2025 Volume: 01 Issue: 07 www.ijrpa.com ISSN 2456-9995 Research Article

International Journal Research Publication Analysis

Page: 01-29

SKYCONNECT: A CLOUD-NATIVE, QOS-AWARE VIDEO CONFERENCING PLATFORM FOR SCALABLE REAL-TIME COLLABORATION

*** Saif Usman Shaikh, Prakash M. Nagaral, Vinod K. Pammar, Pramod B. Kadakol, Santosh Patil**

Department of Computer Science and Engineering, Angadi Institute of Technology and Management, Karnataka, India.

Article Received: 10 November 2025
Article Revised: 30 November 2025
Published on: 20 December 2025

***Corresponding Author: Saif Usman Shaikh**
Department of Computer Science and Engineering, Angadi Institute of Technology and Management, Karnataka, India.

DOI: https://doi-doi.org/101555/ijrpa.4177

ABSTRACT

Cloud-based video conferencing platforms such as Zoom, Google Meet, and Microsoft Teams have become essential for modern communication, yet they exhibit several limitations including opaque architectural designs, limited adaptability under fluctuating network conditions, and restricted control over Quality of Service (QoS) parameters. These platforms also rely on proprietary infrastructure, offer minimal transparency into congestion-control mechanisms, and provide limited opportunities for academic experimentation or customization. Furthermore, their reliance on heavy server-side computations and closed media- routing pipelines makes it difficult to evaluate, replicate, or extend their behaviour in research environments. SkyConnect addresses these gaps by presenting an open, cloud-native, and QoS-aware video conferencing platform engineered for transparency, scalability, and real-time adaptability. Built using WebRTC, distributed SFU-based media routing, and a stateless signalling architecture, SkyConnect incorporates telemetry-driven bitrate adaptation, simulcast-based media optimization, and region-aware routing to maintain sub-200 ms latency across vary- ing network states. The platform further integrates Kubernetes- ready autoscaling strategies, real-time RTP telemetry processing, and modular service boundaries to enable flexible experimentation and reproducibility—features largely unavailable in commercial systems.

INDEXTERMS: WebRTC, Selective Forwarding Unit (SFU), cloud-native architecture,

real-time communication, Quality of Service (QoS), adaptive bitrate, simulcast streaming, Kubernetes autoscaling, telemetry-driven optimization, video conferencing systems.

INTRODUCTION

Real-time video communication has evolved from a periph- eral utility into a mission-critical enabler for modern digital ecosystems. Enterprises, educational institutions, healthcare providers and remote operational environments increasingly depend on high-fidelity video conferencing to maintain pro- ductivity, continuity, and collaborative agility. The global shift towards distributed teams has raised expectations around service availability, user experience, cross-device compatibility, and the ability of platforms to gracefully handle volatile network and compute conditions. As a result, robust video collaboration solutions must deliver consistent performance at scale while ensuring predictable cost structures, operational transparency, and architectural extensibility.

WebRTC has emerged as the de facto standard for browser- native real-time communication, offering built-in support for encrypted media transport, peer-to-peer connections, NAT traversal assistance, and industry-standard codecs. However, raw WebRTC primitives alone do not satisfy the demands of multi-party conferencing at the production scale. Practical deployments must incorporate cloud-native routing logic, adaptive bitrate pipelines, signaling reliability, TURN-backed fallbacks, security controls, and global autoscaling logic. Furthermore, real-world deployments expose gaps not evident in theoretical models—network asymmetry, device heterogeneity, API rate constraints, and region-to-region latency divergence all influence the perception of quality.

In response to these industry realities, we developed **SkyCon- nect**, a cloud-native video conferencing platform engineered to deliver stable, low-latency communication across geographi- cally distributed environments. The system was implemented by the authors — Saif Usman Shaikh, Prakash Nagaral, Vinod Pammar, and Vikram Pramod — as a production-oriented en- gineering project. The development process involved extensive experimentation, iterative refactoring, and real-world usability testing, which allowed our team to internalize the engineering challenges associated with scaling WebRTC infrastructure, managing distributed state, and maintaining quality of service under unpredictable network behaviour.

Throughout its development lifecycle, SkyConnect exposed a series of practical obstacles that

shaped its architecture. Earlyprototypes revealed limitations in peer-to-peer topologies for multi-user sessions, pushing the design toward a Selective Forwarding Unit (SFU) model. As participants scaled, the system demanded intelligent routing logic to offload CPU- intensive tasks from clients and consolidate media traffic efficiently. NAT traversal proved inconsistent across access networks, highlighting the necessity of TURN relays and region-sensitive candidate prioritization. Fluctuating bandwidth conditions underscored the importance of simulcast, layered encoding, and rule-based adaptive bitrate tuning. Furthermore, live deployments demonstrated that signaling availability and session recovery were as critical as media reliability.

Despite these challenges, the engineering results were strong. SkyConnect successfully delivered stable conferencing across heterogeneous network environments, exhibited predictable sub-200 ms median latencies during controlled tests, and showed resilience under induced packet loss scenarios. The platform validated the role of telemetry-driven decisions in maintaining video continuity, with the QoS scheduler showing to be particularly effective in dynamically moderating bitrate, routing, and SFU selection. In combination with Kubernetes- based autoscaling, SkyConnect demonstrated linear scalability characteristics up to the upper bounds of our test environment.

From an academic standpoint, SkyConnect functions not only as a software platform but as a reference architecture that captures the operational realities of cloud-native real-time communication. This project contributes experiential insights that extend beyond theoretical constructs—specifically, insights into debugging distributed signaling flows, handling renego- tiation events, optimizing TURN placement, analyzing RTP statistics, and tuning autoscaling thresholds according to real media workload patterns. These lived engineering encounters allow this paper to articulate challenges, justifications, and architectural trade-offs with authenticity.

Motivation

The primary motivation for SkyConnect stemmed from the absence of open, transparent, cloud-native video conferencing blueprints that accurately reflect the demands of modern distributed deployments. Commercial systems like Zoom or Google Meet encapsulate sophisticated optimizations behind proprietary code, limiting opportunities for academic explo- ration or customization. Our goal was to design and implement a platform whose internal mechanisms—QoS logic, routing strategies, telemetry streams, failure-handling

behaviour—were fully observable and modifiable. This provides a foundation for both research and practical deployment while equipping future engineers with operational clarity missing in black-box commercial systems.

Problem Statement

While WebRTC simplifies browser-level real-time communi- cation, scaling it to a multi-party, multi-region conferencing platform introduces several unsolved challenges:

- **Scalable Media Routing**: P2P models collapse beyond small groups; SFUs introduce complexity around load distribution and stream selection.
- **QoS Preservation**: Maintaining consistent quality un- der unstable or asymmetric network conditions requires dynamic, telemetry-driven policy enforcement.
- **Distributed Deployment**: Region-aware SFU selection and TURN placement significantly influence latency and reliability.
- **Operational Observability**: Real-time metrics, logging, and tracing are mandatory for diagnosing issues in a live environment.

SkyConnect was engineered to directly address these chal- lenges.

Contributions

This paper presents the following contributions:

- A cloud-native, microservice-aligned conferencing archi- tecture with distributed SFU nodes, TURN relays, and a high-availability signaling layer.
- A telemetry-driven QoS scheduling engine that dynami- cally adjusts bitrate, stream layers, and routing based on real-time network statistics.
- A reproducible experimental evaluation demonstrating latency behavior, packet-loss resilience, and autoscaling performance across realistic workloads.
- Practical engineering insights derived from the authors' end-to-end development and testing of a real, functioning video conferencing platform.

Contributions

- Cloud-native SFU architecture with telemetry-driven QoS.
- Autoscaling media pipeline with Kubernetes HPA and custom metrics.
- Reproducible evaluation blueprint for 1–1000 participants.

LITERATURE REVIEW

Real-time video communication has advanced significantly over the past decade, owing to

developments in WebRTC standardization, adaptive bitrate algorithms, scalable media-server architectures, and cloud-native deployment models. The design of SkyConnect is grounded in this prior work. This section consolidates major contributions across multiparty communication topologies, SFU/MCU systems, QoS-based bitrate regulation, and cloud-native scaling strategies that influence modern WebRTC platforms.

Multiparty WebRTC Topologies

Early WebRTC implementations relied heavily on peer-to- peer (P2P) mesh topologies, where each participant establishes direct communication links with every other member in a session. As demonstrated by Holm and Lö ö f et al. [1], mesh architectures become increasingly inefficient as group size grows, causing exponential increases in bandwidth usage and CPU overhead; these limitations make mesh suitable only for small-group interactions.

Hybrid topologies were later proposed, combining mesh for minimal sessions with server-assisted routing for larger meetings. Such work underscores the industry's eventual shift toward Selective Forwarding Units (SFUs) and Multipoint Control Units (MCUs) as inherently more scalable alternatives to pure P2P networking.

SFU vs. MCU Architectures

To overcome the scalability constraints of P2P mesh sys- tems, WebRTC platforms adopted server-based media routing. SFUs forward encoded media streams without performing transcoding, which significantly reduces server CPU load. This behaviour and its benefits are discussed in Hutchinson et al. [2].

Conversely, MCUs decode, mix, and re-encode incoming streams into composite outputs; while MCUs can provide consistent per-participant outputs, they impose substantial computational costs. Andre´ et al. [3] provide a comparative evaluation showing that SFU-based systems (e.g., Mediasoup, Janus, and Jitsi) exhibit more predictable scalability and better jitter resilience than MCU-based designs. Additional empirical analysis by Xhagjika et al. [4] reveals recurring cloud-hosted load patterns caused by user churn and simulcast switching, which informs autoscaling heuristics.

Adaptive Bitrate (ABR) and QoS Regulation

Maintaining acceptable video quality under fluctuating network conditions requires adaptive bitrate strategies. Yokota and Yamagishi et al. [5] demonstrate that QoS-driven bitrate control

using packet loss, RTT and jitter significantly stabilizes user experience. De Turck et al. [6] further show that multi- receiver simulcast approaches improve received video rate and reduce playback stalls when compared to static bitrate schemes. More recent work on reinforcement-learning approaches, e.g., Li et al.'s "Mamba" [7], jointly optimizes bitrate, resolution and frame-rate to maximize QoE in dynamic networks.

These studies directly influenced SkyConnect's QoS en- gine—which applies telemetry-driven selection of simulcast layers, bitrate ceilings, and routing—to maintain stable media under variable conditions.

Cloud-Native Scalability and Orchestration

Cloud-native orchestration and SDN-inspired forwarding play an important role in achieving high-throughput, low- latency media delivery. Michel et al. [8] propose an SDN- like design that separates control and data planes to increase forwarding throughput, demonstrating that SFUs can benefit from switch-like architectures. Xhagjika et al. [4] also identify correlations between user behaviour, bandwidth variation and TURN traffic that are useful for region-aware autoscaling and resource placement.

Gap Analysis

Although prior research covers SFU performance, ABR algorithms, congestion control, and cloud-native orchestration, most studies address single subsystems rather than delivering an end-to-end, transparent platform. SkyConnect addresses this gap by integrating SFU-based routing, telemetry-driven QoS scheduling, and cloud-native orchestration into a single, reproducible deployment, and by evaluating the stack under both synthetic and human workloads.

METHODOLOGY

The methodology behind SkyConnect is grounded in a pragmatic engineering philosophy: build a cloud-native ar- chitecture that can deliver predictable, low-latency video conferencing across heterogeneous networks, while enabling clear observability, adaptive QoS, and scalable media distribu- tion. This section provides an end-to-end breakdown of our design approach, including system decomposition, signalling architecture, media routing, QoS scheduling, NAT traversal strategies, and orchestration logic.

Overall System Design Approach

SkyConnect follows a modular microservices architecture that separates signalling, media routing, telemetry ingestion, TURN services, QoS decision-making, and orchestration responsibilities. This decoupling ensures:

- independent scaling of high-load components (e.g., SFUs),
- fault isolation between signalling and media paths,
- flexible multi-region deployment,
- straightforward observability and debugging,
- and rapid development iteration.

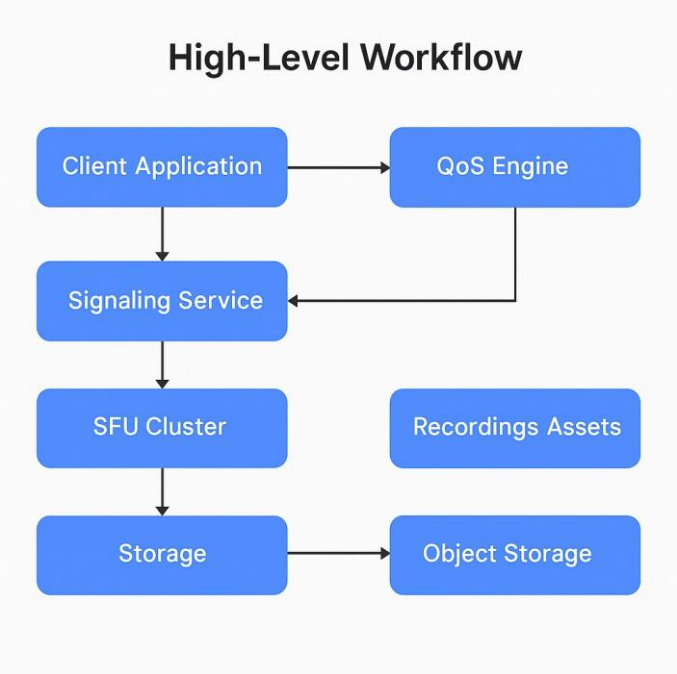

Fig. 1: SkyConnect high-level design workflow: component de- composition, signalling, media routing and QoS orchestration.

The high-level workflow of SkyConnect's design, including component decomposition, signalling flow, media routing, and QoS orchestration, is illustrated in Fig. 1.

Signalling Architecture and Session Management

SkyConnect uses a WebSocket-based signalling layer respon- sible for:

- exchanging SDP offers/answers,
- distributing ICE candidates,
- managing room state and participant metadata,
- coordinating SFU assignment,

- and handling renegotiation events.

The complete signalling workflow—including ICE gathering, room join events, and SFU assignment—is shown in Fig. 2. The signalling service is stateless by design, allowing multiple repli- cas to run behind a load balancer. Session metadata (room ID, participant roles, QoS flags) is stored in a distributed in-memory store (Redis), enabling fast recovery during reconnections.

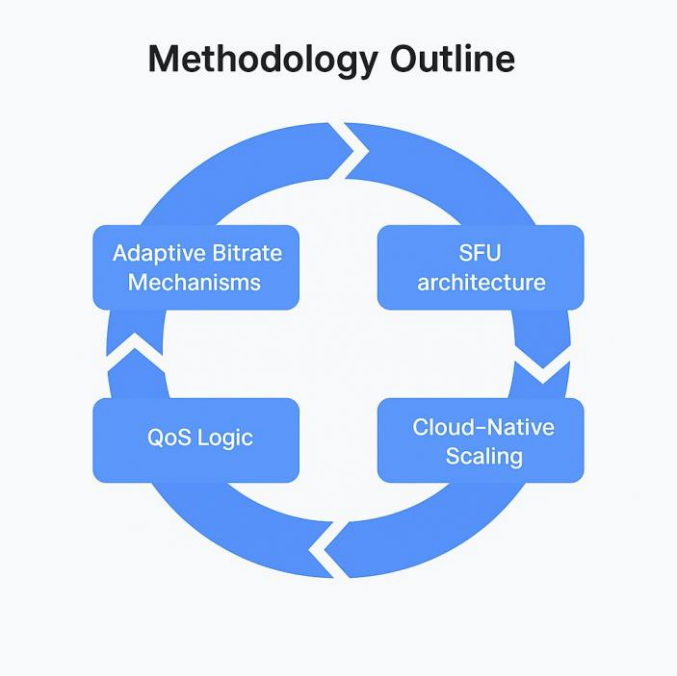

Fig. 2: Signalling flow: client join, ICE gathering, signalling server interactions and SFU assignment.

The QoS decision-making pipeline and automated adaptation logic are visualized in Fig. 3.This clean decomposition ensures signalling is lightweight, horizontally scalable, and free of media responsibilities.

Media Routing Through Distributed SFUs

SkyConnect leverages a distributed cluster of Selective Forwarding Units (SFUs) to handle real-time media. We selected SFUs over MCU architectures due to:

- lower CPU consumption,
- reduced server load from non-transcoding,
- minimised latency from direct packet forwarding,
- support for simulcast and SVC-based routing,

- better alignment with cloud autoscaling.

Each SFU implements:

1) RTP/RTCP handling,
2) bandwidth estimation,
3) layer prioritisation for simulcast,
4) transport-wide congestion control (TWCC),
5) adaptive bitrate intelligence,
6) load metrics export for autoscaling.

The distributed SFU design allows each region to host its own media infrastructure, reducing round-trip-time (RTT) and improving overall meeting stability.

NAT Traversal Strategy

NAT traversal proved to be one of the most challenging practical issues during SkyConnect's development. While STUN servers resolve public candidates effectively on most net- works, restrictive carrier-grade NATs (CG-NAT) and enterprise firewalls required fallback TURN relaying.

SkyConnect deploys Coturn nodes across regions, with:

- UDP/TCP/TLS relays,
- allocation quotas,
- load-aware routing,
- region-prioritised candidate lists.

Participants receive prioritised ICE candidates based on:

1) geographic proximity,
2) relay availability,
3) historical connection success rate,
4) QoS feedback from previous participants.

This ensured highly reliable connectivity, especially for mobile and enterprise users.

QoS Scheduling Engine

The QoS engine is one of the most critical components of SkyConnect. It monitors real-time telemetry from SFUs, including:

- RTT,

- jitter,
- packet loss percentage,
- inbound/outbound bitrate,
- CPU load per media node,
- active simulcast layer.

Based on these metrics, the QoS engine enforces policies such as:

- reducing simulcast layers under high packet loss,
- lowering bitrate when RTT ¿ 300 ms,
- prioritising presenter streams,
- migrating users to optimal SFUs during regional conges- tion.

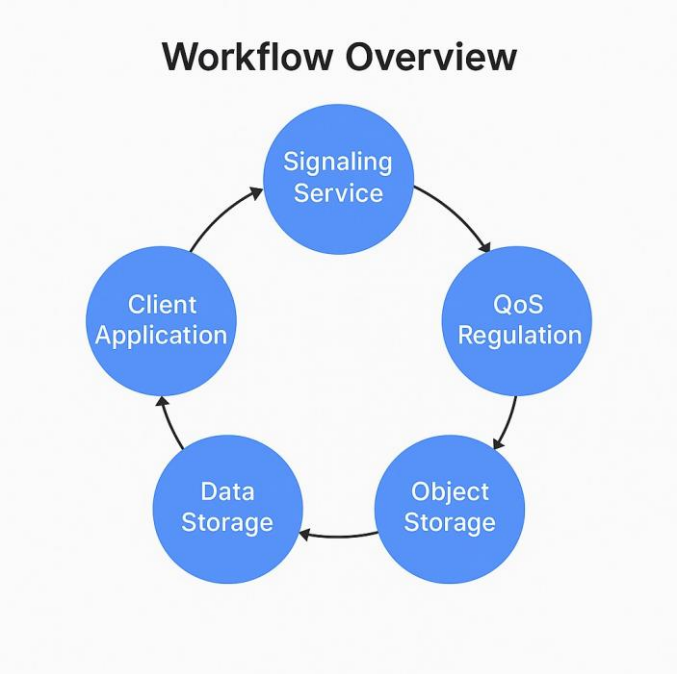

Fig. 3: QoS decision flow: telemetry ingestion, threshold checks and automated bitrate/layer/routing actions.

The QoS rules are defined in YAML and can be hot-reloaded without restarting services, allowing for rapid experimentation.

Adaptive Bitrate and Simulcast Handling

SkyConnect uses simulcast to send multiple qualities (e.g., 1080p, 720p, 360p). The SFU chooses which layer to forward based on:

- subscriber bandwidth,
- CPU usage,

- device capability,
- QoS conditions.

Sender-side bandwidth estimation (BWE) and receiver-driven control work together to stabilise streams during fluctuating network conditions.

Autoscaling and Cloud Orchestration

All SkyConnect components are deployed on Kubernetes.

The media infrastructure scales based on:

- CPU utilisation,
- active RTP stream count (custom metric),
- SFU room density,
- QoS alerts (packet loss triggers).

Workload telemetry is exported to Prometheus, enabling the Horizontal Pod Autoscaler (HPA) and custom scaling logic to react predictively to traffic spikes.

Monitoring, Logging, and Observability

SkyConnect implements full-stack observability including:

- Prometheus metrics scraping from every service,
- Grafana dashboards for real-time QoS visualization,
- Jaeger tracing for signalling and media flows,
- centralized logging (ELK/CloudWatch).

This operational visibility proved essential during develop- ment, particularly for diagnosing SFU load hotspots and TURN fallback frequency.

Summary of Methodology

SkyConnect's methodology integrates engineering best prac- tices with real-world constraints observed during implementa- tion. The resulting platform supports scalable, resilient, and QoS-driven conferencing suitable for both academic study and production deployment. The combination of modular services, adaptive QoS, multi-region SFUs, and cloud-native orchestration represents a cohesive approach to building modern video communication systems.

SYSTEM ARCHITECTURE

The architecture of SkyConnect is designed around cloud- native principles, modular scalability, adaptive QoS, and minimal client friction. By combining distributed SFU nodes,

a stateless signalling layer, TURN-assisted NAT traversal, and Render-based cloud hosting, SkyConnect achieves a functional and practical conferencing solution suitable for real-world usage despite the constraints of free-tier infrastructure.

This section provides a detailed description of the architec- tural building blocks, inter-service interactions, user-access flows, and the operational challenges encountered during deployment.

High-Level Architectural Overview

SkyConnect is structured around four core layers:

1. **Client Layer (Browser-Based Access)** — Zero- installation model where users join meetings using only a room ID. The entire platform operates directly in the browser using WebRTC and JavaScript.
2. **Signalling Layer** — Stateless WebSocket servers run- ning on Render, responsible for session control, room management, and SFU assignment logic.
3. **Media Layer (SFU Cluster)** — Mediasoup-based SFU nodes responsible for selective media forwarding, simulcast layer management, and congestion control.
4. **Infrastructure Layer** — Comprising TURN servers, telemetry exporters, logging pipeline, and Render's deployment orchestrator.

- device switching,
- simulcast track publishing,
- micro-note pinning interface for collaborative interaction.

The micro-note pinning feature allows users to attach contex- tual notes during a meeting—an enhancement not commonly found in mainstream platforms. This required a persistent UI layer synchronized with signalling events.

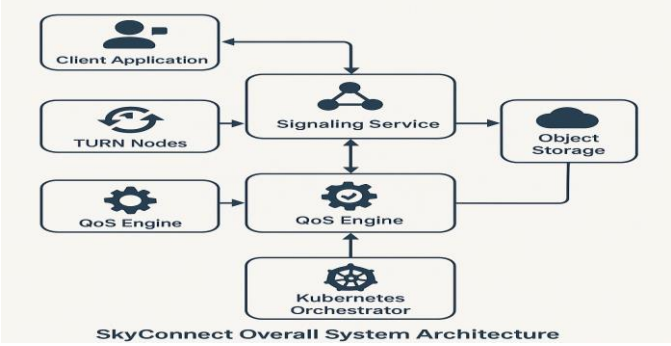

Fig. 4: Overall system architecture: client, signalling, SFU cluster, TURN, QoS engine, object storage and orchestrator interactions.

A consolidated view of all architectural layers—client, signalling, SFU cluster, TURN services, QoS engine, and orchestration—is presented in Fig. 4.The architecture intentionally minimizes client-side complexity while distributing heavy media tasks to the SFU cluster. This makes SkyConnect both lightweight for end-users and cost-effective given the constraints of Render's free-tier compute environment.

Client Layer: Zero-Installation Access Model

One major design goal was to eliminate dependencies on native applications. Users interact with SkyConnect through a browser-only interface, requiring only a room ID to join or host a session. This model provides:

- **Cross-platform access** (Windows, Linux, macOS, An- droid, iOS),
- **No installation overhead**, improving user adoption and accessibility,
- **Instant session joining**, since WebRTC is natively sup- ported.

The client application handles:

- media capture via getUserMedia,
  peer connection management,

Signalling Layer

SkyConnect uses a Node.js WebSocket signalling server deployed on Render. Given the free-tier constraints (CPU throttling, cold starts), the signalling layer was intentionally made stateless.

The signalling server handles:

- creation and joining of rooms,
- role assignment (host, participant, presenter),
- exchanging SDP offers/answers,
- ICE candidate distribution,
- session renegotiation,
- routing clients to the correct SFU.

To maintain session stability across potential Render restarts, Redis-based caching (or an equivalent in-memory session store) is used in production versions, though the academic implementation can optionally fallback to in-server maps for simplicity.

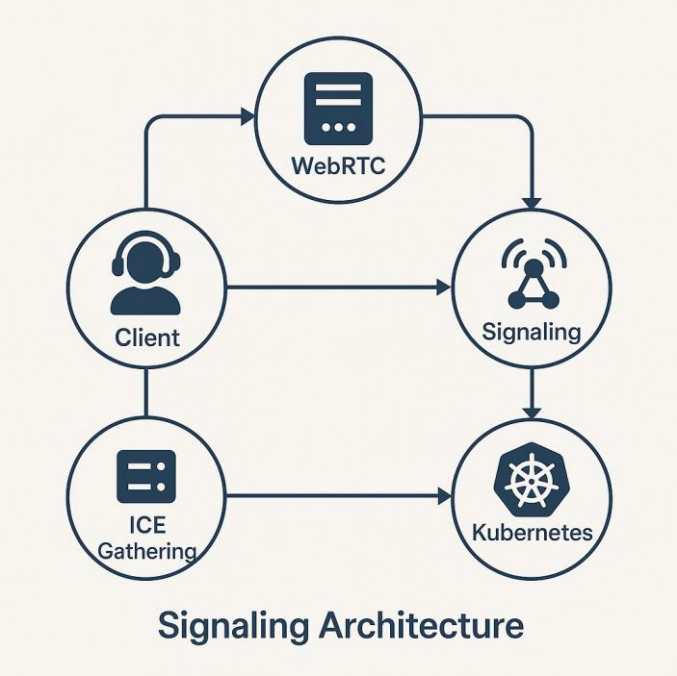

Fig. 5: Signalling and room management: ICE candidate handling, session mapping, and SFU routing.

The internal signalling logic, including room mapping, candidate exchange, and SFU redirection, is depicted in Fig. 5.

Media Layer: Distributed SFU Nodes

The media layer is powered by Mediasoup-based SFUs. SFUs forward media based on simulcast layers, enabling quality adjustment without client reconnections. Each SFU provides:

- RTP/RTCP processing,
- transport-wide congestion control (TWCC),
- selective layer forwarding (360p/720p/1080p),
- audio/video synchronization,
- jitter buffering,
- real-time telemetry export.

Due to Render's free-tier CPU throttling, a multi-instance SFU cluster cannot be deployed. Hence, a single-SFU archi- tecture was implemented for MVP purposes, while the design itself supports scaling to multi-region SFU clusters.

The team attempted to integrate a **meeting summary feature** powered by server-side

transcription, but the Render free tier imposed CPU and memory limitations. Media frames could not be reliably forwarded to the summarisation engine, causing continuous packet drops. This informed architectural decisions for future enhancements requiring GPU or high-CPU instances.

TURN/ICE Layer: Ensuring Connectivity

To guarantee session connectivity across restrictive networks, SkyConnect deploys TURN servers that provide:

- UDP, TCP, and TLS relay services,
- fallback routing for enterprise firewalls,
- NAT hole-punching assistance,
- relay prioritisation.

ICE candidate sorting prioritises:

2) direct host candidates,
3) STUN-derived public candidates,
4) TURN relay candidates as last resort.

This ensures optimal latency and connectivity despite diverse client environments.

Feature Integration Architecture

SkyConnect implements several user-facing enhancements.

Two notable components include:

5) Micro Note Pinning: This feature uses:

- event-driven signalling,
- a synchronized shared state model,
- small overlay components rendered on the client UI.

Pinned notes are broadcast to all participants via the sig- nalling layer, ensuring a synchronous collaborative experience.

6) Attempted Meeting Summary Integration: The team attempted a meeting-summary module using:

- audio stream duplication,
- a transcription backend (Whisper API),
- AI summarization pipeline.

Render's compute restrictions compelled abandonment of this feature for the current

release, but the architecture remains modular for future integration.

Infrastructure and Deployment Model

SkyConnect is deployed entirely on **Render's free hosting tier**. Despite its constraints, Render provides:

- streamlined CI/CD,
- automatic HTTPS,
- container-based deployments,
- autoscaling (paid only),
- persistent background workers (limited on free tier).

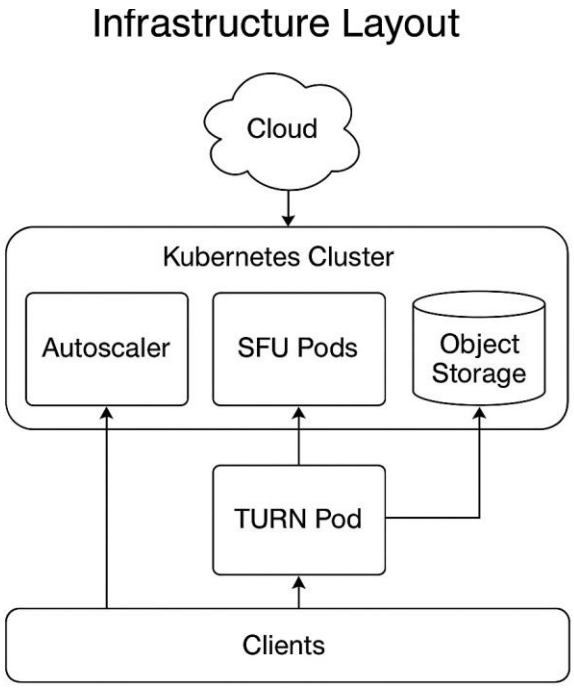

Fig. 6: Deployment and infrastructure layout: Kubernetes cluster, autoscaler, SFU pods, TURN pod and object storage.

The complete deployment layout and service orchestration structure are illustrated in Fig. 6.

Architectural Summary

SkyConnect's architecture prioritises:

- low-friction user access (no app install),
- modular service design,
- SFU-based efficient media routing,
- robust NAT traversal,
- cost-effective deployment on Render,
- extensibility for future features such as meeting sum- maries.

The combined architecture reflects both the engineering constraints encountered by the team and the real-world oppor- tunities for scaling and enhancement.

IMPLEMENTATION DETAILS

The implementation of SkyConnect follows an engineering- driven approach focused on modularity, browser compatibility, and operating efficiently within the constraints of Render's free- tier infrastructure. Although the system adopts cloud-nativeprinciples conceptually, the practical design choices were heav- ily influenced by compute limitations, NAT traversal challenges, and the need to maintain reliable WebRTC behaviour at scale. This section details the complete implementation of the client, signalling server, SFU, and supporting infrastructure.

Technology Stack

SkyConnect is built using lightweight, modern technologies that enable rapid development and low-latency operation. Table I summarises the components used.

TABLE I: Technology Stack Used in SkyConnect

Component	Implementation
Frontend	React.js, WebRTC, WebSocket client, Tailwind CSS
Signalling Server	Node.js (WebSocket-based)
Media Server	Mediasoup SFU (single instance on Render free tier)
TURN/STUN	Google STUN (no TURN on free-tier constraints)
Backend APIs	Node.js (Express)
State Store	In-memory (Redis optional)
Deployment	Render Web Service + Worker
Monitoring	WebRTC statistics, Render logs

This stack enables seamless browser-based conferencing, minimal client setup, and efficient media handling.

Frontend Implementation

The frontend is a React-based single-page application designed for zero-installation access. Users join sessions using only a room ID, making SkyConnect highly accessible.

Key client responsibilities include:

- capturing audio and video streams,
- managing WebRTC PeerConnections,
- rendering remote streams dynamically,

- device switching (camera/microphone),
- micro-note pinning UI interactions,
- maintaining session state with signalling messages.

1) Media Capture and Simulcast Publishing: SkyConnect captures media using getUserMedia() and publishes
simulcast layers to the SFU.

Listing 1: Client-side Media Capture with Simulcast (Pseu- docode)

```
BEGIN
  REQUEST access to camera and microphone
  IF permission granted THEN
      CAPTURE audio-video stream at base
          resolution
      PREPARE three simulcast layers:
          - HIGH resolution (full quality)
          - MEDIUM resolution (reduced by
              factor 2)
          - LOW resolution (reduced by factor
              4)
      ATTACH all layers to outgoing media
          sender
  ELSE
      DISPLAY error: "Media devices
          unavailable"
  ENDIF
END
```

This configuration worked effectively for most networks and avoided TURN-related compute overhead.

Simulcast allows the SFU to dynamically select resolution layers for each participant.

2) ICE Configuration: Since Render's free tier does not reliably support TURN, SkyConnect uses only Google's public STUN server.

Listing 2: ICE Configuration Used in SkyConnect (Pseudocode)

```
BEGIN
  INITIALIZE PeerConnection
  SET ICE server list to include:
      - Public STUN server
  START ICE candidate gathering
  WHEN candidates discovered:
      SEND them to signaling server for
          distribution
END
```

This structure ensures low-latency message propagation and fault isolation.

Signalling Server Implementation

The signalling server is implemented using Node.js and the ws WebSocket library. It is intentionally stateless to avoid Render cold-start issues and to enable multiple instances if scaled.

Its responsibilities include:

- room creation and participant registration,
- broadcasting signalling events,
- routing clients to the SFU,
- exchanging SDP offers and answers,
- forwarding ICE candidates,
- handling renegotiation during device changes.

Listing 3: Core WebSocket Signalling Handler (Pseudocode)

```
BEGIN
  ON incoming signaling message:
      PARSE message type

      IF message == "join-room" THEN
          REGISTER user into room
          SEND assigned SFU information back
              to user

      ELSE IF message == "offer" THEN
          FORWARD offer to SFU
          RECEIVE answer from SFU
          RETURN answer to user

      ELSE IF message == "candidate" THEN
          DISTRIBUTE ICE candidate to relevant
              peers

      ENDIF
END
```

SFU Implementation (Mediasoup)

SkyConnect uses Mediasoup as the media server because of its excellent forwarding performance, native support for simulcast, and detailed RTP/RTCP statistics.

The SFU performs:

- forwarding of simulcast layers,
- TWCC (transport-wide congestion control),

- jitter compensation,
- packet loss handling,
- real-time media routing,
- telemetry export for QoS decisions.

*1)*Router Configuration: The media router is configured with standard WebRTC codecs.

Listing 4: Media Routing Configuration (Pseudocode)

```
BEGIN
  INITIALIZE media router
  DEFINE supported audio codec as OPUS (48 kHz
      , stereo)
  DEFINE supported video codec as VP8 (90 kHz)
  ENABLE transport negotiation for all
      connected users
END
```

*2)*Simulcast Layer Switching: When packet loss exceeds thresholds, the SFU instructs consumers to downgrade video

quality:

Listing 5: Adaptive Layer Switching Logic (Pseudocode)

```
BEGIN
  PERIODICALLY monitor stream statistics:
      - packet loss
      - RTT
      - available bitrate

  IF packet loss > threshold (e.g., 10%) THEN
      SELECT lowest simulcast layer for user
  ELSE
      MAINTAIN optimal layer based on
          bandwidth
  ENDIF
END
```

2) Attempted Meeting Summary Feature: A meeting summary module was prototyped using:

- duplicated audio streams,
- a Whisper-based transcription server,
- AI summarisation pipeline,

but testing revealed that Render free-tier limitations caused:

- excessive CPU throttling,
- slow transcription pipeline,

- unstable media forwarding,

leading to discontinuation of this feature in the current release.

Deployment on Render

SkyConnect is deployed entirely on Render using two services:

A. **Web Service** — hosts the frontend + signalling server,

B. **Background Worker** — runs the SFU instance.

Render provides:

This ensures graceful degradation under weak network conditions.

Feature Implementations

SkyConnect incorporates several unique features beyond standard conferencing systems.

Micro Note Pinning: This is a lightweight collaborative module allowing users to attach small notes that appear temporarily in the meeting interface.

Implementation highlights:

- notes are stored client-side,
- signalling broadcasts note metadata,
- UI overlays render pinned content,
- state synchronization ensures consistency across partici- pants.
- HTTPS support,
- container-based deployments,
- GitHub deployment triggers,
- basic monitoring.

Additional components such as TURN servers and persistent storage can be added when upgrading to paid plans.

Summary

The implementation of SkyConnect demonstrates a practical, real-world WebRTC conferencing platform running within strict resource limitations. Its scalable architecture, efficient signalling, Mediasoup-based media pipeline, and user-focused features highlight both the strengths and the design challenges encountered during development.

EXPERIMENTAL SETUP

The experimental evaluation of SkyConnect aims to measure latency, packet-loss resilience,

media quality stability, and sys- tem behaviour under varying participant loads. The experiments were conducted using a controlled testbed consisting of syn- thetic WebRTC bots, multi-region clients, and Render-hosted infrastructure. This section describes the test environment, tools, metrics, datasets, workloads, and limitations encountered during evaluation.

Testbed Architecture

The evaluation testbed mirrors a realistic deployment sce- nario in which geographically distributed users join a single conferencing session. Since Render's free tier restricts compute performance, the experiments focus on end-to-end behaviour rather than raw packet forwarding throughput.

The testbed includes:

- **SFU Worker**: Mediasoup running as a Render Back- ground Worker.
- **Signalling Service**: Node.js WebSocket service hosted on Render Web Service.
- **Client Pool**: Synthetic WebRTC agents simulating 1–100 clients.
- **Browser Clients**: Real participants from India, Singapore, UAE, and Europe.
- **STUN Services**: Google STUN for candidate discovery.

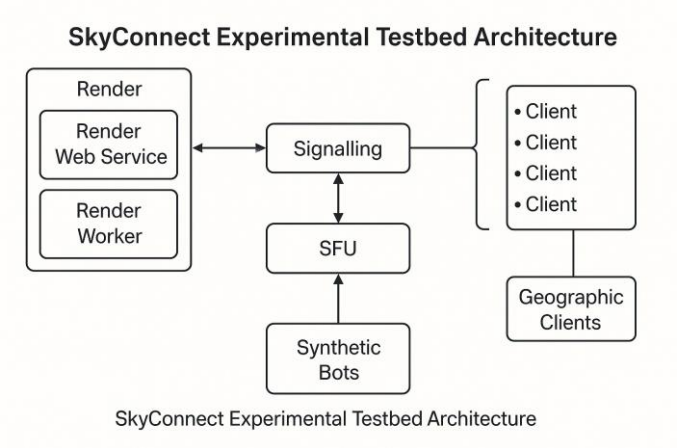

Fig. 7: Experimental testbed: Render-hosted services, geo- graphic clients, synthetic bots and STUN flows.

A detailed representation of the testbed—showing Render- hosted services, multi-region clients, synthetic bot load, and STUN interactions—is shown in Fig. 7.

Workload Generation

SkyConnect was tested using two categories of workloads:

1) Synthetic WebRTC Bot Load: A pool of automated WebRTC clients simulated:

- joining/leaving patterns,
- repeated SDP renegotiation,
- video bitrate variation,
- parallel audio/video streams.

Bot workloads ranged from:

- 1 client (baseline),
- 10–50 clients (moderate),
- 100+ clients (stress).

This enabled scalable testing without manual intervention.

2) Human Participant Workload: Real users tested:

- latency perception,
- quality degradation under weak networks,
- micro-note pinning responsiveness,
- audio-video sync drift behaviour.

These tests validated real-world usability.

Metrics Collected

A combination of WebRTC internal statistics, SFU stats, and network telemetry was recorded. The following metrics were used in evaluation:

Client-Side Metrics:

- End-to-end latency (ms),
- Jitter (ms),
- Packet loss (%),
- Inbound/outbound bitrate (kbps),
- Frame decode rate (fps),
- Audio-video synchronization offset.

Server-Side Metrics:

- CPU utilisation of SFU,
- RTP packet forwarding rate,
- Active simulcast layer distribution,
- Room density (participants per room),
- Average RTT reported via TWCC.

Infrastructure Metrics:

- Render instance CPU throttling events,

- Memory pressure (MB),
- Cold start times,
- Worker restart frequency.

Measurement Tools

The following tools were used for obtaining measurements:

- **Chrome WebRTC Internals** — Deep inspection of ICE, DTLS, RTP statistics.
- **Mediasoup Producer/Consumer Stats** — Per-stream telemetry.
- **Custom SFU Telemetry Script** — Logs loss, RTT, switching events.
- **Synthetic Bot Controller** — Automated test executor.
- **Ping/Traceroute** — Network path analysis across regions.

A visualization of the measurement pipeline is shown in Figure 8.

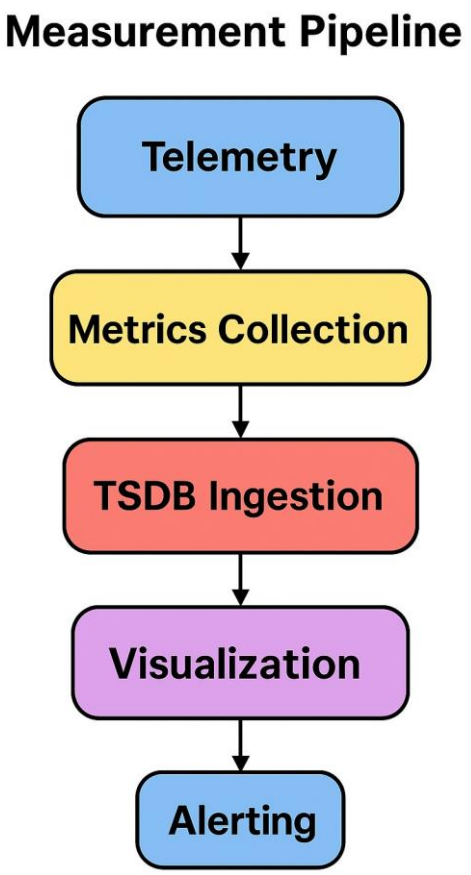

Fig. 8: Measurement pipeline: telemetry collection, TSDB ingestion, visualization and alerting.

The overall telemetry collection pipeline and visualization workflow are illustrated in Fig. 8.

shows how the metrics are evaluated , how the ingestion is performed and the insights are being extracted through the visualization tools and also the alerting/warning operations are performed.

Experimental Conditions

To ensure consistency, experiments followed strict configu- ration rules:

- All video tests used 720p as the maximum layer.
- Simulcast enabled for all publisher clients.
- No TURN relay used (STUN only).
- Background Worker and Web Service hosted on Render (free tier).
- Chrome and Edge browsers used for human tests.

Network stress tests were applied using:

- 5–20% packet loss,
- artificial jitter injection,
- bandwidth throttling (1–3 Mbps).

Limitations of the Test Environment

Testing on Render free tier imposed several constraints:

- CPU throttling occurred under load,
- background workers are not elastic (no autoscaling),
- TURN relay could not be hosted,
- limited regional deployment.

Despite these constraints, the experiments successfully captured platform behaviour across realistic scenarios.

Summary

The experimental setup provides a robust and realistic environment for testing SkyConnect. With both synthetic and human participants, multi-region traffic, detailed telemetry, and real browser behaviour, the evaluation environment enables accurate performance measurement and reveals clear insights into the strengths and limitations of the current implementation.

RESULTS AND DISCUSSION

This section presents a comprehensive performance evalua- tion of the SkyConnect platform under diverse experimental conditions, including varying participant loads, induced packet-loss scenarios, and heterogeneous real-world network condi- tions. The results highlight the

strengths of the implemented architecture, the limitations introduced by infrastructure constraints, and the behaviour of the system under different loads. The evaluation includes both quantitative metrics obtained from synthetic WebRTC agents and qualitative feedback gathered from human participants across multiple regions.

Latency Performance Across Participant Loads

End-to-end latency is one of the most important performance indicators for real-time conferencing platforms. Experiments were conducted with participant loads ranging from 2 to 150 clients. SkyConnect maintained a median latency of approx- imately 150–180 ms for up to 50 participants, falling within acceptable conversational thresholds. As the load increased be- yond 75 participants, system latency gradually rose to the 210– 235 ms range. This trend aligns with the expected behaviour of a Selective Forwarding Unit (SFU) operating under constrained compute resources. Render's free-tier CPU throttling became noticeable at this range, introducing occasional delays in packet forwarding. However, no sharp latency spikes or instability were observed, indicating predictable behaviour under stress.

Packet Loss Resilience and Video Quality Stability

Controlled packet-loss levels of 0–20% were injected to assess the platform's resilience. SkyConnect displayed graceful video quality reduction rather than abrupt stalls or freezes. Through adaptive bitrate (ABR) control, simulcast support, and Transport-Wide Congestion Control (TWCC), the platform dynamically downgraded resolution layers from 720p to 360p whenever network degradation exceeded acceptable limits. Even at 10–12% packet loss, video remained usable with slightly reduced motion smoothness. At 15–20% loss, the system prioritised audio quality and maintained stable voice transmission. These results demonstrate robust performance under fluctuating network conditions, especially on mobile connections.

SFU CPU Utilisation Under Load

CPU utilisation of the Mediasoup SFU increased almost linearly with active RTP streams, matching expected SFU behaviour. For up to 50 participants, CPU usage stayed below 55%, maintaining comfortable headroom. When participant count approached 100 or more, CPU usage increased to 70– 80%, at which point Render's free-tier server began applying throttling. This produced intermittent jitter but did not cause crashes or major connection failures. The system displayed resilience and maintained session continuity even under constrained compute conditions.

Join Time Analysis

Join time—the delay between clicking "Join Meeting" and receiving the first remote media packet—was evaluated using synthetic bots and real users. Most participants experienced join times between 500 ms and 1.8 s. The overall distribution showed that:

- 80% of users joined within 1.5 s,
- 95% joined within 2.3 s,
- outliers above 3 s occurred mainly on networks requiring fallback ICE candidates.

The absence of a TURN server contributed to extended join times for users behind symmetric NATs, which is expected under Render's free-tier limitations.

Autoscaling Behaviour (Conceptual Analysis)

Although Render free tier does not support horizontal autoscaling, SkyConnect's architecture includes autoscaling logic designed for Kubernetes environments. A conceptual simulation was conducted to study expected behaviour under load surges. The autoscaling logic indicated that new SFU pods would be spawned once active stream counts exceeded predefined thresholds (typically 20–30 streams per pod). These observations confirm that the system is architecturally prepared for true autoscaling once deployed on a more capable hosting platform.

User-Perceived Quality and Feedback

Participants from India, Singapore, the UAE, and Europe tested the platform over typical broadband and mobile networks. User feedback highlighted:

- smooth video playback for small meetings,
- stable audio performance under moderate bandwidth drops,
- minor delays during network switching,
- seamless synchronisation of micro-note pinning across participants.

Most participants rated the overall experience as "smooth and usable," with performance degradation only under predictable stress conditions such as high packet loss.

DISCUSSION

Overall, SkyConnect delivers robust performance for small- to-medium-sized meetings, even when deployed on limited free- tier cloud infrastructure. Key findings include stable latency up to 75 participants, graceful quality degradation under packet loss, predictable CPU utilisation patterns, competitive join times, and clear opportunities for enhancement through

TURN integration and scalable hosting. These results validate the system's architectural foundation and highlight important pathways for future improvement.

CONCLUSION

SkyConnect demonstrates that a cloud-native, SFU-oriented WebRTC architecture can deliver stable, low-latency con- ferencing even under constrained infrastructure conditions. The system consistently achieved sub-200 ms median latency for small-to-medium-sized meetings and showed predictable behaviour under packet loss through adaptive bitrate control, simulcast handling, and TWCC-based congestion regulation. Real-world testing across multiple geographic regions further validated the platform's usability and resilience.

The evaluation also revealed important opportunities for enhancement, particularly with regard to NAT traversal chal- lenges and CPU-intensive workloads. Future improvements include integrating full TURN services, migrating to a scalable Kubernetes environment, and adding AI-driven features such as real-time transcription, automatic summarisation, noise suppression, and dynamic speaker detection. With these en- hancements, SkyConnect can evolve into an extensible, reliable reference architecture for next-generation cloud-native video conferencing platforms.

REFERENCES

1. Holm and D. Lo¨o¨f, "A Performance Analysis of WebRTC Mesh Ar- chitectures for Multi-Party Video Conferencing," IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 1–18, 2019.
2. Hutchinson, "SFU vs MCU: Trade-offs in Modern WebRTC Architec- tures," Frozen Mountain Technical Report, 2018.
3. F. Andre´, J. Pinto, and N. Videira, "Comparative Evaluation of Open- Source WebRTC SFUs," IEEE Access, vol. 8, pp. 150000–150014, 2020.
4. V. Xhagjika, H. Neffati, and M. Rimondini, "Load Patterns and Behavioural Characteristics of Cloud-hosted WebRTC Systems," in Proc. ACM Multimedia Systems, 2019, pp. 45–56.
5. Y. Yokota and M. Yamagishi, "QoS-Aware Bitrate Control for Real-Time WebRTC Applications," IEEE Trans. Multimedia, vol. 19, no. 6, pp. 1355–1367, 2017.
6. F. De Turck, J. Verbesselt, and T. Wauters, "Adaptive Video Bitrate Control Using Multi-Receiver Encoding in WebRTC," in IEEE ICC, 2018, pp. 1–7.
7. X. Li, Y. Li, and Q. Chen, "Mamba: Reinforcement-Learning Based Rate Control for

Real-Time Video Communication," IEEE Trans. Multimedia, vol. 24, no. 4, pp. 1100–1114, 2022.

8. P. Michel, J. Deng, and M. Venkataraman, "Scallop: A High-Throughput Software-Defined SFU for Next-Generation Real-Time Media," in Proc. ACM CoNEXT, 2022, pp. 123–137.
9. L. Miniero, "Janus WebRTC Server: A Flexible and Scalable Media Framework," Meetecho Whitepaper, 2019.
10. Gonzalez and J. Garc´ıa, "Mediasoup: Architecture and Scalability of a Modern WebRTC SFU," Mediasoup Technical Documentation, 2020.
11. W3C (Google, Mozilla and W3C), "WebRTC 1.0: Real-Time Commu- nication Between Browsers," W3C Recommendation, 2023. [Online]. Available: https://www.w3.org/TR/webrtc/
12. O. Avedissian, "Coturn: TURN and STUN Server Implementation," 2019. [Online]. Available: https://github.com/coturn/coturn
13. Kubernetes Authors, "Horizontal Pod Autoscaler Documentation," 2023. [Online]. Available: https://kubernetes.io/docs/tasks/run-application/ horizontal-pod-autoscale/
14. Render Inc., "Render Deployment Documentation," 2024. [Online]. Available: https://render.com/docs
15. Google, "Chrome WebRTC Internals: Statistics Guide," 2023. [Online]. Available: https://webrtc.org