RFC 5766 : Traversal Using Relays around NAT (TURN): Relay Extensions to SessionTraversal Utilities for NAT (STUN)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 30 avril 2010

Date de publication du RFC : Avril 2010

https://www.bortzmeyer.org/5766.html

Le protocole TURN, que décrit notre RFC, est le **dernier recours** des applications coincées derrière un routeur NAT et qui souhaitent communiquer avec une application dans la même situation (cf. RFC 5128 [1]). Avec TURN, le serveur STUN ne se contente pas d'informer sur l'existence du NAT et ses caractéristiques, il relaie chaque paquet de données. (Depuis la parution de ce RFC, une nouvelle norme TURN est sortie, dans le RFC 8656.)

Être situé derrière un routeur NAT n'est jamais une situation enviable. De nombreuses applications fonctionnent mal ou pas du tout dans ce contexte. Le groupe de travail IETF behave <https://www.bortzmeyer.org/behave-wg.html> a pour mission de spécifier des mécanismes permettant de faire fonctionner le plus d'applications possibles à travers un NAT. Le socle de tous ces mécanismes est STUN (RFC 5389), où le client STUN (notre machine bloquée par le NAT) communique avec un serveur STUN situé dans le monde libre pour apprendre sa propre adresse IP externe. Outre cette tâche de base, des extensions à STUN permettent d'aider davantage le client, c'est par exemple le cas de TURN que normalise notre RFC.

L'idée de base est que deux machines Héloïse et Abélard, chacune peut-être située derrière un NAT, vont utiliser STUN pour découvrir s'il y a un NAT entre elles (sinon, la communication peut se faire normalement) et, s'il y a un NAT, s'il se comporte « bien » (tel que défini dans les RFC 4787 et RFC 5382). Dans ce dernier cas, STUN seul peut suffire, en informant les machines de leur adresse extérieure et en ouvrant, par effet de bord, un petit trou dans le routeur pour permettre aux paquets entrants de passer.

Mais, parfois, le NAT ne se comporte pas bien et il n'existe aucune solution permettant le transport direct des données entre Héloïse et Abélard. La solution utilisée par tous les systèmes pair-à-pair, par exemple en téléphonie sur Internet, est de passer par un **relais**. Normalement, le serveur STUN ne

sert qu'à un petit nombre de paquets, ceux de la **signalisation** et les données elles-mêmes (ce qui, en téléphonie ou en vidéo sur IP, peut représenter un très gros volume) vont directement entre Héloïse et Abélard. Avec TURN, le serveur STUN devient un relais, qui transmet les paquets de données.

TURN représente donc une charge beaucoup plus lourde pour le serveur, et c'est pour cela que cette option est restreinte au dernier recours, au cas où on ne peut pas faire autrement. Ainsi, un protocole comme ICE (RFC 8445) donnera systématiquement la préférence la plus basse à TURN. De même, le serveur TURN procédera toujours à une authentification de son client, car il ne peut pas accepter d'assurer un tel travail pour des inconnus (l'authentification - fondée sur celle de STUN, RFC 5389, section 10.2 - et ses raisons sont détaillées dans la section 4). Il n'y aura donc sans doute jamais de serveur TURN public.

TURN est défini comme une extension de STUN, avec de nouvelles méthodes et de nouveaux attributs. Le client TURN envoie donc une requête STUN, avec une méthode d'un type nouveau, `Allocate`, pour demander au serveur de se tenir prêt à relayer (les détails du mécanisme d'allocation figurent dans les sections 5 et 6). Le client est enregistré par son **adresse de transport** (adresse IP publique et port). Par exemple, si Héloise a pour adresse de transport locale `10.1.1.2:17240` (adresse IP du RFC 1918 et port n° 17240), et que le NAT réécrit cela en `192.0.2.1:7000`, le serveur TURN (mettons qu'il écoute en `192.0.2.15`) va, en réponse à la requête `Allocate`, lui allouer, par exemple, `192.0.2.15:9000` et c'est cette dernière adresse qu'Héloïse va devoir transmettre à Abélard pour qu'il lui envoie des paquets, par exemple RTP. Ces paquets arriveront donc au serveur TURN, qui les renverra à `192.0.2.1:7000`, le routeur NAT les transmettant ensuite à `10.1.1.2:17240` (la transmission des données fait l'objet des sections 10 et 11). Pour apprendre l'adresse externe du pair, on utilise ICE ou bien un protocole de « rendez-vous » spécifique.

Ah, et comment le serveur TURN a t-il choisi le port 9000 ? La section 6.2 détaille les pièges à éviter, notamment pour limiter le risque de collision avec un autre processus sur la même machine.

TURN ne fonctionne que sur IPv4, car c'est pour ce protocole que des NAT sont présents. (Une extension pour relayer vers IPv6, afin de faciliter éventuellement la transition entre les deux familles, a été normalisée dans le RFC 6156.) TURN peut relayer de l'UDP ou du TCP (section 2.1) mais le serveur TURN enverra toujours de l'UDP en sortie (une extension existe pour utiliser TCP en sortie, RFC 6062). La section 2.1 explique aussi pourquoi accepter du TCP en entrée quand seul UDP peut être transmis : la principale raison est l'existence de coupe-feux qui ne laiseraient sortir que TCP.

Les données peuvent circuler dans des messages STUN classiques (nommés `Send` et `Data`, cf. section 10) ou bien dans des canaux virtuels ("*channels*", sortes de sous-connexions, section 11) qui permettent d'éviter de transmettre les en-têtes STUN à chaque envoi de données.

Enfin, la section 17, très détaillée, note également que TURN ne peut pas être utilisé pour contourner la politique de sécurité : l'allocation ne se fait que pour une adresse IP d'un correspondant particulier (Abélard), TURN ne permet pas à Héloïse de faire tourner un serveur. Ce point permet de rassurer les administrateurs de coupe-feux et de leur demander de ne pas bloquer TURN.

Autre point important de cette section sur la sécurité : comme certains messages ne sont pas authentifiés, un méchant peut toujours envoyer au client des messages qui semblent venir du serveur et

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5128.txt`

réciproquement. Le problème existe, mais c'est un problème plus général d'IP. TURN ne le résout pas mais ne l'aggrave pas (section 17.1.4).

Les implémenteurs seront sans doute intéressés par la section 12, qui explique comment traiter les en-têtes IP (voir aussi la 2.6). TURN travaille dans la couche 7, il n'est pas un « routeur virtuel » mais un relais applicatif. En conséquence, il ne préserve pas forcément des en-têtes comme ECN ou comme le TTL. Cela permet son déploiement sur des systèmes d'exploitation quelconque, où il n'est pas forcément facile <https://www.bortzmeyer.org/ip-header-set.html> de changer ces en-têtes. Pour la même raison, TURN ne relaie pas l'ICMP et la découverte traditionnelle de MTU (RFC 1191) à travers TURN ne marche donc pas.

À propos d'implémentations, il existe au moins une mise en œuvre libre de TURN, turnserver <http://turnserver.sourceforge.net/>.

TURN a mis des années et des années à être publié. Certains débats, "*anycast*" pour trouver un serveur, cf. section 2.9, le fait de ne relayer qu'UDP et qu'IPv4, les allocations "*non-preserving*", c'est-à-dire qui ne garantissent pas le respect des en-têtes IP comme ceux de DSCP, tous ces sujets ont été l'occasion de longs débats...