CASE STUDY 02 OF 07 | CONNECT APP

Video CallingAcross Channels, Groups & DMs

High-quality video calls embedded directly into the communication flow

#Video #WebRTC #RealTime #Calling

Overview

Video communication is not a separate tool in Connect. It is built directly into the messaging experience. Any conversation, whether it is a channel, a group chat, or a direct message, can be escalated to a video call with a single click. No new tab, no separate app, no link sharing required.

This case study covers the engineering behind video calling in Connect: how calls are initiated, how multiple participants join, how the system handles varying network conditions, and how the calling infrastructure integrates cleanly with the rest of the platform.

Project Snapshot

Project	Connect — Workplace Communication Platform
Feature Area	Video Calling: Channels, Groups, Direct Messages
Role	Senior Fullstack Engineer
Protocol	WebRTC with SFU (Selective Forwarding Unit) media server
Call Types	1:1 direct calls, small group calls, channel-wide calls
Outcome	Low-latency video calls embedded in every conversation type

The Problem

In most organizations, video calls happen in one tool and text communication happens in another. Starting a call means opening Zoom or Google Meet, creating a meeting, copying the link, pasting it into a chat, and waiting for people to click through and join. That friction adds up across dozens of calls per week.

Connect's goal was to make video calls feel like a natural part of the conversation. If you are in a DM with someone and want to switch from text to video, that should take one action. If a channel needs to jump on a call, it should happen in context. The technical challenge was building video

infrastructure that could support this without requiring a separate service or breaking the flow of the app.

What Was Built

Call Initiation From Any Conversation

A video call button is present in every channel, group, and DM. Clicking it initiates a call within that conversation context. Other participants see a call notification inside the conversation and can join with one click. There is no separate invite flow or link generation step.

Direct 1:1 Calls

In a direct message, starting a call rings the other person with an in-app notification. If they are offline, they receive a missed call indicator when they return. 1:1 calls use a peer-to-peer WebRTC connection when both parties are on compatible networks, minimizing latency and server load for the most common call type.

Group and Channel Calls

Group calls and channel calls involve multiple participants, which requires a media server to manage the connections. Connect uses an SFU (Selective Forwarding Unit) architecture, where each participant sends their stream once to the server, and the server selectively forwards streams to other participants based on what they need to receive. This scales significantly better than a full mesh approach for groups larger than three or four people.

In-Call Chat

While on a call, participants can send text messages in the call chat panel without ending the call or switching to the conversation view. Call chat messages are stored in the conversation history alongside regular messages, so anything shared during a call is available after it ends.

Technical Architecture

WebRTC Signaling Call negotiation uses WebRTC signaling over the existing WebSocket connection. Offer, answer, and ICE candidate exchange happen through the same real-time channel as messages, so no separate signaling server is needed.	**SFU Media Server** A dedicated media server handles multi-party calls. Each participant's audio and video streams are forwarded selectively, with simulcast support so the server can send different resolution streams based on each receiver's bandwidth.
Adaptive Bitrate The SFU monitors network conditions per participant and adjusts the forwarded stream quality dynamically. Participants on slower connections receive lower-resolution streams without affecting the quality experienced by others.	**TURN Server Fallback** For users behind strict firewalls or NAT that block direct peer-to-peer connections, the system falls back to relaying media through TURN servers automatically, ensuring calls work in any network environment.
Call State Synchronization Active call state (who is in the call, mute/camera status, duration) is synchronized across all	**Call History** Every call is logged with participants, duration, and start time. This record is visible in the conversation

participants via the real-time layer. Joining mid-call gives the new participant the current state immediately without a full reconnect.	history and in the user's call log, with missed calls surfaced prominently.

Key Challenges

NAT Traversal and Firewall Compatibility

Corporate networks often block the ports WebRTC uses by default. A multi-layer connectivity strategy was implemented: try direct peer-to-peer first, fall back to STUN-assisted connection if NAT is involved, then fall back to TURN relay if the direct path is unavailable. The fallback chain is automatic and invisible to the user, with the best available path selected per connection.

Call Quality Under Variable Network Conditions

A participant dropping from 50 Mbps to 2 Mbps mid-call should not end the call or cause everyone else's experience to degrade. The adaptive bitrate system continuously monitors round-trip time and packet loss per participant and adjusts the stream quality it receives accordingly. Audio is always prioritized over video when bandwidth is constrained, since a degraded video signal is more tolerable than broken audio.

Joining Mid-Call State Recovery

When a participant joins a call that has been running for some time, they need to immediately see who else is in the call, who is muted, who has their camera off, and how long the call has been active. This state is maintained in a shared call session object on the server and pushed to each new joiner as part of the join confirmation, so there is no delay or confusion about the current call state.

Outcome

Video calling in Connect removed the friction of switching between a chat tool and a video tool entirely. Users could go from text to video within the same conversation in a single click, and back again without losing context. The SFU architecture handled group calls with stable quality, and the adaptive bitrate system kept calls usable even for participants on congested or mobile networks. Integration with the in-call chat meant nothing shared during a call was lost after it ended.

Technical Highlights

SFU architecture supporting group calls with adaptive per-participant stream quality
STUN and TURN fallback chain ensuring calls work across corporate firewalls and NAT
Simulcast support allowing different resolutions to different participants simultaneously
Zero-friction call initiation from every conversation type in the app