RFC 7395 : An XMPP Sub-protocol for WebSocket

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 novembre 2014

Date de publication du RFC : Octobre 2014

https://www.bortzmeyer.org/7395.html

Le protocole XMPP (surtout connu par son utilisation dans la messagerie instantanée) peut fonctionner directement au-dessus de TCP. Mais pas mal de "*middleboxes*" se permettent de bloquer les protocoles inconnus et leur liste de protocoles connus est bien courte. XMPP a donc une adaptation à HTTP, lui permettant de tourner sur ce protocole. Mais la sémantique de HTTP, requête/réponse, ne convient guère à XMPP et les performances ne sont donc pas géniales. D'où ce nouveau RFC, qui adapte XMPP à WebSocket, lui permettant de tourner sur un protocole peu filtré, avec des meilleurs résultats que HTTP. La technique date de plusieurs années mais elle est maintenant normalisée.

XMPP est normalisé dans les RFC 6120[1] et RFC 6121. L'adaptation à HTTP, connue sous le nom de BOSH ("*Bidirectional-streams Over Synchronous HTTP*") est décrite dans XEP-0124 <http://xmpp.org/extensions/xep-0124.html> et XEP-0206 <http://xmpp.org/extensions/xep-0206.html>. Cette approche a fait l'objet de critiques (RFC 6202) notamment pour ses performances, comparées au XMPP natif (directement sur TCP).

Outre le problème des innombrables boîtiers situés sur le chemin et qui se permettent de bloquer certains protocoles, une limite de ce XMPP natif est liée au modèle « tout dans le navigateur Web ». Ces navigateurs peuvent exécuter du code récupéré sur le Web (code typiquement écrit en JavaScript) mais ce code a des tas de restrictions, notamment pour l'utilisation directe de TCP. Un client de messagerie instantanée écrit en JavaScript a donc du mal à faire du XMPP « normal ». D'où l'utilisation, peu satisfaisante, de HTTP. Mais, plutôt que d'utiliser le HTTP habituel, comme le faisait BOSH, on va utiliser WebSocket (RFC 6455). WebSocket fournit un service simple d'acheminement de messages, bidirectionnel, pour toutes les applications tournant dans un navigateur Web. Ce service est donc proche de celui de TCP (à part qu'il est orienté messages, au lieu d'acheminer un flux d'octets continu). XMPP sur WebSocket sera du XMPP normal, ayant les mêmes capacités que du XMPP sur TCP.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6120.txt

Attention à la terminologie : le mot anglais "*message*" n'a pas le même sens en WebSocket (où il désigne les unités de base de la transmission, équivalent des segments TCP) et en XMPP (où il désigne un type particulier de strophe, celles qui commencent par <message>). Dans ce RFC, « message » a le sens WebSocket.

WebSocket permet de définir des sous-protocoles (qui devraient plutôt être nommés sur-protocoles puisqu'ils fonctionnent au-dessus de WebSocket) pour chaque application. La définition formelle du sous-protocole XMPP est dans la section 3 de notre RFC. Le client doit inclure xmpp dans l'en-tête Sec-WebSocket-Protocol: lors du passage du HTTP ordinaire à WebSocket. Si le serveur renvoit xmpp dans sa réponse, c'est bon (ce xmpp est désormais dans le registre IANA des sous-protocoles WebSocket <https://www.iana.org/assignments/websocket/websocket.xml#subprotocol-name>). Voici un exemple où tout va bien, le serveur est d'accord, le client fait donc du XMPP juste après :

```
Client:
GET /xmpp-websocket HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: xmpp
Sec-WebSocket-Version: 13

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

Client:
<open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
             to="example.com"
             version="1.0" />

Server:
<open xmlns="urn:ietf:params:xml:ns:xmpp-framing"
             from="example.com"
             id="++TR84Sm6A3hnt3Q065SnAbbk3Y="
             xml:lang="en"
             version="1.0" />
```

Chaque message WebSocket doit contenir un et un seul document XML complet (pas question de le répartir sur plusieurs messages), par exemple :

```
<message xmlns="jabber:client" xml:lang="en">
     <body>Every WebSocket message is parsable by itself.</body>
</message>
```

Ce point était un de ceux qui avaient fait l'objet du plus grand nombre de discussions à l'IETF. L'avantage de cette exigence est qu'elle simplifie le travail des programmeurs. Autrement, c'est très proche de ce que fait XMPP sur TCP, à quelques détails près, dans le reste de la section 3.

Du fait de la règle « uniquement des documents XML complets et bien formés », les clients XMPP sur WebSocket ne peuvent plus utiliser le truc traditionnel qui consistait à envoyer des espaces entre les

strophes pour garder la connexion ouverte (section 4.6.1 du RFC 6120). À la place, ils doivent utiliser un mécanisme comme le ping XMPP de XEP 0199 <http://xmpp.org/extensions/xep-0199.html> ou la gestion de flots du XEP 0198 <http://xmpp.org/extensions/xep-0198.html>. Pourquoi ne pas utiliser les services de ping de WebSocket lui-même (section 5.5.2 du RFC 6455) ? On en a le droit mais ce n'est pas obligatoire, en reconnaissance du fait que les navigateurs Web ne donnent en général pas accès à ce service aux programmes JavaScript.

Autre conséquence de cette règle « uniquement des documents XML complets et bien formés », on ne peut pas utiliser du XMPP chiffré avec TLS (cela serait vu comme du binaire, pas comme du XML bien formé). Si on veut sécuriser XMPP sur WebSocket, il faut lancer TLS avant, dans la session WebSocket (URI `wss:`, section 10.6 du RFC 6455).

Mais, au fait, comment un client sait-il qu'un service XMPP donné, mettons `jabber.lqdn.fr`, fournit du XMPP sur WebSocket ou pas ? Cette question fait l'objet de la section 4. L'idéal serait d'avoir l'information dans le DNS, de la même façon que le client XMPP classique découvre, dans le DNS, que pour communiquer avec `bortzmeyer@jabber.lqdn.fr`, il faut passer par le serveur `iota.lqdn.fr` :

```
% dig SRV _xmpp-server._tcp.jabber.lqdn.fr
...
;; ANSWER SECTION:
_xmpp-server._tcp.jabber.lqdn.fr. 3600 IN SRV 0 5 5269 iota.lqdn.fr.
```

Mais les pauvres scripts tournant dans les navigateurs n'ont hélas pas le droit de faire directement des requêtes DNS. Il faut donc une autre méthode d'indirection. La méthode officielle est d'utiliser les métadonnées du RFC 6415, avec les liens du RFC 8288. La relation à utiliser se nomme `urn:xmpp:alt-connections:web` et elle est décrite dans le XEP 0156 <http://xmpp.org/extensions/xep-0156.html>. Pour l'adresse XMPP `bortzmeyer@jabber.lqdn.fr`, on va 1) envoyer une requête HTTP `http://jabber.lqdn.fr/.well-known` 2) chercher dans le document récupéré un lien pour `urn:xmpp:alt-connections:websocket`. Si on trouve (c'est un exemple imaginaire) :

```
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
         <Link rel="urn:xmpp:alt-connections:websocket"
               href="wss://im.example.org:443/ws" />
</XRD>
```

C'est qu'il faut se connecter en WebSocket (avec TLS) à `im.example.org`.

Quelques petits mots sur la sécurité : la méthode de découverte repose sur le DNS, comme pour la méthode originale de XMPP, puisqu'on doit se connecter en HTTP donc faire une requête DNS pour le nom du service XMPP. On peut (et on devrait) sécuriser le tout avec TLS. Mais attention, les applications qui tournent dans un navigateur Web n'ont pas toujours les moyens de faire leur vérification du certificat et notamment la correspondance entre le sujet du certificat et le nom de service souhaité. Si le nom est le même pour WebSocket et XMPP (ce qui n'est **pas** le cas de l'exemple de découverte cité plus haut, et ne sera pas le cas si on sous-traite son service XMPP), le navigateur a déjà fait la vérification lors de la connexion TLS WebSocket et il faut lui faire confiance. Sinon, il faut faire confiance à la délégation citée plus haut (le XRD).

Pour le reste, c'est toujours du XMPP et les mécanismes de sécurité classiques de XMPP, comme l'authentification des utilisateurs avec SASL, suivent les mêmes règles.

Il existe déjà plusieurs mises en œuvre de XMPP sur WebSocket (reflétant la relative ancienneté du protocole). Côté serveur, elles peuvent être dans le serveur lui-même (Node-XMPP-Bosh <https://github.com/dhruvbird/node-xmpp-bosh> qui, en dépit de son nom, fait aussi des WebSockets, un module <https://github.com/superfeedr/ejabberd-websockets> pour ejabberd ou un autre <https://code.google.com/p/openfire-websockets/> pour Openfire) ou bien sous forme d'une passerelle qui reçoit le XMPP sur WebSocket et le transmet au-dessus de TCP vers le vrai serveur (comme celle de Kaazing <http://www.kaazing.com/products/editions/kaazing-websocket> ou wxg <https://github.com/hocken/wxg>). Côté client, il y a les bibliothèques JavaScript Strophe <http://strophe.im/strophejs/>, Stanza.io <https://github.com/otalk/stanza.io> et JSJaC <https://github.com/sstrigler/JSJaC/>. Question documentation, il y a un livre <http://www.amazon.com/dp/0470540710/?tag=stackoverfl08-20> que je n'ai pas lu.