Rapport de stage de première année de Magistère d'Informatique et Modélisation (MIM)

Etablissement de tests permettant de mesurer l'impact sur les performances de la sécurisation d'IPv6 par IPsec

Nicolas Bernard

22 août 2002

Remerciements:

Dans tout rapport de ce type, c'est quasiment un devoir de remercier un tas de monde. Dans le cas présent c'est aussi un plaisir... Je remercie donc d'abord, bien sûr, Isabelle Astic, qui m'a encadré durant ce stage et a su, tout en me laissant très libre, répondre à mes questions, mais aussi Olivier Festor et Eric Fleury qui ont, bien que n'apparaissant pas sur les documents officiels, dans l'ombre donc, co-encadré le susdit stage.
D'une manière plus générale, je remercie également l'équipe Resedas entière qui m'a accueilli chaleureusement avec une pensée particulière pour mes cobureaux, Yves Caniou et Mohammed Essaidi, et d'une manière encore plus générale, globale même, l'ensemble du personnel du Loria, les personnes que j'ai rencontrées s'étant toujours montrées aimables et accueillantes.

Table des matières

Introduction

L'Internet est en passe de devenir indispensable dans la vie courante. Inconnu du grand public il y a moins de dix ans, il a depuis acquis une popularité phénoménale et est maintenant intégré dans la société. Il est néanmoins encore en pleine évolution et de nombreux sujets le concernant sont toujours des thèmes de recherche.

La contrepartie de son succès est la découverte de certaines limitations dans les protocoles actuellement utilisés, tant du point de vue du nombre de machines connectables et des performances que du point de vue de la sécurité, ou plutôt de son absence. C'est pourquoi une nouvelle version du *Protocole Internet*, dites IPv6, a été conçue et est destinée, à terme, à remplacer l'ancienne[1]. Cette nouvelle version inclut notamment de puissants mécanismes de sécurité, IPsec, afin de pourvoir aux demandes de confidentialité des entreprises qui veulent relier des sites distants, comme des particuliers qui aimeraient pouvoir acheter via Internet de manière sécurisée.

Tout cela est encore en phase d'essai. Notre travail a justement consisté à établir des tests afin d'étudier dans quelle mesure l'utilisation de cette sécurité influe sur les performances des transferts. Ils sont destinés à être effectués sur le réseau de recherche très haut-débit français, VTHD++ (*Vraiment Très Haut Débit*, le "++" désignant son utilisation avec IPv6), dont le débit atteint 2,5 Gbit/s.

Notre stage a donc comporté deux phases, une phase d'établissement des tests et une seconde dans laquelle nous avons utilisé ceux-ci sur la plateforme IPv6 du Loria ce qui nous a permis d'avoir une idée des résultats qu'ils permettront d'obtenir.

1. Même si certains plaisantins en sont déjà à parler d'IPv9 [12]...

Chapitre 1

Présentation d'IPv6

On dit souvent que le protocole utilisé sur Internet est IP (*Internet Protocol*). Pour être précis, à l'heure actuelle, il s'agit de la quatrième version de ce protocole ou IPv4. Conçu à l'origine pour un réseau comprenant quelques dizaines de milliers de machines au maximum, ce protocole possède un certain nombre de limitations qui posent des problèmes face à la croissance exponentielle d'Internet. Après une prise de conscience de cet état de fait dans le début des années 1990, il a été décidé de créer un remplaçant pour ce protocole, qui fut appelé IPv6, le nom de "version 5" ayant déjà été attribué à un protocole expérimental.

La nouvelle version possède donc un certain nombre d'améliorations dont la plus visible est le changement de la taille des adresses qui passe de 32 bits dans IPv4 à 128 bits dans IPv6, ce qui permet selon les estimations les plus pessimistes d'attribuer plus de 1500 adresses par mètre carré pour toute la Terre! Un autre changement d'importance est une simplification des en-têtes des paquets, avec notamment la disparition de la somme de contrôle, afin de simplifier le travail des routeurs et leur permettre ainsi de gérer un trafic plus important [1].

Ajoutons qu'IPv6 offre encore un grand nombre de possibilités, par exemple au niveau de la qualité de service ou de la mobilité, dont la description sortirait – tant du point de vue du sujet que par la place requise pour le traiter – du cadre de cette présentation restreinte au minimum nécessaire pour comprendre la suite. Le lecteur intéressé pourra se reporter à l'ouvrage de référence sur le sujet – [3] – ou aux RFCs décrivant IPv6 qu'il pourra trouver en faisant une recherche avec le mot clef "IPv6" sur `http://www.rfc-editor.org`, la liste étant trop longue pour être donnée ici, et en constante évolution.

1. pour une introduction aux problèmes que peut poser le routage à très haut débit, on pourra consulter [1].

1.1 Adresses et adressage

Les adresses IPv6 font donc 128 bits de long. Les concepteurs du nouveau protocole se sont dit que représenter les nouvelles adresses de la même manière que celle d'IPv4 selon la notation pointée traditionnelle serait pénible, les adresses pouvant alors avoir jusqu'à 63 caractères. Ils ont donc proposé une nouvelle notation utilisant l'hexadécimal, ce qui est plus compact[2].

Les adresses sont donc notées par un découpage des 128 bits en huit mots de 16 bits représentés en hexadécimal et séparés par le caractère ':', par exemple:

`FEDC:789A:B456:9A27:EDBC:9276:1BD8:11E1`

Il faut ajouter à cela que dans un champ les zéros en en-tête ne sont pas obligatoires et que plusieurs champs nuls consécutifs sont abrégeables par '::' (qui ne doit figurer qu'une seule fois dans une adresse bien sûr pour éviter les ambiguïtés).

Les en-têtes sont notés avec la notation banalisée par CIDR, par exemple l'en-tête de notre exemple précédent pourrait être:

`FEDC:789A:B456:9A27::/64`

Terminons sur l'adressage en précisant qu'il y a trois types d'adresses avec IPv6: *unicast* ("normale"), *multicast* (plusieurs récepteurs) et *anycast* (n'importe quel récepteur parmi plusieurs), ce dernier type étant une des nouveautés de cette version[3].

1.2 En-têtes IPv6

1.2.1 Format des en-têtes

Le format d'en-tête des paquets IPv6 est décrit dans la rfc 2460 (voir la figure 1.1).

Le champ *version*, de 4 bits, contient la valeur 6. La présence de ce champ en début des paquets permet d'utiliser plusieurs versions d'IP, en l'occurrence les versions 4 et 6, simultanément sur un même réseau. Les champs *classe* et *identificateur de flux* servent pour gérer la qualité de service. Le champ *longueur des données* indique la taille de la charge utile du paquet (et non pas celle du paquet entier comme dans IPv4) en octets. *Prochain en-tête*

2. Cela dit comme le montre [4], il est possible d'avoir des adresses représentées de manière plus compactes encore, mais, allez savoir pourquoi, ces propositions n'ont pas rencontré beaucoup d'échos... ;-)

3. De telles adresses existaient déjà de manière optionnelles mais étaient confinée à des domaines de recherche. Intégrées en standard dans IPv6 pour un usage à grande échelle, elles sont encore sujettes à débats.

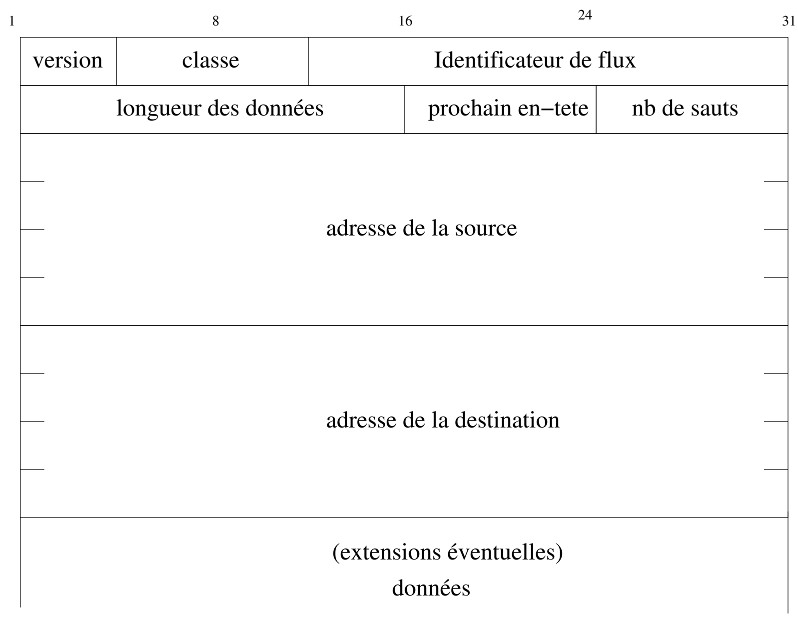

FIG. 1.1 – *Format d'un paquet IPv6*

indique le type de l'en-tête qui suit l'adresse de destination : ce peut être une extension (voir section suivante) ou un protocole de niveau supérieur (par exemple tcp). *nombre de sauts* remplace le champ *TTL* d'IPv4 : il est décrémenté à chaque routeur et le paquet est détruit s'il atteint 0. Les deux champs suivants contiennent respectivement les adresses de l'émetteur et du récepteur. Suivent les éventuelles extensions puis la charge utile.

1.2.2 Les extensions

Valeur	Protocole
6	TCP
17	UDP
41	IPv6
58	ICMPv6

Valeur	Extension
0	proche-en-proche
43	routage
44	fragmentation
50	confidentialité
51	authentification
59	fin des en-têtes
60	destination

TAB. 1.1 – *Les valeurs possibles du champ* en-tête suivant

Comme nous l'avons signalé précédemment, des extensions peuvent suivre l'en-tête IPv6 proprement dite. Elles permettent d'étendre IPv6 pour différents cas : fragmentation des paquets, routage (routage par la source), proche-en-proche (pour les *jumbogrammes* et le routage), et, ce qui va nous intéresser dans la suite, sécurité avec IPsec. Le tableau 1.1 liste les valeurs que peut prendre le champ en-tête suivant.

Chapitre 2

Cryptographie et IPsec

2.1 Présentation d'IPsec

Les protocoles cryptographiques utilisés sur les réseaux sont nombreux, citons par exemple *PGP* [1] pour le chiffrement des mails, *ssl* pour la consultation sécurisée des sites web, ou *ssh* pour celle des sessions en ligne de commande.

La différence entre ces exemples et IPsec est le niveau auquel se place ce dernier : alors que la plupart des autres protocoles se placent au niveau applicatif et sont donc spécifiques à une application donnée – le mail, le web, etc – IPsec se place au niveau réseau, plus bas donc dans la hiérarchie en couches OSI ou – plus adaptée ici – la hiérarchie TCP/IP, et permet donc de sécuriser *l'ensemble* des protocoles applicatifs supérieurs de manière transparente pour les programmes qui les utilisent!

2.1.1 Les attaques sur un réseau IP

Sur un réseau IP, il existe principalement trois types d'attaques [2], qui sont, en simplifiant et dénaturant la réalité :

– l'*IP sniffing* qui consiste à "écouter" les paquets passant sur le réseau.

1. Au-delà du logiciel éponyme bien connu créé par Philip Zimmerman, *PGP* (*Pretty Good Privacy*) est en effet un standard décrit dans plusieurs RFCs et implémenté par plusieurs autres logiciels dont le plus connu est sans doute *GPG* (*GNU Privacy Guard*).

2. Pour une introduction non technique et pleine d'humour mais néanmoins sérieuse à la sécurité informatique, le lecteur pourra se reporter à [15]. Pour une introduction plus technique aux attaques et leurs parades mais orientée Linux, un ouvrage intéressant est [6]. Précisons enfin que dans ce domaine, pour rester à jour et connaître les dernières évolutions, la consultation régulière des sites spécialisés et de certains newsgroups de l'Internet sont indispensables.

Cette attaque permet par exemple de lire des mails ou, surtout, d'intercepter les mots de passe d'utilisateurs utilisant des protocoles non sécurisés (ftp, telnet,...) qui permettront à l'intercepteur de pénétrer par la suite dans le système visé;
– l'*IP spoofing* consiste à se faire passer pour une autre machine, généralement dans le but d'utiliser une relation de confiance existant entre deux machines pour s'introduire dans un réseau;
– et l'*IP flooding* qui est une attaque consistant, comme son nom l'indique (*to flood* : inonder) à submerger une machine de paquets en trop grand nombre pour que celle-ci puisse les traiter, ce qui l'empêche de répondre à des requêtes d'utilisateurs légitimes.

2.1.2 La sécurisation par IPsec

IPsec protège contre les deux premières de ces attaques : la parade utilisée contre l'IP sniffing est la confidentialité : en effet, l'espion ne peut plus alors que lire des paquets chiffrés, ce qui ne lui est guère utile, à moins qu'il ne possède de quoi les déchiffrer. Contre l'IP spoofing, on utilisera principalement des mécanismes d'authentification, d'intégrité, et de détection de rejeu, ainsi que, éventuellement, la confidentialité.

Il faut noter que des attaques sont également possibles aux couches inférieures du réseau, mais elles dépendent alors généralement du type de réseau et ne seront donc pas les mêmes selon que vous utilisez *ethernet* ou *token-ring*, généralement basées sur la falsification de trames de la couche liaison. Néanmoins, une utilisation intelligente de l'authentification fournie par IPsec permet généralement de détecter de telles attaques.

Pour ce faire, IPsec définit deux extensions (voir chapitre précédent) : AH (*Authentification header*) qui fournit l'*authentification* (la certitude que ce que l'on reçoit a bien été envoyé par la personne que l'on croit) et l'*intégrité* (la certitude que les données n'ont pas été modifiées en cours de route). AH peut également fournir, en option, la *détection de rejeu* (la certitude qu'un paquet vient réellement se placer ici dans la communication et n'est pas un ancien paquet enregistré et réémis par un adversaire). Il fournit également avec certaines méthodes d'authentification la *non répudiation* (la possibilité de prouver à un tiers de qui provient un paquet). L'autre extension, ESP (*Encapsulating Security Payload*) fournit la *confidentialité* (la certitude qu'une personne qui peut intercepter le paquet ne peut lire les données qu'il contient) en plus des mêmes services que AH.

La définition d'AH peut donc sembler redondante. Elle existe afin de permettre d'utiliser l'authentification dans des pays où la confidentialité est malheureusement encore illégale.

IPsec peut être utilisé dans deux modes différents : les modes transport et tunnel.

le mode transport En mode transport, IPsec protège principalement la charge utile (*payload*) d'un paquet. Il n'est utilisable dans une communication qu'entre deux équipements terminaux, le chemin emprunté par les paquets pouvant varier.

le mode tunnel Le mode tunnel protège tout le paquet en l'encapsulant dans un autre : le paquet que l'on veut transmettre constitue la charge utile d'un autre paquet. Ce mode permet de protéger plusieurs machines qui utilisent une même connexion réseau.

Association de Sécurité (SA)

Les paramètres d'une communication sécurisée par IPsec (extension de sécurité utilisée, algorithmes et clefs, durée de vie, mode (transport / tunnel / indifférent)) sont stockés dans une *association de sécurité* (SA). Ces associations sont identifiées de façon unique par un triplet (SPI, destinataire, protocole (AH/ESP)) où *SPI* signifie *Security Parameters Index*, *destinataire* est l'adresse du destinataire d'un paquet IP et *protocole* indique que l'on utilise soit AH soit ESP [3].

Ces associations sont unidirectionnelles, par conséquent pour protéger une communication dans les deux sens, il faudra (au moins) deux SAs.

Echange de clef – IKE

Vous devez maintenant vous dire : "Tour cela est bien beau, mais comment fait-on pour se mettre d'accord sur la clef à utiliser?". Un protocole appelé IKE a été créé pour cela : il se déroule en deux phases.

Dans la phase 1, IKE négocie avec la machine d'en face les algorithmes qui seront utilisés (algorithmes de chiffrement, fonctions de hachage, paramètres pour l'échange de clef Diffie-Hellman) puis génère trois clefs (avec Diffie-Hellman) qui serviront, l'une pour chiffrer des messages, la seconde pour les signer, et enfin la dernière pour générer d'autres clefs.

Dans la phase 2, le protocole utilise les secrets créés dans la phase 1 pour négocier les SAs IPsec qui seront utilisées par la suite.

Nous n'entrerons pas ici plus avant dans les détails d'IKE, qui pourraient à eux seuls occuper quelques dizaines de pages. Le lecteur pourra se reporter aux RFCs correspondantes et/ou à la fin du chapitre sur la sécurité de [3].

L'implémentation d'IKE de FreeBSD s'appelle *Racoon*.

3. Notez que si vous voulez utiliser AH *et* ESP, il vous faudra deux SAs.

SPD et SAD

IETF préconise d'utiliser deux bases de données pour gérer les politiques de sécurité, ce que fait l'implémentation intégrée à FreeBSD : l'une la SAD (*Security Association Database*) sert à entreposer les SAs, tandis que l'autre, la SPD (*Security Policy Database*), précise ce qui doit être fait pour un paquet. Si cette dernière base indique que la sécurité est nécessaire pour un paquet, on regarde dans la SAD pour retrouver la SA correspondante.

2.1.3 Limites d'IPsec

IPsec est un grand pas en avant vers un Internet plus sûr. Il n'est néanmoins pas la panacée absolue car il reste des attaques contre lesquelles son utilisation ne change pas grand chose :

- l'analyse de flux est encore très peu étudiée par la recherche universitaire. Il semble néanmoins que la simple analyse du flux de la communication (taille et fréquence des paquets,...), même chiffrée, suffise dans bien des cas (consultation d'un site web par exemple) pour en extrapoler de précieux renseignements. IPsec fournit une protection très modérée à ce type d'attaque (ajout de *padding*, ...) ;
- l'IP flooding, combiné à l'IP spoofing : si un ordinateur envoie énormément de paquets vers un autre en se faisant passer pour un ordinateur de confiance, la machine cible comprendra la tromperie car l'authentification échouera, mais ce processus est lourd et peut ainsi faciliter une attaque de refus de service (DOS).

2.2 *MACs* et algorithmes de chiffrement par blocs

Pour assurer l'authentification des paquets telle que définie dans AH ou ESP, il y a principalement deux moyens: utiliser un système de clef publique / clef privée ou utiliser ce que l'on appelle un MAC (*Message Authentification Code*). L'intérêt du premier système est qu'en plus de fournir l'authentification mutuelle pour les deux personnes qui communiquent, il fournit la *non répudiation*, c'est-à-dire permet de prouver à un tiers que tel paquet a été émis par l'un des deux en sachant qui des deux correspondants l'a émis. Ce n'est pas le cas des MACs, avec lesquels on ne peut déterminer lequel des deux correspondants a envoyé le paquet. Si ce système utilisant la cryptographie à clef publique semble donc supérieur, il est beaucoup plus gourmand en calculs[4], l'utilisation des MACs étant donc privilégiée. Dans l'implémentation du Kame fournie avec FreeBSD, tous les algorithmes d'authentification utilisent des MACs.

Comment fonctionnent donc ces MACs?

Ils fonctionnent à partir de *fonctions de hachage*, c'est-à-dire de fonction qui à partir d'un ensemble de données de taille quelconque fournissent une *empreinte* ou *hachis*. Les fonctions utilisées en cryptographie doivent en plus être telles qu'il soit "impossible" de générer à partir d'un ensemble de données fixé un autre ensemble de données ayant la même empreinte. [14] indique un critère heuristique : le changement d'un bit dans les données devrait provoquer le changement de la moitié des bits de l'empreinte.

La fonction de hachage étant fixée, tout le monde peut l'utiliser. Le fait de joindre dans le paquet aux données leur empreinte ne sert à rien, un assaillant pouvant faire de même. Par contre, si les deux personnes (Alice et Bob selon les dénominations "standard") partagent une clef, le fait de générer l'empreinte des données concaténées à la clef et de mettre dans le paquet l'empreinte avec les données permet de faire l'authentification : le récipiendaire connaissant la clef, il peut aisément recalculer l'empreinte de la même manière que l'émetteur et la comparer avec celle jointe au paquet.

Pour de complexes raisons mathématiques, la simple concaténation de la clef aux données pour faire l'empreinte n'est pas totalement sûre. L'idée est néanmoins là et les MACs réellement utilisés en dérivent. L'un des plus utilisé est HMAC défini comme:

$$HMAC(K,texte)_t = \mathcal{H}((K_0 \oplus opad)||\mathcal{H}((K_0 \oplus ipad)||text))$$

où K est la clef symétrique, K_0 la clef complétée avec des 0 jusqu'à la taille d'un bloc de données utilisé par la fonction de hachage ou l'empreinte de K si au contraire celle-ci est trop longue, $\mathcal{H}$ est la fonction de hachage, t le nombre de bits composant le MAC, *ipad* est un bloc rempli avec des octets de valeur 0x36, *opad* étant lui rempli avec des 0x5c. $||$ est la concaténation et $\oplus$ le ou exclusif.

Les fonctions de hachage présentes dans la souche IPsec du projet Kame (**http://www.kame.net**) qui est celle intégrée à FreeBSD et à laquelle nos tests sont destinés sont *md5* (*Message Digest 5*), *sha-1* (*Secure Hash Algorithm 1*) et *sha-2*. Les empreintes ont respectivement une longueur de 128 et 160 bits pour les premiers, et peuvent être de 256, 384 ou 512 bits pour le dernier. *MD5* est d'origine industrielle (RSA data security) tandis que les sha sont des standards du gouvernement américain.

4. On dit souvent que les algorithmes à clef publique sont mille fois plus lents que ceux à clef privée...

Notons pour information qu'une fonction de hachage de plus en plus répandue dans les cryptosystèmes est RIPE-MD, qui est le standard européen. On peut donc s'attendre à ce que cette fonction soit rajoutée dans une prochaine version.

Algorithmes de chiffrement

Nous n'entrerons pas ici dans le détail des algorithmes de chiffrement, la conception pouvant varier énormément d'un algorithme à l'autre. Notons simplement qu'il s'agit ici d'algorithmes symétriques, c'est-à-dire utilisant une même clef pour le chiffrement et le déchiffrement qui doit bien entendu rester secrète. Une taille de clef de 128 bits (en supposant que l'attaque exhaustive – où l'on essaye toutes les clefs une par une – soit la meilleure) est considérée comme offrant une bonne confidentialité. Naturellement, pour que l'hypothèse soit correcte, il faut que l'algorithme soit valide et on peut à tout moment trouver une nouvelle attaque...

Les algorithmes intégrés dans la souche IPsec de FreeBSD sont *DES*, *Triple-DES*, *Blowfish*, *CAST* et *AES*. Notons que l'*AES* (*Advanced Encryption Standard*) est destiné à remplacer le *DES*, maintenant peu sûr. Il n'a été choisi que récemment par les autorités américaines comme nouveau standard, aussi le trouve-t-on encore parfois sous son nom d'origine – *Rijndael*. Pour cette raison, on constate en examinant le code source de FreeBSD qu'il est une pièce rapportée eu égard aux autres algorithmes.

Chapitre 3

Définition des tests

Notre but est d'observer et autant que possible de quantifier la dégradation des performances due à la sécurisation des communications sur un réseau IPv6 par l'utilisation d'IPsec.

Les procédures d'installation parfois nécessaires – que nous devions également préparer – ont été reportées en annexe.

Les performances (c'est-à-dire le débit) peuvent dépendre de nombreux facteurs ; citons, en vrac, pour ce qui est des ordinateurs qui communiquent:

- l'ordinateur lui-même (CPU [1], Ram);
- plus spécifiquement la carte réseau qui l'équipe et le type de réseau;
- l'implémentation de la pile IPv6 et d'IPsec dans le système d'exploitation.

Dans le cas de l'utilisation d'IPsec en mode tunnel, les mêmes critères s'appliquent aux passerelles de sécurité qui gèrent le tunnel.
Le débit *stricto sensu* dépend également du réseau, c'est-à-dire d'une part des ordinateurs intermédiaires par lequels transitent les paquets et d'autre part des liens entre eux. Néanmoins, leurs performances ne varient normalement pas selon que l'on utilise ou non IPsec.

Les performances peuvent également varier selon la manière dont on utilise IPsec:

- utilisation ou non de l'authentification:
  - authentification avec AH ou avec ESP;
  - algorithme utilisé;
- utilisation ou non du chiffrement:
  - algorithme utilisé;

1. L'architecture du CPU peut jouer un rôle important selon que l'on a un RISC ou un CISC avec certains algorithmes comme RC4.

– longueur de clefs utilisée;
– temps de vie des clefs;

Elle peuvent aussi dépendre de paramètres du réseau, comme par exemple la taille des buffers associés aux sockets.

Ces différents facteurs pouvant se mêler, nous avons donc de nombreux cas possibles. Dans le cas de l'utilisation du mode tunnel, ce problème est reporté sur les passerelles de sécurité. Il faut alors en plus ajouter les cas où l'on utilise ou non IPsec entre l'émetteur/le récepteur et la passerelle correspondante.

Pour visualiser instantanément l'effet de l'ensemble de ces paramètres, il faudrait représenter les résultats des tests par un graphe à [nombre de paramètres + 1 (le débit)] [2] dimensions. Comme nous ne pouvons voir ce type de graphe, les résultats seront visualisés par des coupes à deux dimensions d'un tel graphe, mais surtout par des tableaux présentant les principaux résultats.

Pour ce qui est des premiers facteurs, pour s'en abstraire autant que possible, il faut impérativement faire les différents tests avec les mêmes machines. Il serait intéressant d'exécuter l'ensemble des tests sur des machines possédant des configurations variées de manière à voir l'importance de ce facteur.

3.1 Les programmes de tests

Les tests sont constitués de mesures du débit par les programmes *ttcp* et *netperf* [3]. Ces programmes ne gèrent pas IPsec eux-mêmes, il nous faut donc établir auparavant la politique de sécurité.

3.1.1 Netperf

Le programme *Netperf* est développé depuis 1993 par la division réseau de Hewlett Packard. Ce programme est principalement destiné au système HP-UX, mais il est relativement portable – du moins pour ses fonctionnalités essentielles – sur toutes les plateformes Unix. Ce programme disposait déjà de tests IPv6 (non officiels, la documentation n'en dit pas un mot) mais étant

2. Les résultats sont des hypersurfaces à [nombre de paramètres] dimensions (une hypersurface par outil de test) dans un espace comprenant une dimension supplémentaire puisque à chaque fois c'est le débit que nous mesurons.

3. Sauf indication contraire, les deux programmes sont utilisés successivement pour chaque test.

donné le nombre de corrections qu'il a été nécessaire de faire dans ce code nous doutons qu'il ait jamais été testé, ou alors avec un compilateur C pour le moins spécial! Nous avons par la même occasion mis à jour le code qui utilisait d'anciennes fonctions datant des débuts d'IPv6 (*gethostbyname2*) en les remplaçant par les fonctions du standard actuel (*getaddrinfo*).

Utilisation: Netperf se divise en deux programmes: un serveur et un client. Il faut donc lancer le serveur sur l'un des ordinateurs par la commande

`/opt/netperf/netserver -p 12865`

(en supposant Netperf installé dans /opt/netperf/). 12865 est le numéro de port par défaut de Netperf. C'est sur celui-là que cherche le client si on ne lui précise rien. Il est nécessaire de le préciser au serveur.

Nous pouvons ensuite lancer sur l'autre machine le client. La syntaxe est

`/opt/netperf/netperf -t type_test -H serveur`

où **type_test** est le type de test que l'on veut faire et **serveur** le nom de machine du serveur. Les types de tests disponibles en IPv6 sont *TCPIPV6_STREAM*, *TCPIPV6_RR*, *TCPIPV6_CRR*, *UDPIPV6_STREAM* et *UDPIPV6_RR*. Les tests que nous utilisons sont principalement ceux concernant les flux (stream). Les "RR" sont des tests de réponses aux requêtes (Request/Response). Notez que **serveur** doit être le nom de la machine et ne peut-être une adresse IP, la socket de contrôle de Netperf se faisant en IPv4. Le nom doit de plus être résolu aussi bien avec IPV4 qu'IPv6[4].

De nombreuses options sont disponibles pour Netperf, la plupart nécessitant des paramètres de compilation spéciaux. Pour plus de détails, consulter [7] (les paramètres IPv6 sont les mêmes que ceux indiqués pour *TCP_STREAM*).

3.1.2 ttcp

Les origines du programme *ttcp* (pour *Test TCP*) sont plus floues que celles du précédent: son auteur n'est pas connu. Le programme semble dater d'avant 1984, date de la première modification indiquée. Il a été par la suite modifié par des employés de *Silicon Graphics*. La version que nous utilisons, *ttcp6*, est une version modifiée au CERN pour utiliser IPv6. Nous avons, comme pour Netperf, mis à jour le programme pour pouvoir le compiler en remplaçant les fonctions et structures de données devenues obsolètes par celles utilisées dans le standard actuel.

Utilisation: Il faut lancer le programme sur le serveur et sur le client. La distinction entre le client et le serveur se fait en passant l'option **-r** à ce dernier. Nous lançons donc

4. Eventuellement, il suffit de rajouter deux entrées dans le fichier */etc/hosts*.

```
/opt/ttcp6 -r
```
sur le serveur puis
```
/opt/ttcp6 -t adresse_serveur
```
sur le client.

Routage sur le VTHD

Les ordinateurs qui serviront aux tests sur le VTHD étant également reliés entre eux par Renater, avec IPv6 dans les deux cas, nous avons ajouté une fonction qui vérifie que l'on utilise bien l'adresse VTHD et non l'adresse Renater en vérifiant le préfixe desdites adresses qui doit être `2001:688:1FB8::/48` pour le VTHD.

Afin que l'on puisse toujours utiliser nos versions de ces programmes ailleurs que sur le VTHD, l'utilisation de cette fonction ne se fait que si l'on a compilé les programmes avec l'option **-DVERIF_VTHD** passée au compilateur.

Il faut noter que les sockets de contrôle de Netperf et de nos scripts (voir plus loin) utilisent IPv4 et ne sont pas soumises à cette vérification.

3.2 Tests témoins

Tout d'abord, il nous faut une référence, c'est-à-dire une mesure du débit d'une machine lorsque l'on n'utilise pas IPsec. Pour cela, avant de lancer une série de tests, utilisant IPsec, nous faisons des mesures de débit avec Netperf et ttcp6 avec les mêmes paramètres réseaux (buffers des sockets, ...) mais sans IPsec. Eventuellement, si la série de tests est longue, il peut être intéressant de refaire un tel test à périodes régulières afin de vérifier que le débit n'a pas changé à cause, par exemple, d'un autre trafic.

Le test décrit précédemment nous donne une mesure du débit que peut émettre la machine sur le réseau. Néanmoins, il est également intéressant de voir le débit que peut émettre la machine quand elle n'est pas limitée par la bande passante du réseau.

3.3 Tests en mode transport

3.3.1 Authentification

Avec AH

Il faut donc paramétrer les deux ordinateurs pour utiliser l'authentification avec AH. Pour cela, il faut utiliser (en tant que *root*) l'utilitaire *setkey*

Il faut d'abord ajouter une entrée dans la SAD[5] :

```
setkey -c
add adresse_source adresse_destination ah spi -m transport -A algorithme clef;
```

où **adresse_source** et **adresse_destination** sont les adresses IPv6 des hôtes, **spi** un entier strictement supérieur à 255, **algorithme** appartenant à la liste des algorithmes utilisables, et **clef** une chaîne de caractères de longueur correspondant à celle requise par l'algorithme choisi.

Ensuite, pour que cette entrée soit utilisée, il faut encore ajouter une entrée dans la *SPD*:
cela se fait avec

```
setkey -c
spdadd adresse_source adresse_destination any -P out ipsec
ah/transport/adress_source-adresse_destination/require;
```

sur l'émetteur, et

```
setkey -c
spdadd adresse_source adresse_destination any -P in ipsec
ah/transport/adress_source-adresse_destination/require;
```

sur le récepteur.

Par exemple, pour configurer l'authentification avec AH et l'algorithme hmac-md5 en utilisant la clef *TESTtestTESTtest* entre *sha* (2001:660:301:35:2c0:4fff:febb:af7a) et *thorgal* (2001:660:301:35:2c0:4fff:fe67:6b32) :
sur les deux machines:

```
setkey -c
add 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 ah 576
       -m transport -A hmac-md5 "TESTtestTESTtest";
```

Sur *sha*:

```
setkey -c
spdadd 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 any -P out ipsec
ah/transport/2001:660:301:35:2c0:4fff:febb:af7a-2001:660:301:35:2c0:4fff:fe67:6b32/require;
```

et sur *thorgal*:

```
setkey -c
spdadd 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 any -P int ipsec
ah/transport/2001:660:301:35:2c0:4fff:febb:af7a-2001:660:301:35:2c0:4fff:fe67:6b32/require;
```

5. La SAD et la SPD sont les tables utilisées pour gérer les Associations de Sécurité (SA) (voir le paragraphe à ce sujet dans la section 2.1.2).

NB : Cela n'établit l'authentification que dans un sens (de *sha* vers *thorgal*). Pour l'établir également dans l'autre sens, il faut effectuer la procédure symétrique (avec un *spi* différent).

Algorithmes utilisables: dans l'implémentation actuelle de FreeBSD, les algorithmes utilisables sont: hmac-md5, hmac-sha1, keyed-md5, keyed-sha1, hmac-sha2-256, hmac-sha2-384, et hmac-sha2-512. Vous noterez que la série hmac-sha2-* utilise toujours le même algorithme mais avec des longueurs de clef différentes.

Algorithme	taille de la clef (bits)
hmac-md5	128
hmac-sha1	160
keyed-md5	128
keyed-sha1	160
hmac-sha2-256	256
hmac-sha2-384	384
hmac-sha2-512	512

A ce moment vous pouvez vous demander : “Comment être sûr que AH est réellement utilisé?”. Il y a plusieurs méthodes: la plus simple consiste à faire un test de performances du réseau : à moins que vous n'ayez une machine très puissante ou un réseau bas débit (une liaison modem ou un réseau *ethernet* par exemple), vous devez observer une dégradation de performance par rapport au test témoin (cf. nos résultats). La seconde méthode est plus fiable, dans la mesure où l'on peut prouver que la liaison transmet des paquets possédant un en-tête AH: Pour cela on lance
`tcpdump -x ip6`[6]
sur le récepteur, et on utilise l'utilitaire **ping6** pour envoyer des paquets depuis l'émetteur.

On peut alors recevoir un paquet du type:

```
09:31:58.055251 sha.rlab.loria.fr > thorgal.rlab.loria.fr: AH(spi=0x00000240,seq=0x367): blackjack > commplex-link:
                        6000 0000 05b0 33 40 2001 0660 0301 0035
                        02c0 4fff febb af7a 2001 0660 0301 0035
                        02c0 4fff fe67 6b32 0604 0000 0000 0240
                        0000 0367 902b 873f c7ea 9515 5427 ffdb
```

6. dans un réseau à fort trafic, il peut être utile de filtrer les paquets affichés selon leur émetteur ou leur récepteur. Cela peut se faire en ajoutant **src émetteur** ou **dst récepteur** sur la ligne de commande. Des combinaisons peuvent être faites en utilisant les opérateurs **and** et **or**. Voir la page man de tcpdump pour plus de détails.

```
0401 1389 5da7 7f9c 12f2 f09d 8010 82b8
2576
```

On remarque que le septième octet, qui indique le type du premier en-tête d'extension, indique 33h, c'est-à-dire 51d, qui est la marque d'AH.

L'authentification avec ESP n'est utilisable que si l'on utilise également ESP pour la confidentialité. Une manière détournée de l'utiliser uniquement pour l'authentification est d'utiliser l'algorithme de chiffrement NULL [5] (appelé **simple** dans la terminologie de setkey) qui est en fait la fonction identité. Les tests correspondants sont donc traités dans la section "Authentification et chiffrement".

3.3.2 Chiffrement

Le paramètrage d'IPsec pour utiliser la confidentialité avec ESP est très proche de celui que l'on avait pour l'authentification. Il faut ajouter des entrées dans la SAD et dans la SPD:

Sur les deux machines:

```
setkey -c
add adresse_source adresse_destination esp spi -m transport -E algorithme clef ;
```

Sur l'émetteur:

```
setkey -c
spdadd adresse_source adresse_destination any -P out ipsec
esp/transport/adresse_source-adresse_destination/require;
```

ou, sur le destinataire:

```
setkey -c
spdadd adresse_source adresse_destination any -P in ipsec
esp/transport/adresse_source-adresse_destination/require;
```

Par exemple, toujours de *sha* vers *thorgal*, pour utiliser ESP avec l'algorithme **des-cbc** et la clef **PASSWORD**:

Sur *sha* ET *thorgal*

```
setkey -c
add 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 esp 3000
          -m transport -E des-cbc "PASSWORD" ;
```

puis, sur *sha*

```
setkey -c
spdadd 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 any -P out ipsec
esp/transport/2001:660:301:35:2c0:4fff:febb:af7a-2001:660:301:35:2c0:4fff:fe67:6b32/require;
```

et sur *thorgal*

```
setkey -c
```

```
spdadd 2001:660:301:35:2c0:4fff:febb:af7a 2001:660:301:35:2c0:4fff:fe67:6b32 any -P out ipsec
esp/transport/2001:660:301:35:2c0:4fff:febb:af7a-2001:660:301:35:2c0:4fff:fe67:6b32/require;
```

***NB* :** De la même manière que pour l'authentification, cela n'établit la confidentialité que dans un sens, une procédure duale étant nécessaire pour l'établir dans les deux sens.

algorithmes utilisables: les algorithmes de chiffrements utilisables avec ESP sont, dans cette implémentation, des-cbc, 3des-cbc, simple, blowfish-cbc, cast128-cbc, des-deriv et rijndael-cbc[7].

Algorithme	taille(s) de la clef (bits)	taille(s) réelle(s) (bits)
des-cbc	64	56
3des-cbc	192	112
simple	0 à 2048	0
blowfish-cbc	40 à 448	40 à 448
cast128-cbc	128	128
des-deriv	64	56
rijndael-cbc	128/192/256	128/192/256

3.3.3 Authentification et chiffrement

Les choses se compliquent très légèrement pour utiliser conjointement l'authentification et la confidentialité et il y a quelques variantes selon que l'authentification est considérée comme une option d'ESP ou utilise AH.

En effet, si l'on utilise AH, il faut mettre deux SA dans la SAD…

```
setkey -c
add adresse_source adresse_destination esp 1000 -E algorithme_de_confidentialité Clef_de_confidentialité;
add adresse_source adresse_destination ah  2000 -A algorithme_d'authentification Clef_d'authentification;
```

…avant d'ajouter une entrée dans la SPD:

```
spdadd adresse_source adresse_destination any -P out ipsec
            esp/transport/adresse_source-adresse_destination/require
            ah/transport/adresse_source-adresse_destination/require;
```

7. les lettres "cbc" indiquent que ces algorithmes sont utilisés en mode chaînage de blocs (*cipher block chaining*)

Par contre, si l'on utilise ESP uniquement, une seule entrée de la SAD regroupe les deux fonctions:

```
setkey -c
add adresse_source adresse_destination esp 1000 -E Algorithme_de_confidentialité Clef_de_confidentialité
                                               -A algorithme_d'authentification Clef_d'authentification;
```

et l'entrée dans la SPD est ajoutée par:

```
spdadd adresse_source adresse_destination any -P out ipsec
             esp/transport/adresse_source-adresse_destination/require
```

Effacer des entrées dans la SAD ou la SPD
Il y a plusieurs moyens pour effacer une ou plusieurs entrées dans l'une des tables en cas de besoin. La méthode la plus simple consiste à effacer la totalité de la table avec **setkey -F** pour la SAD ou **setkey -FP** pour la SPD.
Il est également possible de n'effacer qu'une entrée si nécessaire : la syntaxe est alors proche de celle utilisée pour en ajouter une : pour la SAD:

```
setkey -c
delete adresse_source adresse_destination ext spi
```

où **ext** vaut soit **ah** soit **esp**, tandis que pour la SPD la syntaxe est absolument similaire à celle utilisée pour l'ajout, en remplaçant toutefois **spdadd** par **spddelete**.

3.4 Tests en mode tunnel

En mode tunnel, les choses sont très proches de ce qu'elles sont en mode transport:

- il faut partout remplacer "transport" par "tunnel" (Comment? Vous aviez deviné? ;-));
- les adresses source et destination sont celles des passerelles de sécurité sauf à un endroit (voir ci-dessous)

C'est dans la SPD que l'on définit quelles transmissions doivent être chiffrées : la syntaxe (pour un émetteur ESP) est la suivante:

```
setkey -c
spdadd adresse_réseau_source adresse_réseau_destination any -P out ipsec
         esp/tunnel/adresse_source-adresse_destination/require;
```

Les **adresse_réseaux_*** utilisent la notation IPv6 classique pour les réseaux, c'est-à-dire la syntaxe *préfixe/longueur_du_préfixe*.

Il faut noter qu'AH semble ne pas fonctionner en mode tunnel dans la version que nous avons utilisée de l'implémentation. Il semble également y avoir un problème si l'on utilise ESP avec l'AES (*"rijndael-cbc"*).

Pour vérifier le bon fonctionnement du tunnel, la méthode utilisée est la même qu'en mode transport: en écoutant le trafic entre les deux passerelles, on doit voir passer de l'une vers l'autre des paquets utilisant ESP si l'on fait un *ping6* de l'émetteur vers le récepteur.

3.5 Tests avec des émetteurs ou des récepteurs multiples

NetPerf permet d'avoir plusieurs connexions simultanées. Il est donc possible d'avoir un serveur A sur lequel tourne le serveur de Netperf et de faire le test simultanément avec des clients B et C afin de voir si la somme des débits mesurés est supérieure, égale ou inférieure au cas où l'on a un seul client[8]. On peut bien sûr étendre à n clients...

Pour avoir des mesures aussi précises que possible, il faudrait que les clients s'exécutent simultanément. Comme il y a toujours un léger décalage au départ et à la fin du test, il est conseillé d'augmenter la durée de celui-ci afin de limiter les effets du susdit décalage[9]. Avec Netperf, il faut utiliser l'option **-l temps** où **temps** est la durée du test que l'on désire en secondes.

Il est également possible de faire ce type de test avec *ttcp6*, mais cela est plus complexe et les risques de résultats erronés sont grands : comme ttcp6 n'accepte, en mode récepteur, qu'une connexion, il faut en lancer plusieurs instances en spécifiant manuellement le numéro de port que l'on veut. Cela se fait avec l'option **-p port** où **port** est bien sûr le numéro du port que l'on veut utiliser. Le problème qui risque d'influer sur les résultats de manière difficilement quantifiable réside dans le fait que ttcp6 ne permet pas d'avoir un test d'une durée fixée. Le test consiste en effet à envoyer une certaine quantité de données et à voir combien de temps cela prend. Si l'un de nos clients peut donc envoyer ses données plus vite (par exemple parce qu'il est plus puissant, ce qui compte énormément avec IPsec comme le montrent nos

8. Il est conseillé que les clients soient reliés au serveur par des réseaux différents afin d'éviter une baisse des performances due aux collisions.

9. On risque une légère surestimation du débit

résultats) il aura terminé le test plus rapidement et alors l'autre client se retrouvera seul pour la fin de son test ce qui donnera une réponse fausse, le serveur étant beaucoup plus disponible...

Il est bien sûr possible de faire les tests symétriques où l'on a deux serveurs et où l'on lance un test vers chacun des serveurs simultanément sur un unique client [10]. Les mêmes réserves s'appliquent quant à l'emploi de ttcp6 dans ce test.

3.6 Tests avec IKE

Il peut être intéressant de tester l'impact sur les performances de l'échange de clef via IKE et son implémentation dans FreeBSD, Racoon. En effet, bien que le trafic généré ne soit que de quelques paquets, il faut voir si, par exemple, lorqu'une clef expire, le trafic continue avec celle-ci jusqu'à ce que la nouvelle soit prète ou s'il est interrompu jusqu'à ce moment. D'autre part, les protocoles de cryptographie à clef publique sont connus pour leur gourmandise en puissance de calcul. Il faudrait voir si cela influe sur les performances.

Pour faire ces tests, l'idée est de configurer Racoon avec le type de sécurité que l'on veut en choisissant une certaine durée de vie – par exemple 2 minutes – puis de lancer le test en spécifiant une durée supérieure (voir section précédente) à la durée de vie de la clef, 3 minutes dans notre exemple.

Il nous faut donc dans l'ordre:

- modifier le fichier *racoon.conf* selon nos désirs;
- lancer Racoon en indiquant notre nouveau fichier de configuration (option **-f filename**);
- afin d'initialiser la première clef, faire un *ping6* jusqu'à obtenir une réponse;
- faire le test proprement dit.

Il peut être également intéressant de voir le temps que met Racoon pour l'échange de la phase 1 et ce avec les différents modes d'échange de clef dans la phase 1; les tests décrits précédemment ne testant que la phase 2, cela pose néanmoins des problèmes de mise en oeuvre.

3.7 Tests ésotériques

On peut ensuite encore faire des tests que nous qualifierons d'ésotériques car *a priori* ce sont des configurations qui ne seront que très rarement utili-

10. Même remarque que précédemment.

sées.

Ainsi par exemple, nous pouvons imaginer de faire un chiffrement à la fois en mode tunnel et en mode transport. Il faut noter que cette utilisation simultanée peut avoir des effets contraires à ceux recherchés en diminuant le niveau de sécurité : imaginez que la même clef soit utilisée pour les deux modes, les algorithmes étant symétriques [11]... Il faut encore ajouter que l'utilisation du même algorithme même avec une clef différente, si elle ne diminue pas, *a priori*, la sécurité du cryptogramme, ne l'augmente pas forcément non plus: il est tout à fait possible que l'algorithme en question forme un groupe et que donc il existe une troisième clef qui déchiffre directement le message depuis le cryptogramme surchiffré avec les deux clefs des modes transport et tunnel [12]. Ainsi, l'algorithme NULL est à l'évidence un groupe ;-)

3.8 Comment faire tous ces tests?

Cela fait beaucoup de tests à effectuer, ce qui augmente d'autant le risque d'erreurs. Par souci de commodité, nous avons créé quelques scripts Perl afin d'automatiser ces tâches. Ces scripts se lancent sur les différentes machines devant être testées. Ils utilisent des mécanismes basiques de synchronisation et de reprise sur erreur pour le lancement des tests.

Le choix de Perl est dû à la nécessité d'utiliser des sockets pour la synchronisation, ce que ne permet pas de manière simple un *shell-script*, tout en gardant une simplicité d'écriture que ne permet pas le *C*. Perl s'est imposé par rapport à d'autre langages similaires comme *Python* par la présence d'un compilateur pour ce langage dans FreeBSD, ce qui évite d'avoir à en installer un nouveau.

Etant donné que Perl ne supporte pas encore en standard IPv6 et le problème de la socket de contrôle de Netperf, il faut malheureusement leur passer en paramètres à la fois les adresses IPv4 et IPv6 ainsi que le nom d'hôte de chaque machine, ce qui est relativement lourd. Leurs codes-sources se trouvent en annexe.

11. Il semble cependant difficile que les données passent en clair sur le réseau: il faudrait que l'alignement des blocs corresponde et même dans ce cas du fait de l'utilisation du bloc précédent (nous sommes en mode cbc, ne l'oublions pas) pour chiffrer le bloc courant, on ne doit pas pouvoir retomber sur le texte en clair. Il est néanmoins probable que la sécurité en soit beaucoup diminuée.

12. Il a été prouvé que le DES n'est pas un groupe.

transport.pl Comme son nom l'indique, ce script fait des tests en mode transport. Il doit être lancé d'abord sur le récepteur (option **-r**), puis sur l'émetteur (option **-t**).

tunnel.pl Ce script fait les mêmes tests que le précédent (du moins le sous-ensemble qui fonctionne, cf. les résultats) mais en mode tunnel. L'émetteur doit être lancé en dernier.

Il serait facile de combiner les deux scripts pour avoir tous les tests imaginables avec une sécurité en mode tunnel et une sécurité en mode transport entre l'émetteur/le récepteur et les passerelles, mais le nombre de tests serait alors immense et le temps d'exécution déraisonnable[13]. Il est donc plus judicieux de configurer manuellement le mode transport dans une configuration intéressante puis de lancer le script testant le mode tunnel.

Les fichiers créés par ces tests sont volumineux (plusieurs mégaoctets), aussi avons-nous créé une fonction (script **analyse.pl**) qui peut être utilisée pour en extraire les résultats concernant un ou des test(s) particulier(s). Notons tout de même au sujet de cette fonction qu'elle est celle d'un débutant en Perl et nous semble déjà, un mois après, bien naïve (mais elle marche !). Elle pourrait facilement être réécrite de manière bien plus concise...

Savoir ce qui ne va pas en cas de problème : il est parfois utile d'avoir des informations sur le sort des paquets en cas de problème. En plus des traditionnels *tcpdump* et *netstat -nr*, il faut signaler l'existence de **sysctl net.inet6.proto** et **netstat -s -f inet6 -p proto** où les valeurs intéressantes de **proto**, dans notre cas, sont principalement **icmp6** et **ipsec6**.

Nous avons constaté lors de l'utilisation d'IPsec en mode tunnel que le démon de routage *routed* chargé de router les paquets IPv4 avait tendance à planter au bout d'un certain temps, empêchant la poursuite des tests, les sockets de contrôle ne pouvant plus fonctionner. Une solution que nous n'avons pas eu le temps d'essayer serait peut-être d'utiliser un programme comme le *supervise* de D. J. Bernstein pour le relancer si nécessaire[14].

13. Nos scripts actuels testant déjà un grand nombre de possibilités, leurs temps d'exécution sont déjà de l'ordre de quelques jours. Il peut être conseillé de les éditer afin d'enlever les tests inessentiels et d'accélérer les choses.

14. *supervise* fait partie de l'ensemble de programmes *daemontools*. Voir `http://cr.yp.to/daemontools.html` pour plus de détails.

3.9 Que peut-on prédire quant aux résultats?

Il est certain que les processus de chiffrement/déchiffrement et dans une moindre mesure d'authentification, sont très coûteux en calculs. On peut donc supposer que si sur une liaison à faible débit, un modem par exemple, l'utilisation d'IPsec n'introduira pas de baisse de performances dans l'envoi ou la réception des paquets, il n'en sera pas de même sur un réseau plus rapide.

Chapitre 4

Résultats des tests sur la plateforme du Loria

4.1 La plateforme de tests

La plateforme de tests IPv6 du Loria est constituée d'ordinateurs fonctionnant (à l'exception de *canardo* qui utilise Linux) avec FreeBSD reliés entre eux par différents sous-réseaux fast-ethernet. La topologie du susdit réseau est visible sur la figure 4.1. La série des *rork-i* est en réalité une liste d'alias pour une seule machine qui est le serveur NFS sur lequel est monté un certain nombre de partitions sur les machines (voir l'annexe sur l'installation de Racoon) mais cette machine n'est pas un routeur et donc ne connecte pas les différentes sous-réseaux.

Sauf mention contraire, les tests en mode transport ont été effectués entre *sha* (Pentium II 300MHz) et *thorgal* (350 MHz) tandis que les tests en mode tunnel ont utilisé en plus *treize* (800Mhz).

4.2 Résultats

4.2.1 Tests en mode transport

Authentification

Comme on pouvait s'y attendre, le débit dépend fortement de l'algorithme employé (figure 4.2 et tableau 4.1), du moins avec les machines que nous utilisons, pour lesquelles le processeur ne permet pas de faire les opérations suffisamment vite pour saturer le réseau.

Notez que le débit de notre test témoin est limité par les performances du réseau et non par celles de la machine que nous utilisons : en effet les tests avec

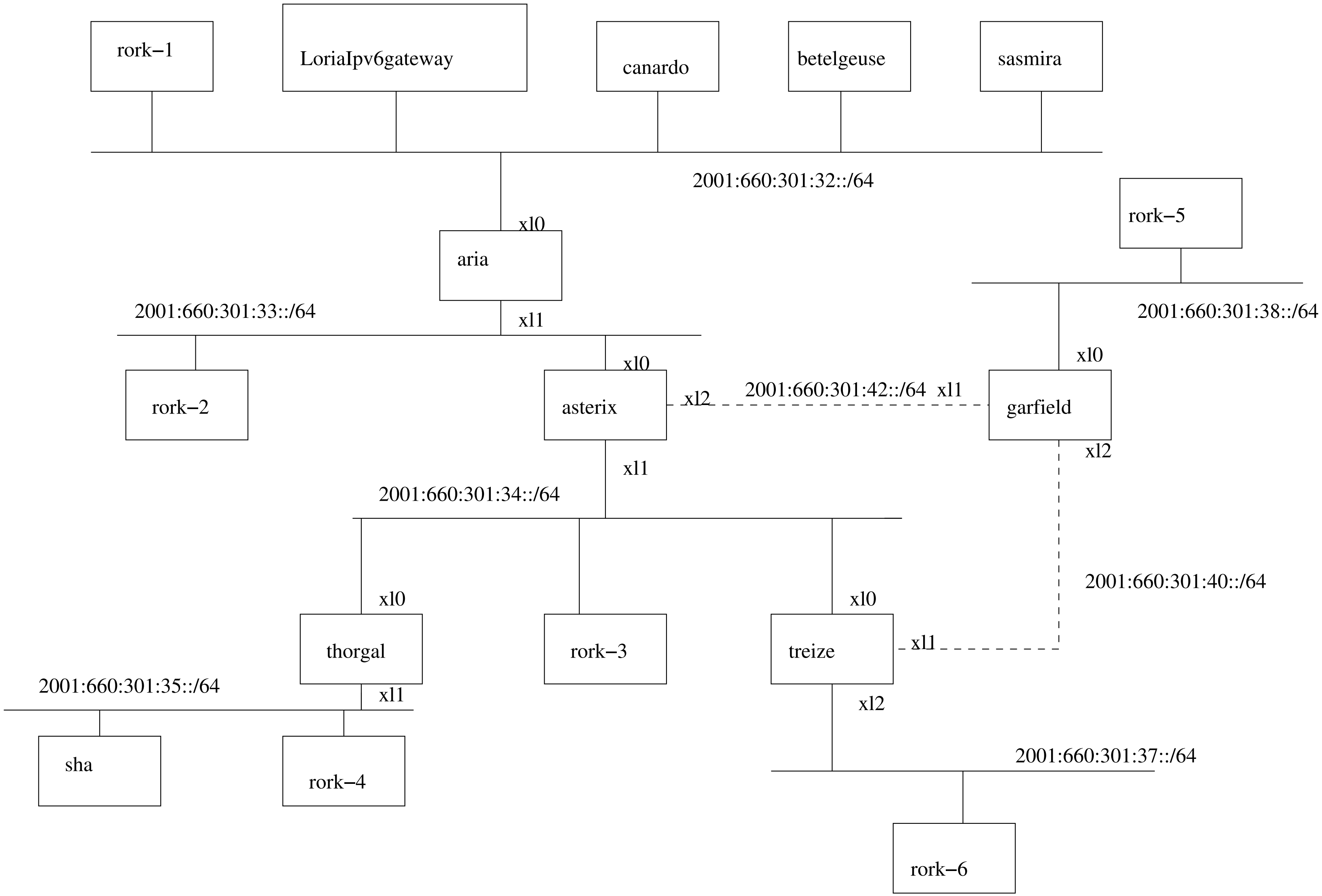

FIG. 4.1 – *La plateforme de tests IPv6 du Loria (juillet 2002)*

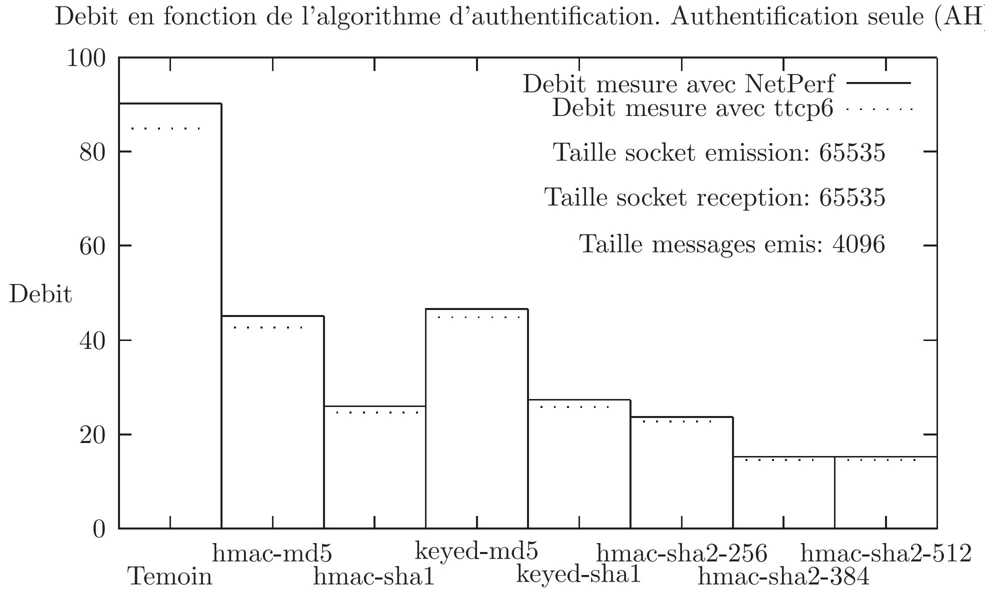

FIG. 4.2 – *Authentification avec AH (Tests en half-duplex et mode transport (débits en Mbit/s)).*

Algorithme	Débit Netperf	Débit ttcp
Témoin	90,27	84,900
hmac-md5	45,10	42,632
hmac-sha1	25,91	24,635
keyed-md5	46,60	44,818
keyed-sha1	27,25	25,850
hmac-sha2-256	23,67	22,686
hmac-sha2-384	15,25	14,588
hmac-sha2-512	15,20	14,482

TAB. 4.1 – *Authentification avec AH (Tests en half-duplex et mode transport (débits en Mbit/s)).*

émetteurs/récepteurs multiples montrent qu'une machine de cette catégorie (processeur Pentium II à environ 300-350 MHz avec 128Mo de mémoire vive) peut traiter un débit légèrement supérieur à 130 Mbit/s, ce qui nous permet de quantifier la dégradation de performance et le surcoût de calculs requis : on peut ainsi penser que la simple utilisation d'IPsec sans algorithme de chiffrement (algorithme NULL) double le temps de traitement d'un paquet (voir section suivante pour l'algorithme NULL).

Les résultats donnés ici sont pour certains paramètres de socket. On ne peut pas dire que ces paramètres influent sur les performances de façon sensible, les légères différences que l'on peut constater étant visibles également sur les tests témoins.

On constate que l'utilisation de MD5, qui est sans doute l'algorithme le moins sûr d'un point de vue cryptographique[1], donne les meilleurs résultats en termes de performances. On note également que les différences entre *sha1* et *sha2* avec une clef de 256 bits sont minimes. Il est intéressant de constater qu'alors que nous avons une nette différence de débit entre *sha2* avec une clef de 256 bits et avec une clef de 384 bits, la différence selon que l'on utilise une clef de 384 ou 512 bits est minime.

Confidentialité

Ici aussi, les algorithmes influent beaucoup sur le débit. Comme on pouvait s'y attendre, les opérations à effectuer étant plus nombreuses que pour l'authentification, les performances sont moindres[2].

Algorithme	Débit Netperf	Débit ttcp
Témoin	90,22	86,634
des-cbc	18,20	17,346
3des-cbc	8,68	8,269
simple (0 bit)	66,09	62.663
simple (576/0 bits)	65,98	62,982
blowfish-cbc (128 bits)	21,23	20,261
blowfish-cbc (256 bits)	21,20	20,226
blowfish-cbc (448 bits)	21,21	20,224
cast-128	25,79	24,523
des-deriv	18,26	17,362
rijndael-cbc (128 bits)	21,56	20,697
rijndael-cbc (192 bits)	20,23	19,446
rijndael-cbc (256 bits)	19,03	18,223

TAB. 4.2 – *Tests en half-duplex et mode transport (débits en Mbit/s).*

1. MD5 est encore considéré comme un bon algorithme mais il a le défaut de générer des empreintes de trop courte taille, ce qui le rend vulnérable à l'attaque des anniversaires. Cela fait qu'il est généralement délaissé au profit de *sha-1* ou de *ripe-md* dans les nouveaux produits cryptographiques.

2. L'algorithme *simple* a un bon débit, mais son cas est, rappellons-le, particulier: étant donné qu'il s'agit de la fonction identité, on peut considérer qu'il provoque quand même une baisse sensible des performances par rapport à son utilité lorsqu'il est utilisé seul...

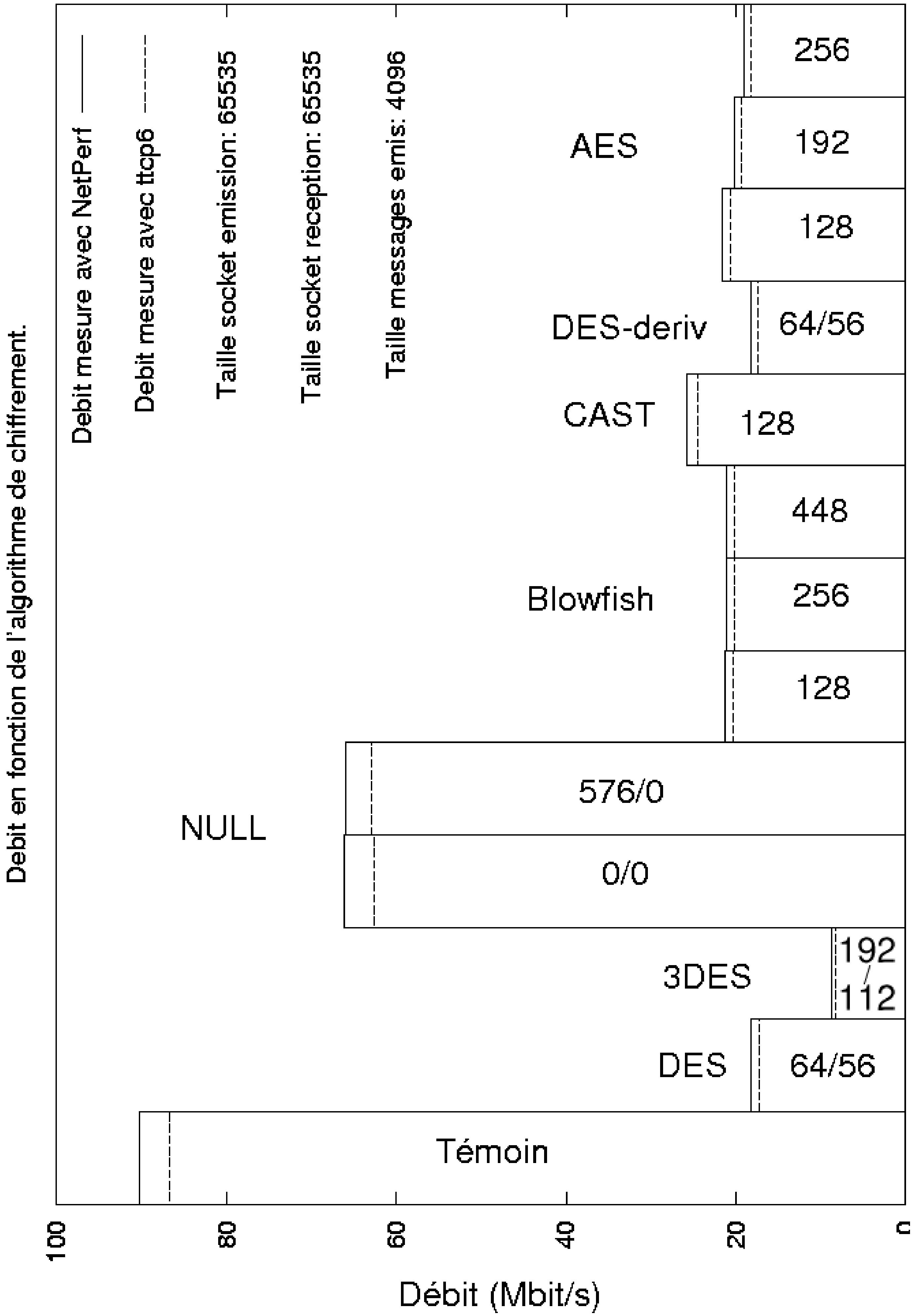

FIG. 4.3 – *Confidentialité (Tests en half-duplex et mode transport (débits en Mbit/s)).*

Comme on pouvait s'y attendre les algorithmes DES et triple-DES sont ceux dont les performances sont les plus faibles en termes de débit (même si la lenteur du DES est une éternelle surprise quand on pense à la faible sécurité qu'il offre...), comme il est indiqué sur le tableau 4.2 et la figure 4.3. C'était prévisible, le DES ayant été conçu à l'origine pour des implémentations *hardware*. On note que l'AES (rijndael-cbc) est nettement plus rapide que le triple-DES, ce qui est sa raison d'être. On note également que ses performances sont influencées par la taille de sa clef, contrairement à ce qui se passe avec Blowfish sur lequel l'influence de la taille de la clef semble être tout à fait minime. L'algorithme Cast est enfin le plus rapide, exception faite bien sûr de Null (simple). Les performances de ce dernier permettent d'évaluer le coût d'IPsec pour la partie ne dépendant pas des algorithmes cryptographiques.

Confidentialité et authentification

Nous avons essayé chaque algorithme de confidentialité avec chaque algorithme d'authentification: les tests ne permettent pas de dire qu'il y a des algorithmes "qui vont mieux ensemble" que d'autres. Par rapport aux précédents, ces tests ne réservent pas de surprise.

On note que, si l'on utilise la confidentialité, l'authentification ESP est plus rapide que l'authentification AH, mais la différence de performance est minime, comme le montre le tableau 4.3. Cette différence était prévisible, étant donné que si l'on utilise AH, il faut utiliser les deux extensions d'entête IPv6 (l'utilisation d'ESP étant toujours nécessaire pour la confidentialité), ce qui augmente le temps de traitement de chaque paquet. *A contrario*, dans le cas ou ESP gère l'authentification, celle-ci est intégrée dans le même en-tête que la confidentialité, ce qui permet de traiter plus rapidement les paquets.

Le tableau 4.4 indique les performances de l'authentification avec ESP (algorithme NULL et authentification). On constate que les performances sont très proches de celles obtenues avec AH (voir le tableau 4.1).

4.2.2 Tests d'IKE

Nous n'avons pas eu le temps de tester en profondeur IKE et son implémentation dans FreeBSD *racoon*. Néanmoins, les quelques tests que nous avons effectués montrent une baisse de débit : nous avons effectué entre *asterix* et *treize* des tests comme indiqué au chapitre précédent, où les algorithmes utilisés étaient le *triple-des* pour la confidentialité et *hmac-md5* pour l'authentification. Les paramètres que nous avons utilisés forçaient le changement de clef au cours du test, ce que nous avons été à même de vérifier grâce à

Algorithme	AH		ESP	
	Débit Netperf	Débit ttcp	Débit Netperf	Débit ttcp
Témoin (blowfish seul)	21,23	20,261	21,23	20,261
hmac-md5	17,59	16,793	18,16	17,421
hmac-sha1	13,51	12,849	14,06	13,352
keyed-md5	17,80	16,989	18,54	17,622
keyed-sha1	13,88	13,224	14,43	13,731
hmac-sha2-256	12,89	12,272	13,46	12,817
hmac-sha2-384	9,84	9,399	10,44	9,909
hmac-sha2-512	9,82	9,373	10,41	9,927

TAB. 4.3 – *Blowfish 128 bits utilisé avec authentification AH ou ESP (Tests en half-duplex et mode transport (débits en Mbit/s)).*

Algorithme	Débit Netperf	Débit ttcp
Témoin (NULL seul)	66,88	63,668
hmac-md5	43,83	41,978
hmac-sha1	25,70	24,397
keyed-md5	45,26	43,502
keyed-sha1	26,96	25,547
hmac-sha2-256	23,70	22,784
hmac-sha2-384	15,76	14,993
hmac-sha2-512	15,69	14,968

TAB. 4.4 – *NULL utilisé avec authentification ESP (Tests en half-duplex et mode transport (débits en Mbit/s)).*

tcpdump depuis une machine d'observation (*thorgal*). Le débit mesuré sans IKE est de 22,80 Mbit/s. Avec IKE, il est de 18,5 Mbit/s. La baisse de performances n'est donc pas négligeable. Il convient donc pour une utilisation réelle d'IKE de positionner la durée de vie des clefs sur une valeur suffisamment longue pour ne pas ressentir l'effet de l'échange de clef en moyenne. Il ne faut bien sûr pas non plus tomber dans l'excès inverse et mettre une durée trop longue qui pourrait éventuellement compromettre la sécurité, mais il n'y a *a priori* guère de risques de ce côté. Tout dépend en fait de la quantité de données que l'on transfère ainsi que de leur "prévisibilité"[3]

On constate sinon qu'il n'y a pas d'interruption du trafic lors du renouvellement de la clef, mais que Racoon a tendance à initier ce dernier plus tôt,

3. Si un intercepteur a une idée de ce que vous transférez, il peut tenter une *attaque par texte en clair connu...*

probablement pour que la clef soit prête au moment voulu.

4.2.3 Symétrie des résultats

Nous avons effectué les tests dans les deux sens (en intervertissant émetteur et récepteur): cette manoeuvre n'a pas permis de détecter d'asymétrie dans la charge induite sur les machines selon que l'on chiffre ou l'on déchiffre (ou que l'on signe ou vérifie une signature). Des tests effectués entre *treize* (800Mhz) et *thorgal* (350MHz) confirment ce résultat, malgré l'asymétrie des vitesses des processeurs.

4.2.4 Utilisation du CPU pendant les tests

Malheureusement, les fonctionnalités de mesure de l'utilisation du processeur des programmes de mesure de performance ne sont pas portables et bien qu'ils se compilent avec FreeBSD les résultats qu'ils donnent sont aberrants.

Nous n'avons ainsi pas été en mesure de quantifier précisement l'utilisation du processeur par IPsec, néanmoins elle est élevée, l'utilisation de manière interactive des machines lors des tests s'avérant impossible à cause du temps de latence (le temps de réponse à une commande interne au shell comme **cd** / peut prendre facilement une dizaine de secondes!).

Le programme *top* exécuté pendant les tests en mode tunnel sur une passerelle de sécurité indique qu'environ 99% du CPU est utilisé, majoritairement par des tâches de traitement d'interruptions. Nous supposons donc qu'à la réception d'un paquet IPsec le déchiffrement est fait directement par la routine de traitement de l'interruption. Il n'est pas possible d'imputer cette utilisation au simple traitement des paquets, l'utilisation des émetteurs/récepteurs en mode tunnel stagnant à 10%, avec environ la moitié en tâche de traitement des interruptions. Sur une machine émettrice ou réceptrice ou en mode transport, le programme de tests lui-même (ttcp ou netperf) utilise jusqu'à 30-35% du CPU. Lors de l'utilisation de Racoon, celui-ci peut atteindre 1% de l'utilisation des ressources durant la génération d'une nouvelle clef (pour la partie en mode utilisateur. Il est néanmoins probable que la majeure partie de l'échange se fasse en mode noyau.).

Des tests en mode transport entre *asterix* et *treize* (tous deux à 800 MHz) confirment l'importance du CPU: ces tests donnent par exemple un débit de 20 Mbit/s avec le triple-DES, soit environ deux fois et demie ce que l'on a obtenu entre *sha* et *thorgal* dont les fréquences sont justement environ deux fois et demie inférieures...

4.2.5 Tests en mode tunnel

Au niveau des performances, il n'y a guère de surprises, les résultats sont proches de ceux obtenus avec le mode transport. Il pourrait y avoir une différence si les passerelles de sécurité avaient une capacité de calcul très différente des machines qui émettent ou reçoivent les paquets, ce qui n'est pas le cas sur la plateforme du Loria. Les tests combinant un tunnel et l'utilisation d'IPsec en mode transport entre les passerelles et les émetteurs/récepteurs montrent une baisse des performances (par exemple si les mêmes algorithmes sont utilisés, le débit est divisé par deux) due à la double charge sur les passerelles. Cette baisse des performances correspond tout à fait à ce que l'on peut attendre, on ne constate pas d'effet de l'utilisation des deux modes.

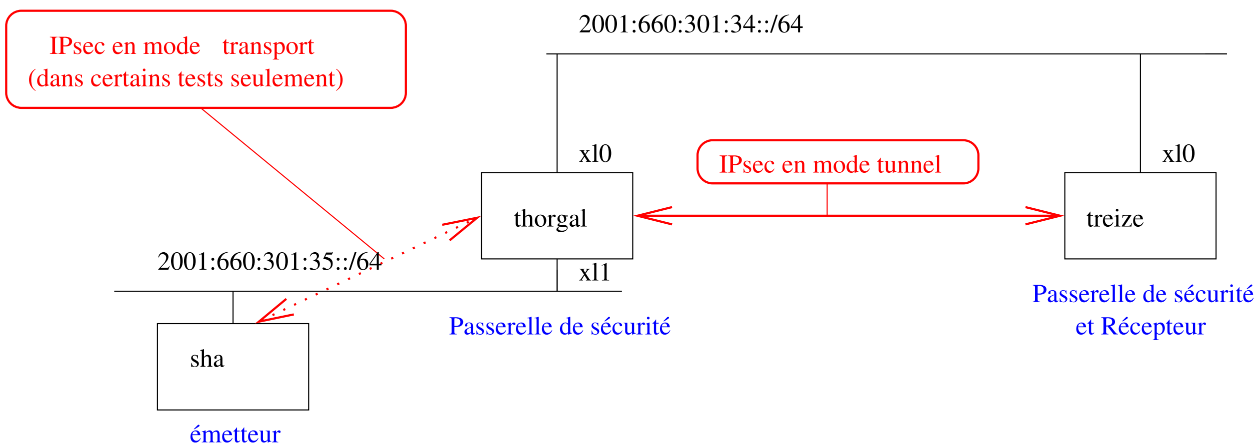

FIG. 4.4 – *Tests en mode tunnels*

On remarque néanmoins que l'implémentation d'IPsec en mode tunnel de FreeBSD n'est pas (encore) complète et que, notamment, AH ne fonctionne pas en mode tunnel. Il y a également un problème si l'on tente d'utiliser la confidentialité ESP avec l'AES (rijndael-cbc). Dans ces deux cas, les paquets sont émis correctement, mais il y a un problème au niveau de la réception par l'autre passerelle.

4.2.6 Tests avec plusieurs émetteurs

Nous avons effectué des tests de 600 secondes (10 minutes) dans une configuration où *thorgal* était le serveur avec *sha*, *asterix* et *treize* comme clients. *Sha* avait donc un accès direct tandis qu'*asterix* et *treize* étaient en concurrence pour l'accès au lien physique : les conflits étaient alors gérés par *ethernet* (protocole CSMA/CD). La taille des buffers était de 65365 octets pour la socket de réception et de 32768 octets pour celle d'émission. La taille des messages était également de 32768 octets.

Les tests étant plus longs, pour minimiser les erreurs de mesure à cause de la synchronisation (voir précédemment), il a fallu en diminuer le nombre.

N°	Algorithme	sha	treize	asterix	somme
0	Témoin	66,28	31,44	32,51	130,23
1	hmac-sha2-512	5,02	8,50	4,68	18,20
2	idem 1 avec hmac-sha2-384 pour sha	5,02	7,63	5,45	18,10
3	idem 2 avec hmac-md5 pour treize	6,17	6,73	9.89	22,59
4	hmac-sha2-512 + 3des-cbc	1,59	3.44	2,87	7,90
5	idem 4 avec blowfish-cbc (128 bits) pour sha	2.74	3.29	2.93	8,96
5	idem 4 avec blowfish-cbc (256 bits) pour sha	3.00	2.79	3.11	8,90
6	3des-cbc seul	4,29	3,83	5,07	13,19

TAB. 4.5 – *Tests avec plusieurs émetteurs (débits en Mbit/s).*

Nous n'avons pas eu le temps de d'étudier le code source du noyau de FreeBSD de manière à expliquer la façon dont se répartissent les débits entre les différents clients. Les observations empiriques semblent indiquer une préférence pour l'interface de plus petit numéro (*sl*0 passe avant *sl*1) et paraissent montrer que les fractions de seconde qui séparent le début des tests influencent le débit par la suite, sans que l'on puisse en tirer de règle absolue.

Notez que si les collisions diminuent le débit sur le brin comportant *treize* et *asterix* dans le test témoin, le trafic est si faible dans les autres tests que l'on peut probablement alors négliger leur influence. On remarque d'ailleurs que pour les tests où toutes les machines utilisent le même algorithme, la somme des débits est proche du débit obtenu pour ce même algorithme lors des tests classiques avec un seul émetteur.

Conclusion

Nous avons établi des tests permettant de voir les performances d'IPsec selon les paramètres que l'on utilise, le tout sur un réseau sous IPv6. Il apparaît que le principal facteur modifiant le débit sur une machine et une implémentation donnée est l'algorithme utilisé et que les paramètres comme la taille des buffers des sockets ne jouent pas de rôle supérieur à ce que l'on observe sans IPsec.

Les tests préliminaires que nous avons effectués sur la plateforme de tests à 100 Mbit/s du Loria nous laissent à penser que l'utilisation d'IPsec à haut-débit sera problématique, les ordinateurs que nous avons utilisés n'arrivant pas – et de loin – à saturer le réseau que nous utilisions lors de nos tests. La différence de débit nous laisse supposer que même pour une station de travail à la pointe de la technologie cela serait problématique, du moins si l'on veut qu'elle puisse encore servir à autre chose en même temps : il serait gênant de relier, dans le cadre d'une grille par exemple, des ordinateurs entre eux et que toute leur puissance de calcul passe dans le chiffrement des données qu'ils échangent!

Et le VTHD++ – sur lequel ces tests doivent être effectués – possédant un débit bien supérieur à cette plateforme de tests, même en utilisant des machines puissantes il sera difficile d'atteindre un débit correct. Il semble donc illusoire de vouloir utiliser IPsec à haut-débit à moins d'avoir des machines équipées de puces dédiées pour les opérations de chiffrement/déchiffrement. On peut imaginer que les constructeurs de routeurs créent des passerelles de sécurité permettant de relier des sites à haut-débit de manière sécurisée grâce à IPsec possédant des ASICs dédiés, ce qui irait dans le sens de la conception actuelle des routeurs gigabits [1].

On peut néanmoins supposer que l'utilisation d'IPsec sur des connexions bas-débit, telles que des connexions par modem, ne provoquerait aucune dégradation sensible des performances[4]. On peut également supposer qu'une

4. Bien sûr, dans le cas où de nombreux utilisateurs se connectent via IPsec à une même machine le problème précédent se repose... Cependant, si l'on imagine un point de présence d'un fournisseur sur lequel 1000 utilisateurs se connecteraient de manière sécurisée via des

solution pour les machines "ordinaires" avec une connexion d'un débit maximal proche du Fast-ethernet, pour ne pas grever les performances, serait une implémentation *Bump-in-the-wire* (BITW), qui consisterait en une boîte à brancher entre la machine et le réseau. Cette solution serait néanmoins coûteuse.

Sur le plan de ce que ce stage nous a apporté, il y a beaucoup de choses à dire: sur un plan "social", la découverte de la vie dans un laboratoire au sein d'une équipe de recherche. Sur un plan plus "informatique", nous avons fait connaissance avec IPv6 et approfondi certaines de ses caractéristiques, notamment bien sûr l'aspect sécurité avec IPsec. Nous avons également mis en pratique le portage d'une application en *C* d'IPv4 vers IPv6. Pendant la période de relative inaction que constitue toujours la première semaine de stage le temps que les choses se mettent en place, nous nous sommes initié au langage Perl pour le mettre immédiatement en pratique pour écrire nos scripts d'automatisation de tests. Le système utilisé étant FreeBSD, cela nous a permis de nous familiariser avec ce système, notamment sur les points de la configuration du noyau et la gestion du réseau, ce qui complète nos connaissances des sytèmes Unix qui incluaient déjà Linux et Solaris.

modems 56k, le trafic serait encore gérable par quelques PCs...

Annexe A

Installation d'IPsec

A.1 Prise en charge d'IPsec par le noyau

Une recompilation du noyau est nécessaire pour utiliser IPsec: en effet, bien que normalement obligatoire sur une implémentation IPv6, le fait d'activer IPv6 lors de l'installation du système n'active pas IPsec, comme en témoigne le programme *setkey*, puisque si on lance `setkey -D` qui devrait afficher la *SAD*, on obtient la réponse "pfkey_open: Protocol not supported"... Que faut-il donc faire pour activer IPsec?

Il faut ajouter au fichier de configuration du noyau (/usr/src/sys/i386/conf/GENERIC pour une machine x86 avec une configuration de base) les lignes suivantes:

```
options         IPSEC                   #IP security
options         IPSEC_ESP               #IP security (crypto; define w/ IPSEC)
```

ainsi que, si on veut pouvoir débugguer, la ligne

```
options         IPSEC_DEBUG             #debug for IP security
```

Il faut ensuite recompiler et installer le noyau,
`config GENERIC` (pour que nos modifications soient prises en compte)
`cd ../../compile/GENERIC/` (ou le nom de la configuration)

```
make depend
make
make install
```

A.2 Installation de *Racoon*

IPsec est maintenant installé. Néanmoins, il faut encore pour l'instant configurer toutes les liaisons manuellement (avec le programme *setkey*). C'est ici qu'intervient *Racoon* (littéralement “raton laveur”) qui est l'implémentation d'IKE de FreeBSD (faite par le projet Kame). Racoon est un démon qui se charge de la configuration automatique des liaisons.

```
cd /usr/ports/security/racoon
make
make install
```

On note ici que le package *racoon* n'est pas installé par l'installation minimale de FreeBSD. Le makefile essaie de l'installer. S'il échoue il faut le télécharger manuellement depuis `ftp.kame.net/pub/kame-v6/misc` [1] et le copier dans le répertoire qu'indique le message d'erreur.

Notons enfin que, sur les ordinateurs de la plateforme de tests du Loria, le répertoire **/usr/ports** n'est pas local et qu'il doit donc être monté avant l'utilisation par la commande

```
mount rork-4:/export/Distributions/FreeBSD/ports /usr/ports
```

où **rork-4** est à remplacer par le nom du serveur NFS se trouvant sur le brin où l'on se trouve (de la forme **rork-**n).

De la même manière le répertoire où le **make install** tente de copier l'exécutable est sur une partition en lecture seule. Il est donc inutile de lancer cette commande. L'utilisation de Racoon se fait donc à partir du répertoire où il a été compilé (**/usr/ports/security/racoon/work/racoon-20020507a/racoon**) ou de l'endroit où on l'a copié manuellement.

Il faut lui indiquer le fichier de configuration que l'on veut utiliser au lancement en utilisant l'option **-f fichier**. Un exemple d'un tel fichier est donné dans l'annexe suivante.

1. Le makefile essaie de télécharger le fichier depuis `ftp.kame.net/pub/kame/misc`. Il faudrait vérifier si cette version est compatible IPv6.

Annexe B

Exemple de fichiers de configuration de racoon

Le fichier de configuration de racoon indique quels paramètres utiliser pour IKE. Voici un exemple d'un tel fichier.

```
# "path" affects "include" directive.  "path" must be specified before any
# "include" directive with relative file path.
# you can overwrite "path" directive afterwards, however, doing so may add
# more confusion.
#path include "/usr/local/v6/etc" ;
#include "remote.conf" ;

# the file should contain key ID/key pairs, for pre-shared key authentication.
path pre_shared_key "/opt/racoon/psk.txt" ;

# racoon will look for certificate file in the directory,
# if the certificate/certificate request payload is received.
#path certificate "/usr/local/openssl/certs" ;

# "log" specifies logging level.  It is followed by either "notify", "debug"
# or "debug2".
log debug;

remote anonymous
{
exchange_mode main,aggressive,base;
#exchange_mode aggressive,main,base;
```

```
#my_identifier fqdn "server.kame.net";
#certificate_type x509 "foo@kame.net.cert" "foo@kame.net.priv" ;

lifetime time 24 hour ; # sec,min,hour

#initial_contact off ;
#passive on ;

# phase 1 proposal (for ISAKMP SA)
proposal {
encryption_algorithm 3des;
hash_algorithm sha1;
authentication_method pre_shared_key ;
dh_group 2 ;
}

# the configuration makes racoon (as a responder) to obey the
# initiator's lifetime and PFS group proposal.
# this makes testing so much easier.
proposal_check obey;
}

# phase 2 proposal (for IPsec SA).
# actual phase 2 proposal will obey the following items:
# - kernel IPsec policy configuration (like "esp/transport//use)
# - permutation of the crypto/hash/compression algorithms presented below
sainfo anonymous
{
pfs_group 2;
lifetime time 12 hour ;
encryption_algorithm 3des, cast128, blowfish 448, des, rijndael ;
authentication_algorithm hmac_sha1, hmac_md5 ;
compression_algorithm deflate ;
}
```

Voici maintenant un exemple de fichier contenant les clefs pré-partagées (avec les adresses d'*asterix* et *thorgal*):

```
# IPv4/v6 addresses
2001:660:301:34:201:2ff:fee3:6015 mekmitasdigoat
2001:660:301:34:201:2ff:fee3:6013 mekmitasdigoat
```

Annexe C

Le cd-rom

A la demande d'un de nos encadrants, nous avons regroupé les programmes nécessaires sur un CD-ROM qui est joint à ce rapport. Celuici contient les programmes Netperf et ttcp dans leurs versions originales (*.orig.tar.gz*) et modifiées (*.modif.tar.gz*), une copie des fichiers de configuration de Racoon tels que présentés dans l'annexe précédente, ainsi que Racoon lui-même, les scripts Perl que nous avons créés pour automatiser les tests et que du coup nous omettons de reproduire dans ce rapport afin d'éviter de l'alourdir d'une vingtaine de pages supplémentaires... Le répertoire contient aussi un fichier de configuration du noyau de FreeBSD avec les modifications nécessaires pour utiliser IPsec, quelques documents que nous avons utilisés, pêchés ici et là sur Internet, et une partie des résultats que nous avons obtenus (tous ceux qui ne sont pas développés dans le présent rapport).

Bibliographie

[1] Nicolas BERNARD and Clément MÉNIER. Architecture des gigarouteurs. Recherche bibliographique – module réseaux Mim2 – ENS Lyon, Mai 2002.

[2] Khadija BOUMAHDI. Expérimentation du protocole IPsec sous la plateforme IPv6. Master's thesis, Université Med V Ecole Nationale Supérieure d'Informatique et d'Analyse (ENSIAS), Institut National de Recherche en Informatique et en Automatique (INRIA), Laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA), 1999.

[3] Gisèle CIZAULT. *IPv6, théorie et pratique.* O'Reilly, troisième edition, 2002.

[4] Robert ELZ. A Compact Representation of IPv6 Addresses. RFC 1924, Avril 1996.

[5] Rob GLENN and Stephen KENT. The NULL Encryption Algorithm and its use with IPsec. RFC 2410, Novembre 1998.

[6] Brian HATCH and James LEE. *Halte aux hackers, Sécurité Linux.* Osman Eyrolles Multimédia, 2001.

[7] Hewlett–Packard Compagny, Information Networks Division. *Netperf: A Network Performance Benchmark*, 2.0 edition, Février 1995.

[8] Stephen KENT and Randall ATKINSON. Security Architecture for the Internet Protocol. RFC 2401, Septembre 1998.

[9] Boris KÖSTER. FreeBSD IPsec mini-howto, Juin 2001.

[10] Ghislaine LABOURET. La sécurité réseau avec IPsec. Technical report, Hervé Schauer Consultants (HSC), 1999.

[11] Ghislaine LABOURET. IPsec: présentation technique. Technical report, Hervé Schauer Consultants (HSC), 2000.

[12] Julian ONIONS. A Historical Perspective on the Usage of IP Version 9. RFC 1606, Avril 1994.

[13] Shoichi SAKANE. Simple Configuration Sample of IPsec/racoon. KAME Project, Septembre 2001.

[14] Bruce Schneier. *Cryptographie appliquée*. International Thompson Publishing, deuxième edition, 1996.

[15] Bruce Schneier. *Secrets et mensonges, sécurité numérique dans un monde en réseau*. Vuibert, 2001.