RFC 6538 : HIP Experiment Report

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 mars 2012

Date de publication du RFC : Mars 2012

https://www.bortzmeyer.org/6538.html

Le protocole HIP <https://www.bortzmeyer.org/hip-resume.html>, normalisé dans le RFC 4423 [1], est maintenant stable depuis des années et a fait l'objet de plusieurs essais sur le terrain. Ce RFC de l'IRTF documente certains de ces essais concrets et indique les leçons à en tirer.

Le groupe de recherche hip <http://irtf.org/hiprg> de l'IRTF a travaillé sur ce sujet de 2004 à 2009. Pendant cette période, un certain nombre de mises en œuvre de HIP ont été programmées et plusieurs bancs de test ont été montés. La section 7 du RFC donne une liste exhaustive des expériences menées, notamment chez Boeing et dans le projet InfraHIP <http://infrahip.hiit.fi>, qui gère des serveurs HIP publics. (Notez que, HIP fonctionnant au dessus d'IP, deux machines HIP peuvent communiquer au dessus de l'Internet public sans problème.) Ce RFC est le compte-rendu de ces expériences, sur les points suivants :

— Implémenter HIP, qu'est-ce que ça coûte, comme efforts ?
— Déployer HIP, quels sont les problèmes ?
— Une fois que HIP marche, quelles conséquences pour les applications ?
— Qu'en pensent les opérateurs ?
— Et la vie privée, que va-t-elle devenir ?

D'abord, la mise en œuvre (section 2). HIP est un protocole pour les machines terminales <https://www.bortzmeyer.org/terminal-host.html>, pas pour les routeurs. Implémenter HIP, c'est donc modifier Linux ou Windows, sans toucher à ses Cisco ou D-Link. Il existe aujourd'hui trois de ces modifications sous une licence libre :

— HIP4BSD <http://www.hip4inter.net>, pour FreeBSD,
— HIPL <http://hipl.hiit.fi>, pour Linux,
— et OpenHIP <http://www.openhip.org>, la seule qui fonctionne entièrement en mode utilisateur, et qui tourne sur Linux, Windows XP/Vista/7, and Apple OS X.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc4423.txt

L'expérience montre bien qu'implémenter un nouveau protocole n'est pas du yakafokon et que plein de pièges insoupçonnés se cachent dans le code. La section 2.1 résume ce qu'il faut modifier au noyau pour intégrer HIP. Les applications fonctionnent comme avant, un HIT (le condensat cryptographique d'une clé HIP) ayant la même forme qu'une adresse IPv6 (avec le préfixe ORCHID du RFC 4843). En section 2.2, c'est le cas d'une implémentation en dehors du noyau qui est couvert (notez que certaines mises en œuvre d'HIP sont mixtes, par exemple avec le traitement des paquets dans le noyau et la cryptographie dans un démon à part). Cela permet de faire fonctionner HIP sur des systèmes qu'on n'a pas le droit de modifier comme Microsoft Windows.

Dans tous les cas, quels ont été les problèmes rencontrés ? En l'absence de système de résolution, manipuler les HIT à la main, se les échanger, etc, a été pénible. Des systèmes de résolution existent désormais (mettre les HIT dans le DNS sous forme d'enregistrements AAAA, comme le propose le RFC 5205, utiliser une DHT dédiée à HIP, etc) et OpenHIP les a tous programmés. Et il y a aussi le mode opportuniste de HIP, où on se connecte sans connaître a priori le HIT du pair. Mais aucune solution n'est à 100 % parfaite.

Le mode opportuniste du RFC 5201, section 4.1.6 a posé des problèmes avec l'interface "*sockets*", car le HIT du pair distant n'est pas encore connu au moment du `connect()`. De toute façon, ce mode a l'inconvénient de ne pas être très sécurisé (il fonctionne en mode TOFU - "*Trust On First Use*" - puisque, la première fois qu'on parle à un pair, on doit accepter son HIT en lui faisant une confiance aveugle). HIPL a mis en œuvre ce mode. OpenHIP également, avec une option explicite (`-o`), débrayée par défaut. Le problème de sécurité est particulièrement crucial lorsqu'il s'agit d'applications traditionnelles, qui ne connaissent pas HIP et ne peuvent donc pas formuler de souhaits, ou interactivement demander confirmation à l'utilisateur, ce que peuvent faire les applications récentes qui utilisent le RFC 6317.

Quant au DNS, certains administrateurs de zones DNS n'ont pas aimé qu'on mette des HIT sous le nom d'AAAA, arguant que c'était un détournement de la sémantique du AAAA.

Autre problème, lorsqu'on connait le HIT du pair (on ne fait donc pas de mode opportuniste) mais pas son adresse IP. Pour lui envoyer des paquets, il faudra alors une résolution, pour trouver cette adresse. Des essais ont été faits avec OpenDHT <`https://www.bortzmeyer.org/opendht-debut.html`>, ou bien avec le DNS. HIPL a même tenté, pour le cas particulier du LAN, d'envoyer un paquet de diffusion, un peu comme on fait en NDP. Cela a paniqué certains antivirus Windows qui considéraient toute diffusion avec un protocole inconnu comme un danger.

Parmi les nombreuses autres questions que décrit le RFC en tenant compte de l'expérience des programmeurs HIP, celui de la gestion des HI ("*Host Identifiers*", l'identité d'une machine HIP, dont le HIT est le condensat cryptographique). Une machine peut avoir plusieurs identités (par exemple pour protéger la vie privée, voir plus loin), et laquelle doit-elle utiliser lorsqu'elle se connecte à un pair ? Dans quelle mesure le choix doit-il être fait par l'utilisateur ? (Intéressant problème d'interface.) Plus gênant, puisque HIP repose sur des clés cryptographiques, comment stocker les clés privées sur la machine ? Toutes les implémentations actuelles utilisent un fichier appartenant à root en mode 0600 (ce qui interdit la lecture par les utilisateurs ordinaires). Cela ne suffit pas si, par exemple, un attaquant a un accès physique à la machine. La solution serait alors de stocker les clés dans un TPM mais aucune mise en œuvre de HIP n'a encore tenté le coup.

Cette section comprend également un problème qui, à mon avis, est plutôt de déploiement que de programmation, l'interaction avec les pare-feux. HIP est un protocole (le numéro 139 des protocoles IP, TCP étant 6, UDP 17 et OSPF 89) marqué comme expérimental et qui est globalement peu connu. Il n'y a donc pas de règle pour lui dans les règles configurées par défaut et le pare-feu refusant tout ce qui n'est pas explicitement autorisé, les paquets HIP se retrouvent bloqués. Il n'est pas forcément trivial de modifier les règles sans tout casser (le RFC rapporte que celles par défaut de SUSE ont plus de cent

entrées) et, de toute façon, l'utilisateur qui veut déployer HIP n'a pas forcément le contrôle du pare-feu. Heureusement, HIP peut tourner sur UDP (au lieu de directement sur IP) ce qui limite les problèmes.

Bien que HIP ait été surtout conçu pour IPv6, et que son déploiement massif ne semble pas devoir précéder celui d'IPv6, toutes les mises en œuvre de HIP ont choisi de gérer également IPv4. Comme on ne peut absolument pas faire rentrer un condensat cryptographique sérieux dans les 32 bits d'une adresse IPv4, il faut ajouter le concept de LSI ("*Local Scope Identifier*"), un alias local à la machine pour référencer un HI donné.

Plus amusant, HIP permet également de faire tourner des applications IPv6 sur un réseau IPv4 (et réciproquement), ce qui fonctionne sur tous les codes cités.

Bref, les programmeurs ont bien travaillé et HIP fonctionne sur plusieurs plate-formes. Quelles leçons peuvent être tirées ? Il n'y a pas eu beaucoup de tests de performances, notamment pour comparer les implémentations en mode noyau et en mode utilisateur. Il semble que, sur une machine généraliste et pour des usages typiques, le mode utilisateur (le plus lent) est acceptable.

Le choix entre les deux modes peut dépendre de bien d'autres choses. Par exemple, HIPL avait une mise en œuvre des fonctions de gestion de clés dans le noyau, mais les développeurs du noyau Linux l'ont rejeté, arguant que d'autres protocoles comme IKE faisaient cette gestion en mode utilisateur et qu'il n'y avait pas de raison de l'intégrer dans le noyau. (Il faut dire aussi que c'était un gros "*patch*", qui touchait à beaucoup d'endroits du noyau.)

La section 3 aborde ensuite les points liés au déploiement, notamment à l'effet sur les infrastructures. La section sur le DNS (extensions HIP du RFC 5205) est curieusement incompréhensible (qu'est-ce qu'un "*binary blob format*" ?) mais, de toute façon, les serveurs DNS peuvent servir des types de données qu'il ne connaissent pas, donc peu de problèmes à attendre.

Plus sérieux, le cas des "*middleboxes*". Les NAT sont hostiles à HIP comme à beaucoup de choses (problème décrit dans le RFC 5207). HIP peut par exemple chiffrer même la connexion initiale, empêchant le routeur NAT de comprendre ce qui se passe. Le mécanisme de rendez-vous du RFC 5204 pourrait aider à résoudre les problèmes du NAT mais il n'a pour l'instant été testé qu'en laboratoire.

Et l'infrastructure de résolution ? Une mise en œuvre de résolution de HIT en adresses IP via OpenDHT <https://www.bortzmeyer.org/opendht-debut.html> a été faite pour HIPL et OpenHIP. OpenDHT est désormais arrêté mais un nouveau projet a pris le relais, Hi3, qui tourne sur PlanetLab. Au cours des premiers essais, le débogage s'est révélé très pénible, et les "*middleboxes*" qui bloquent tout une énorme source de frustration.

Une solution d'avenir pourrait être de mêler DNS (déploiement massif, compatibilité avec le logiciel existant) et DHT (bonne résistance aux pannes, bonne efficacité même dans un espace plat, celui des clés cryptographiques). Par exemple, une organisation ferait tourner une DHT pour stocker les HI ou les HIT de ses machines, connectée avec le DNS pour les clients extérieurs. Ensuite, ces DHT locales pourraient être interconnectées petit à petit. Un test a été fait en modifiant BIND pour qu'il interroge une DHT (Bamboo <http://bamboo-dht.org/>) pour les HIT. Les performances ont été plutôt mauvaises, et se dégradant vite avec la charge, donc des travaux supplémentaires sont nécessaires.

Et pour les applications ? L'expérience du déploiement d'IPv6 a montré qu'il était plus facile de mettre à jour l'infrastructure que les applications. La section 4 explore les conséquences de HIP sur celles-ci. Il y a deux sortes d'applications : celles qui connaissent HIP et celles qui ne le connaissent pas

(la quasi-totalité, aujourd'hui : le RFC 5338 explique comment faire tourner HIP avec les applications traditionnelles).

Par exemple, HIPL a testé une bibliothèque dynamique (chargée avec `LD_PRELOAD`) qui change le comportement du résolveur, permettant aux applications de se connecter à un pair HIP sans s'en rendre compte. Un Firefox et un Apache non modifiés ont ainsi pu se parler en HIP. L'avantage de cette méthode est de permettre de choisir, application par application, qui va faire du HIP (cette finesse de choix pouvant être considérée comme un avantage ou comme un inconvénient). Une autre solution serait de passer par des relais applicatifs mais elle ne semble pas avoir été testée encore.

HIP a même réussi à fonctionner avec des applications qui utilisent des références (passer son adresse IP au pair pour qu'il vous envoie ensuite des messages) comme FTP. D'autres applications seraient peut-être moins tolérantes mais, de toute façon, en raison du NAT, les applications qui se servent de références ont déjà beaucoup de problèmes dans l'Internet d'aujourd'hui.

Une des raisons pour lesquelles il peut être intéressant de mettre une gestion de HIP explicite dans les applications est la performance : l'établissement d'une connexion est plus long en HIP et certaines applications (par exemple un navigateur Web) peuvent être très sensibles à cette latence supplémentaire. Elles pourraient alors choisir de laisser tomber la sécurité de HIP au profit de la vitesse.

Continuons dans les problèmes concrets : quelle va être l'expérience de l'administrateur réseaux qui, le premier, tentera de déployer HIP ? Pour l'instant, aucun ne l'a fait mais. HIP peut tourner entièrement dans les machines terminales, sans que le réseau soit impliqué. Si celui-ci l'est, il aura peut-être les problèmes suivants (détaillés en section 5) :

— Choix des HI ("*Host Identities*"). Celles-ci peuvent être générées par les machines toutes seules, et c'est comme cela que tout le monde fait aujourd'hui. Mais certains administrateurs pourraient préférer le faire eux-même, de manière à connaître les HI de leur réseau. Par exemple, ils généreraient les HI, les signeraient éventuellement avec leur PKI, et les mettraient dans une base AAA, exportant ensuite la configuration d'équipements comme les pare-feux.
— Le monde étant ce qu'il est, la sécurité que fournit HIP, surtout en combinaison avec le mode ESP d'IPsec (qui est très intégrée à HIP) peut ne pas plaire à tout le monde. Si Big Brother veut espionner le trafic, que va pouvoir dire l'administrateur réseaux ? Le groupe de recherche HIP n'a pas trouvé de solution à ce dilemne.
— Autre question posée par le chiffrement, celui de la consommation de ressources, notamment pour les engins les moins pourvus comme les "*smartphones*". Toutefois, au cours d'essais HIP et ESP avec une tablette Nokia N770, les performances sont restées acceptables (3,27 Mb/s en WiFi contre 4,86 sans HIP ni ESP). Le RFC ne dit pas quelle a été la consommation électrique....
— Autre problème typique d'administrateur réseaux, le filtrage. Aujourd'hui, le filtrage va souvent contre la sécurité, par exemple bien des pare-feux bloquent IPsec ou HIP. Normalement, HIP améliore la sécurité puisqu'on peut authentifier les HI (les identités des machines, celles-ci signant leurs paquets). Il ne reste plus qu'à configurer les ACL du pare-feu pour filtrer en fonction du HI. Notons que, contrairement aux adresses IP, celles-ci ne sont pas agrégeables en préfixes et il faut donc typiquement une ACL par machine. Quant au filtrage par nom et plus par HI, il dépend d'un enregistrement sécurisé des HI dans le DNS, qui n'est pas disponible à l'heure actuelle.
— Ceci n'a pas empêché l'université de Technologie d'Helsinski de modifier iptables pour produire un pare-feu HIP. Cela nécessite un état car le pare-feu doit se souvenir de l'échange initial HIP pour reconnaître ensuite les paquets chiffrés liés à cette session (le SPI IPsec figure dans les paquets HIP I2 et R2).

La section 6 du RFC s'attaque ensuite à un difficile problème, celui de la vie privée des utilisateurs. HIP permet enfin de doter chaque machine sur l'Internet d'une véritable identité, indépendante de sa position dans le réseau, qu'on peut vérifier avec la cryptographie. Mais ce gain en sécurité peut se payer en terme de vie privée : si mon ordinateur portable a HIP, je me connecte à un site Web HIP depuis chez

moi, et depuis l'hôtel et, sans "*cookies*" ou autres techniques de suivi Web, le serveur peut reconnaitre que c'est la même machine (les adresses autoconfigurées d'IPv6 avaient un problème du même genre, qui a été résolu par le RFC 4941).

Pire, dans certains cas, un tiers qui écoute le réseau peut apprendre l'identité des deux machines, même si le reste de la session est chiffré.

HIP permet d'utiliser des HIT non publiés (RFC 5201, sections 5.1.2 et 7) mais cela suppose que le destinataire les accepte (ce qui serait logique pour un serveur public, qui ne se soucie pas en général de l'identité de ses clients). Globalement, la protection de la vie privée dans HIP n'est pas encore suffisante mais des travaux sont en cours. Mon opinion personnelle est que la meilleure solution serait analogue à celle du RFC 4941 : des identificateurs jetables, créés pour une courte période (voire pour une seule session) et abandonnés après. Cela nécessite de faire attention aux autres identificateurs (comme l'adresse IP) qui doivent changer en même temps que le HI (sinon, l'attaquant pourrait faire le rapprochement). L'Université Technologique d'Helsinski a déjà une mise en œuvre expérimentale de cette idée.

HIP n'est pas la seule architecture de séparation du localisateur et de l'identificateur. La section 8 du RFC résume les autres expériences en cours. Il ne va pas de soi que cette séparation soit une bonne idée. C'est même un point de vue très controversé, comme l'avait montré l'échec du NSRG ("*Name Space Research group*"). Parmi les efforts récents :

— Les groupes de travail multi6 `<http://tools.ietf.org/wg/multi6/>` et shim6 `<http://tools.ietf.org/wg/shim6/>` qui ont mené au RFC 5533, solution qui a des ressemblances avec HIP (d'ailleurs, OpenHIP implémente à la fois HIP et shim6), ce qui a permis de concevoir des API proches (RFC 6316 et RFC 6317).
— Le groupe de recherche RRG `<http://irtf.org/rrg>` a exploré de nombreuses solutions pour le routage de demain, comme LISP ou ILNP, documentant le résultat dans le RFC 6115.