Signalisation dans l'architecture WebRTC

EFORT
http://www.efort.com

1. WebRTC

WebRTC offre une communication temps réel de manière native à partir d 'un navigateur Web. Les navigateurs Web s'échangent directement des flux de données multimédia; aucun serveur intermédiaire n 'est impliqué. WebRTC est un « media engine » avec des APIs Javascript.
Les APIs sont normalisées par le Web Real-Time Communications Working Group du World Wide Web Consortium (W3C) et l'ensemble des protocoles de communication par le Real-Time Communication in WEB-browsers Working Group de Internet Engineering Task Force (IETF).
WebRTC est une technologie mais n'est pas une solution. Il faut développer le service à partir de WebRTC.
Bien que la communication de base WebRTC utilise le mode peer-to-peer, l'étape initiale d'établissement de cette communication requiert de la coordination.
Cette coordination est fournie par un serveur Web et/ou un serveur de signalisation.
Cela permet à deux ou plusieurs navigateurs Web disposant de la capacité WebRTC de se joindre, d'échanger les informations de contact, de négocier une session qui définit la manière dont ils vont communiquer, puis finalement d 'établir les canaux média peer-to-peer pour le transport des flux média échangés directement entre eux.
Le standard WebRTC ne définit pas le protocole de signalisation à utiliser et comment mettre en œuvre cette signalisation. Chaque solution WebRTC doit définir son propre protocole de signalisation ou réutiliser un protocole existant. Cependant, dans le contexte de l 'interfonctionnement entre WebRTC et l'architecture de réseau et de service IMS des opérateurs de télécommunication (VoLTE, WiFiCalling, Voix sur IP fixe résidentielle et entreprises), l 'usage de SIP est très approprié.
Le but de ce tutoriel est de décrire le plan de signalisation WebRTC et de montrer comment utiliser SIP comme protocole de signalisation.

2. WebRTC et signalisation

Un canal de signalisation est nécessaire afin d'échanger trois types d 'information entre peers WebRTC :

- Contrôle de session média : établir et libérer la communication
- Configuration réseau des nœuds : adresse de transport (adresse IP et numéro de port) pour l 'échange des données temps réels même en présence de NAT
- Capacités multimédia des nœuds : médias supportés, codecs disponibles, résolutions supportées, fréquence d'envoi des paquets, etc.

Aucun flux média ne peut être échangé tant que les informations ci-dessus n'ont pas été proprement échangées et négociées.

HTTP peut être utilisé pour transporter la signalisation WebRTC. Un navigateur peut initier une nouvelle requête HTTP pour envoyer et recevoir l'information de signalisation au/du serveur. L'information peut être transportée via les méthodes HTTP GET ou POST, ou dans les réponses. Si le serveur utilisé pour la signalisation supporte le mécanisme CORS (Cross

Origin Resource Sharing), le serveur de signalisation peut être localisé à une adresse IP différente que celle du serveur WEB.
Cross-origin resource sharing (CORS) est une spécification W3C, qui autorise les requêtes Cross-Domain. Elle permet de gérer les accès à une ressource sur un serveur, lié à un domaine, par un script provenant d'un serveur lié à un autre domaine.
Le standard CORS fonctionne à l'aide du champ « Access-Control-Allow-Origin » qui est rajouté dans le header de la requête HTTP. Ce champ permet de vérifier si les scripts provenant du domaine d'origine ont le droit d'accéder à des ressources depuis le serveur demandé. La figure 1 montre le cas où le serveur Web est dissocié du serveur de signalisation. Il est aussi possible de considérer que le serveur Web et le serveur de signalisation soient confondus.

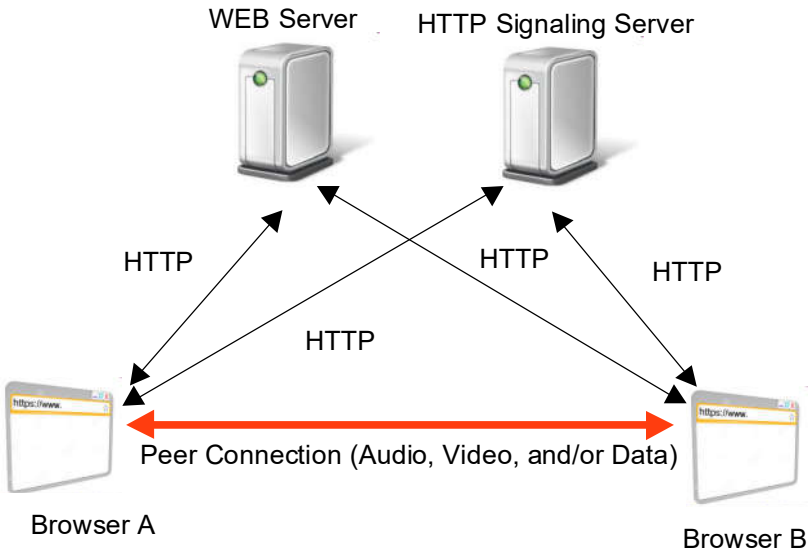

Figure 1 : Cas où le serveur WEB et serveur de signalisation sont dissociés

3. Approche de signalisation XHR

Une approche simple de signalisation propriétaire est l'utilisation du polling HTTP. Un exemple de ceci est le canal de signalisation sur la base XHR.
Des appels XML HTTP request (XHR) au sein de JavaScript permettent à une application JavaScript de générer une nouvelle requête HTTP à un serveur et de traiter la réponse. XHR est une API standard du W3C.
Malgré son nom, XHR peut être utilisé pour envoyer plus que de simples requêtes XML. Les formats JSON (JavaScript Object Notation) ou Plain Text sont aussi possibles.
XHR conduit le navigateur à générer une nouvelle requête HTTP ou HTTPS, telle que GET, PUT, POST et DELETE. L 'API spécifie la méthode à utiliser, ainsi que l'adresse IP et le numéro de port du serveur. La réponse à la requête est retournée au JavaScript.
Pour utiliser XHR comme canal de signalisation pour WebRTC, le serveur Web doit exécuter une application qui reçoit la requête HTTP et relaie ou transmet les informations reçues d'un navigateur à l'autre navigateur sur un autre canal XHR, comme le montre la figure 2.
Pour échanger des informations de signalisation, le JavaScript s'exécutant dans chaque navigateur envoie les messages HTTP au serveur de signalisation à des intervalles réguliers pour l'interroger (polling). Les informations de signalisation envoyées par le navigateur sont incluse dans la méthode HTTP POST. Les informations de signalisation reçues du serveur sont incluses dans la réponse 200 OK associée à la méthode POST. On notera que dans cette approche, chaque message est une nouvelle requête.

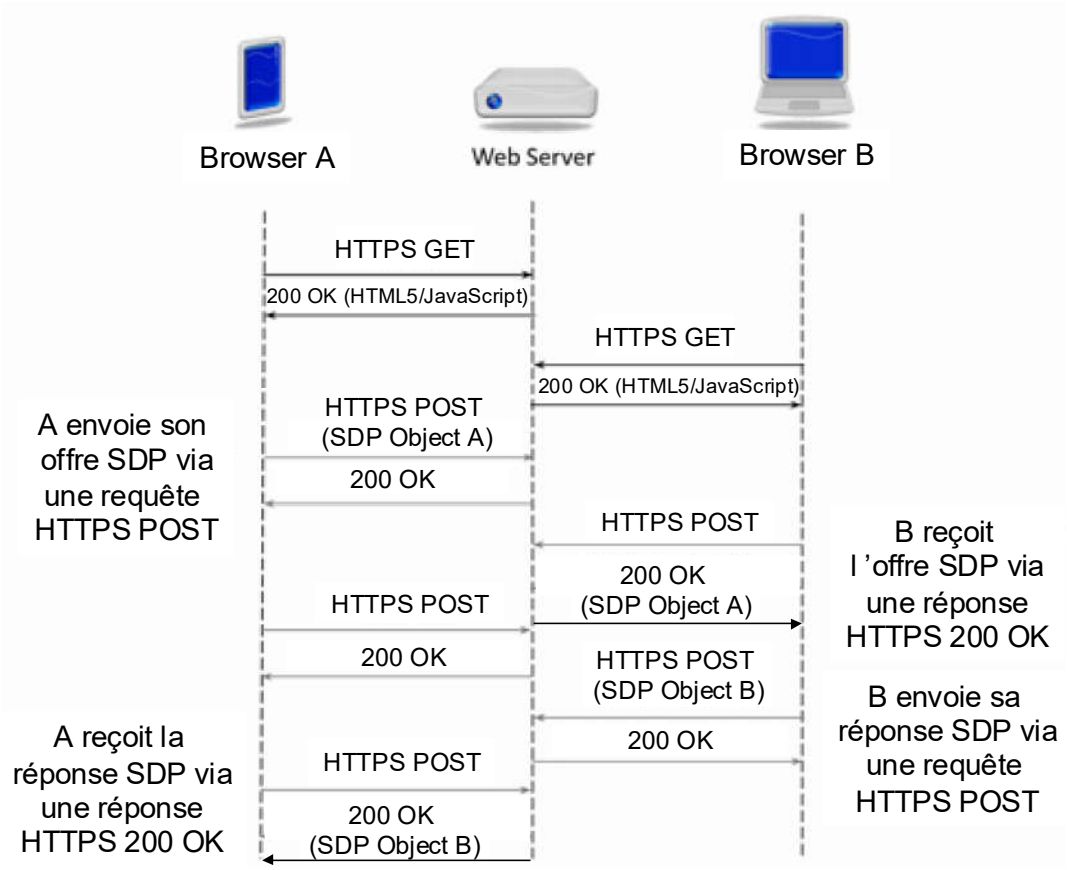

Figure 2 : Approche de signalisation XHR

3.1. Polling HTTP

Le polling est basé sur le client envoyant des requêtes HTTP à intervalle régulier et le serveur retournant immédiatement la réponse HTTP associée. Toutes les données que le serveur doit transporter au client sont incluses dans la réponse HTTP.
Alors que le polling est un mécanisme simple et fonctionne sur la plupart des clients et des serveurs Web, il présente un certain nombre d'inconvénients. Tout d'abord, chaque fois qu'une requête HTTP est à envoyer, une nouvelle connexion TCP vers le serveur doit être établie. La connexion est alors fermée lorsque le client a reçu la réponse HTTP associée. En fonction de la charge du réseau, l'établissement de la connexion TCP peut également prendre un certain temps.
En second lieu, il n'y a pas de garantie que les messages HTTP soient reçus dans le même ordre qu'ils ont été envoyés. Les applications Web doivent prendre cela en considération par exemple en ayant seulement une transaction HTTP vers le serveur Web à un moment donné.
La figure 3 montre un exemple d'utilisation du polling. Le serveur reçoit un événement avec les données qui doivent être transportés vers le client. Lorsque le serveur reçoit une requête HTTP du client, il inclut les données dans la réponse HTTP associée, renvoyée au client.

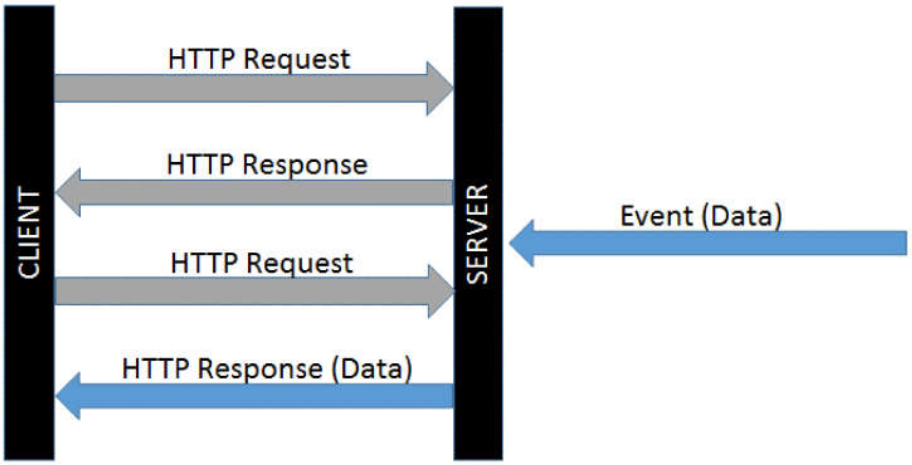

Figure 3 : Polling HTTP normal

3.2. Long polling HTTP

Le mécanisme long polling HTTP utilise également les réponses HTTP pour envoyer des données à partir du serveur Web au client. La principale différence par rapport au polling normal est que le serveur n'envoie pas immédiatement la réponse HTTP quand il reçoit la demande. Au lieu de cela, le serveur attend jusqu'à ce qu'il y ait des données à envoyer. De ce fait, la connexion TCP entre le client et le serveur est maintenue ouverte pendant un temps plus long, éliminant les problèmes associés à l'ouverture et la fermeture fréquentes des connexions TCP.
Cependant, si le serveur doit envoyer fréquemment des données au client, le résultat sera similaire au polling normal.
La figure montre un exemple d'utilisation de long polling. Le serveur reçoit un événement avec les données qu'il doit retourner au client. Lorsque le serveur reçoit une requête HTTP du client, il ne transmet pas la réponse HTTP associé jusqu'à ce qu'il ait reçu des données qu'il doit retourner au client.

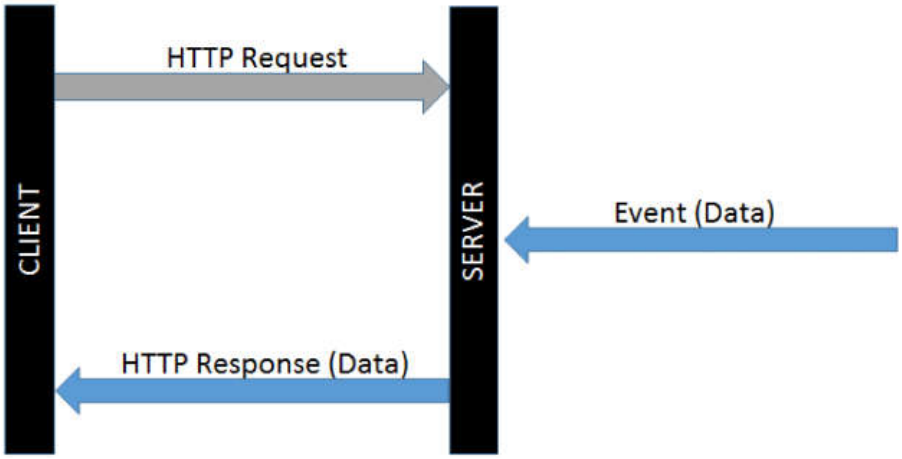

Figure 4 : Long Polling HTTP

4. WebSocket

Avec le besoin croissant d'interactivité dans les applications et les sites web, le mode de communication HTTP classique a montré ses limites en termes d'interactions client-serveur. AJAX a permis quelques améliorations en ce qui concerne l'interactivité, en permettant des

requêtes et un rafraîchissement "à la volée" du contenu d'une page, mais en s'appuyant sur un mode de requêtage traditionnel. Les échanges se font toujours de manière unidirectionnelle : la requête est initiée par le client, le serveur fournit une réponse, le client l'affiche.
Depuis 2011, l'IETF a publié le protocole WebSocket et le W3C la standardisation de son API, déjà implémentée par tous les navigateurs récents.
Le protocole TCP WebSocket permet une communication bidirectionnelle entre un client et un serveur, c'est-à-dire qu'un client connecté à un serveur WebSocket peut non seulement lui envoyer des messages, mais aussi en recevoir sans qu'il n'ait eu à envoyer une requête (communément appelé le "push de données").
La connexion n'est plus établie à chaque requête comme dans un échange HTTP classique, mais de manière persistante, au cours de laquelle un certain nombre de messages sont échangés.

Le principe de fonctionnement est simple (Figure 5):

- La connexion est initiée à la demande du client par une demande de handshake en HTTP, en demandant un upgrade de la connexion HTTP en WebSocket.
- Le serveur, s'il le supporte et s'il l'accepte, répond à la demande de handshake.
- A partir de là, la communication entre le client et le serveur s'effectuera en mode bidirectionnel sur le protocole WebSocket.

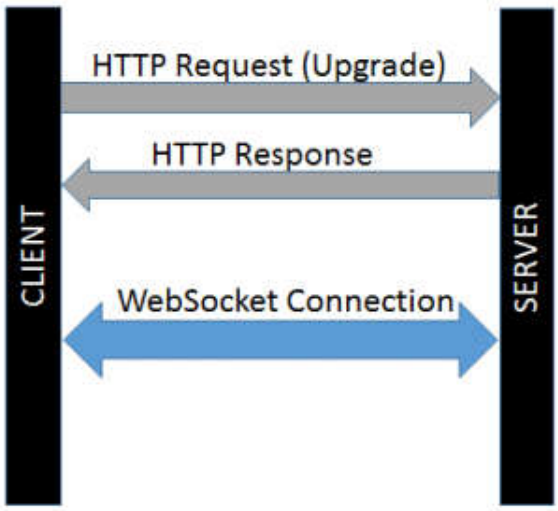

Figure 5 : WebSocket

Un serveur WebSocket utilisé pour la signalisation WebRTC est un serveur qui a une adresse IP publique et qui est accessible par les deux navigateurs établissant la Peer Connection. Chaque navigateur ouvre une connexion WebSocket indépendante avec le même serveur, et le serveur joint les connexions relayant les informations de l'un à l'autre, comme montré à la figure 6.
Comme JavaScript ne supporte pas les interrogations DNS, l'identité du serveur WebSocket devra être fournie par le serveur Web sous forme d'adresse IP et numéro de port.

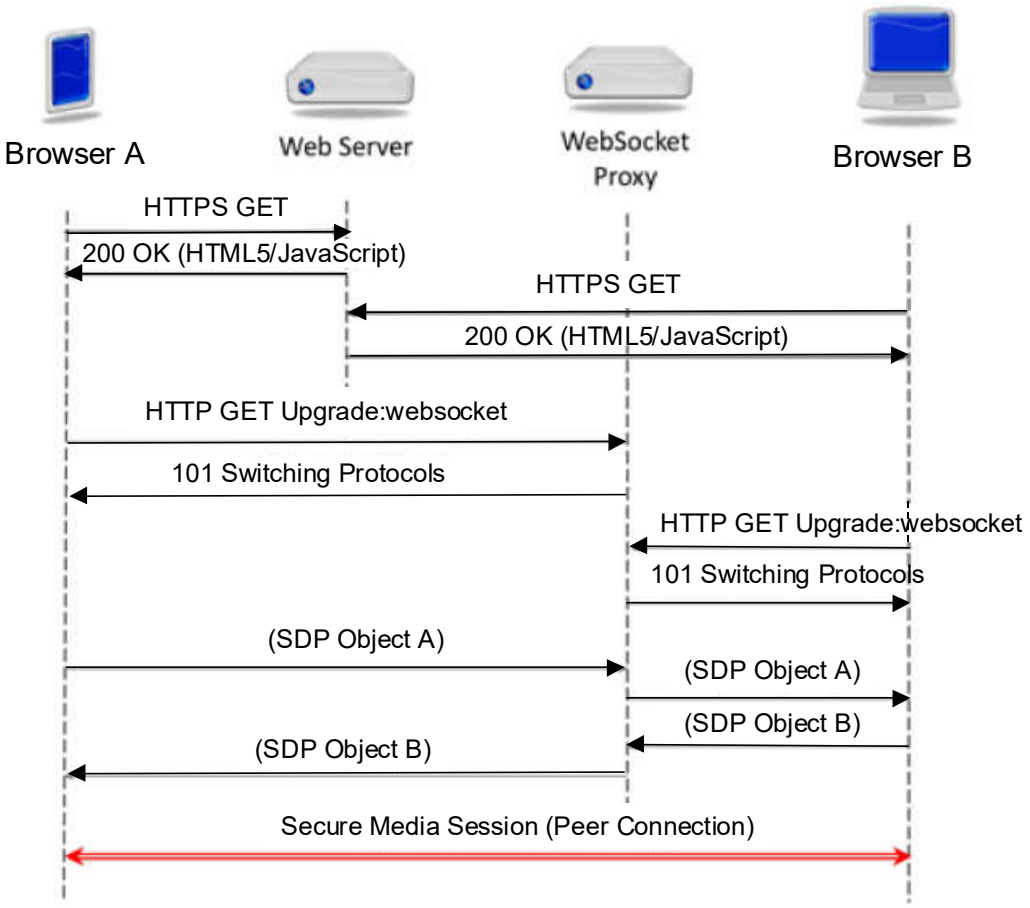

Figure 6 : Serveur WebSocket

5. SIP over WebSocket

La négociation de connexion WebSocket est basé sur HTTP et utilise la méthode HTTP GET avec une requête "Upgrade". Ce message est envoyé par le client et ensuite acquitté par le serveur (si la négociation réussit) avec un code d'état HTTP 101.
Une fois la négociation terminée, le protocole qui utilise la connexion passe du protocole HTTP au protocole WebSocket.
Cette procédure de négociation est conçue afin de réutiliser l'infrastructure HTTP existante.
Lors de la négociation de la connexion, le client et serveur se mettent d'accord sur le protocole d'application à utiliser au-dessus du transport WebSocket. Un tel protocole d'application (également connu sous le nom "Subprotocol WebSocket") définit le format et la sémantique des messages échangés par les points d'extrémité. Cela pourrait être un protocole personnalisé ou être un protocole standardisé (e.g., SIP).
Une fois que la réponse HTTP 101 est traitée, à la fois le client et le serveur réutilisent la connexion TCP sous-jacente pour l'envoi de messages WebSocket et des messages de commande entre eux (Ping, Pong, Close).
Contrairement à HTTP, cette connexion est persistante et peut être utilisée pour de multiples échanges de messages en mode bidirectionnel.

Ci-dessous est présenté un exemple de négociation WebSocket dans laquelle le client demande le support du sous-protocole SIP WebSocket au serveur.

```
GET / HTTP/1.1
    Host: sip-ws.example.com
```

Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13

La réponse du serveur acceptant le sous-protocole SIP WebSocket est :

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip

Les messages WebSocket peuvent être transportés dans des trames UTF-8 (dont l 'ASCII est un sous-ensemble) ou des trames binaires. Il est recommandé d 'utiliser l 'encodage UTF-8 pour les messages SIP transportés sur la connexion WebSocket.
WebSocket [RFC6455] est un protocole fiable. Donc, le sous-protocole WebSocket SIP utilise un transport fiable. Les transactions SIP client et serveur utilisant WebSocket doivent utiliser les procédures et les valeurs de temporisateur pour le transport fiable.
Chaque message SIP doit être transporté dans un seul message WebSocket.
Par ailleurs un message WebSocket ne soit pas contenir plus d'un seul message SIP.
Le header Via contient un identificateur de protocole de transport. Les protocoles de transport possibles sont "WS" pour WebSocket ou "WSS » pour Secure WebSocket lorsque la connexion WebSocket est prise en charge sur TLS sur TCP.
Chaque terminaison de la connexion WebSocket doit envoyer périodiquement une trame Ping pour s'assurer que l'autre terminaison est toujours accessible et ainsi maintient la connexion. Une trame Pong doit être retournée en réponse à la trame Ping.

5.1. Enregistrement SIP over WebSocket

Mary télécharge une page Web en utilisant son navigateur et extrait le code JavaScript implantant le sous-protocole SIP WebSocket.
Le code JavaScript (un client SIP WebSocket) établit une connexion WebSocket sécurisée avec un proxy/registrar SIP (un serveur SIP WebSocket) proxy.example.com. Sur la connexion WebSocket, Mary construit et émet une requête SIP REGISTER indiquant le support de Outbound et GRUU (Globally Routable User Agent URI). Parce que la stack Javascript dans un browser n'a pas de moyen de déterminer l 'adresse locale depuis laquelle la connexion WebSocket a été initiéee, l 'implantation utilise un nom de domaine aléatoire suivi par « .invalid » pour le header Via ainsi que pour le header contact.

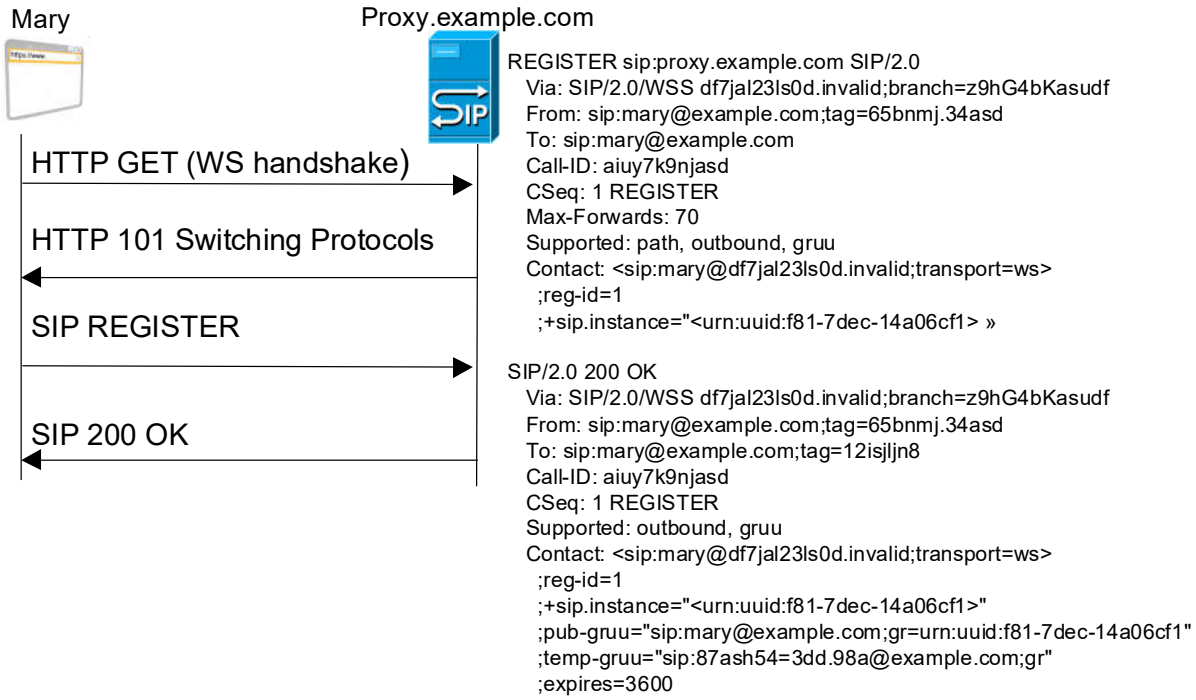

Figure 7 : Enregistrement SIP over WebSocket

5.2. Etablissement/Libération de session SIP over WebSocket

Chaque endpoint SIP est enregistré avec le serveur SIP par un identificateur SIP unique. Il s 'agit de l 'URI SIP, dénoté par le format sip:<username>@<domainname>. Lorsqu'un usager, e.g., Mary, appelle un autre usager, e.g., Mark, via l 'URI de Mark, le serveur SIP WebSocket à Proxy.example.com joue le rôle de nœud proxy SIP et route la requête SIP INVITE à Mark. Mark répond à l'appel afin d'initier une conversation, puis termine l'appel via la requête BYE à la fin de la communication.

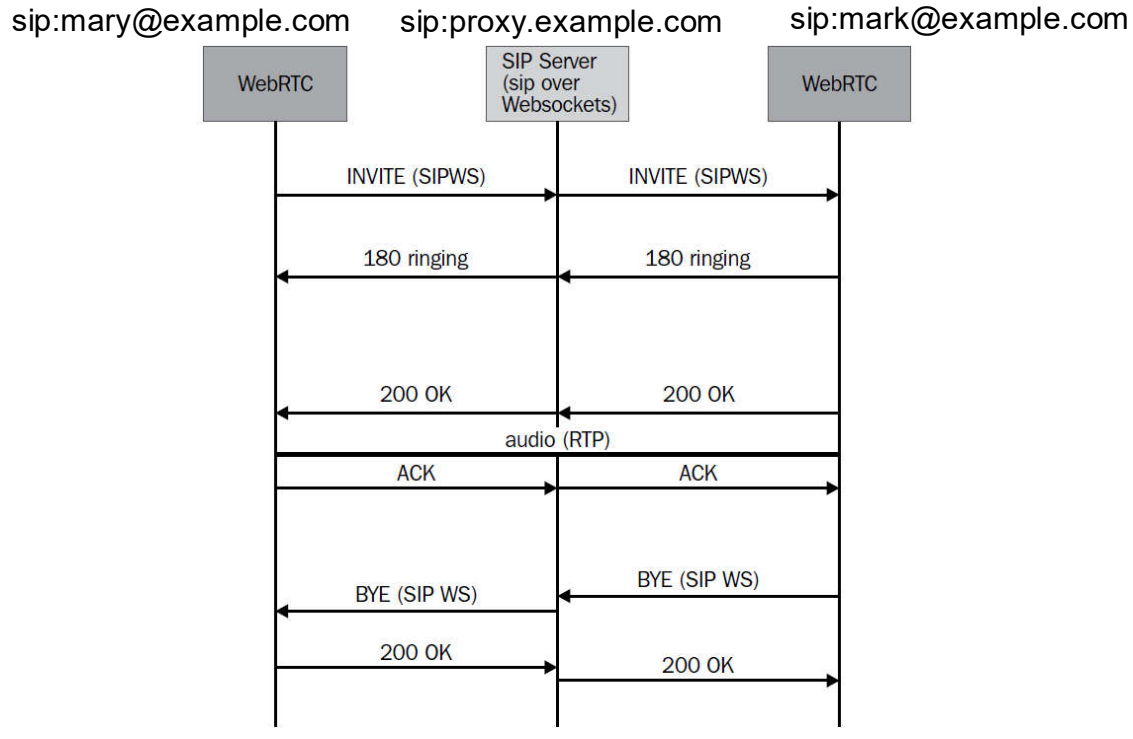

Figure 8 : Etablissement de session SIP over WebSocket

La formation EFORT « WebRTC : Architecture, Plans signalisation et média, service et interfonctionnement avec IMS »permet de comprendre l'architecture WebRTC, les plans signalisation et média, les call flow d'établissement/libération de session et l'interfonctionnement avec IMS.
http://efort.com/index.php?PageID=21&l=fr&f_id=138&imageField.x=4&imageField.y=4