Console Output

```
[14:05:12] INFO  [actix_web::middleware::logger] 192.168.1.15 "GET / HTTP/1.1" 200 3452
"-" "Mozilla/5.0"
[14:05:12] INFO  [api::auth] Verifying Token for user: "uid_12345ABC"
[14:05:13] INFO  [api::stream] New WebSocket Session Established. ID: 89a-f22
```

Fig. 6 Client Connection Handshake Console Output

```
[14:06:45] INFO  [Stream]: Request received for AppID: 104
[14:06:45] INFO  [Stream]: launching streamer from path: ./sunshine/moonlight_stream.exe
[14:06:46] INFO  [Process]: Spawning child process: moonlight_stream.exe
[14:06:46] INFO  [WebRTC]: Exchange SDP Offer/Answer
[14:06:46] INFO  [Stream]: Input Isolation Active
[14:07:00] INFO  [Ipc]: ipc receiver is closed
[14:07:04] INFO  [Stream]: killing streamer
[14:07:04] INFO  [Stream]: killed streamer
```

Fig. 7 Streamer Process Lifecycle Console Output

- *Firebase Console:*

The GUI manages non-volatile state. The Authentication dashboard displays registered users and active sessions, while the Firestore inspector shows the data structure for users, groups, and messages, confirming proper data nesting and access patterns (see Figures 8).

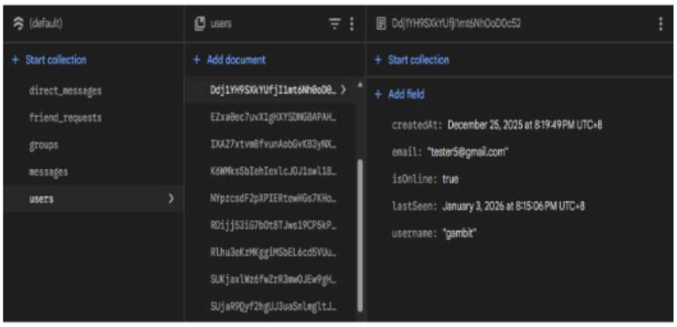

Fig. 8 Firestore users database

B. *Testing Plan*

A comprehensive test plan was executed to validate functional requirements across all modules: Authentication,

https://doi.org/10.31436/ijpcc.v12i1.678

Game Streaming, Social Interaction, and Security. The methodology included unit and integration testing to verify component reliability and end-to-end system behavior.

1. Authentication Module Tests

This module was tested to verify secure user registration, credential validation, and session persistence.

TABLE III
REGISTRATION WITH VALID INPUT

Test Case ID	TC_Auth_001		
Related Feature ID	F001 (Authentication)		
Objective	Verify successful user registration.		
Coverage Items	Req-Auth-01		
Steps	**Expected Result**	**Actual Result**	**Pass/Fail**
1. Navigate to "Sign Up" page. 2. Fill in form with valid credentials. 3. Click 'Register'.	Account is created in Firebase; User redirected to Dashboard.	Registration successful; Dashboard loaded.	Pass

TABLE IIII
LOGIN WITH INVALID CREDENTIALS

Test Case ID	TC_Auth_002		
Related Feature ID	F001 (Authentication)		
Objective	Verify system rejects incorrect passwords.		
Coverage Items	Req-Auth-02		
Steps	**Expected Result**	**Actual Result**	**Pass/Fail**
1. Navigate to "Login" page. 2. Enter valid email but incorrect password. 3. Click 'Login'.	System displays "Invalid credentials" message. Access denied.	Error message displayed; Login prevented.	Pass

2. Game Streaming Module Test Cases

These tests validated the core streaming functionality, automatic system configuration, and the critical input isolation mechanism.

TABLE IVII
STREAMER BINARY AUTO-CONFIGURATION

Test Case ID	TC_Strm_001		
Related Feature ID	F002 (Game Streaming)		
Objective	Verify automatic compilation of streamer binary when missing.		
Coverage Items	Req-Sys-05		
Steps	**Expected Result**	**Actual Result**	**Pass/Fail**
1. Delete streamer.exe. 2. Run web-server executable. 3. Observe console.	Server compiles streamer.exe and updates config.json	Console logs showed compilation and success.	Pass

TABLE VV
START WEBRTC STREAM

Test Case ID	TC_Strm_002		
Related Feature ID	F002 (Game Streaming)		
Objective	Verify successful initiation of video stream.		
Coverage Items	Req-Strm-01		
Steps	**Expected Result**	**Actual Result**	**Pass/Fail**
1. Click "Start Stream" on host. 2. Wait for connection handshake.	Video overlay appears; gameplay is visible.	Overlay opened, video feed visible.	Pass

3. Security Module Tests

These tests confirmed the enforcement of transport security, access control, and system integrity measures.

TABLE VIV
HTTPS ENFORCEMENT

TC ID	**TC_Sec_001**	**TC_Sec_002**	**TC_Sec_003**
Feature ID	F003 (Security)	F003 (Security)	F003 (Security)
Objective	Verify automated generation of self-signed certificates.	Verify server prevents unsecured HTTP connections..	Verify write protection on other user's data.
Steps	1.Remove server/certs. 2. Restart server.	1. Run server with SSL enabled. 2. Attempt to connect via http://.	1.Authenticate as User A. 2. Attempt malicious API write to User B's profile.
Expected Result	key.pem and cert.pem created; HTTPS enabled.	Connection refused or reset (No insecure access).	Firebase Error: Permission Denied.
Actual Result	Certificates regenerated; Server starts on HTTPS.	Browser failed to connect via HTTP (Correct).	Write operation rejected by database rules.
Status	**Pass**	**Pass**	**Pass**

The tests provide empirical validation for key architectural decisions, most notably the efficacy of the Input Isolation Engine in safeguarding the streaming session from interface interference.

https://doi.org/10.31436/ijpcc.v12i1.678

C. *System Performance & Resource Utilization*

Resource utilization was profiled to validate the offloading architecture. Metrics were captured on both the Host (game execution & encoding) and Client (stream decoding) devices.

1) Host System: showed high computational load, with CPU utilization peaking at ~76% and significant memory usage (77%), as expected for simultaneous game rendering and video encoding (see Figure 10).

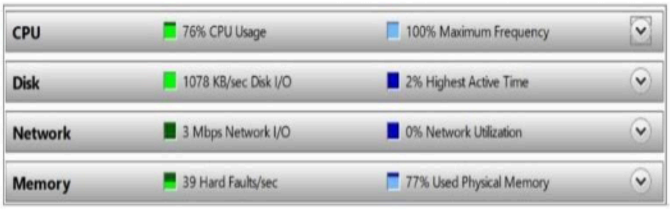

Fig. 10 Host System Performance during active game streaming

2) Client System: Demonstrated efficient low-overhead operation, with CPU utilization averaging ~21% and stable memory usage (68%). Minimal disk I/O and network throughput (~1 Mbps) confirm the client's lightweight role (see Figure 11).

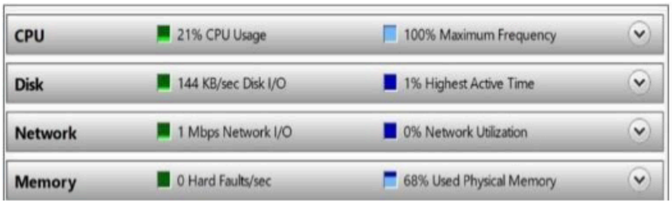

Fig. 11 Client System Performance during stream reception

This disparity (Host CPU load ~3.6x higher than Client) validates the design goal for the intensive processing to be offloaded to the server, enabling high-fidelity gaming on low-end client hardware.

V. Conclusion

This research presented the design and evaluation of a proof-of-concept, browser-based cloud gaming platform that integrates low-latency game streaming with native social integration. The study demonstrates that modern web technologies specifically WebRTC, Rust, and Firebase can be used to implement a functional, interactive gaming experience within a single-host architecture. Key achievements include the implementation of a Hybrid Overlay Architecture, which successfully renders social interfaces directly atop hardware-accelerated game streams without significant performance degradation, and the validation of WebRTC for sub-100ms 1080p/60fps streaming under controlled, real-world conditions. The work acknowledges that its single-host, proof-of-concept design imposes clear limitations on scalability and prevents features such as multi-instance support. Future research should investigate adaptive bitrate algorithms for variable network conditions, dedicated audio-offloading for voice chat, and scalable backend designs using containerization to enable concurrent user sessions. This study provides a foundational implementation model for socially-aware cloud gaming and demonstrates that high-performance, browser-based streaming is technically feasible, offering a pathway toward reducing hardware barriers and improving accessibility in future scalable systems.

Acknowledgment

Authors hereby acknowledge the review support offered by the IJPCC reviewers who took their time to study the manuscript and find it acceptable for publishing.

Conflict of Interest

The authors declare that there is no conflict of interest.

AUTHORS CONTRIBUTION STATEMENT

All authors contributed equally to this work.

DATA AVAILABILITY STATEMENT

There is no external or third-party data that support the findings of this study.

ETHICS STATEMENT

This study did not require ethical approval

REFERENCES

[1] K. Kumar and M. Jha, "Cloud gaming: Redefining the future of entertainment beyond conventional PCs," *Journal of Recent Innovation in Computer Science Technology*, vol. 2, no. 4, pp. 39–51, Oct. 2025, doi: 10.70454/JRICST.2025.20404.

[2] Statista, "Cloud gaming – Worldwide," 2024. [Online]. Available: https://www.statista.com/outlook/amo/media/games/cloud-gaming/worldwide. [Accessed: Jan. 2026].

[3] O. S. Peñaherrera-Pulla, C. Baena, S. Fortes, E. Baena, and R. Barco, "Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks," *Sensors*, vol. 21, no. 4, p. 1387, Feb. 2021, doi: 10.3390/s21041387.

[4] Entertainment Software Association, *Power of Play: Global Report 2023*, Washington, DC, USA, 2023.

[5] NVIDIA Corporation, "GeForce NOW," 2024. [Online]. Available: https://www.nvidia.com/en-my/geforce-now/. [Accessed: Jan. 2026].

[6] J. M. John, "A comparative study on the user experience of PC gaming vs cloud gaming," *EPRA International Journal of Multidisciplinary Research (IJMR)*, pp. 148–152, Apr. 2020, doi: 10.36713/epra4284.

[7] C. Baena, O. S. Peñaherrera-Pulla, R. Barco, and S. Fortes, "Measuring and estimating key quality indicators in cloud gaming services," *Computer Networks*, vol. 231, p. 109808, Jul. 2023, doi: 10.1016/j.comnet.2023.109808.

[8] A. K. Jumani *et al.*, "Quality of experience (QoE) in cloud gaming: A comparative analysis of deep learning techniques via facial emotions in a virtual reality environment," *Sensors*, vol. 25, no. 5, p. 1594, Mar. 2025, doi: 10.3390/s25051594.

https://doi.org/10.31436/ijpcc.v12i1.678

[9] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of QoE in cloud gaming based on subjective tests," in *Proc. 5th Int. Conf. Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, IEEE, Jun. 2011, pp. 330–335, doi: 10.1109/IMIS.2011.92.
[10] M. Manzano, M. Urueña, M. Sužnjević, E. Calle, J. A. Hernández, and M. Matijasevic, "Dissecting the protocol and network traffic of the OnLive cloud gaming platform," *Multimedia Systems*, vol. 20, no. 5, pp. 451–470, Oct. 2014, doi: 10.1007/s00530-014-0370-4.
[11] "WebRTC," in *Multimedia Networks*, Hoboken, NJ, USA: Wiley, 2016, pp. 213–222, doi: 10.1002/9781119090151.ch8.
[12] MrCreativ3001, "moonlight-web-stream," GitHub repository, 2024. [Online]. Available: https://github.com/MrCreativ3001/moonlight-web-stream. [Accessed: Jan. 2026].
[13] J. C. Long and R. J. Toal, "Modeling patterns for JavaScript browser-based games," in *Internet and Multimedia Systems and Applications / 747: Human–Computer Interaction*, Calgary, AB, Canada: ACTA Press, 2011, doi: 10.2316/P.2011.746-018.