RFC 6544 : TCP Candidates with Interactive Connectivity Establishment (ICE)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 20 mars 2012

Date de publication du RFC : Mars 2012

https://www.bortzmeyer.org/6544.html

Le protocole ICE, normalisé dans le RFC 8445 [1], permet de traverser des obstacles comme les routeurs NAT, en essayant plusieurs techniques successives. ICE fonctionne en proposant une liste d'adresses **candidates** et en testant ensuite ces candidates (avec STUN, notamment) jusqu'à ce qu'on en trouve une qui passe. Dans le RFC originel sur ICE, ces candidates ne pouvaient être testées qu'en UDP. Désormais, notre RFC 6544 permet des candidates TCP (ainsi qu'un mélange des deux).

ICE vise surtout les applications pair-à-pair notamment pour le multimédia (par exemple, pour la télé-conférence). Pourquoi est-ce important de pouvoir faire du TCP ? La section 1 rappelle que certains protocoles ne marchent qu'avec TCP (comme le MSRP du RFC 4975) et que certains protocoles ont certes la possibilité d'utiliser UDP mais peuvent choisir TCP dans certains cas (RTP, RFC 4571, par exemple). C'est évidemment surtout utile dans le cas où on utilise TURN (RFC 5766), protocole qui est parfois le seul à passer certains routeurs NAT, en utilisant un relais TCP.

En général, pour les protocoles qui se servent d'ICE, UDP sera préférable (l'annexe A explique pourquoi). Dans les autres cas, ce nouveau RFC leur permet de faire enfin du TCP. (Et cette possibilité est désormais enregistrée à l'IANA <https://www.iana.org/assignments/ice/ice.xml#transport-extensions>.)

La section 3 résume les changements dans ICE liés à ce nouveau service. Mais c'est assez simple. Il y a trois sortes de candidats TCP, les actifs (qui vont ouvrir la connexion eux-mêmes), les passifs (qui attendent) et les S-O ("*Simultaneous Open*") qui peuvent faire les deux. Dans la liste testée, le type du candidat est indiqué et ICE, pour tester une adresse, va évidemment les apparier en fonction du type (une adresse active avec une passive mais pas avec une autre active, par exemple). Le reste sera fait

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8445.txt

par de l'ICE normal, comme en UDP. Un petit piège : les messages STUN utilisés par ICE sont sur le même port que le trafic applicatif. Pour un protocole à messages comme UDP, il fallait donc marquer les messages pour les distinguer du trafic STUN. Pour un protocole à flot continu de données, comme TCP, il a fallu introduire un marquage, celui du RFC 4571 (conçu pour RTP mais qui peut être utilisé avec tout flot TCP).

Les détails de l'offre initiale figurent en section 4, par exemple la priorité accordée à UDP, lorsque le choix existe. S'il n'existe pas, on utilisera TCP, avec une adresse active, qui a plus de chances de pouvoir sortir du NAT.

Comme avec UDP, l'offre d'adresses candidates est encodée en SDP, avec un nouvel attribut pour différencier les adresses utilisées en TCP selon qu'elles soient actives ou passives. L'annexe C du RFC contient plusieurs exemples de descriptions SDP, par exemple, pour un appel audio :

```
   v=0
   o=jdoe 2890844526 2890842807 IN IP4 10.0.1.1
...
   m=audio 45664 TCP/RTP/AVP 0
...
   a=connection:new
   a=candidate:1 1 TCP 2128609279 10.0.1.1 9 typ host tcptype active
   a=candidate:2 1 TCP 2124414975 10.0.1.1 8998 typ host tcptype passive
   a=candidate:3 1 TCP 2120220671 10.0.1.1 8999 typ host tcptype so
...
```

Cet exemple ne montre que des candidats TCP, un de chaque sorte.

Et comment trouver toutes les adresses candidates possibles ? La section 5 donne quelques idées. Attention, plus il y a de candidates, et comme il faut tester chaque couple possible source/destination, plus les tests seront longs.

Les fanas de programmation noteront l'annexe B, qui explique que les opérations à effectuer avec ICE ne correspondent pas tout à fait à l'utilisation habituelle des "*sockets*". Notamment, il s'agit, sur la même "*socket*", d'initier des nouvelles connexions (vers le serveur STUN) et d'en recevoir (depuis les pairs distants). Le pseudo-code suivant est suggéré :

```
for i in 0 to MAX
      sock_i = socket()
      set(sock_i, SO_REUSEADDR)
      bind(sock_i, local) /* Réserver les "sockets" et les lier à un
      port */

   listen(sock_0)
   connect(sock_1, stun)
   connect(sock_2, remote_a)
   connect(sock_3, remote_b)
```