Panorama des algorithmes efficaces et architectures matérielles pour le filtrage réseau haut débit et la détection d'intrusions

Georges Adouko, François Charot, Sylvain Gombault, Tony Ramard, Christophe Wolinski

Panorama des algorithmes efficaces et architectures matérielles pour le filtrage réseau haut débit et la détection d'intrusions*

Georges Adouko(+), François Charot(+), Sylvain Gombault(++), Tony Ramard(++), Christophe Wolinski(+)
(+) IRISA, Campus de Beaulieu, 35042 Rennes Cedex
(++) ENST Bretagne, 2 rue de la Châtaigneraie, 35512 Cesson-Sévigné Cedex
{adouko,charot,wolinski}@irisa.fr
{sylvain.gombault,tony.ramard}@enst-bretagne.fr

Résumé : Le rôle d'un système de détection d'intrusion est de filtrer efficacement le trafic réseau. Le cœur d'un tel système réalise une analyse des paquets réseau, via des règles de détection d'attaques ou d'intrusions. Toute la difficulté consiste alors à manipuler à bon escient l'ensemble des règles de détection et à les appliquer au trafic de manière appropriée. Le filtrage de trafic réseau est aujourd'hui proposé par beaucoup d'équipements de sécurité, logiciels ou matériels. Mais leurs performances ne sont réellement adaptées qu'à l'analyse du trafic sur des liens Ethernet ou équivalents dont les débits sont autour de 100 Mbits/s. Avec le déploiement des réseaux locaux allant de 1 à 10 Gbit/s ou de liaisons longue distance proposant plusieurs dizaines de Gbit/s, le problème des performances se pose. Il est par conséquent important de définir des systèmes de contrôle d'accès et de filtrage réseau à temps d'analyse réduit et borné, condition impérative pour garantir une bonne qualité de service. Pour satisfaire ces contraintes de haute performance, de flexibilité, les dispositifs de filtrage doivent s'appuyer sur l'utilisation de matériels spécialisés. Ce papier propose un panorama des techniques et algorithmes de recherche de motifs vus sous l'angle de leur implémentation matérielle.
Mots-clés : Architecture matérielle, réseau

1 INTRODUCTION

L'essor des réseaux de communication exige de repenser les systèmes de contrôle de trafic. Les systèmes de filtrage réseau actuels ne répondent pas aux besoins futurs qui doivent allier flexibilité et haute performance pour satisfaire la complexité du trafic, et supporter des débits qui pourront à l'avenir être de l'ordre de plusieurs dizaines de Gbits/s. Dans le contexte d'un réseau en plein essor, avec des trafics malveillants de plus en plus nombreux, les systèmes de détection d'intrusion (IDS) ou de prévention d'intrusion (IPS) doivent avoir une architecture polyvalente capable d'appliquer efficacement des règles de sécurité complexes. En effet, il s'agit à la fois d'examiner les champs des protocoles réseau et de filtrer le contenu des paquets à la recherche de chaînes de caractères qui peuvent être des motifs d'attaques – on parle de signature d'attaques. Le système doit être suffisamment modulaire pour permettre une adaptation aux types d'analyses à effectuer sur les paquets réseau (analyses des entêtes et contenus de paquets, gestion des connexions, etc.). En la matière, *snort*[Roesch, 1999] se positionne comme étant la référence.

Cet article dresse un panorama des algorithmes et des techniques permettant la recherche de motifs dans le contexte du filtrage de trafic à haut débit. Ces algorithmes ou techniques exposés peuvent être appliqués dans la conception des IDS ou des IPS à la fois. Dans ce papier, nous privilégions le cadre d'études des IDS. Les caractéristiques et l'organisation d'un système de détection d'intrusion sont tout d'abord présentées (section 2). La section 3 est ensuite consacrée à un examen détaillé des différents algorithmes et techniques permettant de réaliser la recherche de motifs. Cet examen est fait dans la perspective de mises en oeuvre sur des circuits programmables de type FPGA. Enfin, la section 4 conclut et donne quelques perspectives à ces travaux.

2 SYSTÈMES DE DÉTECTION D'INTRUSION

2.1 Fonctionnalités d'un IDS

Les IDS se positionnent sur une liaison réseau pour contrôler le trafic.

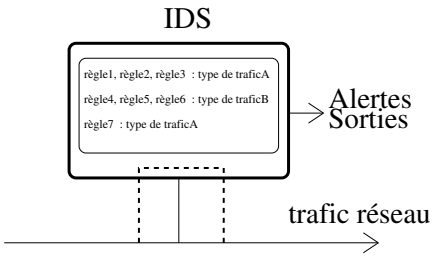

FIG. 1 – IDS filtrant du trafic réseau

*Ces travaux sont soutenus par la région Bretagne dans le cadre du projet FASTNET.

Leur finalité consiste en la détection ou la protection contre les activités malveillantes sur le réseau. Ils sont configurés sur la base d'un ensemble de règles définissant la politique de détection à appliquer au réseau. Les IDS se positionnent en composant passif sur le réseau pour scruter les données sur le trafic réseau et émettre des alertes. Le trafic est alors dupliqué pour analyse. Les fonctionnalités essentielles d'un IDS sont décrites par Northcutt et al[Northcutt et al., 2001]. Se basant sur l'évolution des attaques réseau, les auteurs présentent les contextes auxquels un IDS doit pouvoir faire face.

2.2 Architecture d'un IDS

L'architecture globale d'un système de détection d'intrusion réseau, comme détaillée par [Northcutt et al., 2001], est illustrée par la figure 2. Cette vue est également celle de l'architecture de *snort*. [Ramard, 2005] en fournit une description fonctionnelle plus détaillée, et cela, à travers les différentes versions logicielles de *snort*.

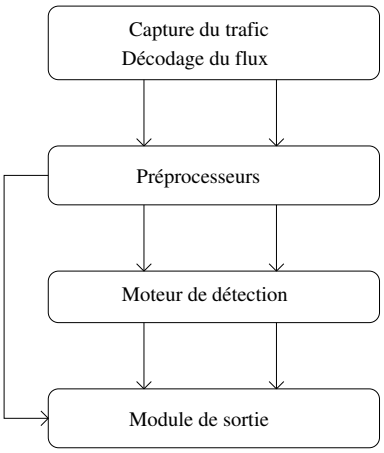

FIG. 2 – Architecture d'un IDS

Le rôle de chaque module de l'IDS est le suivant :

i. *Capture du trafic et décodage*

Dans ce module, le trafic est dupliqué et les données copiées à partir du réseau. Les paquets capturés sont au format de la couche liaison (protocoles Ethernet, ATM, etc.). Le décodage du flux est réalisé par l'extraction des paquets IP à partir de la couche liaison. Les paquets IP sont ensuite stockés dans une structure mémoire adéquate.

ii. *Pré-processeurs de paquets réseau*

Les paquets IP obtenus par le module de décodage doivent subir des traitements afin que l'IDS réalise une analyse plus efficace et plus fiable. Les traitements appliqués permettent de parer aux méthodes d'évasion utilisées par les trafics malveillants. Il s'agit en l'occurrence de l'étape de dé-fragmentation des paquets IP. Les paquets TCP séquencés font aussi l'objet d'un ré-assemblage. D'autres analyses peuvent y être développées (analyse des sessions TCP, etc.).

iii. *Moteur de détection et inspection des paquets*

C'est le cœur du système de détection d'intrusion. Il applique l'ensemble des règles de détection aux paquets. La vérification des règles requiert des opérations très exigeantes en calcul car ces dernières sont complexes (pour pouvoir détecter des attaques toujours plus complexes). Les vérifications sont orientées, soit champ de paquet (entête), soit contenu de paquet (concernant les données utiles transportées par le paquet). L'opération de recherche dans le contenu du paquet est la plus coûteuse en temps et en ressources.

v. *Module de sortie*

Ce module se charge de réaliser les instructions adéquates lorsqu'une règle de détection est vérifiée. Il permet d'imprimer un message d'alerte à l'écran, d'écrire dans un fichier *log*, ou d'envoyer un message à un serveur distant (centralisation de données).

2.3 Défis des IDS actuels

2.3.1 Complexité et modularité

Les types d'attaque évoluent vite. Les opérations à effectuer lors de l'analyse du trafic sont plus complexes. Les IDS doivent de ce fait avoir une architecture polyvalente capable d'appliquer efficacement des règles de détection complexes. La modularité du système servira à adapter efficacement les différents modules de l'IDS aux types d'analyses à effectuer sur les paquets réseau (analyses des entêtes et contenus de paquets en même temps, gestion des connexions, etc.)

2.3.2 Performances en haut débit

C'est l'une des plus grandes problématiques. Les débits réseau augmentent de manière exponentielle car les vitesses de transfert sont décuplées tous les deux ans. Les systèmes de détection d'intrusion actuels (souvent logiciels) sont vite saturés par la quantité insoutenable de traitements à faire sur les nombreux paquets reçus.

Aussi, le filtrage du trafic ne s'arrête plus à quelques analyses de données bien ciblées dans un paquet. La complexité des attaques actuelles exige souvent d'analyser tout le contenu du paquet. Cela alourdit fortement les opérations à faire sur chaque paquet. Le nombre d'opérations pour l'analyse complète d'un paquet explose quantitativement.

2.3.3 Besoin d'accélération matérielle

La flexibilité et le besoin de hautes performances implique intuitivement de rechercher des architectures couplant étroitement une approche logicielle et des parties matérielles. Le matériel est utilisé pour désengorger les nœuds (modules) de traitements logiciels saturés. Dans un IDS comme *snort*, la recherche de motifs dans les paquets est une opération très coûteuse qui peut représenter jusqu'à 80% des opérations d'analyse.

Il faut par conséquent mettre en oeuvre des techniques efficaces pour augmenter les performances de cette fonction de recherche de motifs et augmenter du coup les performances globales du système. La section 3 passe en revue nombre des travaux qui ont été menés sur ce sujet.

3 TECHNIQUES DE RECHERCHE DE MOTIFS

Les techniques de recherche de motifs sont nombreuses, allant des méthodes naïves à recherche unique (Boyer-Moore [Boyer and Moore, 1977], Knutt-Morris-Pratt [Knuth et al., 1977]), jusqu'à l'utilisation d'algorithmes à recherches multiples (Aho-Corasick [Aho and Corasick, 1975], Wu-Manber [Wu and Manber, 1994], etc. Ces algorithmes, utilisés en logiciel, ne permettent pas d'obtenir des performances supérieures à la centaine de Mbits/s.

Les solutions à haute performance passent par l'utilisation de dispositifs matériels spécialisés. Les composants programmables de type FPGA, de part leur fréquence de fonctionnement, leur technologie de réalisation, leur capacités d'entrées/sorties, leur densité, sont adaptés à la mise en œuvre de techniques de recherche de motifs.

Dans cette section, nous passons en revue les techniques mettant en oeuvre des automates déterministes ou non déterministes (paragraphe 3.1), des comparateurs structurés (paragraphe 3.2). Nous présentons également les possibilités de mise en œuvre matérielle d'algorithmes habituellement employés en logiciel (paragraphe 3.3).

3.1 Implémentations adaptées aux expressions régulières

Les expressions régulières sont très utilisées dans l'expression des règles de détection d'intrusions. A partir de ces expressions peuvent être construites des machines à états, de type automates finis non déterministes (NFA) ou automates finis déterministes (DFA).

3.1.1 NFA

Les NFA peuvent être représentés sous forme de graphes orientés construits à partir des expressions régulières à rechercher [Hopcroft et al., 2000]. Les nœuds du graphe sont les états de l'automate et les transitions sont marquées avec les caractères des motifs à rechercher, ou le caractère ε symbolisant la chaîne vide. A titre d'illustration, le NFA associé aux expressions régulières (1/0)*0 et 10*1 est donné à la figure 3. La construction de NFA pour des expressions régulières complexes est possible comme présenté dans [Sidhu and Prasanna, 2001].

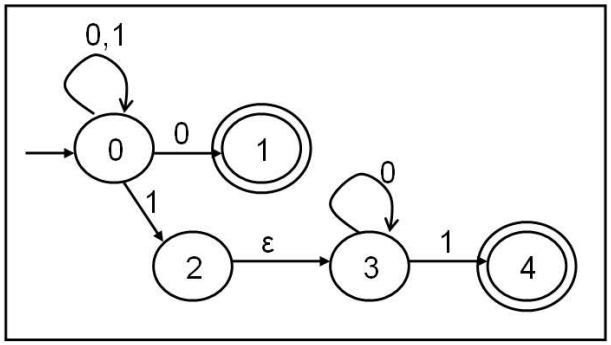

FIG. 3 – NFA des expressions (1/0)*0 et 10*1

Partant de l'état initial, on parcourt le graphe du NFA suivant les transitions appropriées au caractère lu en entrée. A l'issue de chaque cycle de traitement, le caractère suivant est lu et l'opération de mise à jour des états de l'automate est réalisée. Un motif est identifié dans le flux lorsque le NFA est dans un de ses états finaux (noeuds 1 et 4 de l'exemple, figure 3).

Par principe, plusieurs transitions peuvent être prises à la fois, activant ainsi plusieurs états. Ce comportement parallèle intrinsèque rend les NFA adaptés à une mise en œvre matérielle. Pour un ensemble d'expressions régulières de n caractères, la taille du NFA et sa complexité matérielle associée est en $O(n)$.

L'adéquation des NFA au matériel est facilitée par la transcription évidente du graphe du NFA vers un circuit logique : les états du graphe sont modélisés par des registres tandis que les transitions sont câblées avec des portes logiques en combinatoire. La figure 4 propose le circuit logique correspondant au NFA de la figure 3.

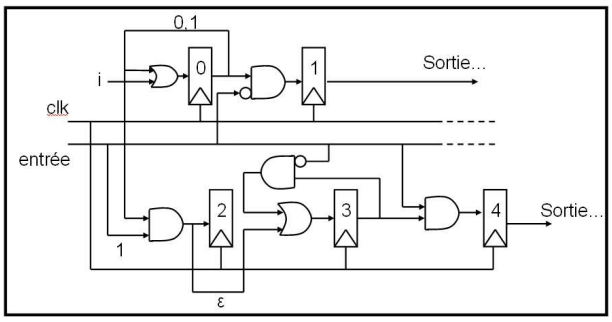

FIG. 4 – Circuit logique qui correspond au NFA décodant les expressions (1/0)*0 et 10*1

Hutchings et al. proposent dans [Hutchings et al., 2002] l'implémentation d'un ensemble de motifs décrits sous forme d'expressions régulières de 8003 caractères sur un FPGA VirtexE-2000. Une fréquence de fonctionnement de 52 MHz permet de scanner un flux à un débit de 400 Mbit/s. Le coût en ressources matérielles est de 2,57 LE/caractère (LE pour élément logique de FPGA).

Dans [Clark and Schimmel, 2004], Clark et al. montrent qu'il est possible de réduire fortement le coût matériel des NFA avec l'utilisation de décodeurs de caractères, tout en augmentant le débit d'entrée par l'utilisation du parallélisme. On parle alors de NFA décodés.

Avec les NFA décodés, l'implémentation d'un ensemble de motifs de 17537 caractères a été réalisée [Clark and Schimmel, 2004]. La mise en œuvre, d'un coût matériel de 1.1 LE/caractère, fonctionne à 100 MHz pour un débit de 800 Mbits/s donc.

En résumé, les NFA se prêtent bien à l'implémentation matérielle d'expressions régulières pour la recherche de motifs. Leur coût en surface est faible et les débits d'entrée peuvent être élevés. Les NFA peuvent nécessiter, à cause de leur non-déterminisme, l'ajout d'un mécanisme de $backtracking$ pour déterminer le motif ou l'expression qui a été détecté.

3.1.2 DFA

Contrairement aux NFA, les graphes des DFA ne comportent pas de transition ayant le caractère ε et il ne peut y avoir de transitions identiques à partir d'un même état. En conséquence, il n'y a qu'un seul état actif à la fois. La construction des DFA peut être faite à partir de motifs simples. La figure 5 met en évidence les différences entre le NFA et le DFA pour les motifs $\{$os, sos$\}$.

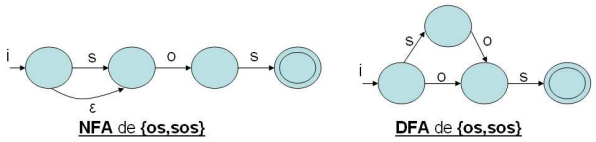

FIG. 5 – NFA et DFA de $\{$os, sos$\}$

Dans le cas d'expressions régulières, [Hopcroft et al., 2000] montrent qu'il est possible de traduire le NFA associé en un DFA. Cette étude montre qu'en théorie le nombre d'états du DFA est plus élevé que celui du NFA. La taille du DFA est estimée à $O(2^n)$ pour un motif de longueur n, ($O(n)$ pour les NFA).

Pour la construction de DFA, Moscola et al. [Moscola et al., 2003] utilisent l'outil $jlex$[Berk and Ananian, 2003] (analyseur lexical basé sur *java*). $Jlex$ génère le NFA puis le DFA associé. Ils montrent qu'en pratique, le nombre d'états du DFA d'une expression régulière est quasiment identique à celui du NFA associé.

La mise en œuvre matérielle des DFA repose sur une modélisation en automate de Moore. Le fait d'avoir un seul état actif permet de créer une représentation compacte des états (un registre de plusieurs bits stocke l'identifiant de l'état actif). En contrepartie, la complexité des fonctions combinatoires du circuit augmente fortement.

En comparaison des NFA les implémentations sont relativement peu performantes. En effet, dans [Moscola et al., 2003], la mise en œuvre sur virtex2000 du DFA d'un groupe de 20 expressions régulières de 21 caractères chacun (420 caractères au total), permet de traiter un flux d'entrée de 296 Mbits/s pour une fréquence de fonctionnement de 37 MHz. Par la mise en parallèle de quatre modules, le débit a été multiplié par quatre pour atteindre 1,18 Gbits/s, au prix d'un coût en surface élevé de 19 LE/caractère (4,75 LE/caractère par module).

3.2 Dispositifs à base de comparateurs structurés

La recherche de motifs repose principalement sur des comparaisons de caractères. Il est donc possible de tirer profit de la régularité et du nombre sans cesse croissant de ressources matérielles de composants programmables de type FPGA, rendu possible par l'évolution technologique des circuits intégrés.

3.2.1 comparateurs pipelinés et DCAM

Les comparateurs pipelinés tirent partie de l'architecture grain fin des FPGA pour exploiter au mieux leurs possibilités. Il s'agit surtout d'avoir une architecture adaptée à la structure du FPGA cible.

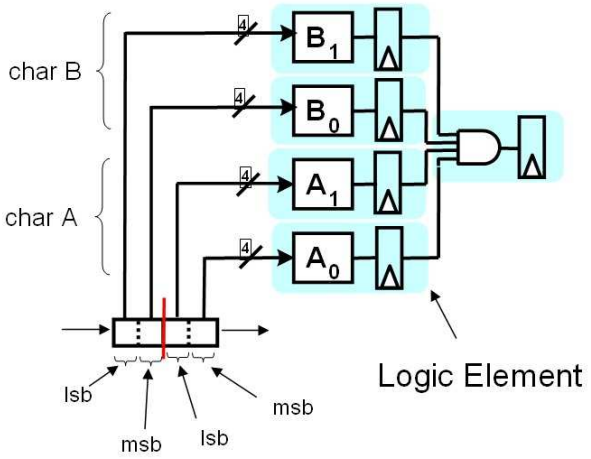

FIG. 6 – Comparateurs pipelinés pour le motif "AB"

La figure 6 montre l'architecture de comparateurs pipelinés réalisant la détection du motif "AB" tels que proposés par Sourdis et Pnevmatikatos[Sourdis and Pnevmatikatos, 2003]. Deux comparateurs de 4 bits sont utilisés pour créer le comparateur d'un caractère de 8 bits. Le premier comparateur teste les bits de poids fort (msb) alors que le deuxième s'occupe des bits de poids faible (lsb). Un comparateur élémentaire 4 bits est réalisé grâce à une LUT (lookup table). L'ensemble $comparateur\ 4bits/registre$ correspond à un élément logique de FPGA.

L'implémentation des comparateurs pipelinés permet d'obtenir des performances très élevées avec plus de 5,5 Gbits/s sur un virtex-1000, et jusqu'à 12 Gbits/s sur un virtex2 avec un facteur de parallélisme égal à quatre. Mais le coût en surface est élevé avec 4 à 5 LE/caractère (à multiplier par le facteur de parallélisme). Au plus, quelques centaines de motifs peuvent être implémentés.

En raison de la quantité considérable de ressources requise par cette approche, des optimisations visant à éliminer les redondances matérielles ont été étudiées. Des solutions à base de décodeurs de caractères comme proposées avec les NFA décodés (section 3.1.1) sont mises en oeuvre. [Sourdis and Pnevmatikatos, 2004] explique les détails d'optimisation qui aboutissent à l'architecture des comparateurs décodés DCAM (*decoded* CAM). Les performances sont grandement rehaussées par rapport aux comparateurs pipelinés. En effet, plus de 18000 caractères (l'ensemble des motifs de *snort*) peuvent être implémentés sur FPGA. Le débit de données analysées atteint 3 Gbits/s sur un virtex2-3000 avec un coût en surface qui descend jusqu'à 0,97 LE/caractère. Aussi, avec un plus gros FPGA (virtex2-6000), il a été possible d'atteindre un débit de 9.7 Gbits/s en exploitant du parallélisme [Sourdis and Pnevmatikatos, 2004] .

Les DCAM se caractérisent par leur haute performance en débit, et une occupation en surface des plus faibles parmi les techniques de recherche de motifs présentées dans cet article.

3.2.2 CAM et TCAM

Ce concept est issu d'une mise en œuvre massive et régulière de comparateurs de caractères aux contenus programmables.
Les CAM (*Content Addressable Memory*) sont des mémoires associatives. Dans [Guccione et al., 2000], Guccione et al. font brièvement, mais efficacement le parallèle entre les RAM et les CAM. Ces dernières font la comparaison d'une chaîne d'entrée avec l'ensemble des motifs de manière parallèle et immédiate. Après chaque comparaison, les données d'entrée sont décalées d'un caractère. La figure 7 présente le dispositif de filtrage de données à l'aide d'une CAM. Un signal $match$ est actif lorsqu'un motif est détecté sur le flux et la sortie $adresse$ de la CAM indique la position en mémoire du motif trouvé.

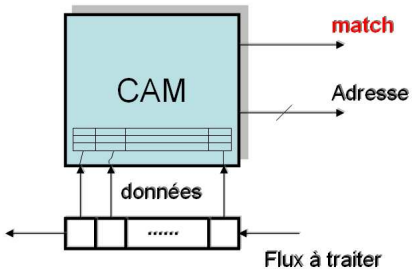

FIG. 7 – Recherche de motifs avec une CAM

Disponible sous forme de composants discrets, la structure des CAM est assez rigide car la longueur et le nombre de motifs sont fixés d'avance.
Les CAM peuvent être développées à l'aide des blocs logiques internes des FPGA. Cette approche est explorée par Gokhale et al. dans [Gokhale et al., 2002] pour obtenir l'implémentation d'un module CAM de 32 entrées de 160 bits de largeur (avec 20 * 32 blocs logiques). En utilisant quatre modules qui fonctionnent à 66 MHz, un flux à 2,2 Gbits/s peut être analysé.

Les CAM peuvent être remplacées avec profit par les TCAM qui sont plus souples grâce à l'introduction du caractère *' ?'*. Le caractère *' ?'* signifie *ne pas faire attention*. Avec cette possibilité, les TCAM permettent de rechercher des motifs plus complexes.
Les TCAM sont réalisables avec les blocs de mémoire disponibles sur les FPGA (bloc ESB sur APEX chez Altera ou bloc *SelectRam* sur virtex II chez Xilinx) qu'il faut assembler pour obtenir les dimensions souhaitées.
Les composants commerciaux permettent actuellement d'avoir jusqu'à 18 Mbits d'espace mémoire configurable en nombre et en longueur de motifs. Ainsi 1 Mbit de mémoire correspond, soit à 1024 motifs de 1024 bits de longueur, ou à 2048 motifs de 512 bits. Avec une fréquence de fonctionnement de 250 MHz, le débit de base est de 2 Gbits/s et on peut rapidement augmenter le débit par l'utilisation du parallélisme (duplication interne des motifs en mémoire, avec offset d'un caractère à chaque duplication).
En raison de leur capacité mémoire, les TCAM sont une bonne solution pour l'implémentation du dispositif de recherche de grande base de motifs (quelques centaines de milliers de motifs) dans un trafic à des débits pouvant atteindre la dizaine de Gbits/s.

3.3 Méthodes algorithmiques

La méthode naïve de recherche de motifs consiste à vérifier de gauche à droite et à chaque position du flux si il y a une correspondance avec le motif recherché. Elle effectue ainsi des petits sauts d'un seul caractère pour parcourir le motif. L'algorithme naïf est acceptable si le premier caractère du motif est rare, on aura alors dans ce cas une complexité en $O(n - m)$, où m est la longueur du flux, et n la longueur du motif. Cette méthode nécessite dans le pire cas $O((n - m + 1) * m)$ opérations pour vérifier le flux.
Pour pallier les fortes exigences en calcul des techniques algorithmiques de recherche de motifs, des algorithmes permettant une recherche en parallèle de plusieurs motifs ont été proposés. La mise en oeuvre matérielle de ces algorithmes constitue un cadre d'étude intéressant. Deux de ces algorithmes sont analysés dans ce paragraphe : Aho-Corasick et Wu-Manber. Un troisième algorithme, bien que fondamentalement différent, est également considéré. Il s'agit des filtres de Bloom réalisant un filtrage approximatif.

3.3.1 Aho-Corasick

L'algorithme d'Aho-Corasick a pendant longtemps été l'algorithme de recherche de motifs de référence de *snort*. Cet algorithme repose sur la construction et la description progressive d'un automate.
Le fonctionnement de l'algorithme repose sur l'utilisation de trois fonctions : *goto, fail* et *output* [Aho and Corasick, 1975]. La fonction *goto* correspond aux transitions représentées en flèche pleine sur la figure 8. *fail* représente les transitions en pointillé et *output* modélise les motifs rattachés aux états finaux (valeurs entre accolades).

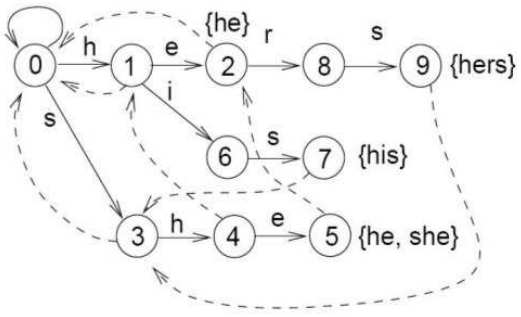

FIG. 8 – Automate des motifs { he, hers, his, she}

Le temps de traitement d'une chaîne de longueur n est de $O(n)$, il est indépendant du nombre de motifs de l'automate.

Utilisé en logiciel, ses performances ne dépassent guère la centaine de Mbits/s. Il est cependant intéressant d'aller vers des solutions matérielles de recherche de motifs basées sur l'algorithme d'Aho-Corasick.

Moult travaux sont en cours sur l'implémentation d'Aho-Corasick à l'aide de FPGA. L'approche la plus étudiée est l'idée qui consiste à stocker la structure de l'automate (de type DFA) dans une mémoire. Les transitions du graphe peuvent être décrites et représentées dans une mémoire de type RAM. Une exigence fondamentale est qu'il doit être possible d'exécuter une transition par cycle d'horloge.

Cette conception permet de profiter des hautes fréquences de fonctionnement des RAM pour atteindre des débits très élevés. La flexibilité matérielle est aussi assurée. Quels que soient les motifs à rechercher, il suffit de recharger la mémoire RAM avec les données décrivant leur automate.

Monther et al. s'inspirent d'Aho-Corasick pour décrire un accélérateur de recherche de motifs dans [Aldwairi et al., 2005]. Leur système, implémenté sur FPGA, s'appuie sur l'utilisation d'une table mémoire. En pratique l'automate, représentant un ensemble de 275 motifs, comporte environ 3000 états et son implémentation dans une RAM occupe 750 koctets. Avec la mise en œuvre de 8 automates en parallèle, le flux d'entrée est scanné 8 fois plus vite et le débit atteint est de 10 Gbits/s. On constate que l'ensemble des motifs de *snort* (1836 motifs avec *snort* v2.1) exige une mémoire de 3Mo pour le stockage de leur automate.

Van Lunteren et al. [van Lunteren et al., 2004] proposent également l'architecture d'un accélérateur de traitement de données XML qui exploite le stockage d'un automate dans une RAM et qui s'inspire d'Aho-Corasick.

3.3.2 Wu-Manber

A l'instar d'Aho-Corasick, l'algorithme de Wu-Manber [Wu and Manber, 1994] est un algorithme de recherche de motifs très efficace. Actuellement implanté sous *snort* en remplacement d'Aho-Corasick, son fonctionnement sur un trafic à 100 Mbits/s a été validé.

L'algorithme de Wu-Manber s'inspire de Boyer-Moore [Boyer and Moore, 1977] pour traiter du trafic en exécutant des sauts de plusieurs caractères à la fois. Wu-Manber utilise une méthode semblable à l'heuristique de mauvais caractère de Boyer-Moore pour analyser un trafic et rechercher plusieurs motifs à la fois (contrairement à Boyer-Moore qui est à motif unique).

Le fonctionnement de cet algorithme passe par une phase de prétraitement où deux tables essentielles sont calculées : la table de décalage et la table de hachage. Le déroulement de l'algorithme se réduit alors à des opérations simples de décalage du flux et d'identification (par accès mémoire rapide) de la valeur du saut futur à réaliser.

Parmi les algorithmes de recherche de motifs multiples, Wu-Manber détient la plus grande moyenne de caractères traités par cycle. Une mise en œuvre matérielle permettrait d'exploiter l'effet d'accélération que représentent les grands sauts effectués pendant le traitement d'un flux.

Pour un système basé sur Wu-Manber, fonctionnant à environ 250 MHz, et traitant en moyenne 3 caractères par cycle, le débit moyen d'entrée est estimé à 6 Gbits/s. Wu-Manber présente des perspectives très intéressantes quant à une mise en œuvre matérielle.

3.3.3 Les filtres de Bloom

Le filtre est constitué d'une structure de mémoire de taille m stockant un ensemble de signatures. En l'occurrence, les signatures d'un nombre n de motifs à rechercher sont calculées à l'aide de k fonctions de hachage, puis stockées en mémoire. C'est l'étape de programmation du filtre.

L'utilisation du filtre consiste à vérifier si la signature d'une chaîne de caractères, obtenue avec ces k mêmes fonctions de hachage, apparaît dans la mémoire du filtre.

Les filtres de Bloom réalisent une recherche approximative. En effet, la probabilité de faux-positifs (détection non valide d'un motif par le filtre de Bloom) est non nulle. Le taux de faux-positif vaut $\left(1 - e^{(\frac{nk}{m})}\right)^k$, comme le montrent Dharmapurikar et al. dans [Dharmapurikar et al., 2004]. Pour $k = m/n * ln2$, le taux de faux-positif est minimal et vaut $(1/2)^k$.

Des fonctions de hachage adaptées à une mise en oeuvre matérielle ont été étudiées par Ramakrishna et al.[Ramakrishna et al., 1994] qui proposent une description de ces fonctions sous la forme $Hi = x_1.d_{i1} \bigoplus x_2.d_{i2} \bigoplus ... \bigoplus x_b.d_{ib}$ où l'ensemble $< x_1, x_2, ..., x_b >$ représente les b bits de la chaîne de caractères (motifs ou flux), d_{ij} sont des entiers choisis entre 1 et m, et i est compris entre 1 et k.

Dharmapurikar et al. [Dharmapurikar et al., 2004] ont évalué les possibilités des filtres de Bloom par une implémentation sur un virtex-1000. Leur dispositif permet de rechercher jusqu'à 10000 motifs sur un trafic au débit de 502 Mbits/s, avec un taux de faux-positif élevé (0.25 = $(1/2)^2$, car il y a 2 fonctions de hachage). Pour 1416 motifs, l'utilisation de 35 fonctions de hachage fait passer le taux de faux-positif à $(1/2)^{35} \approx 3.10^{-11}$, donc quasi-nul. En analysant le flux sur plusieurs offsets consécutifs, des structures de filtres parallèles peuvent être mises en oeuvre pour augmenter le débit.

L'usage de plusieurs filtres de Bloom différents diminue fortement le taux de faux-positif. En réalité, le nombre de fonctions de hachage k augmente, ce qui diminue la probabilité de faux-positif qui vaut $(1/2)^k$.

Les filtres de Bloom imposent d'avoir des motifs de même longueur. Pour B tailles différentes de motifs, il faut implémenter B filtres de Bloom, ce qui peut conduire à un coût en surface et une complexité rapidement non négligeables.

4 CONCLUSION ET PERSPECTIVES

Les solutions pour accélérer la recherche de motifs dans un IDS ou un IPS existent. Selon le contexte du filtrage, il convient de choisir la technique la plus adaptée.

Contexte d'application / Niveau d'adéquation	à coût matériel réduit	à expressivité à base d'expressions régulières	à très haut débit	à nombre de motifs élevé
Excellent	*Filtres BLOOM* *Wu−Manber*	*NFA*	*CAM* *TCAM* *DCAM* *NFA (décodés)*	*Filtres BLOOM*
Bon	*DCAM* *NFA (décodés)*	*DFA*	*Wu−Manber* *Aho−Corasick*	*TCAM* *DCAM* *NFA (décodés)*
Moyen	*Aho−Corasick* *DFA*	*TCAM*	*Filtres BLOOM* *DFA*	*Wu−Manber* *Aho−Corasick*
Faible	*CAM* *TCAM*	—	—	*CAM* *DFA*

TAB. 1 – Classification des techniques selon les contextes d'application

Le tableau 1 propose une classification des algorithmes ou des techniques selon différents contextes d'application (coût matériel réduit, expressivité des motifs, très haut débit, nombre de motifs). Cette classification résulte d'une synthèse de l'ensemble des études réalisées. Ainsi, selon le contexte d'application (colonnes du tableau 1), un niveau d'adéquation (excellent, bon, moyen, faible) peut être attribué aux différentes techniques. A titre d'exemple, le tableau montre en grisé l'adéquation des TCAM avec les différents contextes d'application.

Ainsi, dans un contexte où de très hautes performances en débit sont exigées (10 Gbits/s et plus), ce sont les architectures à base de comparateurs qui s'en sortent le mieux. Il s'agit en l'occurrence des DCAM et des TCAM. Les principaux avantages de ces dispositifs sont d'une part leur capacité à fonctionner à des fréquences très élevées et d'autre part leur facilité de parallélisation. Les NFA décodés sont aussi une sérieuse alternative.

Si la recherche de motifs repose sur l'utilisation d'expressions régulières, l'utilisation des NFA ou des DFA est la possibilité la plus directe. Les TCAM n'offrent qu'une expressivité (et une complexité) limitée aux motifs.

Dans le contexte où le nombre de motifs à rechercher est très grand, l'implémentation des filtres de Bloom est l'une des meilleures solutions. Avec l'utilisation d'une mémoire de quelques mégaoctets, plusieurs centaines de milliers de motifs peuvent être traités. On peut aussi faire appel aux solutions TCAM pour la recherche dans de grandes bases de motifs. Seules les solutions à base de TCAM commerciales ont la taille mémoire (jusqu'à 18 Mbits) nécessaire pour traiter jusqu'à 100 000 motifs, à raison de 10 caractères en moyenne par motif.

Par ailleurs, il est à noter que la mise en œuvre de méthodes algorithmiques doit également être considérée avec attention car comme nous l'avons montré, certaines d'entres elles disposent d'un potentiel élevé en termes de performance qui reste à évaluer.
Nous explorons maintenant les possibilités de développer différentes versions matérielles de moteurs de recherche de motifs dans l'optique de la réalisation d'un système de détection d'intrusions combinant logiciel et matériel et capable d'analyser un flux à des débits de l'ordre de la dizaine de Gbits/s.

BIBLIOGRAPHIE

[Aho and Corasick, 1975] Aho, A. V. and Corasick, M. J. (1975). Efficient String Matching : an Aid to Bibliographic Search. *Commun. ACM*, 18(6) :333–340.

[Aldwairi et al., 2005] Aldwairi, M., Conte, T., and Franzon, P. (2005). Configurable String Matching Hardware for Speeding up Intrusion Detection. *SIGARCH Comput. Archit. News*, 33(1) :99–107.

[Berk and Ananian, 2003] Berk, E. and Ananian, C. S. (2003). Jlex : A lexical analyzer generator for java(tm) http ://www.cs.princeton.edu/ appel/modern/java/jlex/.

[Boyer and Moore, 1977] Boyer, R. and Moore, J. (1977). A Fast String Searching Algorithm. *Communications of the ACM*, 20(10).

[Clark and Schimmel, 2004] Clark, C. R. and Schimmel, D. E. (2004). Scalable Pattern Matching for High Speed Networks. In *FCCM '04 : Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 249–257.

[Dharmapurikar et al., 2004] Dharmapurikar, S., Krishnamurthy, P., Sproull, T. S., and Lockwood, J. W. (2004). Deep Packet Inspection using Parallel Bloom Filters. *IEEE Micro*, 24(1) :52–61.

[Gokhale et al., 2002] Gokhale, M., Dubois, D., Dubois, A., Boorman, M., Poole, S., and Hogsett, V. (2002). Granidt : Towards Gigabit Rate Network Intrusion Detection Technology. In *Field Programmable Logic and Application, 14th International Conference , FPL 2002*, pages 404–413.

[Guccione et al., 2000] Guccione, S., Levi, D., and Downs, D. (2000). A reconfigurable content addressable memory. In *IPDPS '00 : Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 882–889, London, UK. Springer-Verlag.

[Hopcroft et al., 2000] Hopcroft, J. E., Motwani, R., Rotwani, and Ullman, J. D. (2000). *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Hutchings et al., 2002] Hutchings, B. L., Franklin, R., and Carver, D. (2002). Assisting Network Intrusion Detection with Reconfigurable Hardware. In *FCCM '02 : Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 111.

[Knuth et al., 1977] Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6 :323–350.

[Moscola et al., 2003] Moscola, J., Lockwood, J., Loui, R. P., and Pachos, M. (2003). Implementation of a Content-Scanning Module for an Internet Firewall. In *FCCM '03 : Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 31–38.

[Northcutt et al., 2001] Northcutt, S., Novak, J., and McLachlan, D. (2001). *Détection des intrusions réseaux*. Campuspress, Paris, France.

[Ramakrishna et al., 1994] Ramakrishna, M., Fu, E., and Bahcekapili, E. (1994). A Performance Study of Hashing Functions for Hardware Applications. In *Proc. 6th International Conference on Computing and Information*, pages 1621–1636.

[Ramard, 2005] Ramard, T. (2005). Sécurité de réseaux à haut débit. Rapport de Master, ENST Bretagne, Rennes.

[Roesch, 1999] Roesch, M. (1999). Snort - lightweight intrusion detection for networks. In *LISA '99 : Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Berkeley, CA, USA. USENIX Association.

[Sidhu and Prasanna, 2001] Sidhu, R. and Prasanna, V. K. (2001). Fast Regular Expression Matching using FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*.

[Sourdis and Pnevmatikatos, 2003] Sourdis, I. and Pnevmatikatos, D. (2003). Fast, large-scale string match for a 10gbps fpga-based network intrusion detection system. In *proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL 2003)*, pages 880–889.

[Sourdis and Pnevmatikatos, 2004] Sourdis, I. and Pnevmatikatos, D. (2004). Pre-Decoded CAMs for Efficient and High-Speed NIDS Pattern Matching. In *FCCM '04 : Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 258–267, Washington, DC, USA. IEEE Computer Society.

[van Lunteren et al., 2004] van Lunteren, J., Engbersen, T., Bostian, J., Carey, B., and Larsson, C. (2004). XML Accelerator Engine. In *First International Workshop on High Performance XML Processing*.

[Wu and Manber, 1994] Wu, S. and Manber, U. (1994). A fast Algorithm for Multi-Pattern Searching. Technical Report TR-94-17, Department of Computer Science, University of Arizona.