nexcom
SYSTEMS

La traversée de NAT en VoIP SIP

Best Current Practice

O. Gremaud

20 juin 2012

Table des matières

Table des figures

Introduction

Les NAT, en marge d'apporter une solution temporaire à la pénurie latente d'adresses IPv4, ont apporté leur lot de problèmes. Les protocoles de voix sur IP (VoIP ou Voice over Internet Protocol) sont particulièrement affectés, notamment SIP (*Session Initiation Protocol* [1]), SDP (*Session Description Protocol* [2]) et RTP (*Real-time Transport Protocol* [3]). Ce document fait l'état de la pratique actuelle à appliquer en priorité dans le cadre du déploiement d'une solution de traversée de NAT [1].

L'objectif de SIP est de récupérer l'ensemble des informations nécessaires à l'établissement d'une connexion entre deux terminaux. À partir de ces renseignements, les équipements apprennent quel chemin emprunter afin de contacter un destinataire spécifique. Il est alors possible d'utiliser SIP pour de nombreux usages, avec en tête les applications temps-réel (téléphonie, visiophonie ...) grâce à SDP (paramètres de connexion des flux RTP).

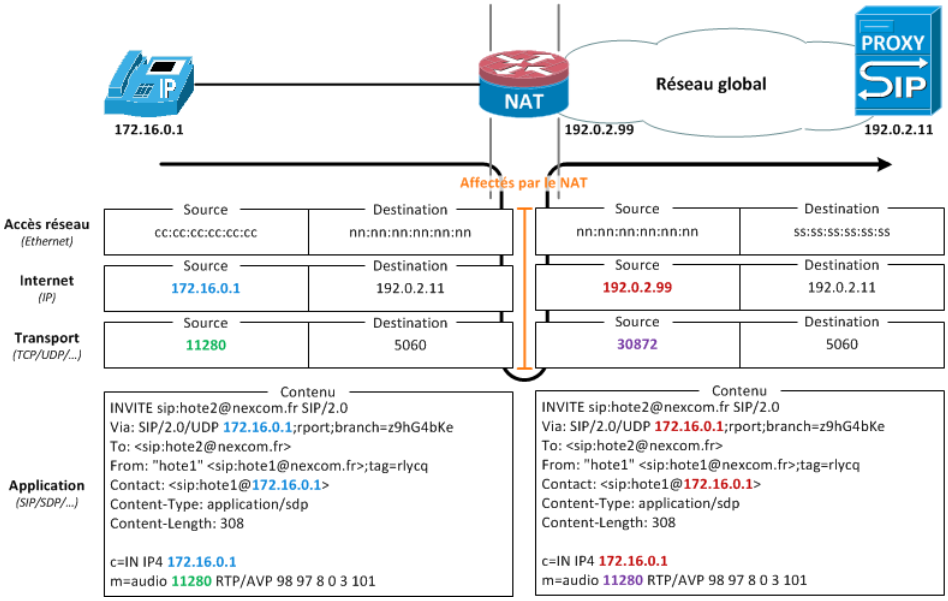


Fig. 1 – *Problématique générale des NAT*

La Figure 1 ci-dessus décrit, dans les grandes lignes, le problème qui va se poser avec la traversée de NAT. Ceux-ci ne travaillent en effet qu'au niveau des premières couches de la pile réseau. De ce fait, toutes les informations transportées à un niveau applicatif ne seront pas altérées, empêchant le bon fonctionnement dans l'établissement des sessions et des communications audio/vidéo [2].

Cette problématique, plus subtile qu'elle n'en a l'air, peut être fragmentée en de multiples aspects bien distincts pour lesquels des solutions dédiées existent. Toutefois, avant d'aborder ces éléments plus en détails, les NAT et leurs comportements doivent être définis de manière claire afin de pouvoir y apporter les solutions appropriées.

1. Si cette approche est ici traitée dans l'optique de la VoIP, elle ne se limite pas à ce cas de figure, et peut être étendue et/ou adaptée à d'autres protocoles (FTP, ...).
2. Les ALG (*Application Level Gateways*) tentent d'apporter une intelligence de niveau applicatif aux NAT. Leurs implémentations apportent toutefois plus de problèmes qu'elles n'en résolvent, et sont donc à proscrire.

Caractérisation des NAT

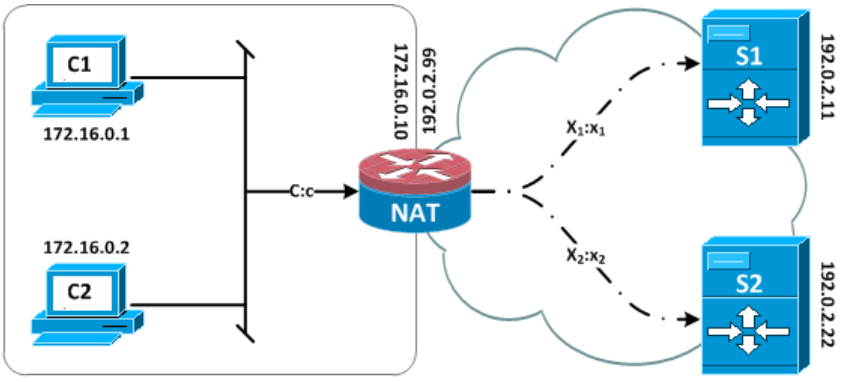
Pendant longtemps, aucune définition n'existait permettant de catégoriser le comportement des NAT. L'IETF a tenté, avec la première version de STUN [4], d'y apporter des définitions simples, connues actuellement sous le nom de terminologie "CONE" (*full cone*, *restricted cone*, *port-restricted cone* et *symmetric NAT*). Il s'est toutefois avéré que si ces définitions couvraient une majorité des NAT disponibles, il était impossible de définir l'ensemble de leurs comportements de manière aussi simple.

Une nouvelle définition, plus précise et donc à privilégier, a été créée dans ce but. Ces définitions, connues sous le nom de terminologie "BEHAVE" [5][6], traitent séparément chacune des fonctions d'un NAT (translation, filtrage ...) [3]. Trois aspects essentiels prennent une part importante en VoIP :

- Le type de translation, spécifiant le comportement du NAT pour le trafic sortant.
- Le type de filtrage, spécifiant le comportement du NAT pour le trafic entrant.
- La sélection du port externe, spécifiant la façon dont le port est choisi pour le trafic sortant.

Le type de translation

Deux catégories principales existent lorsqu'il est question du type de translation effectué. Une translation peut en effet être faite en fonction du destinataire (endpoint dependant) ou non (endpoint independant), comme décrit dans la Figure 2 ci-dessous.



Endpoint Independant		Pour un couple (C:c) donné, $(X_1:x_1) \equiv (X_2:x_2)$, quel que soit (S:s)
Endpoint Dependant	*Address only*	Pour un couple (C:c) donné, $(X_1:x_1) \equiv (X_2:x_2)$, si et seulement si S1 ≡ S2
	Address & Port	Pour un couple (C:c) donné, $(X_1:x_1) \equiv (X_2:x_2)$, ssi (S1:s1) ≡ (S2:s2)

Fig. 2 – *Type de translation/association*

Dans le cadre d'une translation indépendante du destinataire, un client C sera traduit de la même façon, qu'il tente de joindre le serveur S1 ou S2. Seul un changement au niveau de la source, que ce soit l'adresse IP ou le port, modifiera l'association faite par le NAT. Tant que le client utilise systématiquement le même couple adresse IP/port, il sera toujours traduit de la même façon par le NAT. Il est recommandé pour un NAT de

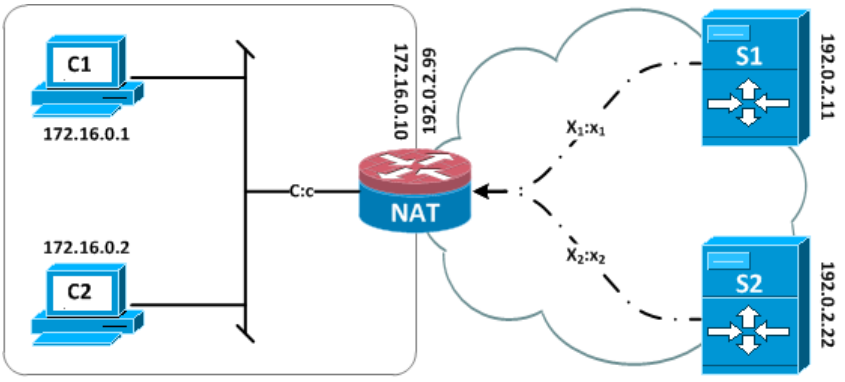
3. Pour une correspondance entre les deux nomenclatures, cf. Annexe [A] Correspondance RFC3489 et RFC4787, page 31.

disposer d'un tel comportement, qui facilite le travail des solutions de traversée de NAT d'offrir une efficacité accrue [4].

À l'opposé, une translation dépendante de la destination imposera au NAT de créer une nouvelle association dès que la destination change, que ce soit uniquement par l'adresse IP ou, dans une variante très restrictive, par le couple adresse IP/port. Ce type de comportement met à mal le fonctionnement des solutions de traversée de NAT, et il est dès lors à proscrire des implémentations actuelles de NAT.

Le type de filtrage

Dans les réseaux actuels, les NAT font généralement office d'équipement de sécurité en filtrant le trafic entrant à partir de permissions plus ou moins strictes selon le type de comportement considéré [5]. Lorsqu'une association est créée au niveau du NAT, une permission est également générée de manière à autoriser les réponses à la requête. De manière similaire aux translations, le filtrage peut également être défini selon qu'il est dépendant ou non du point de terminaison, ici l'origine, du trafic, comme décrit dans la Figure 3 ci-dessous.



Endpoint Independant		Pour (X:x) donnée, tout S → (C:c)
Endpoint Dependant	*Address only*	Pour (X:x) donnée, S → (C:c) possible ssi (C:c) → S au préalable
	Address & Port	Pour (X:x) donnée, (S:s) → (C:c) possible ssi (C:c) → (S:s) au préalable

Fig. 3 – *Type de filtrage*

La permission créée dans le cadre d'un filtrage indépendant va autoriser n'importe quelle destination à émettre du trafic sur le couple adresse IP/port définit lors de la translation. Un filtrage dépendant va, au contraire, filtrer en fonction de l'origine du trafic entrant. Dès lors, il devient nécessaire pour un client C d'autoriser une destination spécifique avant que celle-ci ne puisse utiliser l'association définie. Tout comme pour la translation, deux variantes existent, une dépendant uniquement de l'adresse IP d'origine, alors que l'autre, plus restrictive, considère également le port d'émission.

Il est intéressant de noter que les définitions BEHAVE permettent potentiellement toutes les combinaisons entre translation et filtrage, même les plus aberrantes. Définir une translation dépendante avec un filtrage indépendant serait par exemple un choix peu pertinent

4. Tel que défini dans [5].
5. Il est important de rappeler que translation et filtrage sont ici deux notions parfaitement distinctes.

et illogique, en plus de pouvoir soulever des situations étranges, avec par exemple du trafic pouvant entrer sur un port, mais devant sortir sur un autre. . .Si ces deux aspects sont traités séparément pour plus de précision, ils demeurent néanmoins intimement liés.

La Figure 4 définit deux combinaisons cohérentes recommandées par l'IETF. Comme discuté dans la section précédente, la translation doit être de type indépendante, alors qu'il est possible d'utiliser soit un filtrage indépendant, dans une optique de transparence, ou dépendant de l'adresse IP, si la sécurité est un point essentiel du réseau [6].

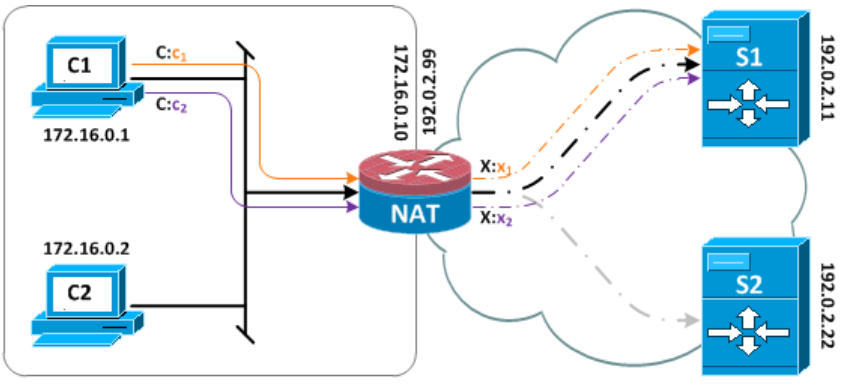
Table de cohérence		Filtrage		
		Indép.	Adresse	A+P
Association	Indép.	R *(Transp.)*	R *(Sécurité)*	X
	Adresse		X	X
	A+P			X

R : combinaison recommandée (RFC 4787)
X : combinaison cohérente (restrictive)

Fig. 4 – *Consistence des définitions*

Le choix du port

Ce dernier élément a potentiellement un impact important sur le fonctionnement d'une communication VoIP. En effet, les flux multimédias (RTP/RTCP) peuvent être soumis à des règles relativement strictes (parité, contigüité ...). Il est dès lors essentiel que les NAT ne perturbent pas ces comportements, tel que décrit dans la Figure 5 ci-dessous.



Parité	$c_1 \% 2 \equiv x_1 \% 2$ c_1 pair $\equiv x_1$ pair (et par corollaire c_1 impair $\equiv x_1$ impair) *OU*
Adjacence	$c_2 \equiv c_1 + 1 \Leftrightarrow x_2 \equiv x_1 + 1$

Fig. 5 – *Choix des ports*

Les NAT doivent en premier lieu tenter de préserver la parité du port reçu. Il est aisé pour un NAT d'implémenter ce type de comportement en testant le reste de la division par deux (modulo). La majeure partie des NAT actuels implémentent ce comportement.

Le second point est par contre bien plus complexe puisqu'il demanderait, pour être traité par le NAT, que ce dernier soit capable de détecter le type de trafic qui le traverse ce qui, nous l'avons discuté précédemment, n'est pas possible. La contigüité des ports RTP et RTCP ne peut être garantie par le NAT. Celui-ci ne peut qu'assigner les ports de manière séquentielle, permettant dès lors aux ports RTP et RTCP d'être adjacent si la source a émis sa requête de la sorte.

6. Il est hautement recommandé de bannir les variantes les plus restrictives (adresse IP et port) de ces deux aspects, tant cette approche met à mal la traversée de NAT (cf. ICE, page 14).

Traversée de NAT : *Best Current Practice*

Afin de fournir une approche complète, l'IETF a fractionné la problématique, rendant leur traitement plus ciblé et donc potentiellement plus aisé, voire, dans certains cas, également plus complet. L'ensemble de ces solutions, en développement parfois depuis plusieurs années, a fini par créer un *framework* complet de traversée de NAT, couvrant l'intégralité de la problématique. La plupart de ces solutions sont le fruit d'une réflexion extrêmement poussée, basée sur des raisonnements complexes, mais également imaginés dans un souci d'évolutivité et de polyvalence qui en font un choix idéal à privilégier pour une interopérabilité optimale.

La gestion de la traversée de NAT en VoIP doit tout d'abord être séparée en deux éléments bien distincts : la gestion de la signalisation (SIP/SDP), puis la gestion des flux multimédias (RTP).

Signalisation

Lors de la définition du protocole SIP, la traversée de NAT n'a jamais été abordée. Dès lors, que ce soit au niveau des réponses ou des requêtes entrantes (invitations), il devient nécessaire de trouver des méthodes pour acheminer les messages correctement.

Les réponses SIP

Par défaut, les réponses SIP sont renvoyées à l'émetteur de la requête à l'aide des entêtes Via. Toutefois, nous l'avons vu dans la Figure 1, ces informations ne sont, après avoir traversé un NAT, plus routables depuis le réseau global. La requête est reçue depuis l'adresse IP et le port d'émission du NAT, qui sont différents de celui du client.

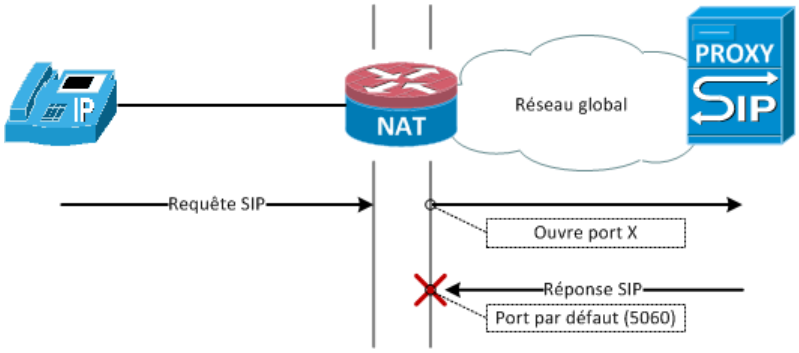


Fig. 6 – *Echec de réponse SIP*

Puisque le NAT a créé une permission sur un couple adresse IP/port donné, il est impératif, en UDP (TCP le fait par défaut), de réutiliser la même connexion pour les réponses. Appelée réponse symétrique, cette approche se sert du paramètre *received*, initialement prévu pour acheminer les réponses sans avoir besoin d'effectuer des requêtes DNS superflues, pour stocker l'adresse IP du NAT. Toutefois, la Figure 6 ci-dessus montre que sans récupération du port, la réponse sera rejetée car elle est, par défaut, envoyée sur le port 5060. Un nouveau paramètre a été ajouté afin de palier à ce problème et permettre de répondre effectivement sur l'exacte même connexion que la requête : *rport* [7].

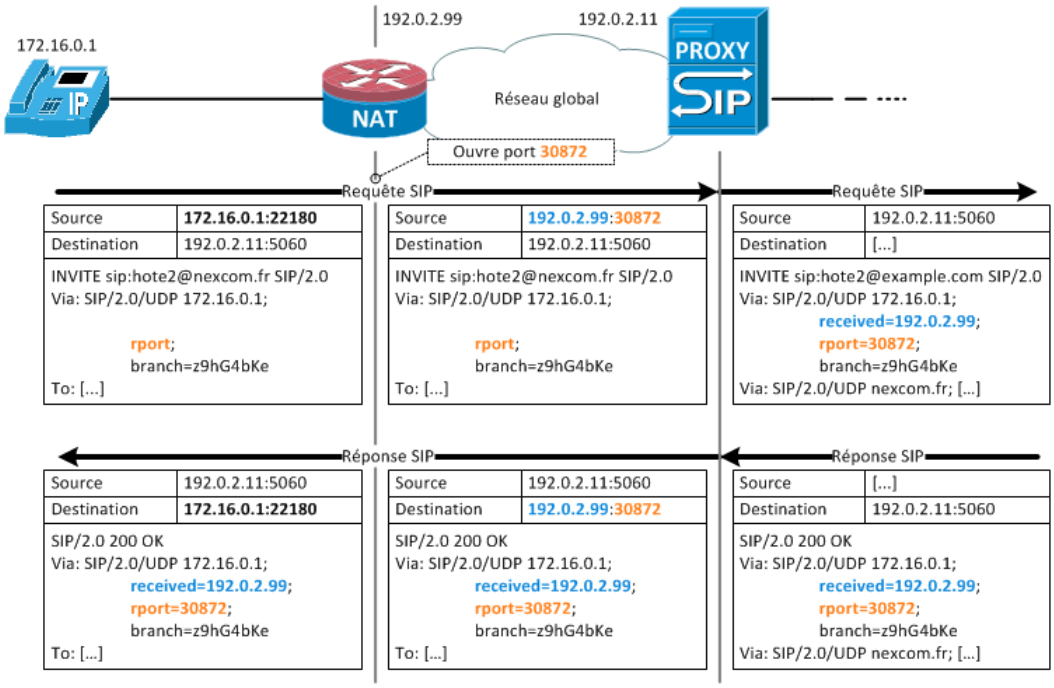


Fig. 7 – *Réponse symétrique*

Ce paramètre est spécifié, sans valeur, par le client souhaitant l'utiliser. Sur réception, le premier saut (un proxy par exemple) ou le destinataire remplit le champ avec la valeur de port reçue, et transmet si nécessaire la requête plus loin sur le réseau. Lorsqu'une réponse est envoyée, celle-ci transitera à l'aide des informations contenues dans le *Via*, correspondant désormais au couple adresse IP/port du NAT sur lequel une permission existe. A noter que TCP n'est pas soumis à de telles contraintes : la connexion créée par la requête est systématiquement réutilisée.

Les invitations SIP

De manière similaire aux réponses, une requête entrante (invitation) ne peut pas traverser les NAT puisque l'adresse et port du NAT sont inconnus. Par défaut, comme les réponses, les requêtes entrantes sont acheminées sur le port 5060...qui n'est toujours pas ouvert sur le NAT !

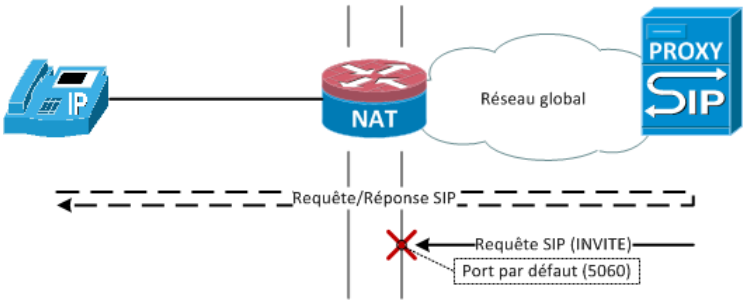


Fig. 8 – *Echec d'invitation SIP*

Ce problème est dû au fait que l'adresse du *Contact* ne pointe pas vers le couple adresse

IP/port du NAT. Le principe des réponses symétriques pourrait être étendu à ce cas de figure (cette "extension" est fréquente dans les implémentations actuelles de traversée de NAT), mais l'intelligence, en SIP est prévue pour être placée au niveau des terminaux, pas de l'infrastructure. Or, une telle utilisation de la réponse symétrique nécessite des routines plus complexes sur les proxys, ce qui va à l'encontre de la conception du protocole.

Le principe sous-jacent demeure toutefois similaire, mais les mécanismes associés, connus sous le nom *SIP-OUTBOUND* [8] rendent l'ensemble plus complet.

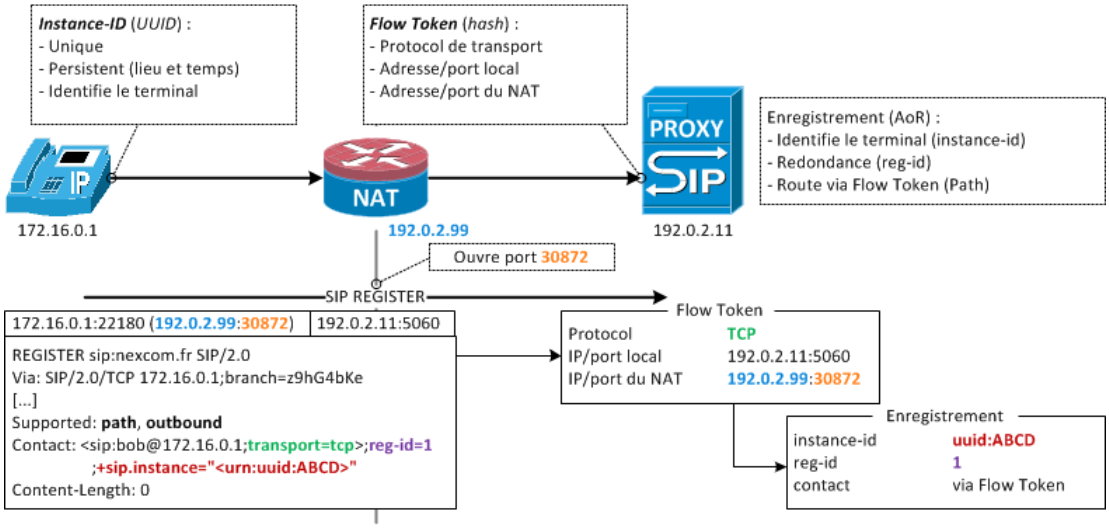


Fig. 9 – *Contact et Flow Token*

La Figure 9 ci-dessus donne une représentation simplifiée du mécanisme. Le client spécifie deux nouveaux attributs au sein de son *Contact* :

– *+sip-instance* : valeur identifiant de manière unique un agent [7].
– *reg-id* : identifie un enregistrement auprès d'un domaine.

Ces paramètres, couplés à une infrastructure compatible, vont indiquer au premier saut de constituer un ticket de connexion (*flow token*) qui contient l'ensemble des informations nécessaires afin de contacter l'abonné. Le contact reste inchangé, et est stocké en l'état sur le registrar. Seul le *path* est ajouté de manière à router les requêtes en conséquence à partir du *flow token*. L'intégralité des informations nécessaires sont stockées dans ces tickets :

– Le protocole de transport : TCP, UDP, ...
– Le couple adresse et port utilisé par le NAT pour émettre la requête. Ce couple est réutilisé pour transmettre les requêtes entrantes.
– Le couple adresse et port par lequel la requête a été reçue par le proxy, permettant de gérer les proxys disposant de multiples interfaces.

Ces deux paramètres permettent d'envisager des topologies plus complexes, offrant une redondance de lien entre le client et le registrar, tout en permettant une mobilité ac-

7. La notion d' "agent" dans le protocole SIP n'a jamais été définie clairement. Un agent SIP est un terminal, comme par exemple un téléphone IP physique ou un softphone, disposant d'une implémentation complète du protocole SIP. Un téléphone physique ne représente toujours qu'un unique agent, même s'il dispose de plusieurs lignes pouvant s'enregistrer sur de multiples registrar. À l'opposé, de multiples instances concurrentes d'un même softphone peuvent représenter de multiples agents différents.

crue des terminaux, qui peuvent désormais circuler et se réenregistrer sans surcharger inutilement l'infrastructure en changeant, par exemple, de NAT, mais simplement en renouvelant leurs enregistrements.

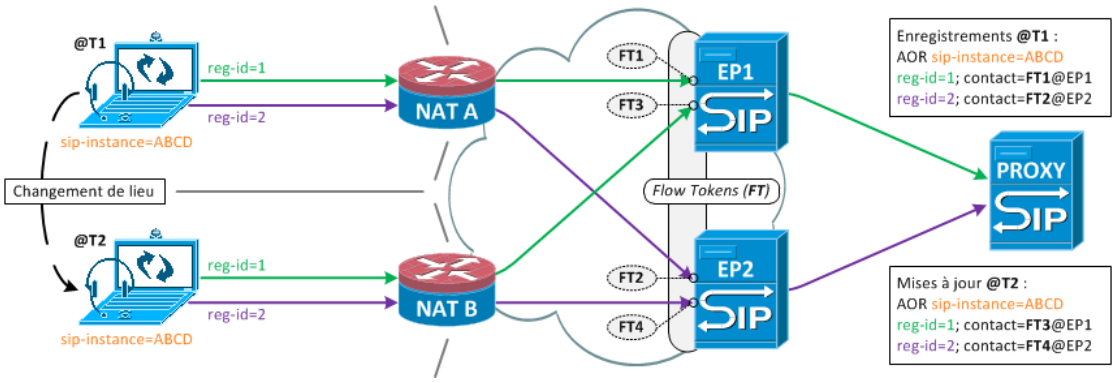


Fig. 10 – *Topologie OutBound complexe*

Cette figure décrit un fonctionnement plus complexe, dans le temps et l'espace, d'un abonné se connectant via de multiples liens différents. Il est intéressant de noter ici les informations stockées par le proxy de routage puisque seuls les *flow token* sont enregistrés, ces mêmes tickets qui sont par la suite transmis sur le reste du réseau . Toutefois, afin d'éviter qu'un nouvel enregistrement ne se crée, il est essentiel que l'agent utilise systématiquement les mêmes *reg-id* afin de renouveler les enregistrements.

Dans le cadre du *Contact*, les translations du NAT doivent être maintenues pour toute la durée de la session. Cette problématique ne se posait pas pour les réponses, celles-ci survenant peu de temps après la requête dont elles découlent. Mais il est, dans le cas présent, parfaitement envisageable qu'aucune requête SIP n'entre ou ne sorte au cours d'une communication multimédia. Des méthodes de *keepalive* sont proposées au travers de *ping*.

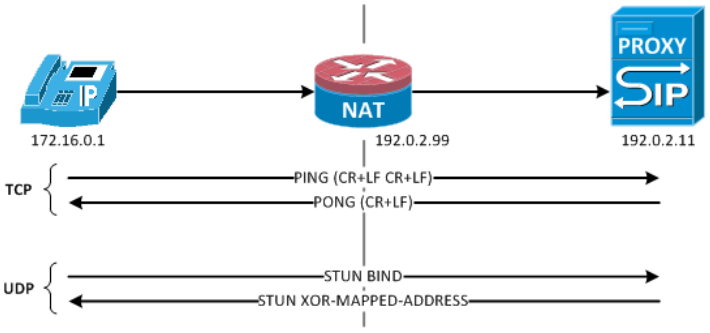


Fig. 11 – *Keepalives TCP/UDP*

Dans le cadre de protocoles de transports orientés connexion (TCP, TLS ...), le mécanisme bien connu du ping pong est utilisé, envoyant un double retour chariot (CR+LF) au proxy qui répond instantanément avec un simple CR+LF.

Pour UDP toutefois, si ce mécanisme fonctionne également, il n'existe aucun mécanisme de retransmission permettant d'assurer le fonctionnement du *keepalive*. Le protocole STUN est ici utilisé, envoyant, à partir du même port source et vers la même destination, une requête STUN à destination de l'*outbound proxy* qui répondra en conséquence [8].

Une réponse différente de celles reçues précédemment réinitialise la connexion (les informations de l'enregistrement ne sont plus correctes).

Flux multimédias

Tout comme pour la signalisation, la gestion des flux multimédias est fragmentée en de multiples problèmes mieux ciblés.

Direction des flux

A l'origine, aucun mécanisme de transmission bidirectionnel n'était prévu pour les communications en temps réel. Il était toutefois beaucoup plus simple, tant pour les implémentations que pour les firewalls et autres NAT, d'émettre et de recevoir les flux RTP/RTCP par la même connexion, par le même couple adresse/port.

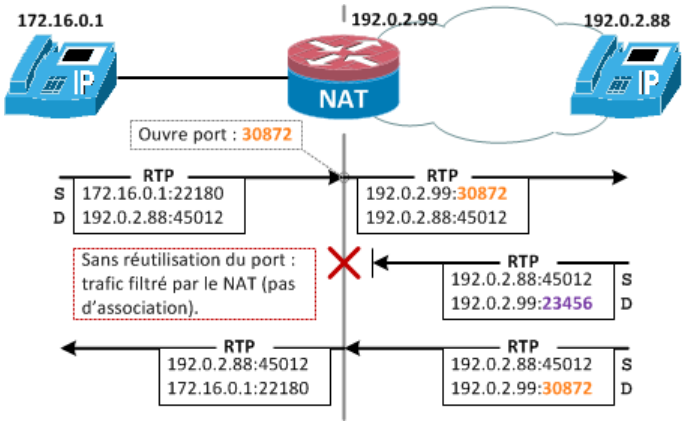


Fig. 12 – *RTP symétrique*

Par cette simplification de l'infrastructure, ce comportement est majoritairement implémenté depuis de très nombreuses années. Il n'a toutefois été formellement énoncé que relativement récemment [9]. Réutilisant les mêmes informations de connexion, la traversée des NAT s'en trouve également grandement facilité.

Contigüité des ports RTP/RTCP

Par conception, si RTP utilise un port K, son protocole de contrôle (RTCP) utilisera automatiquement le port supérieur direct $K + 1$. Cette contigüité n'est toutefois pas garantie lors de la traversée d'un NAT qui ne connaît pas le type de trafic qui le traverse. Il existe deux approches alternatives permettant de s'affranchir de ce comportement par défaut, et par extension de garantir un fonctionnement optimal de RTCP.

La Figure 13 décrit le fonctionnement de ces deux solutions. La première d'entre elles propose de pouvoir annoncer sur quel port le flux RTCP est attendu. Sous la forme de l'attribut SDP *a=rtcp <port>*, il permet de garder une rétrocompatibilité accrue (par

8. Les retransmissions sont gérées par le protocole STUN (cf. STUN, page 10).

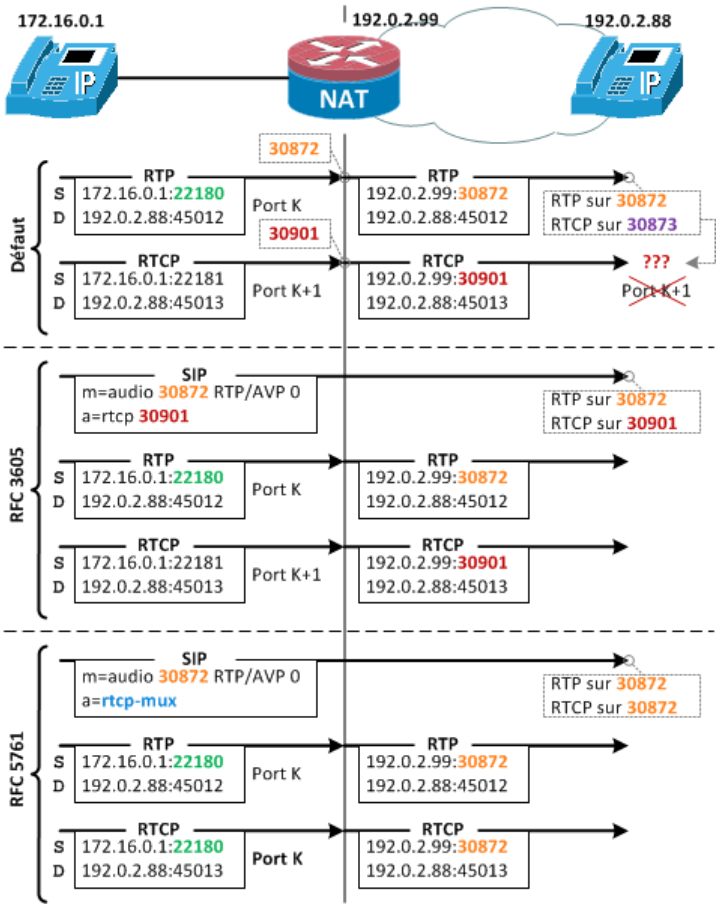


Fig. 13 – *Contigüité RTP/RTCP*

rapport à une annonce au sein de la ligne média *m=*), un destinataire non compatible ignorant simplement l'attribut inconnu.

La seconde méthode s'affranchit d'un certain nombre de requêtes (STUN/TURN [9] notamment) servant à déterminer le port RTCP à annoncer en multiplexant les flux RTP et RTCP sur le même port. Également sous la forme d'un attribut SDP (*a=rtcp-mux*), une destination incompatible ignorera cet attribut et annoncera par exemple son flux RTCP avec la méthode précédente.

Il est fréquent de trouver, à l'heure actuelle, des implémentations de la première solution, la seconde étant un peu plus rare, mais aussi plus intéressante du point de vue de l'infrastructure puisqu'ayant un impact moindre sur le réseau (moins de requêtes, moins de ports utilisés ...).

STUN

Le protocole STUN (*Simple Traversal of UDP throught NATs* [4] ou, dans sa seconde version : *Session Traversal Utilities for NATs* [10]) est une des solutions de traversée de NAT les plus anciennes. Plutôt que d'être une solution à part entière (*standalone*), elle

9. Voir ci-après pour plus d'informations.

sert désormais de fondation à des solutions exhaustives, grâce à sa polyvalence et sa modularité.

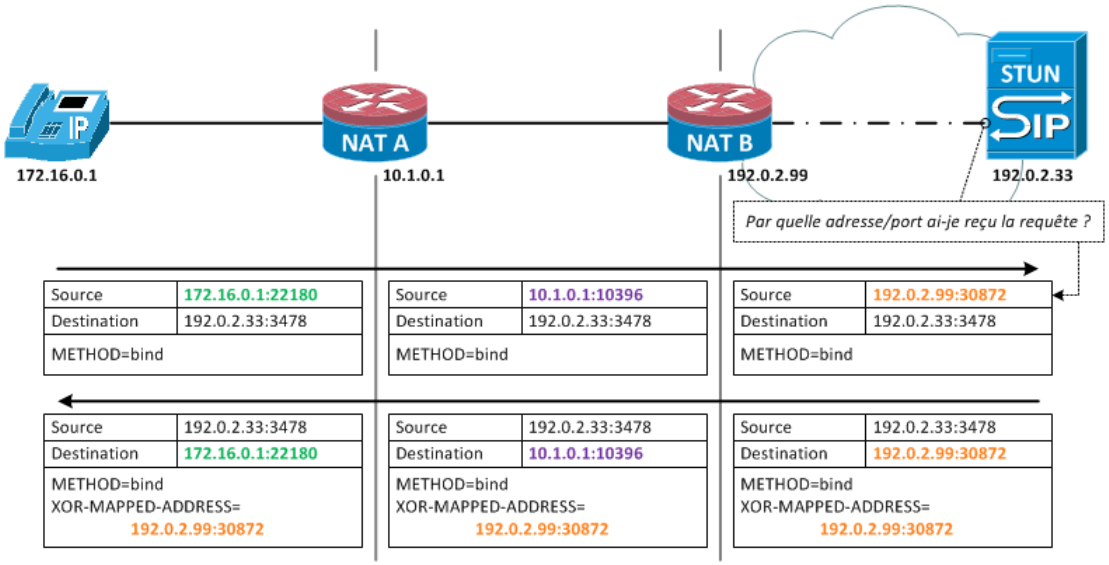
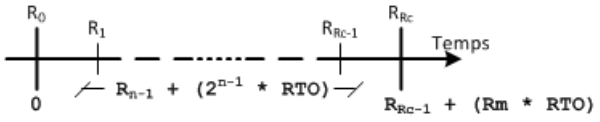


Fig. 14 – *Mécanismes de STUN*

Le principe est simple : situé sur le réseau public (ce point est essentiel), le serveur STUN récupère l'adresse et le port par lesquels la requête a été reçue, et renvoie l'ensemble sous une forme masquée à l'aide d'un *magic cookie* (0x2112A442 pour une adresse IPv4 et 0x2112 pour le port) afin de pouvoir traverser sans encombre d'éventuelles ALG. Le client peut dès lors modifier à la source les couples adresse/port nécessaires pour établir la communication multimédia.

Ce protocole ne se contente toutefois pas de cette information, et introduit un mécanisme de retransmission, inconnu d'UDP, permettant de s'assurer de la bonne marche de certaines requêtes. Il se repose sur l'algorithme de Karn [11], utilisé également pour des protocoles orientés connexion, et fonctionne sur base des formules exposées dans la Figure 15 ci-après.



Valeur	Défaut	Description
RTO	500ms	Timer de retransmission
Rc	7 → n = [0;Rc]	Compteur de retransmission
Rm	16	Multiplicateur final (Rc maximum)

Fig. 15 – *Retransmissions STUN (algorithme de Karn)*

STUN propose par défaut des timers longs, amenant les retransmissions jusqu'à près de 40 secondes. Ces valeurs peuvent (et doivent dans certains cas) être configurées en fonction de la situation, notamment en VoIP où une telle attente est inacceptable [10].

Pour finir, STUN propose deux mécanismes d'authentification, un sur le long terme (permanent) et un second sur le court terme (durée de la session), qui, s'ils ne présentent

10. Cf. ICE, page 14 pour plus d'informations.

aucun intérêt pour STUN directement, s'avèreront très utiles pour deux protocoles découlant de STUN : respectivement TURN et ICE. Le but de cette authentification n'est pas de chiffrer les données transmises (d'autres solutions existent pour cela), mais bien d'en assurer l'intégrité.

TURN

Par rapport aux différents comportements de NAT, il est possible de remarquer que les mécanismes de STUN sont insuffisants dans certains cas [11]. Dès qu'un NAT propose des translations dépendantes, il y a un risque que STUN ne fonctionne plus correctement. TURN (*Traversal Using Relays around NATs* [12]) est une extension à STUN proposant de faire transiter les flux multimédias par un relai, approche qui adresse effectivement les cas non couverts par STUN. Le serveur TURN devient alors la destination des flux multimédias pour chacune des parties.

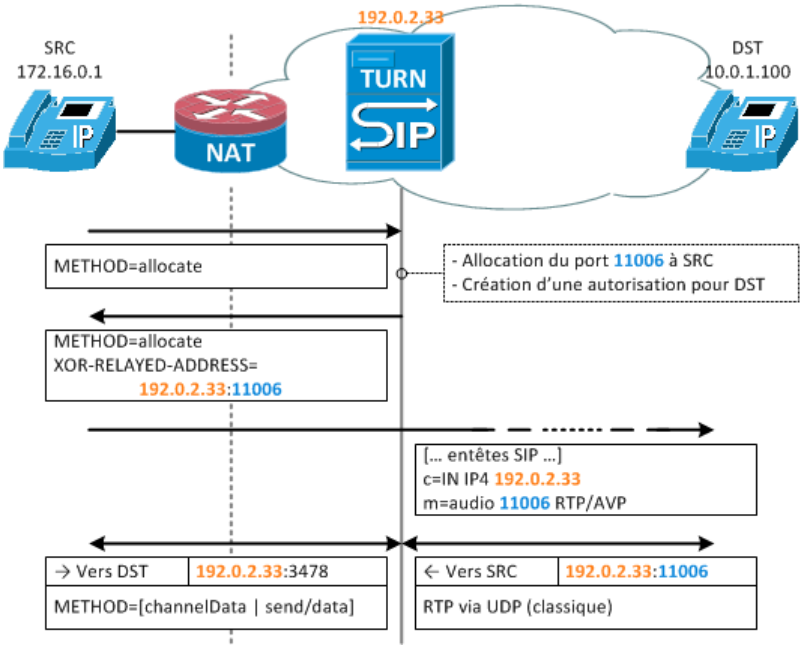


Fig. 16 – *Mécanismes de TURN*

Le but ici est d'effectuer des réservations auprès du serveur TURN qui sera par la suite utilisée pour les communications. Une méthode différente est utilisée ici, qui renvoie cette fois-ci, en sus de l'adresse publique mentionnée précédemment pour STUN, une adresse relayée qui peut également être utilisée au sein du SDP pour préciser la connexion à utiliser.

L'action de relayer des flux audio, et pire encore vidéo, s'avère très coûteux en ressources pour le serveur. Limiter son utilisation à des abonnés spécifiques, après rémunération ou non, devient alors une priorité afin d'éviter tout abus. TURN utilise l'authentification STUN à long terme pour vérifier la validité d'un utilisateur donné. L'objectif n'est pas ici de chiffrer les données (d'autres méthodes existent pour cela), mais bien de prévenir

11. Cf. les exemples d'ICE pour plus d'informations, page 23.

d'éventuelles attaques tentant de pirater le flux entre l'abonné et le serveur [12].

Ce protocole fonctionne de manière très similaire à un NAT : après avoir réservé un couple adresse IP/port, le client crée des permissions qui vont permettre à ses destinataires de le contacter. Ces permissions doivent être créées de manière explicite, à l'aide de méthodes dédiées, mais peuvent être librement interverties. Celles-ci sont décrites dans la Figure 17 ci-dessous. Si le destinataire peut communiquer avec le serveur TURN par la suite de manière classique (en UDP), le client TURN émet ses données dans des formats spécifiques, dédiés à TURN.

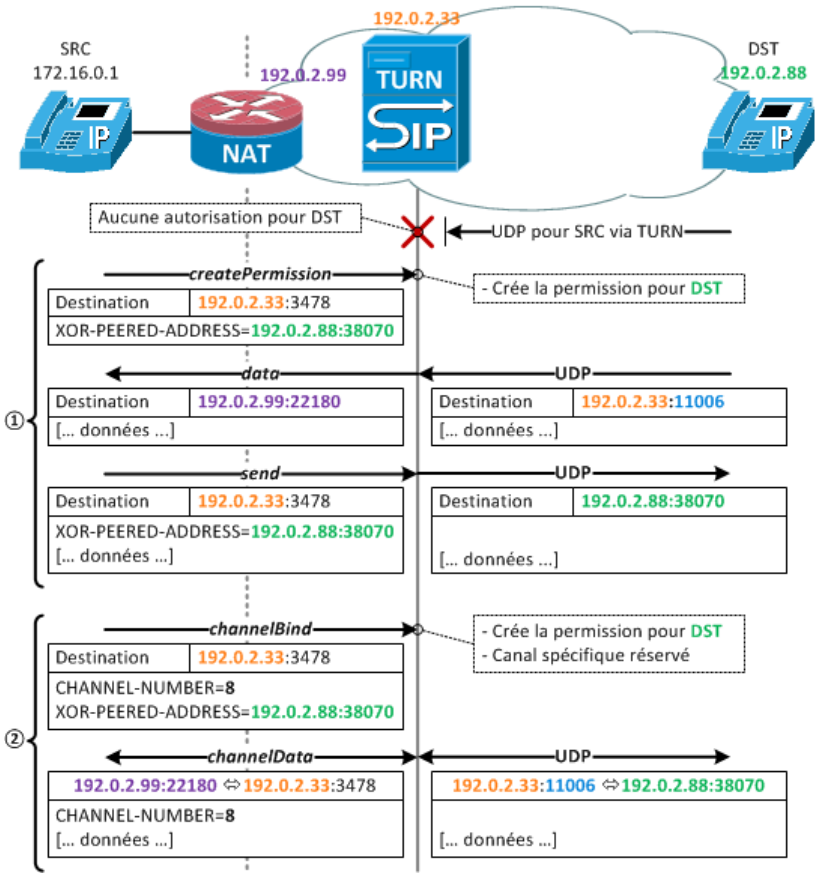


Fig. 17 – *Permissions TURN*

1. *CreatePermission* crée une permission pour 5 minutes. Dans ce mode, le client communique généralement avec le serveur à l'aide des indications *Send/Data* qui spécifie systématiquement le destinataire.
2. *ChannelBind* crée, pour sa part, une permission pour 10 minutes, en créant un tunnel plus permanent. Ce tunnel est réutilisé pour transmettre les données, sans avoir besoin de préciser le destinataire, allégeant la charge du réseau.

En tant qu'extension de STUN, ce protocole couvre beaucoup plus de cas de figure différents, le rendant du même fait sensiblement plus efficace. Toutefois, il demeure encore une interrogation majeure : comment décider quelle adresse publique utiliser : celle du

12. Un exemple détaillé d'authentification se trouve en Annexe [B] Authentification TURN, page 31.

NAT, ou celle du serveur relai ? La réponse à cette question est apportée par un dernier protocole, ICE.

ICE

Cette dernière approche, *Interactive Connectivity Establishment* [13], combine les capacités de STUN et TURN, et y ajoute des aspects de recherche automatique de connectivité. L'objectif d'ICE tient en peu de mots : déterminer la meilleure connexion possible à utiliser pour chaque flux multimédia entre deux terminaux. Son fonctionnement passe par cinq étapes distinctes (Figure 18) :

1. La récupération de candidats.
2. L'échange des candidats via la signalisation (SIP).
3. Le test d'une connectivité pour chaque couple de candidats (paires).
4. La validation et choix de la meilleure paire à utiliser.
5. L'établissement de la communication en elle-même sur base de la paire nominée.

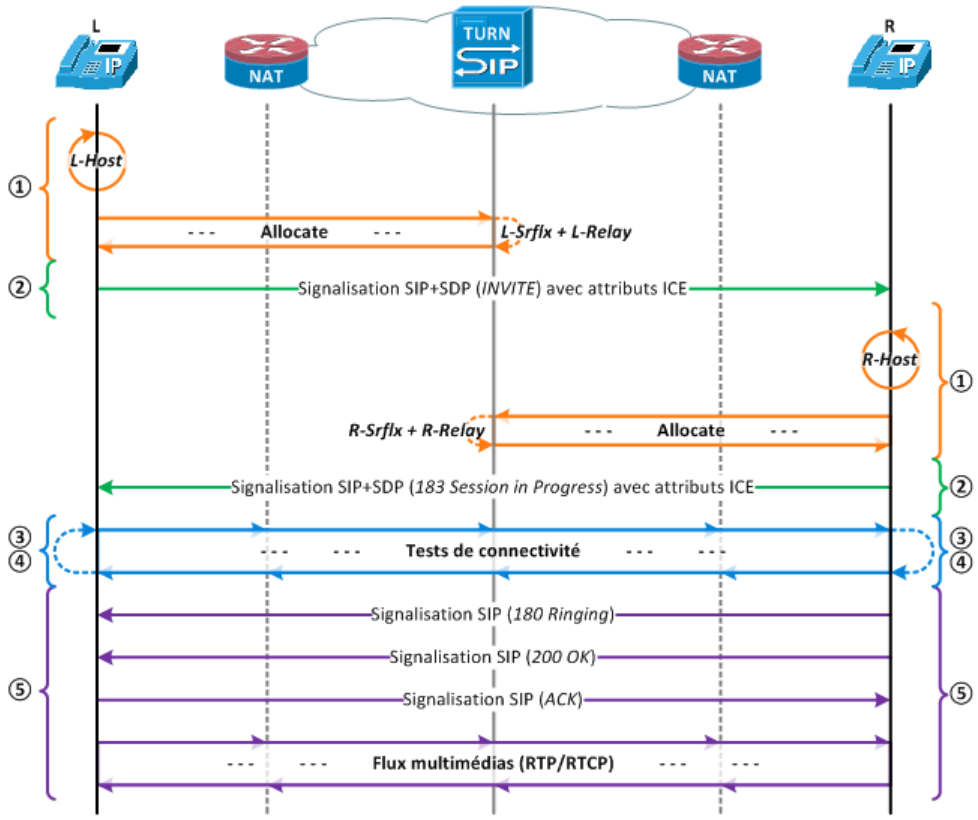


Fig. 18 – *Étapes d'ICE*

La première étape sert de base à l'ensemble du processus ICE. L'objectif est ici de récupérer un maximum de candidats [13] possibles pour chaque flux de la communication à venir. Il existe quatre types de candidats différents, tels que décrits dans la Figure 19 ci-dessous.

13. Un candidat est une adresse de transport, soit l'association d'un protocole de transport (TCP, UDP ...), d'une adresse IP et d'un port, généralement écrite sous la forme protocole:adresse:port (p.ex.

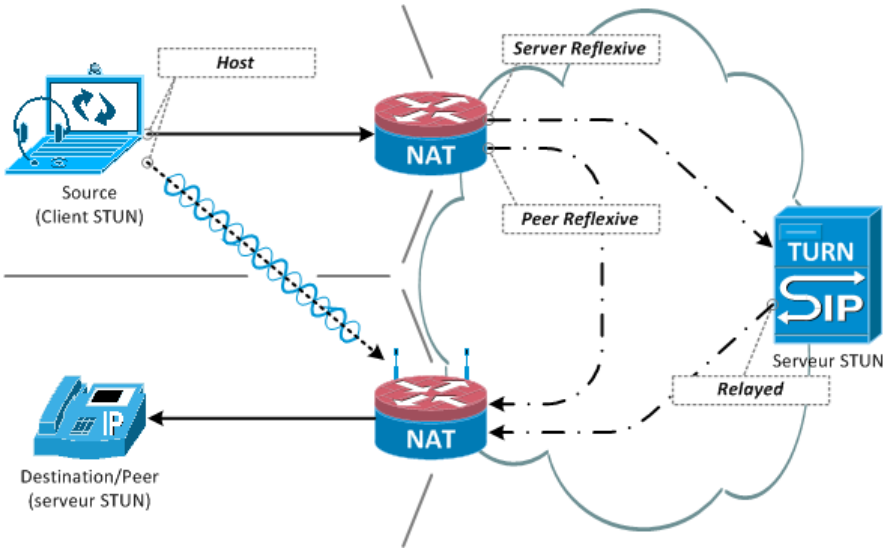


Fig. 19 – *Types de candidats ICE*

La moindre interface locale configurée, qu'elle soit physique (carte réseau filaire ou sans fil) ou logique (VPN, ...) est considérée comme candidat hôte (*host*) potentiel. A partir de chacun de ces candidats, un serveur STUN/TURN est recherché, soit via une configuration manuelle, soit en passant par les enregistrements SRV des DNS. Ce serveur, si disponible, renvoie l'adresse server reflexive pour le candidat hôte, et éventuellement une adresse relayée (*relayed*) dans le cadre d'un serveur TURN.

Lorsque l'ensemble des candidats est récupéré, le terminal émet une requête SIP (signalisation) contenant, au sein du SDP, tous ses candidats dans des attributs *a=candidate*. Ces candidats sont au préalable priorisés selon des règles simples, par ordre d'importance :

- Chaque type de candidat dispose d'une préférence (*host* > *peer reflexive* > *server reflexive* > *relayed*).
- L'interface d'origine dispose également d'une préférence (filaire > sans fil > tunnels ...).
- Chaque composant (flux de la communication) dispose d'une priorité différente, séquentielle, permettant de prioriser deux candidats de même priorité.

Sur réception, un destinataire compatible ICE effectuera les mêmes tâches avant de transmettre ses informations à l'émetteur de la requête. Lorsque l'ensemble de ce processus est terminé, chaque terminaison dispose de l'ensemble des candidats pour chaque partie, permettant de créer des paires de candidats. La Figure 20 ci-dessous décrit l'ensemble des échanges effectués ici, dans une version allégée (une seule interface d'origine pour un unique flux/composant).

Deux points importants ressortent déjà ici. En premier lieu, remarquons que l'adresse réflexive est placée dans les attributs *c=* et *m=*. Ce choix n'est pas aléatoire puisqu'il permet, dans le cas d'un destinataire ne supportant pas ICE (*a=candidate* ignorés), de spécifier le candidat par défaut, relais exceptés, qui dispose de la plus haute probabilité de succès [14]. En second lieu, deux attributs ICE, *ice-ufrag* et *ice-pwd* sont spécifiés. Généré aléatoirement pour chaque nouvelle session, ils vont servir à effectuer, dans le

tcp:192.0.2.11:5060).

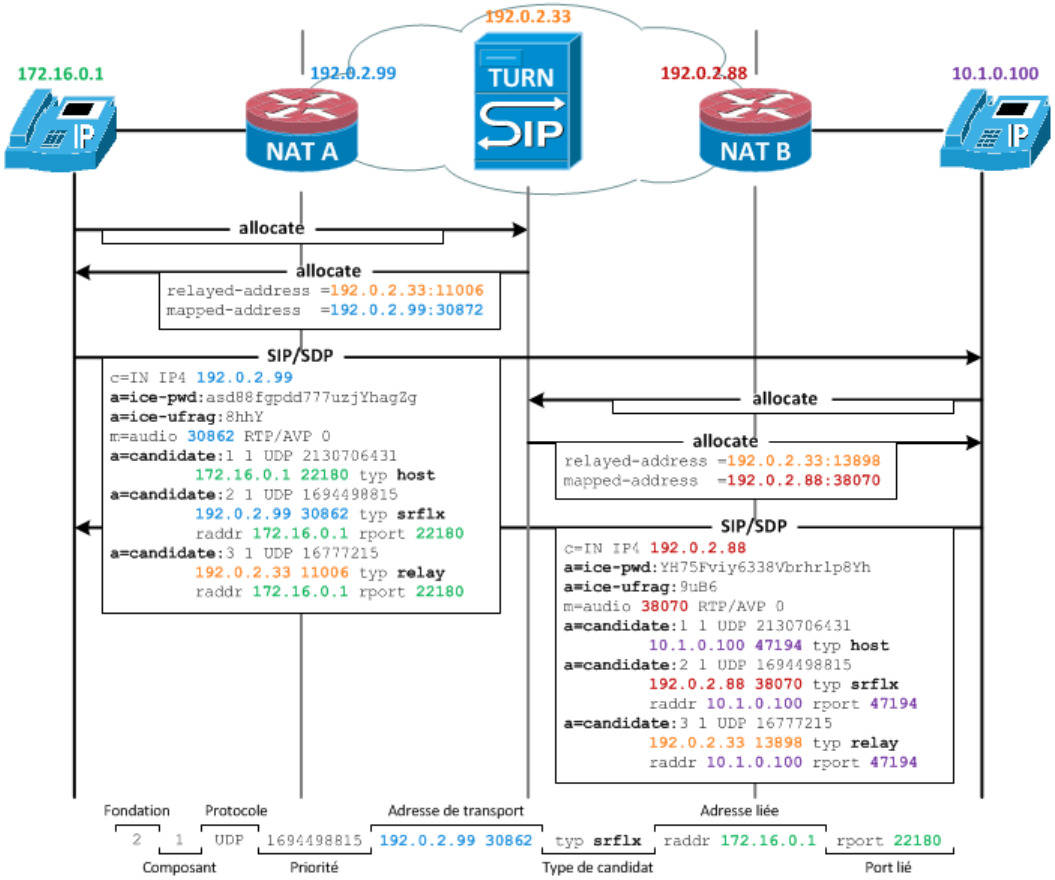


Fig. 20 – *Échange des candidats ICE*

cadre de l'étape suivante (tests de connectivité), une authentification STUN à court terme, uniquement valable le temps de la session, permettant de vérifier l'authenticité du terminal distant et éviter l'usurpation d'identité.

Maintenant que les deux terminaux disposent de l'ensemble des candidats. Des paires sont créées, combinant chaque candidat local avec tous les candidats distants et en éliminant toutes les paires redondantes, notamment celles dont les candidats locaux sont de type *server-reflexive*, car ils émanent d'un candidat hôte. Lorsque ces paires sont créées et épurées, les tests de connectivités peuvent débuter. Le principe général est simple : pour chaque paire, les terminaux émettent une requête STUN à destination du candidat distant depuis leur candidat local. Si une réponse est reçue, la paire est validée.

Il y a plusieurs points intéressants à observer dans cette Figure 21 :

1. Les premiers tests à être effectués sont ceux impliquant les candidats host distant. Toutefois, ceux-ci, dans la majeure partie des cas, ne pourront pas fonctionner (adresse privée non routable globalement), sauf si un réseau "local" relie les deux terminaux d'une manière ou d'une autre (VPN, WiFi ...). A l'opposé, les tests de paires relayées (3^{e} test depuis L dans cette figure) fonctionnent quelle que soit la situation.

14. Le scénario décrit dans la section suivante détaille cet aspect.

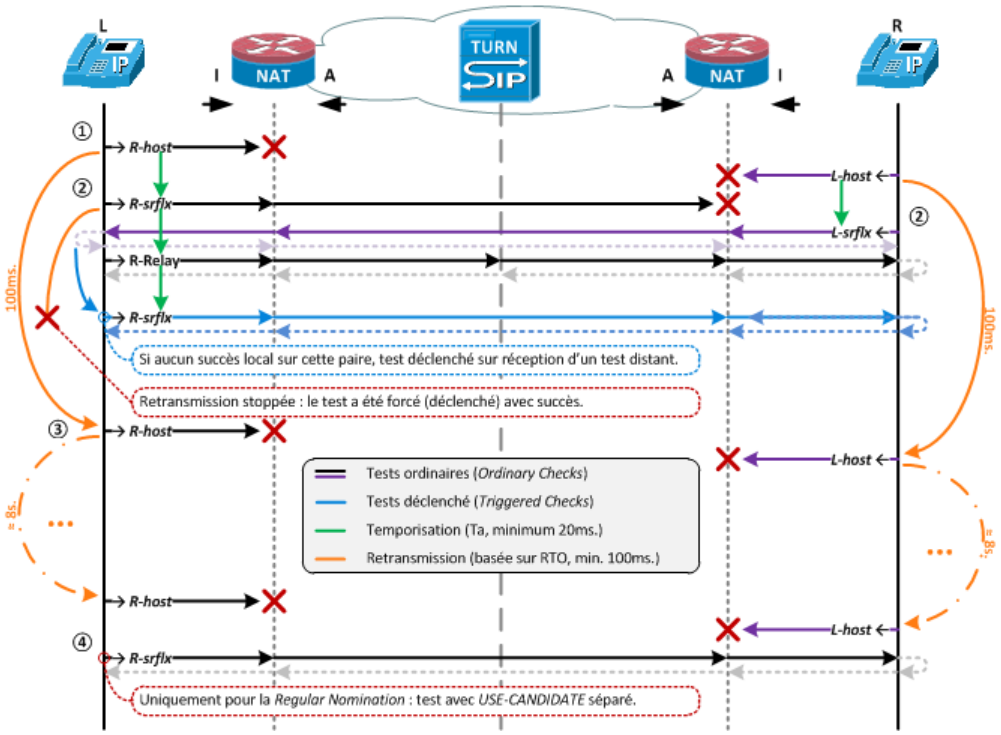


Fig. 21 – *Mécanismes des tests de connectivité*

2. Les tests de connectivité *server-reflexive* ne fonctionnent pas systématiquement du premier coup si un filtrage dépendant existe à la destination. Toutefois, dans une grande majorité des cas, si un terminal ne parvient pas à contacter son destinataire de la sorte, ce dernier ne rencontrera pas de problème, la permission ayant été créée. Le succès de ces tests est directement dépendant du type de NAT utilisé tant localement qu'au niveau de la destination [15].

3. Le mécanisme de retransmission de STUN est mis en œuvre ici de manière extensive. Pour rappel, par défaut, STUN prévoit des valeurs amenant à des retransmissions jusqu'à près de 40 secondes, ce qui est intolérable dans le cadre de la VoIP. Pour ICE, cet élément a été pris en considération : de 40 secondes, les retransmissions peuvent se produire jusqu'à 8 secondes au maximum, ce qui s'avère beaucoup plus raisonnable en VoIP, bien que déjà conséquent.

   Il est intéressant de noter que les retransmissions sont interrompues lorsqu'un *triggered check* est planifié, afin d'éviter toute retransmission inutile. Ce type de test survient lorsqu'un test distant est reçu sur une paire donnée : celle-ci est alors testée au plus vite.

4. La nomination des paires à utiliser pour la communication peut se dérouler selon deux méthodes différentes. Tout d'abord, la méthode classique va attendre soit que l'ensemble des tests et retransmissions soient terminés, soit qu'un facteur déclencheur (par exemple avoir au moins une paire de type reflexive pour tous les composants) est atteint, avant d'émettre à nouveau un test de connectivité précisant, à l'aide de l'attribut USE-CANDIDATE, la paire à utiliser.

15. Le scénario ci-après décrit les différentes combinaisons et le résultat attendu pour chacune d'entre elles. Certaines situations peuvent rapidement s'avérer très complexes à appréhender !

La seconde méthode, dite agressive, annonce l'attribut USE-CANDIDATE dans tous ses tests. De ce fait, dès qu'une paire est validée, celle-ci peut être utilisée pour émettre les flux multimédias. La communication est ici établie bien plus rapidement, mais elle est aussi sujette à des microcoupures puisque les tests ne se terminent pas pour autant : toute paire de meilleure priorité qui serait validée par la suite sera alors immédiatement utilisée. Dans la Figure 21, cela signifie par exemple que le terminal L est susceptible de pouvoir établir son flux RTP à l'aide de la paire relayée avant de basculer sur la paire réflexive lorsque son test est reproduit avec succès.

Par l'ensemble de ces mécanismes, ICE est capable de déterminer quelle connexion est la plus appropriée pour une communication donnée. Le choix effectué peut être imparfait (selon par exemple la qualité d'un lien), mais en règle générale, cette sélection est optimale au regard de la topologie considérée. La mise en situation ci-après rentre dans certains détails spécifiques aux topologies.

Mise en situation

L'ensemble des mécanismes décrits dans la section précédente présente un framework complet, mais également difficile à appréhender dans son ensemble. Il est ici question de fournir un exemple de bout en bout, présentant toutes les étapes d'un call flow complet. La topologie utilisée ici est volontairement complexe afin de couvrir un maximum de situations différentes, avec deux edge proxys (EP1 et EP2) reçoivent les requêtes et les transmettent à l'authoritative proxy (PROXY). Afin de simplifier les *call flow*, certaines figures ne présentent que la traversée de NAT du terminal L.

L'enregistrement

Le client L s'enregistre en utilisant le protocole SIP-OUTBOUND en passant par chacun des deux proxys frontaliers, EP1 (qui fonctionne en UDP) et EP2 (qui fonctionne en TCP).

UDP

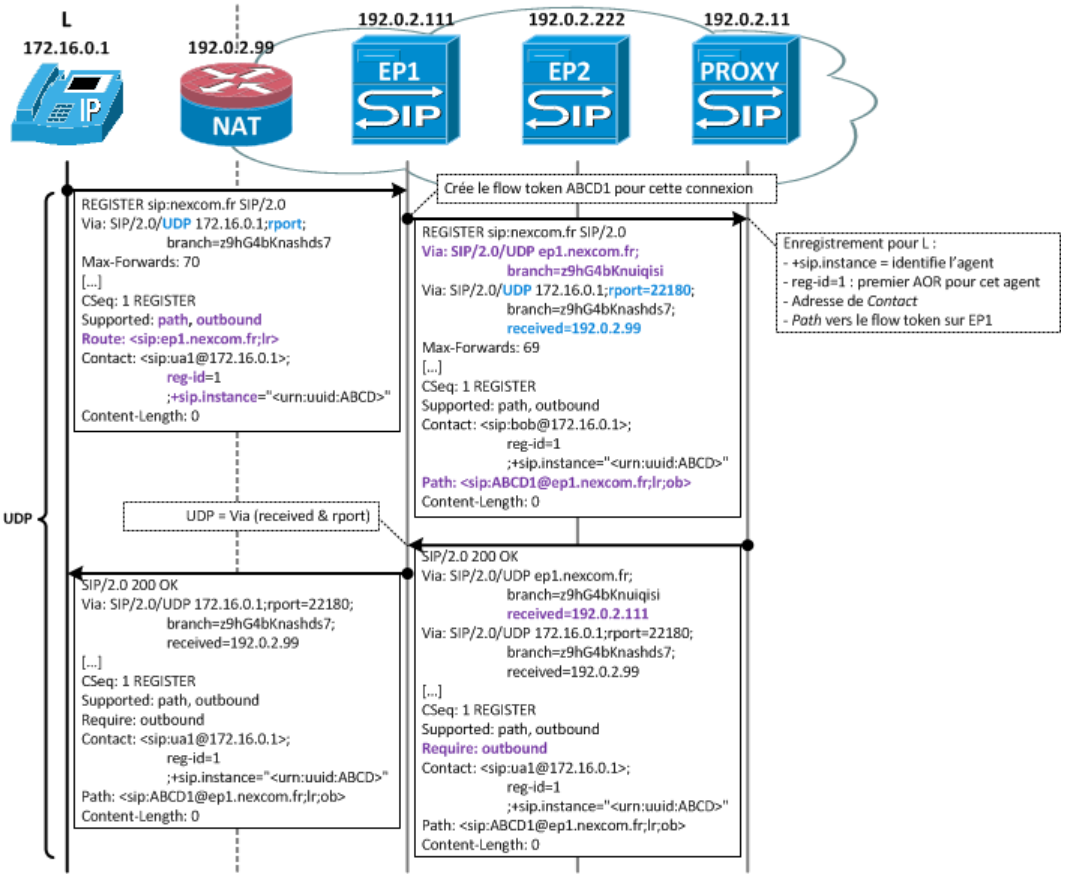


Fig. 22 – *Enregistrement en UDP*

Dans le cas d'UDP (Figure 22), les paramètres *received* et *rport* permettent de renvoyer les réponses au travers d'EP1 à destination de l'association du NAT créée par la requête.

Ces deux informations sont complétées dans le premier *Via* par le premier proxy traversé, ici EP1, par lequel Bob force le passage (entête Route dans sa requête). EP1 complète, selon le protocole SIP-OUTBOUND, la requête en spécifiant notamment le *flow token* créé au sein d'un entête *Path*. Cette information est ensuite stockée par le registrar afin de pouvoir la fournir pour les requêtes à destination de cet abonné.

Protocoles orientés connexion (TCP)

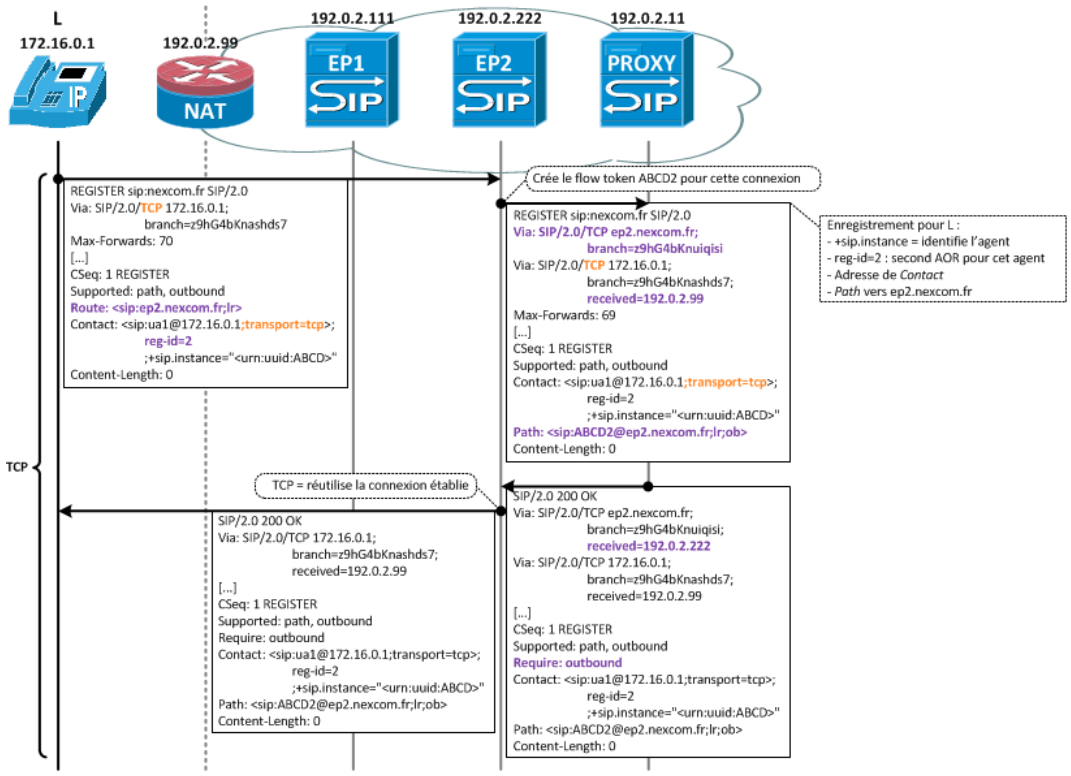


Fig. 23 – *Enregistrement en TCP*

Du côté des protocoles orienté connexion (TCP, TLS, WebSockets, ...), les mêmes contraintes ne s'appliquent pas (Figure 23) : aucun *rport* ici, la connexion ouverte par la requête est réutilisée pour acheminer ses réponses. La requête passe ici par EP2, fonctionnant en TCP uniquement : le *sip-instance* reste inchangé (il identifie l'agent SIP), mais puisque le domaine reste le même, l'identifiant d'enregistrement est incrémenté. Grâce à cela, le proxy disposera de deux liens différents vers un même abonné, qui pourront servir en *failover* (redondance) si nécessaire.

En définitive, ce qui s'avère intéressant ici n'est pas l'enregistrement en TCP à proprement parlé (il y a beaucoup moins à discuter en TCP qu'en UDP), mais bien l'enregistrement en lui-même, avec un nouvel AOR pour cet abonné, pour un même terminal, mais par de multiples connexions.

L'établissement de session

Deux cas de figure se présentent lors de l'établissement d'une session (requête SIP INVITE) : soit elle est initiée par le client (requête sortante), soit elle est reçue depuis l'extérieure (requête entrante). Selon la direction, le comportement est sensiblement différent, particulièrement dans le cadre des requêtes entrantes.

Initiée par le client

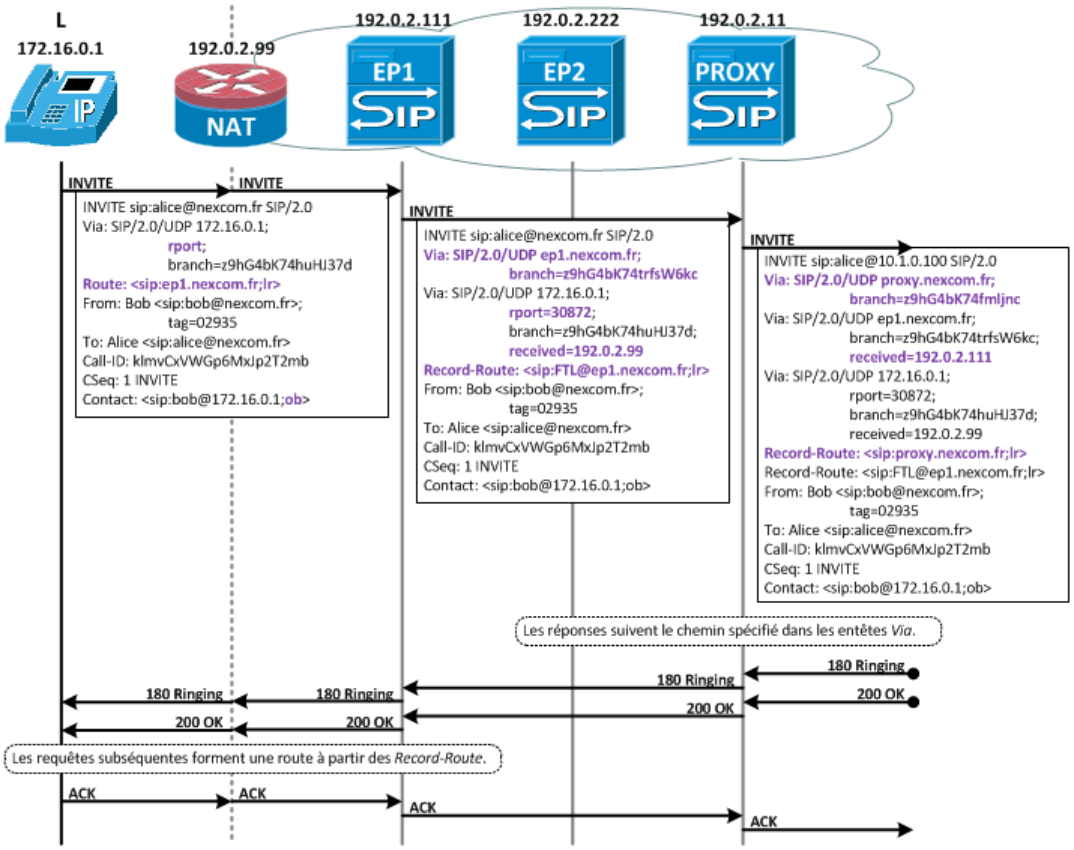


Fig. 24 – *Session initiée par le client*

L'acheminement d'une requête dans cette direction n'est guère surprenant. La route par EP1 est toujours spécifiée, et celui-ci se place systématiquement sur le trajet des requêtes subséquentes à l'aide d'un *Record-Route*. Ce mécanisme est engendré par l'ajout du paramètre *ob* (pour *OutBound*) au sein du contact, forçant les proxys, que ce soit EP1 ou le registrar à travailler selon les mécanismes du protocole SIP-OUTBOUND.

Les réponses et requêtes suivent, de manière classique, les informations contenues respectivement dans les *Via* et les *Record-Route*. Il demeure toutefois intéressant de noter la forme particulière du *Record-Route* d'EP1 qui contient le *flow token* à destination du NAT de Bob. De cette manière, les requêtes subséquentes peuvent transiter sur le réseau SIP en disposant immédiatement de toutes les informations nécessaires.

Recevoir l'invitation à une session

Si le cas précédent s'avère relativement classique, il est beaucoup plus intéressant de détailler ce qui se passe lorsqu'une invitation doit être reçue par Bob lors d'une requête externe. Il existe en effet de multiples chemins vers la destination. Comment l'infrastructure gère-t-elle cet aspect du routage ?

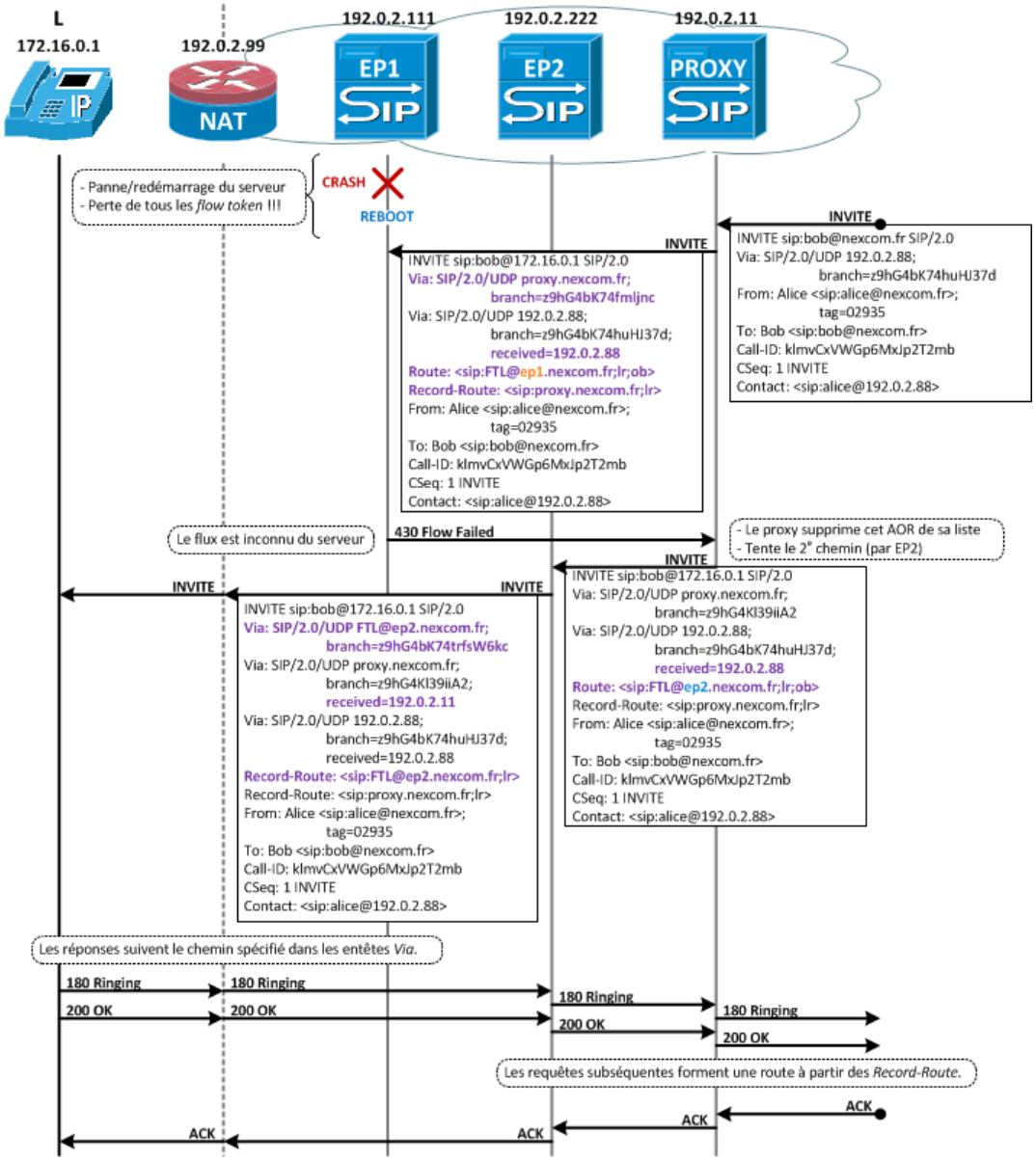


Fig. 25 – *Réception d'une invitation*

Lorsqu'une invitation est reçue pour Bob, deux chemins sont possibles (via EP1 ou EP2). Les routes sont utilisées dans l'ordre, et ici, PROXY tente de passer par EP1, qui a rencontré des difficultés techniques et a redémarré, perdant tous ses tickets de connexion. Puisqu'il ne connaît pas le *flow token* mentionné par PROXY dans la Route, il répond par un *430 Flow failed* indiquant son ignorance.

Le proxy disposant d'un second AOR pour le même terminal, il peut toutefois l'utiliser,

et envoie donc sa requête vers EP2. Les entêtes Route ont ici une grande importance puisque ce sont eux qui détiennent les informations nécessaires aux edge proxys pour retrouver la connexion nécessaire (le *flow token* FTL). Les informations des entêtes Route sont stockées à l'enregistrement de l'abonné comme information de path (de l'entête homonyme). Il est également essentiel de noter que, comme précédemment, le paramètre *ob* impose à l'edge proxy de rester sur le chemin de la signalisation, afin de pouvoir gérer la traversée du NAT pour toutes les requêtes subséquentes.

Interactive Connectivity Establishment

La gestion de la traversée des NAT pour la signalisation SIP est essentielle pour la suite de la communication. Pourtant, la partie la plus complexe reste à venir. Nous ne traiterons pas ici de STUN et TURN en tant que solution *standalone*, qui n'ont que peu d'intérêt en elles-mêmes. Nous préférons nous orienter spécifiquement sur la solution la plus efficace et complète avec ICE que nous allons détailler étape après étape.

L'échange des candidats

Il n'y a ici rien de vraiment surprenant à noter. La récupération des candidats se fait au travers de requêtes *bind* et/ou *allocate*. Les adresses réflexives et relayées sont ensuite échangées, par la signalisation, afin que chaque terminal dispose de l'ensemble des candidats, tant locaux que distant, tel que décrit dans la Figure 26.

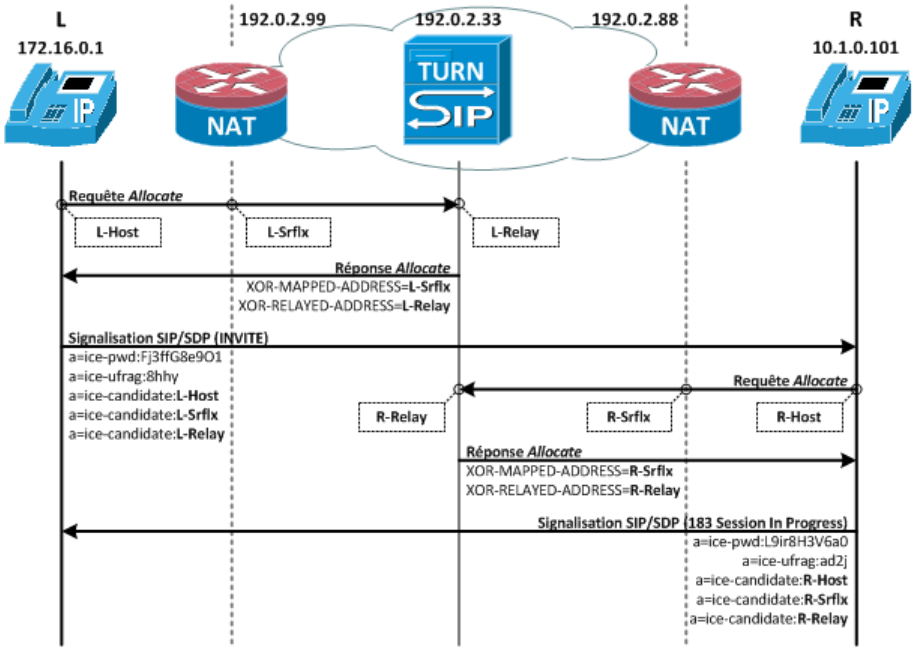


Fig. 26 – *Echange des candidats en SDP*

Le point le plus important à noter ici est l'usage du *183 Session in Progress* pour acheminer les candidats en réponse. Traditionnellement, le SDP du terminal distant est reçu au sein d'un *180 Ringing*, mais ici, la latence introduite par ICE deviendrait trop visible pour être acceptable.

En effet, tel que présenter dans le premier cas de la figure ci-contre, renvoyer les candidats via un *180 Ringing* serait susceptible de générer des situations indésirables dans lesquelles les flux multimédias ne pourraient pas transiter. Si le destinataire décroche son combiné trop tôt notamment, il est parfaitement envisageable qu'aucune connexion bidirectionnelle valide existe, engendrant une dégradation du service (pas de communication possible), du moins temporairement.

Le second cas de figure représente une approche permettant d'absorber cette latence en évitant de déclencher un résultat tangible pour le destinataire (le téléphone sonne) avant que tous les tests ne soient terminés. De cette manière, ICE est en mesure de déterminer si la communication peut effectivement être établie efficacement avant de le signifier au destinataire (*180 Ringing*).

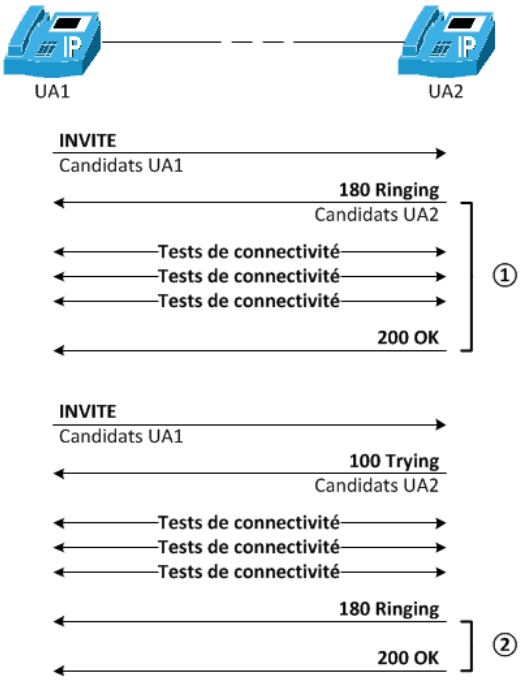


Fig. 27 – *Latence introduite par ICE*

La création des paires

Avec l'ensemble des candidats locaux et distants connus, les terminaux peuvent se constituer des paires qui vont par la suite être testées. Ces paires représentent toutes les possibilités de connexion. La génération de ces paires se déroule en quatre étapes, décrites dans la Figure 28.

1. Toutes les combinaisons possibles sont faites entre candidats locaux et distants, formant des paires de candidats.
2. Ces paires sont priorisées en fonction de la priorité des candidats qui la compose. Par exemple, une paire de candidats hôte vers hôte aura une bien meilleure priorité qu'une paire de candidats relayés (pour prendre les deux extrêmes).
3. Tous les candidats réflexifs locaux sont remplacés par leur base, c'est-à-dire le candidat à l'origine de leur création (le candidat ayant servi à contacter le serveur STUN). Par conséquent, tous les candidats *server-reflexive* sont remplacés par des candidats host.
4. La dernière étape consiste en l'élimination de toutes les paires redondantes. Dans le cas présent, les paires anciennement *server-reflexive* apparaissent à double. Seules les paires de meilleure priorité sont conservées.

Avec les paires restantes, le terminal ICE définit, selon l'algorithme *frozen*, une liste de paires à tester séquentiellement, au minimum un test toutes les 20 millisecondes , jusqu'à en trouver une valide (permettant une communication bidirectionnelle entre les deux terminaux) ou jusqu'à leur épuisement total.

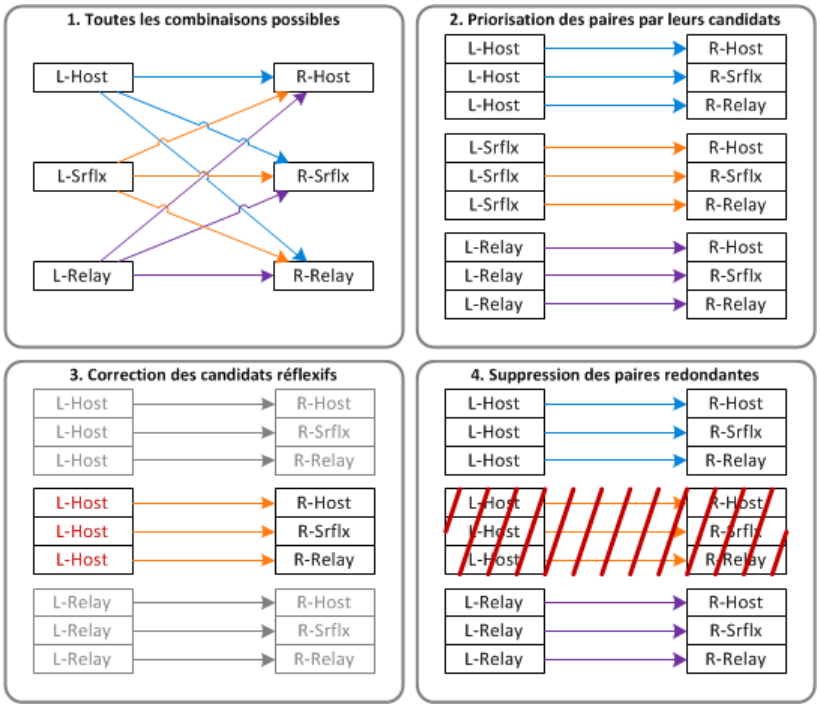


Fig. 28 – *Génération des paires de candidats*

Les tests de connectivité

La véritable complexité d'ICE tient dans ses tests de connectivité. Chaque paire formée va être testée afin de déterminer si une connectivité existe ou non. Comme discuté précédemment, certaines paires sont plus faciles à traiter que d'autres (Figure 29). C'est le cas notamment des paires contactant un candidat host distant, qui ont peu de chances d'aboutir (sauf lien "local" entre les deux terminaux), alors que les paires relayées sont systématiquement couronnées de succès (sauf saturation du serveur relai), mais consomment également beaucoup de ressources sur le serveur relai et sont donc à proscrire autant que possible (dernier recours uniquement).

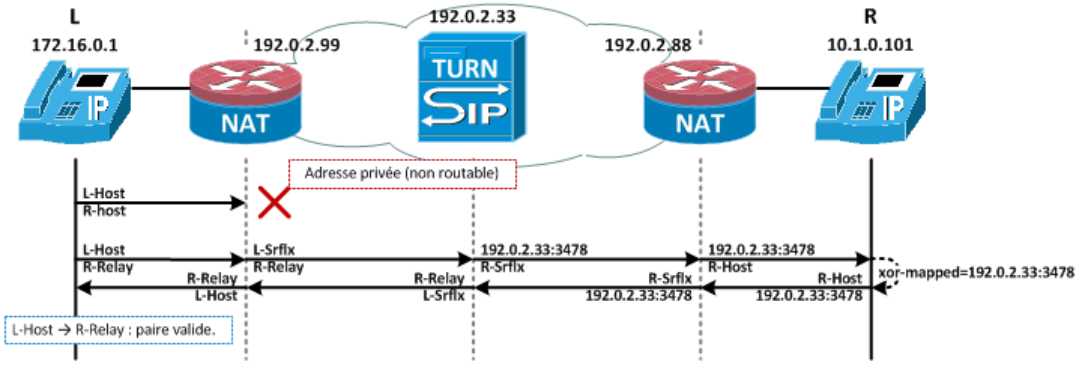


Fig. 29 – *Tests host et relay*

Les paires réflexives vont s'avérer plus complexes à traiter. En effet, leur comportement va grandement dépendre du type de NAT rencontré, tant à la source qu'à la destination. Il existe, selon la nouvelle définition des NAT et en ne considérant que les cas cohérents,

36 combinaisons possibles entre la source et la destination. En étudiant systématiquement chaque combinaison, il est possible de tracer le tableau en Figure 30.

			NAT Destination					
	Combinaison		A	B	C	D	E	F
	Association		I			A		A+P
		Filtrage	I	A	A+P	A	A+P	A+P
NAT Source 1	I	I	srflx → srflx	srflx → srflx[1]	srflx → srflx[1]	srflx → prflx[1,2]	srflx → prflx[1,2]	srflx → prflx[1,2]
2		A	srflx → srflx	srflx → srflx[1]	srflx → srflx[1]	srflx → prflx[1,2]	srflx → prflx[1,2]	srflx → prflx[1,2]
3		A+P	srflx → srflx	srflx → srflx[1]	srflx → srflx[1]	relay → relay[3]	relay → relay[3]	relay → relay[3]
4	A	A	prflx → srflx[2]	prflx → srflx[1,2]	relay → relay[3]	relay → relay		
5		A+P	prflx → srflx[2]	prflx → srflx[1,2]	relay → relay[3]			
6	A+P	A+P	prflx → srflx[2]	prflx → srflx[1,2]	relay → relay[3]			

[1] Succès uniquement après que le destinataire ait effectué son test de connectivité (création des permissions nécessaires).
[2] Fonctionne sans relai si et seulement si l'adresse *srflx* est équivalente à l'adresse *prflx* créée (dans le cas d'un *pool* d'adresses publiques).
[3] Seule une direction est supportée dans le cadre d'adresses réflexives, imposant l'usage de serveurs relais (filtrage trop restrictif).

Légende :

I	Indépendant	srflx	Adresse *server-reflexive*
A	Dépendant de l'adresse	prflx	Adresse *peer-reflexive*
A+P	Dépendant de l'adresse et du port	relay	Adresse relayée (TURN)

Combinaisons recommandées (RFC 4787) — Combinaisons restrictives
Combinaisons acceptables (filtrage A+P à proscrire) — Combinaisons à proscrire

Fig. 30 – *Combinaisons de NAT et résultats*

Ce tableau présente l'ensemble de ces cas sous une forme concentrée. Nous avons sélectionné, parmi ces nombreuses situations, des regroupements fonctionnant sur des principes similaires et qui vont démontrer toute la complexité, mais aussi toute l'efficacité d'ICE. Dans un premier temps, attardons-nous sur les cas 1 à 3 des colonnes B et C :

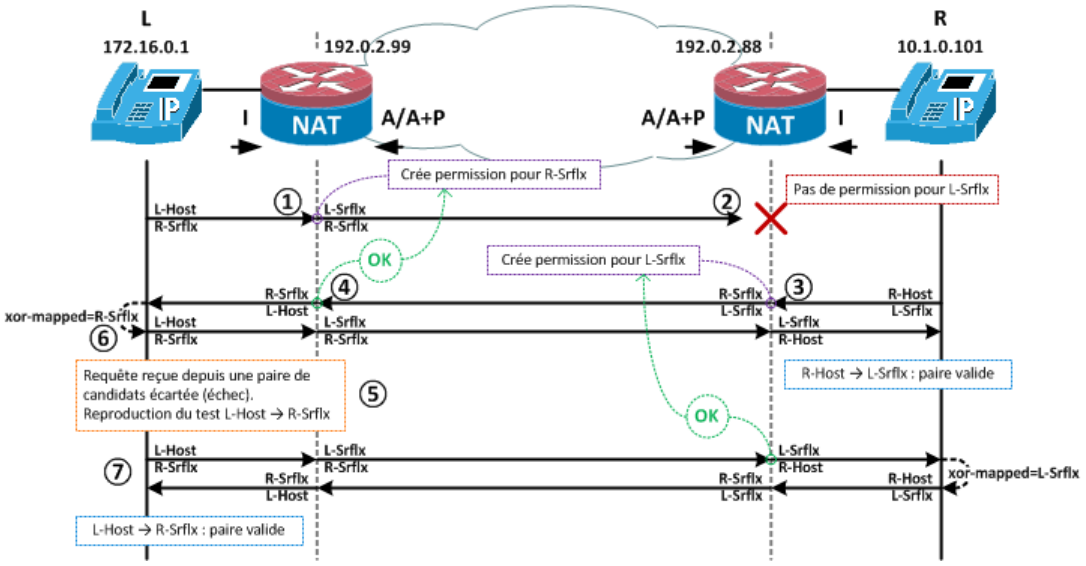


Fig. 31 – *Combinaisons 1B:3C*

Certaines de ces combinaisons (1B et 2B, privilégiant d'un côté ou des deux un comportement de filtrage axé sur la sécurité) font partie des situations recommandées par la RFC 4787. Il n'en demeure pas moins que le filtrage par adresse et port ne change pas grand-chose au cas présent, mais il est en règle générale écarté car, nous le verrons dans les cas suivants, il s'avère beaucoup trop restrictif dans certaines situations, sans avoir

de réelle plus-value compensant ses défauts. L'établissement de la connectivité se déroule comme suit :

1. La requête initiale de Bob (terminal L) crée une permission sur NAT-L (filtrage dépendant) pour l'adresse *server-reflexive* (Srflx) d'Alice (terminal R).
2. Cette requête est rejetée par NAT-R (filtrage dépendant) qui ne dispose que d'une permission pour le serveur STUN/TURN, différente de l'adresse publique de NAT-L.
3. Alice, tout comme Bob, effectue ses propres tests de connectivité, et tente de contacter Bob sur son adresse *server-reflexive* (Srflx), créant une permission pour celle-ci sur NAT-R.
4. La requête est autorisée par NAT-L car une permission pour cette adresse a été créée en (1).
5. Le terminal de Bob (L) détecte qu'il vient de recevoir une requête à partir d'une paire de candidat qu'il avait au préalable écartée (pas de réponse). Il place à nouveau cette paire dans sa liste de test afin de vérifier si une connectivité existe désormais.
6. Il envoie la réponse à Alice. Toutes les permissions adéquates étant créées, la requête va aboutir sans encombre et le terminal R valide la paire de candidats.
7. Bob reproduit le test de cette paire. Ici aussi, toutes les permissions adéquates existant, le processus se déroule sans encombre et Bob reçoit une réponse d'Alice, lui permettant également de valider la paire de candidats.

La seconde série de cas que nous allons détailler ici, couvrant les cas 2D à 2F, fonctionne de manière globalement similaire à ce que nous venons de voir. La principale différence vient du fait qu'un des deux NAT propose un comportement "symétrique", dont les associations et le filtrage sont tous deux de type dépendant.

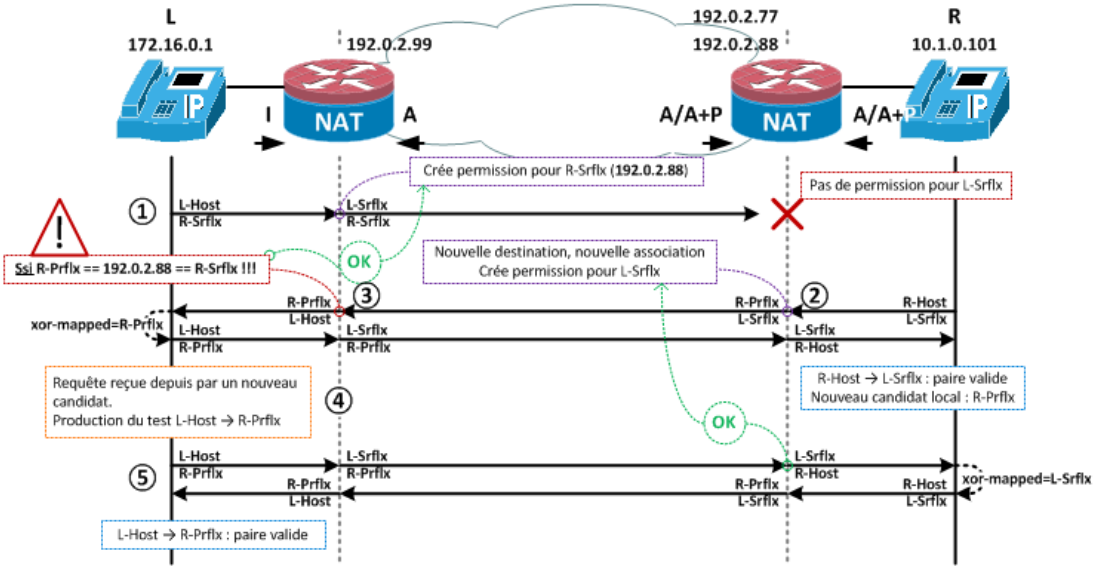


Fig. 32 – *Combinaisons 2D:2F*

Il est ici important de noter les différences avec les cas précédents. Outre le NAT "symétrique" (à association et filtrage dépendant) en destination, le filtrage du NAT source

(NAT-L) est uniquement dépendant de l'adresse (qui est un type de NAT extrêmement rare actuellement). Ce dernier point est particulièrement important puisqu'un filtrage A+P au niveau du NAT source empêcherait toute utilisation d'adresses réflexives pour la communication, la requête (3) étant rejetée par le NAT-L. Avec ceci définit, voici le déroulement complet, avec la subtilité qui peut avoir un impact conséquent sur l'ensemble du processus :

1. Tout comme précédemment, la requête initiale de Bob (terminal L) crée une permission sur NAT-L autorisant l'adresse 192.0.2.88 (soit l'adresse *server-reflexive* apprise par l'échange de signalisation) en entrée, avant d'être rejetée par NAT-R faute de permission nécessaire.
2. Alice effectue le même test, créant une permission sur NAT-R pour l'adresse réflexive de Bob. Ce NAT étant de type symétrique (dépendant dans un sens comme dans l'autre), une nouvelle association est générée (adresse *peer-reflexive* R-Prflx), avec comme destination l'adresse Srflx de Bob.
3. La requête est autorisée par NAT-L si et seulement si l'adresse de la nouvelle association R-Prflx est identique à l'adresse de R-Srflx. En imaginant que NAT-R dispose d'un pool d'adresse publique, si R-Prflx se repose sur l'adresse 192.0.2.77 au lieu de .88, la requête serait rejetée par le NAT de Bob, faute de permission suffisante, forçant l'utilisation de serveurs relais ! Il est intéressant de noter que puisque les requêtes utilisent STUN, Alice découvrira, en réponse, sa nouvelle association R-Prflx, et marquera celle-ci comme valide.
4. Si le point précédent est couronné de succès, Bob détecte qu'une paire encore inconnue a été utilisée pour le contacter (via l'adresse *peer-reflexive* d'Alice). Il planifie de produire un test pour cette nouvelle paire afin de tester qu'une connectivité existe effectivement.
5. En utilisant cette nouvelle adresse R-Prflx en destination de sa requête, toutes les permissions nécessaires existent afin de permettre à l'échange complet d'aboutir sans problème.

Il peut exister de petites subtilités selon les topologies considérées. La probabilité que le cas présenté ici se produise est faible : il faudrait un concours de circonstances malheureux pour que deux requêtes pratiquement successives utilisent deux adresses publiques différentes. Le choix des combinaisons de NAT recommandées par la RFC 4787 prend ici tout son sens : il s'agit d'éviter ce genre de cas particuliers qui peuvent avoir besoin de passer par des serveurs relais.

Les cas de figure 4C à 6C nous permettent de décrire d'autres situations particulièrement vicieuses dans le cadre d'ICE. En considérant ici NAT-L en tant que NAT "symétrique" afin de réduire la taille du call flow, nous avons simplement changé le type de filtrage du second NAT pour un filtrage dépendant de l'adresse et du port (A+P). Que se passe-t-il dans ce cas ?

1. Comme pour les cas précédents, un filtrage dépendant existant au niveau du NAT de destination, la requête est rejetée, mais une permission pour l'adresse *server-reflexive* d'Alice est créée.
2. Lorsqu'Alice émet sa requête à destination de Bob, elle utilisation son adresse réflexive (R-Srflx), et crée une permission relative à l'adresse et au port de l'adresse *server-reflexive* de Bob.

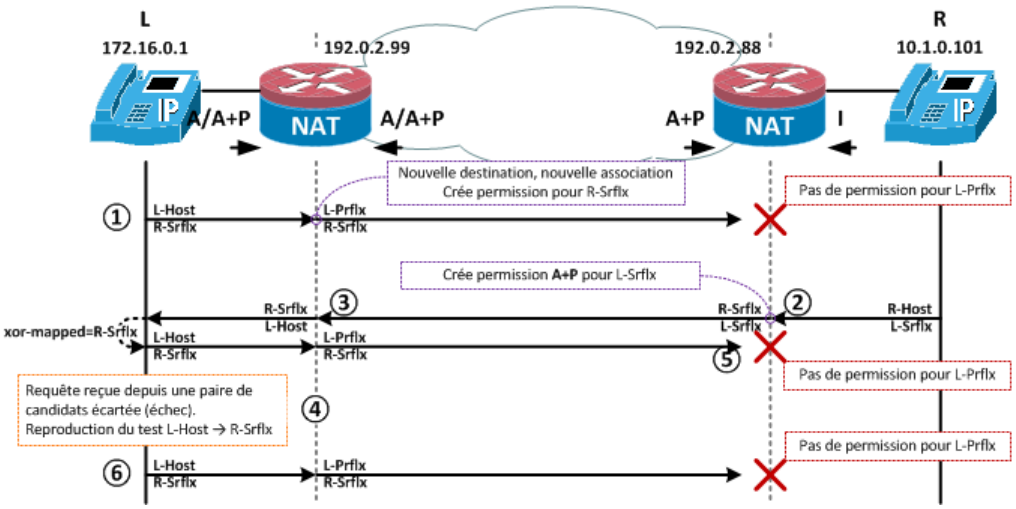


Fig. 33 – *Combinaisons 4C:6C*

3. Le filtrage étant une fonction totalement séparée des associations, la permission créée en (1) sur NAT-L va pouvoir s'appliquer, autorisant le passage de la requête à destination de Bob.
4. Le processus ICE du terminal L détecte qu'il vient de recevoir une requête depuis une paire écartée auparavant. Cet événement déclenche un test, qui est placé à la suite des tests déjà planifié (triggered check).
5. Lorsque le terminal L émet sa réponse, son NAT à association dépendante va lui faire réutiliser l'association *peer-reflexive* (L-Prflx) créée en (1). Arrivée sur NAT-R, la réponse sera rejetée : aucune permission n'existe pour cette source (L-Prflx), qui est différente, par le port, de la permission existante créée en (2) pour L-Srflx. Alice ne recevra donc jamais de réponse.
6. Toutefois, puisque Bob a reçu la requête, la reproduction du test va tout de même avoir lieu. Mais n'aboutira de nouveau pas, car il n'existe toujours aucune permission pour L-Prflx au niveau de NAT-R.

De nombreuses retransmissions vont avoir lieu dans ces cas. Afin d'absorber au maximum leur délai et ainsi permettre une attente oscillant autour de ces 8 secondes, ICE vient compléter ceci en permettant au terminal d'effectuer toutes ses requêtes initiales avant que les premières retransmissions ne puissent avoir lieu. Le temps de retransmission de chaque requête n'est par conséquent pas cumulé. Toutefois, cette approche peut aussi mener à un choix peu optimal par ICE, qui risque par exemple de se servir d'un serveur relai avant que toutes les retransmissions ne soient terminées (ce qui, dans le cas présent, serait bénéfique, puisque de toute façon, retransmettre la requête n'aboutirait à rien de plus qu'auparavant).

L'objectif des timers utilisés par ICE est de simuler les mêmes conditions requises par, notamment, le codec utilisé pour la communication. Les tests de connectivité se comportant comme un flux RTP, les NAT aussi se comporteront de manière similaire, permettant à ICE de détecter efficacement, par rapport aux paramètres à sa disposition, quelle connexion utiliser entre les deux terminaux.

Choix de la paire finale

Le choix final de la paire à utiliser pour chaque composant ne présente pas de difficultés particulières. Dans le cadre d'une nomination classique, le test de la paire valide est reproduit, en fin d'algorithme *frozen*, en incluant le paramètre USE-CANDIDATE. Cette approche offre l'avantage d'être compatible très largement, avec n'importe quelle implémentation d'ICE, mais est aussi relativement lente à déterminer quelle paire utiliser. Au contraire, la méthode agressive annonce systématiquement le paramètre USE-CANDIDATE, permettant de déterminer rapidement une paire utilisable, mais pouvant également introduire des sauts de flux RTP, celui-ci changeant de cible au fur et à mesure que d'autres paires valides, de meilleure priorité, sont détectées.

Conclusion

Le framework conçut par l'IETF dans le cadre de la traversée de NAT s'avère efficace. Il représente la solution la plus efficace et évolutive disponible actuellement. Son plus gros problème est un manque d'implémentation dans les réseaux actuels. Ce document a tenté d'apporter, en quelques pages, les clés nécessaires afin de se lancer dans une implémentation de ce type, sans pour autant rentrer dans des détails trop spécifiques des différents protocoles (ceux-ci peuvent se retrouver dans leurs spécifications respectives). Au contraire, nous nous sommes ici focalisés sur certains aspects qui, s'ils sont couverts par les différentes normes, sont difficiles d'accès au premier abord, que ce soit au travers de la théorie, mais surtout d'un côté plus pratique, abordant des situations particulièrement vicieuses.

Annexes

[A] Correspondance RFC3489 et RFC4787

CONE (RFC 3489)	BEHAVE (RFC 4787)
Full Cone NAT	Association : indépendante de la terminaison. Filtrage : indépendant de la destination
Restricted Cone NAT	Association : indépendante de la terminaison. Filtrage : par adresse uniquement.
Port-Restricted Cone NAT	Association : indépendante de la terminaison. Filtrage : par adresse et port.
Symmetric NAT	Association : dépendante (adresse ou adresse/port). Filtrage : par adresse et port.

[B] Authentification TURN

L'authentification des requêtes Allocate de TURN se déroule au travers d'un challenge. Une première requête est envoyée, sans authentification, et est rejetée par le serveur qui fournit alors un domaine (*realm*) et un nonce aléatoire (*nonce*).

À partir de ces informations, l'abonné génère sa requête à nouveau en précisant, en plus des informations citées ci-dessus, son nom d'utilisateur. Grâce à l'ensemble de ces informations, ainsi qu'au mot de passe qu'il partage avec le serveur, le client TURN peut également générer un paramètre d'intégrité, sous la forme d'un hash du message, qui permet au serveur de valider l'identité du client.

Le nonce n'est valide que pour une période limitée. En cas de nouvelle requête, un nouveau *nonce* est envoyé au client qui effectue à nouveau les mêmes démarches.

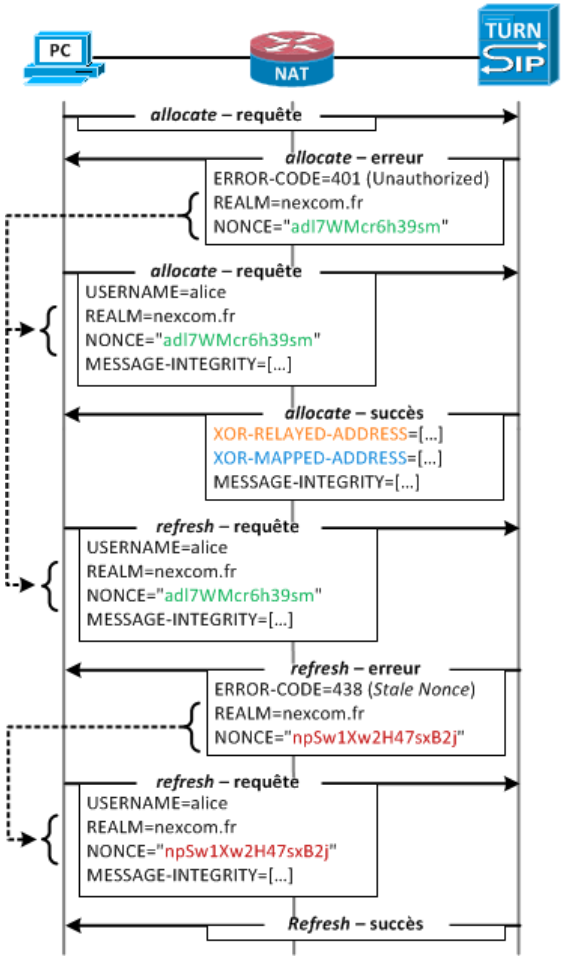


Fig. 34 – *Authentification TURN*

Références

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley et E. Schooler : SIP : Session Initiation Protocol. RFC 3261 (Proposed Standard), Juin 2002.

[2] M. Handley, V. Jacobson et C. Perkins : SDP : Session Description Protocol. RFC 4566 (Proposed Standard), Juillet 2006.

[3] H. Schulzrinne, S. Casner, R. Frederick et V. Jacobson : RTP : A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), Juillet 2003.

[4] J. Rosenberg, J. Weinberger, C. Huitema et R. Mahy : STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), Mars 2003.

[5] F. Audet et C. Jennings : Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787 (Best Current Practice), Janvier 2007.

[6] S. Guha, K. Biswas, B. Ford, S. Sivakumar et P. Srisuresh : NAT Behavioral Requirements for TCP. RFC 5382 (Best Current Practice), Octobre 2008.

[7] J. Rosenberg et H. Schulzrinne : An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing. RFC 3581 (Proposed Standard), Août 2003.

[8] C. Jennings, R. Mahy et F. Audet : Managing Client-Initiated Connections in the Session Initiation Protocol (SIP). RFC 5626 (Proposed Standard), Octobre 2009.

[9] D. Wing : Symmetric RTP / RTP Control Protocol (RTCP). RFC 4961 (Best Current Practice), Juillet 2007.

[10] J. Rosenberg, R. Mahy, P. Matthews et D. Wing : Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), Octobre 2008.

[11] P.Karn et C. Partridge : Improving Round-Trip Time Estimates in Reliable Transport Protocols, Août 1987.

[12] R. Mahy, P. Matthews et J. Rosenberg : Traversal Using Relays around NAT (TURN) : Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard), Avril 2010.

[13] J. Rosenberg : Interactive Connectivity Establishment (ICE) : A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard), Avril 2010.