# TP3 – Plateforme WebRTC P2P
# Communication multimédia pair-à-pair en temps réel

Abdessabour Ahzab,Dadda mohamed laghdaf

Année universitaire 2025–2026

## Table des matières

**1 Capture des flux audio et vidéo** **2**
- 1.1 Objectif . . . 2
- 1.2 Description . . . 2
- 1.3 Interface utilisateur . . . 2
- 1.4 Résumé . . . 3

**2 Échange de données via RTCDataChannel** **3**
- 2.1 Objectif . . . 3
- 2.2 Description . . . 3
- 2.3 Interface utilisateur . . . 3
- 2.4 Résumé . . . 3

**3 Signalisation WebRTC avec Node.js et Socket.io** **4**
- 3.1 Objectif . . . 4
- 3.2 Description . . . 4
- 3.3 Interface utilisateur . . . 4
- 3.4 Résumé . . . 4

**4 Intégration Vidéo + Chat P2P** **5**
- 4.1 Objectif . . . 5
- 4.2 Description . . . 5
- 4.3 Interface utilisateur . . . 5
- 4.4 Résumé . . . 5

**5 Extensions multipoints et comparaison des technologies de signalisation** **6**
- 5.1 Objectif . . . 6
- 5.2 Description . . . 6
- 5.3 Interface utilisateur . . . 6
- 5.4 Résumé . . . 7

# Introduction générale

Ce TP a pour but de mettre en place une application WebRTC permettant à deux pairs (dans des navigateurs web) d'échanger des flux audio/vidéo et des messages texte en temps réel. L'application repose sur quatre briques principales :

— la capture des flux locaux via `getUserMedia()`,
— l'échange de données arbitraires via `RTCDataChannel`,
— la signalisation entre navigateurs à l'aide d'un serveur Node.js + Socket.io,
— l'intégration finale vidéo + chat, prolongée par une extension multipoints.

Les sections suivantes reprennent ces étapes dans l'ordre de l'énoncé du TP, puis présentent l'extension multiparty réalisée.

# 1 Capture des flux audio et vidéo

## 1.1 Objectif

Capturer le flux vidéo du pair local (et l'audio en option) à partir de la webcam, et l'afficher dans la page web pour vérifier le bon fonctionnement de la capture multimédia.

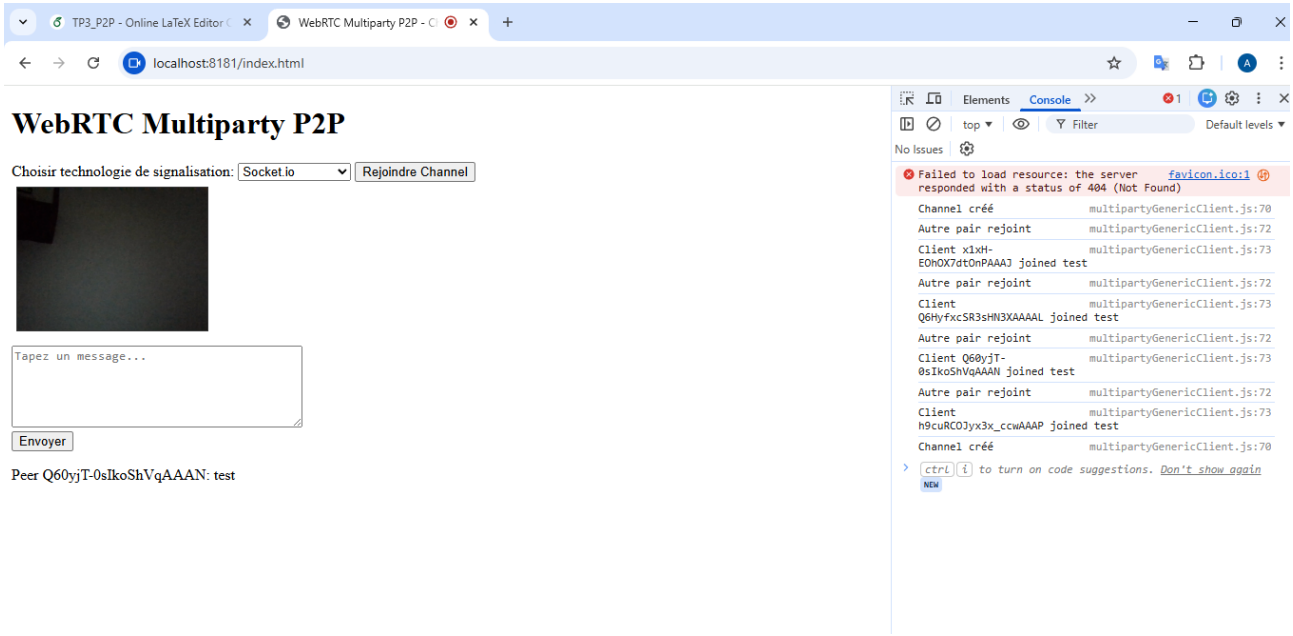
## 1.2 Description

La capture est réalisée en utilisant l'API `MediaStream` de WebRTC. Le code JavaScript :

— demande l'autorisation d'accéder à la caméra (et éventuellement au micro),
— applique des contraintes simples (activation de la vidéo, résolution par défaut),
— injecte le flux obtenu dans une balise `<video autoplay>` de la page.

## 1.3 Interface utilisateur

L'interface se limite à un titre et à une vidéo montrant le flux du pair local.




figureCapture du flux vidéo local avec `getUserMedia()`

## 1.4 Résumé

Cette première étape sert à valider l'accès aux périphériques multimédia et à vérifier que le navigateur peut afficher le flux local avant toute connexion P2P.

# 2 Échange de données via RTCDataChannel

## 2.1 Objectif

Permettre l'échange de messages texte entre deux pairs via un canal de données P2P fiable basé sur `RTCPeerConnection` et `RTCDataChannel`.

## 2.2 Description

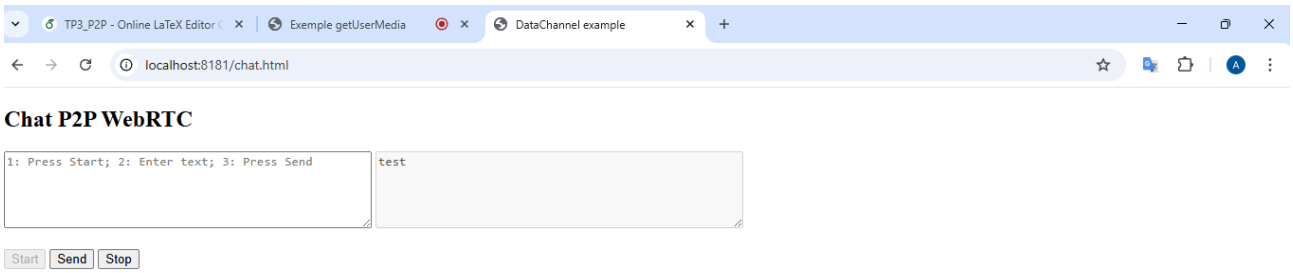
Le code met en place deux objets `RTCPeerConnection` (local et distant) dans le même navigateur, crée un `DataChannel` côté local, et relie les deux pairs en échangeant les descriptions SDP et les ICE candidates. Une fois le canal ouvert, l'utilisateur peut envoyer des messages texte qui sont reçus en face distante.

## 2.3 Interface utilisateur

L'interface contient :

— deux zones de texte : une pour saisir les messages à envoyer, une pour afficher les messages reçus,

— trois boutons : `Start`, `Send`, `Stop`.




figureÉchange de messages via `RTCDataChannel`

## 2.4 Résumé

Cette étape permet de tester la communication P2P directe pour des données texte, indépendamment de la vidéo, tout en manipulant la logique de `RTCPeerConnection` côté client.

# 3 Signalisation WebRTC avec Node.js et Socket.io

## 3.1 Objectif

Mettre en place un canal de signalisation entre navigateurs pour échanger les descriptions SDP et les ICE candidates nécessaires à l'établissement d'une session WebRTC P2P.

## 3.2 Description

Un serveur Node.js est démarré sur le port 8181. Il sert les fichiers statiques (HTML/JS) et héberge Socket.io. Le serveur :

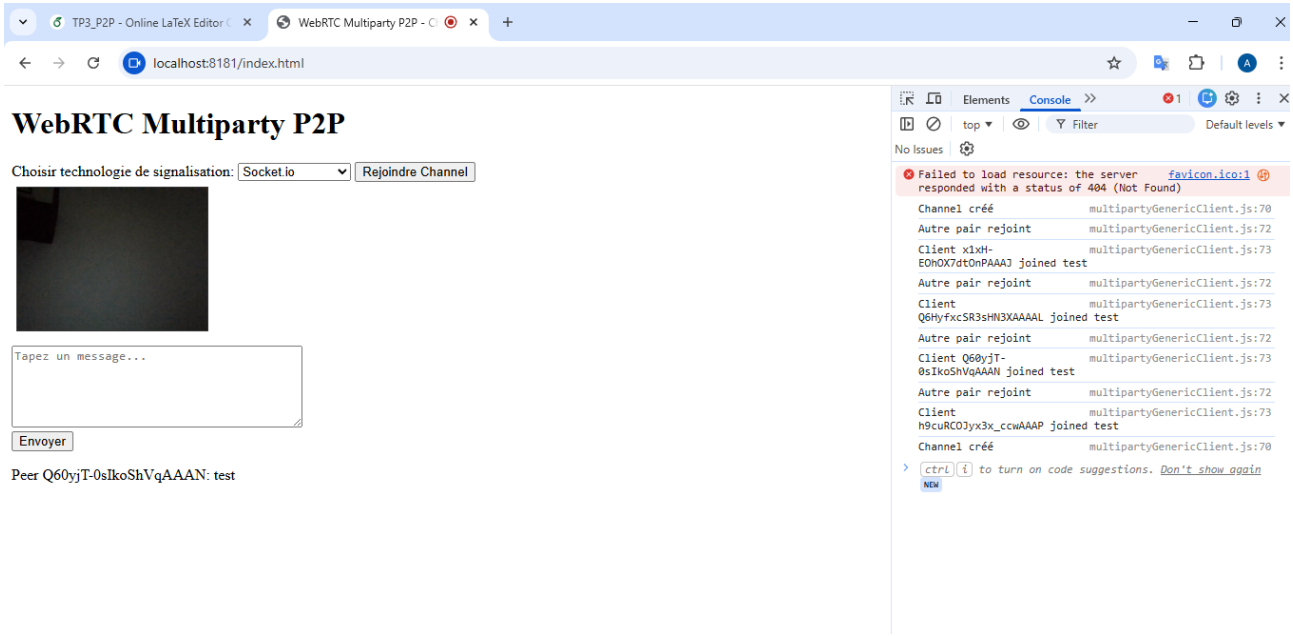
— gère la création et la jointure de canaux (rooms),
— limite le nombre de clients par canal,
— relaie les messages de signalisation (offres, réponses, candidates ICE, messages de chat).

Côté client, un script JavaScript :

— se connecte à `http://localhost:8181` via Socket.io,
— demande un nom de canal à l'utilisateur,
— envoie et reçoit les événements (`created`, `joined`, `message`, `Bye`, etc.).

## 3.3 Interface utilisateur

L'interface de test affiche un "scratchPad" où sont logués les événements de signalisation (création de canal, arrivée d'un pair, messages échangés).




figureCanal de signalisation Node.js + Socket.io

## 3.4 Résumé

Cette étape fournit l'infrastructure de signalisation indispensable pour que les navigateurs se découvrent et puissent ensuite établir la connexion P2P WebRTC.

# 4 Intégration Vidéo + Chat P2P

## 4.1 Objectif

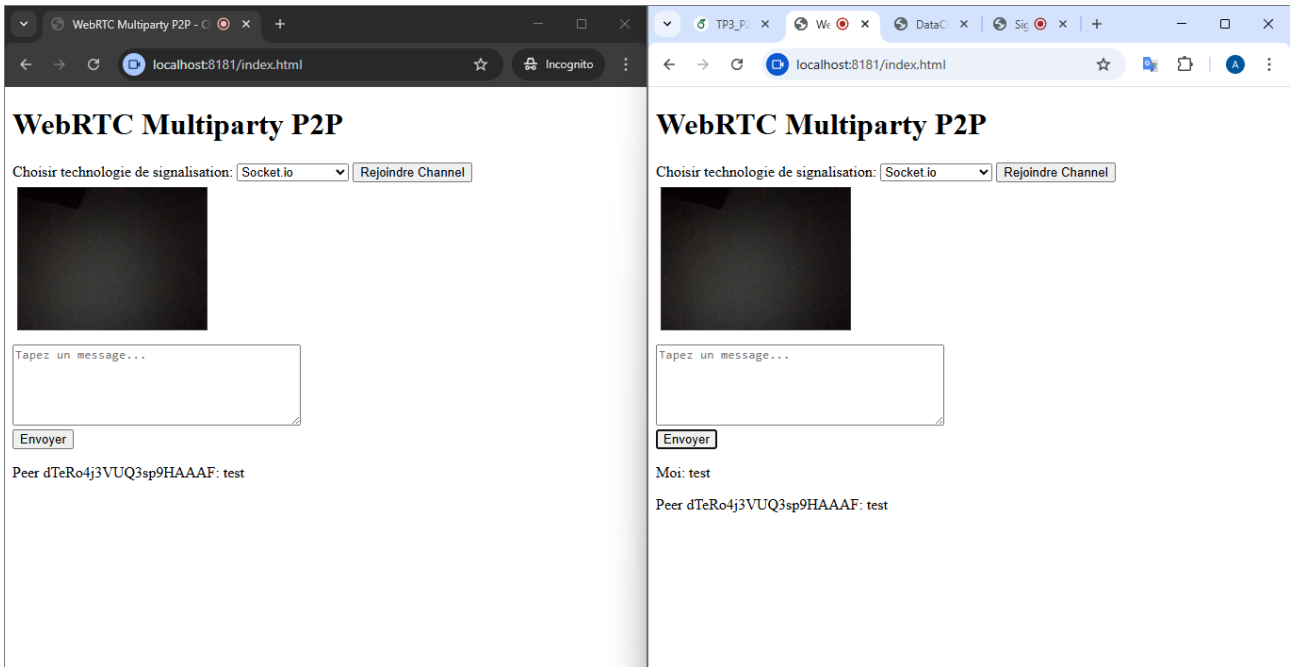
Combiner dans une même application :
— l'échange de flux audio/vidéo entre deux pairs,
— et un chat texte s'appuyant sur `RTCDataChannel`.

## 4.2 Description

L'application finale :
— affiche le flux local et le flux distant dans deux vidéos,
— établit la session P2P en utilisant la signalisation Node.js + Socket.io,
— ouvre un `DataChannel` pour le chat texte,
— permet de démarrer/arrêter la session et d'envoyer des messages.

## 4.3 Interface utilisateur

Deux navigateurs sont utilisés :
— un navigateur normal,
— un navigateur en mode privé/incognito.
Les deux se connectent au même canal et voient leur propre flux local ainsi que le flux du pair distant, avec le chat actif.




figureIntégration Vidéo + Chat P2P entre deux navigateurs

## 4.4 Résumé

Cette étape aboutit à une application P2P multimédia complète : la signalisation est assurée par le serveur Node.js, mais les flux audio/vidéo et les messages texte circulent directement entre navigateurs via WebRTC.

# 5 Extensions multipoints et comparaison des technologies de signalisation

## 5.1 Objectif

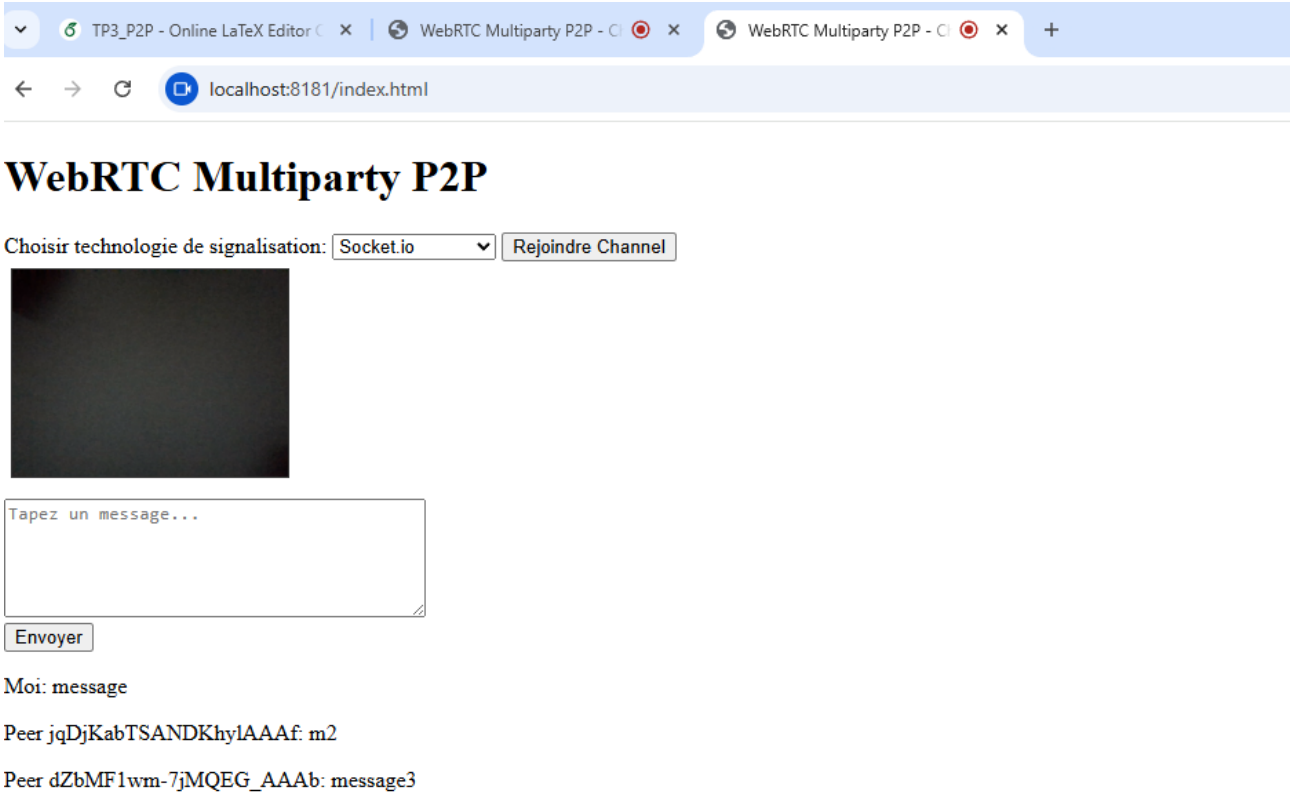
Étendre l'application pour supporter plusieurs participants sur un même canal, et tester différentes technologies de signalisation (Socket.io, WebSocket, XHR).

## 5.2 Description

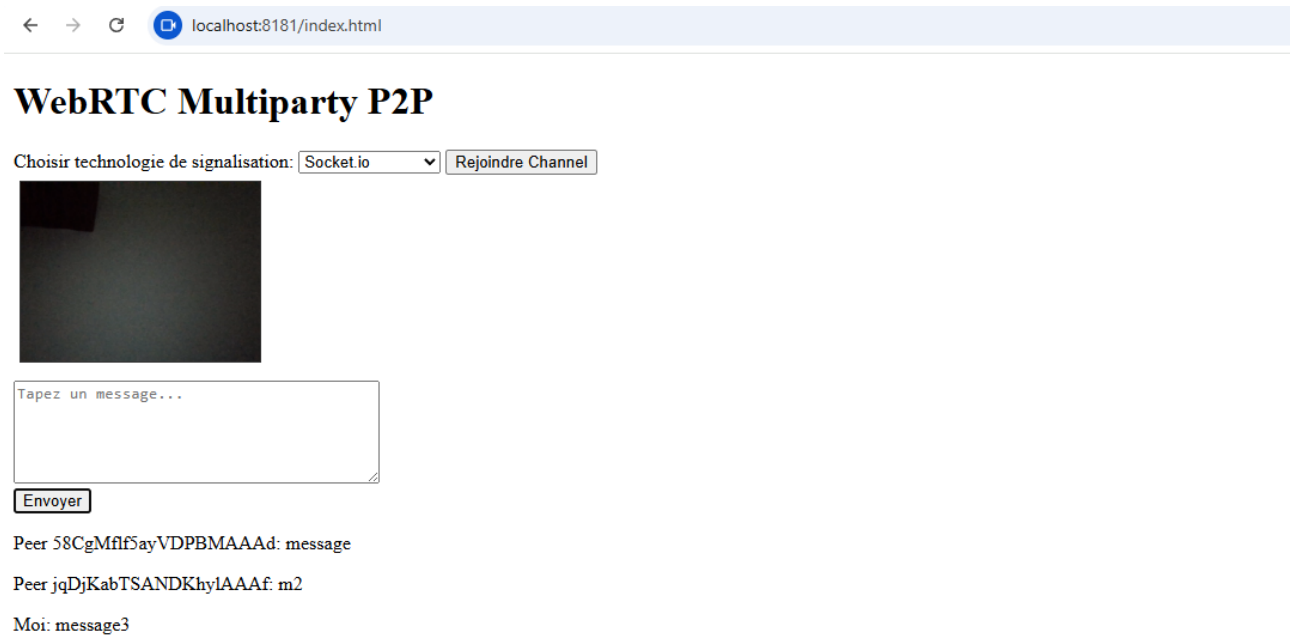
Dans la version multiparty :
— l'utilisateur choisit la technologie de signalisation (Socket.io, WebSocket, XHR),
— plusieurs participants peuvent rejoindre le même canal,
— chaque participant voit son flux local et échange des messages texte avec tous les autres,
— le serveur affiche l'état des connexions (clients Socket.io, connexions WebSocket).

## 5.3 Interface utilisateur

L'interface "WebRTC Multiparty P2P" comporte :
— une liste déroulante pour la technologie de signalisation,
— un bouton *Rejoindre Channel*,
— la vidéo locale,
— une zone de texte et un bouton *Envoyer* pour le chat,
— la liste des messages (pseudo local et identifiants des pairs).




figureClient multiparty (Socket.io) avec chat P2P




figureDeuxième client multiparty rejoignant le même canal

Une capture de la console serveur illustre le nombre de clients connectés sur les deux serveurs de signalisation :

```
>>
WebSocket WS running on ws://localhost:8182
HTTP+Socket.io running on http://localhost:8181
WebSocket connected
WebSocket connected
Socket.io connected: 8ZEBmqbHuKo0zrWzAAAB
Socket.io connected: Rq-PXdCN_EaV9IAyAAAD
Socket.io connected: dTeRo4j3VUQ3sp9HAAAF
WebSocket connected
WebSocket connected
Socket.io connected: M3gNc-bP6tiR35-0AAAH
```

figureServeurs Socket.io et WebSocket actifs avec plusieurs clients

## 5.4 Résumé

Les tests montrent que :

— Socket.io et WebSocket offrent une signalisation rapide et bidirectionnelle,

— XHR (long-polling) introduit des délais perceptibles et ne convient pas pour un usage temps réel intensif.

L'extension multipoints confirme que WebRTC peut être utilisé pour des scénarios de vidéo-conférence avec plusieurs participants.

# Conclusion générale

Ce TP a permis de construire progressivement une application WebRTC complète : capture locale, échange de données via `RTCDataChannel`, signalisation Node.js/Socket.io, intégration vidéo + chat, puis extension multiparty. Au-delà du fonctionnement de l'application, il met en évidence l'importance :

— d'une signalisation efficace (Socket.io/WebSocket),

— de la séparation claire entre signalisation et médias P2P,
— et des problématiques pratiques de déploiement (permissions, NAT, qualité réseau).

L'application obtenue constitue une base solide pour des projets plus avancés de communication multimédia en temps réel sur le web.