Protocoles basés sur UDP

Juliusz Chroboczek

23 novembre 2021

Dans les chapitres précédents, nous nous sommes intéressés aux protocoles basés sur HTTP, ce qui a plusieurs avantages :

- les ressources HTTP sont identifiées par des URL, ce qui est plus pratique à manipuler que des adresses de *socket* ou même des paires d'un nom de hôte et numéro de port ;
- le protocole traversait les NAT et les *firewalls*, même fascistes, et était facile à cacher ou à faire suivre *(proxy)* ;
- le protocole était sécurisé par TLS, qui est complexe mais disponible partout.

Nous avons payé pour cela un prix élevé :

- impossible de faire des notifications asynchrones ;
- *overhead* important ;
- transfert ordonné et fiable, donc peu adapté au temps réel (VoIP, vidéoconférence, jeux en ligne) ;
- impossible de faire du pair-à-pair, tout le traffic doit passer par le serveur, ce qui peut être coûteux.

Si WebSocket nous a permis de nous libérer des deux premières limitations (notifications asynchrones et *overhead*), il ne résoud pas les autres problèmes. Pour nous en libérer, il faudra utiliser un protocole *ad hoc* basé sur UDP.

1 Protocole hybride

Un protocole basé sur UDP est difficile à déployer et difficile à implémenter. Pour simplifier la tâche, nous combinerons un protocole basé sur UDP à un protocole client-serveur traditionnel, par exemple de type REST. Ce protocole aura les rôles suivants :

- dans le cair pair-à-pair, identifier les pairs par des URL et indiquer leurs adresses de *socket* aux autres pairs ;
- servir de canal sécurisé pour transférer les clés cryptographiques entre les pairs.

Deux structures sont possibles (figure 1). Dans le cas purement client-serveur, le traffic UDP est parallèle au trafic du canal de contrôle. Dans le cas pair-à-pair, le traffic UDP se fait directement entre les pairs, ce qui est plus difficile à implémenter, mais réduit la latence et la charge sur le serveur.

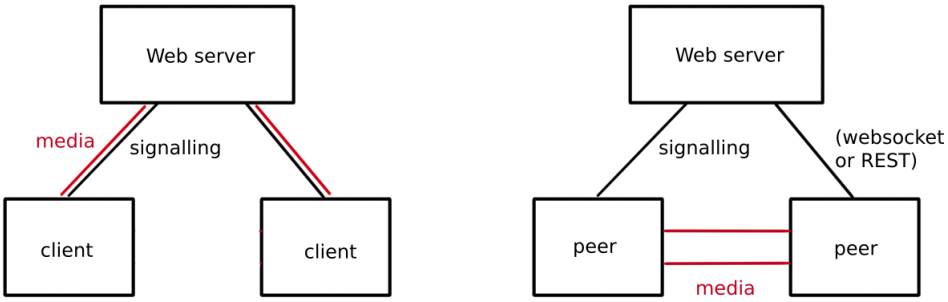

FIGURE 1 — Protocole hybride client-serveur et pair-à-pair

2 Coûts d'UDP

En passant à UDP, nous perdons les fonctionnalités spécifiques à TCP, qu'il nous faudra donc implémenter nous mêmes.

2.1 *Handshake* initial

L'établissement de l'association entre deux pairs UDP s'appelle le *handshake initial*. Ce handshake initial doit obligatoirement consister d'au moins trois paquets : un paquet du pair A vers le pair B, qui indique l'intention de communiquer ; un paquet de B vers A, qui confirme à A que B l'entend et désire communiquer avec lui ; et enfin un deuxième paquet de A vers B, qui indique à B que A l'entend aussi. Ce n'est qu'après que cet échange initial a abouti que l'accessibilité symétrique a été établie, et qu'il est raisonnable de transmettre des quanités conséquentes de données.

Pour éviter la confusion entre paquets, il est important d'étiqueter l'échange initial avec des *nonces* qui identifient les paquets. De plus, pour éviter les attaques par réflexion, il est important que le premier paquet échangé soit au moins aussi volumineux que le deuxième.

2.2 Terminaison d'association et *keepalives*

A priori, il semblerait que la terminaison d'association puisse se faire par un *handshake* de fin. Cependant, un pair peut quitter le réseau sans envoyer aucun paquet : il est donc nécessaire d'implémenter un *timeout* qui supprime une association après qu'on n'a reçu aucun paquet d'un pair donné.

Pour éviter de se faire éliminer, chaque pair devra envoyer des *keepalives* périodiques à chacun des pairs avec lesquels il désire maintenir une association. Comme nous le verrons au cours suivant, ces pairs seront aussi nécessaires pour maintenir les *mappings* dans les NAT et les *firewalls* avec état.

2.3 Réémissions

UDP n'est pas un protocole fiable. Dans certains cas, ce n'est pas problématique : par exemple, dans un jeu en ligne où un pair envoie périodiquement la position de l'avatar du joueur, il n'est pas nécessaire de reémettre la position ; en cas de perte, la mise à jour subséquente mettra la position à jour.

Lorsque la fiabilité est nécessaire, il faut acquitter les transferts et reémettre les paquets. Deux algorithmes sont possibles. Si l'on n'a aucune information sur le pair, l'algorithme du *backoff exponentiel* est un peu lent mais évite de surcharger le réseau. Si l'on peut se permettre de maintenir des statistiques, c'est l'algorithme du RTO qui est généralement employé. Ces deux algorithmes ont été vus en TP.

2.4 Contrôle de la congestion

Si un pair ne maintient qu'un paquet en vol, il ne risque pas de surcharger le réseau. Si par contre il utilise un protocole à fenêtre coulissante qui lui permet de maintenir plusieurs paquets en vol, ce qui est nécessaire pour effectuer des transferts rapides sur des liens à RTT élevé, il faudra implémenter un algorithme de contrôle de congestion.

2.5 Sécurité

Le traffic TCP est protégé par TLS, qui authentifie le serveur au client et chiffre le traffic dans les deux sens. Ce n'est pas le cas du traffic UDP, et il nous faudra donc implémenter notre propre protocole cryptographique.

Cette tâche est simplifiée par la présence du serveur, avec lequel la communication est protégée par TLS. De ce fait, le canal de contrôle pourra être utilisé pour transmettre les clés cryptographiques, soit en clair, soit, ce qui est préférable, en faisant un échange Diffie-Hellman (sur des courbes elliptiques, ou, si l'on est vraiment paranoïaque, utilisant des algorithmes post-quantiques). Ces clés doivent être éphémères : elles ne servent que pendant un temps fini, et doivent être effacées dès qu'elles ne servent plus, afin de garantir la *perfect forward secrecy*.

3 Bénéfices d'UDP

Les coûts décrits ci-dessus peuvent sembler importants, mais l'utilisation d'UDP nous permet de nous libérer des lourdes contraintes imposées par TCP.

3.1 Traffic pair-à-pair

S'il est en principe possible d'implémenter un protocole pair-à-pair au-dessus de TCP, c'est difficile en pratique :

- TCP ne traverse pas facilement les NAT et *firewalls* ;
- les implémentations de TLS demandent un nom de hôte et un certificat associé, ce qui n'est généralement disponible que sur un serveur.

Nous verrons au cours suivant des techniques qui permettent au traffic UDP de traverser les NAT et *firewalls*. Nous verrons en TP comment on peut implémenter un protocole sécurisé pour le trafic UDP.

3.2 Transport non-fiable

UDP est un protocole non-fiable, la fiabilité, si nécessaire, est implémentée par l'application. Cette dernière dispose donc d'une grande liberté, par exemple de mélanger trafic fiable et trafic non-fiable, ou d'abandonner une transmission immédiatement lorsqu'elle n'est plus utile (par exemple parce que l'utilisateur a abandonné la transaction).

3.3 Transport non-ordonné

UDP n'est pas un protocole ordonné, et ne souffre donc pas du blocage de tête de file : un paquet réémis ne bloque pas la livraison des données subséquentes. Dans certaines applications, de tels paquets remis dans le désordre peuvent être utilisés immédiatement, ce qui peut rendre l'application plus responsive.

3.4 Mobilité côté client

Dans la pile de protocoles TCP/IP, l'adresse IP sert à la fois d'identificateur et de locateur : elle indique à la couche transport l'identité d'un pair, et elle indique à la couche réseau la localisation d'un pair. Un épisode de mobilité (par exemple un mobile qui passe d'un lien WiFi à un lien LTE) se traduit par un changement d'adresse IP, ce qui mène à la rupture de toutes les connexions TCP et de la perte des données en tampon.

Un protocole basé sur UDP, s'il est bien conçu, peut être capable de survivre à un changement d'adresse IP du client. Il suffit pour cela que le pair mobile envoie un nouveau *handshake* à chaque fois qu'il change d'adresse. Cependant, implémenté naïvement, un tel protocole permet à n'importe quel pair de se faire passer pour un autre. Pour éviter cela, il faut soit utiliser un identificateur de session, gardé secret, qui permet d'authentifier la nouvelle adresse IP, ou alors de protéger le traffic par des signatures cryptographiques.