AmarOk

CC

Geste de savoir

p2p internals #1

2 juin 2019

Table des matières

J'aime les systèmes distribués pour énormément de raisons. Mais il s'agit de systèmes complexes qui requièrent des interactions sur de nombreux niveaux. Cependant, même si de nombreuses applications pairs à pairs existent, il est très rare de pouvoir les utiliser sans peine pour énormément de différentes raisons. Par exemple, aujourd'hui, nos systèmes d'exploitation évitent un maximum les calculs et déplace la plupart des calculs ailleurs (ex : Android pour économiser la batterie et l'utilisation de données), IPv4 étant encore massivement utilisé le problème de NAT est toujours présent, la plupart des interfaces utilisateurs sont développées pour des applications centralisées, etc. C'est pourquoi j'écris cette série (encore une) pour expliquer quelques problèmes communs qu'on peut rencontrer et quelles solutions peuvent être mises en place lorsqu'on créé une application distribuée.

Dans ce premier article, imaginons que Alice souhaite créer une application de transfert de fichiers directement entre les appareils, sans serveur de stockage. Il existe de nombreuses applications de ce type dans notre navigateurs (exemple ShareDrop - merci webrtc), messageries distribuées (la plupart des exemples que je vais prendre vont sans doute venir de Jami , etc. Appelons l'application d'Alice *DEL* (*del* voulant dire *partager* en norvégien mais aussi le diminutif de *delete* en anglais).

1. Premiers transferts

DEL est vraiment simple à comprendre. Pour un transfert, deux clients sont nécessaires, sur deux appareils. Chaque client aura une ou plusieurs adresses (correspondant à l'IP et un port que le client écoute, sur les réseaux locaux et publics). Si l'utilisateur veut transmettre un fichier, il a juste à donner à son client le fichier à transmettre et l'adresse de l'autre client. *DEL* va alors créer un lien TCP entre les deux clients et transmettre le fichier. Alice teste alors son application avec ses appareils. Elle est heureuse car elle a réalisé sa première application pair à pair ! Plus qu'à en informer son ami, Bob.

Petite note : lors d'un transfert de fichier, nous ne voulons pas de pertes de paquets. Il faut donc un transport fiable, comme TCP et non UDP.

Dès qu'Alice l'informe, Bob télécharge directement l'application, lance son client et tente de partager une photo de Clémentine à la plage à son amie... mais rien, le transfert échoue, impossible d'avoir une connexion directe entre les deux amis.

En réalité de nombreuses raisons peuvent être à l'origine de l'impossibilité d'avoir ce lien direct. La raison la plus commune est sans doute le **NAT** (dont je parlerais sans doute dans de futurs

épisodes). Pour faire court, le **NAT** a été créé pour disposer de plus d'adresses IP qu'il n'y en a réellement. Mais ce **NAT** va aussi rendre votre adresse dans votre réseau local privée et non accessible depuis l'extérieur sauf si vous demandez à votre routeur de vous transférer le traffic ou que vous utilisez une des nombreuses techniques de contournement de NAT comme *UPnP, STUN, TURN, etc*

2. Contournons le NAT !

N'ayez pas peur du **NAT** ! Beaucoup d'applications utilisent des méthodes pour passer au travers de ce problème. *DEL* aussi va pouvoir le contourner ! De plus, beaucoup de littérature existe à ce sujet. Dans ce billet, je vais seulement expliquer comment l'utilisation d'un serveur *TURN* (*Traversal Using Relays around NAT*) fonctionne (mais seulement pour TCP). D'autres méthodes existent, mais cette méthode à l'avantage de ne pas dépendre de beaucoup de paramètres des réseaux. Il est seulement nécessaire d'avoir un serveur *TURN* à disposition.

Le processus complet est expliqué dans la RFC 6062 , mais en résumé, un serveur *TURN* sera utilisé comme relai pour la connexion entre Alice et Bob. À la place de tenter d'ouvrir une connexion à l'adresse de Bob, Alice se connectera à l'adresse préparée par Bob sur le serveur relai (si le relai fonctionne en mode écoute de connexion), ou alors le TURN tentera de se connecter à Bob (ce cas ne sera pas traité ici). Ainsi, le **NAT** ne pourra rien faire car Bob ne contactera jamais Alice directement, seulement un point qu'elle écoute en dehors de son réseau.

Pour résumer, voici à quoi la connexion ressemblera :

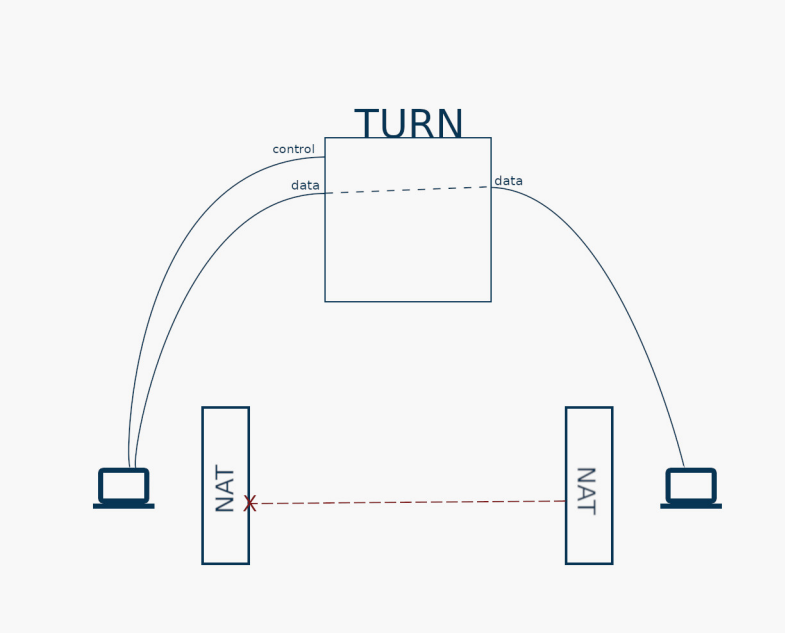

Figure 2. – TURN - RFC 6062

3. Comment communiquer par un serveur TURN

Dorénavant, *DEL* doit être compatible avec un serveur *TURN*. Voici comment l'application procède :

La première étape s'agit de préparer le terrain. L'application se connecte alors au relai et lui envoie une première requête demandant de lui ouvrir un transport de type TCP (avec l'attribut `REQUESTED-TRANSPORT`). Ainsi, le relai va ouvrir un port où il écoutera le traffic entrant. La combinaison adresse du *TURN* et du port ouvert s'adresse une adresse relai. *DEL* aura donc, en plus de son adresse locale, de son adresse publique (sur internet) une adresse relai que l'utilisateur pourra partager à ces contacts. À cet instant, la connexion entre le client et le *TURN* s'appelle la connexion de contrôle. Car les requêtes de contrôle sont transmises via ce lien. Deux scénarios sont alors possibles à cet instant.

— Soit le *TURN* est utilisé pour envoyer des données. Donc il se connectera à un pair voulu. Ce n'est pas le cas que je décrirais ici.
— Soit le *TURN* est utilisé pour recevoir des données, c'est le cas qu'utilise *DEL*

Dans le second cas, si Bob veut recevoir un fichier d'Alice, le client de Bob devra alors envoyer une requête au serveur *TURN* de type `CreatePermission` afin d'autoriser Alice de se connecter à l'adresse relai. Dès qu'Alice va tenter de se connecter, le client de Bob va recevoir un requête de type `ConnectionAttempt` avec un id. Si le client souhaite accepter la connexion, il va alors se connecter au serveur une seconde fois et envoyer une requête de type `ConnectionBind` (sur ce nouveau lien). Ces deux nouvelles connexions (Alice->TURN, Bob->TURN) sont appellées les connexions de données, car c'est ici que le fichier sera transféré.

Lorsque le *TURN* reçoit cette requête il pourra alors connecter les deux liens ensemble et Alice et Bob pourront s'échanger des fichiers sans que le *NAT* bloque quoi que ce soit !

Si vous souhaitez creuser un peu le sujet, voici quelques liens :

— Pour monter votre propre serveur TURN, je recommande https ://github.com/coturn/coturn Simple à utiliser et supporte TCP
— Si vous voulez une bibliothèque, je recommande *pjproject*, qui supportera la **RFC 6062** dans la version 2.9 (https ://trac.pjsip.org/repos/ticket/2197 , https ://github.com/pjsip/pjproject/commit/fa6616c43c7e19797084f4e02a67d1fb6fd99473)