

**System Design**

**From Code to Business Value**

**Architectural Insights**

# **WebRTC vs. mediasoup**

**Audience:**

MobileEngineering,BackendCall

**Infrastructure,TechnicalLeadership**

**Date: 2026-07-28**

**by Esmaeel Moradi**

Senior Android Engineer



# WebRTC vs. mediasoup

Architectural Reference & E2EE Engineering Spike

**Audience:** Mobile Engineering, Backend Call Infrastructure, Technical Leadership

**Date:** 2026-07-28



Esmaeel Moradi

Senior Android Engineer

# 1. Executive Summary & Core Concept

WebRTC is the on-device audio/video engine. mediasoup is the SFU architecture that connects that engine to the server. They are not alternatives to one another — they are layered: one runs inside the other, at different points of the media path.

**Core concept:** WebRTC is the engine and the wheels. mediasoup is the highway and the central junction. Every car (client) drives to the junction; the junction routes traffic between cars. Without WebRTC there is nothing to hear or see. Without mediasoup — in our current architecture — a 1:1 or group call never makes it past the SFU hop.

## Mental model

| Role                 | Analogy                    | What it actually does                                                      |
|----------------------|----------------------------|----------------------------------------------------------------------------|
| WebRTC               | Engine + wheels            | Microphone capture, speaker playback, codecs, RTP packets                  |
| mediasoup            | Highway + central junction | Every client connects to the SFU; the SFU forwards traffic between clients |
| call:* socket events | Traffic radio              | Who joined, who is producing, who should consume, where TURN is            |

# 2. Deep Dive: WebRTC vs. mediasoup in Our Android Architecture

## 2.1 WebRTC — the media engine

WebRTC is the standard browser/mobile library for real-time media. In this codebase it is responsible for:

- Capturing the microphone and camera
- Codecs — Opus for audio, VP8/H264 for video
- Network connectivity — ICE, STUN/TURN, DTLS
- Sending and receiving RTP packets

Key class: `core/data/.../webrtc/WebRTCManager.kt`, alongside `PeerConnectionFactory`, `AudioTrack`, and `VideoTrack`. This layer's job is simply: capture audio, play audio, monitor quality.

## 2.2 mediasoup — the SFU

mediasoup is a Selective Forwarding Unit (SFU) for WebRTC. Each client connects to the server rather than connecting directly to every peer (as in pure mesh P2P). The server does not decrypt media content — in the normal path it only forwards packets — but those packets do pass through the server's infrastructure.

|         |          |           |        |          |         |
|---------|----------|-----------|--------|----------|---------|
| Phone A | --RTP--> | mediasoup | server | --RTP--> | Phone B |
| Phone A | <--RTP-- | mediasoup | server | <--RTP-- | Phone B |

Key classes on the client:

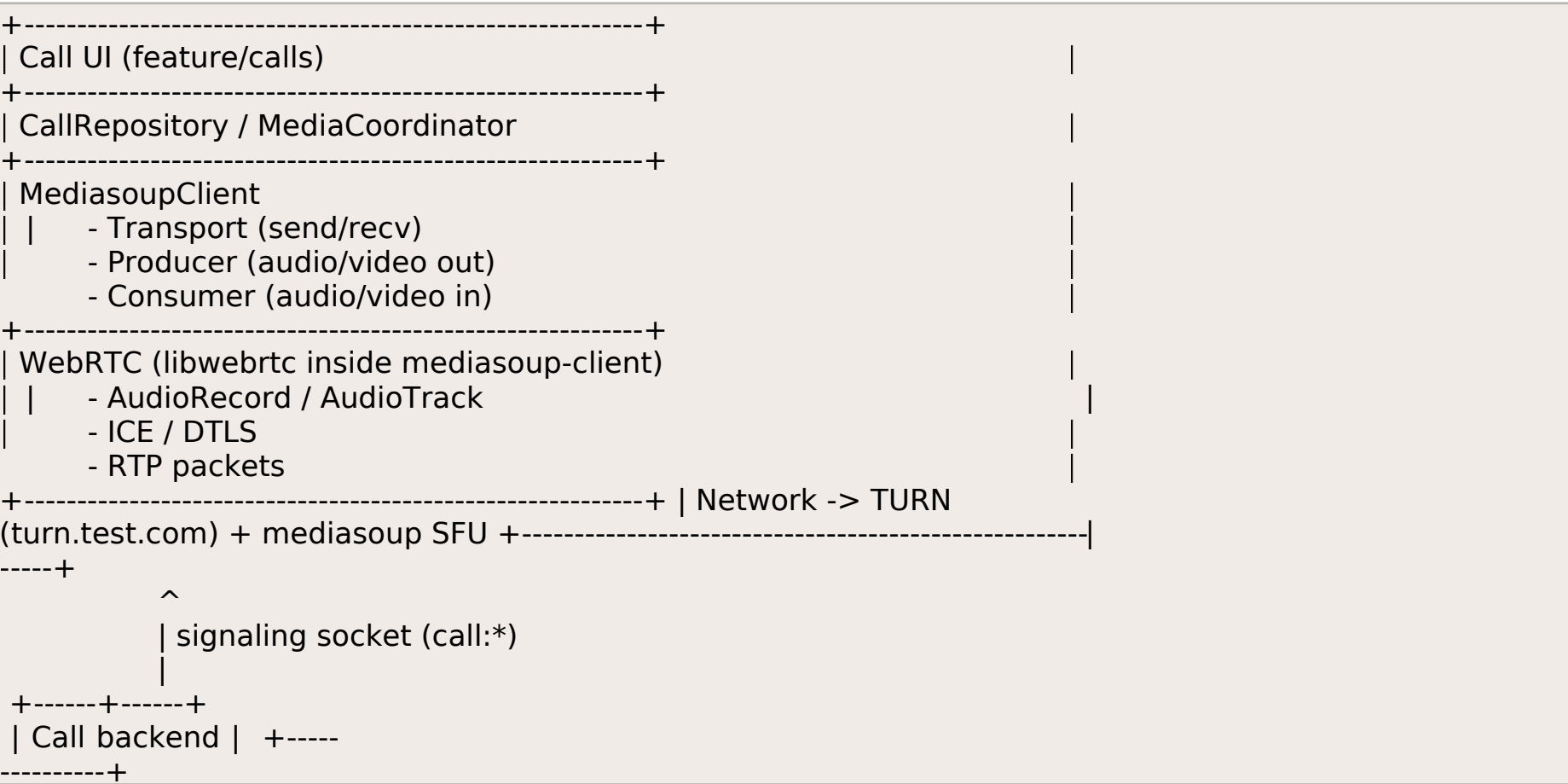
- `MediasoupClient.kt` — SFU coordination on the client
- `MediasoupProducerManager` — sends media to the SFU (Producer)
- `MediasoupConsumerManager` — receives media from the SFU (Consumer)
- `MediasoupTransportManager` — `SendTransport` / `RecvTransport`

Corresponding signaling events:

call:join · call:create:webrtc · call:connect:webrtc · call:produce · call:consume  
call:get:producers · call:new:producer · call:resume

### 3. Complete Layered Architecture & Signal Flow

#### 3.1 Layered architecture stack



#### 3.2 Step-by-step audio call flow

- 1.call:start / call:incoming / call:accepted are exchanged (plus TURN credentials).
- 2.Each side emits call:join and creates its send/recv transports.
- 3.Each side produces audio; the SFU returns a producerId.
- 4.The other side learns about it via call:new:producer or call:get:producers, and consumes it.
- 5.RTP flows: phone <-> SFU <-> phone.
- 6.WebRTCManager reads the microphone and plays back through the speaker.

### 4. The Misconception & Logging Map

mediasoup-client on Android bundles WebRTC internally and uses it directly — so it is never a case of “WebRTC or mediasoup.” mediasoup is managing the same WebRTC engine for an SFU topology. Both layers appear in the same log stream because they are active simultaneously, not because one replaced the other.

| Log line                                         | Layer                           |
|--------------------------------------------------|---------------------------------|
| WebRTCManager: AudioRecord started               | WebRTC                          |
| WebRTCManager: AudioTrack playback started       | WebRTC                          |
| SendTransportListener: ICE connected             | WebRTC, via mediasoup transport |
| MediasoupProducerManager: Audio producer created | mediasoup                       |

| Log line                                           | Layer               |
|----------------------------------------------------|---------------------|
| MediasoupConsumerManager: Consuming producerId=... | mediasoup           |
| call:produce / call:consume                        | mediasoup signaling |

## 5. Advanced Engineering Spike: E2EE Call Integration

Call encryption is split into two independent layers with very different levels of maturity.

| Layer                                                                          | Status (July 2026)                       |
|--------------------------------------------------------------------------------|------------------------------------------|
| 1. Media key exchange (callMediaKey) via Signal, on call:start / call:incoming | Implemented and verified on real devices |
| 2. Encrypting the RTP frames themselves before they leave the device           | Not yet implemented                      |

Layer 2 is typically implemented on WebRTC using FrameEncryptor / FrameDecryptor attached to RtpSender / RtpReceiver.

**Blocker:** The Java API surface of Producer / Consumer in mediasoup-client 3.4.0 does not expose the underlying RtpSender / RtpReceiver to the app, even though the bundled libwebrtc itself has that API. Full details in call-e2ee-spike-notes.md.

Current security posture:

- The E2EE key is exchanged between the two devices
- Audio/video packets are still plaintext RTP on the SFU path — the server does not read the content, but could observe or record it if it chose to

### 5.1 Pure P2P WebRTC vs. the current model

|                  | Pure P2P WebRTC              | Current model (mediasoup SFU)                               |
|------------------|------------------------------|-------------------------------------------------------------|
| Media path       | Phone <-> phone              | Phone <-> SFU <-> phone                                     |
| Restrictive NAT  | Usually depends on TURN      | SFU + TURN; scales better for groups                        |
| Group calls      | Hard (mesh)                  | Natural fit for an SFU                                      |
| Frame-level E2EE | Direct on the PeerConnection | Must be reached through the mediasoup API down to RtpSender |

This project has chosen the SFU path for calls. Switching abruptly to pure P2P means rewriting the call architecture — it is not a one-line flag flip.

### 5.2 Spike resolution paths

The question is not “should we drop WebRTC” — it's how to reach RtpSender / RtpReceiver through mediasoup so a FrameEncryptor can be attached to the same WebRTC instance that is already sending media. Three paths are under consideration:

Patch or fork mediasoup-client-android to expose the sender/receiver.

Upgrade to a mediasoup-client version that already exposes the needed API.

Evaluate an alternative path (e.g. a higher-level FrameCryptor) if one is available in our stack.

**Decision gate:** Do not start a full AES implementation on voice/video until the spike above has produced an answer.

## 6. Summary for Technical Decision Makers & Key Repository References

Bottomline: WebRTC and mediasoup are not competing technologies in this codebase— they are two layers of the same call stack, and the open engineering question is how far into mediasoup's API surface we can reach to attach frame-level encryption to the WebRTC layer it already manages.



**Esmaeel Moradi**

Senior Android Engineer

**System Design**

From Code to Business Value

**Architectural Insights**

# Let's Build a Scalable Future

by Esmaeel Moradi  
Senior Android Engineer

**Let's Connected**