UDP et serveurs de contrôle

Juliusz Chroboczek

30 novembre 2022

1 Limitations de TCP

Dans la première partie de ce cours, nous avons étudié la conception de protocoles implémentés au-dessus de HTTP(S), et donc, indirectement, au-dessus de TCP. Or TCP a deux limitations : il est obligatoirement *fiable* et *ordonné*, ce qui le rend indadapté aux applications en temps réel, et il n'est pas capable de traverser les NAT, ce qui le rend inadapté aux applications pair-à-pair.

1.1 Applications temps réel

Une *application temps réel* recquiert que les données arrivent au destinataire après un temps borné. Comme on ne peut pas tout avoir, les applications temps réel sont conçues pour être capables de survivre à la perte occasionnelle d'une donnée.

Considérons par exemple un jeu compétitif en ligne. L'émetteur envoie périodiquement (par exemple 10 fois par seconde) la position de l'avatar du joueur, ce qui permet au récepteur de l'afficher au bon endroit sur l'arène du jeu. Si un échantillon est perdu, le récepteur interpole la position, il n'est pas nécessaire de reémettre l'ancienne position maintenant obsolète.

Il en est de même d'une application de voix sur IP. Si un échantillon de voix est perdu, le récepteur le reconstruit, soit en interpolant entre les échatillons adjacents soit en utilisant l'information de correction d'erreurs que l'émetteur entrelace avec les données. Il n'est pas nécessaire de reémettre l'échantillon perdu, maintenant obsolète.

TCP est fiable et ordonné TCP est fiable : si un segment est perdu, il est reémis jusqu'à être acquitté par le récepteur. TCP est aussi ordonné : si un segment a du retard, tous les segments qui suivent sont retardés jusqu'à ce que le segment perdu soit reçu. En particulier, si un segment est perdu, tous les segments subséquents sont retardés jusqu'après la réémission de ce dernier. C'est donc la combinaison de ces deux caractéristiques (et non, comme on le croit souvent, seulement la fiabilité) qui rend TCP inadapté au traffic en temps réel.

1.2 Protocoles pair-à-pair

Les protocoles que nous avons vus dans la première partie de ce cours étaient strictement client-serveur : plusieurs clients se connectent à un ou plusieurs serveurs. Dans un protocole pair-à-pair,

les pairs (clients) communiquent directement entre eux, sans passer par le serveur, ce qui réduit la charge sur le serveur (qu'il faut payer) et minimise la latence du transfert de données (qui suit une route plus directe, sans passer par le serveur).

TCP est inadapté au pair-à-pair TCP est principalement prévu pour les applications client-serveur, et n'est capable de traverser les NAT que dans le « bon » sens. Cela le rend inadapté au traffic pair-à-pair dans l'Internet actuel, où entre deux pairs se trouvent généralement deux NAT, orientés dans des sens opposés.

TLS est inadapté au pair-a-pair Si TLS supporte plusieurs formes d'authentification, le profil de TLS utilisé dans HTTPS ne supporte que l'authentification par certificat serveur. Comme il est impossible d'associer un certificat à une adresse IP, cela rend TLS (tel qu'il est utilisé dans HTTP) difficile ou impossible à utiliser dans des applications pair-à-pair.

2 Rappel : UDP

UDP est un protocole de couche transport qui fournit un service minimal au-dessus de IP : essentiellement, UDP n'ajoute que les numéros de port à IP. UDP implémente donc essentiellement le même service que IP :

- communication par datagrammes (paquets) de taille limitée ;
- communication non-fiable et non-ordonnée ;
- pas de contrôle de flot ou de congestion.

Une application basée sur UDP doit donc être capable de :

- se limiter à des paquets de taille limitée, soit en structurant le protocole pour n'utiliser que de courts messages, soit en découpant les messages en fragments ;
- compenser pour les pertes et les reordonnancements des paquets, soit en numérotant les messages et les remettant dans l'ordre, soit en étant capable de survivre à la perte d'un paquet ou à l'arrivée dans le désordre ;
- limiter le débit des paquets émis à un taux que le récepteur et le réseau peuvent accepter.

De plus, une application basée sur UDP devra résoudre elle-même deux problèmes difficiles :

- la localisation des pairs : il n'y a pas d'URL en UDP, il faudra arriver à déterminer les adresses de socket (IP et numéro de port) des pairs ;
- la sécurité des échanges, soit en utilisant DTLS (la variante datagramme de TLS) soit en implémentant des mécanismes cryptographiques directement dans l'application.

3 Serveur de contrôle

Les protocoles par-à-pair purs sont difficiles à implémenter et difficiles à débogguer. Pour cette raison, la plupart des protocoles pour les applications temps réel ou pair-à-pair utilisent un serveur de contrôle : ils utilisent un canal de données pair-à-pair implémenté au-dessus de UDP, et un canal de contrôle implémenté à travers un serveur, utilisant par exemple HTTPS ou *WebSocket* au-dessus de TLS.

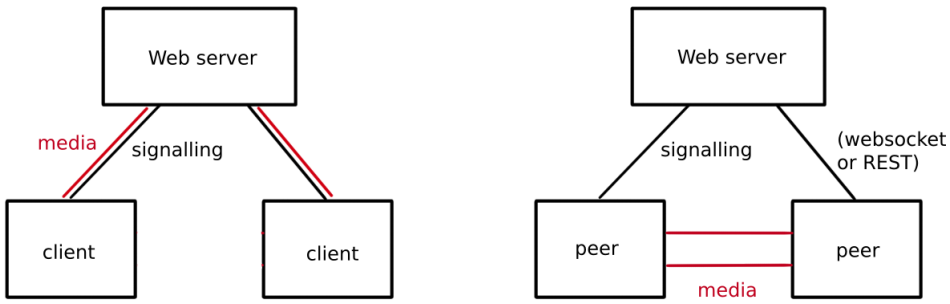

FIGURE 1 — Protocole hybride client-serveur et pair-à-pair

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 réduit la latence et la charge sur le serveur, mais rend le protocole plus difficile à implémenter et surtout à débogguer.

Le canal de contrôle a notamment les rôles suivants :

- localisation : le serveur sert d'annuaire pour indiquer les adresses de socket où envoyer les paquets UDP ;
- *bootstrap* de la sécurité : comme le canal de contrôle est protégé par TLS, il peut servir de canal sécurisé pour l'échange de *tokens* de sécurité ou de clés cryptographiques ;
- traversée de NAT ;
- gestion des sessions, le serveur peut indiquer quand un client donné a quitté l'application.

Comme la traversée de NAT n'est pas fiable, les protocoles pair-à-pair de production sont capables de gérer gracieusement l'échec de la communication directe entre pairs en se rabattant sur une structure client-serveur : le protocole reste essentiellement le même, mais les données transitent toutes à travers le serveur. La latence augmente, mais uniquement pour les utilisateurs coincés derrière un « mauvais » NAT. Quant à l'augmentation de la charge du serveur, elle ne concerne que le traffic d'une petite fraction des utilisateurs.