

**INSA Lyon – Département Télécommunications**

**Année universitaire : 2025 – 2026**

# **Appareil de diffusion vidéo non intrusif**

## **WebRTC**

### **Encadrant :**

Stéphane Frenot - Damien Reimert

### **Réalisé par :**

- Orhon Gabriel
- Chkoundali Yasmine
- Mohammi Islam
- Abidi Jean
- Adjami Axel

## Sommaire

<b>Abstract.....</b>	<b>4</b>
<b>1. Définition.....</b>	<b>5</b>
<b>2 Composants WebRTC.....</b>	<b>5</b>
<b>2.1 Composants clés.....</b>	<b>5</b>
<b>2.2 TURN/STUN:.....</b>	<b>7</b>
<b>2.3 Le rôle du navigateur :.....</b>	<b>7</b>
<b>2.4 Le pipeline de traitement du flux.....</b>	<b>8</b>
<b>3. Mécanismes utilisés par WebRTC.....</b>	<b>8</b>
<b>3.1. Signalisation et établissement de la connexion.....</b>	<b>8</b>
<b>3.3. Transport et sécurité.....</b>	<b>10</b>
<b>4. Performances et Limites.....</b>	<b>12</b>
<b>4.1. Qualité de service et adaptation.....</b>	<b>12</b>
<b>4.2. Limites.....</b>	<b>12</b>

Version	Date	Auteur	Modifications
v0	19/12/2025	Jean	<p><b>Création initiale :</b> Analyse basée sur les impacts Qualité de service / Confidentialité. Identification des risques de Hijack, DoS, et VP-Spoofing.</p>
v1	09/01/2026	Islam	<ul style="list-style-type: none"> <li>– <b>Modifications:</b> Abstract, Définition , composants webRTC , description DTLS,SRTP et SCTP</li> <li>– <b>Ajouts :</b> Cas d'usage avec tc-net</li> <li>– <b>Suppressions :</b> Codecs et performances , perspectives , conclusion</li> </ul>

## Abstract

Ce document explique le fonctionnement de **WebRTC** et son utilisation dans le cadre de notre projet de diffusion d'écran en temps réel. Le projet utilise WebRTC pour diffuser l'écran d'un PC utilisateur vers un Raspberry Pi connecté à un vidéoprojecteur via une connexion pair-à-pair (P2P). Nous détaillons les composants essentiels de WebRTC. Nous expliquons également le rôle de **STUN** et **TURN** dans la découverte des adresses réseau et la gestion de la connectivité à travers des NAT, et précisons que, bien que **TURN** et **STUN** ne soient pas nécessaires dans notre cas, nous utilisons **STUN** pour découvrir les chemins réseau potentiels, mais étant donné que les deux appareils sont sur le même réseau local eduroam, nous optons finalement pour l'utilisation de l'adresse locale, qui est le chemin le plus efficace et direct. Ce document couvre aussi la sécurisation des échanges via DTLS et SRTP, ainsi que les processus de signalisation et de traitement des flux multimédia. Enfin, nous discutons des performances de WebRTC, des mécanismes de contrôle de qualité et de congestion, et des limitations de l'architecture .

## 1. Définition

WebRTC est un standard open source permettant l'échange de flux audio, vidéo et de données en temps réel entre navigateurs web. Il repose sur plusieurs principes fondamentaux :

- **Communication temps réel** : optimisation pour des échanges interactifs à faible latence ;
- **Modèle pair-à-pair (P2P)** : transport direct des flux multimédias entre les terminaux lorsque les conditions réseau le permettent ;
- **Intégration native** : absence de plugins ou de logiciels tiers, WebRTC étant directement implémenté dans les navigateurs modernes à l'instar de Chrome, Firefox, Safari, Edge, etc.

Dans ce projet, WebRTC est utilisé spécifiquement pour le dépôt d'affichage. Plus précisément, il permet la diffusion d'écran en temps réel depuis un PC utilisateur vers un Raspberry Pi connecté en HDMI à un vidéoprojecteur dans une salle. Ce processus repose sur la capacité de WebRTC à capturer le flux vidéo de l'écran de l'utilisateur via l'API `getDisplayMedia()` et à le transmettre de manière sécurisée et fluide au Raspberry Pi via une connexion pair-à-pair. Ce dernier redirige ensuite le flux vidéo vers un vidéoprojecteur pour afficher le contenu à l'écran.

Aussi, l'architecture globale de WebRTC repose sur une séparation claire entre le **plan de contrôle** (signalisation) et le **plan de données** (transport des médias). Elle implique trois entités principales :

- **Client A** : navigateur initiateur de la communication ;
- **Client B** : navigateur récepteur ;
- **Serveur de signalisation** : intermédiaire chargé de l'échange des informations de session.

Le serveur de signalisation ne transporte pas les flux multimédias. Une fois la session établie, les flux audio et vidéo sont échangés directement entre les deux clients selon un modèle pair-à-pair.

## 2 Composants WebRTC

### 2.1 Composants clés

La spécification WebRTC définit deux composants principaux accessibles via des API JavaScript :

- **RTCPeerConnection** : composant central assurant la négociation de session, la gestion des flux RTP, la connectivité réseau et la sécurité ;

- **RTCDataChannel** : canal de données bidirectionnel reposant sur SCTP, permettant l'échange de messages ou de fichiers.

## Interactive Connectivity Establishment (ICE)

ICE est un protocole utilisé par RTCPeerConnection pour assurer que les flux multimédias peuvent être envoyés directement, même dans des environnements réseau complexes. Il est responsable de la découverte des adresses réseau et de la sélection du meilleur chemin entre les pairs.

Le processus ICE repose sur plusieurs éléments clés qui facilitent cette découverte et cette sélection de chemin :

- **RTCIceCandidate** :

Un ICE Candidate représente un chemin réseau potentiel que l'une des parties peut utiliser pour communiquer. Ce candidat contient des informations comme l'adresse IP, le port, et le type de transport (UDP, TCP, etc.). Ces informations sont échangées entre les deux pairs pour permettre à **RTCPeerConnection** de tester les différents chemins réseau disponibles.

Exemple de types de candidats :

- **Host Candidate** : Représente les adresses locales de la machine.
- **Server Reflexive (srflx)** : L'adresse publique visible de la machine, obtenue via un serveur STUN.
- **Relay Candidate (relay)** : Un candidat d'un serveur TURN, utilisé comme relais si la connexion directe échoue.

- **RTCIceCandidatePair** :

Un **pair de candidats ICE** représente une combinaison de deux candidats. ICE teste la connectivité entre ces candidats pour déterminer si la communication peut avoir lieu via ce chemin.

- **RTCIceConnectionState**

Une fois les candidats collectés, **RTCIceConnectionState** indique l'état de la connexion, c'est-à-dire si la connectivité a été validée. Les états incluent :

- **New** : Connexion non encore établie.
- **Checking** : Le mécanisme ICE est en train de tester les différents candidats.

- **Connected** : Un chemin valide a été trouvé et la connexion est établie.
- **Failed** : Aucun chemin valide n'a pu être trouvé.

- **RTCIceRole**

Cette interface définit le rôle d'un pair dans une session ICE. Un pair peut être dans l'un des deux rôles suivants :

- **Host** : L'initiateur de la connexion, qui commence le processus de négociation.
- **Server** : Le pair qui attend la connexion, typiquement après avoir reçu une offre de connexion.

- **RTCIceTransportState** :

Une fois un chemin valide sélectionné, **RTCIceTransportState** indique l'état du transport des données.

## 2.2 TURN/STUN:

**Session Traversal Utilities for NAT (STUN)** et **Traversal Using Relays around NAT (TURN)** sont des protocoles utilisés dans WebRTC pour faciliter la découverte des pairs et la communication à travers des **Network Address Translation(NAT)** et des pare-feu. **STUN** aide à découvrir l'adresse IP publique d'un appareil, tandis que **TURN** fonctionne comme un relais de données lorsque la connexion directe échoue, servant de solution de secours pour traverser des configurations NAT plus complexes.

Dans le cadre de notre projet, nous n'utilisons pas de serveur TURN car les deux appareils sont sur eduroam. Dans ce contexte, la communication directe entre les appareils peut se faire via leurs adresses IP locales, rendant ainsi l'utilisation de **TURN** inutile. De plus, puisque les deux pairs sont dans le même réseau, nous n'avons même pas besoin de **STUN** pour découvrir une adresse publique, car les chemins locaux suffisent à établir une connexion P2P efficace.

## 2.3 Le rôle du navigateur :

Le navigateur intègre un moteur WebRTC complet. Il prend en charge :

- l'encodage et le décodage des flux audio et vidéo ;
- la gestion des codecs ;
- l'application de traitements multimédias (annulation d'écho, jitter buffer) ;

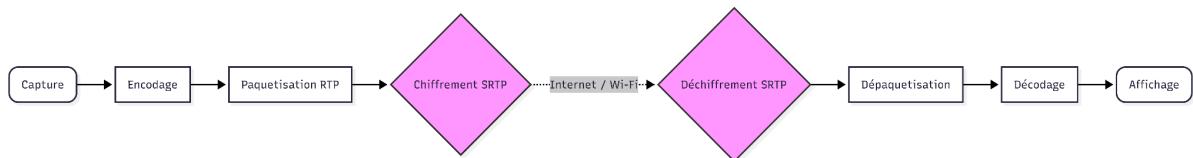
- le contrôle des accès matériels via un système de permissions utilisateur.

## 2.4 Le pipeline de traitement du flux

Le traitement des flux multimédias suit un pipeline bien défini :

1. Capture du média ;
2. Encodage audio ou vidéo ;
3. Paquetisation RTP ;
4. Chiffrement SRTP ;
5. Transport réseau ;
6. Déchiffrement, dépaquetisation et décodage côté réception.

Le schéma suivant illustre cela comme il faut :



## 3. Mécanismes utilisés par WebRTC

### 3.1. Signalisation et établissement de la connexion

#### 3.1.1. Principe de la signalisation

WebRTC ne spécifie pas de protocole de signalisation. Cette phase est volontairement laissée au choix du développeur afin de permettre une grande flexibilité. La signalisation a pour rôle l'échange des métadonnées nécessaires à l'établissement de la communication avant le transport direct des médias.

Quand un agent WebRTC démarre, il n'a aucune idée avec qui il va communiquer et en quoi consiste cette communication. La signalisation résout ce problème. La signalisation est utilisée pour amorcer l'appel pour que deux agents WebRTC puissent commencer à communiquer.

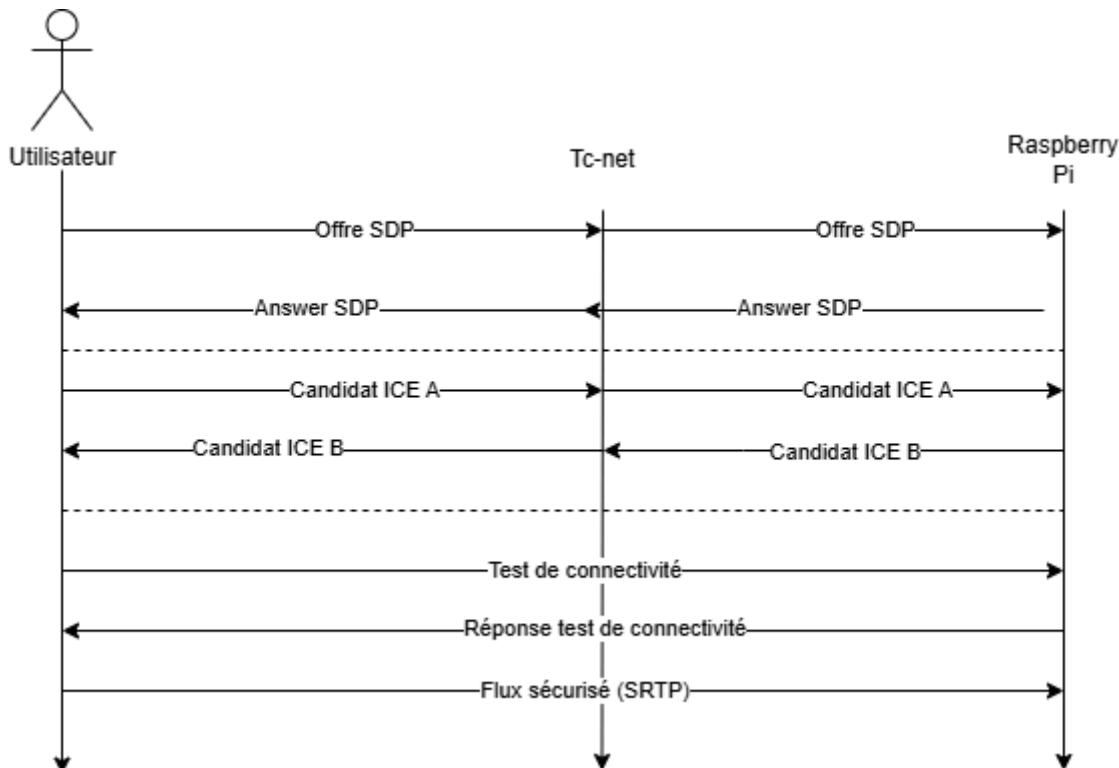
#### 3.1.2. SDP (Session Description Protocol)

Les informations de session sont encapsulées dans le format SDP. Celui-ci contient notamment :

- la description des médias (audio, vidéo) ;
- les codecs supportés et leurs paramètres ;

- les adresses réseau et ports ;
- les candidats ICE ;
- les empreintes de sécurité (fingerprints DTLS).

### 3.2. Diagramme de séquence d'échanges WebRTC dans le scénario d'usage en TC



#### 3.2.1 Établissement d'une connexion WebRTC dans tc-net

Dans le cadre de ce projet, WebRTC est utilisé afin d'établir une communication temps réel entre deux clients connectés sur eduroam, à l'aide de l'infra tc-net du département TC. L'application est servie via un serveur HTTPS hébergé sur tc-net, garantissant un contexte sécurisé conforme aux exigences de WebRTC. Les échanges de signalisation nécessaires à l'établissement de la connexion, incluant les descriptions de session SDP et les candidats ICE, sont assurés par un serveur WebSocket également hébergé sur tc-net, distinct du transport effectif des flux multimédias.

Lorsqu'un utilisateur initie une session, les deux pairs se coordonnent à l'aide du serveur de signalisation afin d'échanger une offre et une réponse SDP décrivant les paramètres de la communication, notamment les codecs, les flux médias et les mécanismes de sécurité. Chaque pair collecte ses informations de connectivité réseau à l'aide du mécanisme ICE. Dans le contexte d'eduroam, cette collecte produit principalement des candidats de type *host*, correspondant aux adresses IP internes attribuées aux machines, ainsi que, dans certains cas, des candidats de type *server reflexive* obtenus via un serveur Turn configuré sur tc-net. Ce dernier permet de connaître l'adresse réseau sous laquelle un pair est visible depuis l'extérieur, bien que la connexion directe locale soit généralement privilégiée.

Les candidats ICE sont échangés via le serveur de signalisation et soumis à des tests de connectivité. Une fois la connectivité établie, le flux vidéo est transmis entre les machines à l'aide des protocoles sécurisés imposés par WebRTC, notamment DTLS pour l'échange des clés et SRTP pour le chiffrement des flux multimédias.

### 3.3. Transport et sécurité

#### 3.3.1. Transport

WebRTC privilégie nativement le protocole **UDP** (User Datagram Protocol) pour le transport des flux multimédias (via RTP). Contrairement à TCP, UDP ne garantit pas la réception ordonnée des paquets ("Fire and Forget"), ce qui élimine le problème de **Head-of-Line Blocking**. En temps réel, il est préférable de perdre une image (glitch visuel mineur) plutôt que d'attendre sa retransmission et de geler le flux. En cas de blocage strict par un pare-feu d'entreprise, WebRTC peut basculer en mode dégradé sur **TCP** (via un serveur TURN ou ICE-TCP), au prix d'une latence accrue.

#### 3.3.2. Sécurité intégrée

La sécurité n'est pas une option dans WebRTC, elle est obligatoire. L'architecture repose sur la pile **DTLS-SRTP** :

- **DTLS** est un protocole de sécurité basé sur TLS (Transport Layer Security), mais spécifiquement conçu pour être utilisé avec des protocoles sans connexion, tels que UDP. Ce protocole assure la sécurisation de l'échange de clés et l'authentification des pairs dans des environnements où la latence est critique, comme dans les communications en temps réel de WebRTC. Lors de l'établissement d'une connexion entre deux pairs, DTLS garantit que les deux parties sont bien celles qu'elles prétendent être. Pour cela, il effectue une vérification des certificats via des empreintes numériques et procède à un handshake sécurisé où les clés de chiffrement sont échangées de manière protégée. Ce processus permet de se prémunir contre les

attaques de type Man-in-the-Middle (MitM), où un attaquant pourrait tenter d'intercepter ou de modifier les communications.

Dans le contexte de notre projet, où un PC utilisateur diffuse son écran vers un Raspberry Pi connecté à un vidéoprojecteur, DTLS est essentiel pour établir une connexion sécurisée et authentifiée entre les deux appareils. Lorsqu'un utilisateur initie la diffusion d'écran, DTLS protège l'échange de clés et les certificats numériques entre le PC et le Raspberry Pi, assurant ainsi qu'aucun acteur malveillant ne puisse intercepter ou altérer les flux multimédia. Ce protocole garantit que les informations échangées, sont bien sécurisées avant d'être envoyées sur le réseau, empêchant ainsi toute écoute non autorisée.

- **Secure Real-time Transport Protocol (SRTP)** est un protocole conçu pour sécuriser les flux multimédia en temps réel, en les chiffrant. SRTP repose sur des mécanismes de chiffrement, authentification et protection contre les attaques de rejouement pour garantir que les données transmises ne soient ni interceptées ni modifiées en transit. Le protocole utilise généralement AES-128 pour le chiffrement des flux, ce qui assure que les contenus multimédia sont inaccessibles pour les acteurs malveillants. En plus du chiffrement, SRTP ajoute des mécanismes d'authentification pour vérifier l'intégrité des paquets et protéger contre les altérations des données. Il offre également une protection contre les attaques de type replay, en empêchant les paquets capturés d'être réutilisés.

Dans le cadre de notre projet, SRTP est utilisé pour sécuriser le flux vidéo de l'écran du PC utilisateur pendant la diffusion vers le Raspberry Pi. Le chiffrement des paquets vidéo garantit que le contenu de l'écran projeté sur le vidéoprojecteur reste confidentiel et inaccessible à toute personne non autorisée. Cela est particulièrement important dans un environnement de réseau local comme eduroam, où plusieurs utilisateurs pourraient potentiellement accéder au même réseau. Grâce à SRTP, même si des paquets sont interceptés par un attaquant sur le réseau, leur contenu reste illisible, garantissant ainsi la protection de l'information projetée.

- **Stream Control Transmission Protocol (SCTP)** est un protocole de transport qui permet de gérer plusieurs flux de données de manière fiable tout en offrant un multiplexage de flux sur un même canal de communication. Contrairement à TCP, qui est conçu pour un flux de données unique, SCTP permet de transporter plusieurs types de données simultanément tout en assurant leur fiabilité et leur ordonnancement correct. Ce protocole est particulièrement utile pour des applications nécessitant l'envoi simultané de différentes sortes de données (comme l'audio, la vidéo et les fichiers) tout en garantissant leur livraison dans le bon ordre.

Dans notre projet, SCTP est utilisé pour sécuriser les échanges de données de contrôle via les RTCDataChannels. Par exemple, si l'utilisateur souhaite envoyer des

commandes de contrôle au Raspberry Pi , ces commandes sont envoyées via un RTCDataChannel sécurisé par SCTP. Le protocole SCTP permet de gérer ces commandes de manière fiable et sécurisée, tout en permettant le multiplexage de ces messages et des flux multimédia. Cela garantit que, même si plusieurs types de données sont envoyés simultanément, la communication reste fluide et les données ne sont pas perdues ou désordonnées. En combinant SCTP avec DTLS pour le chiffrement, toutes les données échangées sont protégées contre les risques d'interception ou de modification.

### 3.3.3. Contrôle et feedback

Le protocole de contrôle RTP Control Protocol (RTCP) agit comme une boucle de rétroaction. Il remonte périodiquement des statistiques (Perte de paquets, RTT - Round Trip Time) à l'émetteur. Pour la résilience aux erreurs, WebRTC utilise des mécanismes avancés :

- **Negative Acknowledgement (NACK)** : Demande la retransmission sélective d'un paquet critique manquant.
- **Picture Loss Indication (PLI) / Full Intra Request (FIR)** : En cas de corruption majeure de l'image (artefacts), le récepteur demande l'envoi immédiat d'une nouvelle image clé (Keyframe) pour réinitialiser le décodeur.

## 4. Performances et Limites

### 4.1. Qualité de service et adaptation

WebRTC intègre des mécanismes de contrôle de congestion et de surveillance de la qualité de service (latence, jitter, pertes). Les statistiques exposées via `getStats()` permettent d'analyser les performances en temps réel.

### 4.2. Limites

- **Scalabilité (Mesh vs SFU)** : En architecture P2P (Mesh), chaque client doit envoyer son flux à tous les autres. La bande passante montante (Upload) du client sature rapidement au-delà de 3-4 participants. Pour des usages à grande échelle, un serveur central (SFU) est requis.
- **Consommation Énergétique/CPU** : L'encodage temps réel est coûteux. Sur des appareils mobiles ou embarqués, l'absence d'accélération matérielle peut drainer la batterie ou causer une surchauffe.