Institut de la Francophonie pour l'Informatique

Rapport pour TIPE

# *Le protocole STUN pour des connexions filtrées par des pare-feux*


| | |
|---|---|
| Sous la direction de Professeur | : M. Victor Moraru |
| Réalisé par | : DAO Xuan Sang |
| Promotion | : P11 |


Ha noï, Décembre 2006

## Remerciements

Tout d'abord, je voudrais exprimer mes grands remerciements à professeur ***VICTOR Moraru***, Mon responsable du TIPE, pour son aide et ses conseils pendant mon travail de recherche.

Je voudrais également remercier tous les professeurs de l'IFI pour m'avoir donné des connaissances et des conditions nécessaires pour réaliser ce travail.

Enfin, je tiens à remercier mes camarades de promotion XI pour ses soutiens et leur ambiance amicale et je leur souhaite de bon succès.

# *Table de manière*

# Liste des Acronymes

| | |
|---|---|
| STUN | Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NAT) |
| UDP | User Datagram Protocol |
| TCP | Transmission Control Protocol |
| NAT | Network Address Translator |
| TLS | Transport Layer Security |
| RFC | Request For Comments |
| HMAC | Hashing for Message Authentication |

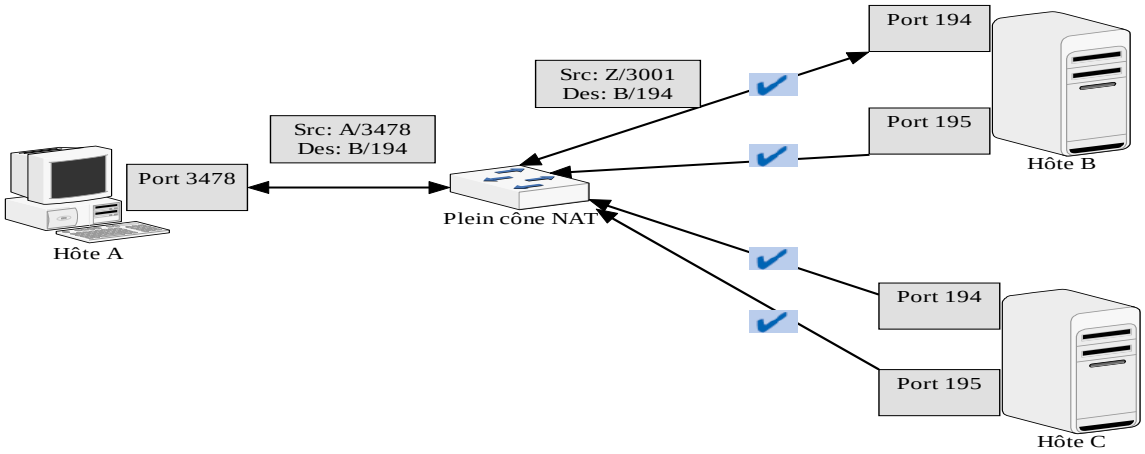
# Introduction

Aujourd'hui on utilise le réseau pair à pair est de plus en plus pour partager des données. Donc on veut toujours établir des réseaux partagés plus des données, mais le réseau doit être sécurité. Donc une proposition d'utiliser des pare-feux (fire-wall) pour assurer la sécurité de réseau. On affirme qu'il est possible de connecter deux noeuds si seulement l'un des deux est derrière un pare-feux. On veut s'échanger des données en UDP entre deux clients qui sont derrière des pare-feux en utilisant le protocole STUN. Alors on doit utiliser un intermédiaire non restreint par un pare-feux, il joue de rôle de serveur et tout d'abord deux clients connectent au serveur, puis ils se connectent directement. Le sujet du stage consiste à étudier en détail cette problématique et le protocole STUN pour communiquer par UDP. En travail pratique, on doit avoir un programme client – serveur pour découvrir les types de NAT et pour échanger des fichiers entre deux clients qui sont dans deux réseaux privés différences et tous dernières des NAT.

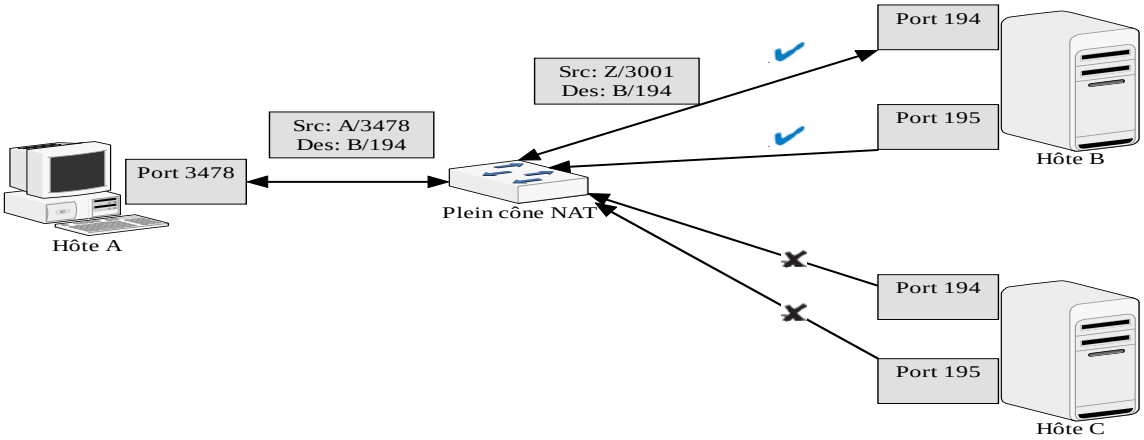
# Type de NAT

NAT, signifie Network Address Translator en anglais, est traducteur d'adresse réseau. NAT traduit l'adresse des postes dans le réseau privé à l'adresse public du réseau privé. NAT a besoin un ou plusieurs adresses publics. Selon traitement d'UDP, on classifie NAT en quatre types.

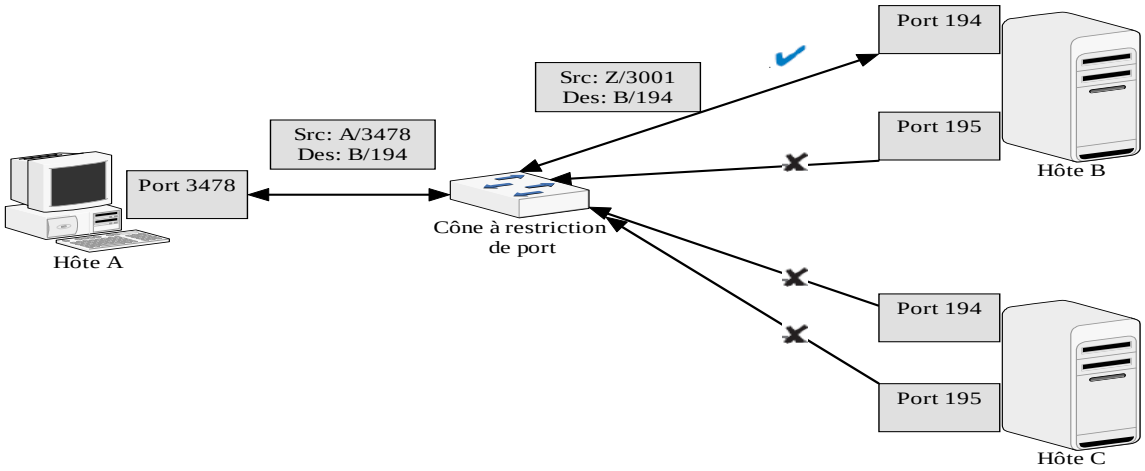
***Plein cône***: Un NAT en plein cône est un NAT où toutes les demandes provenant de la même adresse et port IP internes sont transposées sur la même adresse et port IP externes. Dans le cas NAT plein cône, tout hôte externe peut envoyer un paquet à l'hôte interne, en envoyant un paquet à l'adresse externe transposé.



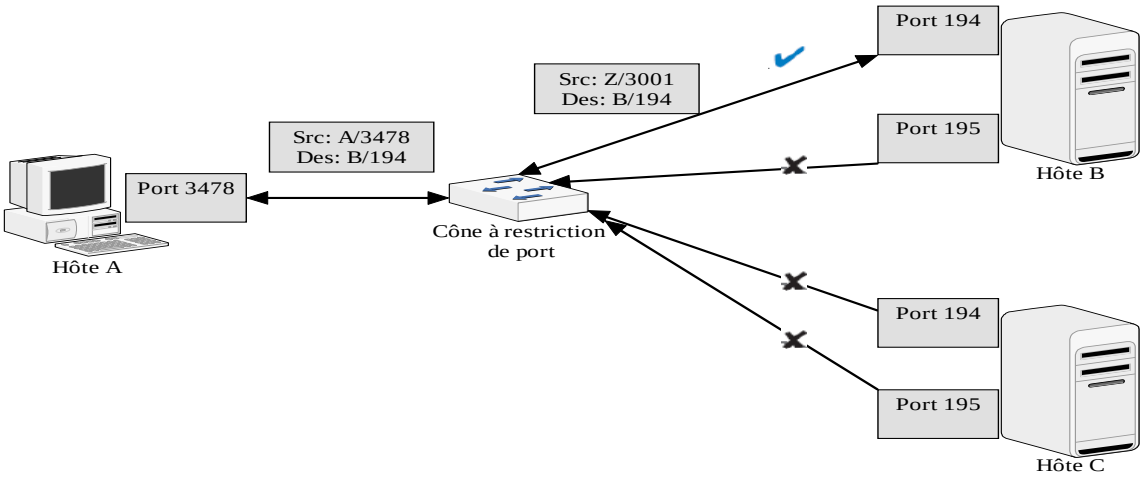

***Cône restreint***: Un NAT en cône restreint est celui où toutes les demandes provenant de la même adresse et port IP internes sont transposées sur la même adresse et port IP externes. A la différence du NAT en plein cône, un hôte externe (avec l'adresse IP X) ne peut envoyer un paquet à l'hôte interne que si l'hôte interne avait préalablement envoyé un paquet à l'adresse IP X.




***Cône à restriction de port***: Un NAT en cône à restriction de port est comme un NAT à cône restreint, mais la restriction inclut les numéros de port. Précisément, un hôte externe ne peut envoyer un paquet, avec l'adresse IP de source X et le port de source P, à l'hôte interne que si l'hôte interne avait préalablement envoyé un paquet à l'adresse IP X et au port P.




***Symétrique***: Un NAT symétrique est celui dans lequel toutes les demandes provenant de la même adresse et port IP internes, à une adresse et port IP de destination spécifique, ont transposées sur la même adresse et port IP externes. Si le même hôte envoie un paquet avec la même adresse et port de source, mais une destination différente, on utilise une transposition différente. De plus, seul l'hôte externe qui reçoit un paquet peut renvoyer un paquet UDP à l'hôte interne.



# Protocol STUN

Le protocole STUN signifie Simple Traversal of UDP throught NAT, c'est un protocole destiné à permettre la traversée des routeurs NAT, il permet d'établir des connexions UDP d'un poste, qui est situé derrière des routeurs NAT de découvrir son adresse IP publique ainsi que le type de routeur NAT derrière lequel il est. Ces informations sont utilisées pour échanger correctement des données UDP avec l'extérieur d'un réseau NATé. Pour ce faire, le poste situé en réseau privé émet des requêtes vers le serveur STUN public. Le serveur STUN renvoie l'adresse publique et le port du routeur NAT associé au réseau privé du poste.

Dans le protocole STUN un poste joue de rôle client et autre poste joue de rôle serveur :
Client STUN : un client STUN (aussi appelé simplement client) est une poste qui envoie des demandes STUN à serveur. Un client STUN peut travailler sur un système terminal, ou fonctionner dans un élément de réseau.
Serveur STUN : un serveur STUN (aussi appelé simplement serveur) est une poste qui reçoit des demandes STUN de client, et envoie des réponses STUN à client. Les serveurs STUN sont normalement situés à l'Internet public.

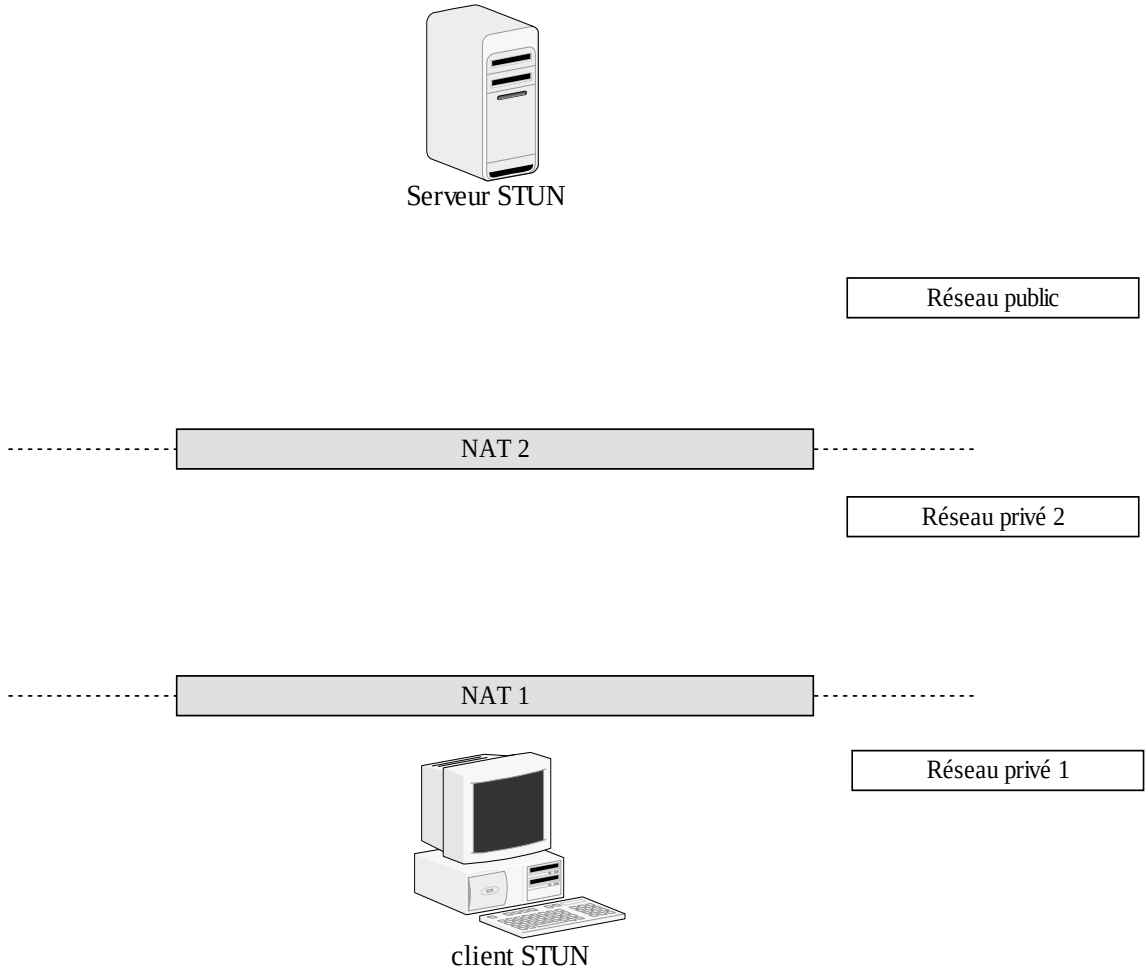
## *Fonctionnement générale du protocole STUN*

La configuration STUN typique est indiquée à la Figure 1. Un client STUN est connecté au réseau privé 1. Le réseau privé 1 se connecte au réseau privé 2 à travers le NAT 1. Le réseau privé 2 se connecte à l'Internet public à travers le NAT 2. Le serveur STUN demeure dans l'Internet public.

STUN est un simple protocole client - serveur. Un client envoie une demande à un serveur, et le serveur retourne une réponse. Il y a deux types de demandes, les demandes de lien, envoyées sous UDP, et les demandes de secret partagé, envoyées sous TLS [2] sur TCP. Correspondance avec les deux types de demande sont la réponse de lien, réponse d'erreur de lien, réponse de secret partagé et réponse d'erreur de secret partagé. Les demandes de secret partagé demandent au serveur de retourner un nom d'utilisateur (*username*) et un mot de passe (*password*) temporaires. Ce nom d'utilisateur et ce mot de passe sont utilisés dans une demande de lien et réponse de lien ultérieures, pour les besoins de l'authentification et de l'intégrité du message.

Les demandes de lien sont utilisées à déterminer les liens alloués par les NAT. Le client envoie une demande de lien à serveur, en forme UDP. Le serveur examine l'adresse et le port IP de client qui a envoyé la demande, et les copies dans une réponse qui est renvoyée au client. Il y a certains paramètres dans la demande qui permettent au client de demander que la réponse soit envoyée ailleurs, ou que le serveur envoie la réponse à partir d'un port et adresse différents. Il y a des attributs pour l'intégrité du message et l'authentification.

***Figure 1 : Configuration le protocole STUN***




Un grand but du protocole STUN est de découvrir la présence d'un ou plusieurs NAT, apprendre et utiliser les liens qu'il alloue.

Normalement, le protocole STUN est incorporé dans une application pour obtenir une adresse et port IP publics qui puissent être utilisés pour recevoir des données. Par exemple, il pourrait avoir besoin d'obtenir une adresse et un port IP pour recevoir du trafic de protocole de transport en temps réel (RTP, *Real Time Transport Protocol*) [12]. Tout d'abord, le client STUN dans l'application

envoie une demande de secret partagé STUN à serveur STUN, obtient un nom d'utilisateur et un mot de passe, et lui envoie alors une demande de lien (*Binding Request*). Le serveur STUN peuvent découvrir à travers les enregistrements SRV de DNS [3], et le client est configuré avec le domaine à utiliser pour trouver le serveur STUN. En général, un client peut déterminer l'adresse ou le nom de domaine d'un serveur STUN par d'autres moyens. Un serveur STUN peut toujours être incorporé dans un système terminal.

La demande de lien STUN est utilisée pour découvrir la présence d'un NAT, et à découvrir les transpositions d'adresse et port IP publics générées par le NAT. Les demandes de lien sont envoyées au serveur STUN par client en forme UDP. Quand une demande de lien arrive au serveur STUN, la demande de lien peut être passée à travers un ou plusieurs NAT entre le client STUN et le serveur STUN. L'adresse et IP de source de la demande reçue par le serveur seront l'adresse et IP qui sont transposée par le NAT le plus proche du serveur (comme dans la figure 1, l'adresse et IP sont transposés par le NAT 2). Le serveur STUN va copier cette adresse et port IP de source dans une réponse de lien STUN, et il va la renvoyer à l'adresse et port IP de source de la demande STUN. Pour tous les types de NAT ci-dessus (Plein cône, restreint, restriction de port et symétrique), cette réponse arrivera au client STUN.

Quand le client STUN reçoit la réponse de lien STUN du serveur STUN, il va comparer l'adresse et le port IP du message de la réponse avec l'adresse et le port IP du client, qu'il y a lié lors de l'envoi de la demande. Si l'adresse et port IP ne correspondent pas, on sait que le client STUN est derrière un ou plusieurs NAT. Avec le NAT de plein cône, l'adresse et port IP du message de la réponse STUN sont publics, et peuvent être utilisés par tout hôte sur l'Internet public pour envoyer des message à l'application qui a envoyé la demande STUN. L'application doit écouter l'adresse et le port IP à partir desquels la demande STUN a été envoyée. Tous messages envoyés par un hôte sur l'Internet public à l'adresse et port publics appris par STUN seront reçus par l'application. Dans une application, le protocole STUN est toujours utilisé pour découvrir les types de NAT.

Normalement, avec un NAT en plein cône, l'hôte ne sait pas encore derrière quel type de NAT il se trouve. Pour déterminer les types de NAT, le client utilise des demandes de lien STUN et utilise des réponses pour avoir quels types de NAT que le client est dernier. Le client devrait envoyer une seconde demande de lien STUN, mais avec une adresse IP différente, et à partir de la même adresse et port IP de client. Si l'adresse et le port IP de la réponse de serveur sont différents de ceux dans la première réponse, on sait que le client

est derrière un NAT symétrique. Pour déterminer un NAT en plein cône, le client envoie une demande de lien STUN avec des fanions qui sont utilisés pour dire au serveur STUN d'envoyer une réponse à partir d'une adresse et port IP différents de ceux sur lesquels la réponse a été reçue. En d'autres termes, le client envoie une demande de lien à l'adresse/port IP Aa/Pb en utilisant une adresse/port IP Ax/Py, le serveur STUN devrait envoyer la réponse de lien à Ax/Py en utilisant l'adresse/port IP de source Ac/Pb. Si le client reçoit cette réponse, il sait qu'il est derrière un NAT en plein cône.

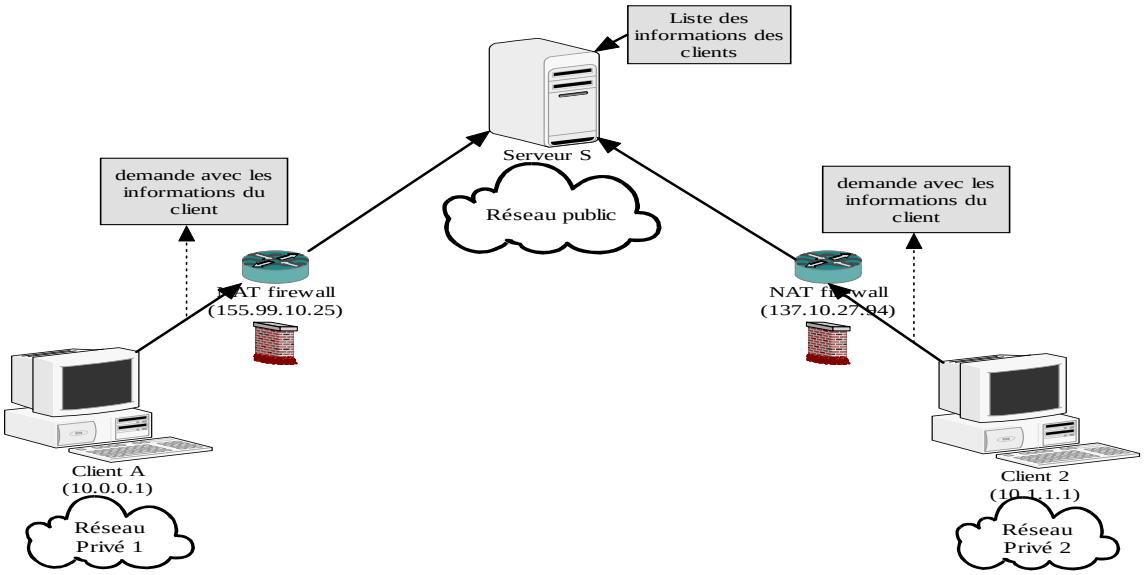
Pour déterminer le NAT en cône à restriction de port, ou le NAT en cône restreint. Le protocole STUN permet aussi au client de demander au serveur d'envoyer la réponse de lien à partir de la même adresse IP que celle où la demande a été reçue, mais avec un port différent. Cette façon peut déterminer le NAT en cône à restriction de port, ou le NAT en cône restreint.

## *Connexion*

### Client – serveur

C'est le modèle client – serveur du protocole STUN pour établir des communications à travers les pare-feux et les routeurs NAT. Dans le modèle client – serveur, il est facile de placer un ou plusieurs des pare-feux et des NAT entre les clients et serveur.

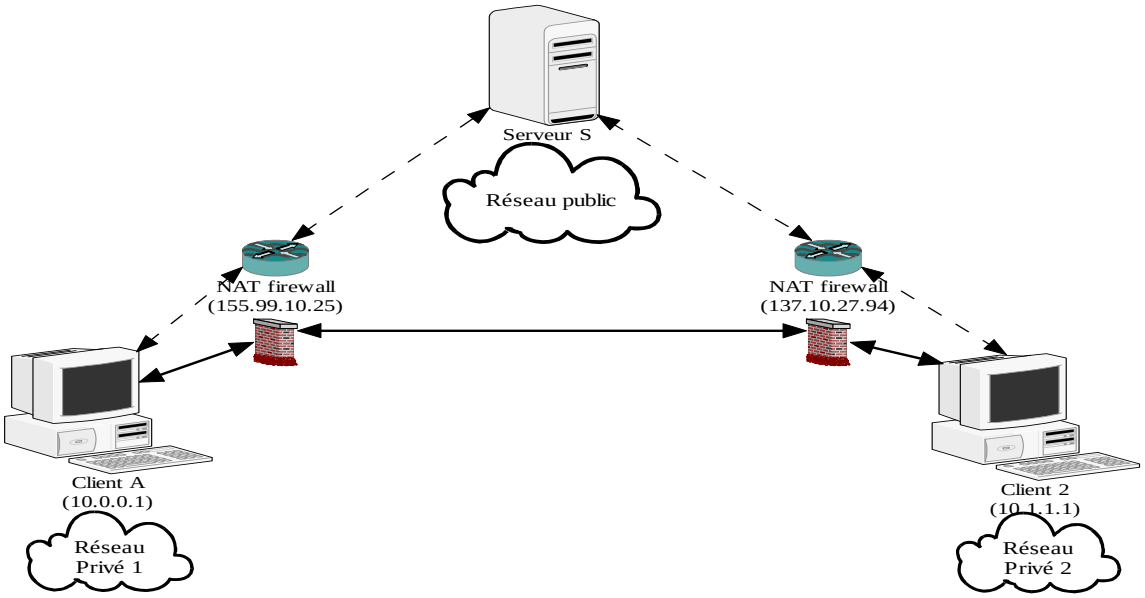
Pour le protocole STUN, client doit enregistrer à serveur. C'est – à – dire, le client envoie une demande de lien à serveur avec les informations du client et le serveur copie les informations du client dans une liste des informations des clients. Donc, le serveur a une liste des informations des tous clients qui ont enregistrés à serveur.




## Client – client

Modèle communication client – client (poste à poste) est le modèle que deux client, après enregistrement à serveur, peuvent s'échanger des données directement.

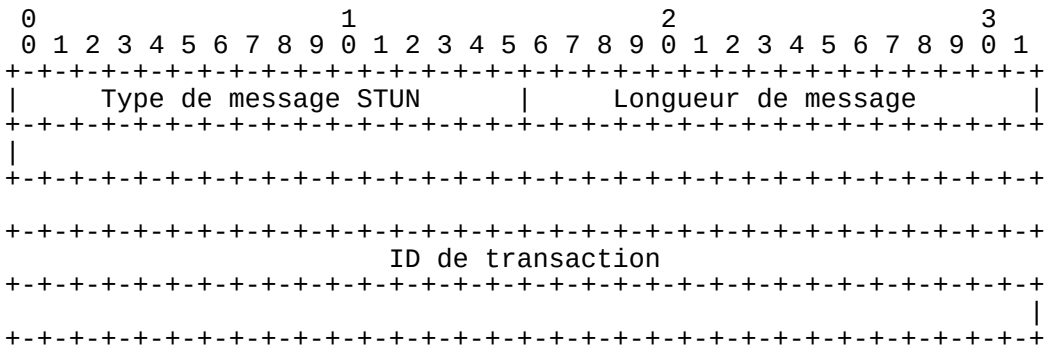
Dans le protocole STUN, après d'enregistrement des clients au serveur, les clients ont des informations des autres clients, donc, les deux clients peuvent communiquer directement.



## *Message du protocole STUN*

### En – tête du message

Tous les messages STUN comportent un en-tête de message en 20 octets de donnée. Entête du message STUN contient 3 attributs : Type de message STUN, Longueur de message et ID de transaction de message. Comme ci-dessous:




Dans ce cas, on utilise premier attribut ***Type de message*** pour savoir quels type de message que le protocole STUN utilise pour communication entre client et serveur. Type de message STUN contient 6 valeurs pour chaque type de message :

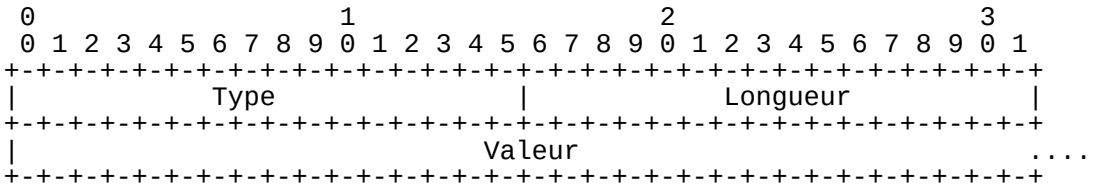
0x0001 : Demande de lien
0x0101 : Réponse de lien
0x0111 : Réponse d'erreur de lien
0x0002 : Demande de secret partagé
0x0102 : Réponse de secret partagé
0x0112 : Réponse d'erreur de secret partagé

Le deuxième attribut dans en-tête c'est la ***longueur de message***. Longueur de message est le compte, en octets, de la taille du message, non inclus l'en-tête de 20 octets.

Troisième attribut dans en-tête c'est l'***ID de transaction***. L'ID de transaction est un identifiant de 128 bits. Il sert aussi d'élément pour rendre aléatoire la demande et la réponse. Toutes les réponses portent le même identifiant que la demande à laquelle elles correspondent.

### Attributs du message

Dans message STUN, tout d'abord, on trouve un en-tête de 20 octets, puis on peut trouver plusieurs attributs. Tous les attributs sont codés en TLV (Type – Longueur – Valeur). Type contient de 16 bits de donnée, Longueur contient aussi de 16 bits de donnée.




Type contient 11 valeurs de l'attribut :

0x0001: MAPPED-ADDRESS *(adresse transposée)*
0x0002: RESPONSE-ADDRESS *(adresse de réponse)*
0x0003: CHANGE-REQUEST (*demande de modification)*
0x0004: SOURCE-ADDRESS *(adresse de source)*
0x0005: CHANGED-ADDRESS *(adresse modifiée)*
0x0006: USERNAME *(nom d'utilisateur)*
0x0007: PASSWORD *(mot de passe)*
0x0008: MESSAGE-INTEGRITY *(intégrité du message)*
0x0009: ERROR-CODE *(code d'erreur)*
0x000a: UNKNOWN-ATTRIBUTES *(attributs inconnus)*
0x000b: REFLECTED-FROM *(réfléchi de)*

Longueur de message se réfère à la longueur d'élément valeur, elle est exprimée comme un nombre entier arithmétique d'octets.

## *Comportement du client*

Le comportement du client est très simple. Il est de découvrir les types de NAT entre client et serveur STUN, obtenir un secret partagé, formuler la demande de lien, traiter la demande, et traiter les réponses de lien. Selon les demandes de lien et les réponses des serveurs, client sait qu'il est dernière quels types de NAT.

## Découvert

Le client du protocole STUN utilise "udp" pour l'envoi des demandes de lien, ou "tcp" pour l'envoi des demandes de secret partagé. Les procédures de la RFC 2782 sont suivies pour déterminer le serveur à contacter. La RFC 2782 décrit le détail de la façon de trier un ensemble d'enregistrements du client au serveur. Cependant, elle établit seulement que le client devrait "essayer de se connecter au (protocole, adresse, service)" sans donner aucun détail sur ce qui se passe en cas d'échec. Ces détails sont décrits ici pour le protocole STUN.

Pour les demandes STUN, il existe une défaillance du transport de quelque sorte (généralement, due à des erreurs fatales d'ICMP dans UDP ou des défaillances de connexion dans TCP). L'échec survient aussi si la transaction échoue à cause d'un dépassement de temporisation. Cela survient 9,5 secondes après l'envoi de la première demande, à la fois pour la demande de secret partagé et pour la demande de lien. Voir au paragraphe «formulation de la demande de lien » les détails sur les dépassements de temporisation de transaction pour les demandes de lien.
Le port défaut pour les demandes du client STUN est 3478, pour TCP et UDP. Les administrateurs DEVRAIENT utiliser ce port dans leurs enregistrements au serveur, mais PEUVENT en utiliser d'autres, par exemple, on peut utilise le port 3479.

Si le client ne peut pas d'enregistrement au serveur, le client effectue une recherche d'enregistrement A sur le nom de domaine. Le résultat sera une liste d'adresses IP, dont chacune peut être contactée au port par défaut.

## Obtention d'un secret partagé

Dans le protocole STUN il y a plusieurs attaques possibles sur les systèmes STUN. Beaucoup d'entre elles sont prévenues à travers l'intégrité des demandes et des réponses. Pour contre les attaches, STUN utilise le demande de secret partagé et réponse de secret partagé entre le client et le serveur, qui sert de matériau de clé pour un HMAC utilisé dans la demande de lien et la réponse de lien.

STUN permet le demande de secret partagé et réponse de secret partagé soit obtenu de n'importe quelle façon (par exemple, Kerberos [14]). Cependant, il DOIT avoir au moins 128 bits aléatoires. Afin d'assurer l'interopérabilité, la

présente spécification décrit un mécanisme fondé sur TLS. Ce mécanisme DOIT être mis en oeuvre par les clients et les serveurs.

Tout d'abord, le client détermine l'adresse et port IP auxquels il va ouvrir une connexion TCP avec le serveur. Cela est fait en utilisant les procédures de découverte, qui a parlé ci-dessus. Le client ouvre la connexion à cette adresse et port, et commence la négociation TLS [2] avec serveur. Le client DOIT vérifier l'identité du serveur. Pour ce faire, il suit les procédures d'identification définies dans la RFC 2818 [5]. Pour les besoins de l'utilisation avec la présente spécification, le client traite le nom de domaine et adresse IP utilisée au paragraphe « Découvert ».

Une fois que la connexion est ouverte, le client envoie une demande de secret partagé. Cette demande n'a pas d'attributs, seulement l'en-tête, dans l'en-tête, il y a trois attributs : type de demande, longueur de demande et ID transaction. L'identifiant de transaction dans l'en-tête DOIT satisfaire aux exigences formulées pour l'identifiant de transaction dans une demande de lien, décrites au paragraphe « formulation de la demande de lien » ci-dessous. Le serveur renvoie une réponse, c'est une Réponse de secret partagé ou une Réponse d'erreur de secret partagé.

Si la réponse est la Réponse d'erreur de secret partagé, le client va vérifier le code de réponse dans l'attribut ERROR-CODE. L'interprétation de ces codes de réponse est identique au traitement du paragraphe « Traitement des réponses de lien» pour la réponse d'erreur de lien.

Si un client reçoit une Réponse de secret partagé avec un nom d'utilisateur et mot de passe à courte durée de vie, codés respectivement dans les attributs USERNAME et PASSWORD. Le client PEUT créer plusieurs demandes de secret partagé sur la connexion, et il PEUT faire avant de recevoir les Réponses de secret partagé aux demandes de secret partagé précédentes. Le client DEVRAIT fermer la connexion dès qu'il a fini d'obtenir les noms d'utilisateur (USERNAME) et les mots de passe (PASSWORD).

Le paragraphe « Formulation de la demande de lien » ci-dessous décrit comment ces mots de passe sont utilisés pour fournir la protection de l'intégrité sur les demandes de lien.

## Formulation de la demande de lien

Le client formule la demande de lien suit les règles de syntaxe définies par les attributs du message, c'est – à – dire, selon les attributs du message STUN, client formule les réponses du serveur. Dans le cas de deux demandes ne sont pas identiques au bit près et ne sont pas envoyées au même serveur à partir de la même adresse et port IP, les demandes DOIVENT ajouter des identifiants de transaction différents dans les demandes.

L'identifiant de transaction DOIT suivre le distribué uniformément et aléatoirement entre 0 et 2**128 - 1. Cette large gamme est nécessaire parce que l'identifiant de transaction sert de forme aléatoire, qui sert à prévenir le rejeu de réponses signées antérieures à partir du serveur. Le type de message de la demande DOIT être "Binding Request".

Dans la demande de lien, l'attribut RESPONSE-ADDRESS est utilisé si le client veut la réponse soit envoyée à une adresse et port IP différents de ceux d'où la demande a été envoyé. C'est très utile pour déterminer si le client est derrière un pare-feu, et pour des applications qui ont des composants pour contrôle et de données séparés.

Dans la demande de lien, le client DEVRAIT ajouter un attribut MESSAGE-INTEGRITY et USERNAME. L'attribut MESSAGE-INTEGRITY contient un HMAC [13]. La valeur du nom d'utilisateur, et la clé à utiliser dans l'attribut MESSAGE-INTEGRITY dépendent du mécanisme de secret partagé. Le USERNAME DOIT être un nom d'utilisateur valide obtenu d'une Réponse de secret partagé, si la demande de secret partagé STUN a été utilisée. Le secret partagé pour le HMAC est la valeur de l'attribut PASSWORD obtenu de la même Réponse de secret partagé.

## Traitement des réponses de lien

La réponse de la demande de lien peut être une Réponse de lien ou une Réponse d'erreur de lien. La Réponse d'erreur de lien est toujours reçue à l'adresse et port de source d'où la demande a été envoyée. La Réponse de lien sera reçue à l'adresse et port placés dans l'attribut RESPONSE-ADDRESS de la demande. Si aucune n'était présente, la réponse de liens sera reçue à l'adresse et port de source d'où la demande a été envoyée.

Si la réponse de la demande de lien est une Réponse d'erreur de lien, le client vérifie le code de réponse dans l'attribut ERROR-CODE de la réponse. Pour chaque code de réponse dans l'attribut ERROR-CODE, le client génère une réponse correspondance.

Si le client a reçu une réponse avec un attribut et le type de l'attribut est supérieur à 0x7fff, le client DOIT ignorer l'attribut. Si le client reçoit une réponse avec un attribut dont le type est inférieur ou égal à 0x7fff, les retransmissions de demandes DOIVENT cesser, mais la réponse en elle-même est ignorée.

Si le client reçoit une Réponse de lien, il DEVRAIT vérifier la réponse à la recherche d'un attribut MESSAGE-INTEGRITY. Si cet attribut n'est pas présent, et le client a placé un attribut MESSAGE-INTEGRITY dans la demande, le client DOIT écarter la réponse. S'il en est un présent, le client calcule le HMAC sur la réponse. La clé à utiliser dépend du mécanisme de secret partagé. Si la demande de secret partagé STUN a été utilisée, la clé DOIT être calculé l'attribut MESSAGE-INTEGRITY dans la demande.

Si le client calcule le HMAC et il trouve qu'il est différence de celui de la réponse, le client DOIT écarter la réponse, et DEVRAIT alerter l'utilisateur peut-être il y a une attaque possible. Si le HMAC calculé correspond à celui de la réponse, le processus continue.

Le client reçoit une réponse de lien (Peut-être, Réponse d'erreur de lien ou Réponse de lien), la demande de lien terminera les retransmissions de cette demande. Cependant, le client DOIT continuer à guetter pendant 10 secondes, après la première réponse, les réponses à une Demande de lien. Si le client reçoit d'autres réponses dans cet intervalle de temps avec des types de message différents (Réponses de lien et Réponses d'erreur de lien, par exemple) ou des MAPPED-ADDRESS différentes, peut-être, il y a une attaque possible

## *Comportement du serveur*

Selon le type de demande du client, le serveur va créer la réponse correspondances. Avec la demande de lien du client, le serveur peut produire la réponse de lien ou la réponse d'erreur de lient. Avec la demande de secret partagé, le serveur peut envoyer la réponse de secret partagé ou la réponse d'erreur de secret partagé.

## Demandes de liens

Pour le serveur STUN, on doit avoir deux interfaces de réseaux, c'est – à – dire dans le serveur STUN, on a toujours deux IP adresses publics. Le serveur STUN est toujours prêt à recevoir des demandes de lien sur quatre combinaisons d'adresse/port - (A1, P1), (A2, P1), (A1, P2), et (A2, P2). (A1, P1) sont l'adresse et port primaires. Normalement, par défaut, P1 sera le port 3478, et P2 sera le port 3479.

Il y a un attribut de MESSAGE-INTEGRITY, comme j'en ai parlé ci-dessus, dans la demande de lien, le Serveur vérifie toujours cet attribut dans la demande de lien du client. Si cet attribut n'est pas présent, et si le serveur exige les vérifications d'intégrité sur la demande, il génère une réponse d'erreur de lien. Si le serveur trouve l'attribut MESSAGE-INTEGRITY dans la demande de lien, le serveur va calculer le HMAC (L'attribut MESSAGE-INTEGRITY contient un HMAC-SHA1 du message STUN. Il peut être présent dans les Demandes de lien ou les Réponses de lien) sur la demande de lien.

En supposant que la demande est correctement, le serveur DOIT générer une seule réponse de lien. Dans la réponse de lien, il DOIT contenir le même ID de transaction que celui contenu dans la demande de lien. La longueur dans l'en-tête de message DOIT contenir la longueur totale du message en octets, non compris l'en-tête. La réponse de lien DOIT avoir le type de message "Réponse de lien".

Si le serveur reçoit une demande de lien, il DOIT ajouter un attribut MAPPED-ADDRESS à la réponse de lien. L'adresse IP de cet attribut DOIT être réglé à l'adresse IP de source, qui est observée dans la demande de lien. Le port de cet attribut DOIT être réglé au port de source, qui est observé dans la demande de lien.

Si on ne trouve pas d'attribut RESPONSE-ADDRESS dans la demande de lien, l'adresse et port de destination de la réponse de lien DOIT être la même que celle de l'adresse et port de source de la demande de lien.

L'adresse et port de source de la réponse de lien dépendent de la valeur de l'attribut CHANGE – REQUEST et de l'adresse et port sur lesquels la demande de lien a été reçue. Elles sont résumées au Tableau ci-dessous.

| Fanions | Adresse de source | Port de source | CHANGED-ADDRESS |
|---|---|---|---|
| Aucun | Da | Dp | Ca:Cp |
| Change IP | Ca | Dp | Ca:Cp |

| | | | |
|---|---|---|---|
| Change port | Da | Cp | Ca:Cp |
| Change IP et Change port | Ca | Cp | Ca:Cp |

Dans cette table, Da représentant l'adresse IP de destination de la demande de lien (qui sera A1 ou A2, deux interfaces de réseau), et Dp représentant le port de destination de la demande de lien (qui sera P1 ou P2). Ca représentant l'autre adresse, de sorte que si Da est A1, Ca est A2 et vice versa. De même, soit Cp représentant l'autre port, de sorte que si Dp est P1, Cp est P2, et vice versa. Si on met le fanion "change port" dans l'attribut CHANGE-REQUEST de la demande de lien, et on ne met pas le fanion "change IP", c'est – à – dire on veut changer seulement port de la demande de lient, l'adresse IP de source de la réponse de lien DOIT être Da et le port de source de la réponse de lien DOIT être Cp. Si on met le fanion "change IP" dans la demande de lien, et on ne met pas le fanion "change port", c'est – à – dire, on veut changer seulement le port de la demande de lien, l'adresse IP de source de la réponse de lien DOIT être Ca et le port de source de la réponse de lien DOIT être Dp. Si on met tous les deux fanions dans la demande de lien, c'est – à – dire, on veut changer tous le port et l'adresse, l'adresse IP de source de la réponse de lien DOIT être Ca et le port de source de la réponse de lien DOIT être Cp. Si on ne met pas aucun des fanions dans la demande de lien, c'est – à – dire, on ne veut pas changer aucun le port et l'adresse, l'adresse IP de source de la réponse de lien DOIT être Da et le port de source de la réponse de lien DOIT être Dp.

L'attribut CHANGED-ADDRESS est ajouté à la réponse de lien par le serveur. La réponse de lien contient l'adresse et port IP de source qui seraient utilisés si le client avait établi les fanions "change IP" et "change port" dans la demande de lien. Comme résumé dans la table ci-dessus, L'adresse et le port respectivement Ca et Cp, indépendamment de la valeur des fanions de CHANGE-REQUEST.

Si dans la demande de lien, le serveur trouve les deux attributs USERNAME et MESSAGE-INTEGRITY, il DOIT ajouter un attribut MESSAGE-INTEGRITY à la réponse de lien. L'attribut contient un HMAC [13] dans la réponse de lien. La clé à utiliser dépend du mécanisme de secret partagé. Si le serveur reçoit une demande de secret partagé STUN, la clé DOIT être celle associée à l'attribut USERNAME présent dans la demande de lien.

Si dans la demande de lien, on trouve un attribut RESPONSE-ADDRESS, le serveur DOIT ajouter l'attribut REFLECTED-FROM à la réponse de lien. Si la demande de lien a été authentifiée en utilisant un nom d'utilisateur (USERNAME) obtenu d'une demande de secret partagé, l'attribut REFLECTED-FROM DOIT contenir l'adresse et port IP de source d'où venait

cette demande de secret partagé. Si le nom d'utilisateur (USERNAME) présent dans la demande n'a pas été alloué en utilisant une demande de secret partagé, l'attribut REFLECTED-FROM DOIT contenir l'adresse et port de source de l'entité qui a obtenu le nom d'utilisateur, au mieux qu'il puisse être vérifié avec le mécanisme utilisé pour allouer le nom d'utilisateur (USERNAME). Si le nom d'utilisateur (USERNAME) n'était pas présent dans la demande, et si le serveur avait l'intention de traiter la demande, l'attribut REFLECTED-FROM DEVRAIT contenir l'adresse et port IP de source d'où la demande vient.

## Demandes de secret partagé

Le serveur est toujours prêt de recevoir les demandes de secret partagé sur des connexions TLS (Transport Layer Security). Parce que le serveur reçoit une demande d'établissement d'une connexion TLS de client, donc, il DOIT continuer avec TLS (Transport Layer Security), et il DEVRAIT présenter un certificat de site.

Dans la demande de secret, On devrait utiliser la suite chiffrant TLS TLS_RSA_WITH_AES_128_CBC_SHA [4]. L'authentification du client TLS NE DOIT PAS être effectuée, car le serveur n'alloue aucune ressource aux clients, et la charge du calcul peut être une source d'attaques.

Si la demande de secret partagé est reçue sur le serveur STUN, il DOIT vérifier la demande de secret partagé, qui est arrivée sur une connexion TLS. Si la demande de secret partagé n'a pas reçu sur TLS, le serveur DOIT générer une Réponse d'erreur de secret partagé. La destination pour la réponse d'erreur de secret partagé dépend du transport sur lequel la demande a été reçue. Si le serveur reçoit la demande de secret partagé sur TCP, il génère la Réponse d'erreur de secret partagé, qui est envoyée sur la même connexion que celle sur laquelle la demande a été reçue. Si le serveur reçoit la demande de secret partagé sur UDP, il DOIT générer la Réponse d'erreur de secret partagé, qui est envoyée à l'adresse et port IP de source d'où la demande vient.

Toutes les Réponses d'erreur de secret partagé de serveur doivent contenir le même ID de transaction qu'il a reçu une demande de secret partagé de serveu. La longueur dans l'en-tête de message doit contenir la longueur totale de message en octets, non compris l'en-tête. La Réponse d'erreur de secret partagé doit contenir un type de message de "Réponse d'erreur de secret partagé" (0x0112) comme j'en ai parlé ci-dessus.

Si le client envoie une demande de secret partagé, le serveur crée une Réponse de secret partagé. La Réponse de secret partagé doit contenir le même ID de transaction que celui contenu dans la demande de secret partagé. La longueur de l'en-tête du message doit contenir la longueur totale du message en octets, non compris l'en-tête. La Réponse de secret partagé de serveur doit avoir un type de message de Réponse de secret partagé. La Réponse de secret partagé de serveur doit contenir un attribut nom d'utilisateur (USERNAME) et un attribut mot de passe (PASSWORD). L'attribut nom d'utilisateur (USERNAME) sert d'indice au mot de passe (PASSWORD), qui est contenu dans l'attribut mot de passe (PASSWORD). Le serveur peut utiliser quelques mécanismes de son choix pour générer le nom d'utilisateur. Validité signifie que le serveur peut calculer le mot de passe pour ce nom d'utilisateur. Le serveur doit être un mot de passe (PASSWORD) unique pour chaque nom d'utilisateur (USERNAME). Le serveur doit saisir un nom d'utilisateur différent pour chaque demande de secret partagé distincte. Distincte, dans ce cas, implique un ID de transaction différent.

Le mot de passe (PASSWORD) doit contenir le mot de passe lié à ce nom d'utilisateur (USERNAME). Le mot de passe (PASSWORD) doit avoir au moins 128 bits. La probabilité que le serveur alloue le même mot de passe à deux noms d'utilisateurs (USERNAME) différents doit être extrêmement faible, et les mots de passe DOIVENT être non devinables.

Le serveur utilise une approche possible est de construire le USERNAME comme:

USERNAME = <prefix,rounded-time,clientIP,hmac>

Où prefix est une chaîne de texte aléatoire (différente pour chaque demande de secret partagé), rounded-time est l'heure courante modulo en 20 minutes, clientIP est l'adresse IP de source d'où vient la demande de secret partagé, et hmac est un HMAC [13] sur prefix, rounded-time, et client IP, en utilisant une clé privée du serveur.

Le serveur contruit le mot de passe, qui est alors calculé comme :

password = <hmac(USERNAME,autrecléprivée)>

Avec cette structure, le nom d'utilisateur (USERNAME) lui -même, qui sera présent dans la demande de lien, et il contient l'adresse IP de source d'où vient

la demande de secret partagé. il est permet au serveur de satisfaire aux exigences spécifiées ci-dessus pour la construction de l'attribut REFLECTED-FROM. Le serveur peut vérifier que le nom d'utilisateur (USERNAME) n'a pas été falsifié, en utilisant le hmac présent dans le nom d'utilisateur (USERNAME).

# Programme

Programme travaille en deux parties. Première partie, c'est de découvrir les types de NATs entre client et serveur, deuxième partie, c'est de transmission les fichiers entre deux clients.

Dans mon programme, j'ai utilisé le projet de l'équipe Vivoda comme une librairie pour développer mon programme.

Dans la partie client, j'ai ajouté un attribut qui s'appelle ClienInfo pour tenir toutes les informations du client.

Dans partie serveur, j'ai implémenté une liste de client ClienInfoList pour contenir les clients qui ont enregistrés à serveur.

## *Découvert les types de NATs entre client et serveur*

Protocole STUN est le simple protocole, le but du protocole STUN est découvert les types de NATs. Protocole STUN est souvent une petite partie des applications pour découvrir des types de NATs.

STUN utilise l'algorithme pour découvrir les NATs dans le réseau

**Etape 1**

Le client STUN envoie une demande de lien à serveur avec la même adresse et le même port.

S'il y a une réponse de serveur, le client examine IP public et IP qu'il envoie

Si IP public est IP privé, le client sait qu'il n'y a pas de NAT entre client et serveur, le client fait deuxième étape.

Si IP public et IP privé sont différents, il y a NAT entre client et serveur, pour déterminer le type de NAT, le client retient IP public et le client fait troisième étape.

Si non, il n'est pas reçu une réponse, le client sait qu'il est bloqué UDP.

**Etape 2**

Le client STUN envoie une autre demande de lien à serveur avec la différente adresse et le différent port.

S'il y a une réponse de serveur, le client ouvre internet.

Si non, il n'est pas reçu une réponse, le client sait qu'il y a des symétriques pare-feux entre client et serveur.

**Etape 3**

Le client STUN envoie une autre demande de lien à serveur avec la différente adresse et le différent port.

S'il y a une réponse de serveur, client sait que le NAT entre client et serveur est NAT plein cône.

Si non, le client fait quatrième étape.

**Etape 4**

Le client STUN envoie une demande de lien à autre serveur mais avec la même adresse et le même port. Le client examine IP public.

Si IP public n'est pas changé, le client fait cinquième étape.

Si non, IP public est changé, le client sait que le NAT est symétrique.

**Etape 5**

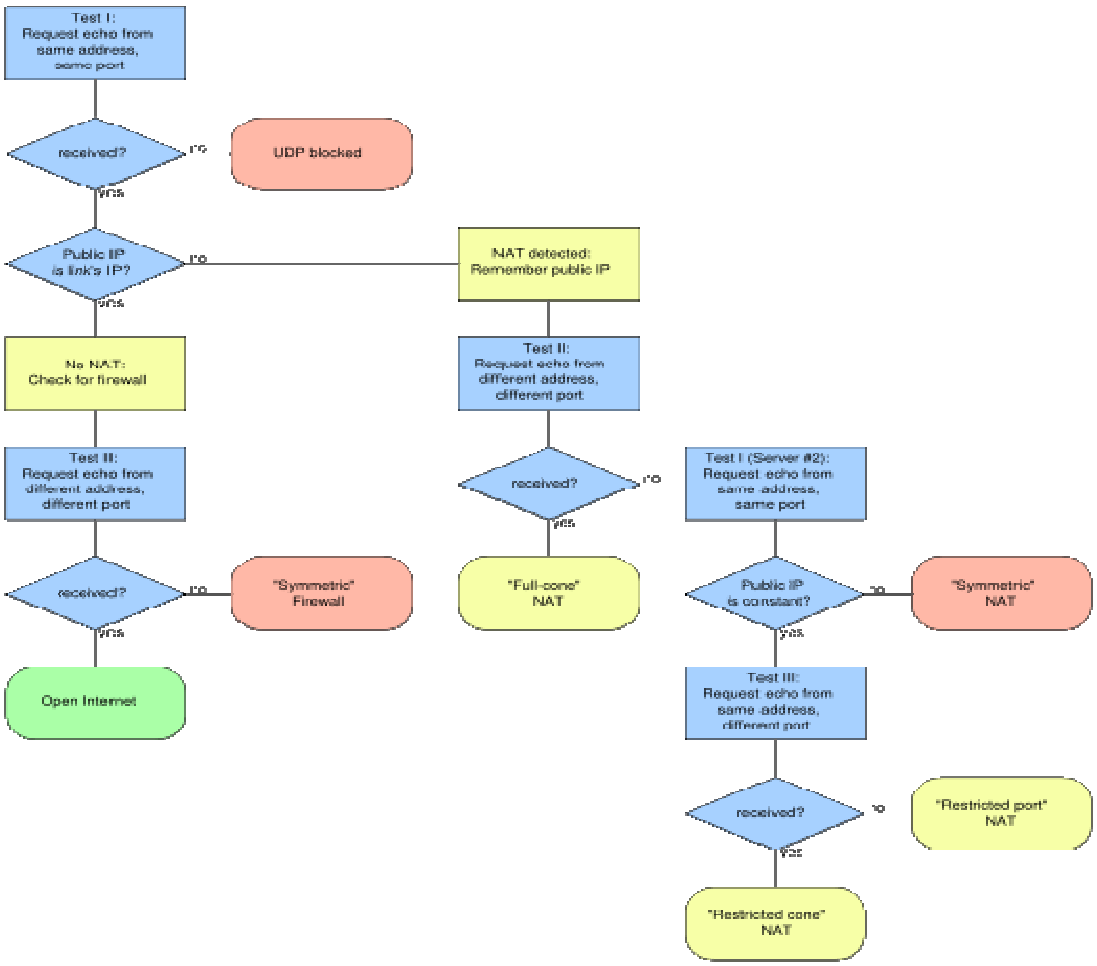
Le client STUN envoie une demande de lien à autre serveur mais avec la même adresse et le différent port.

S'il y a une réponse, le client sait qu'il y a NAT cône restreint.

Si non, le client sait qu'il y a NAT cône à restriction de port.

***Figure 2 : Algorithme de découvert des NATs entre client et serveur***

Src : http://en.wikipedia.org/wiki/STUN




Si on trouve les rouges figures, c'est – à – dire, on ne peut pas établir une connexion entre client et serveur. Si on trouve les jaunes et bleus figures, on peut établir une connexion entre client et serveur.

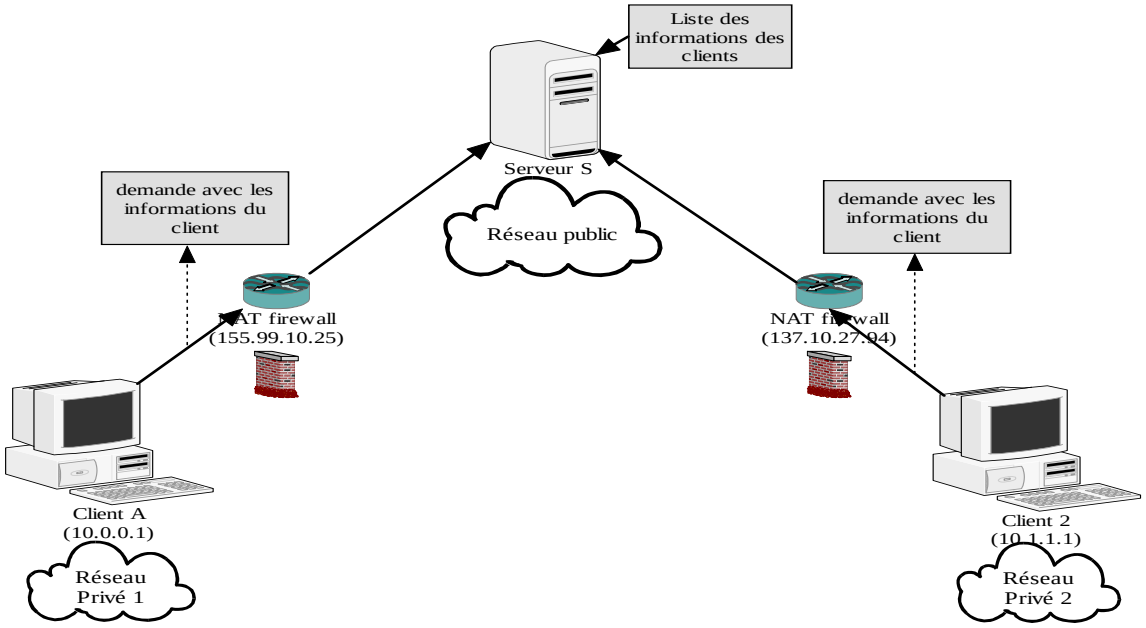
## *Transmission les fichiers entre deux client*

La deuxième partie, c'est de transmission des fichiers à travers les NATs entre deux clients. Je vais tester avec 4 types de NATs : Plein Cône NAT, Cône restreint NAT, Cône à restreint NAT et Symétrique NAT.

### Étape I : enregistrement

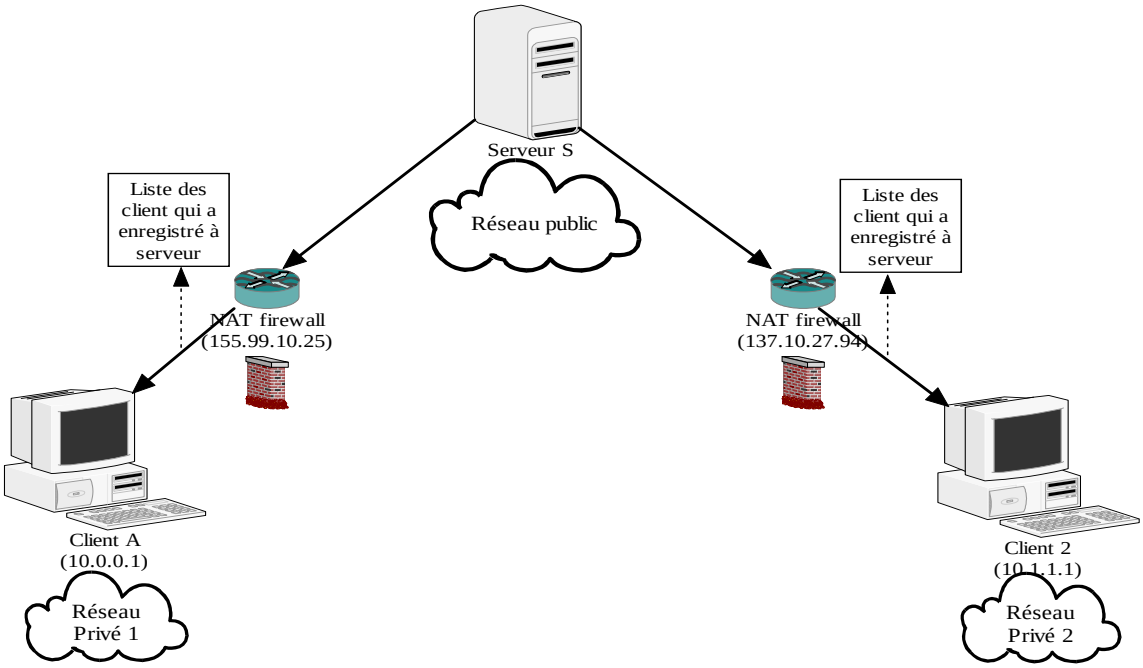
Tout d'abord, deux clients veulent communiquer ensembles, doivent enregistrer au serveur pour avoir des informations des autres clients. En envoyant une demande au serveur, le client A, par exemple, envoie la demande avec un

attribut ClientInfo qui contient tous les informations du client, puis le serveur copie les informations du client dans une liste des informations des client.



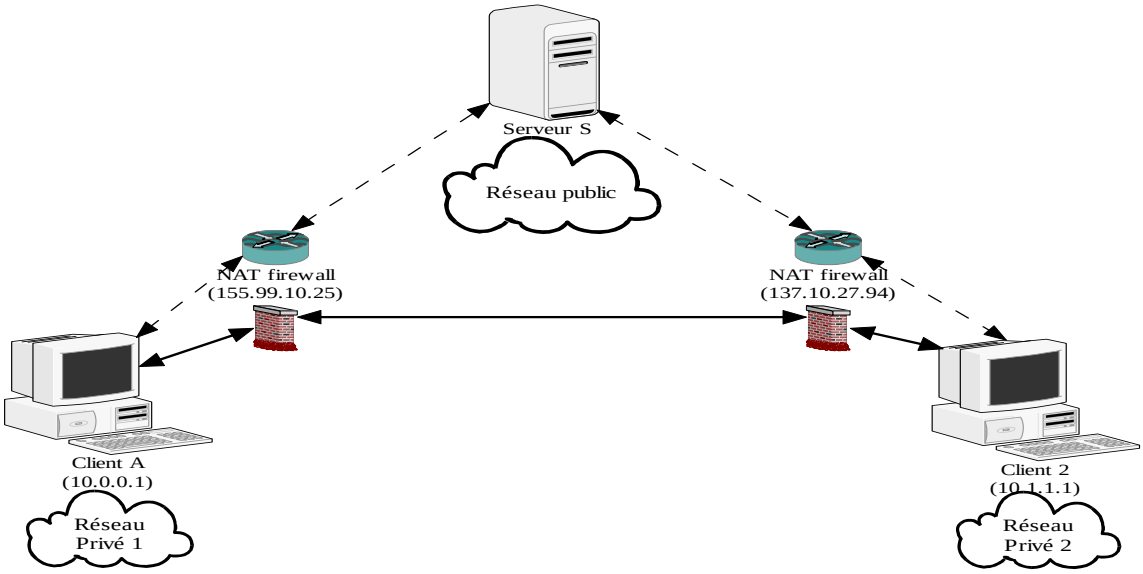

### Étape II : obtenu la liste de client

Après d'enregistrement, le serveur STUN envoie une réponse avec la liste de client au client. Par exemple, clientA a une liste de client, il peut choisir et envoyer des données à l'autre client.

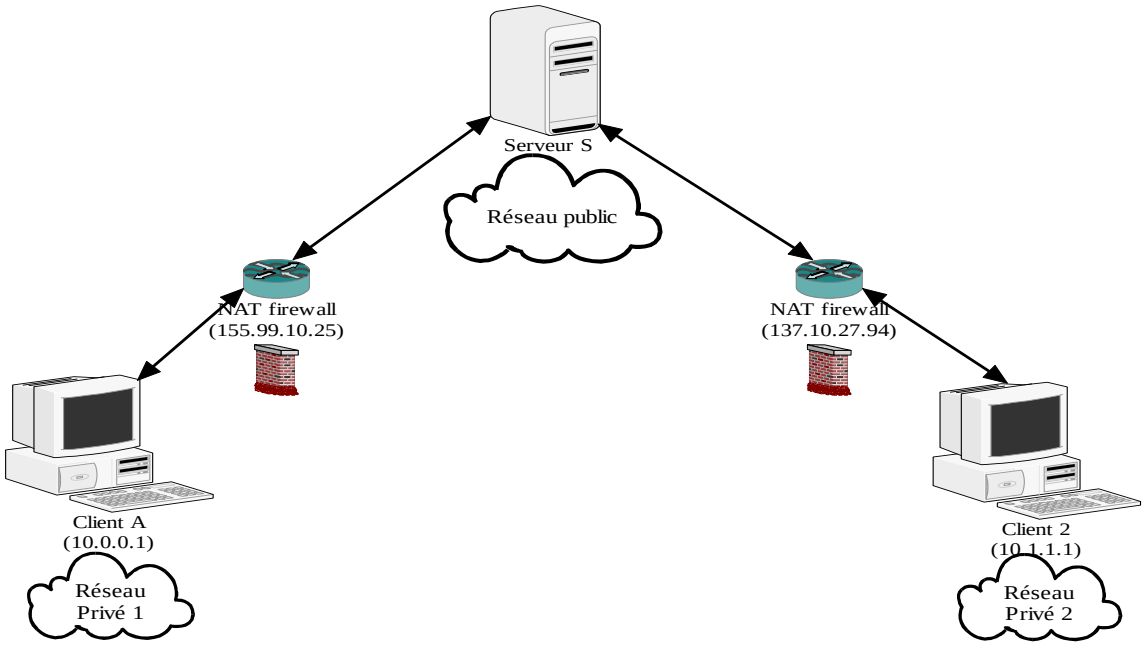



## Étape III : changement des données

On support que le clientA a une liste des clients, dans la liste des client il y a des informations comme : adresse et port IP. Client A peut choisir le client B dans la liste de client, puis il peut envoyer des données au client B.
Selon le type de NAT, si le NAT est plein cône NAT, les deux clients peuvent échanger des données directement, comme ci-dessous

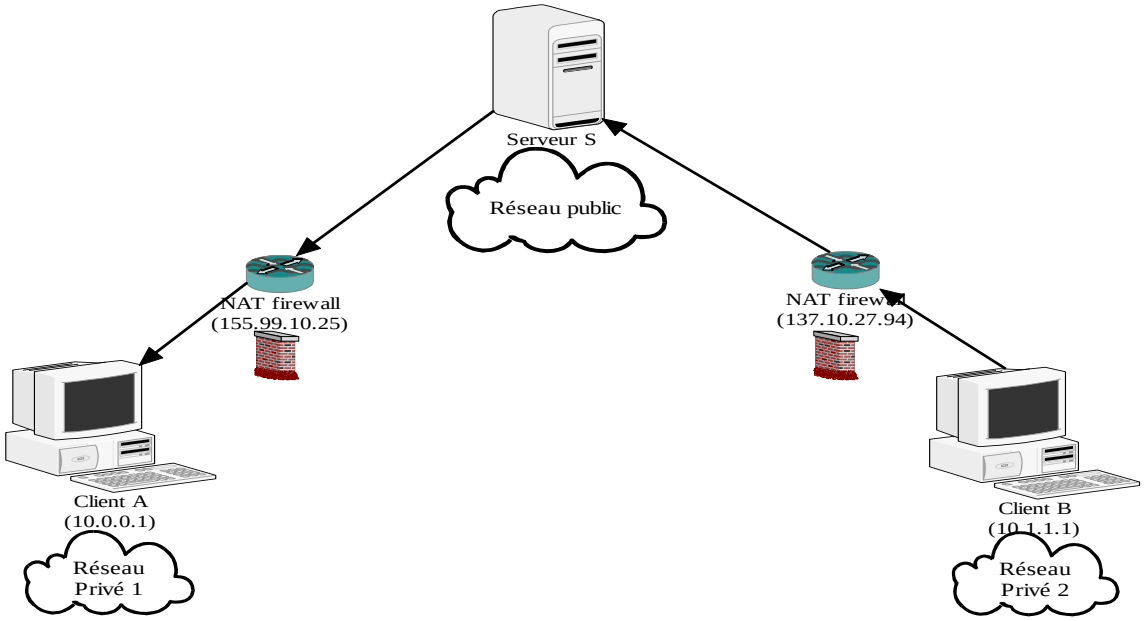


Avec les autres types de NAT, ce sont le NAT cône restreint, le NAT cône à restriction de port et le NAT symétrique, les deux clients peuvent échanger des données par le serveur, le serveur joue le rôle d'un intermédiaire, comme ci-dessous.




### Étape IV : terminaison d'échanger des données

Dans cette étape, clientB a reçu des données, puis il envoie une réponse pour confirmer que les données sont reçues par client B. En général, clientB envoie une réponse au serveur, puis serveur renvoie cette réponse au clientA, processus d'échanger des données entre deux clients est finis.




## *Résultat*

Tout d'abord, j'ai lancé le programme serveur sur le serveur de l'IFI. Le serveur est lancé comme la table ci-dessous, par défaut, le port 3478 et 3479 sont utilisés et deux interface de réseaux : interface 203.162.5.194 et alternate 127.0.0.1 (localhost) sont utilisés sur le serveur de l'IFI.

```
Running with on interface 203.162.5.194:3478 with alternate 127.0.0.1:3479
Stun server on 203.162.5.194:3478
Binding to interface 0xc205a2cb
Opened port 3478 with fd 3
Binding to interface 0xc205a2cb
Opened port 3479 with fd 4
Binding to interface 0x100007f
Opened port 3478 with fd 5
Binding to interface 0x100007f
Opened port 3479 with fd 6
**********************************************************************************
**********************************************************************************
```

Après lancement du serveur, le serveur va attendre les connexions des clients à l'extérieur.

Ici, par exemple j'ai lancé un client dans la salle de TP.

```
**********received on A1:P1
Got a request (len=34) from 192.168.102.140:31136
Received stun message: 34 bytes
ClientInfo = 0.0.0.0:31136
Request parsed ok
Encoding stun message:
Encoding ClientInfoList 0

**********
**********
```

Dans ce cas, la liste de client ClientInoList dans le serveur est 0 parce qu'il y a seulement un client, qui a enregistré à serveur.

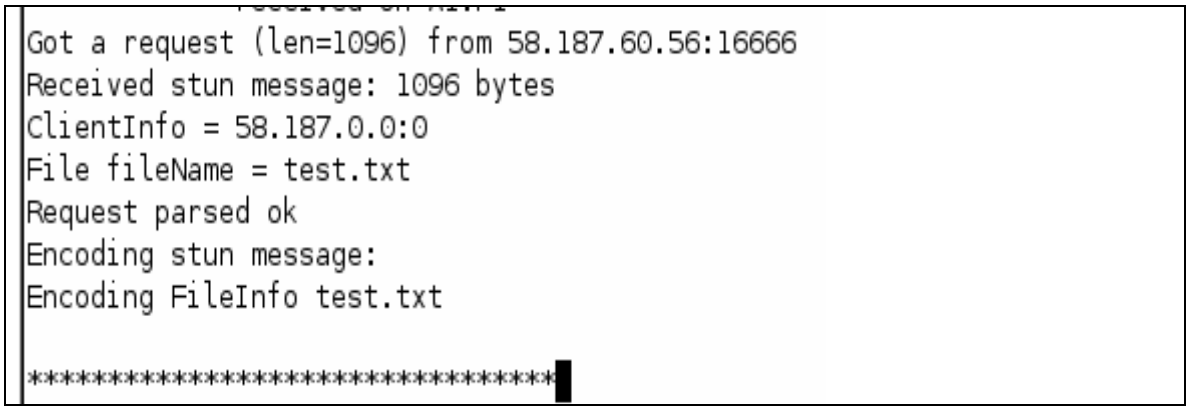
Maintenant, j'ai lancé un autre client à chez moi, la liste de client ClientInoList dans le serveur est 1, c'est – à – dire, il y a deux clients, qui ont enregistrés à serveur.

```
Got a request (len=34) from 58.187.60.56:1981
Received stun message: 34 bytes
ClientInfo = 0.0.0.0:1981
Request parsed ok
Encoding stun message:
Encoding ClientInfoList 1

**********
```

Maintenant, j'ai échangé des fichiers entre deux clients. Selon le type de NAT, on peut échanger les fichiers directement entre deux clients ou on doit échanger les fichiers traverser par un serveur.

Parce que à chez moi, le NAT n'est pas NAT plein cône, donc je dois utiliser un serveur pour échanger des fichiers entre deux clients.




Ici, j'ai envoyé un fichier test.txt à serveur, puis le serveur analyse le message et il sait qu'il doit renvoie ce fichier à quel client.

### *Evaluation des résultats*

Dans mon programme, il existe quelques limitations. La taille de fichier doit être petite, parce que j'ai empaqueté le fichier dans les messages du protocole STUN.

On ne peut pas envoyer les fichiers à plusieurs clients en même temps, car dans le mode de lancement du programme, on doit choisir chaque client pour recevoir les fichiers.

Le programme n'a pas d'interface graphique, donc il est un peu difficulté pour les utilisateurs.

Par contre, le programme a quelques avantages aussi. Le programme a été écrit en langage C/C++ donc il peut être implémenté sur plusieurs opérations.

## Conclusion

La traversée simple du protocole de datagramme d'utilisateur (UDP) à travers les traducteurs d'adresse réseau (NAT) est un protocole léger qui permet aux applications de découvrir la présence et le type des NAT et pare-feu entre eux et l'Internet public. Protocole STUN peut fournir aussi aux applications la capacité de déterminer les adresses IP public qui leur sont allouées par le NAT.

STUN fonctionne avec de nombreux NAT existants, mais il ne fonctionne pas avec NAT symétrique. Il permet à une grande d'applications de fonctionner sur l'infrastructure de NAT existante. L'idée principale du TIPE est de construire un système qui peut découvrir les types de NAT et échanger des fichiers entre deux clients dans deux réseaux privés différences.

# Références

[1] Bradner, S., "Key words for use in RFCs to indicate requirement levels" *(Mots clé à utiliser dans les RFC pour indiquer les niveaux d'exigence)*, BCP 14, RFC 2119, mars 1997.

[2] Dierks, T. et C. Allen, "The TLS protocol Version 1.0" *(Protocole TLS, version 1.0)*, RFC 2246, janvier 1999.

[3] Gulbrandsen, A., Vixie, P. et L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)" *(RR DNS pour la spécification de la localisation des services)*, RFC 2782, février 2000.

[4] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)" *(Suites de chiffrement pour norme de chiffrement avancée (AES) pour la sécurité de la couche transport (TLS))*, RFC 3268, juin 2002.

[5] Rescorla, E., "HTTP over TLS" *(http sur TLS)*, RFC 2818, mai 2000.

[6] Postel, J., "Internet Protocol" *(Le protocole Internet)*, STD 5, RFC 791, septembre 1981.

[7] Ferguson, P. et D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing" *(Filtrage à l'entrée du réseau : vaincre les attaques de déni de service qui utilisent le camouflage d'adresse de source IP)*, BCP 38, RFC 2827, mai 2000.

[8] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines"
*(Lignes directrices pour la conception d'applications faciles de traducteur d'adresse réseau*
*(NAT))*, RFC 3235, janvier 2002.

[9] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. et A. Rayhan, "Middlebox Communication Architecture and Framework" (*Architecture et cadre de travail des communications par boîtier de médiation)*, RFC 3303, août 2002.

[10] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. et E. Schooler, "SIP: Session Initiation Protocol" *(SIP :Protocole d'initialisation de session)*, RFC 3261, juin 2002.

[11] Holdrege, M. et P. Srisuresh, "Protocol Complications with the IP Network Address Translator" *(Difficultés du protocole avec le traducteur d'adresse réseau IP)*, RFC 3027, janvier 2001.

[12] Schulzrinne, H., Casner, S., Frederick, R. et V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications" *(RTP : un protocole de transport pour les applications en temps réel)*, RFC 1889, janvier 1996.

[13] Krawczyk, H., Bellare, M. et R. Canetti, "HMAC: Keyed-Hashing for Message Authentication" *(HMAC : hachage de clés pour l'authentification de message)*, RFC 2104,
février 1997.

[14] Kohl, J. et C. Neuman, "The kerberos Network Authentication Service (V5)" *(Le service
d'authentification de réseau Kerberos (Version 5))*, RFC 1510, septembre 1993.

[15] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. et T. Berners-Lee,
"Hypertext Transfer Protocol -- HTTP/1.1" *(Protocole de transfert hypertexte – http/1.1)*, RFC 2616, juin 1999.

[16] Baugher M., et al., "The secure real-time transport protocol" *(Protocole sécurisé de transport en temps réel)*, Travail en cours.

[17] Daigle, L., Editeur, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation" *(Considérations de l'IAB sur la fixation unilatérale de sa propre adresse (UNSAF) à travers la traduction d'adresse réseau)*, RFC 3424, novembre 2002.

[18] Huitema, C., "RTCP attribute in SDP" *(Attribut RTCP en SDP)*, Travail en cours.

[19] J.Rosenberg, J. Weinberger, C. Huitema et R. Mahy "STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)" *(STUN – Traversée simple du protocole de datagrammes d'utilisateur (UDP) à travers les traducteurs d'adresse réseau (NAT))*, RFC 3489, 2003