Rapport SAE5.ROM.03

PLUVIOSE Louis - KARAPETYAN Mikhail

Application de communication WebRTC

Sujet proposé par : Monsieur Philipe Hensel

Université de Haute-Alsace
Institut Universitaire de Technologie de Colmar
Département Réseaux et Télécommunications

7 décembre 2023

Table des matières

LISTINGS

Table des figures

1 Introduction

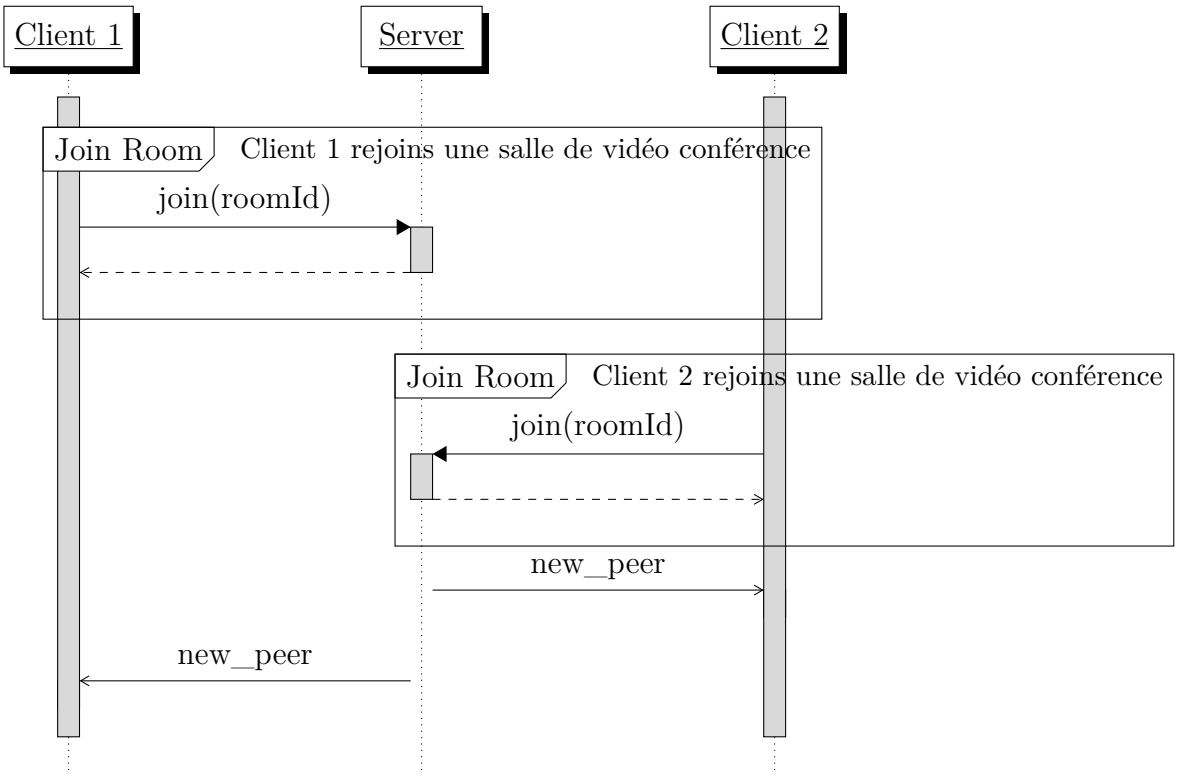
Dans un monde de plus en plus connecté, la communication en temps réel via Internet est devenue une nécessité cruciale, tant pour les interactions personnelles que professionnelles. Notre projet pour la SAE5.ROM.03, s'inscrit dans cette dynamique en offrant une solution de communication basée sur la technologie WebRTC (Web Real-Time Communication). Nous avons voulu via ce projet nous lancer un défi : celui de permettre des échanges audio et vidéo en temps réel directement depuis le navigateur web, sans nécessité de télécharger des logiciels tiers ou de créer des comptes d'utilisateur.

La technologie WebRTC, un standard ouvert et gratuit, permet de réaliser des appels vidéo et audio de haute qualité avec une faible latence, garantissant ainsi une communication fluide et efficace. Notre application SAE5.ROM.03 est conçue pour être intuitive et facilement accessible, offrant une interface utilisateur élégante et des fonctionnalités adaptées à divers contextes, que ce soit pour des réunions, des sessions de travail collaboratif, ou des conversations personnelles.

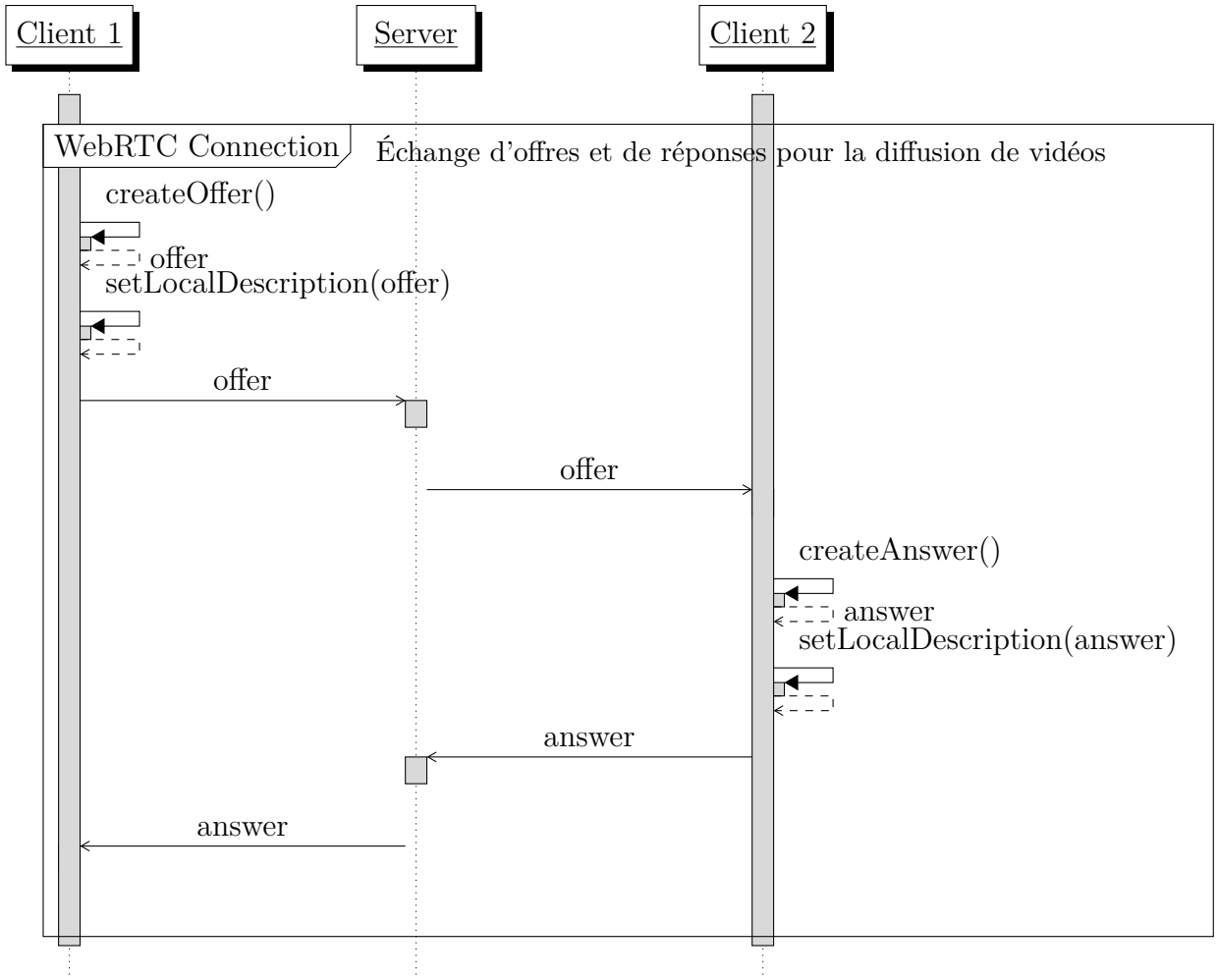
L'accent est mis sur la facilité d'utilisation. Les utilisateurs peuvent créer et rejoindre des salles de conférence virtuelles en quelques clics, tout en bénéficiant d'une connexion fiable. De plus, l'application prend en charge plusieurs participants.

2 Diagrammes de séquence

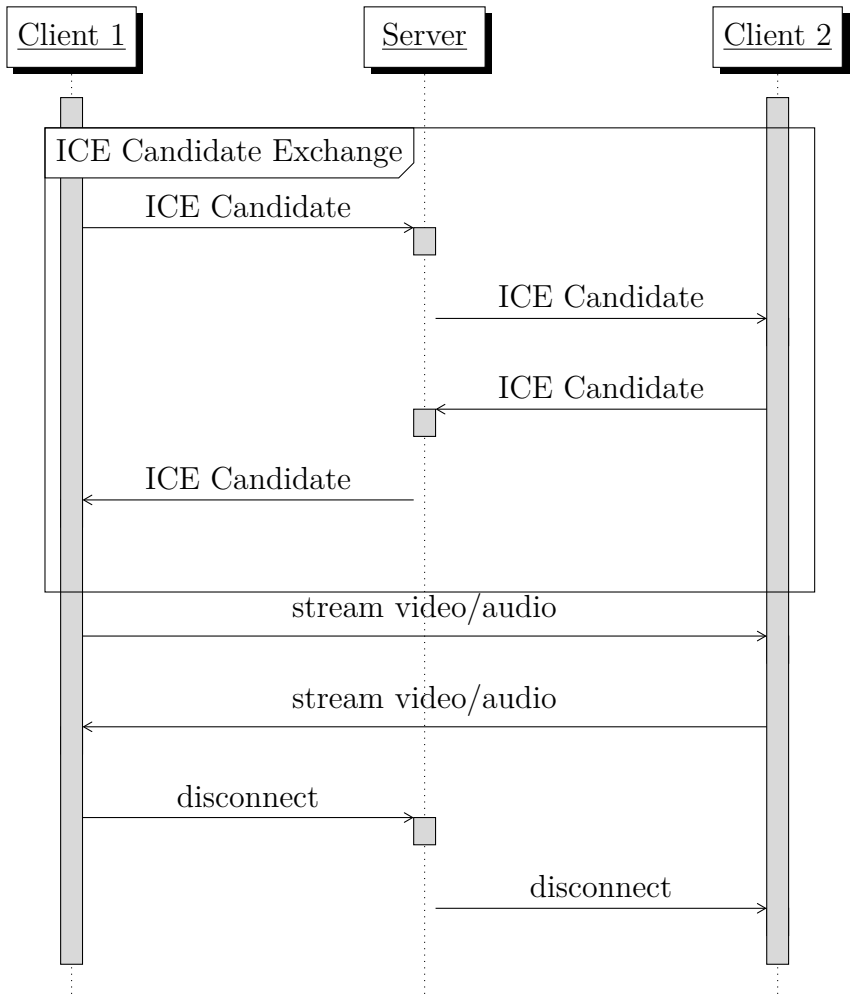
2.1 Connexion à la salle de réunion



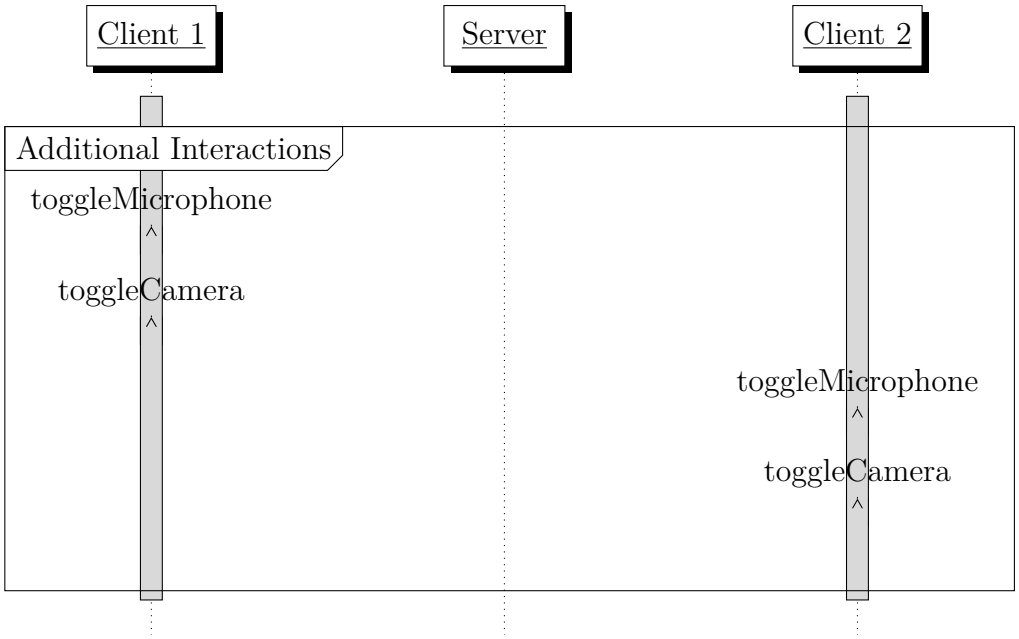
2.2 Connection WebRTC



2.3 Échanges de candidats ICE



2.4 fonctionnalités supplémentaires



3 Utilisation de l'application

4 Fonctionnement de l'application

4.1 DOM Elements

Cette section récupère et stocke des références à divers éléments du DOM qui seront manipulés ou utilisés tout au long du script.

```
1  const roomSelectionContainer =
       document.getElementById('room-selection-container');
2  const roomInput = document.getElementById('room-input');
3  const connectButton =
       document.getElementById('connect-button');
4  const videoChatContainer =
       document.getElementById('video-chat-container');
5  const localVideoComponent =
       document.getElementById('local-video');
```

1: DOM Elements

4.2 Variables

```
1  const socket = io();
2  const mediaConstraints = { audio: true, video: { width:
       1280, height: 720 } };
3  let localStream;
4  let roomId;
5  let peerConnections = {}; // Dictionary to hold all peer
       connections
6
7  const iceServers = {
8      iceServers: [
9          { urls: 'stun:stun.l.google.com:19302' }, //
               Serveur STUN existant
10         // Ajout de la configuration TURN
11         {
12             urls: 'turn:relay1.expressturn.com:3478', //
                   URL du serveur TURN
13             username: 'efJJ0L80U0GANH5V0A', // Nom
                   d'utilisateur
14             credential: 'LO5Tdr8aoKohTHDL' // Mot de passe
15         }
16     ]
17 };
```

2: Variables

— `socket` : Crée une connexion socket.io pour la communication en temps réel avec le serveur.
— `mediaConstraints` : Spécifie les contraintes des médias (audio et vidéo) pour WebRTC.
— `localStream` : Représente le flux média local (audio et vidéo) de l'utilisateur.
— `roomId` : Stocke l'ID de la salle de chat actuelle.
— `peerConnections` : Un dictionnaire pour stocker les connexions peer WebRTC.
— `iceServers` : Contient les serveurs STUN et TURN utilisés pour la traversée de NAT et le relais.

4.3 Button Listeners

```
1   connectButton.addEventListener('click', () => {
2   joinRoom(roomInput.value);
3 });
4
5 const hangUpButton =
      document.getElementById('hangup-button');
6 const toggleMicButton =
      document.getElementById('toggle-mic-button');
7 const toggleCameraButton =
      document.getElementById('toggle-camera-button');
8
9 hangUpButton.addEventListener('click', hangUpCall);
10 toggleMicButton.addEventListener('click', toggleMicrophone);
11 toggleCameraButton.addEventListener('click', toggleCamera);
12
13 let isRoomCreator = false;
```

3: Button Listeners

- `connectButton.addEventListener` : Écouteur d'événements pour le bouton de connexion pour rejoindre une salle.
- `hangUpButton.addEventListener` : Écouteur d'événements pour le bouton de raccrochage.
- `toggleMicButton.addEventListener` : Écouteur d'événements pour activer/désactiver le microphone.
- `toggleCameraButton.addEventListener` : Écouteur d'événements pour activer/désactiver la caméra.

4.4 Socket Event Callbacks

```
  socket.on('room_created', async () => {
  console.log('Socket event callback: room_created');
  await setLocalStream(mediaConstraints);
  isRoomCreator = true;
});

socket.on('room_joined', async () => {
  console.log('Socket event callback: room_joined');
  await setLocalStream(mediaConstraints);
  isRoomCreator = false;
  socket.emit('start_call', roomId);
});

socket.on('full_room', () => {
  console.log('Socket event callback: full_room');
  alert('The room is full, please try another one');
});

socket.on('start_call', async () => {
  console.log('Socket event callback: start_call');
  if (isRoomCreator) {
    createPeerConnections();
  }
});

socket.on('webrtc_offer', async (data) => {
  console.log('Socket event callback: webrtc_offer');

  if (!localStream) {
    console.log("Waiting to set local stream before handling
       offer");
    await setLocalStream(mediaConstraints);
  }

  if (!peerConnections[data.peerId]) {
    await setupPeerConnection(data.peerId, false); // false
       = not an initiator
  }
  await handleOffer(data);
});

socket.on('webrtc_answer', (data) => {
  console.log('Socket event callback: webrtc_answer');
  handleAnswer(data);
});

socket.on('webrtc_ice_candidate', (data) => {
```

```
46    console.log('Socket event callback: webrtc_ice_candidate');
47    handleIceCandidate(data);
48  });
49
50  socket.on('new_peer', async (peerId) => {
51    console.log('Socket event callback: new_peer');
52    await createPeerConnection(peerId, false);
53  });
```

4: Socket Event Callbacks

- `socket.on('room_created')` : Gère l'événement de création de salle.
- `socket.on('room_joined')` : Gère l'événement de rejoindre une salle.
- `socket.on('full_room')` : Gère l'événement lorsque la salle est pleine.
- `socket.on('start_call')` : Gère le début d'un appel WebRTC.
- `socket.on('webrtc_offer')` : Gère la réception d'une offre WebRTC.
- `socket.on('webrtc_answer')` : Gère la réception d'une réponse WebRTC.
- `socket.on('webrtc_ice_candidate')` : Gère la réception d'un candidat ICE.
- `socket.on('new_peer')` : Gère l'ajout d'un nouveau pair dans la salle.

4.5 title

```
1  const socket = io();
2  const mediaConstraints = { audio: true, video: { width:
       1280, height: 720 } };
3  let localStream;
4  let roomId;
5  let peerConnections = {}; // Dictionary to hold all peer
       connections
6
7  const iceServers = {
8      iceServers: [
9          { urls: 'stun:stun.l.google.com:19302' }, //
               Serveur STUN existant
10         // Ajout de la configuration TURN
11         {
12             urls: 'turn:relay1.expressturn.com:3478', //
                   URL du serveur TURN
13             username: 'efJJ0L80U0GANH5V0A', // Nom
                   d'utilisateur
14             credential: 'LO5Tdr8aoKohTHDL' // Mot de passe
15         }
16     ]
17 };
```

5: Variables

— `socket` : Crée une connexion socket.io pour la communication en temps réel avec le serveur.
— `mediaConstraints` : Spécifie les contraintes des médias (audio et vidéo) pour WebRTC.
— `localStream` : Représente le flux média local (audio et vidéo) de l'utilisateur.
— `roomId` : Stocke l'ID de la salle de chat actuelle.
— `peerConnections` : Un dictionnaire pour stocker les connexions peer WebRTC.
— `iceServers` : Contient les serveurs STUN et TURN utilisés pour la traversée de NAT et le relais.

Pour obtenir un serveur TURN gratuitement et rapidement, nous avons utilisé le service ExpressTURN : https://www.expressturn.com/. Il fournit un serveur TURN gratuit et public avec une limite de 500 MB par mois.

4.6 title

```
1  const socket = io();
2  const mediaConstraints = { audio: true, video: { width:
       1280, height: 720 } };
3  let localStream;
4  let roomId;
5  let peerConnections = {}; // Dictionary to hold all peer
       connections
6
7  const iceServers = {
8      iceServers: [
9          { urls: 'stun:stun.l.google.com:19302' }, //
               Serveur STUN existant
10         // Ajout de la configuration TURN
11         {
12             urls: 'turn:relay1.expressturn.com:3478', //
                   URL du serveur TURN
13             username: 'efJJ0L80U0GANH5V0A', // Nom
                   d'utilisateur
14             credential: 'LO5Tdr8aoKohTHDL' // Mot de passe
15         }
16     ]
17 };
```

6: Variables

— `socket` : Crée une connexion socket.io pour la communication en temps réel avec le serveur.
— `mediaConstraints` : Spécifie les contraintes des médias (audio et vidéo) pour WebRTC.
— `localStream` : Représente le flux média local (audio et vidéo) de l'utilisateur.
— `roomId` : Stocke l'ID de la salle de chat actuelle.
— `peerConnections` : Un dictionnaire pour stocker les connexions peer WebRTC.
— `iceServers` : Contient les serveurs STUN et TURN utilisés pour la traversée de NAT et le relais.

4.7 title

4.8 title

4.9 title

4.10 title

5 Annexes

5.1 Sous-titre 1

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse auctor elit vitae mauris dignissim bibendum. Fusce facilisis, sapien sed varius finibus, quam lacus auctor lorem, ac dapibus sapien ante quis odio. Sed tincidunt pharetra dui, in ultricies enim tincidunt ac. Suspendisse quis tincidunt justo. Duis interdum vitae ipsum ac venenatis. Nullam bibendum ex ac nisl tristique, vel euismod ex faucibus.

5.2 Sous-titre 2

```
1  def hello(name):
2    print("Hello, " + name + "!")
3
4  hello("World")
```

7: Exemple de code Python