REPOBLIKAN'I MADAGASIKARA
FITIAVANA - TANINDRAZANA - FANDROSOANA

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE SUPÉRIEURE POLYTECHNIQUE D'ANTSIRANANA

Mention STIC

PROJET DE FIN DE SEMESTRE

Parcours Télécommunication et Réseau

Réalisation d'une application de communication en temps réel (WebRTC)

Par :

ANDRIANARISOA Daniel

Encadreurs :

Mme. RAOELIVOLOLONA Tefy

Mr. RAKOTOARIJAONA Raonirivo

⋆ Année Universitaire : 2019 - 2020 ⋆

REPOBLIKAN'I MADAGASIKARA
FITIAVANA - TANINDRAZANA - FANDROSOANA

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE

ÉCOLE SUPÉRIEURE POLYTECHNIQUE D'ANTSIRANANA

Mention STIC

PROJET DE FIN DE SEMESTRE

Parcours Télécommunication et Réseau

Réalisation d'une application de communication en temps réel (WebRTC)

Par :

ANDRIANARISOA Daniel

Encadreurs :

Mme. RAOELIVOLOLONA Tefy
Mr. RAKOTOARIJAONA Raonirivo

Membre de jury :

Pr./Dr./Mme/Mr. NOM Prénoms	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre
Pr./Dr./Mme/Mr. NOM Prénoms,	Grade	Titre

⋆ Année Universitaire : 2019 - 2020 ⋆

Remerciements

Je tiens à remercier spécialement Madame RAOELIVOLOLONA Tefy et Monsieur RAKOTOARIJAONA Rivo pour leur encadrement, orientation, conseils et disponibilité qu'ils nous ont témoignés pour me permettre de mener à bien ce projet.

Je souhaite aussi à exprimer mes remerciements aux membres du jury de soutenance, pour leur précieux jugement de ce projet. Qu'ils trouvent ici l'expression d'une reconnaissance profonde et sincère.

Sans oublier de remercier mes collègues étudiants de L'ESP Antsiranana et les gens de mon entourage, pour l'environnement serrein qui'ls ont apportés, propice à la completion de ce projet.

Un grand merci à toutes les personnes qui ont contribué au succès de ce projet et qui m'ont aidée lors de la rédaction de ce rapport.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes et symboles

Acronymes

API Application Programming Interface

CSS Cascading Style Sheet

HTML HyperText Markup Language

HTTP HyperText Transfert Protocol

ICE Interactive Connectivity Establishment

IETF Internet Engineering Task Force

IP Internet Protocol

NAT Network Address Translation

RTC Real Time Communication

SDP Session Description Protocol

SIP Session Initiation Protocol

STUN Session Traversal Utilities for NAT

TURN Traversal Using Relay NAT

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

W3C World Wide Web Consortium

WebRTC Web Real-Time Communications

Chapitre 1

Introduction au WebRTC

Sommaire

1.1 Introduction

La conception structurelle web classique est basée sur un modèle client-serveur, dans lequel les navigateurs envoient une demande de contenu HTTP au serveur web, qui répond en leur envoyant une réponse contenant les informations demandées. Le décodage de la réponse par les navigateurs est possible car elles contiennent les composants pour se conformer aux normes du W3C, ici en l'occurrence, de balises HTML, styles CSS et code Javascript[5, 12].

De même, en ce qui concerne le modèle de navigateurs, afin de supporter WebRTC, chaque navigateur doit ajouter un autre composant pour pouvoir la prendre en charge. La plupart des principaux navigateurs ont ajouté le composant WebRTC depuis 2014[5].

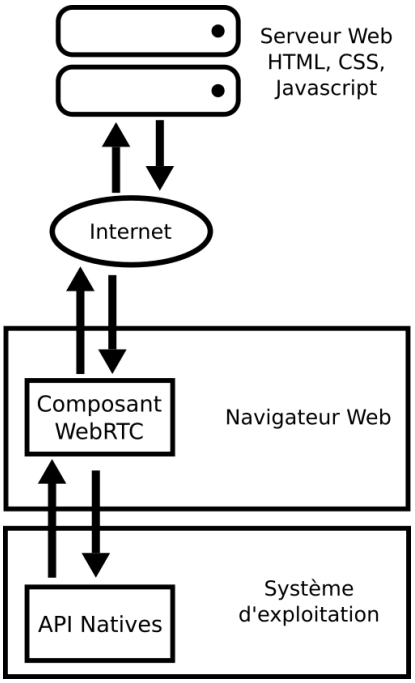

FIGURE 1.1 – composant WebRTC

À partir du diagramme 1.1, nous pouvons clairement voir le composant WebRTC intégré dans le navigateur. Il se charge d'appeler les API audio et vidéo natives du système d'exploitation.

Javascript joue un rôle important dans WebRTC, car la technologie s'exécute sur le navigateur. Javascript est le choix idéal en tant que langage de programmation pour assurer la médiation entre le WebRTC et l'application via un ensemble d'API exposées[5].

1.2 Normes WebRTC

Les normes WebRTC sont actuellement développées conjointement par le W3C et l'IETF. Le W3C travaille sur la définition des API nécessaires aux applications web Javascript pour interagir avec la fonction RTC du navigateur. L'IETF développe les protocoles utilisés par la fonction RTC du

navigateur pour communiquer avec un autre navigateur ou point de terminaison de communications Internet[6, p.11]. Le travail du W3C est centré sur le groupe de travail WebRTC et le travail de l'IETF est centré sur le groupe de travail RTCWeb (Real-Time Communications Web). Les deux groupes sont indépendants, mais se coordonnent étroitement[6, p.11].

1.3 Architecture WebRTC

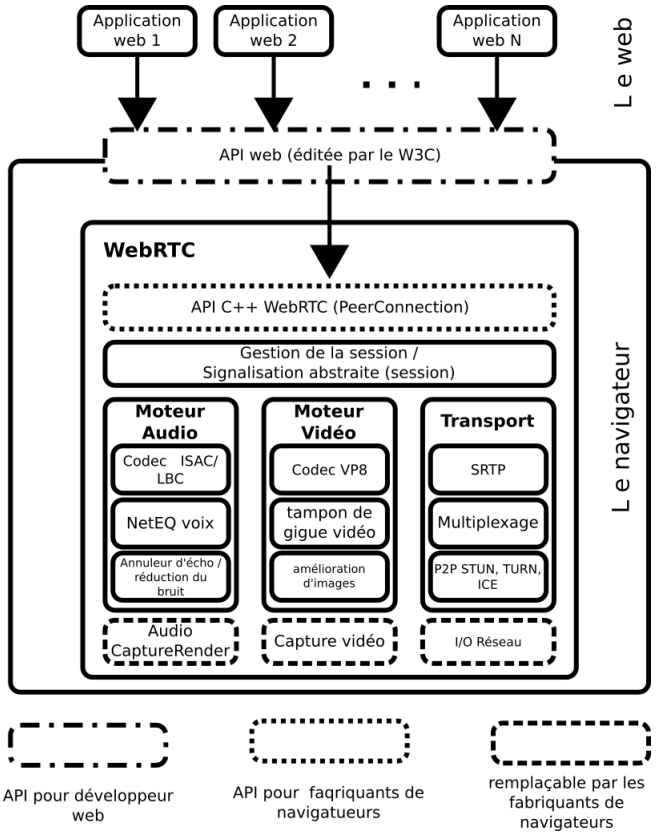

FIGURE 1.2 – architecture WebRTC

Le navigateur cache la grande partie de l'implémentation WebRTC, tout le travail de capture des médias, de la description de session et des transmissions réseau, tout est abstrait de la surface et n'expose que de simples API. L'application ne peut appeler que l'API web qui est normalisée par le W3C, à consommer par les développeurs web[5].

1.3.1 Le triangle WebRTC

Le scénario le plus répandu est probablement celui où les deux navigateurs exécutent la même application web WebRTC, téléchargée à partir de la même page web. Cela produit le «triangle» WebRTC illustré à la figure 1.3. Cet arrangement est appelé triangle en raison de la forme de la signalisation (côtés du triangle) et des flux médiatiques ou de données (base du triangle) entre les

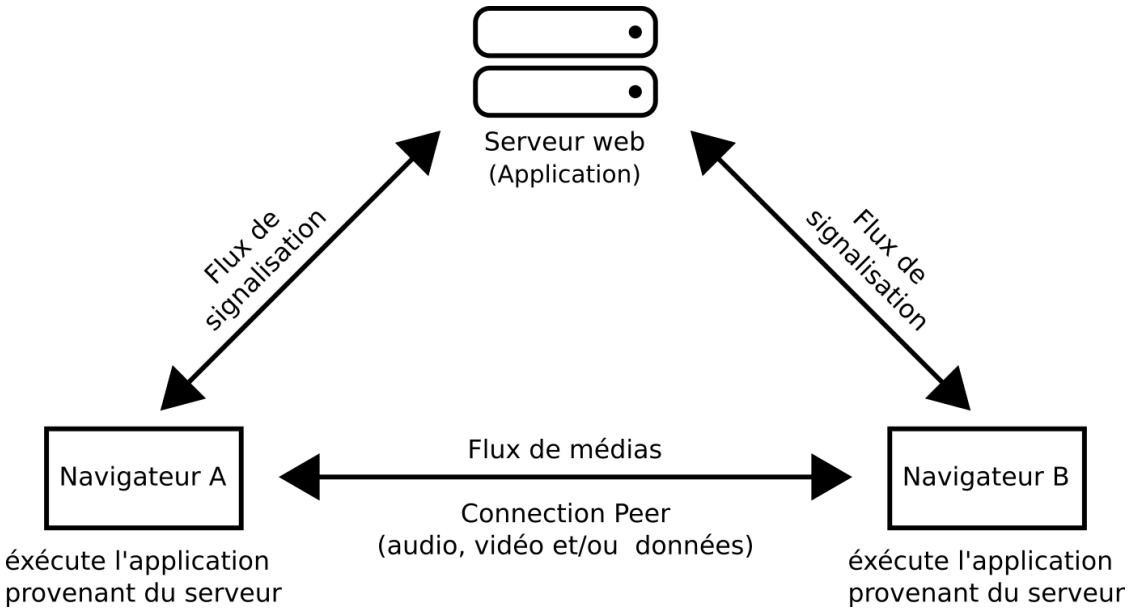

FIGURE 1.3 – triangle WebRTC

trois éléments. Une connexion peer établit le transport pour les médias vocaux et vidéo et les flux de canaux de données directement entre les navigateurs[6, p. 4-5].

1.4 Composants Importants du WebRTC

1.4.1 Flux de médias

Les appareils d'aujourd'hui sont dotés de plusieurs capacités matérielles multimédias. Dans les appareils mobiles, vous obtenez une caméra avant, une caméra arrière et un microphone audio. Sur les ordinateurs de bureau et les tablettes, vous obtenez le même ensemble de sources multimédias. Chacune de ces sources multimédia peut générer un flux multimédia[5].

Ce flux multimédia peut être capturé par l'API du navigateur WebRTC et fournira ce flux en tant que variable au langage Javascript pourque puissent les développeurs d'applications l'afficher sur une page comme n'importe quel flux vidéo normal[5].

1.4.2 Peer to peer

L'un des avantages distinctifs du WebRTC est le fait qu'il s'agit d'une communication peer to peer. Cela signifie évidemment que la vidéo passera directement de l'ordinateur d'un utilisateur à un autre[5].

1.4.3 Description de la session

En général, pour qu'une communication ait lieu entre deux périphériques quelconques, il existe un mécanisme de prise de contact qui doit être effectué en premier, pour établir la session entre les deux périphériques, puis le flux des messages ou des paquets peut démarrer[5].

C'est la même chose avec WebRTC. Il existe un protocole de déscription de session : SDP. Il s'agit d'un ensemble de messages qui doivent être échangés entre les deux navigateurs avant qu'ils ne commencent à envoyer des flux multimédias[5].

Pour échanger les données de description de session, nous avons besoin d'une machine intermédiaire. C'est là que le concept de signalisation entre en jeu[5].

1.4.4 Signalisation

Supposons que le navigateur A essaie de communiquer avec le navigateur B. Le navigateur A enverra ses données de description de session au serveur de signalisation S, et le même processus pour le navigateur B également[5].

Une fois ce processus terminé, chaque navigateur a maintenant à sa disposition tout le nécessaire pour établir une connexion et envoyer le flux multimédia à l'autre navigateur[5]. La figure 1.3 aidera à expliquer ce concept.

Rôle de la signalisation

La signalisation a un rôle important dans les communications en temps réel :

— Négociation des capacités et des paramètres des médias ntre les appareils dans le même appel

— Identification et authentification des participants à une session

— Contrôler la session multimédia, indiquer la progression, modifier et terminer la session

— Résolution de l'éblouissement, lorsque les deux côtés d'une session essaient d'établir ou de modifier une session en même temps

Pourquoi la signalisation n'a pas été standardisé ?

La signalisation n'est pas normalisée dans WebRTC pour permettre l'interopérabilité entre différents navigateurs. Son flux se situe entre les navigateurs web et le serveur web, et non entre deux navigateurs dans le même appel ou session. Dans ce cas, le serveur ou plutôt le developpeur sélectionne le protocole de signalisation et s'assure que les utilisateurs des applications web ou du support du site utilisent le même protocole[5].

Signalisation et négotiation des médias

La fonction la plus importante de la signalisation est l'échange d'informations contenues dans l'objet de protocole de description de session (SDP) entre les navigateurs. SDP contient toutes les informations nécessaires au fonctionnement de la communication, y compris les types de codecs multimédias (audio, vidéo et données) utilisés et des informations sur la bande passante de la connexion[5].

Un autre rôle de la signalisation est l'établissement de connectivité interactive (ICE). Il s'agit de l'adresse candidate représentant l'adresse IP et les ports UDP où des paquets multimédias potentiels pourraient être reçus par le navigateur. Mais il ne peut pas être démarré tant que l'adresse candidate n'a pas été échangée sur le canal de signalisation[5].

Identification et authentification

Le canal de signalisation peut également fournir l'identité des participants à un appel. Par exemple, dans une application web, l'application se chargera d'abord d'authentifier les utilisateurs, et lorsqu'un utilisateur doit appeler un autre utilisateur, l'application web présentera le nom d'utilisateur pour le canal de signalisation comme un identifiant authentifié[5].

WebRTC a d'autres moyens d'authentifier les utilisateurs sans s'appuyer sur des applications externes. En utilisant le canal multimédia, qui ne repose pas sur des applications externes de confiance, une identité et une authentification de l'appelant peuvent être fournies dans le chemin multimédia sans dépendre du tout du canal de signalisation[5].

Contrôle de la session de médias

La signalisation est nécessaire pour lancer l'appel ; cependant, il n'est pas nécessaire d'indiquer l'état ou de mettre fin à une session. Il y a une machine d'état ICE dans le navigateur qui peut fournir ces informations[5].

Résolution de l'éclat

La résolution de l'éblouissement intervient lorsque les deux côtés de la communication essaient de changer de session en même temps. C'est une condition de concurrence qui pourrait entraîner un état non déterminé pour la session. Le protocole de signalisation SIP intègre une résolution d'éblouissement[5].

1.5 Les API WebRTC

WebRTC a trois fonctionnalités principales :

1. Acquérir l'audio et la vidéo

2. Communication audio et vidéo
3. Communication de données arbitraires

Et il existe trois API Javascript majeures exposées pour appeler ces foncionnalités :

1. Flux multimédia (getUserMedia)
2. Connexion Peer RTC (RTCPeerConnection)
3. Canal de données RTC (RTCDataChannel)

1.5.1 Flux multimédia

Nous désignons les caméras et les microphones des ordinateurs des utilisateurs sous le nom de média local. Par conséquent, la première étape consiste à obtenir un flux multimédia local. Cela peut être fait de plusieurs manières. Voici un moyen simple de le faire, juste une méthode Javascript que nous pouvons appeler `getUserMedia()`[5]. Cette méthode récupère les médias des périphériques de la machine.

Pour les problèmes de confidentialité, le navigateur doit obtenir l'autorisation de l'utilisateur pour accéder à la caméra et au microphone de l'utilisateur. Le navigateur demandera à l'utilisateur d'autoriser l'accès. Une fois l'accès accordé, l'application recevra un flux de la caméra et du microphone[5].

1.5.2 Connexion peer RTC

Il est important de noter que cette connexion est une connexion peer. Cela signifie que cela peut être fait avec n'importe quel ordinateur ou appareil sur Internet. Et la connexion est directement établie entre les deux machines sans serveur impliqué[5] : les paquets vidéo et audio circuleront d'un navigateur à un autre.

Un fait très important qui mérite d'être mentionné est le nombre de connexions qui seront créées pour chaque utilisateur ajouté à une conférence téléphonique[5]. Disons, par exemple, un appel démarré avec deux utilisateurs, tels que l'utilisateur A et l'utilisateur B. Cela signifie qu'une connexion peer sera créée entre les navigateurs des deux utilisateurs, une connexion de A à B. Si un autre utilisateur rejoint l'appel, disons l'utilisateur C, cela modifiera les connexions pour chaque utilisateur de l'appel. Comme nous l'avons expliqué précédemment, il s'agira d'une connexion peer, ce qui signifie que chaque utilisateur doit être connecté à la machine de chaque utilisateur lors de l'appel. Chaque machine aura 2 connexions peer établies si l'appel a 3 utilisateurs.

$$Connexions\ par\ machine = nombre\ d'utilisateurs\ en\ appel - 1$$
$$Connexions\ par\ machine = 3 - 1$$

1.5.3 Canal de données RTC

Les canaux de données prennent en charge les connexions à haut volume et à faible latence ; un canal de données est un canal non multimédia qui prend uniquement en charge le transfert de données[5].

Il contourne les serveurs et fournit aux développeurs web des canaux configurables pour transférer des données. Les canaux de données dans WebRTC sont construits sur des WebSockets. De cette façon, il obtient une fonctionnalité en temps réel. Des méthodes simples telles que l'envoi sur le gestionnaire de messages peuvent être définies pour obtenir les données. Le canal de données utilise également la même connexion peer que le média, ce qui est une bonne chose, car cela signifie qu'un seul processus de négociation offre/réponse est nécessaire[5].

Les canaux de données prennent également en charge les deux modes de livraison : livraison garantie (fiable) et rapide (non fiable). Le premier peut être utilisé pour des événements critiques et le second peut être utilisé pour les mises à jour de la position d'un jeu en ligne[5].

Chapitre 2

Configuration des Interfaces WebRTC

Sommaire

2.1 Introduction

WebRTC offre des opportunités sans précédent aux développeurs qui souhaitent intégrer des communications en temps réel dans leurs applications.

Les API WebRTC getUserMedia, RTCPeerConnection et RTCDataChannel jouent chacune leur propre rôle dans la capture, la transmission et la diffusion de données en temps réel (depuis la webcam et le microphone d'un ordinateur) vers un autre navigateur, sans qu'un utilisateur ait à télécharger des plug-ins ou des modules complémentaires[11].

2.2 Signalisation

La spécification WebRTC inclut des API pour communiquer avec un serveur ICE , mais le composant de signalisation n'en fait pas partie(1.4.4). La signalisation est nécessaire pour que deux pairs partagent la façon dont ils doivent se connecter. Habituellement, cela est résolu via une API Web standard basée sur HTTP (c'est-à-dire un service REST ou un autre mécanisme RPC) où les applications Web peuvent relayer les informations nécessaires avant que la connexion entre homologues ne soit lancée[15].

L'extrait de code provenant de [15] suivant, montre comment un service de signalisation fictif peut être utilisé pour envoyer et recevoir des messages de manière asynchrone.

```
// Configurer un canal de communication asynchrone qui
// sera utilisé lors de la configuration de la connexion peer
const signalingChannel = new SignalingChannel(remoteClientId);
signalingChannel.addEventListener('message', message => {
  // Nouveau message du client distant reçu
});

// Envoyer un message asynchrone au client distant
signalingChannel.send('Hello!');
```

2.3 RTCPeerConnection

L'API RTCPeerConnection est chargé de connecter deux (02) navigateurs ensemble afin qu'ils puissent partager des médias en temps réel[11]. Elle est au cœur de la connexion peer-to-peer entre chacun des navigateurs.

2.3.1 RTCConfiguration

Chaque connexion peer est gérée par un objet RTCPeerConnection. Le constructeur de cette classe prend un seul objet RTCConfiguration comme paramètre. Cet objet définit la configuration de la connexion peer et doit contenir des informations sur les serveurs ICE à utiliser [1][15].

L'établissement de connectivité interactive (ICE) est une technique utilisée dans les réseaux informatiques pour trouver des moyens pour que deux ordinateurs communiquent aussi directement que possible dans les réseaux peer-to-peer. Il est le plus souvent utilisé pour les médias interactifs tels que la voix sur IP (VoIP), les communications peer-to-peer, la vidéo et la messagerie instantanée. Dans de telles applications, mieux vaut éviter de communiquer via un serveur central (ce qui ralentirait la communication et serait coûteux), mais la communication directe entre les applications clientes sur Internet est très délicate en raison des traducteurs d'adresses réseau (NAT), des pare-feu et d'autres barrières du réseau[16].

2.3.2 Initialisation

Une fois la configuration éffectuée il suffit d'instancier l'interface de l'API RTCPeerConnection afin d'initialiser une connection et ce sans oublier de fournir l'argument configuration dans la signature du constructeur. Le bout de code suivant, tiré de [15] montre comment éffectuer celà.

```
const configuration: RTCConfiguration = {
  iceServers: [{'urls': 'stun:stun.l.google.com:19302'}],
};
const peerConnection = new RTCPeerConnection(configuration);
```

2.3.3 RTCSessionDescription

RTCPeerConnection coordonne l'échange de métadonnées cruciales entre deux navigateurs. Ces données définissent le numéro IP et l'adresse de port publiquement identifiables d'un navigateur afin que les médias en temps réel puissent être échangés.

Pour que deux points de terminaison WebRTC commencent à se parler, trois types d'informations doivent être relayées :

1. Les informations de contrôle de session déterminent quand initialiser, fermer et modifier les sessions de communication.
2. Les données réseau relayent l'adresse IP et le numéro de port de chaque point de terminaison afin que les appelants puissent trouver les personnes appelées.

1. Le dictionnaire complet des membres de RTCConfiguration peut être accédée à l'addresse https ://developer.mozilla.org/en-US/docs/Web/API/RTCConfiguration

3. Les données média concernent les codecs et les types de média que les appelants ont en commun.

Ces informations seront encapsulées dans un objet RTCSessionDescription.

L'interface RTCSessionDescription décrit une extrémité d'une connexion (ou connexion potentielle) et comment elle est configurée. Chaque objet RTCSessionDescription consiste en un type de description indiquant quelle partie du processus de négociation il décrit (offre ou réponse) et du descripteur SDP de la session[10].

Le processus de négociation d'une connexion entre deux peers implique l'échange d'objets RTCSessionDescription dans les deux sens, par le biais du canal de signalisation(1.4.4), chaque description suggérant une combinaison d'options de configuration de connexion que l'expéditeur de la description prend en charge. Une fois que les deux pairs se sont mis d'accord sur une configuration pour la connexion, la négociation est terminée[10].

2.3.3.1 Initialisation, transmission et négotiation

Voici comment une description de session sera initiée en tant qu'une offre puis envoyée de la part du client local vers un client distant, sans oublier comment ce même client négociera une potentielle réponse de son correspondant[15].

```
// Écouter la réception d'un message sur le canal de signalisation
signalingChannel.addEventListener('message', async message => {
  if (message.answer) { // Si le message comporte une réponse
    // assigner la réponse en tant que description de session distante
    const remoteDesc = new RTCSessionDescription(message.answer);
    await peerConnection.setRemoteDescription(remoteDesc);
  }
});

// Créer une description de session associée à l'offre
const offer = await peerConnection.createOffer();
await peerConnection.setLocalDescription(offer);
// Puis envoyer cette offre au client distant
signalingChannel.send({'offer': offer});
```

Et ci-après, comment du côté d'un client distant auquel sera demandé une communication, la négiociation d'une offre qui lui aboutit sera faite[15].

```
// Écouter la réception d'un message sur le canal de signalisation
signalingChannel.addEventListener('message', async message => {
```

```
  if (message.offer) { // Si le message contient une offre
  // assigner l'offre en tant que description de session distante
    const remoteDesc = new RTCSessionDescription(message.offer);
    peerConnection.setRemoteDescription(remoteDesc);

    // Puis créer une description de session associée à la réponse
    const answer = await peerConnection.createAnswer();
    await peerConnection.setLocalDescription(answer);

  // Efin renvoyer cette réponse au client ayant envoyé l'offre
    signalingChannel.send({'answer': answer});
  }
});
```

2.3.4 Candidats ICE

Aavant que deux peers puissent communiquer en utilisant WebRTC, ils doivent echanger des informations de connectivité. Étant donné que les conditions du réseau peuvent varier en fonction d'un grand nombre de facteurs, un service externe est généralement utilisé pour découvrir les candidats possibles pour se connecter à un peer. Ce service s'appelle ICE et utilise un serveur STUN ou TURN. STUN signifie «Session Traversal Utitlities for NAT», et est généralement utilisé indirectement dans la plupart des applications WebRTC[15].

TURN (Traversal Using Relay NAT) est la solution la plus avancée qui intègre les protocoles STUN et la plupart des services commerciaux basés sur WebRTC untilsent un serveur TURN pour établir des liens entre peers. L'API WebRTC prend en charge à la fois STUN et TURN directement et il est regroupé sous le terme complet «Internet Connectivity Establishment»[15]. Lors de la création d'une connexion WebRTC, nous fournissons généralement un ou plusieurs serveurs ICE dans la configuration de RTCPeerConnection(2.3.1).

Provisionnement incrémentiel (ruisselement ICE)

Une fois qu'un objet RTCPeerConnection est créé, le framework sous-jacent utilise les serveurs ICE pour rassembler les candidats à L'établissement de la connectivité (candidats ICE). L'évènement `icegatheringstatechange` de l'API RTCPeerConnection signale à quel état se trouve la collecte ICE (`new`, `gathering` ou alors `complete`)[15].

Bien qu'il soit possible pour un peer d'attendre la fin de la collecte ICE, il est généralement beaucoup plus éfficace d'utiliser une technique de «ruissellement» et de transmettre chaque can-

didats ICE au peer distant au fur et à mesure qu'il en est découvert. Celà résulte par une considérable réduction du temps d'installation de la connectivité des peers et permet d'entamer une conversation vidéo avec moins de retard[15].

Pour rassembler des candidats ICE, il suffit d'ajouter un écouteur pour l'évènement `icecandidate` . L'évenement RTCPeerConnectionICEevent émis sur cet écouteur contiendra la propriété candate qui représente un nouveau candidat qui devrait être envoyé au peer distant(1.4.4).

```
// Écouter les candidats ICE locaux sur l'objet RTCPeerConnection local
peerConnection.addEventListener('icecandidate', event => {
  if (event.candidate) {
    signalingChannel.send({'new-ice-candidate': event.candidate});
  }
});
// Écouter les candidats ICE distants et les ajouter
// à la connexion RTCPeerConnection locale
signalingChannel.addEventListener('message', async message => {
  if (message.iceCandidate) {
    try {
      await peerConnection.addIceCandidate(message.iceCandidate);
    } catch (e) {
      console.error('Error adding received ice candidate', e);
    }
  }
});
```

Une fois que les candidats ICE sont reçus, nous devons nous attendre à ce que l'état de notre connexion peer passe éventuellement à un état connecté. Pour détecter cela, nous ajoutons un écouteur à notre RTCPeerConnection où nous écoutons les événements `connectionstatechange` [15].

2.4 getUserMedia

L'API getUserMedia permet d'accéder aux flux multimédias (vidéo, audio ou les deux) à partir des périphériques locaux. À elle seule, cette API est capable d'acquérir de l'audio et de la vidéo, sans envoyer les données ni les stocker dans un fichier. Pour avoir un chat fonctionnel, nous devons envoyer ces données en utilisant l'API RTCPeerConnection.

À elle seule, cette API est uniquement capable d'acquérir de l'audio et de la vidéo, sans envoyer les données ni les stocker dans un fichier. Pour avoir un chat fonctionnel, nous devons envoyer ces

données en utilisant l'API RTCPeerConnection[14].

2.4.1 Mediadevices

L'interface Navigator représente l'état et l'identité de l'agent utilisateur. Il permet aux scripts de l'interroger et de s'enregistrer pour effectuer certaines activités. Un objet Navigator peut être récupéré à l'aide de la propriété readonly `window.navigator`[9].

L'interface Navigator dans le navigateur fournit des fonctions et des propriétés pour accéder à l'état et aux fonctionnalités du navigateur grâce auxquelles nous pouvons obtenir l'état du navigateur ou accéder à un large éventail de fonctionnalités. Par exemple, en utilisant `navigator.online`, nous pouvons obtenir l'état de connexion du navigateur avec Internet[13].

L'interface MediaDevices comprise dans Navigator, permet d'accéder aux périphériques d'entrée multimédia connectés tels que les caméras et les microphones, ainsi que le partage d'écran. En substance, il vous permet d'accéder à n'importe quelle source matérielle de données multimédias[9].

Les fonctions de l'objet mediaDevices fournissent des fonctionnalités telles que le partage d'écran, l'obtention de flux depuis la caméra et les microphones. La fonction getUserMedia est celle dont nous avons besoin pour récupérer les flux de la caméra et des microphones[13].

2.4.2 Capture des médias et contraintes

La méthode `getUsermedia()` à besoin qu'on lui passe un objet MediaStreamConstraints en tant qu'argument pour qu'uelle fonctionne. Cet argument, active le type de communication voulu (vidéo, audio ou les deux) et contient les contraintes ainsi que les paramètres du flux de média qui sera généré par la suite. La suppression de bruit, l' annulation de l'écho, le ratio de la video, le volume du son, la luminosité de l'image, etc sont quelques éxemples de ces paramètres en question [2].

```
// Caméra avec une résolution aussi proche que possible de 640x480
{
  'video': {
    'width': 640,
    'height': 480
    }
}
```

2. Une liste complète des contraintes et paramètres de flux de média peut être consultée à l'addresse url : https ://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints

Le dictionnaire MediaStreamConstraints est utilisé lors de l'appel à getUsermedia pour spécifier les types de pistes à inclure dans le MediaStream renvoyé et, éventuellement, pour établir des contraintes pour les paramètres de ces pistes[8].

L'implémentation minimale requise par l'interface MediaStreamConstraints est la suivante : `{ video: true, audio: true }`. Cette dernière, indique que nous voulons que le flux généré comporte une voix et une vidéo en utilisant les configurations de contraintes et paramètres par défaut.

Capturer les médias à partir des périphéques médias, dans le navigateur, en utilisant ces configurations de contraintes et de paramètres consiste à écrire les lignes de code suivantes.

```
const constraints: MediaStreamConstraint = { audio: true, video: true };
navigator.mediaDevices.getUserMedia(constraints);
```

2.4.3 Échange des flux miltimédia

Une fois qu'un objet RTCPeerConnection est connecté à un peer distant, il est possible de diffuser de l'audio et de la vidéo entre eux. C'est le point où nous connectons le flux que nous recevons de `getUserMedia()` à RTCPeerConnection.

Ajout des pistes locales

Un flux multimédia se compose d'au moins une piste multimédia, et celles-ci sont ajoutées individuellement à la RTCPeerConnection lorsque nous voulons transmettre le média à l'homologue distant[15].

```
const localStream = await navigator.mediaDevices.getUserMedia({video: true,
   audio: true});
const peerConnection = new RTCPeerConnection(iceConfig);
localStream.getTracks().forEach(track => {
  peerConnection.addTrack(track, localStream);
});
```

Les pistes peuvent être ajoutées à un RTCPeerConnection avant qu'il ne soit connecté à un pair distant, il est donc logique d'effectuer cette configuration le plus tôt possible au lieu d'attendre que la connexion soit terminée[15].

Ajout des pistes distantes

Pour recevoir les pistes distantes qui ont été ajoutées par l'autre pair, nous enregistrons un écouteur sur l'objet RTCPeerConnection local écoutant l'événement `track`. Puisque la lecture est effectuée sur un objet MediaStream, nous créons d'abord une instance vide que nous remplissons ensuite avec les pistes du pair distant au fur et à mesure que nous les recevons[15].

```
const remoteStream = new MediaStream();
const remoteVideo = document.querySelector('#remoteVideo');
remoteVideo.srcObject = remoteStream;
peerConnection.addEventListener('track', async (event) => {
  remoteStream.addTrack(event.track, remoteStream);
});
```

Chapitre 3

Relevé des protocoles mis en jeux dans WebRTC

Sommaire

3.1 Introduction

Il existe un certain nombre de protocoles liés à WebRTC. Les plus importantes sont répertoriées dans les figure et tableau suivants.

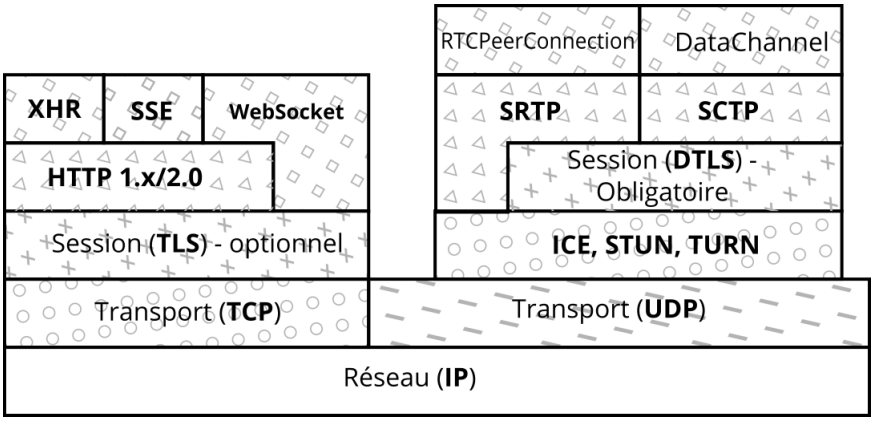

FIGURE 3.1 – Pile de protocoles WebRTC

Sur l'aile gauche de la figure précédente, s'illustrent les protocules uitlisés pour la signalisation tandisque dans l'aile droite, sont montrés ceux utilisés par RTCWeb elle même.

Protocole	Usage	Norme IETF	Couche
HTTP	Hyper-Text Transport protocol	RFC7230 - RFC7235	Application
WebSocket	Socket entre navigateur web et serveur	RFC6455	
SRTP	Secure Real-Time Transport Protocol	RFC3711	
SDP	Session Description Protocol	RFC8866	
STUN	Session Traversal Utilities for NAT	RFC8489	
TURN	Traversal Using Relays around NAT	RFC8656	
ICE	Interactive Connectivity establishment	RFC8445, RFC8839	
TLS	Transport Layesr Security	RFC5246	Transport
TCP	Transmission Control Protocol	RFC793	
DTLS	Datagram Transport Layer Security	RFC6347	
UDP	User Datagram Protocol	RFC768	
SCTP	Streal Control Transport Protocol	RFC4960	
IP	Internet Protocol	RFC791, RFC8200	Réseau

TABLEAU 3.1 – Les normes IETF et couche correspondantes des protocoles webRTC

Un développeur Web en général n'interagira jamais directement avec les protocoles, car les paramètres et configurations par défaut des protocoles répondront généralement à leurs besoins. Cependant, dans certains cas, en particulier lorsqu'un client WebRTC communique avec un client non WebRTC, une configuration et une connaissance des protocoles utilisés par WebRTC sont nécessaires. De plus, s'il est nécessaire d'ajuster l'objet RTCSessionDescription dans l'API WEBRTC, comme cela peut se produire s'il y a des problèmes d'interopérabilité SDP entre les agents utilisateurs, l'auteur de l'application devra avoir une meilleure compréhension du fonctionnement de

la négociation. Dans tous les cas, une compréhension de base des protocoles utilisés par WebRTC est utile pour le développeur. Un développeur de téléphonie qui souhaite utiliser WebRTC, en revanche, devra avoir une compréhension détaillée des protocoles utilisés[6, p. 125].

3.2 Protocoles de la couche application

3.2.1 HTTP

Bien sûr, WebRTC utilise HTTP, Hyper-Text Transport Protocol. C'est le protocole du World Wide Web et il est utilisé entre un navigateur Web et un serveur Web. WebRTC utilise HTTP de la même manière que n'importe quelle application Web. En tant que tel, aucune connaissance spécifique de HTTP n'est nécessaire. La version actuelle de HTTP est 1.1. Des travaux sont en cours à l'IETF pour définir la prochaine version de HTTP, connue sous le nom de 2.0. Ce protocole augmentera probablement la vitesse et l'efficacité des téléchargements et des applications Web. WebRTC pourra utiliser cette version et toute autre version future de HTTP[6, p. 125].

3.2.2 WebSocket

Le protocole WebSocket permet à un navigateur d'ouvrir des connexions TCP bidirectionnelles supplémentaires à un serveur Web. L'ouverture de la connexion est signalée via HTTP et possède des propriétés de sécurité similaires à celles de la session Web HTTP, et peut réutiliser l'infrastructure HTTP existante. Cela évite les interrogations HTTP et l'ouverture de plusieurs connexions HTTP entre le navigateur et le serveur Web. Le navigateur indique l'application utilisant le WebSocket dans l'ouverture. C'est ce qu'on appelle le sous-protocole WebSocket. Le sous-protocole SIP WebSocket ou le sous-protocole XMPP WebSocket sont des exemples [6, p. 125 - 126].

3.2.3 RTP et SRTP

Le protocole le plus important utilisé par WebRTC est le Real-time Transport Protocol, ou RTP. WebRTC utilise uniquement le profil sécurisé de RTP ou Secure RTP, SRTP. SRTP est le protocole utilisé pour transporter et acheminer les paquets multimédias audio et vidéo entre les clients WebRTC. Les paquets multimédias contiennent les échantillons audio numérisés ou les images vidéo numérisées générées par un microphone ou une caméra ou une application, et sont rendus à l'aide d'un haut-parleur ou d'un écran. Une configuration réussie d'une connexion par les peers, ainsi qu'un échange offre/réponse complet, entraînera l'établissement d'une connexion SRTP entre les navigateurs ou un navigateur et un serveur, et un échange d'informations sur les médias. SRTP fournit des informations essentielles pour transporter et restituer avec succès les informations du média : le codec (codeur/décodeur utilisé pour échantillonner et compresser l'audio ou la vidéo,

la source du média (la source de synchronisation ou SSRC), un horodatage (pour une lecture correctement chronométrée), le numéro de séquence (pour détecter les paquets perdus) et d'autres informations nécessaires à la lecture. Pour les données non audio ou vidéo, SRTP n'est pas utilisé. Au lieu de cela, un appel à l'API RTCDataChannel entraînera l'ouverture d'un canal de données entre les navigateurs permettant l'échange de données arbitrairement formatées[6, p. 216].

3.2.4 SDP

Les descriptions de session WebRTC sont décrites à l'aide de la Session Description Protocol (SDP). Une description de session SDP (codé comme un objet RTCSessionDescription) est utilisé pour décrire les caractéristiques de support de la connexion peer. Il existe une longue et compliquée liste d'informations qui doivent être échangées entre les deux extrémités de la session SRTP pour qu'elles puissent communiquer. Les appels d'API à RTCPeerConnection entraîneront une description de session SDP, un ensemble de données formatées d'une manière particulière, générées par le navigateur et accessibles à l'aide de JavaScript par l'application Web. Une application qui souhaite avoir un contrôle étroit sur le média peut apporter des modifications à la description de la session avant de la partager avec l'autre navigateur. Lorsque des modifications sont apportées à une connexion peer, cela entraînera des modifications de la description de session que les deux peers échangeront. C'est ce qu'on appelle un échange offre/réponse(2.3.3). Tout développeur souhaitant avoir un contrôle fin sur les sessions multimédias doit comprendre SDP[6, p. 127].

SRTP et SDP sont des protocoles normalisés par l'IETF et largement utilisé par les appareils et services de communication sur Internet, tels que les téléphones, les clients et les passerelles Voix sur IP (VoIP), ainsi que les appareils de vidéoconférence et de collaboration. En conséquence, la communication entre l'un de ces appareils ou clients et un client WebRTC est possible. Cependant, peu d'appareils ou de clients VoIP ou vidéo prennent aujourd'hui en charge l'ensemble complet des capacités et des protocoles de WebRTC. Ces appareils devront être mis à niveau pour prendre en charge ces nouveaux protocoles, ou une fonction de passerelle utilisée entre le client WebRTC et le client VoIP ou vidéo pour effectuer la conversion[6, p. 127].

3.2.5 STUN

STUN est un outil utilisé par d'autres protocoles, tels que Interactive Connectivity Establishment (ICE), le Session Initiation Protocol (SIP) et WebRTC permettant de détecter et de traverser les traducteurs d'adresses réseau situés sur le chemin entre deux points de terminaison de communication dans les applications de voix, vidéo, messagerie et autres communications interactives en temps réel. Ill est implémenté en tant que protocole client-serveur léger, ne nécessitant que des composants de requête et de réponse simples avec un serveur tiers situé sur le réseau commun et facilement accessible, généralement Internet. Le côté client est mis en œuvre dans l'application

de communication de l'utilisateur, telle qu'un téléphone VoIP ou un client de messagerie instantanée[17].

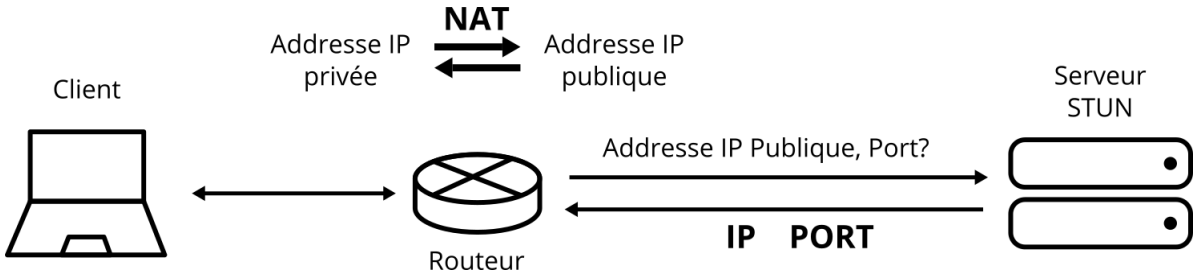

FIGURE 3.2 – Requête à un serveur STUN

Le protocole de base fonctionne essentiellement comme suit : Le client, opérant généralement à l'intérieur d'un réseau privé , envoie une demande de liaison à un serveur STUN sur l'Internet public. Le serveur STUN répond avec une réponse de succès qui contient l' adresse IP et le numéro port de du client, comme observé du point de vue du serveur. Le résultat est masqué par un mappage «ou exclusif»(XOR) pour éviter la traduction du contenu du paquet par des passerelles de couche d'application (ALG) qui effectuent une inspection approfondie des paquets dans le but d'exécuter d'autres méthodes de traversée NAT[17].

Les messages STUN sont envoyés dans des paquets UDP. Comme UDP ne fournit pas un transport fiable , la fiabilité est obtenue par des retransmissions contrôlées par l'application des demandes STUN. Les serveurs STUN n'implémentent aucun mécanisme de fiabilité pour leurs réponses. Lorsque la fiabilité est obligatoire, le protocole TCP peut être utilisé, mais induit une surcharge réseau supplémentaire. Dans les applications sensibles à la sécurité, STUN peut être transporté et crypté par TLS. Le numéro de port d'écoute standard pour un serveur STUN est 3478 pour UDP et TCP et 5349 pour TLS[17].

Lorsqu'un client a évalué son adresse externe, il peut l'utiliser comme candidat pour communiquer avec des pairs en partageant l'adresse NAT externe plutôt que l'adresse privée, qui n'est pas accessible depuis les pairs sur le réseau public. Si les deux pairs en communication sont situés dans des réseaux privés différents, chacun derrière un NAT, les pairs doivent se coordonner pour déterminer le meilleur chemin de communication entre eux. Certains comportements NAT peuvent restreindre la connectivité des homologues même lorsque la liaison publique est connue. Le protocole d' établissement de connectivité interactive (ICE) fournit un mécanisme structuré pour déterminer le chemin de communication optimal entre deux homologues. Les extensions SIP (Session Initiation Protocol) sont définies pour permettre l'utilisation d'ICE lors de l'établissement d'un appel entre deux hôtes[17].

3.2.6 TURN

Traversal Using Relays around NAT, TURN, est une extension du protocole STUN qui fournit un relais média pour les situations où le «hole punching » (trou de perforation en anglais) ICE échoue. Dans WebRTC, l'agent utilisateur du navigateur inclura un client TURN, et un serveur Web, un fournisseur de services ou une entreprise fournira un serveur TURN. Le navigateur demande une adresse IP publique et un numéro de port comme adresse de relais de transport au serveur TURN. Cette adresse est ensuite incluse en tant qu'adresse candidate dans la "perforation" ICE. Le numéro de port pour TURN peut être déterminé à l'aide d'une recherche DNS SRV ; le port UDP par défaut pour TURN est 3478[6, p. 132].

TURN peut être utilisé pour établir des adresses de transport relayées qui utilisent le transport UDP, TCP ou TLS. Cependant, la communication entre le serveur TURN et le client TURN (via le NAT) est toujours UDP[6, p. 132].

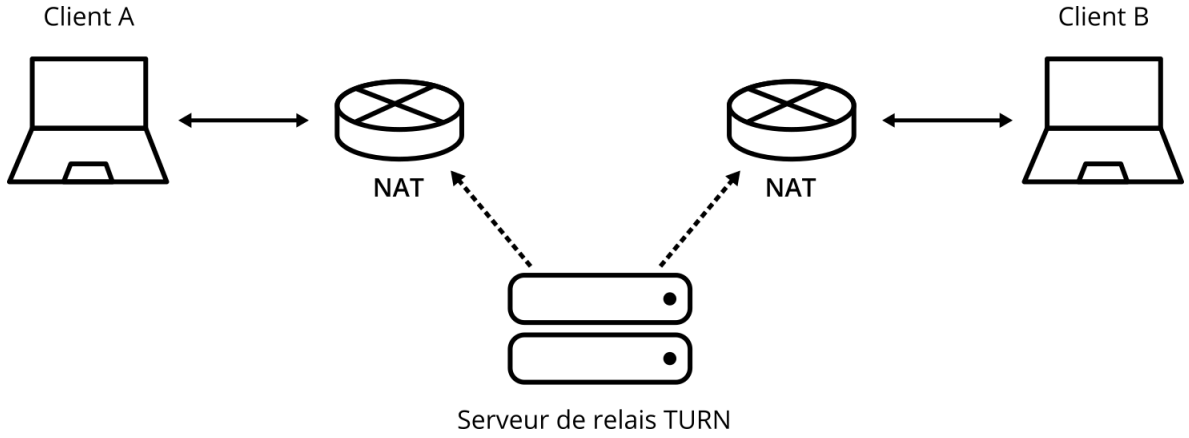

FIGURE 3.3 – Serveur de relai TURN

Le processus commence lorsqu'un ordinateur client souhaite contacter un ordinateur peer pour une transaction de données, mais ne peut pas le faire car le client et son homologue se trouvent derrière leur NAT respectifs. Si STUN n'est pas une option car l'un des NAT est un NAT symétrique (un type de NAT connu pour être non compatible STUN), TURN doit être utilisé[18].

Tout d'abord, le client contacte un serveur TURN avec une requête "Allocate". La demande d'allocation demande au serveur TURN d'allouer une partie de ses ressources au client afin qu'il puisse contacter un homologue. Si l'allocation est possible, le serveur alloue une adresse au client à utiliser comme relais et envoie au client une réponse "Allocation réussie", qui contient une "adresse de transport relayée allouée" située au niveau du serveur TURN[18].

Deuxièmement, le client envoie une demande CreatePermissions au serveur TURN pour créer un système de vérification des autorisations pour les communications homologue-serveur. En d'autres termes, lorsqu'un pair est finalement contacté et renvoie les informations au serveur TURN pour qu'elles soient relayées au client, le serveur TURN utilise les autorisations pour vérifier que la communication du serveur pair à TURN est valide[18].

Une fois les autorisations créées, le client a deux choix pour envoyer les données réelles, (1) il peut utiliser le mécanisme d'envoi, ou (2) il peut réserver un canal à l'aide de la requête ChannelBind. Le mécanisme d'envoi est plus simple, mais contient un en-tête plus grand, 36 octets, qui peut augmenter considérablement la bande passante dans une conversation relayée TURN. En revanche, la méthode ChannelBind est plus légère : l'en-tête ne fait que 4 octets, mais elle nécessite la réservation d'un canal qui doit être périodiquement rafraîchi, entre autres considérations[18].

En utilisant l'une ou l'autre méthode, Send ou channel binding, le serveur TURN reçoit les données du client et les relaie à l'homologue à l'aide de datagrammes UDP, qui contiennent comme adresse source l'"adresse de transport relayée allouée". L'homologue reçoit les données et répond, à nouveau en utilisant un datagramme UDP comme protocole de transport, en envoyant le datagramme UDP à l'adresse de relais au niveau du serveur TURN[18].

Le serveur TURN reçoit le datagramme UDP pair, vérifie les autorisations et si elles sont valides, le transmet au client[18].

Ce processus contourne même les NAT symétriques car le client et le pair peuvent au moins parler au serveur TURN, qui a alloué une adresse IP de relais pour la communication.[18]

Alors que TURN est plus robuste que STUN en ce sens qu'il aide à traverser plus de types de NAT, une communication TURN relaie l'ensemble de la communication via le serveur nécessitant beaucoup plus de bande passante que le protocole STUN, qui ne résout généralement que l'adresse IP et les relais publics. l'information au client et au pair pour qu'ils l'utilisent dans la communication directe. Pour cette raison, le protocole ICE impose l'utilisation de STUN en premier recours, et l'utilisation de TURN uniquement lorsqu'il s'agit de NAT symétriques ou d'autres situations où STUN ne peut pas être utilisé[18].

3.2.7 ICE

L'établissement de connectivité interactive (ICE) est utilisé dans les problèmes où deux nœuds sur Internet doivent communiquer aussi directement que possible, mais la présence de NAT et de pare-feu rend difficile la communication entre les nœuds. Il s'agit d'une technique de mise en réseau qui utilise STUN (Session Traversal Utilities for NAT) et TURN (Traversal Using Relays Around NAT) pour établir une connexion aussi directe que possible entre deux nœuds[2].

ICE est la technique qui utilise les protocoles STUN et TURN pour établir une connexion. Au début, les points d'extrémité ne sont pas conscients de leurs propres topologies de réseau, qu'ils soient sous un seul NAT ou plusieurs niveaux de NAT et le type de NAT, ICE laisse les points d'extrémité le découvrir à l'aide d'un serveur STUN à travers lequel ils peuvent trouver un chemin établir un connexion directe. Si oui, la connexion est établie, sinon un serveur TURN est utilisé comme relais pour échanger des informations entre les points de terminaison[2].

L'algorithme du framework ICE recherche le chemin à la plus faible latence pour connecter les deux pairs, en essayant ces options dans l'ordre [7] :

— Connexion UDP directe (dans ce cas, et uniquement dans ce cas, un serveur STUN est utilisé pour trouver l'adresse réseau d'un pair)

— Connexion TCP directe, via le port HTTP

— Connexion TCP directe, via le port HTTPS

— Connexion indirecte via un serveur relais/TURN (si une connexion directe échoue, par exemple, si un pair est derrière un pare-feu qui bloque la traversée NAT)

ICE exige que STUN soit utilisé par défaut car une communication TURN nécessite l'utilisation continue d'un serveur TURN, la connexion n'est pas d'égal à égal et davantage de ressources de serveur sont utilisées. ICE a été développé par l'Internet Engineering Task Force sous le nom de RFC 8445[2].

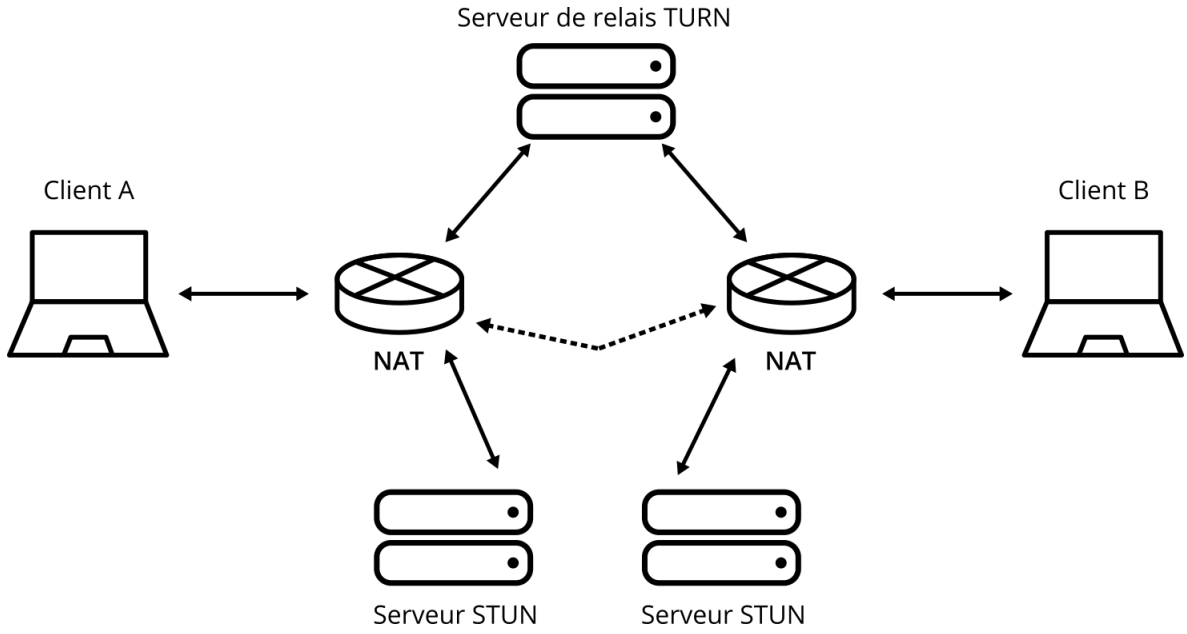

FIGURE 3.4 – Tentatives ICE, options de connectivité directe STUN et TURN

3.3 Protocoles de la couche transport

3.3.1 TLS

Transport Layer Security, TLS, dont les anciennes versions étaient connues sous le nom de Secure Sockets Layer ou SSL, est une couche intermédiaire entre TCP et l'application qui fournit des services de confidentialité et d'authentification. La confidentialité est assurée par le cryptage des paquets « over the wire ». L'authentification est fournie à l'aide de certificats numériques. La navigation Web sécurisée aujourd'hui (HTTPS) utilise uniquement le transport TLS. WebRTC peut tirer parti de TLS pour la signalisation et la sécurité de l'interface utilisateur. Il existe également une

version de TLS qui s'exécute sur UDP, appelée Datagram TLS (3.3.3), et une version qui peut être utilisée pour générer des clés pour SRTP connue sous le nom de DTLS-SRTP.

3.3.2 TCP

Transmission Control Protocol, TCP, est n protocole de couche de transport dans la pile de protocoles Internet qui fournit un transport fiable avec contrôle d'encombrement et contrôle de flux. TCP est utilisé pour transporter le trafic Web (HTTP), mais n'est pas adapté pour transporter le trafic de communication en temps réel tel que RTP, car les retransmissions utilisées pour mettre en œuvre la fiabilité introduisent des délais inacceptables. Comme UDP, TCP utilise un concept de ports, un nombre entier de 16 bits, pour séparer les flux et les protocoles. TCP est fourni par le système d'exploitation sous le navigateur.

3.3.3 DTLS

Datagram TLS, est une version de TLS qui s'exécute sur UDP. Les mêmes propriétés de confidentialité et d'authentification sont fournies. UDP est plus facile à passer par NAT et peut être mieux adapté aux applications peer-to-peer.

3.3.4 UDP

User Datagram Protocol, UDP, est un protocole de couche de transport dans la pile de protocoles Internet qui fournit un service de datagramme non fiable pour les couches supérieures. UDP est couramment utilisé pour transporter de petits échanges de paquets courts (par exemple des paquets DNS, Domain Name Service) ou pour transporter des médias en temps réel tels que RTP. UDP permet un échange d'informations très rapide et efficace ; cependant, les utilisateurs d'UDP doivent faire face à une éventuelle perte de paquets. De plus, UDP n'a pas de contrôle de congestion, les utilisateurs doivent donc être sensibles à la perte de paquets et à la congestion pour éviter de surcharger les connexions Internet. Comme TCP, UDP utilise un concept de ports, un nombre entier de 16 bits, pour séparer les flux et les protocoles[6, p. 138 - 139].

La plupart des applications Internet utilisent un transport fiable, tel que TCP, Transmission Control Protocol, dans lequel les paquets perdus sont automatiquement retransmis. La navigation sur le Web, la messagerie électronique et le streaming audio et vidéo utilisent un transport fiable. Les paquets reçus sont acquittés et l'absence d'accusé de réception après un certain temps déclenche une retransmission des paquets jusqu'à réception d'un accusé de réception. La communication en temps réel ne peut pas tirer parti de ce type de transport fiable en raison du délai impliqué dans la détection de la perte de paquets et la réception des paquets retransmis. Les paquets perdus lors du chargement d'une page Web peuvent entraîner un chargement complet d'une ou deux secondes supplémentaires. Une session de communication en temps réel ne peut

pas s'arrêter une seconde ou deux au milieu d'une conversation vocale, ni geler la lecture d'une vidéo pendant une seconde en attendant la retransmission des informations manquantes. Au lieu de cela, les systèmes de communication en temps réel doivent simplement faire de leur mieux lorsque des informations sont perdues. Les techniques pour couvrir la perte ou minimiser les effets sont connues sous le nom de masquage de perte de paquets ou CPL[6, p. 138 - 139].

La perte moyenne de paquets en général sur Internet est extrêmement faible, de l'ordre de quelques fractions de pour cent. Bien qu'elle se produise rarement, la perte de paquets se produit en rafale, ce qui entraîne une perte élevée de paquets sur de courts intervalles. La capacité à gérer ces événements de perte de courte durée a un impact majeur sur la qualité perçue d'un système de communication[6, p. 138 - 139].

Les codecs avancés, en particulier le codec audio Opus, sont conçus pour offrir une bonne expérience utilisateur, même en cas de perte de paquets élevée. De plus, le retour d'information en temps réel du récepteur des médias offre la possibilité de réduire la bande passante ou la résolution pendant le paquet congestion, offrant une meilleure expérience utilisateur et partageant la bande passante équitablement avec les autres internautes[6, p. 138 - 139].

UDP est fourni par le système d'exploitation sous le navigateur[6, p. 138 - 139].

3.3.5 SCTP

Stream Control Transport Protocol, SCTP, est une couche de transport qui fournit un transport fiable ou non fiable sur IP ainsi qu'un contrôle d'encombrement et plusieurs flux dans une session. Le contrôle de la congestion est la capacité d'un protocole à détecter le début de la perte et du retard de paquets Internet et à ajuster dynamiquement son taux d'envoi pour minimiser les effets. Plusieurs sessions permettent à une seule session d'être divisée en plusieurs flux, chacun pouvant partager la bande passante disponible de la session de manière égale[6, p. 139 -140].

SCTP n'est généralement pas pris en charge dans les systèmes d'exploitation, les navigateurs auront donc leur propre pile de protocole SCTP intégrée pour le canal de données[6, p. 139 -140].

3.4 Conclusion

WebRTC utilise UDP au niveau de la couche de transport en raison de ses exigences en temps réel. SRTP et SCTP sont utilisés à une couche supérieure pour multiplexer les flux. Pour des raisons de sécurité, DTLS est utilisé. Parce que les pairs peuvent s'asseoir derrière des pare-feu et des NAT, ICE/STUN/TURN sont nécessaires[3].

Pour la signalisation, TLS et TCP sont utilisés. Pour échanger des paramètres de session, SDP est utilisé[3].

Références

[1] H.W. BARZ et G.A. BASSETT. *Multimedia Networks : Protocols, Design and Applications*. Anglais. 1re éd. Wiley, 2016. ISBN : 9781119090137.

[2] GEEKSFORGEEKS. *Interactive Connectivity Establishment (ICE)*. Anglais. 17 sept. 2019. URL : https://www.geeksforgeeks.org/interactive-connectivity-establishment-ice/ (visité le 24/07/2021).

[3] Sangeetha-Prabhu GOOGLER77. *WebRTC*. british. 6 fév. 2021. URL : https://devopedia.org/webrtc (visité le 15/07/2021).

[4] Ilya GRIGORIK. *High Performance Browser Networking : What every web developer should know about networking and web performance*. Américain. 1re éd. O'Reilly Media, 2013.

[5] A. HAYTHAM. "Real-time Communication Using WebRTC". Anglais. In : *Special Interest Group on Computer-Human Interaction* (2018).

[6] A.B. JOHNSTON et D.C. BURNETT. *WebRTC : APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Anglais. 2e éd. Smashwords Edition, 2013. ISBN : 978-0-9859788-5-3.

[7] MOZILLA ORG. *ICE - MDN Web Docs Glossary : Definitions of Web-related terms*. Anglais. 13 jan. 2021. URL : https://developer.mozilla.org/en-US/docs/Glossary/ICE (visité le 24/07/2021).

[8] MOZILLA ORG. *MediaTrackConstraints - Web APIs*. Anglais. 14 juin 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/MediaTrackConstraints.

[9] MOZILLA ORG. *Navigator - Web APIs*. Anglais. 15 juin 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/Navigator (visité le 16/06/2021).

[10] MOZILLA ORG. *RTCSessionDescription - Web APIs*. Anglais. 31 mai 2021. URL : https://developer.mozilla.org/en-US/docs/Web/API/RTCSessionDescription (visité le 16/06/2021).

[11] ONSIP. *RTCPeerConnection - WebRTC Explained*. Anglais. 2021. URL : https://www.onsip.com/voip-resources/voip-fundamentals/rtcpeerconnection (visité le 16/06/2021).

[12] K.G. PATHARE et P.M. CHOURAGADE. "WebRTC Implementation and Architecture Analysis". Anglais. In : *International Journal of Scientific & Engineering Research* 7 (2016).

[13] S. RAJAN. *getUserMedia API - An introduction*. Anglais. 16 jan. 2021. URL : https://www.thegeeksclan.com/getusermedia-api-an-introduction/ (visité le 18/06/2021).

[14] A. ROSA. *An Introduction to the getUserMedia API*. Anglais. 1er jan. 2014. URL : https://www.sitepoint.com/introduction-getusermedia-api/ (visité le 18/06/2021).

[15] WEBRTC ORG. *Getting started with peer connections*. Anglais. 2014. URL : https://webrtc.org/getting-started/peer-connections (visité le 16/06/2021-06/07/2021).

[16] WIKIPEDIA CONTRIBUTORS. *Interactive Connectivity Establishment*. Anglais. 23 août 2020. URL : https://en.wikipedia.org/wiki/Interactive_Connectivity_Establishment (visité le 16/06/2021).

[17] WIKIPEDIA CONTRIBUTORS. *STUN*. Anglais. 10 avr. 2021. URL : https://en.wikipedia.org/wiki/STUN (visité le 19/07/2021).

[18] WIKIPEDIA CONTRIBUTORS. *Traversal Using Relays around NAT*. Anglais. 20 jan. 2021. URL : https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT (visité le 23/07/2021).

Annexes

A Exemple test de code Javascript dans LaTeX

A.1 Premier snippet

```
(function(){
  var user = getCookie("username");
  document.getElementById("#date-field").innerHTML = new Date();
  document.getElementById("#greetings").innerHTML = "<p>Hello, " + user.name +
    ".</p>";
})()

function getCookie(cname) {
  var name = cname + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var ca = decodedCookie.split(';');
  for(var i = 0; i < ca.length; ++i) {
    var c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

A.2 Second snippet

```
/* eslint-env es6 */
/* eslint-disable no-unused-vars */
import Axios from 'axios'
import { BASE_URL } from './utils/api'
import { getAPIToken } from './utils/helpers'

export default class User {
  constructor () {
    this.id = null
    this.username = null
    this.email = ''
    this.isActive = false
    this.lastLogin = ''  // ISO 8601 formatted timestamp.
    this.lastPWChange = ''  // ISO 8601 formatted timestamp.
  }
}

const getUserProfile = async (id) => {
  let user = new User()
  await Axios.get(
    `${BASE_URL}/users/${id}`,
    {
      headers: {
        'Authorization': `Token ${getAPIToken()}`,
      }
    }
  ).then{response => {
    // ...
  }).catch(error => {
    // ...
  }) }
```