RFC 9297 : HTTP Datagrams and the Capsule Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 septembre 2022

Date de publication du RFC : Août 2022

https://www.bortzmeyer.org/9297.html

Ce RFC est le premier du groupe de travail MASQUE <https://datatracker.ietf.org/wg/masque/> qui développe des techniques pour relayer du trafic IP. Dans ce RFC, il s'agit de faire passer des datagrammes sur HTTP.

Pourquoi, sur HTTP ? Parce que c'est le seul protocole dont on est raisonnablement sûr qu'il passera partout. En bâtissant un service de datagrammes sur HTTP (et non pas directement sur IP), on arrivera à le faire fonctionner à tous les endroits où HTTP fonctionne.

En fait, ce RFC décrit deux mécanismes assez différents, mais qui visent le même but, la transmission de datagrammes sur HTTP. Le premier mécanisme est le plus simple, utiliser HTTP juste pour lancer une session qui utilisera les datagrammes de QUIC <https://www.bortzmeyer.org/quic.html>, normalisés dans le RFC 9221 [1]. (QUIC seul ne passe pas partout, d'où la nécessité d'insérer une couche HTTP.) Ce premier mécanisme dépend de QUIC donc ne marche qu'avec HTTP/3 (RFC 9114). Un second mécanisme est normalisé, dans notre RFC ; nommé **Capsule**, il permet de faire passer des datagrammes (ou à peu près) sur HTTP/2 (RFC 9113) et HTTP/1 (RFC 9112). Capsule est plus général, mais moins efficace.

Ces mécanismes ne sont pas vraiment prévus pour des utilisations par des applications, mais plutôt pour des extensions à HTTP (RFC 9110, section 16). Par exemple, le service CONNECT (RFC 9110, section 9.3.6) pourrait être doublé par un service équivalent, mais utilisant des datagrammes, au lieu du transfert fiable que fait CONNECT (cf. RFC 9298). Même chose pour les Web sockets du RFC 6455.

La section 2 de notre RFC explique de quels datagrammes on parle. Il s'agit de paquets de données dont l'ordre d'arrivée et l'acheminement ne sont pas garantis, et qui vont sans doute consommer moins

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc9221.txt

de ressources. Sur HTTP/3 (qui utilise QUIC), ils vont utiliser les trames QUIC de type `DATAGRAM` (RFC 9221). Ce sont les « meilleurs » datagrammes HTTP, ceux qui collent le plus à la sémantique des datagrammes. Sur HTTP/2, tout acheminement de données est garanti. Si on veut faire des datagrammes, il va falloir utiliser le protocole **Capsule**, décrit dans la section 3. (Notez que HTTP/2 ne garantit pas l'ordre d'arrivée si les datagrammes sont transportés dans des ruisseaux différents.) Et pour HTTP/1 ? Le protocole ne convient guère puisqu'il garantit l'ordre d'arrivée et l'acheminement, justement les propriétés auxquelles on était prêt à renoncer. Là aussi, on se servira de Capsule.

HTTP utilise différentes méthodes pour faire ses requêtes (les deux plus connues sont GET et POST). Les datagrammes ne sont utilisables qu'avec des méthodes qui les acceptent explicitement. Donc, pas de GET ni de POST, plutôt du CONNECT.

Sur HTTP/3, le champ de données du datagramme QUIC aura le format :

```
HTTP/3 Datagram {
     Quarter Stream ID (i),
     HTTP Datagram Payload (..),
   }
```

Le "*quarter stream ID*" est l'identificateur du ruisseau QUIC du client où a été envoyée la requête HTTP, divisé par 4 (ce qu'on peut toujours faire, vu la façon dont sont générés ces identificateurs, RFC 9000, section 2.1).

Ah, et comme les datagrammes ne sont pas acceptés par défaut en HTTP/3, il faudra envoyer au début de la session un paramètre `SETTINGS_H3_DATAGRAM` (enregistré à l'IANA `<https://www.iana.org/assignments/http3-parameters/http3-parameters.xml#http3-parameters-settings>`)

Les capsules, maintenant. Inutiles en HTTP/3, elles sont la seule façon de faire du datagramme en HTTP/1 et HTTP/2. Le protocole Capsule permet d'envoyer des données encodées en TLV sur une connexion HTTP. On indique qu'on va l'utiliser, en HTTP/1 avec le champ `Upgrade:` (RFC 9110, section 7.8) et en HTTP/2 avec un CONNECT étendu (cf. RFC 8441). Les "*upgrade tokens*" (les identificateurs de protocoles utilisés, enregistrés à l'IANA `<https://www.iana.org/assignments/http-upgrade-tokens>`) sont décrits dans la section 16.7 du RFC 9110.

Le format des capsules est décrit en section 3.2 :

```
Capsule {
     Capsule Type (i),
     Capsule Length (i),
     Capsule Value (..),
   }
```

Les différents types possibles de capsules figurent dans un registre IANA `<https://www.iana.org/assignments/http-capsule-protocol/http-capsule-protocol.xml#http-capsule-types>`. Une fois le protocole Capsule sélectionné via un "*upgrade token*", on passe du HTTP classique au protocole Capsule. Un premier type de capsule est DATAGRAM (type 0) dont le nom indique bien la sémantique. D'autres types pourront être ajoutés en suivant la procédure « spécification nécessaire » du

RFC 8126, sauf les valeurs basses du type (de 0 à 63) qui exigent une action de normalisation, ou bien une approbation par l'IESG.

Si on utilise Capsule, la requête HTTP est accompagnée du champ `Capsule-Protocol` (un champ structuré, cf. RFC 9651), champ désormais enregistré à l'IANA <https://www.iana.org/assignments/http-fields/http-fields.xml#field-names>.

Il existe plusieurs mises en œuvre de ces datagramms HTTP. En logiciel libre, on a :
— Quiche <https://github.com/google/quiche>,
— Mvfst <https://github.com/facebookincubator/mvfst>.
Notez que le protocole Capsule n'est a priori pas accessible au code JavaScript chargé dans le navigateur Web (il faut pouvoir accéder aux "*upgrade tokens*"). Mais rappelez-vous que tout ce mécanisme de datagrammes sur HTTP est conçu pour des extensions de HTTP, pas pour l'application finale.