# RFC 8656 : Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 février 2020

Date de publication du RFC : Février 2020

https://www.bortzmeyer.org/8656.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. Ce nouveau RFC remplace la définition originelle de TURN (dans le RFC 5766), et inclut plusieurs changements qui étaient jusqu'à présent spécifiés dans des RFC séparés, comme IPv6 ou DTLS, et ajoute la possibilité de relayer les messages ICMP.

Être situé derrière un routeur NAT n'est jamais une situation enviable. De nombreuses applications fonctionnent mal ou pas du tout dans ce contexte, nécessitant des mécanismes spécifique de traversée des NAT. Le socle de tous ces mécanismes de traversée est STUN (RFC 8489), 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, par exemple parce qu'il a le comportement « "*address-dependent mapping*" » (RFC 4787, section 4.1). Dans ce cas, 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 8489, section 9.2 - et ses raisons sont détaillées dans la section 5). Il n'y aura donc sans doute jamais de serveur TURN complètement public mais certains services sont quand même disponibles comme celui de Viagénie <https://numb.viagenie.ca/>.

TURN est défini comme une extension de STUN, avec de nouvelles méthodes et de nouveaux attributs (enregistrés à l'IANA <https://www.iana.org/assignments/stun-parameters/stun-parameters.xml#stun-parameters-2>). 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 6 et 7). 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 11 et 12). Pour apprendre l'adresse externe du pair, on utilise ICE ou bien un protocole de « rendez-vous » spécifique (RFC 5128.) Un exemple très détaillé d'une connexion faite avec TURN figure dans la section 20 du RFC.

Ah, et comment le serveur TURN a-t-il choisi le port 9000 ? La section 7.2 détaille les pièges à éviter, notamment pour limiter le risque de collision avec un autre processus sur la même machine, et pour éviter d'utiliser des numéros de port prévisibles par un éventuel attaquant (cf. RFC 6056).

Et pour trouver le serveur TURN à utiliser ? Il peut être marqué en dur dans l'application, mais il y a aussi la solution de découverte du RFC 8155. Pour noter l'adresse d'un serveur TURN, on peut utiliser les plans d'URI `turn:` et `turns:` du RFC 7065.

TURN fonctionne sur IPv4 et IPv6. Lorsqu'IPv4 et IPv6 sont possibles, le serveur TURN doit utiliser l'algorithme du RFC 8305, pour trouver le plus rapidement possible un chemin qui fonctionne.

TURN peut relayer de l'UDP, du TCP (section 2.1), du TLS ou du DTLS, mais le serveur TURN enverra toujours de l'UDP en sortie (une extension existe pour utiliser TCP en sortie, RFC 6062) La section 3.1 explique aussi pourquoi accepter du TCP en entrée quand seul UDP peut être transmis : la principale raison est l'existence de pare-feux qui ne laisseraient sortir que TCP. La meilleure solution, recommandée par le RFC, serait quand même qu'on laisse passer l'UDP, notamment pour WebRTC (RFC 7478, section 2.3.5.1 et RFC 8827.)

Ah, et à propos d'UDP, notre RFC recommande d'éviter la fragmentation, et donc d'envoyer vers le pair qui reçoit les données des paquets UDP suffisamment petits pour ne pas avoir besoin de la fragmentation, celle-ci n'étant pas toujours bien gérée par les "*middleboxes*".

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

Les données peuvent circuler dans des messages STUN classiques (nommés `Send` et `Data`, cf. section 11) ou bien dans des canaux virtuels ("*channels*", sortes de sous-connexions, section 12) qui permettent d'éviter de transmettre les en-têtes STUN à chaque envoi de données.

Enfin, la section 21, 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 pare-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 réciproquement. Le problème existe, mais c'est un problème plus général d'IP. TURN ne le résout pas mais ne l'aggrave pas (section 21.1.4). Pour le cas du multimédia, on peut par exemple utiliser SRTP (RFC 3711) pour empêcher cette attaque.

Comme TURN relaie les paquets, au lieu de simplement les router, l'adresse IP source va identifier le serveur TURN et pas le vrai expéditeur. Un méchant pourrait donc être tenté d'utiliser TURN pour se cacher. Il peut donc être utile que le serveur TURN enregistre les données comme l'adresse IP et le port de ses clients, pour permettre des analyses a posteriori.

Une question intéressante est celle du traitement des en-têtes IP (voir la 3.6). 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. D'ailleurs, un serveur TURN tournant comme une application sans privilèges particuliers n'a pas forcément accès aux valeurs de l'en-tête IP. Tout 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, la découverte traditionnelle de MTU (RFC 1191) à travers TURN ne marche donc pas.

À propos d'implémentations, il existe plusieurs mises en œuvre libres de TURN, comme turnserver <http://www.turnserver.org/>, CoTurn <https://github.com/coturn/coturn>, Restund <http://www.creytiv.com/restund.html> ou Pion <https://github.com/pion/turn>.

Les changements depuis les RFC précédents, notamment le RFC 5766, sont résumés dans les sections 24 et 25 :

— Notre RFC intègre le RFC 6156, qui ajoutait IPv6 à TURN. Le client peut spécifier la version d'IP souhaitée, avec le nouvel attribut `REQUESTED-ADDRESS-FAMILY` et il y a de nouveaux codes d'erreur comme 440 ou 443 <https://www.iana.org/assignments/stun-parameters/stun-parameters.xml#stun-parameters-6>,
— TURN peut maintenant accepter du DTLS en entrée,
— La découverte du serveur (RFC 8155) et les plans d'URI `turn:` et `turns:` (RFC 7065) ont été intégrés,
— TURN relaie désormais l'ICMP.