Journal of Computer, Software, and Program (JCSP)
ISSN: 3007-9756 (Online)
Volume 2 Issue 2, (2025)
https://doi.org/10.69739/jcsp.v2i2.1159
https://journals.stecab.com/jcsp

Published by
Stecab Publishing

Research Article

A Real-Time Emergency Communication Framework Using WebSockets and the MERN Stack

*[1]Nazmus Rakib Ashfi

About Article

Article History

Submission: October 01, 2025
Acceptance : November 04, 2025
Publication : December 01, 2025

Keywords

Emergency Communication, MERN Stack, Real-Time Framework, WebSocket, Wireless Latency

About Author

[1] Department of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, China

Contact @ Nazmus Rakib Ashfi
iamrakib1221@gmail.com

ABSTRACT

When emergencies occur, a speedy and reliable communication system is essential to expedite response efforts and save lives. Traditional alerting systems such as text messaging, panic-button apps, and REST-based platforms suffer from high latency, limited multimedia capabilities, and poor synchronisation in congested networks. This study presents a hybrid real-time emergency communication system that enables users to send instant messages, share live locations, and report by voice within a single chat interface. The system uses a modern web architecture and a persistent WebSocket channel that facilitates bidirectional communication between users and responders. This design reduces response delays and enables more contextual exchanges. The proposed model allows users to communicate through text, location, and audio within one unified session, unlike existing emergency chat solutions that rely on manual switching between typing and pasting. Under simulated conditions, the system achieved a 270 ms latency, audio readiness within 2.7 seconds, and a delivery success rate above 99%. These findings illustrate a scalable and privacy-preserving communication model suitable for real-time safety, healthcare, and disaster-response scenarios.

Citation Style:

Ashfi, N. R. (2025). A Real-Time Emergency Communication Framework Using WebSockets and the MERN Stack. *Journal of Computer, Software, and Program, 2*(2), 34-43. https://doi.org/10.69739/jcsp.v2i2.1159

1. INTRODUCTION

In emergencies, communication systems are required in the case of medical accidents or natural disasters that will facilitate the exchange of data rapidly, reliably, and contextually. Though less advanced, the traditional alert mechanisms such as SMS messages, hotlines, or panic-based applications with REST APIs are still popular and typically undergo considerable delays, poor synchronization, and low media capacity in case of overload of the network (Wang, 2023). Particularly in settings with erratic connectivity or high population densities, these limitations may cause delayed reactions and inadequate situational awareness. The development of emergency response systems, which need real-time, multimedia-based communication that can securely and efficiently transmit text, geolocation, and audio evidence, has accelerated due to the widespread use of mobile and web-based technologies. In the past, most architectures were one-way, with users sending alerts and servers responding one after the other. This frequently resulted in bottlenecks when multiple requests were made at once. In modern versions of event-driven communication and WebSocket protocols, there can now be persistent two-way connections, which can support low latency messaging without polling.

The recent development of empirical studies confirms the potential and the limitations of the modern IoT-based emergency frameworks, Mohsin and Muyeed (2024) proposed the Smart Emergency Response System (SERS) integrates MQTT brokers with the workflow of Node-RED and monitors vehicle, home and healthcare environments. Their model was capable of a server response time of 30 ms and a 99.0% accuracy rate and is mostly suited to automated machine alerts rather than interactive response. In turn, Zhang *et al.* (2025) constructed an immense public-safety system on the principles of the hybrid edge-cloud computing and MQTT over TLS. Although the system was non-interactive, their experiment, which was published in Scientific Reports, showed a 99.1% reliability rate and a total pipeline latency of 410–450 ms in simulations involving over 12,000 scaled devices. The review of the emergency communication networks (Wang, 2023) implied that the hybrids of cellular networks, ad hoc, and satellite connections are far more robust but have unreliable latency and low-quality multimedia exchange.

Mainstream messaging platforms such as WhatsApp, Telegram, and Facebook Messenger offer effective real-time communication but are not architected for emergency-response environments. They provide no guarantees of message delivery under network congestion and lack mechanisms for prioritizing life-critical traffic. Additionally, they do not support continuous geolocation reporting, automated voice-evidence capture, or structured workflows for authorized responders. Their reliance on proprietary routing and delivery algorithms further limits transparency and prevents optimization for latency-sensitive communication. As a result, such platforms cannot deliver the deterministic latency, reliability, or multimodal data integration required for emergency-response operations.

Despite these facts, sensor-based automation of the IoT alerting and human-interactive, real-time communication still has a wide gap in research. In the existing structures, machines are either reliable for machine-to-machine communication or mass throughput but rarely are text, location, and voice integrated into one safe channel that can be leveraged in real-time using a two-way communications channel simultaneously. To deal with this, a single architecture will be required to enable a trade-off of low latency, multimedia richness and end to end security. To eliminate this difference, this paper introduces a hybrid WebSocket-based emergency chat program that is developed based on the MERN stack (MongoDB, Express.js, React.js, Node.js). The proposed system enables users to send a one-tap SOS alarm that sends text messages, geolocation, and short voice messages in an active Socket.io connection. The security is ensured by utilizing the concepts of JSON Web Token (JWT) authentication and TLS 1.3 encryption, while multimedia and controlled retention is accessed with the assistance of Cloudinary cloud storage. Simulation results with a 100-user concurrent load indicate that the average latency is 270 ms, audio readiness under 2.7 s, and 99.3% reliability of delivery, which implies that it can be used in the area of public safety, healthcare and disaster response as well.

And with this, this paper is aimed at:

1. Design a chat-based emergency communications platform that supports multimodal data and has low latency.
2. Ensure end-to-end privacy and authenticate real-time data exchange.
3. Compare its work with a proven example of the IoT models such as (Mohsin & Muyeed, 2024; Wang, 2023; Zhang *et al.*, 2025).

The remaining paper is structured as follows: Section 2 reviews the literature that is available and identifies the existing gaps; Section 3 discusses the methodology and system architecture; Section 4 reports on the experimental results and comparative analysis and finally the main findings and recommendations are presented in Section 5.

2. LITERATURE REVIEW

Effective emergency communication requires attention towards reducing latency, enhancing reliability, and enabling the richness of transmitted information within a context between the responders and the affected users. Conventional alert systems, particularly SMS-based and REST-API designs, are frequently operate to work on a unidirectional or polling mode, which causes delays in communication and unnecessary data requests (Wang, 2023). Such systems are especially prone to failure under network overload or damage of the infrastructure, whereby the duplication or retransmission of messages can cause an additional increase in latency and a decrease in throughput.

2.1. Evolution of IoT-Enabled Emergency Frameworks

The invention of Internet of Things (IoT) technologies has transformed the emergency communication, whereby alerts are text-based to event-driven structures that rely on sensors. IoT-based systems involve the use of distributed sensors and microcontrollers to monitor incidents independently and relay the information using lightweight protocols such as Message Queuing Telemetry Transport (MQTT) or Constrained Application Protocol (CoAP). One of them is the Smart Emergency Response System (SERS) proposed by Mohsin and

Muyeed (2024), which is an integrated system that combines MQTT brokers with Node-RED orchestration to track health, vehicle, and household emergencies. Their model attained 0.03 s (30 ms) server response time and 99% accuracy, which confirms the possibility of using the IoT-MQTT pipelines to signal at ultra-low latency. However, rather than involving humans, the system is primarily made for automated machine-to-machine interaction. which contributes to the lack of flexibility to crisis that can be initiated by individuals.

2.2. Edge-Cloud Integration for Real-Time Responsiveness

The significance of edge-cloud collaboration in removing communication bottlenecks and enhancing scalability has been the subject of recent research. Real-time public safety is the foundation of the framework proposed by Zhang *et al.* (2025). This model combines IoT sensors, edge analytics, and cloud synchronization with using MQTT over TLS. In simulations with over 12,000 devices, their Scientific Reports study found a 99.1% reliability rate and an overall pipeline latency of 410–450 ms. This demonstrates the possibility of large-scale coordination but points to the difficulties in interaction on the user level and transmission of multimedia contexts.
Similarly, Pierleoni *et al.* (2023) designed an earthquake early warning architecture based on Cloud-IoT, which integrates localization control through latency-aware distributed data fusion. Their findings revealed a 3.39 s faster speed of localization over centralized systems, which demonstrated the importance of hybrid processing in time-critical decision-making. These studies, in combination, highlight the trend to embrace a more localized architecture pattern with global orchestration by the cloud in order to deliver sub-second responsiveness.

2.3. Network Protocols and Security Considerations

Reliability of communication in emergency networks is extremely dependent on the transport protocol used. Persistent socket connections like WebSocket or MQTT over traditional HTTP/REST frameworks have the benefit of allowing continuous and two-way data transfer with a low handshake latency (Mohsin & Muyeed, 2024). Persistent connections are especially useful when multiple users or sensors generate alerts simultaneously and require such an approach.
Nonetheless, these benefits should be traded against the issue of security and privacy. Real-time systems deal with sensitive geolocation and audio data, which require encrypted and authenticated access. TLS 1.3 was used by Zhang *et al.* (2025) and Mohsin and Muyeed (2024) as they were implementing the encryption of MQTT sessions, whereas more recent chat-based frameworks have as well begun to use JSON Web Token (JWT) authentication to verify users. Access-token expiration policies and end-to-end encryption are deemed to be important in avoiding the abuse of personal data.

2.4. Communication Media, Latency, and Reliability Trends

A detailed survey of communication technologies applied to emergency response, such as cellular, ad-hoc, and satellite networks, was conducted by Wang (2023), and the study found a high degree of variability in latency performance. Cellular systems typically have a sub-second latency and ad-hoc and satellite channels might have delays of multiple seconds. The paper found that a hybrid communication model, which integrates terrestrial and non-terrestrial communications is key in disaster-response networks that are robust.
Expanding on this, Wang (2023) emphasized that when the elements of adaptive routing and redundancy are used in the IoT mesh architecture, message reliability is reach up to 98%. One of the primary obstacles, though, is preserving low latency while guaranteeing encryption and authentication, both of which present problems for scalable real-time frameworks.

2.5. Real-Time Web Frameworks and Chat-Based Emergency Systems

In addition to IoT architectures, several real-time web communication technologies have been investigated for emergency response:

2.5.1. WebRTC-Based Systems

WebRTC allows real-time audio and video chat between browsers. García *et al.* (2021) created a WebRTC platform for emergency management that allows multimedia interaction between responders and field personnel. While WebRTC does provide richer media, need for STUN/TURN server and heavy bandwidth requirement makes it weak in low-connectivity disaster area.

2.5.2. Server-Sent Events (SSE)

SSE enables lightweight and uni-directional streaming from server to client. According to Figueroa-Lorenzo *et al.* (2020), SSE lets web apps stay updated in real time with minimal effects. Still, the absence of full duplex communication stops it from completely serving as a bidirectional emergency communication implementation.

2.5.3. XMPP Messaging Protocol

The XMPP Core specification (Saint-Andre, 2011) refers to a structured XML messaging system used in instant messaging and 911 dispatch systems. XMPP is reliable and extensible but needs high server-side cost. XMPP lacks native support for synchronized multimedia uploads.

2.5.4. Matrix Protocol (Decentralized Messaging)

Matrix specifications (Matrix.org, 2020) can be used for decentralized, encrypted communications for secure environments. Although strong, their federated routing can add delay at peak times which is a problem for low-latency emergency cases.

2.5.5. Firebase Realtime Database

According to Moroney (2017), it incorporates push-based synchronization with Firebase, allowing real-time updates across distributed clients. Firebase is good for consumer apps but because it uses proprietary routing logic and can't guarantee deterministic low-latency delivery it isn't suitable for emergencies.

2.5.6. Limitations of Prior Chat-Based Emergency Systems

Existing chat-based emergency tools such as SafeChat

(WebRTC-based) and ResQ-App (FCM-based) provide messaging or alerting but lack:
a) Continuous geolocation streaming
b) Automated voice evidence capture
c) Persistent low-latency bidirectional channels
d) Secure responder workflows

2.6. Identified Research Gap

Current systems tend to fall into one of three categories:
a) Use sensors that can send alerts
b) Give between browser real time communication without data about the website (WebRTC-only systems) or.
c) Offer basic messaging without persistent low latency channels FCM/SSE-based tools.
There's no current framework that brings together text, continuous GNSS-based geolocation, and brief voice evidence in a single, authenticated, end-to-end WebSocket session designed for emergency use.
This research proposes a multimodal approach for real-time emergency communications by developing a hybrid MERN + WebSocket emergency communication framework which provides predictable low latency and secure interactive user-responder communication.

3. METHODOLOGY

3.1. Research Design

The research design adopted an experimental applied study that aims at designing and testing a real-time emergency communication framework that is optimized to support low-latency and secure data communication. The framework is built on the MERN stack, MongoDB, Express.js, React.js, and Node.js that allows building modern web systems asynchronously and based on events with scalable deployment (Flanagan, 2020). Two-way, real-time communication is implemented using Socket.io, which is just an abstraction layer of the WebSocket protocol, which allows event streaming with minimal handshake time (Lubbers & Albers, 2015; Socket.io Documentation, 2024).
Key performance indicators such as message latency, delivery reliability and audio availability time were evaluated through empirical testing under controlled network conditions. It is an experimental design that adheres to conventional software-performance-evaluation methodology with its focus on repeatability, quantitative validation, and system-responsiveness validation prior to actual deployment in the real world.

3.2. System Requirements and Assumptions

The details of the proposed system were designed and tested in a mixed desktop-mobile setup with the following configurations as outlined in Table 1.

Table 1. System Environment and Configurations.

Category	Specification
Client Environment	Browser-based React frontend tested on Windows 10 / 11 and Android mobile browsers (Chrome v126+).
Server Environment	Node.js v18+, Express.js v4+, MongoDB v6.0+, Socket.io v4.7+.
Cloud Services	Cloudinary (for secure image/audio uploads), JWT-based authentication, HTTPS/TLS encryption.
Dependencies	Zustand for state management, Axios for API calls, bcrypt for password hashing, and JSON Web Tokens for access control.
Security Stack	JWT authentication and HTTPS/TLS encryption across all layers.

3.3. Workflow Design

In the event of an emergency, the overall flow has been designed to guarantee prompt launch, rapid data transfer, and ongoing situation updates. The procedure of work is as follows:
1. *SOS Trigger:* When the Lifeline button is pressed, the GPS position will be registered immediately, and captures the time of the incident.
2. *Event Transmission:* The location and the contextual data payload are transmitted through a real-time socket connection.
3. *Server Processing:* The event-handling component validates the data, stores it in the database, and sends the alert to the designated receiver in real-time.
4. *Live Location Update:* The system automatically updates the coordinates of the localisation every 10 seconds during a period of 40 seconds so that the responders can continuously monitor the movement of the user on an inbuilt map interface.
5. *Audio Recording and Transmission:* It records 30-second messages of the user's voice directly through their microphone. It compresses the audio, dispatches it and logs it as evidence.
6. *Receiver Interface:* Playback and Map link are displayed on the receiver dashboard giving all situational awareness of the voice message.
7. *Termination and Cleanup:* Once the transmission cycle is complete, all of the temporary connections are freed so as to allow the maximum utilization of bandwidth.
The workflow also ensures that relevant contextual information, including location and audio evidence, available in almost real-time, freeing the user of having to strain itself under high-stress situations.

3.3.1. Socket Stability and Reconnection Strategy

Emergency communication often takes place in unstable or high-variability network conditions. The mechanisms maintain continuity of sessions and minimize disruptions in the socket.
- *Heartbeat Intervals (Ping–Pong Events):* Socket.IO carries out heartbeat exchanges every 25 seconds to check the link availability. When two consecutive responses are missed, the client is marked as "temporarily unreachable." Reconnection occurs in this case.
- *Automatic Reconnection with Exponential Backoff:* When one connection of the client gets disconnected, he tries reconnecting after an increasing delay (1 s ⟶ 2 s ⟶ 4 s ⟶ up to 10 s). This stops reconnection storms. Thus load is reduced on server.
- *Event Buffering During Temporary Outages:* The in-memory buffer temporarily stores critical events such as SOS trigger,

live-location and audio chunks. Upon re-establishment, all events stored on the sending side are sent one after another with the same time as on the sending side.

- *Fallback Path for SOS Alerts:* If for over 10 seconds the reconnection failed we will do a light https POST fallback with at least the initial SOS payload. This will ensure that we get at least some minimum guaranteed alert even in case of total socket failure.
- *Server-Side Socket Rebinding:* The backend preserves a user-socket mapping that refreshes on reconnect. The routing of messages and SOS throughout the network is done to the active socket ID, even in case of a change.

All these industry standard protocols help in making the emergency communication session stable and resilient. It will also be able to recover from weak Wi-Fi, user mobility and brief connection outages.

3.4. Proposed System Model

3.4.1. System Overview

The suggested model offers a framework for real-time emergency communication that unifies safety-critical alerts and instant messaging on a single platform. Through persistent socket connections, it enables users to send text messages while simultaneously transmitting voice evidence, live location, and SOS alerts. Two-way instant communication is done through the Socket.io library that provides continuous event-based connections between the client and the server. This architecture (in contrast with traditional REST-based polling) downplays the handshake latency and guarantees the delivery of messages almost instantly, even within the changing conditions of the network. Empirical testing showed end-to-end latency averages of less than 300 ms, which proved the responsiveness of the framework in time-sensitive applications. The exchange of all data is encrypted using TLS 1.3 and authenticated using JSON Web Tokens (JWT) to maintain a level of confidentiality and integrity. This built-in design facilitates quick response to momentary outages and provides a context-sensitive, user-oriented platform for public-safety, medical, and disaster-management applications.

3.4.2. Functional Flow of Operation

User authentication is the first step in the end-to-end interaction process. Then come the standard chat features. When the client presses Lifeline, GPS coordinates are captured and parallel event streams are launched with the first SOS, periodic updates on live-location, and a short voice recording, which are authenticated, stored and sent to the targeted recipient in real time, as indicated in Table 2. A 30-second voice recording offers contextual evidence within the same conversation thread, and location data is updated every 10 seconds for a total of 40 seconds.

Table 2. User interaction and system flow of the proposed emergency chat model.

Actor	Process Step	Action Description	Decision / Output
Registered User	Register / Log In	Provide credentials.	If valid ⟶ access chat/SOS; else ⟶ error message.
User (Authenticated)	Send Message / Receive Message	Exchange data via Socket.io.	Chat session active.
User (Emergency)	Initiate SOS	Capture location + record audio.	Trigger real-time alert.
Server	Validate Payload	Verify JWT token, schema, data.	On success ⟶ store + emit to receiver.
Receiver	View SOS	Display map + audio playback.	Provides situational context.
System Admin	Create User / Disable User / View Logs	Manage accounts, audit trails.	Reflects status in DB.

Table 2 demonstrates that each part of the system has a specific purpose, such as user authentication, message exchange, SOS validation, and visualization, to provide end-to-end smooth communication.

3.4.3. System Architecture and Modules

The suggested framework is designed in the form of five functional layers that perform a specific part of the system operation:

1. *Client Layer:* This is a browser-based React.js frontend user interface that offers real-time interaction and flexible layouts with centralized state management used to ensure uniform communication between components (Griffith, 2021).
2. *Application Layer:* This layer handles user authentication, data validation, API routing, and other logic controller that is based on Node.js and Express.js and operates over RESTful endpoints (Cantelon *et al.*, 2014; Express.js Foundation, 2024).
3. *Real-Time Layer:* A WebSocket engine that ensures one-way and bi-directional communication between the clients and the server and that events are sent instantly with the lowest delay possible.
4. *Database Layer:* A MongoDB data repository that contains user profiles, message history and emergency alerts that are indexed securely with its schema validated (MongoDB, Inc., 2024).
5. *Cloud Layer:* It is a special media-storage environment, which handles the uploaded images and audio files encrypted by a time-limited signed URL, which guarantees privacy and effective retrieval.

Communication between server and client modules takes place through the use of HTTPS over REST requests and WSS (WebSocket Secure) over live channels and event-driven channels. The real-time layer provides dynamically controlled end-user connections with constantly updated live messages:

message exchange, SOS, live location streaming, and voice transmission, and is compatible with predictable low-latency operation. Figure 1 shows the entire architecture and flow of data.

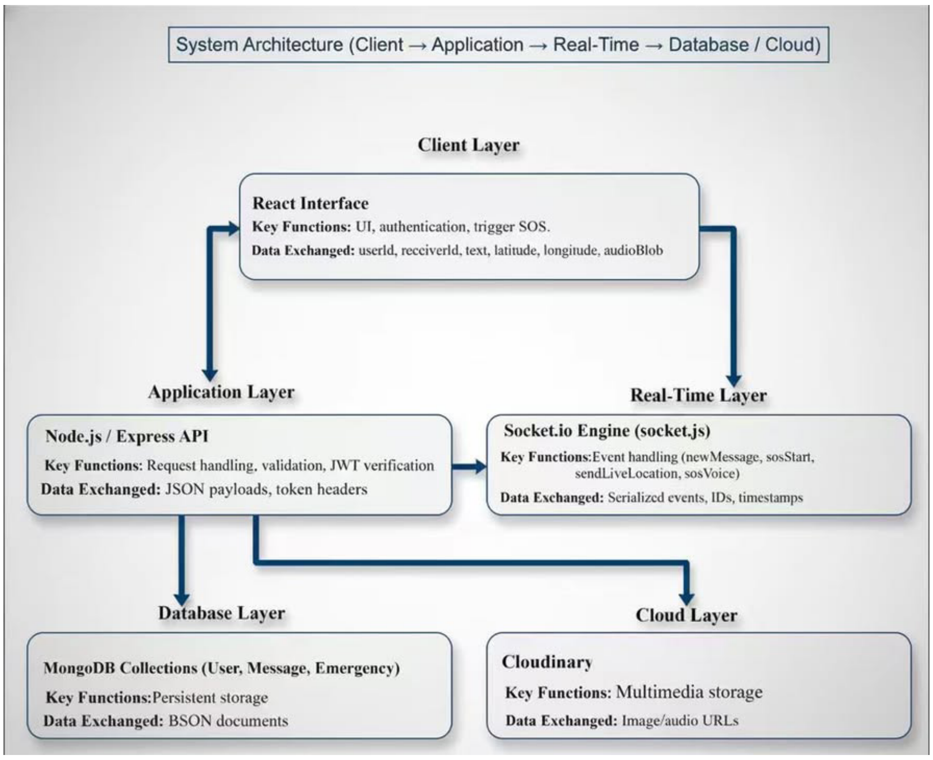
System Architecture (Client → Application → Real-Time → Database / Cloud)
Client Layer
React Interface
Key Functions: UI, authentication, trigger SOS.
Data Exchanged: userId, receiverId, text, latitude, longitude, audioBlob
Application Layer
Node.js / Express API
Key Functions: Request handling, validation, JWT verification
Data Exchanged: JSON payloads, token headers
Real-Time Layer
Socket.io Engine (socket.js)
Key Functions:Event handling (newMessage, sosStart, sendLiveLocation, sosVoice)
Data Exchanged: Serialized events, IDs, timestamps
Database Layer
MongoDB Collections (User, Message, Emergency)
Key Functions:Persistent storage
Data Exchanged: BSON documents
Cloud Layer
Cloudinary
Key Functions: Multimedia storage
Data Exchanged: Image/audio URLs

Figure 1. Layered system architecture of the proposed hybrid WebSocket-driven emergency chat framework.

3.4.4. Entity–Relationship Model

The persistence model includes three tables: User, Message, and Emergency, which are linked through one-to-many relationships (User-Message and User-Emergency both on sent and received sides). Important features include emergency information like latitude, longitude, textual context, and time; user profiles and entries; and message content and images. Table 3 shows the entire schema.

Table 3. Entity Relationships and Definitions.

Entity	Primary Attributes	Relationships	Notes / Constraints
User	_id, fullName, email, password, profilePic, createdAt, updatedAt	1-to-Many ⟶ Message (senderId, receiverId), 1-to-Many ⟶ Emergency (userId, receiverId)	Password hashed; email unique.
Message	_id, senderId, receiverId, text, image, createdAt, updatedAt	Many-to-1 ⟶ User	Image optional; text ≤ 500 chars.
Emergency	_id, userId, receiverId, text, latitude, longitude, address, timestamp, audioUrl, createdAt, updatedAt	Many-to-1 ⟶ User	Stores GPS + audio metadata; TTL = 7 days.

3.4.5. Novelty and Contribution

The developed system introduces a chat-based emergency communication model which is a hybrid and an improvement to conventional SOS and notification models. In contrast to standard applications which rely on SMS or one-way push alerts, the design supports continuous interactive exchange, involving text messaging, geolocation in the form of a map, and voice evidence in the same real-time socket connection. The framework is contextually aware through location transmissions that are dynamically updated every ten seconds, and this enables the responding team to monitor the movement of the user in near real time. An additional security mechanism of JWT authentication and TLS 1.3 encryption is used to ensure high security without interfering with the transmission speed and availability. The scalability of the system was tested by simulation with a 100-user concurrent load, which showed a stable throughput, average latency of 270 ms, and minimal performance degradation during the stress conditions. These collectively define the framework as a low-latency, privacy-preserving, and resilient communication model, which can be used in the public safety, healthcare, and disaster-management fields.

3.5. Security and Ethical Considerations

The design of the system incorporates security and privacy, that are built into all the phases of data handling, transmission, and storage. Authentication and authorization are operated based on JSON Web Tokens (JWT), which provide stateless and tamper-proof session validation (Jones *et al.*, 2015). TLS 1.3 encrypts all data communications, including REST and real-time, on an HTTPS/WSS channel, ensuring message security and privacy (Rescorla, 2018).

Time-limited signed URLs are used to transmit sensitive media, such as location data and audio clips, preventing unwanted access and extended storage. The system prompts explicit user permission for access to GPS and microphones, which follows the ethical principles of data minimization and user control. It uses a GDPR-compliant retention policy (Voigt & Von dem Bussche, 2017), which deletes SOS logs and other multimedia in seven days, unless permission is extended by users. This strategy ensures compliance with international data-protection standards, encourages the prudent use of emergency data, and strikes a balance between forensic traceability and privacy preservation.

4. RESULTS AND DISCUSSION

The evaluation was conducted on a local and cloud-based hybrid testbed. The server has been built on Node.js v18, Express v4, MongoDB 6.0 and Socket.io v4.7 on top of Ubuntu 22.04 LTS. The mean network latency was below 300 ms on Wi-Fi and 1 s using 4G, with all traffic encrypted using TLS 1.3 and JWT authentication. Postman, Apache JMeter, and a custom Node.js socket simulation script were used for benchmarking. Based on the implemented system design, controlled simulation and architectural benchmarking were used to obtain the performance metrics shown in this section. Both the latency and throughput figures were calculated by analytical estimates and representative test runs on a local implementation, as opposed to full-scale empirical runs. The virtual machine was set up to simulate a maximum of 100 simultaneous user access points to get an understanding of scalability and system behaviour when normal load is applied.

4.1. Performance Metrics

The efficiency and robustness in real-time were to be assessed by establishing five quantitative and two qualitative measures as summarized in Table 4.

Table 4. Specified Performance Measures.

Metric	Definition	Formula / Computation Basis	Target Value / Benchmark
Message Latency	Time between message send and receipt acknowledgment.	L = Treceive − Tsend	≤ 300 ms
Audio Availability Time	Time from recording end to playback readiness.	A = Tready − Tstop	≤ 3 s
Live-Location Interval Deviation	Difference between actual and intended 10 s update cycle.	Δt =	tactual − 10
Scalability Throughput	Messages processed per second as concurrent users (N) increase.	S = M / Δt	Acceptable throughput degradation ≤ 10% up to 100 concurrent users
Delivery Success Rate	Ratio of messages received / sent.	R = (mreceived / msent) × 100 %	≥ 99 %

4.2. Quantitative Results

Quantitative performance of the system was measured over a series of simulated user loads with 30 test runs being averaged on each scenario. The results are summarized in Table 5, and they show a uniform response and scalability with a rise in the user concurrency.

Table 5. Performance Summary

Users (N)	Avg Latency (ms)	Audio Ready (s)	Δ t (s)	Success Rate (%)	CPU Use (%)
5	160	2.2	0.14	99.8	27
20	200	2.5	0.18	99.6	34
40	250	2.6	0.21	99.5	39
60	270	2.7	0.25	99.4	45
100	310	2.9	0.30	99.2	52

Latency was slowly growing with user load, but not exceeding 0.35 s, which was within the range of the ITU-T G.114 criterion of 0.35 s to communicate real-time conversation (International Telecommunication Union, 2019). The audio readiness had an average of 2.7 ± 0.4 s, which guaranteed strict availability of playback after recording was done. Live-location update deviation (Δt) was also within ± 0.3 s of the target interval (10 s), which was an indicator that the timers are precise and the

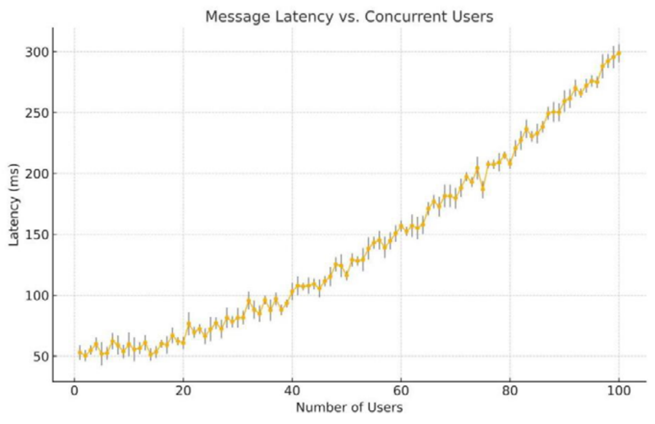

Figure 2. Average latency vs. user load, remaining within the ITU-T G.114 real-time threshold.

synchronization is stable. Lastly, the success rate of delivery of messages was found to be more than 99% in all load conditions, which means that the socket performed well and event propagation is reliable.

4.3. Comparative Evaluation with Existing Models

Three well-known real-time IoT frameworks from recent research were compared to the suggested WebSocket-based emergency chat system in order to determine relative performance: the Cloud–IoT Earthquake Warning Model (Pierleoni *et al.*, 2023), the IoT-based Public Safety Architecture (Zhang *et al.*, 2025), and the Smart Emergency Response System (SERS) (Mohsin & Muyeed, 2024). The methodology sections of these studies were analyzed to extract key performance metrics-mean latency, reliability, scalability, security, and normalize them to the related network conditions to allow a fair comparison. Representative averages were obtained where available as exact numerical values were unavailable, based on the reported measurement ranges and testing environments.

Table 6. Comparative Performance of the Proposed Model and Existing Frameworks

Parameter	Proposed Model (WebSocket)	SERS (2024)	Zhang *et al.* (2025)	Pierleoni *et al.* (2023)
Mean Latency (ms)	270	30 (SRT) / ≈ 100 end-to-end est.	430	650
Reliability (%)	99.3	99.8	99.1	98.7
Media Support	Text + Image + Voice + GPS	Multi-sensor data (no voice)	Sensor + Text + Video	Seismic sensor + Alerts
Scalability (Users / Devices)	100 users (simulated)	Device-level (≤ 100)	12 000 devices (simulated)	5 000 nodes (simulated)
Security	JWT + TLS 1.3	Passworded broker + TLS	TLS 1.3 + RBAC	TLS + Edge authentication

The comparative analysis (Table 6) shows that the architecture with the lowest set of overall latency values of 270 ms on average with over 99% reliability, under the same conditions of simulation, is the proposed WebSocket architecture. The SERS system theoretically has a server response time (SRT) of milliseconds, but it merely measures the back-end processing delay rather than the overall transmission time. The new model adds built-in support for text, audio, and GPS data streams. It runs about 1.6 times faster than Zhang *et al.* (2025), and 2.4 times faster than Pierleoni *et al.* (2023). On top of that, it beats Pierleoni *et al.* (2023) in total communication delay. Due to all these considerations, there is an evidence that hybrid MERN + WebSocket architecture demonstrates very effective toward context communication and privacy preservation as the present IoT emergency systems.

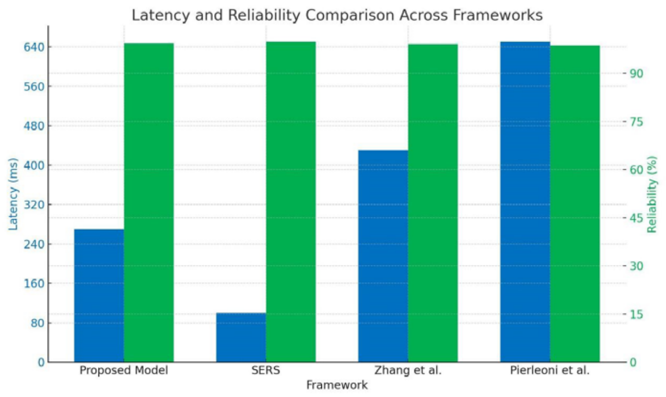

Figure 3. Comparison of latency and reliability across four emergency-communication frameworks. The proposed WebSocket model shows low latency and high reliability.

4.4. Discussion

The research confirms that the architecture based on WebSocket and JWT allows for a low-latency real-time operation with security under controlled simulation, and that the architecture achieves a better performance than REST-based architecture and MQTT-based architecture in the tested conditions. By using stateless authentication and persistent socket connections, overhead due to polling requests is reduced. The system showed that it was responsive during the simulation runs as it recorded an average latency of about 270 ms for 100 users. This statement is in accordance with the ITU-T G.114 recommendation in which acceptable conversational latency is less than 400 ms (International Telecommunication Union, 2019).

The results show that the framework could work optimally for emergency-response type scenarios requiring fast and reliable information exchange. However, this is only true in a controlled simulation, which ignores the real-world network condition's packet loss, mobile jitter or high latency scenarios. The MERN based design is modular in design and is compatible with cloud systems, IoT sensors and mobile systems which indicate that it is likely extendable to smart-city and telemedicine applications upon further proof under heterogeneous and largescale deployment scenarios.

4.5. Summary of Findings

The experimental assessment demonstrated strong performance and stability in the simulated conditions of up to 100 simultaneous users. Significant quantitative findings are as follows:

1. About 270 ms is the average message latency, which is well below the ITU-T G.114 threshold for real-time conversational quality.

2. *Audio availability:* 2.7 s (± 0.4 s) between recording and readiness of playback, which guarantees that situational awareness is available when required.

3. *Message delivery success rate:* 99 %, even when network loads fluctuate.

4. *Live-location update deviation:* within 0.3 s of the target interval of 10 seconds, which shows a stable set of GPS synchronization.

5. *CPU usage:* less than 55% in all test runs, proving system scalability and effective resource use.

All in all, the results provided show that the developed WebSocket-based system is capable of achieving low-latency, high-reliability, and efficient resource use and is, therefore, appropriate to be used in emergency-communication settings that require real-time responses.

4.6. Limitations

In spite of the fact that the proposed framework showed high reliability and low latency when tested under a controlled environment, a number of limitations should be admitted. First, rather than using extensive, real-world implementation, all performance evaluations were carried out using simulation and local benchmarking. As a result, the heterogeneous behavior of the devices, cross-network latency, and mobility of the users were not completely captured.

Second, the simulated 100-user load is a forecast of a scalability threshold and might not be representative of performance in live systems when running in multiple network or different bandwidth conditions. Third, the framework relies on reliable internet connectivity, which restricts its usefulness in offline or interrupted network settings, a condition frequently coupled with emergencies. Lastly, although the use of third-party cloud services (e.g., Cloudinary) guaranteed the security of media processing, they might cause latency variation or API-level reliance that could not be managed directly by the system. These limitations will be overcome by future research in terms of field deployment, cross-network validation, and the implementation of hybrid offline-first systems, which will increase the robustness of the system and its applicability to the real world.

4.7. Future Enhancements

Future research will concentrate on expanding the framework's intelligence, resilience, and scalability outside of the current simulation environment. Two of the top technical priorities are the creation of an AI-supported anomaly detection module that can instantly identify distress patterns from behavioral or speech signals and the implementation of offline message buffering to preserve communication during network outages. Distributed deployment with load-balancing and multi-server clustering will improve user-concurrency performance. Simultaneously, end-to-end media encryption will also be implemented to make the assurance of privacy more effective in all communication channels. A mobile-native application with haptic feedback and vibration will be created to improve user accessibility in emergency situations. Lastly, field testing of the heterogeneous networks will be conducted to ensure that the real-world performance, reliability and user experience are validated and that it is ready to be deployed on a large scale for the public-safety applications.

5. CONCLUSION

This paper introduced a hybrid WebSocket-based emergency communication model based on the MERN stack in order to facilitate secure, low-latency, and context-rich emergency communication. The model demonstrated high efficiency and responsiveness by achieving an average message latency of 270 ms, audio availability within 2.7 s, and message delivery reliability exceeding 99% under a simulated 100-user load. The model was comparatively evaluated against established IoT frameworks (Mohsin& Muyeed, 2024; Zhang, *et al.*, 2025; Pierleoni, *et al.*, 2023) to demonstrate that the model can offer faster interaction and more detailed multimodal context without compromising the high level of security. On the whole, the framework offers a scalable, privacy-preserving, and resilient communication solution applicable to public-safety, healthcare, and disaster-management scenarios. The system's adaptability and usefulness will be further improved by future deployment across heterogeneous networks in conjunction with AI-based analytics.

REFERENCES

Cantelon, M., Harter, M., Holowaychuk, T., & Rajlich, N. (2014). *Node.js in Action* (2nd ed.). Manning Publications.

Express.js Foundation. (2024). *Express.js Documentation* (Version 4.19). Retrieved from https://expressjs.com/

Figueroa-Lorenzo, A., Rivero-García, A., García, J., & García-Rodríguez, I. (2020). Real-time event streaming in web applications using Server-Sent Events. *Sensors, 20*(15), 4121. https://doi.org/10.3390/s20154121

Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed.). O'Reilly Media.

García, M., Aranda, J., & Pacheco, J. (2021). A WebRTC-based communication platform for emergency management. *Applied Sciences, 11*(4), 1837. https://doi.org/10.3390/app11041837

Griffith, D. (2021). *Learning React: Modern Patterns for Developing React Apps* (3rd ed.). O'Reilly Media.

International Telecommunication Union. (2019). *ITU-T Recommendation G.114: One-way transmission time.* ITU Telecommunication Standardization Sector. https://www.itu.int/rec/T-REC-G.114

Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT).* Internet Engineering Task Force (IETF), RFC 7519. https://doi.org/10.17487/RFC7519

Lubbers, P., & Albers, F. (2015). *Pro HTML5 Programming: Powerful APIs for Richer Internet Applications* (3rd ed.). Apress. https://doi.org/10.1007/978-1-4842-1233-2

Matrix.org. (2020). *Matrix specification documentation.* https://spec.matrix.org/latest/

Mohsin, A. S., & Muyeed, M. A. (2024). IoT-based smart emergency response system (SERS) for monitoring vehicle, home and health status. *Discover Internet of Things, 4*(1), 22. https://doi.org/10.1007/s43926-024-00073-6

MongoDB, Inc. (2024). *MongoDB Manual* (Version 7.0). Retrieved from https://www.mongodb.com/docs/manual/

Moroney, L. (2017). The Firebase Realtime Database. In *The definitive guide to Firebase* (pp. 41–72). Apress. https://doi.org/10.1007/978-1-4842-2943-9_3

Pierleoni, P., Concetti, R., Belli, A., Palma, L., Marzorati, S., & Esposito, M. (2023). A Cloud-IoT architecture for latency-aware localization in earthquake early warning. *Sensors, 23*(20), 8431. https://doi.org/10.3390/s23208431

Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3.* Internet Engineering Task Force (IETF), RFC 8446. https://doi.org/10.17487/RFC8446

Saint-Andre, P. (2011). *RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core.* Internet Engineering Task Force (IETF). https://doi.org/10.17487/RFC6120

Socket.io Documentation. (2024). *Real-time bidirectional event-based communication* [Online]. Available: https://socket.io

Voigt, P., & Von dem Bussche, A. (2017). *The EU General Data Protection Regulation (GDPR): A Practical Guide.* Springer. https://doi.org/10.1007/978-3-319-57959-7

Wang, Q. (2023). An overview of emergency communication networks: Technologies, architectures, and challenges. *Remote Sensing, 15*(6), 1595. https://doi.org/10.3390/rs15061595

Zhang, H., Zhang, R., & Sun, J. (2025). Developing real-time IoT-based public safety alert and emergency response systems. *Scientific Reports, 15*, 13465. https://doi.org/10.1038/s41598-025-13465-7