RFC 7118 : The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 29 janvier 2014

Date de publication du RFC : Janvier 2014

https://www.bortzmeyer.org/7118.html

Aujourd'hui, de plus en plus de communications sur l'Internet passent sur le port 80, celui de HTTP, parce que c'est souvent le seul que des réseaux d'accès fermés et mal gérés laissent passer. Le protocole WebSocket a été créé juste pour cela, pour fournir un « nouveau TCP » aux applications, notamment celles tournant dans un navigateur Web. Ce nouveau RFC normalise le transport du protocole de communciation SIP sur WebSocket. Une de ses applications sera le développement de "*softphones*" sous forme d'applications JavaScript. « Téléphonie IP "*over*" Web », en somme.

Le principe de WebSocket, normalisé dans le RFC 6455 [1], est d'ouvrir une connexion HTTP (donc, sur TCP et, éventuellement, sur TLS) puis de la transformer en un canal permanent et bi-directionnel. Sur ce canal passent des messages (et pas un flot d'octets non structuré comme avec TCP). Tous les gros (très gros) navigateurs récents ont déjà un client WebSocket et en permettent l'accès aux applications JavaScript via l'API officielle <http://www.w3.org/TR/websockets/>. Dans le futur, on pourra voir d'autres clients WebSocket, par exemple sous forme d'une bibliothèque pour les programmeurs de "*smartphones*".

Un concept important de WebSocket est celui de **sous-protocole** (section 1.9 du RFC 6455), un protocole applicatif au dessus de WebSocket. Indiqué par l'en-tête `Sec-WebSocket-Protocol:`, ce sous-protocole permet d'indiquer l'application qui va utiliser WebSocket, ici SIP.

Voici donc (section 4 de notre RFC) une demande de transformation d'un canal HTTP ordinaire en WebSocket, pour le sous-protocole SIP :

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6455.txt

```
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
```

Si elle est acceptée par le serveur HTTP, cette demande produira un canal WebSocket de sous-protocole `sip`. Voici une réponse positive du serveur :

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

Le sous-protocole SIP est désormais dans le registre IANA des sous-protocoles WebSocket <https://www.iana.org/assignments/websocket/websocket.xhtml#subprotocol-name>.

WebSocket peut avoir des messages textes ou binaires. SIP peut utiliser des données textes ou binaires (pas dans les en-têtes mais dans le corps des messages SIP). Donc, clients et serveurs SIP-sur-WebSocket doivent gérer les deux types de messages.

WebSocket fournit un transport fiable (section 5) donc SIP n'a pas besoin de prévoir de retransmission. Chaque message SIP se met dans un et un seul message WebSocket et chaque message WebSocket ne contient qu'un seul message SIP. Comme WebSocket a son propre mécanisme d'indication de la longueur, le champ `Content-Length:` de SIP (section 20.14 du RFC 3261) est inutile.

Le champ `Transport` de l'en-tête `Via:` de SIP (section 20.42 du RFC 3261) vaut `WS` (ou `WSS` si TLS est utilisé) lorsque les messages SIP sont transportés sur WebSocket. Le paramètre `received` de cet en-tête `Via:`, qui indique normalement l'adresse IP du client, n'a guère de sens pour WebSocket, où le passage par un relais HTTP sera probablement fréquent. Et il est inutile puisque la réponse peut uniquement être transmise sur la connexion WebSocket. Le RFC 3261, section 18.2.1 est donc modifié pour rendre ce `received` facultatif. Ah, et, sinon, dans un URI SIP, `ws:` ou `wss:` (S pour "*secure*") indique l'utilisation de WebSocket. On verra par exemple `wss://sip.example.com:8443/` (comme dans la documentation de JSSIP <http://jssip.net/documentation/devel/api/ua_configuration_parameters/#parameter_ws_servers>). Pour trouver un serveur SIP WebSocket en utilisant NAPTR (RFC 3263), on met la valeur `SIP+D2W` dans l'enregistrement NAPTR (valeur désormais notée dans le registre IANA <https://www.iana.org/assignments/sip-table/sip-table.xhtml>). Mais c'est purement théorique puisque, malheureusement, le code JavaScript ne peut en général pas faire de requête DNS de type NAPTR (c'est hélas pareil pour les SRV).

La section 8 fournit des nombreux exemples de dialogues SIP sur WebSocket. Par exemple, lorsqu'Alice, utilisant son navigateur Web, charge la page avec le code SIP-sur-WebSocket, elle va commencer par s'enregistrer auprès de son serveur SIP, ici codé en dur comme étant `proxy.example.com`. Elle commence en HTTP :

```
GET / HTTP/1.1
Host: proxy.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: https://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

Et, lorsque la connexion WebSocket est acceptée, elle continue en SIP :

```
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
     ;reg-id=1
     ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
```

Notez le `.invalid` dans l'adresse mise en `Contact:`, parce que le code JavaScript n'a pas de moyen de déterminer l'adresse IP locale. Notez aussi que TLS était utilisé, d'où le `WSS` dans le `Via:`.

Ensuite, Alice appelle l'habituel Bob en envoyant le message SIP :

```
INVITE sip:bob@example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 70
Supported: path, outbound, gruu
Route: <sip:proxy.example.com:443;transport=ws;lr>
Contact: <sip:alice@example.com
    ;gr=urn:uuid:f81-7dec-14a06cf1;ob>
Content-Type: application/sdp
...
```

Le relais `proxy.example.com` va alors router cet appel vers Bob (pas forcément avec WebSocket, dans l'exemple de la section 8 de notre RFC, c'est UDP qui est utilisé) et le relais va dire à Alice que c'est bon :

```
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
     <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
CSeq: 1 INVITE
Contact: <sip:bob@203.0.113.22:5060;transport=udp>
Content-Type: application/sdp
```

Il y a apparemment au moins deux mises en œuvre de ce mécanisme, comme JSSIP <http://jssip.net/>, SIP router <http://sip-router.org/> (c'est documenté <http://sip-router.org/docbook/sip-router/branch/master/modules/websocket/websocket.html>) ou encore OverSIP <http://www.oversip.net/> (voir sa documentation WebSocket <http://www.oversip.net/documentation/misc/sip_websocket/>). L'annexe B du RFC donne des détails pour les programmeurs.