Protocoles des Services Internet VI
BROUILLON

Juliusz Chroboczek

28 novembre 2021

1 *Middleboxes*

En M1, nous avons vu que les couches 4 et 7 étaient *de bout en bout* : les routeurs n'interprètent les paquets que jusqu'à la couche trois, les couches 4 et 7 ne sont implémentées que dans les hôtes, qui se trouvent aux bords du réseau. Cette propriétée, appelée parrois le *end-to-end principle*, a deux conséquences importantes :

- elle rend l'Internet flexible : pour déployer une nouvelle application, il suffit de mettre à jour les hôtes qui participent à celle-ci (comparez le déploiement de *Skype* en 2003-2004 au déploiement des nouveaux services sur le réseau téléphonique) ;
- elle rend l'Internet robuste et efficace : les routeurs ne contiennent pas d'état « dur », ils peuvent donc rebooter à tout moment sans causer de perturbations majeures.

Dans l'Internet d'aujourd'hui, il existe de nombreuses entités se trouvant à l'intérieur du réseau et qui interprètent les données de couche supérieure ; ce sont les *middleboxes*. La plupart des *middleboxes* contiennenet de l'état « dur », c'est à dire qui interfère avec l'application lorsque le *middlebox* reboote, par exemple en interrompant les connections en cours.

Digression : IPv6 Nous croyions jadis que le déploiement d'IPv6, qui ne recquiert pas de recourir à des NAT, résoudrait le problème des *middleboxes*. Nous savons aujourd'hui qu'il n'en est rien : les opérateurs ont déployé des *firewalls* avec état dans les routeurs IPv6, et ce n'est qu'une question de temps avant que quelqu'un décide de résoudre un problème de routage IPv6 en déployant des NAT.

1.1 *Firewalls* avec état

Un *firewall* (parfois *pare-feu*) est un *middlebox* qui autorise le passage d'un paquet selon certains critères, par exemple l'adresse IP, le numéro de port, ou l'état simulé d'une connexion IP. Il existe deux catégories de *firewalls* :

- les *firewalls sans état*, qui décident de laisser passer un paquet ou pas en se basant uniquement sur les données contenues dans ce paquet ;

– les *firewalls avec état*, qui utilisent l'historique des paquets précédents pour décider d'autoriser un paquet ou non.

Typiquement, un *firewall* avec état divise l'Internet en deux parties, l'intérieur (qu'il faut protéger) et l'extérieur (qui est supposé source d'attaques), et n'autorise le traffic que depuis des clients situés à l'intérieur vers des serveurs situés à l'extérieur. Pour ce faire, il mantient une table de connexions ouvertes (indexées par des paires d'adresses de *socket*) ; une connexion ouverte est créée lorsqu'un paquet SYN traverse le *firewall* depuis l'intérieur vers l'extérieur.

1.2 NAT

1.2.1 Types de *mapping*

1.2.2 Types de filtrage

1.2.3 CGNAT

1.3 Accélérateurs

Les protocoles de couche transport et application ont des performances qui dépendent du RTT de la route employée. Par exemple, l'algorithme *Slow Start* de TCP double le débit une fois par RTT, et ne converge vers le débit optimal que lorsque le taux de pertes est inférieur à un paquet par RTT. Pire encore, certaines versions du protocole SMB (le protocole de partage de disques de *Windows*) exhibent des performances qui sont inversément proportionnelles au RTT (elles utilisent une fenêtre de taille fixée).

Ces protocoles sont généralement adaptés à des RTT typiques, de l'ordre de 20 ms. Cependant, les liaisons Internet utilisation des laisions par satellite en orbite géosynchrone exhibent des RTT importants, de l'ordre de plusieurs centaines de millisecondes. (Cela ne s'applique pas aux setallites en orbite basse, qui ont des RTT excellents mais sont beaucoup plus coûteux.)

Les fournisseurs d'Internet par satellite géosynchrone installent souvent des deux côtés du lien des *accelérateurs TCP* et des *accélérateurs d'application*, des *middleboxes* qui terminent localement les connexions TCP ou les sessions SMB. La communication à travers le lien satellite a alors lieu entre les deux *middleboxes*, qui emploient un protocole propriétaire adapté au lien utilisé.

La plupart de ces accélérateurs sont bien faits, ils n'interfèrent pas avec le traffic qu'ils ne comprennent pas. Certains cependant sont sujets à des bugs, qui font qu'ils ne laissent pas passer certains types de traffic.

1.4 *Black boxes*

Depuis Snowden, nous savons que les services de sécurité observent le traffic Internet. Pour cela, ils installent chez les opérateurs des *black boxes*, des périphériques qui capturent une copie du traffic et l'analysent (par exemple pour établir le graphe social des utilisateurs dans le but de prévoir les complots terroristes) ou le stockent pour être analysé plus tard. Peu d'informations sont disponibles à propos des *black boxes*, mais il semblerait que ce ne soient pas des *middleboxes* : ils n'interfèrent pas avec le transfert des données, et se limitent à l'observer.

Digression : le grand *firewall* de Chine Le déploiement à grande échelle de *black boxes* le plus médiatisé est sans doute celui de la République Populaire de Chine, parfois appelé « le grand *firewall* de Chine » (GCF). Les *blackboxes* du GCF ne sont pas des *middleboxes*, ils observent passivement le traffic et ne le coupent que lorsqu'un traffic suspect est détecté. En particulier, ils ont tendance à couper toute connexion chiffrée, même légale, au bout de quelques minutes, ce qui cause de sérieux problèmes pour les transferts de données (par exemple pour envoyer les *firmwares* des objets électroniques dans les usines chinoises).

2 Problèmes liés aux *middleboxes*

Le déploiement massif des *middleboxes* a plusieurs conséquences néfastes sur les applications Internet.

2.1 Ossification de l'Internet

La plupart des *middleboxes* ne comprennent que le traffic TCP et UDP, et jettent le traffic qui emploie d'autres protocoles de couche transport. La solution à ce problème consiste à nous limiter à TCP et UDP, et à structurer les nouveaux protocoles de transport au-dessus d'UDP (voyez la discussion de QUIC dans un cours précédent).

Certains *middleboxes* interfèrent avec les fonctionnalités optionnelles des protocoles qu'ils interprètent. Par exemple, beaucoup de NAT réagissent mal aux options TCP. De ce fait, une application robuste devra être capable de gérer le cas où une fonctionnalité optionnelle ne traverse pas un *middlebox*. Par exemple, le protocole MP-TCP, qui implémente une extension *multipath* à TCP, est capable de se rabattre sur TCP tout bête lorsqu'il détecte que les options *multipath* ne sont pas transmises correctement.

2.2 Instabilité due à l'état

Un réseau à commutation de paquets pur est robuste : le plantage d'un routeur se traduit par une interruption de traffic que quelques secondes, le temps que le protocole de routage contourne la panne ou que le routeur en faute reboote. En présence de *middleboxes* avec état, les connections qui traversaient le *middlebox* sont interrompues en cas de panne.

Une application robuste doit donc surveiller toutes ses connections, par exemple en envoyant des *keepalives* périodiques, et être capable de rétablir la connexion en cas de panne.

2.3 Expiration des entrées

Les entrées dans un *middlebox* avec état ont un temps de vie limité. Par exemple, les NAT maintiennent les entrées UDP pendant deux minutes (selon la norme, moins en pratique), et les entrées TCP pendant deux heures (selon la norme).

Une application conçue pour traverser les *middleboxes* devra envoyer des *keepalives* périodiques pour maintenir les entrées.

2.4 Connectivité asymétrique

Les NAT et les *firewalls* avec état sont asymétriques : tout traffic doit être initié par un client se trouvant à l'intérieur vers un serveur se trouvant à l'extérieur. Si le hôte qui veut initier la connexion se trouve à l'extérieur, il est nécessaire d'utiliser une tierce partie pour s'arranger pour que la communication soit initiée à l'intérieur : c'est le problème de la *traversée de NAT* our plus généralement la *traversée de middleboxes*, dont nous traitons en plus de détail dans la partie 3 ci-dessous.

La traversée de NAT est particulièrement difficile dans le cas de la communication pair-à-pair, où les deux pairs se trouvent probablement « à l'intérieur » du point de vue de deux *middleboxes* distincts.

2.5 Traduction d'adresses

En plus d'interférer avec le traffic légitime, les NAT traduisent les adresses des paquets qui les traversent. Par conséquent, en présence de NAT les adresses IP ne sont plus utiles pour identifier un hôte :

- l'adresse « intérieure » que voit le hôte est différente de l'adresse « externe » que voient ses correspondants ;
- une seule adresse externe peut servir pour tout un ensemble de hôtes ;
- l'adresse externe peut occasionnellement changer (par exemple parce que le NAT a rebooté ou parce que le hôte a changé de routeur sans fil).

Un protocole capable de traverser les NAT devra donc identifier les pairs par des idenficateurs distincts des adresses IP, par exemple des identificateurs de session tirés alétoirement ou même des clés cryptographiques de session.

3 Traversée de *middleboxes*

Pour survivre dans l'Internet actuel, une application doit implémenter des techniques de traversée des *middleboxes*, notamment des NAT.

3.1 Communication client-serveur

Le cas le plus simple est celui d'un protocole client-serveur, où le client se trouve à l'intérieur d'un NAT ou *firewall* et le serveur à l'extérieur. Le problème principal dans ce cas est le temps d'expiration des antrées du *middlebox* (paragraphe 2.3), qui fait qu'une connexion oisive risque d'être interrompue.

Il y a deux solutions à ce problème : on peut soit envoyer des *keepalives* périodiques sur la connexion (ce sera par exemple le cas sur une connection *WebSOcket*), ou, ce qui est moins efficace, se limiter à des connexions de courte durée qu'on relance à chaque requête (ce serait le cas d'une connexion REST basée sur HTTP/1.0).

3.2 *Port forwarding*

Dans le cas d'une communication où le serveur se trouve à l'intérieur du NAT ou *firewall*, le *middlebox* empêche l'établissement de connexion. Dans la cas d'une connection pair-à-pair, les deux pairs sont généralement derrière des NAT, ce qui fait qu'aucun des deux pairs ne peut initier la communication.

Une solution manuelle au problème consiste à établir manuellement des entrées dans le *middlebox* : c'est le *port forwarding*. L'utilisateur humain configure manuellement son NAT ou *firewall*, typiquement à l'aide d'une interface *web* où il rentre les numéros de port et les adresses auxquelles il faut envoyer les paquets.

Si le *port forwarding* résout le problème dans certains cas, il demande une intervention manuelle de la part de l'utiliseteur, ce qui n'est généralement pas acceptable. Dans le cas d'un CGNAT (paragraphe 1.2.3), l'utilisateur n'a pas la main sur le NAT, et il ne peut donc tout simplement pas effectuer de *port forwarding* manuel.

***Port forwarding* automatique : uPNP et NAT-PMP** uPNP (dû à Microsoft) et NAT-PMP (dû à Apple) sont des protocoles qui permettent à l'application de demander un *port forwardng* automatique. uPNP est complexe et mal conçu, car il permet à l'application d'insteller des antrées pour une durée indéterminée et qui ne seront donc pas forcément nettoyés lorsque l'application termine (pensez à un plantage). NAT-PMP est beaucoup plus simple, et limite la durée des entrées installées, que l'application doit renouveler périodiquement, et qui seront donc nettoyés au bout d'un temps broné si l'application disparaît ; il est à préférer lorsqu'il est disponible.

Malheureusement, uPNP et NAT-PMP ne sont pas disponibles sur les CGNAT, et ne fonctionnent de toute façon pas très bien lorsque le hôte se trouve derrière plusieurs couches de NAT. Pour ces raisons, une application robuste doit implémenter les techniques de traversée de NAT décrites ci-dessous même si elle est capable de parler uPNP ou NAT-PMP.

uPNP n'a jamais été normalisé. NAT-PMP a été décrit dans un document de l'IETF, mais il est considéré obsolescent, et doit à terme être remplacé par PCP, un protocole basé sur les idées de NAT-PMP mais supportant IPv6 et capable en principe de traverser plusieurs couches de NAT. Je ne l'ai jamais vu dans la nature.

3.3 Ouverture simultanée et STUN

Si deux hôtes sont derrière des NAT, il leur est impossible de se parler directement, ce qui rend la communication pair-à-pair impossible. Cependant, s'ils sont tous deux connectés au même serveur, ils peuvent utiliser ce dernier pour se synchroniser et ouvrir des connections simultanément.

Deux types de serveurs sont en jeu, le serveur d'adresses (pas forcément le même pour tous les pairs) et le serveur d'application (auquel tous les pairs sont connectés). Le pair *A* contacte d'abord le serveur d'adresses pour obtenir l'adresse IP et le numéro de port externes que le NAT lui a affectés. Il contacte le serveur d'application pour lui demander de faire suivre ces informations au pair *B*. *B* obtient alors son adresse auprès de son serveur d'adresses, et la communique à *A* à travers le serveur d'application. Une fois l'échange d'adresses complété, les deux pairs se connectent simultanément l'un à l'autre, établissant ainsi des entrées dans les deux NAT.

En principe, l'ouverture simultanée fonctionne aussi bien pour TCP que pour UDP. En pratique, cependant, il est difficile de la faire fonctionner avec TCP, et elle est donc généralement implémentée par des protocoles basés sur UDP.

L'ouverture simultanée dépend du fait que les NAT de *A* et *B* implémentent un *mapping* indépendant (paragraphe 1.2.1) ; il ne fonctionne que dans 90% des cas environ dans l'Internet actuel. La communeauté réseau travaille à remplacer les NAT symétriques par des NAT conservatifs.

Optimisations Plusieurs optimisations à ce procédé sont possibles. Lorsque *A* connaît déjà l'adresse externe de *B*, la deuxième branche de l'échange n'est pas nécessaire. Si le serveur d'application et le serveur d'adresses sont sur le même hôte, il n'est pas nécessaire pour *A* de faire deux échanges, il peut obtenir son adresse et demander une connexion de la part de *B* en un seul message.

STUN Il existe un protocole normalisé de détermination d'adresses, nommé STUN. Il existe de nombreuses implémentations libres de STUN, et plusieurs serveurs STUN ouverts sur l'Internet.

3.4 Relais et TURN

L'expérience montre que les techniques d'ouverture simultanée ne fonctionnent que dans 90% des cas environ ; dans 10% des cas, on rencontre soit du fait de NAT ayant un *mapping* symétrique, soit un *firewalls* fascistes. Or, 10% est un chiffre très élevé : dans un amphi de 120 personnes, ce seraient 12 personnes qui ne peuvent pas communiquer. Dans ces cas, il n'existe pas d'autre solution que de passer par un serveur relais.

Comme dans le cas de l'ouverture simultanée, on peut soit utiliser le serveur d'application comme relais, soit utiliser un serveur de relais indépendant du serveur d'application.

TURN TURN est un protocole servant à établir des relais pour du traffic UDP. Un client se trouvant sur un réseau restrictif contacte un serveur TURN, en UDP si le réseau le permet, en TCP sinon, et lui envoie le traffic que le serveur TURN fait suivre en UDP vers la destination. Si les deux pairs sont sur des réseaux restrictifs, et s'ils utilisent des serveurs TURN distincts, le traffic transite à travers deux serveurs TURN.

À la différence de STUN, qui est très économe, TURN est un protocole relativement coûteux (un serveur TURN fait suivre tout le traffic), et il n'existe donc pas de serveurs TURN ouvers dans l'Internet. Cependant, il existe d'excellentes implémentations de TURN libres et faciles à installer.

3.5 ICE

Du fait des *middleboxes*, le protocole de connexion devient extrêmement complexe : pour établir une connexion, une application pair-à-pair doit, pour chacune de ses adresses et pour chacune des adresses du pair :

1. tenter d'établir une connexion directe ; puis
2. tenter une connexion simultanée ; puis
3. tenter une connexion à travers un relais.

L'étape (1) réussit si le pair n'est pas derrière un NAT, s'il utilise un protocole de *port mapping*, et, dans certains cas, s'il est derrière un NAT conique. L'étape (2) réussit si aucun des pairs n'est derrière un NAT symétrique. Enfin, l'étape (3) réussit dans la quasi-totalité des cas, mais mène à une connexion plus coûteuse (elle charche le serveur relais).

ICE est un algorithme d'établissement de connexion normalisé qui définit comment faire toutes ces étapes. Comme le problème est complexe, ICE est lui-même extrêmement complexe, mais permet de réussir une connexion même dans les réseaux les plus restrictifs ; il existe plusieurs bibliothèques libres qui implémentent ICE.

Digression : la couche session Dans la suite de protocoles TCP/IP, l'application est écrite directement au-dessus de la couche transport. Dans la suite OSI qui devait à un moment succéder à TCP/IP, il existait une couche session (couche 5) qui avait plusieurs tâches, dont celle de cacher à l'application les adresses de couche transport et de simplifier l'automate d'établissement de connexion.

Je me demande parfois si ICE ne serait pas un protocole de couche session.