RFC 7285 : ALTO Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 5 septembre 2014

Date de publication du RFC : Septembre 2014

https://www.bortzmeyer.org/7285.html

Couronnant un travail commencé six ans auparavant <https://www.bortzmeyer.org/alto-wg.html>, avec un net ralentissement de l'enthousiasme vers la fin, ce nouvel RFC normalise enfin le protocole ALTO ("*Application-Layer Traffic Optimization*"). Ce protocole sert notamment dans le cas de transferts de fichiers en pair à pair. Lorsqu'une machine qui veut récupérer un fichier (disons, au hasard, un épisode de "*Game of Thrones*") a le choix entre plusieurs pairs possibles, qui ont chacun les données convoitées, lequel choisir ? ALTO permet d'interroger un oracle, le serveur ALTO, qui va nous donner des informations guidant le choix.

Il existe déjà des tas de façons, pour une application, de mesurer ou de calculer le « meilleur » pair. Par exemple, l'information de routage (annonces BGP, notamment) est publique et une application peut y accéder, par exemple via un "*looking glass*". Mais cela ne donne pas une vue complète et, surtout, cela ne donne pas la vision de l'opérateur réseau, qui pourrait souhaiter, par exemple, qu'on utilise plutôt un pair qui fasse passer par un lien de "*peering*" (gratuit), plutôt que par un lien de transit (payant). ALTO est donc avant tout un moyen de faire connaître les préférences du FAI, de manière à ce que l'application puisse faire un choix plus informé. Par exemple, ALTO permet de distribuer des cartes simplifiées du réseau, suffisantes pour que l'application repère les liens intéressants.

Le but est donc bien que les applications changent leur comportement, en utilisant l'information ALTO, afin de mieux utiliser les ressources du réseau (le transfert en pair à pair peut représenter une bonne part du trafic et il est dans l'intérêt de tous de l'optimiser). La même méthode et le même protocole ALTO peut aussi être utilisé pour optimiser l'accès à un CDN et le serveur ALTO peut donc être fourni par d'autres acteurs que le FAI (le protocole lui-même est neutre, au sens où il n'impose pas de critères de choix ; le coût financier, utilisé dans l'exemple précédent "*peering*"/transit, n'est qu'une possibilité parmi d'autres).

Le problème que résout ALTO est formellement décrit dans le RFC 5693 [1]. Pour le résumer : en raison du modèle en couches, les applications n'ont pas assez d'information sur le réseau sous-jacent et ne

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5693.txt

peuvent donc pas prendre de décisions « intelligentes ». Il s'agit donc de leur donner cette information. Ainsi, le réseau sera mieux utilisé, on aura moins de gaspillage (avantage collectif) et les applications auront de meilleures performances (temps de téléchargement plus courts, par exemple, un avantage individuel, cette fois).

Techniquement, ALTO est un protocole HTTP/REST, où requêtes et réponses sont en JSON (RFC 8259). Que du classique, donc, et qui utilisera l'infrastructure HTTP existante.

Un petit rappel de terminologie (s'appuyant sur le RFC 5693) est fait en section 2. On y trouve les termes d'extrémités ("*endpoint*"), "*ALTO information base*" (l'ensemble des informations connues d'un serveur ALTO), etc.

Place maintenant à l'architecture d'ALTO, en section 3. Le cas typique est celui d'un opérateur, ayant un AS et un serveur ALTO pour distribuer l'information sur cet AS (bien sûr, des tas d'autres cas sont possibles). L'"*ALTO information base*" de cet AS est composée d'une série de **coûts** entre extrémités du réseau. Une extrémité est exprimée sous forme d'un préfixe IP (ce n'est donc pas forcément une seule adresse IP). Dans le cas le plus simple, la base peut tout simplement contenir quelque chose du genre « nos préfixes ont un coût fixe et faible, le reste de l'Internet un coût fixe et deux fois plus élevé » ou bien des informations bien plus complexes. La base contient donc les coûts, vus de cet opérateur. Elle va être distribuée par un serveur ALTO, interrogé par des clients ALTO. Plus tard, d'autres moyens de désigner une extrémité existeront, un registre IANA `<https://www.iana.org/assignments/alto-protocol/alto-protocol.xhtml#address-types>` a été créé pour stocker les types d'adresses (section 14.4 de notre RFC).

Et d'où la base vient-elle ? Le protocole ALTO ne spécifie pas de méthode particulière. La base a pu être remplie à la main par les employés de l'opérateur réseau, en fonction de leur politique. Ou générée automatiquement à partir des données de l'opérateur sur la topologie du réseau. Ou elle a pu être assemblée dynamiquement par le biais d'un système de mesure, ou à partir des informations d'un protocole de routage. Ou encore elle peut être l'agrégation de plusieurs sources. La base n'est donc pas statique, mais elle est prévue pour ne pas changer en temps réel, elle ne tient donc typiquement pas compte de la congestion, phénomène très transitoire. On peut aussi imaginer des serveurs ALTO dialoguant entre eux, pour échanger leurs informations, mais il n'existe pas de protocole standard pour cela (notre RFC concerne uniquement le protocole d'interrogation d'un serveur par un client).

L'information ALTO sera peut-être dans le futur également distribuée par d'autres moyens (une DHT ?) mais rien n'est normalisé pour l'instant. Cela nécessiterait sans doute des mécanismes permettant d'indiquer la fraîcheur de l'information (date d'expiration, par exemple) et des mécanismes de sécurité (signatures) qui n'existent pas actuellement. Pour l'instant, ces services sont fournis par le protocole de transport, HTTP.

ALTO distribue l'information sous la forme de **tables** ("*map*"), indiquant les caractéristiques du réseau, et les coûts associés. (Comme indiqué plus haut, ces tables peuvent être de simples fichiers, distribués comme tels.) Une utilisation directe des tables (ce qu'on nomme le "*Map Service*") ferait beaucoup de travail pour le client : télécharger les tables (qui peuvent être très grosses), puis faire des calculs compliqués pour trouver le meilleur pair. ALTO fournit donc en plus des services auxiliaires pour les clients paresseux. Le premier est le service de filtrage ("*Map Filtering Service*") : le client donne des informations supplémentaires et le serveur lui envoie des tables simplifiées, ne correspondant qu'à ce que le client a demandé. Le second service auxiliaire est le service d'information sur une extrémité ("*Endpoint Property Service*") : le client demande des détails sur une extrémité, par exemple son type de connectivité (on préférera un pair connecté en FTTH à un pair ADSL). Enfin, le troisième service auxiliaire, "*Endpoint Cost Service*" permet d'obtenir directement les coûts d'une extrémité donnée. Mais, avant de détailler le

protocole pour ces trois services, voyons le service de base, le "*Map Service*", d'abord pour les tables, puis pour les coûts.

D'abord, l'accès aux informations sur le réseau, la "*Network Map*" (section 5). Elle part d'une observation : il est coûteux de distribuer l'information sur chaque machine connectée et c'est surtout inutile, les machines se groupant selon la topologie du réseau. Une "*Network Map*" est donc surtout une série de regroupements. Chaque regroupement est identifié par un PID ("*Provider-defined IDentifier*") qui, comme son nom l'indique, n'a de sens que pour un opérateur donné. Chaque groupe désigné par un PID va comprendre un ou plusieurs préfixes d'adresses IP (d'autres identificateurs des extrémités sont possibles). Un PID peut désigner toute une ville, ou bien tout ce qui est connecté à un POP donné. On verra plus loin que les coûts sont indiqués par PID et pas par préfixe ou par adresse, de manière à diminuer la quantité d'information à manier. Dans le futur, des groupes pourront être identifiés par autre chose qu'un PID, par exemple un pays ou un AS. Un registre IANA <https://www.iana.org/assignments/alto-protocol/alto-protocol.xhtml#endpoint-property-types> a été créé pour stocker ces futurs propriétés des extrémités. À la réunion IETF de Toronto, en 2014, ont été ainsi discutées des propriétés comme « type d'accès » (ADSL ou 3G...), « volume limité » (pour les offres illimitées des opérateurs 3G/4G, etc.

Un exemple d'une table triviale serait :
— PID1 regroupe `192.0.2.0/24` et `198.51.100.0/25`.
— PID2 regroupe `198.51.100.128/25`.
— PID3 regroupe `0.0.0.0/0` (c'est-à-dire tout le reste de l'Internet).

Ensuite, la table des coûts ("*Cost Map*", section 6). Elle est souvent plus dynamique que la table du réseau (ce qui justifie leur séparation) et indique les coûts vus par l'opérateur. Un « type de coût » comprend une métrique (une définition de comment on mesure un coût) et un mode (comment comparer les coûts, notamment est-ce qu'on peut appliquer des opérations arithmétiques comme l'addition). Les métriques possibles sont enregistrées à l'IANA <https://www.iana.org/assignments/alto-protocol/alto-protocol.xhtml#cost-metrics> et l'enregistrement de nouvelles métriques requiert un RFC qui ne soit pas un document individuel.

Au moins une métrique doit être utilisée par le serveur, `routingcost`, qui doit refléter le coût de routage. Elle n'est pas davantage définie donc cela peut être un nombre de kilomètres à vol d'oiseau, un nombre de sauts IP, un calcul à partir des coûts indiqués par l'IGP, etc. Quant aux modes, deux sont obligatoires, `numerical` et `ordinal`. Le premier utilise des nombres en virgule flottante et permet toutes les opérations arithmétiques classiques (comme l'addition de deux coûts), le second utilise des entiers et ne permet que les comparaisons. À la réunion IETF de Toronto, en juillet 2014, on a discuté de futures métriques comme par exemple le taux de perte de paquets, la gigue...

Une table des coûts triviale serait :
— En partant de PID1, le coût d'aller à PID2 est de 2 et le coût vers tout l'Internet (PID3) est de 5,
— En partant de PID2, le coût pour aller à PID1 est de 1 (les coûts ne sont pas forcément symétriques) et le coût vers le reste de l'Internet (PID3) de 4.

Pour la plupart des exemples ci-dessous, je vais utiliser un des rares serveurs ALTO publics <http://www.ietf.org/mail-archive/web/alto/current/msg01789.html> existants, `http://alto.alcatel-luc`
Le plus simple est d'utiliser curl, combiné avec jsonlint <http://deron.meranda.us/python/demjson/> pour mieux afficher le JSON :

```
% curl -s http://alto.alcatel-lucent.com:8000/network/full | jsonlint -f
{
  "meta" : { "vtag" : {
          "resource-id" : "default-network",
          "tag" : "192.11.155.88/f9d21e031e5cf58103d4687d65025cfb"
        } },
  "network-map" : {
```

```
      "defaultpid" : {
          "ipv4" : [ "0.0.0.0/0" ],
          "ipv6" : [ "::/0" ]
        },
      "mypid1" : { "ipv4" : [
              "15.0.0.0/8",
              "10.0.0.0/8"
            ] },
      "mypid2" : { "ipv4" : [ "192.168.0.0/16" ] },
      "mypid3" : { "ipv4" : [ "192.168.10.0/24" ] },
      "peeringpid1" : { "ipv4" : [ "128.0.0.0/16" ] },
      "peeringpid2" : {
          "ipv4" : [ "130.0.0.0/16" ],
          "ipv6" : [ "2001:DB8::/32" ]
        },
      "transitpid1" : { "ipv4" : [ "132.0.0.0/16" ] },
      "transitpid2" : { "ipv4" : [ "135.0.0.0/16" ] }
    }
}
```

Et voilà, on a la table complète du réseau d'un FAI (fictif). La table contient des préfixes IP, avec le PID associé (par exemple, `128.0.0.0/16` a le PID `peeringpid1`).

La section 8 de notre RFC en vient au protocole : ALTO est un protocole REST pour accéder à des ressources (les tables) et à des annuaires de ressources, les IRD ("*Information Resource Directory*"). Les ressources sont récupérées en HTTP et ont donc un type de média indiqué (`application/alto-networkmap+json` pour la table du réseau, et `application/alto-costmap+json` pour la table des coûts). Ces types MIME ont été enregistrés dans le registre officiel <`https://www.iana.org/assignments/media-types/media-types.xhtml`>. En général, les requêtes ALTO se font avec la méthode HTTP `GET`. Comme tous les protocoles s'appuyant sur HTTP, ALTO a à sa disposition toutes les fonctions rigolotes de HTTP, comme la mise en cache (RFC 7234). De même, la sécurité est fournie par les mécanismes HTTP habituels (HTTPS, authentification du RFC 7236, etc). Les codes de réponse habituels de HTTP sont utilisés, comme 200 si tout s'est bien passé, 405 si la méthode utilisée ne plait pas au serveur et 503 si le serveur a un problème temporaire.

Si la requête HTTP s'est bien passée, mais qu'un problème spécifique à ALTO est survenu, le serveur renvoie des données JSON de type `application/alto-error+json`. Elles comprennent au moins un champ `code` qui indiquera le type d'erreur par exemple `E_SYNTAX` pour des paramètres d'entrée incorrects, `E_MISSING_FIELD` pour l'absence d'un champ obligatoire, etc. Testons avec le serveur public, en envoyant du JSON incorrect (accolade ouvrante sans la fermante) :

```
% curl -s -d '{' -H "Content-Type: application/alto-networkmapfilter+json" \
   -X POST http://alto.alcatel-lucent.com:8000/network/filtered | jsonlint -f
{ "meta" : {
      "code" : "E_SYNTAX",
      "syntax-error" : "Premature EOF in JSON object; expecting key"
    } }
```

Les erreurs possibles sont dans un registre IANA <`https://www.iana.org/assignments/alto-protocol/alto-protocol.xhtml#error-codes`> (section 14.5 de notre RFC).

La section 9 présente l'IRD, l'"*Information Resource Base*", le mécanisme de distribution de méta-information (où trouver la "*Network Map*", où trouver la "*Cost Map*", etc). Elle a aussi la forme de données JSON, servies sous le type `application/alto-directory+json`. Voici celle du serveur ALTO public indiqué plus haut :

```
% curl -s http://alto.alcatel-lucent.com:8000/directory | jsonlint -f
{
  "meta" : {
      "cost-types" : {
          "hop-num" : {
              "cost-metric" : "hopcount",
              "cost-mode" : "numerical",
              "description" : "Simple hopcount"
            },
...
      "default-alto-network-map" : "default-network",
      "priv:alu-server-info" : {
          "Build-Date" : "2014-04-28 13:57:08 EDT",
          "Build-OS" : "Mac OS X 10.9.2",
          "Build-OS-Nodename" : "wdr-i7mbp2.mh.lucent.com",
          "Build-User" : "wdr",
          "Implementation-Title" : "ALTO Server Implementation",
          "Implementation-Vendor" : "Alcatel-Lucent (Bell Labs/Murray Hill)",
          "Implementation-Version" : "(2014-04-28 13:57:08 EDT)",
...
  "resources" : {
      "costs-end" : {
          "accepts" : "application/alto-endpointcostparams+json",
          "capabilities" : {
              "cost-constraints" : true,
              "cost-type-names" : [
                  "rtg-num",
                  "rtg-ord",
                  "hop-ord",
                  "hop-num"
                ]
            },
          "media-type" : "application/alto-endpointcost+json",
          "uri" : "http://alto.alcatel-lucent.com:8000/costs/end"
        },
...
      "default-network" : {
          "media-type" : "application/alto-networkmap+json",
          "uri" : "http://alto.alcatel-lucent.com:8000/network/full"
        },
...
    }
}
```

On y voit, par exemple, que si je veux obtenir la table complète du réseau, je dois demander `http://alto.alcatel-l` comme fait plus haut.

Bon, ce n'est pas tout, ça, il faut maintenant des données. La section 10 indique les types de base qui sont utilisés : les PID (identificateurs des réseaux) sont des chaînes de caractères ASCII, les ressources comme les "*Network Maps*" ont également un identificateur, le "*Resource ID*" (encore une chaîne ASCII), les extrémités sont en général identifiées par une adresse ou un préfixe IP, à la syntaxe traditionnelle, les mode et métriques des coûts sont des chaînes ASCII...

Avec ces types, on peut construire les données. Quand un client ALTO a l'adresse d'un pair potentiel, et cherche dans la table le PID correspondant, il peut y avoir plusieurs préfixes qui correspondent. Dans ce cas, c'est le plus spécifique qui gagne (règle dite du « "*longest match*" », cf. RFC 1812). Pour éviter toute ambiguité, la table doit être complète et sans recouvrement. Complète signifie que toutes les adresses doivent être présentes. Dans le cas d'un opérateur ayant une connectivité avec tout l'Internet, cela veut dire qu'il faut représenter toutes les adresses de l'Internet. Une entrée pour le réseau `0.0.0.0/0` en IPv4 et `::0/0` en IPv6 est le moyen le plus simple d'atteindre ce but : cette entrée, attrapera toutes les adresses non couvertes par un des préfixes plus spécifiques. Quant au non-recouvrement, il signifie que

la table ne doit pas avoir deux préfixes identiques (il peut y avoir recouvrement mais uniquement entre préfixes de longueurs différentes, la règle du "*longest match*" permettra alors de sélectionner).

Par exemple, cette table est complète et sans recouvrement :

```
   "network-map" : {
     "PID0" : { "ipv6" : [ "::/0" ] },
     "PID1" : { "ipv4" : [ "0.0.0.0/0" ] },
     "PID2" : { "ipv4" : [ "192.0.2.0/24", "198.51.100.0/24" ] },
     "PID3" : { "ipv4" : [ "192.0.2.0/25", "192.0.2.128/25" ] }
   }
```

Avec cette table du réseau, `192.0.2.1` sera dans `PID3` (le préfixe dans `PID2` est moins spécifique), `198.51.100.200` sera dans `PID2` et `203.0.113.156` sera dans `PID1`, en raison de la règle attrape-tout qui couvre tout l'Internet IPv4. Notez l'utilisation de la désagrégation : `192.0.2.0/24` a été représenté avec deux préfixes dans `PID3` car, autrement, la table aurait eu un recouvrement.

Et la table des coûts ? Voici un exemple :

```
% curl -s http://alto.alcatel-lucent.com:8000/costs/full/routingcost/num | jsonlint -f
{
  "cost-map" : {
      "defaultpid" : {
          "defaultpid" : 15.0,
          "mypid1" : 15.0,
          "mypid2" : 15.0,
          "mypid3" : 15.0,
          "peeringpid1" : 15.0,
          "peeringpid2" : 15.0,
          "transitpid1" : 15.0,
          "transitpid2" : 15.0
        },
      "mypid1" : {
          "defaultpid" : 4.0,
          "mypid1" : 15.0,
          "mypid2" : 15.0,
          "mypid3" : 15.0,
          "peeringpid1" : 15.0,
          "peeringpid2" : 15.0,
          "transitpid1" : 5.0,
          "transitpid2" : 10.0
        },
...
      "peeringpid1" : {
          "defaultpid" : 15.0,
          "mypid1" : 15.0,
          "mypid2" : 15.0,
          "mypid3" : 15.0,
          "peeringpid1" : 15.0,
          "peeringpid2" : 15.0,
          "transitpid1" : 15.0,
          "transitpid2" : 15.0
        },
...
    },
  "meta" : {
      "cost-type" : {
          "cost-metric" : "routingcost",
          "cost-mode" : "numerical"
        },
```

```
      "dependent-vtags" : [ {
            "resource-id" : "default-network",
            "tag" : "192.11.155.88/f9d21e031e5cf58103d4687d65025cfb"
          } ]
    }
}
```

On y voit, par exemple, que le coût entre

`mypid1`

et

`peeringpid1`

est de 15 (en utilisant la métrique `routingcost`).

On l'a vu, le client ALTO peut ne demander qu'une partie d'une table, pour réduire la consommation de ressources réseau et de temps de calcul. Ici, un client envoie un objet JSON (de type `application/alto-networkmapf` qui indique quel(s) PID l'intéresse(nt) :

```
POST /networkmap/filtered HTTP/1.1
Host: custom.alto.example.com
Content-Length: 33
Content-Type: application/alto-networkmapfilter+json
Accept: application/alto-networkmap+json,application/alto-error+json

{
     "pids": [ "PID1", "PID2" ]
}
```

Notez qu'on utilise la méthode HTTP `POST` au lieu de `GET`. On peut aussi filtrer les tables de coûts, par exemple en n'acceptant qu'un seul mode.

Et voici une récupération des propriétés d'une extrémité, l'adresse de l'extrémité étant indiquée dans l'objet JSON `application/alto-endpointpropparams+json` :

```
POST /endpointprop/lookup HTTP/1.1
Host: alto.example.com
Content-Length: 181
Content-Type: application/alto-endpointpropparams+json
Accept: application/alto-endpointprop+json,application/alto-error+json

{
    "properties" : [ "my-default-networkmap.pid",
                     "priv:ietf-example-prop" ],
    "endpoints"  : [ "ipv4:192.0.2.34",
                     "ipv4:203.0.113.129" ]
}

HTTP/1.1 200 OK
Content-Length: 396
Content-Type: application/alto-endpointprop+json

...
    "endpoint-properties": {
      "ipv4:192.0.2.34"    : { "my-default-network-map.pid": "PID1",
                               "priv:ietf-example-prop": "1" },
      "ipv4:203.0.113.129" : { "my-default-network-map.pid": "PID3" }
    }
  }
```

Et ici une récupération des coûts associés à une extrémité. Par exemple, on est un client BitTorrent qui hésite entre trois pairs potentiels de l'essaim, `192.0.2.89`, `198.51.100.34` et `132.0.113.45`. On les envoie au serveur (notez les conséquences pour la vie privée : ce point est développé plus loin) :

```
# Les données JSON de la requête sont mises dans un fichier
% cat /tmp/select
{
    "cost-type": {"cost-mode" : "ordinal",
                  "cost-metric" : "routingcost"},
    "endpoints" : {
      "srcs": [ "ipv4:10.0.2.2" ],
      "dsts": [
        "ipv4:192.0.2.89",
        "ipv4:198.51.100.34",
        "ipv4:132.0.113.45"
      ]
    }
  }

# On donne ce fichier à curl comme source des données à envoyer
# (option -d)
% curl -s -d '@/tmp/select' -H "Content-Type: application/alto-endpointcostparams+json" \
   -X POST http://alto.alcatel-lucent.com:8000/costs/end | jsonlint -f
```

Et le serveur ALTO répond en indiquant des coûts :

```
{
  "endpoint-cost-map" : { "ipv4:10.0.2.2" : {
          "ipv4:132.0.113.45" : 2,
          "ipv4:192.0.2.89" : 1,
          "ipv4:198.51.100.34" : 1
        } },
  "meta" : { "cost-type" : {
          "cost-metric" : "routingcost",
          "cost-mode" : "ordinal"
        } }
}
```

Ici, depuis la source `10.0.2.2`, les meilleurs pairs sont `192.0.2.89` et `198.51.100.34`, ex-aequo.

Pour rendre les choses plus concrètes, la section 12 du RFC présente un certain nombre de scénarios d'usage. Le premier, et le plus « évident » est celui d'un client ALTO placé dans un "*tracker*" pair à pair, un de ces logiciels qui aiguillent les pairs vers un pair qui a le contenu convoité. Chaque "*tracker*" gère des essaims (ensemble de pairs qui téléchargent le même fichier et a la tâche délicate de choisir quels pairs sont annoncés à chaque machine connectée au "*tracker*" (sauf si l'essaim est petit, auquel cas la tâche est triviale, on envoie tout). Les "*trackers*" utilisent déjà un ensemble de mécanismes pour cela et ils peuvent y ajouter ALTO. Comme un "*tracker*" pour un contenu populaire (mettons le dernier épisode d'une série télévisée très populaire) peut avoir des dizaines, voire des centaines de milliers de machines connectées, la meilleure solution ALTO est sans doute de télécharger la table du réseau et la table des coûts complètes et de les analyser. Pour chaque membre de l'essaim, le "*tracker*" cherchera le PID dans la table du réseau, puis le coût dans la table des coûts. Il enverra les pairs de préférence vers un pair du même PID ou, sinon, vers le pair au coût le plus bas.

Mais, pour diverses raisons, les pairs peuvent souhaiter faire la sélection eux-même. Les "*trackers*" sont des cibles évidentes pour la répression du pair à pair (ou, tout simplement, des SPOF...) et on peut vouloir se débrouiller sans eux, utilisant un mécanisme de rendez-vous complètement pair à pair

comme une DHT. Dans ce cas, le client ALTO doit forcément être dans le logiciel de partage pair à pair (par exemple rTorrent). La technique la plus coûteuse, mais aussi celle qui préserve le mieux la vie privée de l'utilisateur, est que le logiciel télécharge la table du réseau et la table des coûts, puis fasse les calculs lui-même. Ainsi, il ne donnera presque aucune information au serveur ALTO.

Troisième scénario d'usage proposé par notre RFC, la sous-traitance des calculs et du filtrage au serveur ALTO, en utilisant le "*Map Filtering Service*". Le logiciel pair à pair, cette fois, ne télécharge pas les tables entières, il indique au serveur ALTO une liste de pairs potentiels (trouvée, par exemple, sur la DHT), des critères de choix, et le serveur ALTO choisit le « meilleur » selon ces critères.

Notre RFC comprend ensuite une section 13 de discussion sur divers points du protocole. Premier point : comment un client trouve-t-il le serveur ALTO ? La norme ne définit pas une méthode unique et obligatoire. Le serveur peut être trouvé par une configuration manuelle, ou bien par la technique DNS du RFC 7286 sur la découverte de serveurs ALTO.

Et le cas d'une machine qui a plusieurs adresses IP ? C'est d'autant plus compliqué qu'il y a une interaction entre l'adresse utilisée, le FAI par lequel on passe et l'adresse du pair distant. ALTO ne fournit pas de solution immédiate, uniquement un mécanisme de description des extrémités suffisamment souple pour qu'on puisse y intégrer d'autres identificateurs que les bêtes adresses IP (voir plus haut les registres IANA de types d'adresses).

Reste à régler la question du passage par les routeurs NAT, le cauchemar habituel du pair à pair. En raison de ces obstacles, un protocole devrait éviter de transporter des adresses IP dans ses données, justement ce que fait ALTO. On a vu que, dans certains cas, le client ALTO indique son adresse au serveur, pour lui permettre un bon choix du pair. En raison du NAT, cette méthode ne marche pas forcément et le RFC autorise donc le serveur ALTO à utiliser l'adresse IP source des paquets ALTO entrants (adresse traduite par le routeur NAT) plutôt que celle indiquée dans les données JSON. Si le client ALTO n'est pas satisfait et veut indiquer lui-même son adresse IP dans les données, il doit utiliser un service comme STUN (RFC 8489) pour découvrir son adresse publique.

Et la sécurité dans tout ça ? La section 15 du RFC explore en détail les questions de sécurité d'ALTO. Le RFC cahier des charges, le RFC 6708, reconnaissait le problème et demandait des solutions minimum. D'abord, l'authentification. En effet, un attaquant peut vouloir distribuer de la fausse information ALTO, pour tromper un pair et l'envoyer vers des pairs plus lents. Cela serait une attaque par déni de service sur le réseau pair à pair. ALTO s'appuyant sur HTTP, les solutions de sécurité à utiliser sont celles de HTTP, essentiellement TLS. Pour l'authentification via un certificat, les clients et serveurs ALTO doivent bien suivre le RFC 6125. Attention, cette authentification se fait en fonction du nom présent dans l'URL utilisé par le client ALTO. Si celui-ci utilise un mécanisme de découverte de l'URL qui n'est pas fiable, la protection de TLS arrivera trop tard. Aujourd'hui, il n'y a pas de protection des données ALTO (pas de signature des "*maps*", par exemple), seulement du canal par lequel elles transitent.

C'est bien joli d'authentifier le serveur ALTO mais encore faut-il qu'il soit digne de confiance. Si un méchant serveur ALTO distribue de mauvaise informations, comment peut-on se protéger ? Le RFC 5693 pointait déjà ce risque. Si les serveurs ALTO sont gérés par les FAI, comme beaucoup d'entre eux ont affirmé leur goût pour les violations de la neutralité du réseau <https://www.bortzmeyer.org/neutralite.html>, pourquoi les clients leur feraient-ils confiance ? Deux approches sont possibles pour le client ALTO. La première est d'analyser les conséquences de son choix. Si un logiciel pair à pair utilise parfois ALTO et parfois choisit les pairs au hasard, en mesurant les débits obtenus, il peut savoir si l'utilisation d'ALTO est dans son intérêt. (Attention, le but d'ALTO n'est pas uniquement la satisfaction du client : un FAI peut légitimement envoyer les clients vers des pairs qui ne sont pas plus rapides mais qui lui coûtent moins cher, par exemple parce qu'ils sont accessibles par un lien de "*peering*" gratuit.) Une seconde approche pour le client est de ne pas utiliser les serveurs ALTO du FAI (qui n'a pas forcément

les mêmes intérêts que lui <https://www.bortzmeyer.org/tussle-cyberspace.html>) mais des serveurs ALTO « neutres » fournis par une tierce partie (les exigences AR-20 et AR-21 dans le RFC 6708). Rappelez-vous aussi qu'ALTO n'est qu'une source d'information parmi d'autres, le client l'utilise pour sa prise de décision mais ne lui obéit pas aveuglément.

Et la confidentialité ? Bien sûr, une bonne partie des informations distribuées par ALTO sont publiques ou presque. La "*network map*" peut être vue comme confidentielle par le FAI mais, si elle est distribuée à tous ses clients, elle ne peut pas rester trop longtemps secrète. Néanmoins, on ne souhaite pas forcément que n'importe quel espion lise tout le trafic ALTO. Après tout, en combinant la table du réseau et celle des coûts, un attaquant peut récolter plein d'information sur le réseau d'un opérateur, à des fins d'espionnage industriel, ou bien pour préparer une attaque par déni de service. Et les informations données par les clients (liste de pairs potentiels...) sont également très sensibles. Que peut-on faire contre l'espionnage ? D'abord, au moment de créer les tables qui seront utilisées par ALTO, il faut se rappeler qu'ALTO n'impose pas un niveau de détail particulier. Le FAI qui veut avant tout optimiser le trafic fournira un grand niveau de détails, celui qui craint l'espionnage économique donnera des tables plus grossières. Ensuite, pour éviter l'écoute par un tiers, ALTO utilise, comme pour l'authentification, les solutions habituelles de HTTP, en l'occurrence TLS.

Notre RFC se penche aussi sur la vie privée des clients ALTO. D'abord, en raison de l'utilisation de HTTP, certains des risques de vie privée de HTTP affectent ALTO (par exemple l'utilisation de "*cookies*", pas obligatoire dans ALTO mais qui peuvent être activés par défaut par certains clients HTTP). Ensuite, il y a clairement des risques de confidentialité différents pour le client et pour le serveur ALTO. Le client soucieux de vie privée a tout intérêt à télécharger l'intégralité des tables (réseau et coûts) et à faire la sélection soi-même, ne transmettant ainsi aucune information au serveur. En revanche, le serveur soucieux de sa vie privée à lui va vouloir le contraire : ne pas envoyer les tables complètes mais demander au client de lui fournir des critères de filtrage, avec lesquels le serveur fera son choix. On peut donc s'attendre, si ALTO se répand, à des conflits entre utilisateurs et FAI, par exemple si un FAI décide de ne pas fournir le service « tables complètes » et impose l'utilisation des services avec filtrage.

Dernier risque de sécurité pour ALTO, le risque de panne. Si on est habitué à utiliser ALTO, on sera bien embêté si un attaquant trouve le moyen de planter le serveur ALTO. Cela peut servir à aggraver une attaque par déni de service. Ce risque est d'autant plus important qu'un serveur ALTO peut se trouver, si on lui demande de faire des filtrages, dans l'obligation de faire des gros calculs, décidés par son client. Un client malveillant peut ainsi générer facilement une table avec N origines et M destinations, forçant le serveur à calculer N*M chemins. Un serveur ALTO doit donc être prudent sur ce point et prévoir de la limitation de trafic, et sans doute de l'authentification des clients (par exemple, pour un FAI, restreindre l'accès à ses abonnés).

Reste la gestion d'ALTO (section 16). Suivant le RFC 5706, cette section va se pencher sur les problèmes concrets des pauvres ingénieurs système qui vont devoir déployer ALTO et le maintenir en état de marche. ALTO reposant sur HTTP, cela impose l'installation et le maintien d'un serveur HTTP, soit un serveur standard avec un module ALTO, soit un serveur ALTO incluant le protocole HTTP. Mais il y a des problèmes plus sérieux, notamment les choix des services rendus. Par exemple, quel niveau de détail envoie-t-on, quelles requêtes permet-on ? Si on refuse d'envoyer la table du réseau complète, que doit-on faire lorsqu'une requête avec filtrage arrive, avec un filtre vide (un client ALTO astucieux peut utiliser cela pour contourner les restrictions d'accès à la table complète) ? Il faut aussi décider d'un mécanisme pour distriber l'URL où trouver le serveur ALTO. Dans la documentation accessible aux utilisateurs ? Uniquement avec la méthode du futur RFC sur l'utilisation du DNS pour trouver le serveur ? ? (Le RFC n'en parle pas mais le faible nombre de clients ALTO existants n'aide pas à trancher sur ces points.) Heureusement, il n'y a pas encore de question de migration à assurer, puisque ALTO est un nouveau service.

Pour l'opérateur réseau, ALTO va, s'il est utilisé, mener à des déplacements du trafic vers d'autres lignes. C'est bien son but mais cela peut entraîner des surprises : le responsable du serveur ALTO a désormais les moyens d'influencer sur la localisation du trafic, autant que celui qui configurait MPLS ou OSPF.

Pour la journalisation, le RFC recommande que les serveurs ALTO utilisent le standard syslog (RFC 5424), ce que ne font typiquement pas les serveurs HTTP habituels.

ALTO ne fournit pas encore d'interface standard de communication entre serveurs ALTO (par exemple pour agréger de l'information provenant de diverses sources), ni de MIB, ni d'interface standard de configuration, style NETCONF.

Pour les passionnés d'histoire, l'annexe B rappelle tous les ancêtres d'ALTO, notamment P4P (voir « "*P4P : Provider Portal for (P2P) Applications*" », présenté à SIGCOMM en 2008.

Et les mises en œuvre existantes ? On a vu le serveur d'Alcatel-Lucent. Il existe aussi celui fait à partir d'un dessin fameux de XKCD <http://alto.tilab.com/alto-xkcd>. Trois mises en œuvre différentes ont été montrées à la réunion IETF 80 <http://www.ietf.org/meeting/80/index.html> et bien plus ont été testées rigoureusement lors d'un test d'interopérabilité à l'IETF 81 <http://www.ietf.org/meeting/81/index.html>. Les résultats sont documentés en ligne <http://www.ietf.org/mail-archive/web/alto/current/msg01181.html>. Un autre test a eu lieu à l'IETF 85 <http://www.ietf.org/meeting/85/index.html> et est également documenté <http://www.ietf.org/proceedings/85/slides/slides-85-alto-0.pdf>.

Merci à Wendy Roome pour le serveur ALTO public utilisé ici, et pour ses patientes explications.