**# LAN-Based Video Calling & Chat System (Offline WebRTC)**

**1. Overview**

This document explains how to build a **LAN-based video calling and chat system** using **mDNS, WebSockets, and WebRTC** without internet dependency. The goal is to enable devices on the same local network to **discover each other, establish a connection, and communicate in real-time**.

**2. Key Technologies**

|  |  |
| --- | --- |
| **Technology** | **Purpose** |
| **mDNS** | Peer Discovery - Finds devices on LAN without needing an IP address. |
| **WebSocket** | Signaling - Exchanges WebRTC connection details between peers. |
| **WebRTC** | Peer-to-Peer Communication - Enables video, audio, and text chat. |

**3. System Architecture**

**Step 1: Peer Discovery Using mDNS**

* mDNS allows peers to find each other using hostnames (e.g., device-1.local) instead of manually entering IP addresses.
* Alternative: If mDNS is not used, users must manually enter LAN IPs.

**Step 2: WebSocket Signaling**

* Since WebRTC **does not handle peer discovery**, we need WebSockets to **exchange connection details (SDP, ICE candidates)**.
* A **LAN-based WebSocket server** helps clients communicate initially.
* Once WebRTC establishes a connection, WebSockets **are no longer needed**.

**Step 3: WebRTC Peer-to-Peer Connection**

* WebRTC enables **direct** communication between users **without a server**.
* Used for **video, audio, and chat messages**.

**4. Implementation**

**4.1. Setting Up mDNS (Multicast DNS)**

**Server-Side (Node.js mDNS Setup)**

const mdns = require('multicast-dns')();

mdns.on('query', function (query) {

if (query.questions[0].name === 'video-call.local') {

mdns.respond({

answers: [{ name: 'video-call.local', type: 'A', data: '192.168.1.100' }]

});

}

});

**4.2. WebSocket Signaling Server**

const WebSocket = require('ws');

const server = new WebSocket.Server({ port: 3000 });

server.on('connection', (socket) => {

socket.on('message', (message) => {

server.clients.forEach(client => {

if (client !== socket && client.readyState === WebSocket.OPEN) {

client.send(message);

}

});

});

});

**4.3. WebRTC Client-Side Code**

const peer = new RTCPeerConnection();

navigator.mediaDevices.getUserMedia({ video: true, audio: true }).then(stream => {

document.getElementById('video').srcObject = stream;

stream.getTracks().forEach(track => peer.addTrack(track, stream));

});

peer.ontrack = (event) => {

document.getElementById('remoteVideo').srcObject = event.streams[0];

};

**5. Security & Networking Considerations**

**5.1. Handling Windows Firewall Rules**

netsh advfirewall firewall add rule name="WebRTC LAN" dir=in action=allow protocol=UDP localport=3478

**5.2. Ensuring Local Communication Only**

* Configure the WebSocket server to **reject external connections**.
* Use LAN-specific IP ranges (e.g., 192.168.x.x).

**6. Conclusion**

This system enables **offline peer-to-peer communication** over LAN using **mDNS, WebSockets, and WebRTC**. This approach eliminates the need for an internet connection while maintaining high-quality **video, audio, and messaging capabilities**.