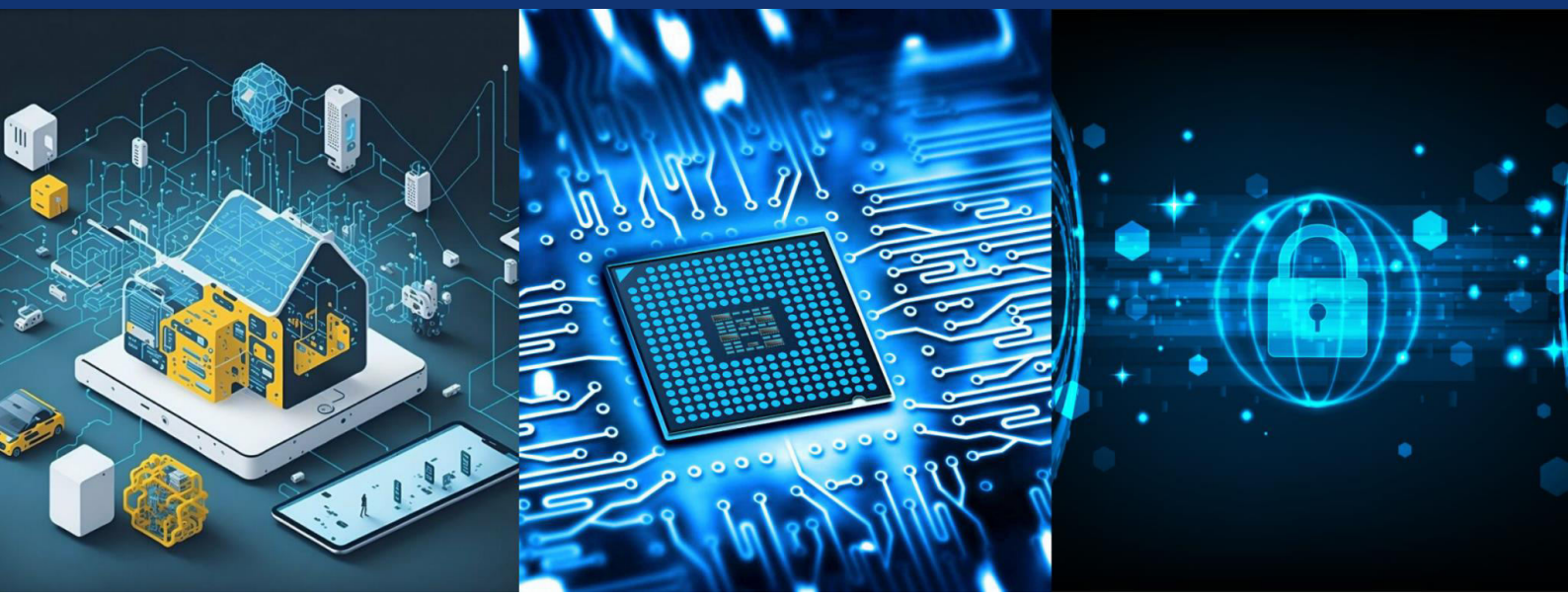
IJIRCCE

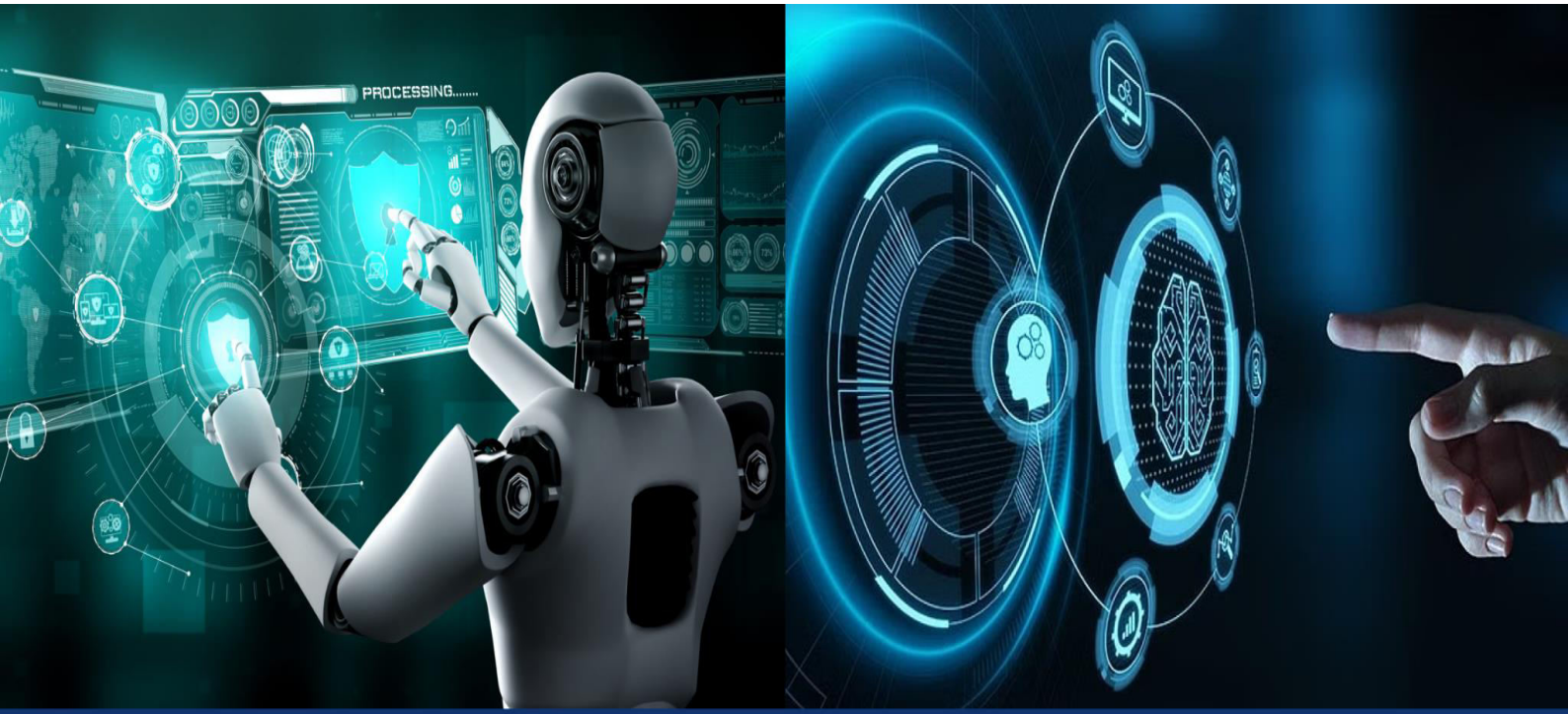
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 6, June 2026

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Design and Implementation of Intranet-Based Voice and Video Calling System over Wi-Fi

Illapu Sankara Srinivasa Rao, Potnuru Jahnavi, Bantupalli Varun Kumar, Thamadapu Shireesha, Kola Sampath Kumar

Associate Professor, Department of Computer Science & Engineering – Data Science, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India

Department of Computer Science & Engineering – Data Science, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India

Department of Computer Science & Engineering – Data Science, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India

Department of Computer Science & Engineering – Data Science, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India

Department of Computer Science & Engineering – Data Science, Anil Neerukonda Institute of Technology and Sciences, Visakhapatnam, India

ABSTRACT: When signals get trapped inside places like college campuses, normal calls and video talks over shared networks tend to drop. Costs climb because links depend on external connections. Suddenly, service fades, cutting words off halfway through speech. Leaks appear as data slips past secure borders into unknown areas. Another way works better - reducing spending yet keeping communication alive even when power fails. When things stay put, away from outside hubs, privacy gets stronger. Inside a closed system made only for certain areas, solutions already exist. Calls shaped by private radio nets keep chats within school borders. One gadget reaches out straight to its partner, bypassing go-betweens completely. Hidden flows protect voice and video as they travel through invisible paths. A quiet change takes shape through subtle shifts: routine tasks follow updated guidelines, tucked away from broad internet pathways. Nearness shapes connections inside the system, where users detect those close by. Devices pair up using a dedicated signal hub that operates independently. Sound and visuals get processed right where they are captured. Information stays within local boundaries, so lag remains minimal. Even with poor internet, links stay strong. Where data runs thin or cell reception fades, it keeps going.

KEYWORDS: Intranet-based communication – Communication within a private intranet network, Wi-Fi calling system – Wi-Fi–based calling system, Peer-to-peer communication – Direct device-to-device communication, Real-time media transmission – Real-time voice and media transmission, Local signaling server – Local server for handling signaling and connections, User discovery – Easy user discovery within the network, Secure communication – Secure and protected communication, Low-latency network – Fast, low-latency network performance.

I. INTRODUCTION

Everywhere now, updates in wireless gear shift how fast folks swap messages. Phones once ruled alone; nowadays, talk rides on cellular links or web platforms - common yet fragile when pressure hits. Even if used nonstop by crowds, these routes frequently stumble over lag, price spikes, spotty flow, or chances of private data slipping toward distant servers. In moments where crisp, reliable chatter is key - say, emergencies - they begin to fray at the edges. As demands grow heavier and doubts stack higher, leaning entirely on public channels seems thinner each time. Out of nowhere, stability shows up when paths less traveled get a try. Starting small changes everything - stronger connections form while depending on others fades away. Quiet moments start doing the work of smoothing things out.

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Not every wireless system needs the internet at all. With Wi-Fi spreading widely, devices within closed areas can now communicate easily. Running on close-knit links brings quicker replies, consistent performance, better control over data. Rather than connecting outwards, tools share information directly through peer-to-peer channels, real-time streams, and private signaling methods. Still, plenty of prebuilt systems rely on web addresses or control centers hidden across faraway machines. This setup fails once complete isolation becomes the top priority.

From within a closed Wi-Fi system, voice and video chats happen only among nearby gadgets. Instead of routing far away, data jumps straight from one device to another using secure tunnels. On-site hardware acts like a quiet guide, helping people connect and launch sessions smoothly. Since signals never leave the building, lag fades, links stay strong, information remains close. Only local paths carry the flow - no detours, no gaps. When public servers aren't involved, risks drop. Even if the internet cuts out, things keep moving without a hitch. Mobile coverage gaps stop mattering - functionality stays intact despite spotty signals. A single closed system holds it all together, making performance steadier. Growth adapts easily to places where connectivity is weak, restricted, or avoided on purpose.

Not leaning on outside systems, this approach demonstrates real-time video plus voice sharing that runs only inside private networks. Building from past work on localized chat methods, it links secure stream layers with smart gadget detection over closed links. Starting a talk session takes fewer steps, cutting lag so talks flow better compared to typical setups. Trials checking endurance during heavy load, privacy strength, and regular use show solid performance when managing your own setup is key. Results point toward usefulness in places where disconnecting improves reliability along with security for ongoing chats.

Here comes a peek at how the rest of the pages are set up.Later on, section two looks at attempts linked to real-time talking formats. A little further ahead, tucked inside section three, sits an audio-visual chat feature complete with core pieces. Flow patterns across the network emerge afterward, laid out carefully by part four. Performance outcomes plus actual efficiency step forward in section five. Near the end, limitations appear alongside final reflections found in section six.

II. WORK RELATED TO THE REAL-TIME COMMUNICATION PROTOCOLS

Instant chats over modern apps rely on speedy, secure transfer of voice, images, and data. With this updated method working solely within internal networks - no outside internet involved - it demands clear paths for gadgets to reach each other via localized radio links. Right inside your browser, WebRTC makes live audio and video jump straight from person to person. Tough encryption tags along every bit, wrapping privacy into how connections form and flow [1].

A. WebRTC Framework Overview

A quiet beginning at Google sparked what would become WebRTC, later shaped by joint work within W3C and IETF. Straight pathways form inside apps or browsers - carrying sound, moving images, even raw information - all without extra software. Cameras and microphones connect through one core part; linking devices happens thanks to another. The third moves streams of info with little oversight, flowing where required. These parts go by specific labels when reading technical notes: getUserMedia(), RTCPeerConnection, RTCDataChannel appear across references [1], [2].

Speedy links. UDP handles that - WebRTC uses it a lot. Live audio, live video, both demand quick delivery. Skipping delayed fixes makes sense here. Rather than hold up everything, packets just keep flowing. Trying again later can mess things up worse in real time. Fast movement of data cuts delay, though losing bits now and then does happen. Timing matters more than flawless arrival for fluid talk. Clear voice holds up better when wait times are short. That's what pushes WebRTC toward UDP at the start. Getting things quickly beats waiting for every piece to show up.

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

B. Core Protocols in WebRTC

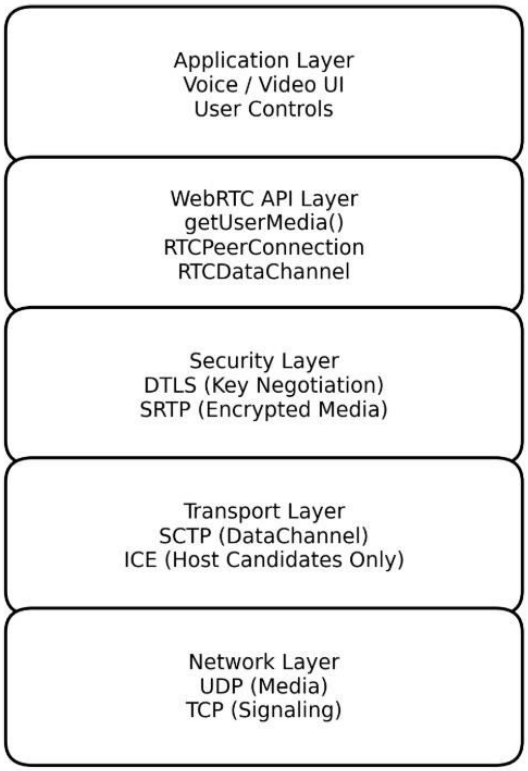


Fig1. Core Protocols in the Flow of Application

1. ICE and Local Candidate Gathering

Looking into WebRTC starts with ICE, which figures out the smoothest way for two gadgets to connect. When locked within a private network like this one, external aids such as STUN or TURN aren't needed at all. One device passes its inner address info right over to the other. Because all units hook into the identical local Wi-Fi structure, contact happens without delays. Paths stay clear since nothing has to jump across boundaries. Seamless ties appear naturally if all pieces live on the same system [3].

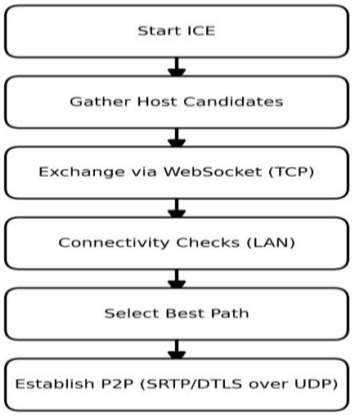


Fig2. ICE Workflow

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

2. DTLS and SRTP Security

Fresh from the ground up, WebRTC locks down media through DTLS key negotiation. Instead of loose connections, it pairs that with SRTP to guard every sound and image. End-to-end shielding rides inside the setup whenever gadgets swap info face to face. These rules aren't guesses - they're laid out in IETF guidelines [4].

Messages that stay secure flow via SRTP over UDP, keeping lag low for real-time audio or video talks [7].

3. SCTP for Data Channels

Flying messages rely on SCTP within WebRTC DataChannels, wrapped by DTLS and flowing over UDP - much like goods carried downstream swiftly. When delivery needs accuracy, data arrives whole; but if pace is key, checks get skipped entirely. Split decisions like these keep texts and remote triggers moving fast, avoiding slowdowns for audio or video feeds in motion [5].

C. WebSocket Signaling Mechanisms

WebRTC does not carry its own way to signal. Something else must step in to exchange SDP info, ICE candidates, initial pings, plus any additional bits. Here that job falls to a WebSocket link - operating over TCP - making sure messages between client and server stay reliable, unbroken, orderly.

Signals need to get through, that is key when starting a call - TCP guards against messages disappearing along the way. Smooth handshakes happen because nothing slips through the cracks. The whole thing moves forward, quietly held together.

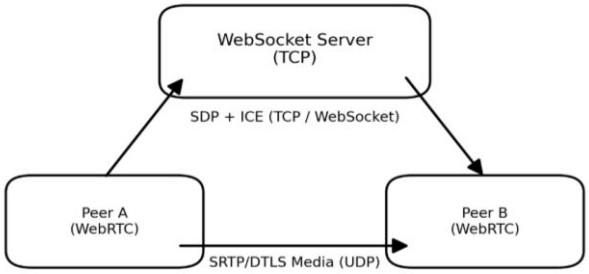


Fig3. Signaling Flow for Establishing WebRTC Connection

D. Relevance to the Proposed System

A foundation forms from the steps described above, setting up live talk and face-to-face links within a close-range network. Direct device to device flow happens without middle points, sound and picture travel fast using UDP, powered by WebRTC, cutting lag while holding clarity firm. Each stream locks down securely through DTLS-SRTP, shielding data with tight coding. Messages ride along the connection neatly, managed by SCTP without hiccups. From time to time, signaling handles session setup and keeps things running through a steady TCP connection via WebSockets. Because of this design, quick private talks happen easily inside a locked Wi-Fi space without touching outside internet links.

III. PROPOSED SYSTEM

A signal jumps between devices without needing the internet at all. One device holds the core software, while nearby gadgets join via short-range radio waves. Rather than depend on external servers, communication happens only within reach of that primary node. Instructions travel as formatted packets, whereas chat flows freely through unbroken channels. Alerts appear instantly since pathways remain active till someone walks away. Faster when messages bounce straight back without detours through distant hubs. Works smoothly even where connections get cut off, because everything stays close by.

A. System Overview

Instant chats on private networks rely on systems that shift voice and visuals quickly, smoothly, yet keep things locked down. Housed within a shut-off Wi-Fi bubble, these calls operate solo - zero public internet required. WebRTC forms

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

the base, linking gadgets straight across, sending voices in protected streams, pulling live footage from lenses, handling handshakes all at once [1], [2]. Setup clues shuttle via a custom WebSocket hub living right inside the system, guiding units to link up, align modes, dodge external pings. With info staying strictly behind internal walls, replies snap faster, spying chances shrink, glitches pop up way less compared to net-tied conversations.

B. System Architecture

Two gadgets able to handle WebRTC take part, sitting beside a lightweight signal hub tucked inside the local network. Each gadget captures sound and picture at once, yet leans on familiar WebRTC features described by W3C rules [2] for setting up talks. Rather than sending single bursts, the hub holds open channels using WebSocket - a way defined in RFC 6455 [8] - to keep information moving freely back and forth. Packets built with Session Description Protocol meet up with ICE suggestions traveling along this live wire, making sure handshakes go smoothly. Right after signals pass back and forth, information flows directly from one device to another using UDP - exactly how WebRTC lays it out in RFC 8835 [7]. Because there is no signal relay involved, media avoids detours, keeping internal network speeds high even when serving many users at once.

C. WebSocket-Based Signaling Mechanism

Without native signaling support, WebRTC relies on alternate paths for coordination. Here, WebSockets step in - carrying essential handshake information between peers aiming to connect directly. Communication flows through the WebSocket API, linking gadgets to a central relay that stays responsive and bidirectional. This kind of constant loop works smoothly for real-time links within closed networks [6]. When someone begins a session, their gadget forms an SDP Offer, packing media capabilities and internal configurations as WebRTC standards outline [2]. The path it follows? First reaches the signal manager, then moves out to the intended receiver. The receiver sends back its own SDP Answer from that point on. Once both sides exchange ICE candidates, pathfinding finishes - trying paths until a stable link clicks into place, closing the handshake. Signaling messages follow WebSocket standards defined in RFC 6455 [8], letting them move through various browsers without snagging. This design lets WebRTC links start fast, while keeping all connection details within the local network.

D. ICE Candidate Gathering and LAN Connectivity

Inside a private network, devices connect straight to one another. Since they live in the same local area, just simple addressing is needed. Rather than reaching outward, communication relies on close-by points built into every device. Local choices avoid outside helpers such as public relay systems. Not sticking to one rigid method, testing moves forward by checking each path in sequence. Faster results show up because there are no roundabout trips relying on external tools. When verification happens nearby, getting things ready takes less time. Things get simpler once you remove added levels that deal with distant access issues. Inside a private network, WebRTC trials point to a straightforward truth: limiting choices to local options cuts waiting times while holding stable links, different from what occurs online . When connections form within sealed networks, routes never shift - movement follows a single line, never straying. Without outside jumps, linked nodes create steady patterns simply because they rely only on one another.

E. Secure Peer-to-Peer Media Channel Establishment

With signaling and ICE negotiations complete, an encrypted tunnel forms directly between the devices. Rather than linking immediately, WebRTC relies on DTLS to exchange encryption keys and verify identities, as defined by RFC 8827 [4]. After the handshake closes, audio and video streams are secured before transmission via SRTP, ensuring confidentiality, integrity, and protection against duplication. Traveling over UDP - following RFC 8835 [7] - the method keeps latency low, making dialogue flow smoothly and naturally. Speed comes first, yet safety never slips behind by much. Because media moves straight from user to user, the signaling server plays no part once connection begins. Inside the private network alone does chat stay alive, blocked off beyond its edges. When encryption works all the way across devices, defense shows up without force. Without added layers weighing down movement, smooth operation continues as is. When things move inside, they stay contained. The flow runs better if routes do not cross. Clean splits mean fewer hiccups later.

F. Implementation Details

One user grabs camera and mic using `getUserMedia()`, sticking to the W3C plan [2]. Without bulky systems, just a light WebSocket server manages messages between people. As soon as `RTCPeerConnection` wakes up, links form - shaped by browser standards listed in [4]. During chats, they exchange connection info through ICE, slipping it across live

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

channels. Always running beneath, the connection system verifies things itself, secures information, picks routes without human help. Following the main WebRTC documentation [1] closely, this entire process stays lean, nearly unseen

High up, there's a Node.js server using the `ws` library, following RFC 6455 [8] to manage WebSocket signals without deviation. Messages move straight through - SDP Offers, Answers, ICE candidates - not touched, just forwarded between clients on an intranet. Rather than alter anything, it does one thing: relay. Because it interferes with nothing, peer connections stay direct, even if guided by a middle node.

Right away, pressing a button brings up options to start or pause communication. When connection locks in, video streams pop into view - your feed in one spot, theirs appears where their data lands. A window forms around each participant after handshake completes. Live visuals load directly into the page as signals travel both ways. Changes reflect fast, piece by piece, pulled from continuous flow of arriving packets. Starting off solid, this setup sticks to familiar techniques that power live web features in any browser. Without slowing down, it keeps data moving quickly, yet still guards each stream from start to finish.

Components:

1. Every person watching the screen spots the chat space right away. When someone types a message or steps into a room, those actions send cues to the software. Someone online gets marked by one module, meanwhile updates to conversation chunks happen nearly without delay. Whether on a handheld device or desktop machine, the interface stays consistent throughout. Each tap triggers information flow via active channels or silent requests behind the scenes. When a person speaks, bits of the screen move right away - no refresh needed. Communication reaches those close by, directly, skipping delays. One device works just like another since all run the same script. Pressing send pushes your words out instantly. Changes appear on their own while users join or leave, moment by moment.
2. Right at the beginning, configuration steps depend on a component made for slow-burn jobs - not immediate actions. Rather than sprinting through information quickly, it communicates with servers using preset endpoints and common request methods. Its utilities sit within client/src/api/, stored in clearly labeled JavaScript files. As the program launches, or when preferences shift, these sections send quiet requests outward. Quiet steps guide every task, even behind the scenes. With duties split like this, conversations flow free from sudden snags.
3. Far out near the boundary, communication lives through Socket.IO linking machine and host. Connection holds firm across local web by way of WebSocket methods. Once open, data moves both ways - fast, no lag. On arrival of updates, response follows just as fast if a user steps in or drops off line. Something typed? It zips across like a spark. Open paths let alerts race far without delay. Screens hold their pose - never flickering back. Underneath, shortcuts vanish into thin air, leaving only speed. The moment arrives fast, already there before you blink.
4. Messages flow through the setup using a Wi-Fi network as their path. Connected gadgets talk inside a common area, such as a household hub or private cluster of devices. No web connection is used - data remains within that boundary. Data travels end to end by mixing HTTP with real-time socket pathways in the same territory. Security improves because communication never steps outside the immediate loop. Even without nearby support hubs, it keeps running where signals fade. When links to worldwide connections break, tasks go on just the same.
5. A tiny chat space lives directly on your computer. Running behind it? Node.js paired with Express, sitting in server/server.js. Instead of only answering REST API requests, it holds real-time connections alive through WebSockets. Each newcomer is logged without gaps. Over there, messages find their way without confusion. A visitor’s progress? Always current while they’re around. Updates flow - systems chatting together keeps it that way. Running nearby, not across distant servers. The cloud doesn’t get involved at all. When things keep close, they hold tight - no need to reach beyond. One piece leans on another, yet none stand apart. Stillness keeps them linked, each part fitting where it lies.
6. Tucked within server/routes, the REST API takes shape across distinct files, each handling its own type of web request. Following sequence matters - userRoutes.js handles signups and logins. Chat interactions? They move through chatRoutes.js. When it comes to admin duties, adminRoutes.js holds that responsibility. System-level data shows up thanks to systemRoutes.js. Profile-related changes travel a separate route, managed by userUpdateRoutes.js. A single incoming request gets guided exactly - no guessing by Express. Splitting tasks this way means tomorrow’s tweaks won’t cause collapse, allows growth while staying intact.
7. Later on, device conversations happen instantly using WebSockets via Socket.IO. Messages pop up without delay because chat.js takes care of the flow, not by waiting. Who's online shifts quietly behind the scenes, managed by presence.js watching in silence. A voice call begins, then call.js activates, responding to each signal it finds. Starting it all, index.js connects the sockets early, tying behaviors together when things boot up. From the moment someone hits

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

send, the server snatches the message, figures its path, shoves it forward. On local networks, data flows instantly since devices speak straight to one another.

8. Back inside server/utils, the Real-Time Event Handler stays active. Each time an event occurs, work begins without noise. Through linked gadgets, updates flow at steady pace. Watching user presence lands within this space. Tasks like these run right here. Changes move fast, no waiting around. Because it checks in often, live sessions never disappear. Clarity comes from leaving this piece on its own. Even under load, speed holds firm. With duties split cleanly, everything fits better now.

9. Inside memory it runs, the local server status sits in server/utils/systemState.js. When a person arrives, their data shows up here - sockets, links, ongoing sessions. Leaving users? Their records disappear fast. No web connection means no bulky databases are required. Quick movement counts, which is why staying slim works. As shifts occur on the spot, second after second, entry remains fast - tasks begin right away because of that. Shaped like this, lag has no chance to pile up.

G. Feasibility and Performance Considerations

Fine performance comes through placing the setup on a secluded inside net, since WebRTC likes nearby, steady paths. Without relays in sight, voice and video hop straight between devices - UDP pushes chunks swiftly, holding lag tight, matching what transport standards note [7]. Rather than slow things, coded layers slide into place naturally: every flow gets sealed end to end by SRTP, keeping secrecy strong without dragging pace [4].

A tiny signaling server takes care of only initial connections, which means minimal demand on processing resources. When teams study WebRTC within internal networks, they find that local setups operate well without straining hardware.

IV. WORKFLOW OF THE SYSTEM

A single server runs on-site, linking up with nearby devices across the local network. Instead of relying on outside connections, it uses REST APIs for requests while staying live via WebSocket links. Real-time updates flow even when disconnected from the internet. Presence status shifts automatically based on who is active at any moment.

Fig4. System Workflow

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Step 1: Start Local Server

Right away, the local server kicks into motion within a Wi-Fi network. Running on Node.js and Express, it links all components as its core role. As it starts, REST API routes load alongside Socket.IO event watchers, at the same time preparing short-term data space for real-time states. Missing this step means live feedback, incoming requests, plus monitoring active users fall apart - particularly during offline moments.

Step 2: Client Launches Application

Once the server wakes up, the user's gadget loads the React app through Vite. The chat window appears, along with a space to type and a list showing who is online. At the same time, links form using both REST API and Socket.IO. With those live, actions flow freely while the interface preps to sync without hiccups to the back end.

Step 3: Connect via Wi-Fi LAN

Right away, the app fires up and reaches out to the server through whatever Wi-Fi sits nearby. Instead of touching the web at all, everything travels just within that private link. Because of this setup, it runs fine even when no connection to the wider internet exists - think classrooms, far-off areas, or campus zones. So long as things stay tied together on the local side, information flows by way of either HTTP or WebSocket rules.

Step 4: User Presence Registered

Right away, when the connection activates, Socket.IO opens a real-time path between both ends. At that moment, the system flags the user as active, updating its current tally on the spot. Afterwards, details about who's online flow out to every connected member. Not until then does each person get the latest presence info without lag.

Step 5: Process REST and Socket Events

Right after signing up, tracking kicks in without pause. For fetching personal info or tweaking options - tasks where delays hardly matter - data travels through REST APIs. Live messages shoot across instantly, powered by Socket.IO connections. One path handles slow, reliable exchanges, another takes fast, time-sensitive jobs. This separation sharpens performance since every system runs its own race. Each piece works at full strength, nothing more, nothing less.

Step 6: Broadcast Messages and Updates

Out into each linked device, messages travel the moment they're passed along live streams. The instant a chat shows up, or a sign changes, or warnings land on gadgets - no need to reload anything. Tight sync holds through the whole scattered system since pauses shrink almost to nothing.

V. OUTPUTS

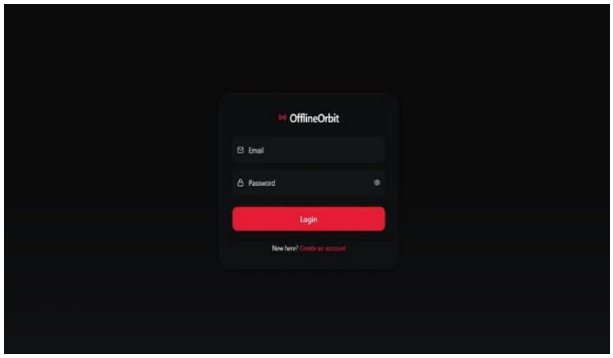


Fig .5.1 Login page

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

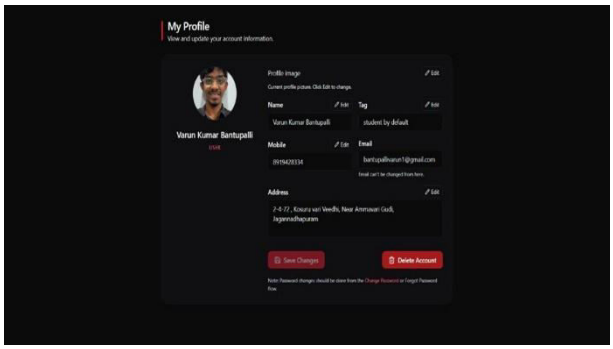


Fig.5.2 profile page

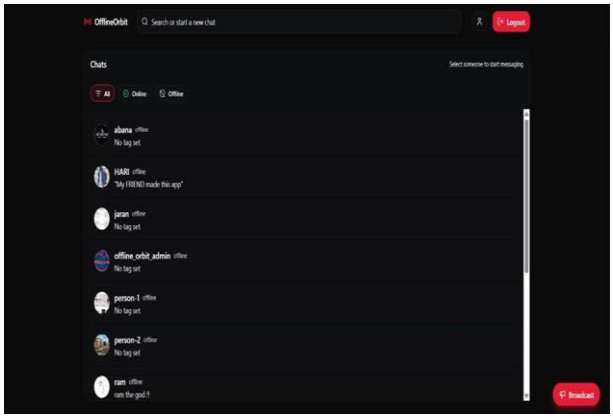


Fig.5.3 Available Users

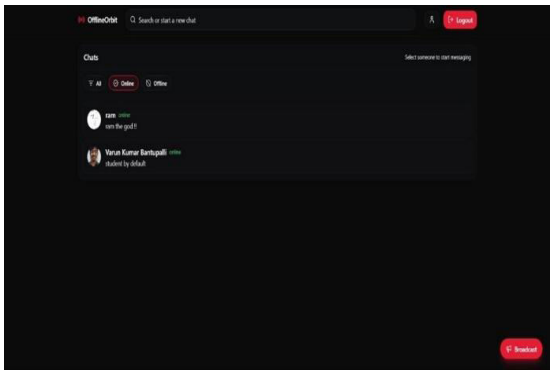


Fig.5.4 Online users

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

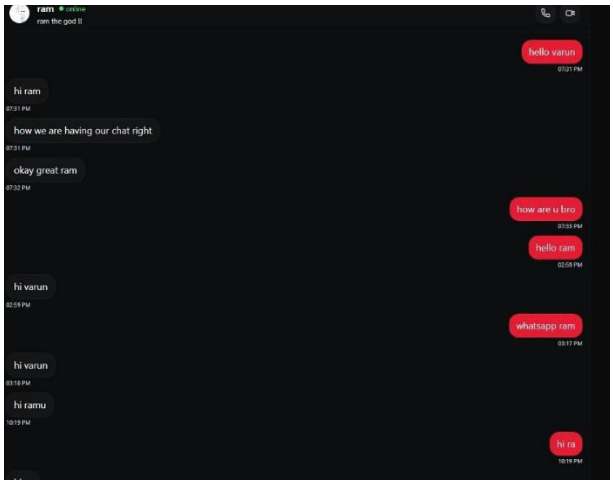


Fig.5.5 chart page

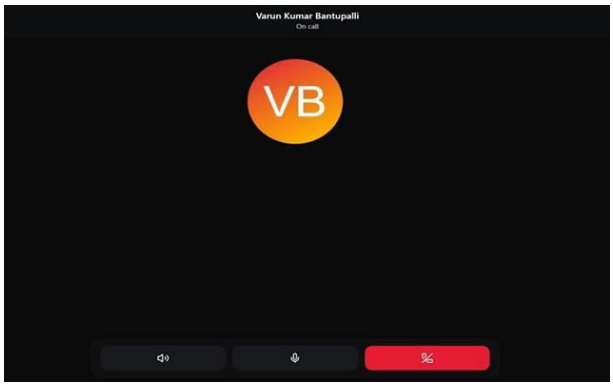


Fig.5.6 voice call

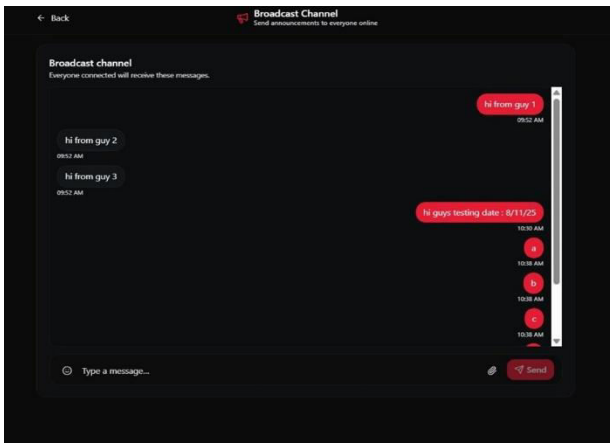


Fig.5.7 Broadcast chat

© 2026 IJIRCCE | Volume 14, Issue 6, June 2026 DOI: 10.15680/IJIRCCE.2026.1406100

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Fig.5.8 video call

VI. CONCLUSION

When the web drops out, Offline Orbit keeps things linked without fuzz. Thanks to paths built into local networks, gadgets spot one another straight off - no middle steps needed. One single router might cover a huge zone, handy inside factories or large campuses. Chatter flows person to person, including live audio and face-to-face calls that stay steady even when pushed hard. Out in the open, far from cities, even at crisis sites - spots where regular software gives up - it still hums along. No reliance on external connections means it moves without the usual web-based chains. Built so changes can slip in later, no wreckage required. The pieces click, thanks to a plan shaped long before coding started. When stress hits, its toughness shows, proven not guessed

Working well at first, it could take on extra jobs without trouble. Smoother later, its performance fits more people's ways of using it. Growth happens naturally, not forced.

Right now, Offline Orbit handles messages without a connection just fine. Built with space to expand, so future changes won't demand frequent upgrades. It sticks to its original purpose clearly. New tools can slip in later - no rebuild needed. Down the road, companies could start using it more often. What we've seen shows it works outside of trial runs.

REFERENCES

[1] Google Developers, "WebRTC — Real-Time Communication," https://webrtc.org/
[2] W3C, "WebRTC 1.0: Real-Time Communication Between Browsers," https://www.w3.org/TR/webrtc/
[3] IETF, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translation Traversal," RFC 8445.
[4] IETF, "WebRTC Security Architecture," RFC 8827.
[5] IETF, "WebRTC Data Channel Establishment Protocol," RFC 8831.
[6] Mozilla Developer Network (MDN), "WebSockets API," WebSockets Docs
[7] IETF, "Transports for WebRTC," RFC 8835.
[8] IETF, "The WebSocket Protocol," RFC 6455.

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details