Élements de conception de systèmes embarqués fortement contraints

Michaël Hauspie

▶ **To cite this version:**

Michaël Hauspie. Élements de conception de systèmes embarqués fortement contraints. Systèmes embarqués. Université Lille 1, 2014. tel-01104105

HAL Id: tel-01104105
https://hal.science/tel-01104105v1

Submitted on 16 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution - NonCommercial - ShareAlike 4.0 International License

Élements de conception de systèmes embarqués fortement contraints

Mémoire pour l'obtention de

l'Habilitation à diriger des recherches
Discipline : Informatique

par

Michaël Hauspie

présenté le
4 décembre 2014

Composition du jury :

Président :	Pierre Boulet	*Professeur - Université Lille 1 - LIFL*
Rapporteurs :	Gilles Muller	*Directeur de Recherche - Inria Rocquencourt - LIP6*
	Vivien Quéma	*Professeur - Grenoble INP-ENSIMAG - LIG*
	Laurent Réveillère	*Maître de conférences HDR - Bordeaux INP - LaBRI*
Examinateurs :	Éric Fleury	*Professeur - ENS Lyon - LIP*
	David Simplot-Ryl	*Directeur - Inria Lille Nord Europe*
Directeur :	Gilles Grimaud	*Professeur - Université Lille 1 - LIFL*

Numéro d'ordre : 41629

Remerciements

Je tiens en premier lieu à remercier Gilles Muller, Vivien Quéma et Laurent Réveillère pour avoir accepté la lourde tâche de rapporter mes travaux. Merci également à Pierre Boulet et Éric Fleury d'avoir bien voulu participer au jury.

Je ne serais sûrement pas arrivé à ce moment de ma carrière sans David Simplot-Ryl et Gilles Grimaud. Merci tout d'abord à David de m'avoir repéré alors que j'étais étudiant et jeté en pâture à Gilles lorsqu'il avait besoin de « petites mains » pour réaliser son prototype de thèse. C'est certainement grâce à vous deux que j'ai choisi d'embrasser la carrière de chercheur. C'est avec une joie non dissimulée que j'aurais le plaisir de vous compter parmi les membres du jury. Merci à Gilles également pour l'aventure 2XS, pour tes conseils avisés lors du co-encadrement de nos doctorants et pour toutes les fois où j'ai pu compter sur toi, tant dans ma vie professionnelle que personnelle.

Merci également à Nathalie Mitton pour le travail que nous avons effectué lors de l'encadrement de la thèse de Tony.

Les résultats présentés dans ce document sont le fruit d'un travail d'équipe, tant avec les doctorants que les ingénieurs, les assistantes et tous les membres des équipes RD2P et 2XS. Merci donc à Geoffroy, Tony, Damien, François, Thomas, Julien, Samuel, David, Jean-François, Anne, Karine, Michèle, Maryline et tant d'autres.

Merci à tous mes collègues du département informatique de l'IUT A, et à tous les étudiants qui m'ont subit, qui contribuent à rendre passionnant le volet enseignement de mon métier.

Merci également aux membres de Melting Pot, qui se reconnaîtront, à Sylvain, Grégory, Florian et bien d'autres qui m'aident à profiter agréablement d'une bonne partie du temps que je ne passe pas au travail.

Un merci particulier à Denis Simándy pour la qualité de son enseignement musical. Jouer du piano à forcement dû contribuer d'une façon ou d'une autre à mes travaux, au moins en me permettant de me détendre.

Merci à mes parents, ma famille, à Bernard, Chantal, Lydie, Christophe, Yann, Pauline, Julien, Gabrielle, Nicolas, Jacques, Armelle et tous les autres pour les moments exceptionnels passés ensemble.

Enfin, un énorme merci à Julie, Louise et Émilie pour le bonheur que vous m'offrez chaque jour.

Résumé

Au cours des dernières années, les systèmes embarqués sont de plus en plus présents dans notre vie de tous les jours. Ils sont dans notre portefeuille, dans notre voiture ou dans nos appareils ménagers. La tendance actuelle est à contrôler de plus en plus d'objets à l'aide de ces systèmes. Les grands acteurs de l'industrie ont déjà commencé à envisager le futur de l'internet, l'internet des objets. Dans les années à venir, de plus en plus de nos objets seront « intelligents », connectés... et sujets à des fautes logicielles.

Les micro contrôleurs, de minuscules ordinateurs possédant quelques centaines d'octets de mémoire, sont au cœur de cette révolution. Ils deviennent de moins en moins cher et de plus en plus puissant. Néanmoins, à la différence de nos ordinateurs de bureau ou des serveurs, le but de l'industrie des micro contrôleurs n'est pas la puissance. En effet, contrôler la température de notre maison et le taux d'humidité de notre cave à vin ne nécessite pas des processeurs cadencés à plusieurs gigahertz. Cela ne nécessite même pas plusieurs cœurs de calculs. Le véritable besoin de ces équipements est d'être bon marché, d'être produit en très grand volumes, d'êtres petits, facilement intégrables et d'utiliser une faible quantité d'énergie électrique.

Mes recherches de ces dernières années vont dans le sens de la réduction du fossé séparant logiciels bon marché et logiciels sûrs et performants pour ces équipements. Je suis convaincu qu'en offrant des outils « intelligents » et des chaînes de production logicielle efficaces, nous pouvons aider les développeurs débutants, ou non spécialisés, à produire du logiciel embarqué de bonne qualité pour un coût de développement raisonnable. J'ai également porté mon attention sur l'optimisation et la sécurisation des logiciels et des protocoles réseau afin que l'internet des objets puisse devenir une réalité tout en respectant la vie privée des utilisateurs et en offrant une alternative durable sur le plan énergétique.

Je me suis principalement intéressé à trois champs de recherche. Tout d'abord, j'ai cherché à permettre l'utilisation de technique de développement standard (objets, composants, programmation en Java...) à la faible capacité mémoire des équipements embarqués. À l'inverse, je me suis également intéressé à l'utilisation de langages dédiés à une application afin de permettre à des spécialistes du domaine de la sécurité réseau d'exprimer des algorithmes de détection d'intrusions. À l'aide d'une suite d'outils dédiée, ces algorithmes sont compilés et optimisés automatiquement pour être utilisés dans une architecture distribuée de sondes embarquées. Enfin, je me suis intéressé aux protocoles réseau économes en énergie pour interconnecter les équipements embarqués dans le cadre des villes intelligentes.

Abstract

In the past decades, embedded systems become more and more present in our daily life. They are present in our wallet, in our car, in our house appliances and the current tendency is to control more and more things by using them. Major companies already started to envision the future of internet which become known as the Internet of Things. In the next years, more and more of our things will be smart, connected... and subject to software flaws.

Micro-controllers, very small devices with a few hundreds bytes of memory, are at the heart of this revolution. They become cheaper, smaller and more powerful. Yet, contrary to the technologies used in our desktop computers or our server farms, the goal of the industry that creates these devices is not power. Indeed, controlling temperature in your home or humidity in your wine cellar does not need Gigahertz core. It even does not need multiple cores. What these devices need is to be cheap, to be able to be produced in very high volumes and to use small amount of electrical energy. In this context, the moore's law does not help us to have more cores in one device, but it allows to produce more devices with the same amount of silicon.

The research I made in the past years try to reduce the gap between cheap software and robust and efficient software for these devices. I think that, by providing smart tools and production toolchains we can help junior or non specialized developers so that they can produce good embedded software for a reasonable development cost. I also focus my work on optimizing and providing efficient and secure software and network protocols, so that the Internet of Things can become a reality while respecting user privacy and being sustainable from a energy point of view.

I focused my research on three aspects. First, I focused on how to allow larger softwares do be developped in embedded systems. In particular, we proposed techniques to allow Java/JavaCard code to be executed from a non-addressable memory using a full software cache. I also looked at how to use domain specific languages to ease the implementation of a collaborative network intrusions detection system. Thanks to adapted tools, the software, writen in the DSL, can be compiled for a wide range of probes while offering garanties on the produced software. Last, I focused on energy aware network protocols in the context of smart cities.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte professionnel

Après l'obtention de ma thèse en 2005, et un passage dans l'industrie de la synthèse d'images temps réel, j'ai été recruté en tant que maître de conférences dans l'équipe RD2P/POPS dans laquelle j'ai travaillé sur les protocoles pour réseaux de capteurs. Après quelques résultats sur la sécurité des protocoles de routage [M5, C12, C13], j'ai commencé à m'intéresser à la façon d'implémenter réellement, et sur du matériel contraint, les protocoles que j'avais précédemment étudiés en simulation. Même si mes différentes expériences personnelles m'avait sensibilisé aux problématiques des systèmes d'exploitation et aux implémentations qui nécessitent performance et faible empreinte mémoire, les travaux que j'ai effectués pendant le projet européen WASP [M3, M4] sur les implémentations de piles de protocoles m'ont amené à reconsidérer mon orientation scientifique.

Cette volonté a coïncidé avec la fin de l'équipe projet POPS d'Inria dont je faisais partie. J'ai donc décidé de rejoindre Gilles Grimaud, avec qui je n'avais pas encore travaillé directement en recherche malgré notre appartenance à la même équipe, mais avec qui je partageais de nombreuses affinités, dans la création de l'équipe 2XS [1]. Nous avons élaboré ensemble l'axe scientifique de l'équipe et décidé de concentrer nos recherches sur les problématiques liées à la performance et à la sécurité des systèmes embarqués fortement contraints. C'est donc assez naturellement que j'ai épaulé Gilles dans l'encadrement de la thèse de Geoffroy Cogniaux, très orientée système pour carte à puces.

Les contributions que je présente dans ce mémoire reflètent ma légère réorientation scientifique et, même si le lien entre ces travaux n'est pas absolument direct, ils ont tous été réalisés avec la même idée en tête : rendre plus accessible l'utilisation des systèmes embarqués, soit de façon générale, soit comme moyen pour répondre à un problème précis. Mes travaux sont généralement initiés suite à des contacts avec des industriels et portés par la volonté de résoudre leur problème avec une approche scientifique.

1. *eXtra Small, eXtra Safe*

1.2 Contexte scientifique

Le principal cadre qui qualifie mes recherches est la production de systèmes fortement contraints, basés sur des micro-contrôleurs ou des circuits dédiés. Deux exemples de ce type de système sont la carte à puce et les réseaux de capteurs. Bien qu'il désigne normalement un éventail plus large, comme les systèmes temps réel de contrôle dans l'avionique par exemple, je désignerai dans ce document les systèmes qui m'intéressent par le terme « système embarqué ». L'écrasante majorité des équipements informatiques déployés à l'heure actuelle sont des systèmes embarqués. Presque tout le monde dispose d'au moins deux de ces systèmes sur lui en permanence : sa carte bancaire et sa carte vitale. L'essor de l'informatique embarquée n'est pas limité aux seules cartes à puce et de plus en plus de produits les utilisent pour offrir de nombreux de services. Ils sont présents dans les voitures, dans les appareils ménagers, dans les jouets ou dans les processus de contrôle industriels.

1.2.1 Spécificités des systèmes à micro-contrôleurs

Ce qui caractérise les systèmes embarqués qui nous intéressent est la faible quantité de mémoire, de puissance de calcul, de capacité de communication et, de plus en plus souvent, d'énergie disponible. Contrairement aux ordinateurs de bureau ou aux serveurs, les systèmes très contraints n'ont pas, ou peu, bénéficié de la loi de Moore pour augmenter aussi significativement leur capacité que nos bon vieux ordinateurs personnels. Ce qui a principalement motivé l'industrie dans la possibilité d'intégrer de plus en plus de transistors dans la même surface de silicium est la production de plus en plus d'unités à coût équivalent. À titre d'exemple, le marché de la carte à puce est passé de quelques centaines de millions d'unités produites en 1994 à une prévision de 7,7 milliards pour l'année 2014 [2]. Si les capacités des cartes ont bien sûr également évolué, c'est surtout ce nombre d'unités produites qui a connu la plus forte progression.

Les systèmes qui nous intéressent sont basés sur des microprocesseurs dont les caractéristiques ont été dictées par la réduction du coût de fabrication et/ou de la consommation énergétique. Ces caractéristiques sont :

- un microprocesseur 8, 16 ou 32 bits dont l'horloge dépasse rarement les 20 Mhz ;
- une mémoire RAM d'une taille d'au plus quelques dizaines de kilooctets ;
- une mémoire persistante, adressable, utilisée pour stocker le logiciel, d'une capacité d'au maximum quelques centaines de kilooctets (et le plus souvent moins de cent).

Cette particularité des systèmes embarqués nous ramène à une informatique dont la puissance est comparable à celle des années 80. Cependant, la complexité des fonctions que l'on demande à ces systèmes d'assurer est bien plus importante. Ils sont connectés, sujets à de nombreuses formes d'attaques et, de plus en plus, limités par la quantité d'énergie à leur disposition. Les challenges pour qui doit développer un système embarqué sont donc bien différents de ceux qui occupent les développeurs d'applications à destination de serveurs ou de grosses stations de travail multi-cœurs, disposant de

2. Source Eurosmart

plusieurs gigaoctets de mémoire vive.

1.2.2 Challenges du développement sur micro-contrôleurs embarqués

L'informatique embarquée est un univers radicalement différent de l'informatique de bureau, pour laquelle les outils et méthodes de production de logiciels sont connus de la base des programmeurs. En effet, un vaste panel de méthodes et d'outils est enseigné à tous niveaux des cursus universitaires informatiques. Les étudiants diplômés par ces formations sont familiers de la programmation objet (souvent en langage Java), des patrons de conceptions, de la programmation web, etc. De fait, le langage C est de plus en plus perçu comme un langage quasi ésotérique auquel on ne veut se frotter que si l'on a pas d'autre choix et sûrement pas si, en plus, il n'y a pas de système d'exploitation ou de librairie standard pour fournir un minimum d'outils comme `malloc` ou `printf`.

Dans ces conditions, trouver un bon développeur étant rapidement capable d'écrire du logiciel pour des cibles aux contraintes fortes, sans système d'exploitation (ou réduit à sa plus simple expression) peut relever de la gageure. Quand, en plus, le logiciel que l'on veut produire doit respecter des impératifs de sécurité et/ou de performance, le terme « gageure » devient un doux euphémisme.

Nous avons la conviction que pour rendre accessible le développement de logiciels pour l'embarqué, il est indispensable d'adapter les outils et les méthodes de production de logiciel au développeur et non l'inverse. C'est fort de cette conviction que j'ai proposé les contributions des chapitres 2 et 3.

Un autre aspect limitant des systèmes embarqués est leur capacité de communication. Dans le cas de la carte à puce, la liaison est limitée à 115 kbit/s voire moins. Dans les réseaux de capteurs, les interfaces utilisées sont sans fil et la norme la plus utilisée actuellement, IEEE 802.15.4, est limitée à 250 kbit/s. L'adaptation des protocoles habituels (comme IP, TCP, HTTP...) sur ces équipements se confrontent à cette limitation ainsi qu'à la faible quantité de mémoire des équipements. L'utilisation d'interfaces de communication sans-fil permet d'envisager un déploiement à large échelle sans besoin de créer une infrastructure particulière. Cependant, ceci implique que les objets utilisent des protocoles réseau adaptés et qu'ils possèdent leur source d'alimentation propre. Or, le composant qui consomme le plus dans un capteur embarqué est l'interface de communication radio. Les protocoles conçus devront donc prendre en compte cet aspect. C'est cette problématique que j'ai adressée dans le chapitre 4.

Enfin, l'adoption massive des systèmes embarqués, ainsi que la volonté de diminuer leur coût de développement en font une cible privilégiée pour des actions malveillantes. En effet, ces systèmes nous authentifient, nous permettent d'effectuer des paiements, contrôlent nos voitures, nos avions, nos équipements ménagers et leur compromission permettrait d'altérer nos vies et de récolter de précieuses informations sur nos comportements. La sécurité de tels systèmes n'est donc pas à négliger [3, 9]. Que l'on s'intéresse à la confidentialité de nos données personnelles, à leur intégrité ou à la disponibilité des services offerts par les systèmes embarqués, le logiciel produit pour eux doit être sûr. Cependant, les contraintes évoquées précédemment rendent encore plus complexe l'établissement de mesures assurant cette sécurité. Un angle d'attaque pour une partie

de ces problématiques est d'utiliser les méthodes formelles pour avoir une garantie forte de correction du logiciel produit. Nous avons démarré une thèse sur ce sujet et, comme les résultats sont encore trop succincts pour être présentés comme une contribution en tant que telle, j'énoncerais des perspectives à mon travail en m'appuyant sur cette piste dans la conclusion du document.

1.3 Contributions

Je présente dans ce document mes recherches effectuées durant les cinq dernières années. Mes trois contributions principales sont portées par trois thèses dont deux ont été soutenues. La troisième sera normalement soutenue au cours de l'année universitaire en cours.

Exécution de code depuis une mémoire non-adressable Dans la thèse de Geoffroy Cogniaux, thèse CIFRE en collaboration avec la société Gemalto, nous avons proposé une méthode permettant l'exécution de code depuis une mémoire non-adressable. La cible visée par cette étude est la carte à puce, dont la quantité de mémoire adressable, utilisée pour stocker le logiciel, est faible et limite fortement l'utilisation de paradigmes de programmation évolués (objets, composants, etc.) et la réutilisation de librairies. Les cartes disposent cependant d'une mémoire non-adressable, accessible par un bus série, de taille beaucoup plus importante mais que le processeur est incapable d'utiliser directement pour l'exécution de code.

Nous avons rendu possible l'utilisation de cette mémoire pour exécuter du code interprété à l'aide d'une technique de pré-interprétation. En plus de cette solution au problème, nous avons proposé une étude poussée du comportement de mémoires cache implémentées sans support matériel. Cette étude a permis l'élaboration de la technique de pré-interprétation en nous donnant une vision claire des goulots d'étranglement liés à l'utilisation de la mémoire série à travers un cache logiciel.

DISCUS, une architecture de détection d'intrusions distribuée Dans la thèse de Damien Riquet, financée par une bourse ministère, nous proposons une architecture de détection d'intrusions réseau distribuée, basée sur une multitude de sondes hétérogènes. Pour pallier aux difficultés de produire du logiciel pour des cibles très hétérogènes (du pare-feu puissant au circuit dédié FPGA, en passant par des microcontrôleurs), nous avons proposé un langage dédié et une suite d'outils permettant d'optimiser le logiciel produit en fonction de la cible et de garantir des propriétés de correction sur ce dernier.

Communication dans le contexte des villes intelligentes Dans la thèse de Tony Ducrocq, effectuée dans le cadre d'un projet ANR ayant pour objet l'optimisation de la collecte de déchets dans les villes, nous avons proposés des algorithmes de routage de données économes en énergie. Ces algorithmes sont adaptés à deux scénarios de collecte dans lesquels les conteneurs de déchets sont équipés de systèmes embarqués capables de faire des mesures sur l'environnement et de communiquer sans-fil entre

eux. Nos contributions mettent à profit les spécificités de l'application visées pour offrir de bonnes performances.

1.4 Organisation du document

Ce document est organisé de la façon suivante. Le chapitre 2 présente notre contribution sur l'exécution de logiciels écrits en langage standard Java/JavaCard depuis une mémoire non-adressable. Le chapitre 3 présente la solution que nous apportons à la détection d'intrusions à l'aide de sondes embarquées. Le chapitre 4 propose des algorithmes de collecte de données économes en énergie. Enfin le chapitre 5 conclu mes travaux, en propose des perspectives et dresse un bilan personnel sur mon expérience d'encadrement de thèses.

Lorsque le texte citera les publications issues de mes travaux, la référence sera notée *[Jx]* pour un article de revue, *[Cx]* pour un article de conférence ou atelier, *[Mx]* pour des publications diverses tels que des livrables de projets ou des posters et *[Tx]* pour les thèses que j'ai encadrées.

Ce document se veut volontairement synthétique. Il présente donc les travaux que j'ai réalisés de façon succincte en essayant d'en dégager les aspects les plus importants des résultats et la méthodologie employée pour y parvenir.

Pour des résultats détaillés, j'invite le lecteur à se rapporter à [T4, C9, C10, C11] pour le chapitre 2, [C3, C4, C7, C8] pour le chapitre 3 et [T3, J1, C1, C2, C5, C6] pour le chapitre 4, dont une sélection est fournie en annexe de ce document.

Chapitre 2

Production de logiciels embarqués à l'aide d'outils de développement standards

Après plusieurs années passées à écrire du code ne pouvant dépasser quelques kilo-octets, il m'est difficile de me souvenir d'un seul logiciel sérieux qui ne m'a pas amené à me débattre avec le compilateur, l'éditeur de lien, l'optimisation des structures de données et l'architecture même du logiciel pour gagner les quelques octets de trop pour qu'il puisse être utilisé. Produire du code compact nécessite des compétences qui s'acquièrent lentement et un développeur expérimenté dans ce domaine livrera du code plus propre, et le fera plus rapidement qu'un développeur habitué au monde de l'ordinateur personnel. Il n'est même pas rare que la différence de vitesse de production soit d'un ordre de grandeur.

Bien que je reste persuadé qu'il est important, voire indispensable, de comprendre les implications d'un élément d'architecture sur la taille du code produit, je suis également convaincu que dépasser la limite de taille de logiciel de quelques dizaines de kilooctets à ne serait-ce que d'un megaoctet peut ouvrir le développement de logiciel embarqué à un nombre bien plus important de programmeurs, en particulier par l'utilisation de librairies.

Les travaux présentés dans ce chapitre ont démarré après plusieurs discussions avec notre partenaire industriel historique, Gemalto. À l'époque de ces discussions, Gemalto produisait des cartes à puce qui pouvaient stocker plusieurs megaoctets de données utilisateur mais étaient – et sont toujours – contraintes par la faible quantité de mémoire pouvant contenir du code. En effet, ces deux types de données – utilisateur et code – sont stockés dans deux types de mémoires persistantes différentes, utilisant des technologies qui se distinguent par leur densité [1], et leur mode d'accès. La cible typique choisie pour notre étude est une carte à puce possédant 8 kilooctets de RAM, 64 kilooctets de flash NOR et plusieurs mégaoctets de flash NAND.

Dès le début de notre réflexion, nous étions confiants sur la possibilité d'utiliser cette

1. Surface de silicium par bit stocké

flash NAND, disponible en grande quantité, pour stocker du code et l'exécuter. La vraie question était plutôt de savoir si cette exécution allait être suffisamment performante pour être utilisable.

Ce chapitre présente les résultats de la thèse de Geoffroy Cogniaux, financé par un contrat CIFRE avec Gemalto.

2.1 Motivations

2.1.1 La capacité de stockage comme principal verrou

L'espace de stockage constitue certainement le verrou majeur qui empêche un accès confortable aux cibles telles que la carte à puce aux développeurs habitués aux ordinateurs de bureau. Il est évident que les faibles capacités de calcul de ce type de matériel n'arrange rien, mais un programmeur qui développe pour la première fois dans ce contexte est en général confronté d'abord à la réalité de la taille du code plutôt qu'à son manque de performance.

Quand il s'agit de programmer pour l'embarqué, le développeur doit faire un usage minimal de deux types de mémoires :

— la mémoire volatile (RAM) qui est utilisée par le logiciel pour ses données de travail (variable globales, tâs, pile) ;
— la mémoire persitante utilisée pour stocker le code du logiciel ainsi que les données utilisateurs qui doivent rester valides même après un redémarrage de la cible.

La RAM est disponible en quantité très limitée. Un logiciel embarqué doit donc limiter ses données de travail. Il faut alors utiliser peu de variable locales, peu ou pas d'allocations dynamique et limiter la profondeur d'appel des algorithmes.

Deux types de mémoires persistantes sont disponibles sur les microcontrôleurs actuels. Ces deux types ont été inventés par Toshiba en 1984 [84] et diffèrent principalement par l'organisation des cellules de stockage. Ceci influe sur la surface de silicium nécessaire au stockage d'un bit (et donc au coût de la mémoire) et sur la façon qu'a le microprocesseur d'accéder aux données en lecture et en écriture.

Le premier type, la flash NOR, est utilisé principalement pour le logiciel et est accessible directement par le microprocesseur à la granularité d'un octet. Même si la quantité disponible est supérieure à celle de la RAM, elle reste faible. Le logiciel produit devra donc avoir une faible empreinte mémoire, de l'ordre de quelque kilooctets voire quelques dizaines de kilooctets. Le deuxième type de mémoire persistante disponible est utilisé généralement pour les données utilisateurs (souvent sous la forme d'un système de fichiers). Cette mémoire, de la flash NAND similaire aux stockages de masses USB, est disponible en grande quantité (plusieurs centaines de kilooctets, voire plusieurs mégaoctets) mais n'est accessible par le microprocesseur qu'à la granularité d'une page et le plus souvent par un bus série de type SPI [67]. Ce mode d'accès rend impossible l'exécution **directe** de code qui s'y trouverait par le microprocesseur. Il faudra impérativement charger le code depuis la mémoire série vers la RAM ou la flash NOR avant de pouvoir l'exécuter.

La tendance actuelle des techniques de développement est à la réutilisation, à l'usage de patrons de conception ou de *framework* logiciels. La majeure partie des développeurs sortant des formations en informatique est donc plus à l'aise avec des langages comme Java, C#, Python, PHP et leurs librairies imposantes tant du point de vue fonctionnalités que du point de vue taille de code. Ces librairies ne sont néanmoins pas la seule cause de surpoids des logiciels. Les paradigmes de programmation sont également coupables. Dans le contexte de Java embarqué, une étude [31] a montré que, pour une même fonctionalité, le paradigme utilisé influence la taille du code produit. Dans cette étude, un ordonnanceur de tâches est implémenté en Java de plusieurs façons différentes, d'une version simple en programmation impérative à une version utilisant au maximum les paradigmes de la programmation par objets comme les classes abstraites, les interfaces, l'héritage ou les accesseurs. La taille du bytecode Java est augmentée d'un facteur deux entre la version impérative et la version complètement objet. Dans le contexte de la carte à puce, où le nombre d'unités produites par an s'élève à plusieurs milliards [2], doubler la quantité de flash NOR nécessaire n'est pas envisageable pour de simples raisons de coût de fabrication. C'est également pour cette raison que les cartes sont souvent taillées au plus juste pour l'application visée. Cependant, depuis une vingtaine d'années, les cartes à puces peuvent non seulement embarquer plusieurs applications, mais proposent également de la *post issuance* en permettant le déploiement d'applications alors que la carte est en possession de son utilisateur final. Cette possibilité technique est devenue un standard grâce à JavaCard [13] mais est encore limitée par la faible quantité de mémoire persistante depuis laquelle le microprocesseur peut exécuter du code.

Si ces observations sont d'ores et déjà vraies sur la carte à puce, elle le seront également dans d'autres systèmes devant être déployés à large échelle comme les capteurs en réseaux, la RFID active ou les actionneurs domotiques.

2.1.2 Technologies de mémoire persistante

A l'heure actuelle, la technologie la plus utilisée pour stocker le code dans les microcontrôleurs ou les cartes à puces est l'EEPROM de type flash NOR [84] proposée par Toshiba. Cette mémoire est persistante et peut-être lue avec une granularité d'un octet. Néanmoins, l'écriture est plus problématique et se fait uniquement par page (généralement d'une taille de 512 octets à 32 kilooctets). Sur les équipements embarqués, elle est généralement aussi rapide que la RAM en lecture séquentielle ou aléatoire. Cette propriété rend cette mémoire particulièrement adaptée au stockage du logiciel que le microprocesseur doit exécuter.

Par contre, cette mémoire est chère à produire. Elle est donc disponible en quantité limitée à quelques centaines de kilooctets dans le meilleur des cas [3]. Une autre technologie, moins chère, a été proposée également, et au même moment, par Toshiba. Cette mémoire est la mémoire flash de type NAND qui est maintenant produite à très large

2. 7 milliards d'unités produites en 2013 (source Eurosmart).

3. La plupart des cartes à puces du marché sont plus souvent limitées à une quantité inférieure à 64 kilos.

échelle grâce à son adoption pour les « clés USB » sur le marché grand public. Cette mémoire est également disponible dans les cartes à puce ou les plateformes à base de micro-contrôleurs, mais n'est pas utilisée pour stocker du code.

La flash NAND utilise une technologie similaire à celle de la flash NOR mais ne permet pas un accès en lecture à une granularité fine. La lecture se fait au travers d'un bus série, en sollicitant tout d'abord le contrôleur de la mémoire pour charger une page dans un registre interne au contrôleur. Cette page est ensuite accessible, octet par octet via le bus série. Cette complexité en terme d'utilisation permet de limiter le nombre de portes nécessaires à la réalisation de la puce et donc la surface de silicium qu'elle occupe. De plus, l'utilisation massive de cette technologie dans les domaines grand public (clé USB, carte mémoire multimédia, ...) a augmenté le volume de production de ce type de mémoire et donc contribué à en diminuer le coût. Grâce à cela, intégrer plusieurs mégaoctets de stockage flash NAND est devenu viable sur le plan industriel pour la production de cartes à puce ou d'autres systèmes embarqués.

Néanmoins, la flash NAND n'est pas si parfaite qu'il n'y paraît. En effet, son interface paginée, via un bus série, si elle est parfaitement adaptée à une utilisation comme support d'un système de fichiers, est un frein à l'exécution de code qui y serait stocké. Il est intuitivement facile d'imaginer que, si l'accès aux données n'est pas purement séquentiel, les performances d'une telle mémoire sont loin d'égaler celle d'une flash NOR. Les tableaux 2.1, 2.2 et 2.3 confirment cette assertion en présentant respectivement les performances en lecture d'une flash NOR, d'une flash NAND et le facteur de ralentissement correspondant. Ces valeurs ont été mesurées sur une architecture typique de carte à puce, avec un microprocesseur cadencé à 20 MHz. La taille de page de la mémoire considérée est de 2048 octets, auxquels s'ajoutent 64 octets de contrôle qui sont également chargés dans le registre à chaque chargement de page.

Récupération	Lecture aléatoire	Lecture séquentielle
50 ns	19,07 Mo/s	19,07 Mo/s

TABLE 2.1 – Performance d'une mémoire flash NOR, cadencée à 20Mhz.

	Chargement	Récupération d'un octet	D'une page complète	Lectures aléatoires	Lectures séquentielles
Meilleur	25 µs	50 ns	145,6 µs	0,0065 Mo/s	13,41 Mo/s
Plus Mauvais	25 µs	75 ns	198.4 µs	0.0048 Mo/s	10.04 Mo/s

TABLE 2.2 – Latence et débit crête d'une mémoire flash NAND cadencée à 20 Mhz, à partir de mesure obtenues dans des conditions réelles.

Dans ces conditions, récupérer chaque instruction depuis la mémoire flash NAND avant de l'exécuter n'est pas une option viable. En effet, le schéma d'accès au code, s'il n'est pas complètement aléatoire, n'en est pas moins loin d'être totalement séquentiel.

	Lecture séquentielle	Lecture aléatoire
Meilleur	1,46	2933
Plus mauvais	1,89	3972

TABLE 2.3 – Facteur de ralentissement entre une flash NAND et une flash NOR.

Des méthodes statiques, basées sur de l'analyse de code, ont été proposées dans [28] pour utiliser la mémoire flash comme mémoire de données. Le code y est analysé à la compilation, sur un poste de travail classique, afin de planifier les accès à la flash et donc de pouvoir les déclencher à l'avance, de façon asynchrone, au moment de l'exécution du système.

L'inconvénient principal de cette technique, outre la nécessité d'avoir un contrôleur de mémoire flash qui supporte le chargement de page asynchrone, est qu'une connaissance complète du code est nécessaire à la création du système final.

Dans le cas de l'industrie des cartes à puce, les systèmes cartes doivent être capables de charger des applications après l'émission de la carte (et donc du système). Nous cherchons donc à proposer un système dynamique, qui sera capable d'exécuter du code stocké en mémoire flash NAND. Ce type d'approche a été étudié de façon approfondie pour les systèmes classiques (ordinateurs de bureau, serveurs, ...) et les solutions qui ont été proposées peuvent globalement se diviser en trois catégories.

Tout d'abord, le code complet de l'application peut être chargé dans une mémoire adressable avant d'exécuter cette dernière. Dans notre cas, la première candidate, la RAM, est disponible en quantité bien trop faible pour se permettre cette approche. La deuxième candidate, la flash NOR a des temps d'écriture très importants. Si on souhaite que le système exécute plusieurs applications, les latences de commutation d'applications seraient trop élevées.

Une deuxième solution est d'utiliser un tampon en RAM et de ne charger qu'une partie de l'application à exécuter. Quand l'application demande à exécuter du code qui n'est pas chargé dans le tampon, on récupère les données souhaitées depuis la flash vers le tampon et on reprend l'exécution. Avec cette solution, les transferts entre la flash et le tampon seront important et les performances seront mauvaises. La figure 2.1 présente le débit effectif observé sur une mémoire flash sur cible réelle à 20 Mhz. Les tampons utilisés sont stockés en RAM et sont de la taille d'une page de flash. Le programme utilisé était un simple programme de test de quelques centaines de milliers d'instructions. On voit que sacrifier 8 Ko de RAM (ce qui est énorme dans notre contexte) ne permet d'accéder qu'à un débit de 0.42 Mo/s, soit plus de 47 fois plus lent que la NOR.

La dernière solution est d'utiliser un mécanisme de mémoire cache qui va tenter de conserver en mémoire les données les plus fréquemment utilisées à l'aide de multiples tampons de taille plus réduite. Ces mémoires ont été fortement étudiées depuis les années 80 [86] pour étendre la mémoire des machines à l'aide de mémoire de stockage de masse.

On peut alors penser qu'il suffit de reprendre les algorithmes bien connus de la littérature afin de proposer une solution à l'exécution de code depuis une mémoire flash NAND dans le contexte des cartes à puce. Cependant, nos cibles ne disposent d'aucun

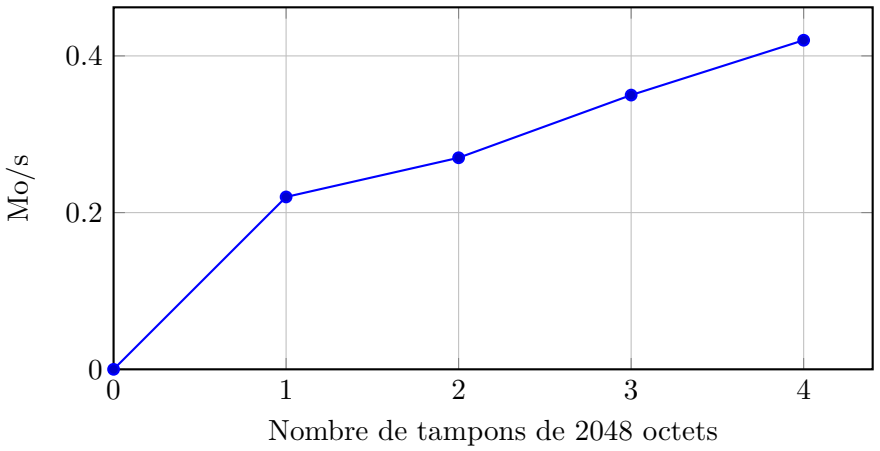


FIGURE 2.1 – Exécution en place sur une puce à 20 Mhz, avec des tampons de 2048 octets.

support matériel tel que la MMU [4] pour proposer un espace d'adressage virtuel et déclencher des fautes quand une donnée n'est pas disponible à l'aide d'une interruption. De plus, la quantité de mémoire RAM disponible sur nos système cible étant très faible, les structures de gestion des mémoires cache vont avoir un impact non négligeable sur la quantité de mémoire disponible pour le système et les applications.

La première idée qui est venue est que les mécanismes de cache, et en particulier des politiques de remplacement de page, devaient être repensés pour s'adapter à notre contrainte. J'ai alors poussé Geoffroy à étudier en profondeur le comportement des mémoires cache dans le contexte qui nous intéresse de façon à confirmer cette idée et à identifier les faiblesses des mécanismes connus dans notre contexte.

2.2 Analyse de performances d'un système de cache logiciel

Cette section discute brièvement de l'étude que nous avons mené sur les performances des mémoires cache dans le contexte d'équipement tels que les cartes à puces.

2.2.1 Mémoires cache

Une mémoire cache est un morceau de mémoire qui contient des copies de fragments d'une mémoire plus lente (la flash NAND dans notre cas). Ces copies sont partielles, limitées et généralement désordonnées (dans le sens où la continuité de la mémoire d'origine n'est pas respectée dans la mémoire cache). Les données présentes dans la mémoire cache doivent rester dans celle-ci le plus longtemps possible pour limiter les accès à la mémoire lente.

4. Memory Management Unit

Les mécanismes internes à une mémoire cache sont implémentés par deux algorithmes principaux :

— une stratégie de recherche dont le but est de répondre la question « *la donnée x est-elle disponible en mémoire cache ?* » ;
— une politique de remplacement qui est en charge de décider quelles données doivent être supprimées de la mémoire cache pour pouvoir y placer de nouvelles.

Dans les systèmes standards, la stratégie de recherche est « implémentée » de façon matérielle grâce à une MMU. L'espace d'adressage virtuel qui correspond aux données non situées en cache génère une faute quand le microprocesseur tente d'y accéder, ce qui permet au système de reprendre la main et de charger les données nécessaires de la mémoire lente à la mémoire cache. Dans notre cas, comme aucun support matériel n'est disponible, le logiciel qui s'exécute devra être en mesure de vérifier que les prochaines instructions à exécuter sont bien chargées dans la mémoire cache.

Quant à la politique de remplacement, elles ont été étudiées de façon très approfondies depuis des années et la littérature sur ce sujet est disponible à foison. Les algorithmes les plus connus sont « First-In-First-Out » (FIFO) ou « Least-Recently-Used » (LRU) et leur approximations [87, 63] ou variations [78, 43, 50]. Un autre algorithme bien connu est l'algorithme MIN [90]. Cet algorithme a été prouvé comme étant optimal dans le sens ou aucune autre politique de remplacement ne peut obtenir moins de défaut de cache. Cependant, il n'est pas implémentable en pratique car il est basé sur un Oracle. Il permet, par contre, d'obtenir une borne maximale sur la performance d'un cache si on étudie le système à l'aide d'une trace d'exécution passée.

2.2.2 Méthodologie d'évaluation

Pour la suite de cette section, nous utilisons le terme de bloc élémentaire défini comme étant une succession d'octets qui doivent être chargés **ensemble** en mémoire pour pouvoir être utilisés. Un bloc peut être un élément d'une structure de données, une variable, ou un bloc de code basique. Si l'on doit charger un bloc de la mémoire flash vers la mémoire cache, il faut donc que tous les octets le constituant soit chargés.

Quand la quantité de mémoire attribuée au cache diminue, le nombre de blocs pouvant être chargés en RAM simultanément diminue. La performance du cache va donc dépendre de sa capacité à gérer ces blocs et donc :

1. des propriétés de ces blocs comme leur taille ou la fréquence à laquelle ils sont utilisés ;
2. de la politique de remplacement de page du cache qui choisira les blocs à supprimer quand le cache est plein ;
3. de l'interface entre le système et le cache, et la façon dont le gestionnaire de cache va chercher si un bloc est déjà présent dans le cache ou non.

Il est évident que diminuer la quantité de mémoire RAM disponible va influer sur ces trois mécanismes.

Pour analyser le comportement de ces différents mécanismes, nous avons exécuté plusieurs jeux de tests dans un environnement d'exécution contrôlé qui génère une

trace de tous les accès mémoire du micro-processeur. À l'aide de ces informations, nous avons pu simuler différentes stratégies de cache et mesurer les impacts de sa taille, des algorithmes utilisés sur son efficacité en terme de débit pur, de nombre de défauts de cache ou de coût d'exécution. Les performances du cache sont obtenues à partir du temps T nécessaire à la lecture effective d'une donnée :

$$T = T_{\text{recherche}} + T_{\text{cache}} \tag{2.1}$$

$$T_{\text{cache}} = \begin{cases} T_{\text{màj}} + T_{\text{RAM}} & \text{si la donnée est en cache } (\textit{hit}) \\ ou & \\ T_{\text{FLASH}} + T_{\text{màj}} + T_{\text{RAM}} & \text{en cas de défaut de cache } (\textit{miss}) \end{cases} \tag{2.2}$$

où T_{cache} dépend de la disponibilité de la donnée dans la mémoire cache et $T_{\text{màj}}$ est le temps nécessaire pour mettre à jour les structures internes du cache. Ce dernier temps dépend principalement de l'algorithme de gestion des pages utilisé.

Les propriétés de la flash NAND sont prises selon les données constructeur d'une mémoire flash disponible dans les matériels mis à notre disposition par Gemalto. Les résultats sont donnés pour un processeur ARM7 cadencé à 20 Mhz où T_{RAM} est égal à 50 ns et T_{FLASH} est de 145,6 μs. Les algorithmes de fonctionnement du cache sont implémentés tels que décrits dans la littérature et dans de nombreuses implémentations open-source.

Le choix des jeux de test a été fait de façon à refléter les applications que l'on peut trouver sur des cartes à puce tout en étant suffisamment gros pour nécessiter la mise en place d'une mémoire cache. Ainsi, nous avons observé le comportement du cache pour du code natif et du bytecode Java. Le code natif est basé sur la suite MiBench [58] (en particulier le sous-ensemble sécurité). Nous avons également observé l'exécution de la machine virtuelle KVM [73], implémentation de référence de J2ME. Concernant le code Java, nous avons choisi d'utiliser la suite de test Richards [32] et de l'exécuter sur la machine virtuelle KVM et sur JavaCard. Dans le cas particulier de JavaCard, nous avons également observé le comportement d'une application mimant le comportement d'un assortiment d'applications JavaCard réelles, que nous avons nommé JCProfil. Ce choix a été fait principalement en raison de problème de propriété intellectuelle des applications réelles.

2.2.3 Résultats et analyses

La première chose qui peut être observée est la fréquence d'utilisation des blocs de données. La figure 2.2 donne une cartographie de la mémoire occupée par le code de la machine virtuelle KVM colorée par la fréquence d'utilisation des différents blocs de code. On peut ici utiliser une métaphore courante avec la température et distinguer les blocs « chauds », utilisés très fréquemment, et les blocs « froids » utilisés rarement. Comme on peut le voir, seul un petit sous ensemble du code est fortement utilisé. Ceci est plutôt de bonne augure pour la mise en place d'un cache car, avec la bonne configuration, il devrait pouvoir rapidement placer tous les blocs utilisés de façon intensive en RAM,

tout en restant de taille raisonnable. Nous avons pu faire un constat similaire sur les autres programmes de nos jeux de test ainsi que sur le bytecode ou les métadonnées liées à JavaCard [C9].

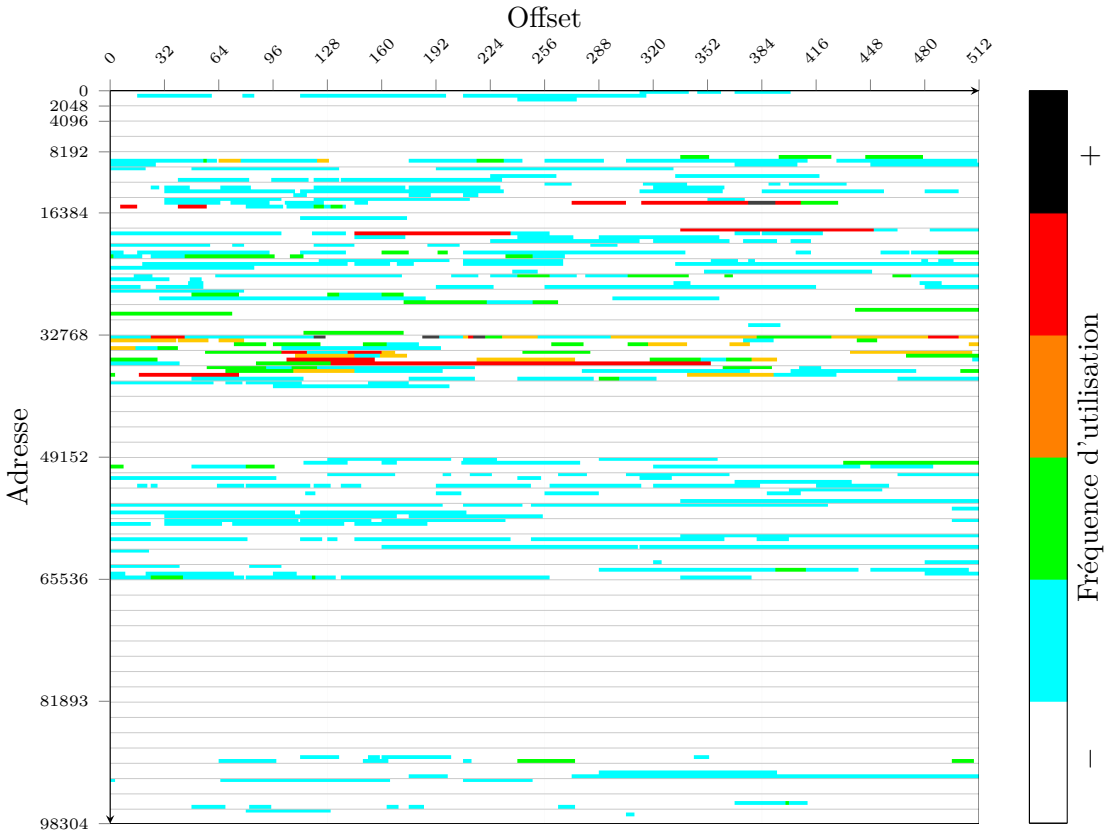


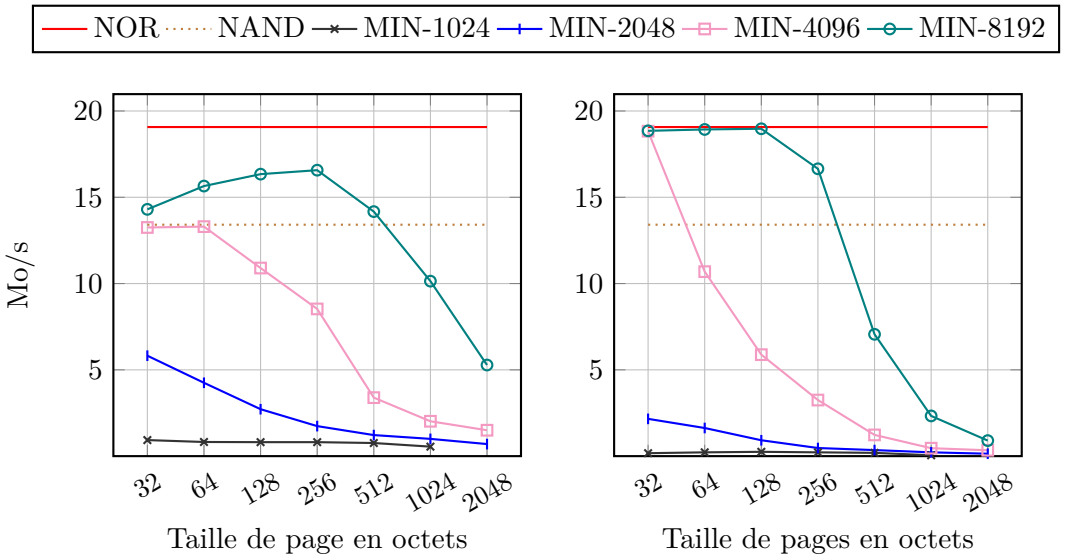
FIGURE 2.2 – Cartographie d'accès du programme KVM.

Nous avons choisi d'observer indépendemment chaque paramètre de configuration du cache, puis d'observer son comportement global. Les résultats concernant tous les paramètres méritant de nombreuses discussions, sortant du cadre synthétique de ce document, je présenterai surtout les résultats qui nous ont permis d'éclairer notre réflexion sur le chemin à suivre pour l'exécution de code depuis la mémoire série.

Nous avons tout d'abord cherché à obtenir une borne supérieure des performances que l'on serait en mesure d'obtenir dans un cas idéal. Ceci nous aurait permis d'écarter immédiatement la piste des mémoires cache dans le cas d'une borne supérieure très défavorable et d'estimer sur quel paramètre nous pourrions agir pour rendre le système efficace.

Pour obtenir cette borne, nous avons évalué les débits du cache en utilisant l'algorithme de renouvellement MIN. Comme notre évaluation est réalisée à base de trace après l'exécution réelle du système, MIN devient implémentable pour l'analyse de ces traces car nous disposons de l'oracle nous permettant toujours de choisir le renou-

vellement optimal des pages de cache. Toujours pour assurer l'obtention de la borne supérieure, le coût de l'interface de gestion du cache est considéré comme nulle, du même que le coût de gestion des pages (*i.e.* $T_{\text{recherche}} = T_{\text{màj}} = 0$). La figure 2.3 montre le débit atteignable par un cache de code et un cache de données en fonction de la taille des pages gérées par le cache.



(a) Code compilé (b) Structures de données

FIGURE 2.3 – Borne supérieure de performance en fonction de la taille des pages d'une mémoire cache implémentant l'algorithme de remplacement MIN.

À des fins de comparaison, la figure présente également le débit de la flash NOR ainsi que le débit crête de la flash NAND en accès séquentiel. Nous avons utilisé des mémoires cache de taille totale allant de 1Ko à 8Ko et des tailles de page variant de 32 à 2048 octets. La première constatation est, de façon évidente, qu'augmenter la taille du cache augmente ses performances. Il est plus intéressant de noter que la taille des pages, et donc le nombre de pages gérées par le cache, influe fortement sur sa performance et qu'à taille de cache constante, il vaut mieux diminuer la taille des pages pour permettre à la politique de remplacement de fonctionner efficacement. On observe également, ce qui encore une fois est relativement intuitif, que si le cache est trop petit pour contenir un ensemble suffisamment large des points chauds du code ou des données, même la meilleur politique de remplacement ne peut le rendre efficace.

Une fois cette borne obtenue, nous avons comparé MIN et d'autres politiques de remplacement. L'intérêt de cette approche et de confirmer (ou d'infirmer) que la bonne piste de recherche est de proposer une stratégie de renouvellement pertinente. Nous avons donc évalué, dans les mêmes conditions que précedemment, les politiques de remplacement LRU, aléatoire et FIFO. La figure 2.4 présente les résultats sur l'exécution de KVM avec un cache de 4096 octets.

Les résultats présentés confirment la tendance observée précédemment sur MIN.

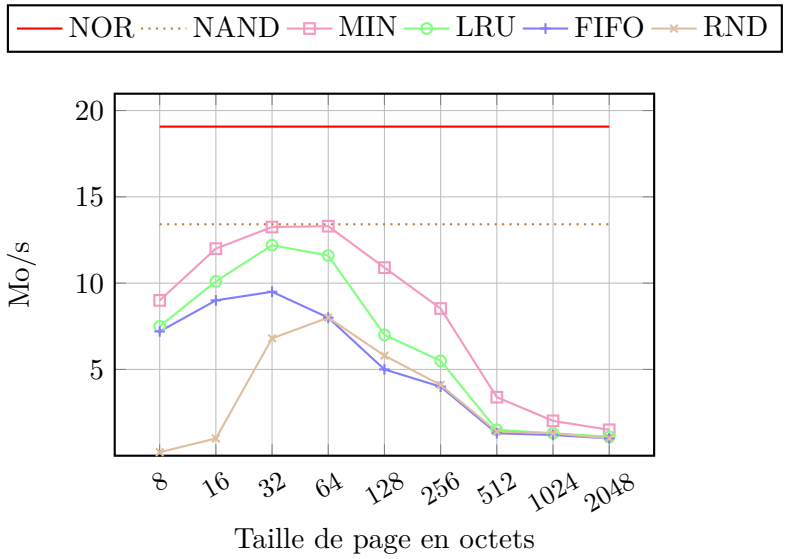


FIGURE 2.4 – Comparaison de MIN avec d'autres stratégies de remplacement.

La taille des pages manipulées est d'une importance capitale dans la performance du système. Ce qui nous paraissait moins intuitif par contre, est que LRU, dans une bonne configuration de taille de pages, est presque aussi performant que MIN. Dans ces conditions, l'idée de gagner en efficacité en changeant la politique de remplacement de page est tout simplement invalidée. Il suffit de choisir LRU avec une bonne taille de pages. Il nous a donc fallu poursuivre l'analyse pour identifier le véritable frein à l'utilisation des mémoires cache pour exécuter le code situé en mémoire flash NAND.

Comme nos cibles ne nous permettent pas de bénéficier d'un support matériel à l'implémentation d'un cache, nous avons ensuite choisi d'observer l'influence de l'algorithme de recherche permettant de trouver si un bloc de données est disponible dans le cache ainsi que le coût de l'interface logicielle du cache. Ce qui rend les systèmes embarqués particuliers par rapport aux systèmes standards, et qui justifie d'autant plus l'étude de ces paramètres, est que la mémoire est si limitée que l'algorithme de recherche aura, en plus de son impact en terme de complexité en temps, un impact fort par sa complexité en mémoire. Nous avons considéré trois algorithmes de recherche : les listes chainées, les arbres binaires de recherche (en particulier l'arbre rouge-noir [89]) et les tables de hachage.

Il est généralement considéré que le surcoût mémoire de ces algorithmes est négligeable au regard de la taille des données manipulées. Ce n'est en revanche pas le cas dans notre contexte. Le tableau 2.4 donne le surcoût mémoire des différents algorithmes pour des mémoires cache de 1 Ko à 4 ko. Pour que la comparaison des algorithmes soit valable dans notre contexte, il faut que la quantité de mémoire allouée au cache (données plus structure de contrôle) soit la même. En effet, si l'on compare les arbres rouge-noir manipulant un cache de 2048 octets divisé en 128 pages et une liste chaînée qui manipule également un cache de 2048 octets divisé en 8 pages, la quantité de mémoire

nécessaire est très différente. Dans le premier cas, 3088 octets seront affectés au cache alors que dans le deuxième, seuls 2156 octets seront nécessaire. Même si on suppose que le premier algorithme offre de meilleures performances, sur un équipement dont la quantité de RAM ne dépasse pas 8 Ko, le surcoût de près de 1000 octets est loin d'être négligeable et peu affecter le fonctionnement du système et des applications.

Algorithmes	Cache	Nombre de pages				
		128	64	32	16	8
Liste chaînée	1024	+151.17%	+76.17%	+38.67%	+19.92%	+10.55%
	2048	+75.59%	+38.09%	+19.34%	+9.96%	+5.27%
	4096	+37.79%	+19.04%	+9.67%	+4.98%	+2.64%
Arbre rouge-noir	1024	+301.56%	+151.56%	+76.56%	+39.06%	+20.31%
	2048	+150.78%	+75.78%	+38.28%	+19.53%	+10.16%
	4096	+75.39%	+37.89%	+19.14%	+9.77%	+5.08%
Table de hachage	1024	+169.14%	+94.14%	+56.64%	+37.89%	+10.55%
	2048	+84.57%	+47.07%	+28.32%	+18.95%	+14.26%
	4096	+42.29%	+23.54%	+14.16%	+9.47%	+7.13%

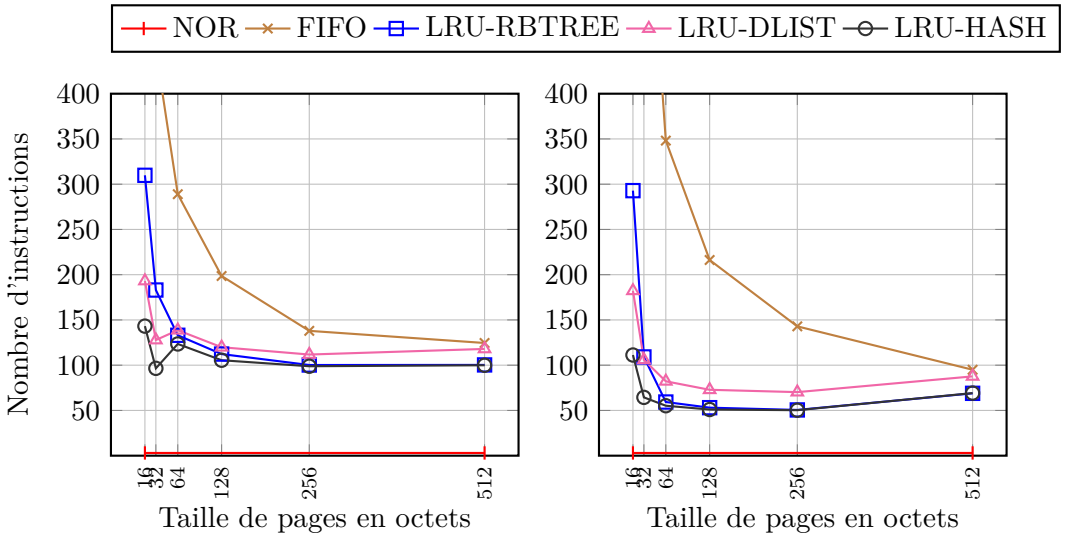
TABLE 2.4 – Surcoût mémoire induit par les différents algorithmes, exprimés en pourcentage de la taille du cache.

Le Tableau 2.5 permet de mettre en perspective les différences des algorithmes à coût mémoire constant. Cette fois, la taille de cache inclue les données gérées *et* les structures de données nécessaires à leur gestion.

Algorithmes	Cache	Taille des pages				
		32	64	128	256	512
Aucun	1024	32	16	8	4	2
	2048	64	32	16	8	4
	4096	128	64	32	16	8
Liste chaînée	1024	23	13	7	3	1
	2048	46	26	14	7	3
	4096	92	53	29	15	7
Arbre rouge-noir	1024	18	11	6	3	1
	2048	36	23	13	7	3
	4096	72	46	26	14	7
Table de hachage	1024	18	10	5	3	1
	2048	42	24	13	6	3
	4096	88	51	27	14	7

TABLE 2.5 – Nombre de pages disponibles en fonction de leur taille et de l'algorithme de gestion utilisé.

L'étude complète discute des aspects mémoire et coût en instructions de façon séparée puis observe l'unification de tous les paramètres. La figure 2.5 présente le résultat



(a) KVM (b) Richards V7 C++

FIGURE 2.5 – Coût en instructions moyen pour l'utilisation d'une donnée au travers du cache.

final le plus intéressant de notre étude pour la réflexion autour de l'exécution de code depuis une mémoire flash NAND.

Cette figure présente le coût moyen, en nombre d'instructions processeur, d'utilisation d'une donnée située en flash NAND en passant par un cache purement logiciel. Les traces sont générées sur l'exécution de la machine virtuelle KVM et sur le jeu de test Richards dans sa version la plus évoluée implémentée en C++. Les mêmes résultats sont présenté figure 2.6 sur l'exécution par la machine virtuelle JavaCard de bytecode situé en mémoire flash NAND mise à notre disposition par Gemalto, ce qui correspond à notre cible privilégiée. Le cache est ici de 1024 octets, ce qui est plus raisonnable pour une cible qui ne dispose que de 8 Ko de RAM.

Ici, tous les aspects sont pris en compte : le coût de l'interface du cache, de l'algorithme de recherche, du mécanisme de renouvellement de pages et les latences du contrôleur réel de la flash NAND. De plus, les 4096 octets attribués au cache incluent le surcoût mémoire imposé par les algorithmes de recherche présenté précédemment. Les deux algorithmes de renouvellement implémentés sont FIFO et LRU (avec ses trois variantes basées sur les trois algorithmes de recherche des pages).

L'information importante que l'on tire de ces résultats est qu'il faut en moyenne 50 instructions pour utiliser une donnée en utilisant le mécanisme de cache. La figure 2.7 nous montre que ce coût est dû à l'interface d'accès au cache et pas à la latence de la mémoire flash NAND. En effet, on observe que les algorithmes de gestion du cache fonctionnent bien et que, sur des pages de 64 à 256 octets, seuls 10 à 15% des accès se soldent par un défaut de cache.

Ces résultats nous permettent de conclure deux choses :

1. si la mémoire flash NAND est utilisée pour stocker du code natif, il s'exécutera

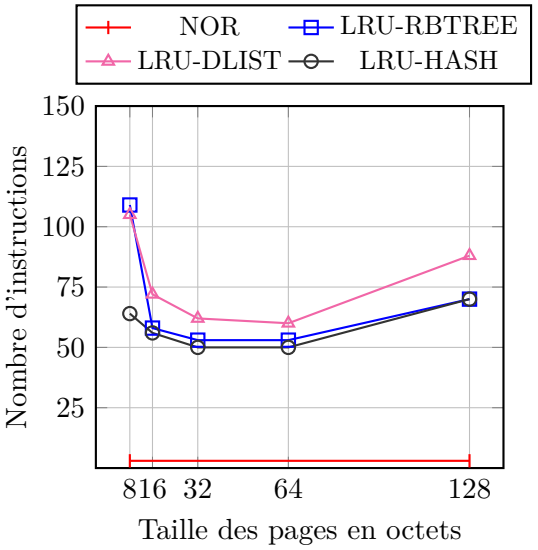


FIGURE 2.6 – Coût en instructions moyen pour l'utilisation d'une donnée au travers du cache sur une JavaCard.

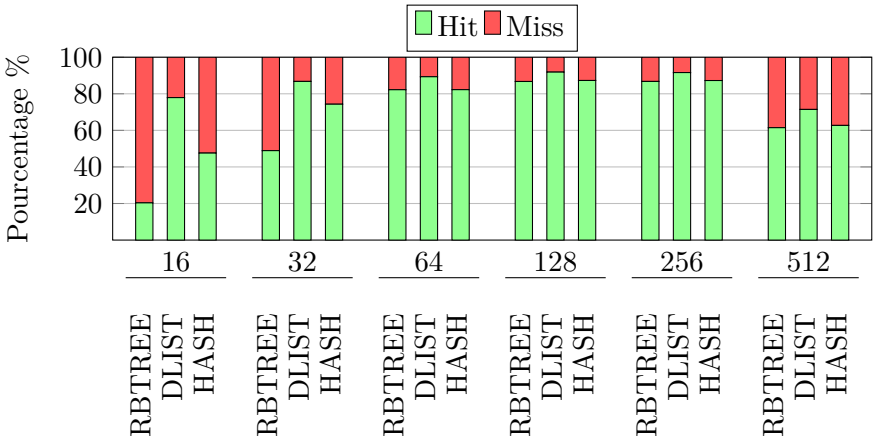


FIGURE 2.7 – Taux de défauts de cache.

au mieux 50 fois plus lentement que s'il était stocké en flash NOR ;

2. on ne peut espérer améliorer la situation en travaillant sur la diminution des défauts de cache.

Considérant notre cible, le premier point n'est pas forcement critique. En effet, la principale technologie utilisée sur les cartes à puce à l'heure actuelle est JavaCard. Les applications ne sont donc pas du code natif, mais du bytecode, interprété par une machine virtuelle. Si la flash NOR est réservée pour stocker le code binaire de la machine virtuelle et la flash NAND utilisée pour stocker le bytecode et les métadonnées liées à ce dernier, le facteur de ralentissement entre traiter un bytecode et le lire via le cache ne sera plus de 50 comme dans le cas du code natif.

Le deuxième point nous permet de conclure que le ralentissement lié au cache est directement causé par la nécessité d'utiliser son interface. En effet, sans support matériel, chaque accès au cache doit passer par un appel de fonction (ou équivalent) qui va vérifier si la donnée est présente dans le cache, la charger si ce n'est pas le cas et la retourner. En pratique, dans le cas d'une machine virtuelle, chaque chargement de bytecode et des métadonnées associées se fait via un appel de fonction à l'interface du cache, même si le bytecode souhaité est déjà dans le cache. Si on veut optimiser l'exécution de la machine virtuelle, il faudra éviter au maximum ces appels.

Cependant, dans le cas de code interprété, ce n'est pas uniquement le code (ou plus précisément le bytecode) qu'il faudra mettre en cache, mais également les métadonnées associées (informations sur les classes, les tables méthodes virtuelles, etc.). Dans le même esprit que l'étude du comportement d'accès à du code natif, nous avons mené une étude poussée sur les particularités de ces méta-données dans le cas de Java et JavaCard [C9]. L'ensemble des leçons de ces études nous a permis de proposer un mécanisme de pré-interprétation du bytecode JavaCard. Cette pré-interprétation permet (1) de s'affranchir de l'utilisation systématique de l'interface logicielle du cache et (2) d'anticiper les défauts de cache afin de profiter de l'asynchronisme offert par le contrôleur de la flash NAND.

2.3 Exécution de bytecode par pré-interprétation depuis une mémoire non-adressable

2.3.1 Interprétation basique

Pour que l'utilisation du cache ne pénalise pas trop l'exécution, nous devons limiter les appels à son interface. La figure 2.8 décrit une implémentation basique d'un interpréteur de bytecode.

```
addr_t pc;

while (1)
{
  bytecode bc = cache_get(pc); /* Récupération du bytecode */
  switch (bc)                  /* Décodage */
  {
    case ADD:
      do_add();                /* Exécution */
      pc++;                    /* Bytecode suivant */
      break;
      /* ... */
  }
}
```

FIGURE 2.8 – Implémentation basique d'un interpréteur avec accès au cache.

On voit que la boucle d'interprétation doit faire appel à l'interface du cache pour

récupérer le prochain bytecode à exécuter. En plus de cet appel, les fonctions qui implémentent chaque bytecode devront également faire appel au cache pour récupérer les opérandes et les métadonnées associées au bytecode comme, par exemple, les propriétés des classes manipulées. Cette interprétation est schématisée par la figure 2.9. La récupération des métadonnées dans le cache lors des phases d'exécution n'est pas représentée pour des raisons de clarté du schéma mais est bien présente à l'interprétation.

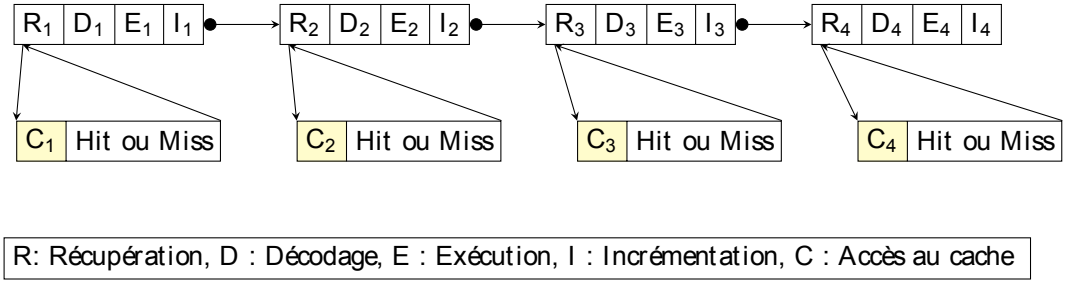


FIGURE 2.9 – Interprétation classique avec accès systématique au cache.

On peut voir ici que chaque récupération de bytecode nécessite un appel au cache. Or, comme le cache fonctionne par page, il est fort probable que si un bytecode est dans le cache, le suivant, et tout ou partie du bloc de base [5] qui le contient, y sera également. La figure 2.10 confirme cette hypothèse. La majorité des blocs de base sont d'une taille inférieure à 32 octets et vont donc pouvoir être logés en quasi intégralité dans une page.

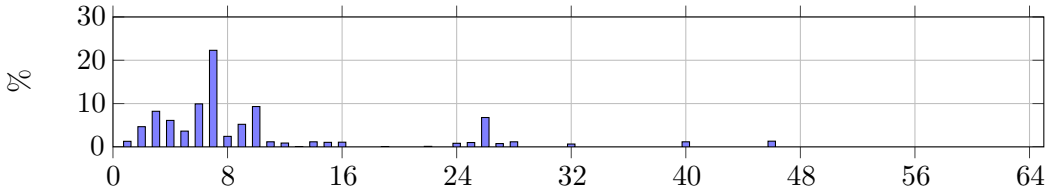


FIGURE 2.10 – Distribution des tailles de blocs de base en octets sur un exemple d'application JavaCard.

Dans ces conditions, l'interpréteur devrait pouvoir lire les bytecodes d'un bloc de base directement depuis l'espace mémoire du cache sans passer par son interface s'il a la garantie que les bytecodes s'y trouvent et que les accès aux métadonnées nécessaire à l'exécution de ces bytecodes peuvent être résolus sans défaut de cache. Pour cela, il a besoin, avant d'interpréter une succession de bytecodes, de savoir combien il peut en interpréter sans devoir refaire un accès au cache via son interface.

2.3.2 Analyse et pré-décodage d'un bloc de base

Pour déterminer l'ensemble des bytecodes pouvant être exécutés en place dans le cache, nous avons mis au point une analyse de code dynamique qui est exécutée avant

5. Ici, on considère un bloc de base comme une suite de bytecodes ne contenant pas de rupteur de flot de contrôle et donc s'exécutant de façon séquentielle.

l'exécution d'un bloc. Les bytecodes du bloc sont pré-décodés pour vérifier s'ils peuvent être exécutés dans le cache. Les conditions d'arrêt de cette analyse sont :

1. un appel ou un retour de méthode ;
2. un saut conditionnel ou inconditionnel ;
3. un dépassement de page ;
4. l'accès à une méta-données.

L'analyse permet alors d'assurer à l'interpréteur qu'il pourra exécuter les bytecodes, sans faire appel à l'interface du cache, jusqu'au prochain qui nécessitera une nouvelle analyse. De plus, l'analyse permet également d'anticiper les défauts de cache. Si le matériel dispose d'un contrôleur de flash pouvant charger des données de façon asynchrone, l'analyseur peut déclencher le chargement des pages manquantes et poursuivre l'interprétation effective des bytecodes pendant que la page demandée sera chargé depuis la flash.

Pour éviter de payer deux fois le coup de décodage des bytecodes, l'interpréteur tire parti du décodage partiel effectué par l'analyseur. Le nouveau schéma d'interprétation devient alors celui de la figure 2.11. Les étapes D_1 à D_4 correspondent à l'analyse et E_1 à E_4 à l'interprétation effective des bytecodes. Toutes ces étapes sont réalisées sans faire appel à l'interface du cache et donc sans payer le coût lié à la recherche d'une donnée. De plus, le pré-chargement de la page qui résoudra le prochain défaut de cache peut être déclenché en recouvrement de l'interprétation concrète des bytecodes.

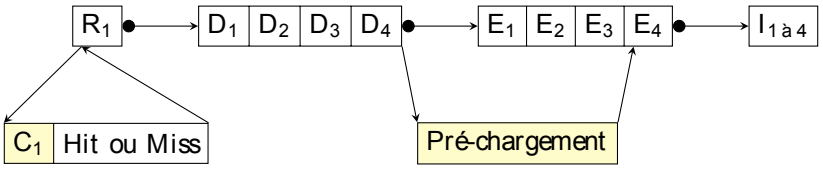


FIGURE 2.11 – Interprétation avec analyse.

2.3.3 Résultats expérimentaux

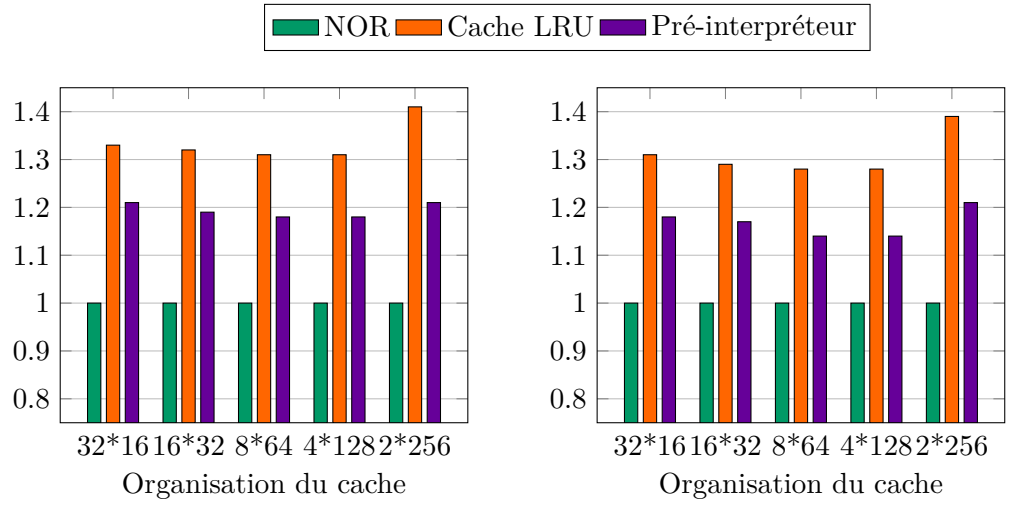
Une preuve de concept a été réalisée sur une plateforme carte à puce réelle. Le matériel ainsi que le code de la machine virtuelle JavaCard ont été fournis par Gemalto. Cette machine virtuelle, ainsi que l'API JavaCard utilisée, est conforme aux spécifications [48] et [49].

La carte à notre disposition est orchestrée par un processeur ARM7 32bits cadencé à 17,5 MHz. Ce processeur dispose de 32 Ko de ROM, 48 Ko de RAM et 768 Ko de flash NOR adressable. Le microcontrôleur propose une interface d'accès à de la mémoire flash NAND sur un bus série. Les propriétés de cette flash NAND sont celles que nous avons donné Table 2.2 page 16 avec des pages de 2048 octets. Malheureusement, le matériel dont nous disposions ne permettait pas le pré-chargement asynchrone. Les gains observés sont donc uniquement ceux apportés par le fait que la pré-interprétation permet d'éviter un accès systématique à l'interface du cache. Les mesures que nous avons

effectuées par simulation montre que le pré-chargement permettrait de recouvrir entre 5 et 50% du coût en instructions de la pré-interprétation en fonction des applications que nous avons étudiées.

Les tests réels sont réalisés, entre autres, à partir de notre application JCProfil. Pour chaque mesure, l'application est chargée, vérifiée et installée. L'application est ensuite lancée par une commande APDU [6] envoyée à la carte. La prise de mesure commence à cet instant et se termine par un point d'arrêt dans un débogueur matériel. Le temps d'exécution final est obtenu avec une précision à la centaine de milli-secondes près.

Notre approche est alors comparée au temps d'exécution de l'application située en flash NOR et située en flash NAND avec une interprétation basique effectuant un accès systématique à l'interface d'un cache LRU.



(a) Cache de 512 octets.

(b) Cache de 1024 octets.

FIGURE 2.12 – Comparaison entre la NOR, un cache LRU à accès systématique et notre pré-interpréteur sur une carte à puce à 17,5 MHz.

La figure 2.12 illustre le facteur de ralentissement qu'impose notre technique vis-à-vis d'un stockage de l'application en flash NOR. Le temps d'exécution de l'application stockée en NOR est normalisé à 1. On y voit que la pré-interprétation offre un apport significatif par rapport à l'exécution d'un cache LRU avec accès systématique, et surtout, on y voit que le temps d'exécution de l'application stockée sur la mémoire série n'est « que » 20% plus lent ce qui, pour la plupart des transactions JavaCard reste quasi-transparent pour l'utilisateur de la carte.

La norme ISO-7816-3 impose cependant une norme qui pourrait rendre non-transparent un ralentissement trop fort. Cette norme stipule qu'une durée normale de transaction est située entre 3 et 5 secondes. Si une telle transaction s'exécute en 4 secondes avec l'application stockée en flash NOR, elle durera alors 5,2 secondes en utilisant un cache

6. Application Protocol Data Unit : format d'échange pour le dialogue avec les cartes à puce défini dans la norme ISO-7816-4

simple, ce qui met en défaut la norme. Avec notre approche, cette transaction s'exécuterait en 4,55 secondes.

Cet écart pourrait encore être réduit si le contrôleur de flash permettait d'utiliser le pré-chargement que peut déclencher notre pré-interprétation.

2.4 Conclusion

Ce chapitre présente la technique de pré-interprétation qui nous a permis de rendre possible l'exécution d'applications dont le code est stockée dans une mémoire non-adressable. Ce chapitre présente également le cheminement qui a été suivi pour parvenir à ce résultat.

La force du travail que j'ai mené avec Geoffroy aura justement étée de pousser l'analyse des mémoires cache logicielles pour être en mesure de mettre en lumière les bonnes pistes de recherche. L'idée que j'avais à l'origine, selon laquelle la politique de remplacement de page ou une réorganisation du code seraient probablement les bon leviers pour rendre possible l'exécution, a été complètement balayée par notre étude. Les analyses pratiquées par Geoffroy, dont ce document ne présente qu'une infime partie, nous ont non seulement permis de trouver les bons paramètres et les bonnes tailles de cache à utiliser, mais également de voir que le manque de performance du système n'était pas causé par les défauts de cache, mais par l'utilisation de l'interface de celui-ci. C'est à partir de ce constat, éclairés par des résultats qui n'étaient pas nécessairement intuitifs, que notre idée de la pré-interprétation a vu le jour.

Enfin, la validation sur une plateforme réelle, industrielle et le transfert de la technologie à Gemalto aura également contribué à rendre le travail de Geoffroy particulièrement intéressant et gratifiant.

La solution proposée permet d'envisager le développement sur des plateformes embarquées tout en bénéficiant de langages standards (à la condition qu'ils soient interprétés) et surtout en s'affranchissant de la limite de faible quantité de mémoire adressable disponible. Ce ne doit évidemment pas empêcher les développeurs de prendre un minimum de soin pour optimisation de leur logiciel, mais l'élévation de la barrière de taille de code permet d'augmenter la taille des API standards et donc d'offrir plus de fonctionnalités « prêtes à réutilisation » pour les développeurs.

Il resterait néanmoins à poursuivre cette étude en prenant en compte les impératifs de sécurité liés au monde de la carte. Notre solution, si elle est efficace, augmente la surface d'attaque possible d'une carte. En particulier, le comportement du cache peut très certainement être analysé via des attaques physiques (analyse de consommation, de rayonnement...) et donner des informations sur le code exécuté.

Cette thèse fut la première que j'ai co-encadrée et j'ai conservé lors de l'encadrement des thèses suivantes cette volonté d'analyser le système étudié pour guider et orienter les pistes de recherche.

La thèse dont les résultats sont présentés dans ce chapitre a été soutenue le 13 décembre 2012. Elle a conduit à plusieurs publications [C9, C10, C11] et à un transfert à la société Gemalto. Geoffroy est maintenant ingénieur de recherche aux « Gemalto Research & Innovation Labs ».

Chapitre 3

Production de logiciels embarqués grâce à un langage dédié

Les travaux présentés dans ce chapitre ont été initiés suite à plusieurs discussions avec des membres de la société Stormshield [1] travaillant sur le noyau de filtrage de leurs pare-feux. Nous discutions alors de la difficulté de gérer la sécurité des applications web hébergées sur des plateforme de virtualisation telle que celles fournies par Amazon ou, plus localement, OVH. Il nous a vite paru évident que le faible coût de location d'une telle puissance de calcul et d'une telle quantité de bande passante pouvait être utilisée en remplacement, ou du moins en complément, des botnets pour réaliser des attaques distribuées de types déni de services ou propagation de vers.

Fort d'une précédente expérience [2] de localisation physique de serveur au sein d'un data center à l'aide de petits équipements embarqués, nous savions qu'un gestionnaire de datacenter était prêt à dépenser une somme supplémentaire (bien que modique) par serveur physique s'il y voyait un intérêt pratique. Pour la localisation, cette somme correspondait à un capteur embarqué équipé d'une interface de communication sans fil collé sur la façade du serveur et « détectant » sa position à l'aide de LED infra-rouges.

Nous avons alors pensé que le même principe, de petits équipements adossés à chaque serveur physique, pouvait être appliqué à la sécurité si ces équipements étaient installés sur le lien ethernet du serveur, surveillant ainsi ses communications avec l'extérieur. Si tous les serveurs étaient équipés de tels équipements, ceux ci pourraient alors collaborer pour corréler les différents trafics et détecter des attaques non détectables par un pare-feu situé sur le lien central du data center.

Restait alors la question du développement du logiciel de ces sondes. Comment permettre à un responsable de la sécurité de data center, dont la spécialité est la mise en place de politique de sécurité mais pas le développement de logiciel embarqué, de définir sa politique de sécurité et de la déployer efficacement sur un parc de sondes ?

1. Netasq à l'époque de nos discussions
2. dans un projet collaboratif IPER (Innovation dans les Processus d'Entreprises par la RFID) du ministère de l'industrie.

Ce chapitre présente les travaux de la thèse de Damien Riquet que je co-encadre avec Gilles Grimaud. Cette thèse, financée par une bourse ministérielle, sera soutenue au cours de l'année universitaire 2014-2015 et consiste principalement en la définition de l'architecture de notre système de sondes ainsi que d'un langage dédié et de sa suite d'outils permettant le développement de logiciel de sécurité à destination d'un parc de sondes hétérogènes.

3.1 Motivations

Depuis quelques années, le terme « Cloud » fait son entrée dans le vocabulaire courant. Il n'est maintenant pas rare de le voir employé dans les séries télévisés, les médias généraux et toute autre communication à destination du grand public.

L'aspect éthéré qu'inspire le terme aux yeux du plus grand nombre montre le succès de ce type d'informatique qui se fait oublier. Nos données, parmi les choses les plus importantes de notre vie, sont « dans le cloud ». La plupart des gens ne savent ni vraiment où, ni par quelle magie la photo du petit dernier peut être à la fois sur notre ordinateur, notre tablette, notre téléphone ainsi que sur tous les écrans de nos contacts. Peu importe, ces photos sont « en sécurité dans le cloud ».

Néanmoins, la mise en place de ces services et de ce « cloud » est au contraire très concrète. Derrière l'apparente magie de Google, Facebook ou Dropbox, se cachent des milliers, des millions de serveurs bien réels, installés bien au chaud dans les centres de données [3], avec leurs processeurs, leurs cartes réseau, leurs systèmes d'exploitation et leurs logiciels mettant en œuvre tout ces services. Bien que peu de gens y accordent une importance, nos données ne sont pas simplement magiquement « dans le cloud » mais bien présentes physiquement sur des centaines de disques durs dont le contenu est, la plupart du temps, visible non seulement par le gérant du service fourni mais également par quiconque réussirait à pénétrer et à compromettre ces serveurs. Or, la réputation de sécurité des services cloud n'est pas toujours au beau fixe, ce qui a été encore une fois montré par la très récente affaire de publication de photos privées « volées » sur la plateforme iCloud d'Apple [5].

Le respect de la vie privée des utilisateurs est donc sujette d'une part au contrat liant le fournisseur de service à l'utilisateur, mais également au fait que le système qui héberge le service soit sûr. Si le premier cas relève plutôt d'aspects juridiques et de la confiance que l'utilisateur veut bien accorder au fournisseur, le problème de la sécurité des données est lui complètement technique et primordial pour la sauvegarde de cette vie privée.

3.1.1 Sécurité des services cloud

La sécurité d'un système d'information peut être compromise par un large éventail de nuisances. Pour la compromission directe, on peut citer les virus [80], les vers [52] ou les chevaux de troie et autres « *back doors* » [1]. Ces compromissions visent généralement

3. J'utiliserai plutôt dans la suite du document le terme *datacenter*, moins fidèle à la langue française mais plus courant dans le domaine qui nous occupe.

la prise de contrôle des machines (pour l'établissement d'un botnet [26]) ou le vol de données privées. La plupart de ces attaques sont déployées grâce à une bévue de l'utilisateur ou à une faille du système d'exploitation ou d'un logiciel du système. Le plus souvent la faille est exploitée via le réseau, par exemple par injection SQL, XSS [27] ou l'exploitation d'un bug comme, plus récemment, la faille Heartbleed [2] sur openssl.

Mais le réseau n'est pas uniquement un vecteur de logiciels malveillants. Il est également une « arme » à lui tout seul. Quand on pense aux attaques impliquant directement le réseau, on pense aux attaques par déni de service, soit directe, soit par amplification (par exemple en utilisant le protocole DNS [22]). Mais il faut également citer les détections de port [25] qui permettent de rechercher les services susceptibles d'être exploités, les usurpations d'adresse (IP, MAC...), de session (par vol de cookie sur un réseau WiFi ouvert par exemple), les attaques par relais, le spam ainsi qu'un large éventail d'actions plus nuisibles les unes que les autres [4].

La sécurité des services cloud n'est pas épargnée par ces différentes actions malveillantes, mais l'impact de ces dernières peut s'en trouver amplifié et la mise en œuvre de contre-mesures bien plus délicate.

Tout d'abord, la quantité de données sensibles qu'il est possible de se procurer en compromettant un service cloud très utilisé est plus importante que dans le cas d'un service auto-hébergé par une entreprise. Dans ce cas, les données auxquelles on peut accéder sont uniquement celle de l'entreprise en question. Dans le cas d'un service cloud partagé, c'est un éventail très large de données utilisateurs, de secrets industriels ou d'états qui peuvent êtres compromises.

Deuxièmement, si dans le cas d'un service auto-hébergé, l'entreprise qui le déploie et l'utilise peut restreindre fortement les accès au service (à son réseau intranet par exemple) et donc limiter les points d'entrée possibles, cela est plus difficile dans le cas d'un service cloud partagé. Les possibilités d'attaques sont donc bien plus importantes dans ce cas.

Enfin, la complexité de l'architecture qui met en œuvre les services rend plus difficile sa surveillance et sa sécurisation et offre de nouvelles possibilités aux attaquants.

En particulier, la location de serveurs ou de machines virtuelles à bas coût permet de remplacer assez efficacement un réseau de botnets, difficile à maintenir, pour une somme modique. Si l'on veut effectuer une attaque de grande envergure en déni de service, il est facile de louer suffisamment de puissance de calcul et de bande passante pour le faire. Ceci a été démontré par une preuve de concept à la conférence DEF CON de 2010 [14]. Les orateurs ont montré qu'avec seulement quelques dollars, il était possible de mener des attaques en déni de service en louant des machines virtuelles sur la plateforme Amazon EC2.

Même s'il peut être possible pour un hébergeur de service cloud de détecter que leur propre matériel mène une attaque vers l'extérieur, qu'en est-il si l'on loue des machines chez un fournisseur pour attaquer un service hébergé chez ce même fournisseur, voire dans le même datacenter ? La figure 3.1 résume les trois directions d'attaques que l'on peut retrouver dans une configuration de type cloud : utilisation des serveurs pour

4. de façon non exhaustive, on peut ajouter, les attaques « *man in the middle* », redirection DNS frauduleuse ou le phising...

attaquer l'extérieur (en bleu uni), l'attaque d'un serveur du datacenter par l'extérieur (en rouge rayé horizontalement) et l'attaque de serveurs du datacenter par d'autre serveurs de ce même datacenter (en vers rayé verticalement). Dans ce cas, la protection est à l'heure actuelle quasi inexistante.

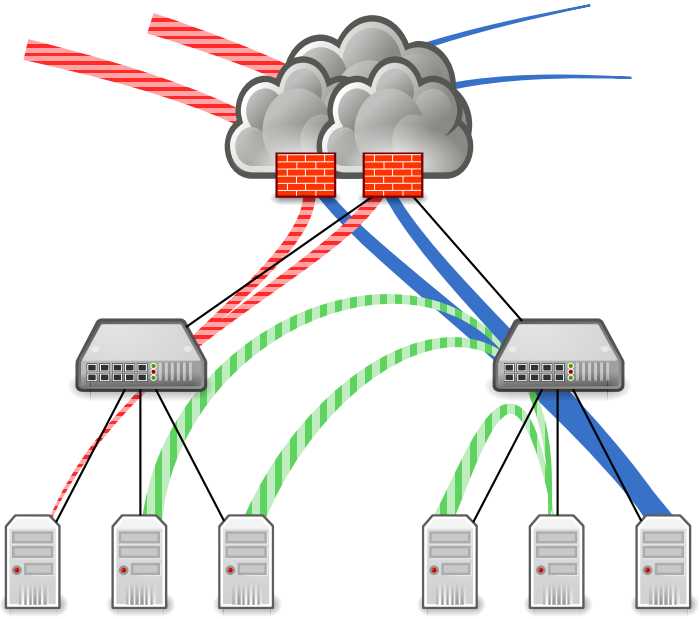


FIGURE 3.1 – Les différentes directions d'attaques dans une architecture de type cloud.

En effet, les protections sont en général assurées par des pare-feux placés en coupure des accès principaux d'un datacenter à Internet et ne peut donc surveiller le trafic émis entre deux machines d'un même centre. Les pare-feux sont généralement constitués de deux composants, le système de filtrage (qui permet de laisser passer ou non des paquets) et le système de détection d'intrusion, qui nous intéresse dans ce chapitre.

3.1.2 Protection des serveurs : les systèmes de détection d'intrusion

Le rôle principal d'un système de détection d'intrusions (IDS) [6, 8, 20, 69, 81] est de surveiller un lien réseau afin d'y déceler des anomalies pouvant être considérées comme dangereuses. Ces anomalies peuvent consister en une certaine séquence d'octets dans un paquet ou dans un flux, ou dans un comportement anormal dans la fréquence d'émission des paquets ou dans le non respect des protocoles standard. Une fois un tel motif détecté, l'IDS lève une alerte qui pourra ensuite être traitée par le responsable de la sécurité du centre.

Les performances d'un IDS seront évaluées selon sa capacité à faire la distinction entre du trafic normal et du trafic anormal. Il faudra alors, pour comparer des IDS, établir les taux de vrais positifs (une alerte est levée pour une anomalie réelle), de faux positifs (une alerte est levée mais le trafic est normal), de vrais négatifs (aucune alerte n'est levée et le trafic est normal) et de faux négatifs (aucune alerte n'est levée mais le

trafic est anormal).

Pour détecter les attaques, deux techniques principales sont utilisées par les IDS : la détection de signature et l'analyse de comportement. Dans le premier cas, l'IDS effectue une recherche de données particulières, de motifs connus à l'avance. Ici, il s'agit de détecter des tentatives d'exploitation de faille connues comme par exemple des injections SQL. Une base de données des vulnérabilités connues ainsi que de la façon de les exploiter est alors utilisée par l'IDS qui compare le trafic qu'il surveille avec la base de donnée. S'il détecte un motif présent dans la base de données, il lève l'alerte.

Dans le second cas, on cherche à modéliser le comportement normal du système et à détecter quand celui-ci dévie de ce qui est attendu d'un système sain. L'avantage d'un tel système est indéniablement qu'il est *a priori* capable de détecter des attaques sur des vulnérabilités non connues au préalable (failles *zero day*). Cependant, ce type d'IDS, le plus souvent basé sur de l'apprentissage, des systèmes experts ou des approches statistiques, est encore peu utilisé en production réelle car il est difficile à mettre en place et peut avoir un taux de faux-positifs plus important [10].

Un IDS est donc un composant essentiel si l'on veut apporter de la sécurité à un centre de données. Les enjeux pour la création d'IDS efficaces sont donc important, d'autant que, comme le signalait une étude de 2010 [16], une part importante[5] des petites et moyennes entreprises ne désirent pas passer au cloud computing pour des raisons de sécurité. Des chercheurs [18], des industriels [21] ou des organisations gouvernementales [19] partagent la même inquiétude.

Mais, comme nous venons de le voir, le domaine de la détection d'intrusions est encore très actif tout comme l'est la communauté des attaquants qui redouble d'efforts pour trouver de plus en plus de failles et les exploiter. En plus de trouver de nouvelles contre-mesures, il faut donc également que les IDS du marché soient facilement configurables et adaptables pour que les responsables de la sécurité des systèmes d'information puissent les mettre en place. C'est dans ce sens que va l'IDS Snort [72] qui propose une base de règles de détection d'attaques sous forme de scripts facilement partageables par la communauté. Cette aspect de facilité de reconfiguration ne doit donc pas être négligé.

Les IDS tels que nous venons de les décrire sont des éléments qui se placent sur un lien réseau (le plus souvent sur le lien de connexion à Internet). Seulement, comme nous l'avons précisé dans la section précédente, la géométrie des réseaux cibles d'attaques n'est pas aussi simple que les *gentils* d'un côté et les *méchants* de l'autre, séparés par un pare-feu qui coupe le seul point d'accès entre les attaquants et les attaqués. J'ai souhaité observer comment la distribution des attaques affecte les moyens mis en œuvre à l'heure actuelle. J'ai donc proposé à Damien de mener une étude sur le comportement des IDS dans un contexte distribué.

5. 62% d'après l'étude

3.1.3 Comportements des IDS dans un contexte distribué

Afin de comprendre les problèmes liés aux architectures complexes que nous avons décrites, nous avons souhaité observer le comportement des IDS standards face aux attaques distribuées. Ces attaques sont menées en subdivisant une attaque, de manière à ce que les éléments de sécurité ne soient pas capables de détecter les sous-parties de celle-ci. De plus, l'attaquant peut utiliser la structure du Cloud Computing pour être davantage discret. En effet, si un attaquant peut emprunter plusieurs chemins pour s'adresser à une machine, sa discrétion est accrue puisque chaque solution de sécurité ne voit qu'une partie du trafic. Ces études ont été publiées dans [C7, C8], je reprendrai donc uniquement les aspects les plus importants. Le lecteur souhaitant en savoir plus pourra se référer aux articles.

Dans cette étude, nous avons choisi d'observer le comportement de Snort [72] et d'un pare-feu commercial en réponse à une attaque en détection de ports (ou scan de ports). Le but de cette attaque est de fournir à un attaquant l'ensemble des services ouverts sur la ou les machines cibles. En plus de ports TCP ou UDP ouverts, les détections de ports essayent également généralement de déceler les versions des différents services une fois qu'ils en ont détecté la présence. Cette attaque est relativement simple à mettre en œuvre mais également bien détectée par les IDS dans un contexte habituel, ce qui nous permet de mettre en lumière la faiblesse de ces derniers dans un contexte distribué.

En effet, ces expérimentations, réalisées dans des environnements différents (différents nombres d'attaquants, de cibles, d'éléments de sécurité, de type de balayage de ports, etc.) indiquent que les solutions de sécurité réseaux ne sont plus adaptées aux nouveaux types d'attaques. En effet, dans certains cas, seuls 32 attaquants sont suffisants pour mener une attaque en toute discrétion. Ceci est illustré par la figure 3.2. Celle-ci présente le taux de succès d'une attaque en détection de ports exprimé en pourcentage de réussite. Ce taux est calculé comme étant le nombre de ports détectés correctement ouverts sans détection de l'attaque par l'IDS divisé par le nombre qu'il fallait découvrir. Une valeur de 100% correspond donc à une attaque menée complètement à bien.

On peut facilement voir que, pour les deux IDS considérés, augmenter le nombre de machines utilisées pour effectuer la détection de ports améliore significativement le taux de réussite de l'attaque. Ceci s'explique tout simplement car les méthodes de détection de cette attaque, mises en place sur les IDS, ne considèrent en général pas que les attaquants vont collaborer pour obtenir la liste des ports. Chaque machine « attaquante » fait donc une action faible au regard de l'attaque dans sa globalité et n'est donc pas détectée comme malveillante par l'IDS. De nos jours, de telles ressources sont accessibles facilement, grâce à des botnets, après une infection virale de type ver informatique, ou tout simplement en louant des ressources auprès des fournisseurs de cloud.

Pourtant, le cas présenté par cette figure est favorable aux IDS standards. En effet, dans tous les cas présentés, les IDS ont vu passer l'intégralité du trafic de l'attaque et c'est uniquement la multiplication des sources d'attaques qui les ont déjoués.

Dans le cas ou plusieurs chemins sont possibles pour atteindre l'ensemble des machines cibles de l'attaque, la situation empire car les IDS placés sur chacun des chemins

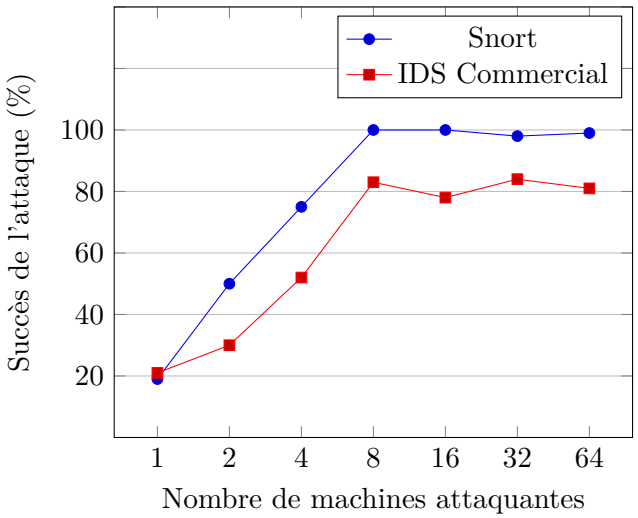
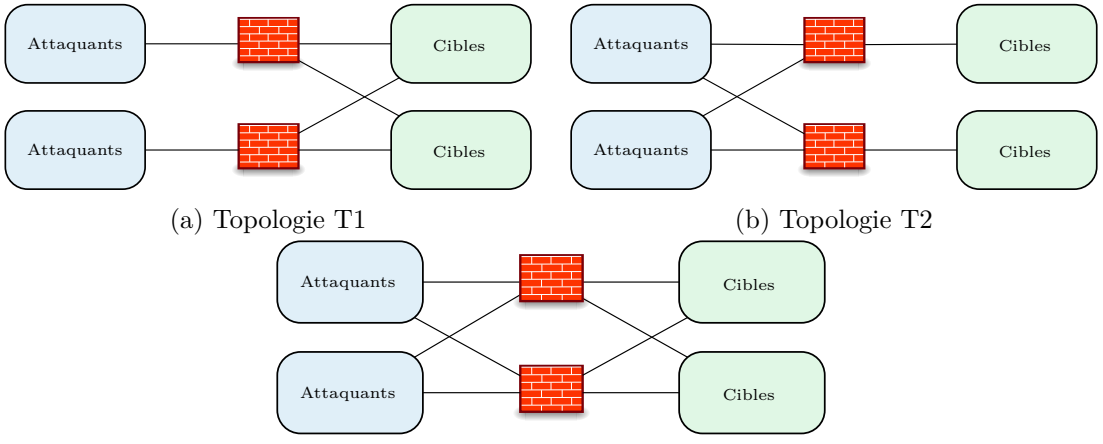


FIGURE 3.2 – Succès de l'attaque en détection de ports sur 4 cibles en fonction du nombre d'attaquants.

ne voient qu'une sous partie de l'attaque. Cette situation est tout à fait envisageable dans le cas de fournisseur d'accès multiples, de configuration de haute-disponibilité, etc.

C'est cette situation que nous avons voulu évaluer dans [C7]. Nous avons effectué à nouveau la même attaque distribuée de détection de ports. Seulement, cette fois, nous avons considéré les topologies illustrées par la figure 3.3.



(a) Topologie T1

(b) Topologie T2

(c) Topologie T3

FIGURE 3.3 – Topologies étudiées dans le cas d'un environement à deux routes.

La topologie T1 correspond à une isolation géographique des attaquants ; ces derniers transitent par un seul point d'entrée pour s'adresser à l'ensemble des cibles, mais des attaquants différents peuvent utiliser des points d'entrée différents. La topologie T2 correspond à l'inverse à une isolation des cibles, plusieurs groupes de ces dernières

étant sous la surveillance d'un seul IDS. Enfin, la topologie T3 correspond à une interconnexion totale, un attaquant peut utiliser plusieurs chemins pour s'attaquer à toutes les cibles. Nous avons effectués nos mesures à l'aide de la plateforme Grid'5000 [6], ce qui nous a permis de mesurer les résultats avec un nombre suffisant d'attaquants, de cibles et de routes. La figure 3.4 montre le gain que peut espérer un attaquant s'il peut bénéficier de multiples routes pour mener son attaque. Les résultats sont pour 64 machines cibles, 32 attaquants et l'IDS Snort. Le taux de succès de l'attaque est mesuré de la même façon que précédemment.

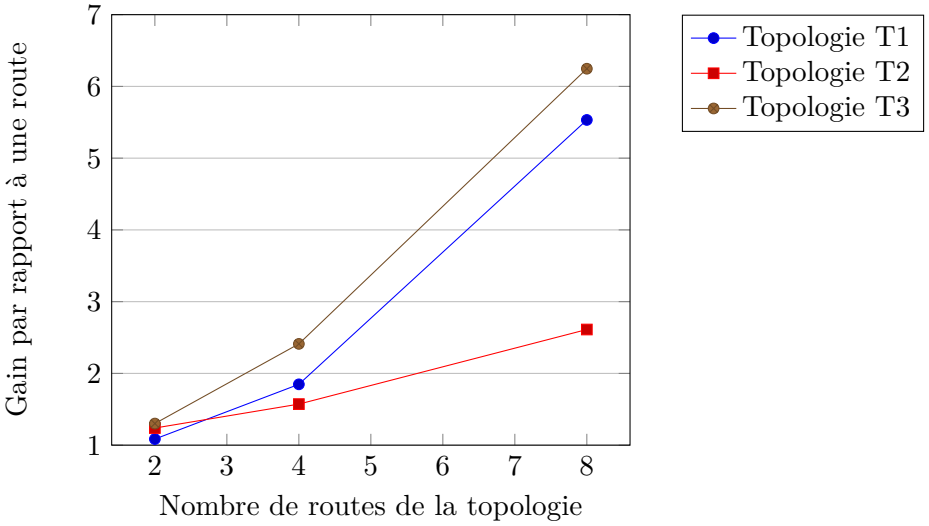


FIGURE 3.4 – Gain obtenu sur l'efficacité de l'attaque par augmentation du nombre de routes.

Comme on peut le voir, complexifier la topologie est particulièrement favorable aux attaquants qui vont bénéficier de cette dernière pour se cacher aux yeux des IDS. En comparant les résultats sur les topologies T1 et T3 avec ceux obtenus sur la topologie T2, on voit à quel point les IDS ont du mal à œuvrer correctement quand ils ne voient pas passer l'intégralité du trafic de l'attaque. En effet, dans le cas de la topologie T2, l'IDS voit passer l'intégralité du trafic vers une cible donnée et va donc réagir presque aussi bien que s'il n'y avait qu'une route.

Il est donc important pour améliorer l'efficacité des IDS de leur permettre de voir passer l'intégralité du trafic, ou tout au moins de collaborer pour qu'ils soient capables de prendre des décisions en rapport avec l'attaque vue dans sa globalité et non par la lorgnette du lien réseau surveillé par un seul IDS. C'est sur ce constat que nous avons élaboré notre solution, DISCUS.

6. `http://www.grid5000.fr`

3.2 DISCUS

Notre solution, DISCUS, a été imaginée à partir du constat qu'il fallait pouvoir inspecter le trafic réseau au plus près des machines afin d'être capable d'obtenir un maximum d'informations susceptibles de permettre une détection d'attaque. De part notre culture des systèmes embarqués, nous avons imaginé une solution dans laquelle chaque serveur physique se voit attribuer une sonde matérielle embarquée, installée directement en sortie de la carte ethernet qui pourrait alors observer le trafic reçu et généré par la machine. Ainsi, ce serait donc une multitude d'IDS, et non plus un seul, qui pourraient collaborer afin de détecter une attaque.

3.2.1 Systèmes de détection d'intrusions collaboratifs

Les systèmes de détection d'intrusions collaboratifs ou CIDS [17], sont une évolution des IDS locaux. L'idée principale est de faire collaborer des IDS situés sur différents domaines réseau et de corréler les données qu'ils collectent pour faire la détection d'attaques dont les sources ou les destinations sont situées sur plusieurs réseaux. Afin de collaborer, un système CIDS va devoir partager de l'information entre les IDS.

Ce partage peut être fait de manière centrale ; tous les IDS envoient l'information à un système central qui analyse ces dernières et prend une décision en ayant une connaissance globale de l'état du système. On peut citer par exemple DIDS [79].

L'architecture peut également être hiérarchique. Les différents IDS sont alors répartis en groupes dont un membre est choisi pour effectuer la corrélation des données du groupe. Ce membre fait également partie d'un groupe de niveau supérieur dont un membre analyse et agrège les données reçues par les sous-groupes et ainsi de suite. Les groupes sont formés en fonction de la géographie (et donc de la qualité du lien de communication), des entités qui administrent les réseaux impliquant les IDS ou d'autres critères. Ce type d'organisation permet un passage à l'échelle ainsi qu'une meilleure robustesse du système puisqu'il ne dépend plus d'un seul chef d'orchestre qui prend toutes les décisions. À titre d'exemple, on peut citer GrIDS.[76].

Enfin, le système peut être organisé de façon totalement distribuée où chaque membre du CIDS fait partie du système de corrélation de données. Ces systèmes utilisent généralement des algorithmes de type pair à pair comme des tables de hachages distribuées [60, 46, 62] pour partager l'information à tous les membres. Par exemple, dans [38], chaque IDS stocke dans la table distribuée les adresses qu'il considère suspicieuses et chaque membre du CIDS peut donc mettre en relation cette liste avec le trafic qu'il observe pour obtenir plus d'informations sur une éventuelle attaque.

3.2.2 Mise en œuvre d'un CIDS hétérogène

Dans un projet précédent, j'ai déjà été amené à proposer un système embarqué sur chaque serveur pour permettre un inventaire et une localisation précise des serveurs au sein du datacenter. L'objet final, qui a maintenant évolué et est devenu une solution industrielle proposée par un partenaire du projet, la société Noolitic, est un petit capteur embarqué, communicant par radio ZigBee pour indiquer la position du serveur sur

lequel le capteur est collé (au sens propre, sur la façade du serveur). La position est obtenue grâce à un système de diodes infra-rouges. Le partenaire qui représentait la partie hébergement, datacenter et donc client d'une telle technologie, nous indiquait alors qu'un tel module pourrait coûter jusqu'à une cinquantaine d'euros pour que son installation à grande échelle puisse être envisagée.

Cette ordre de grandeur du coût d'un système de capteurs embarqués accolés à chaque serveur nous a donc orienté dans le choix de l'architecture DISCUS. En effet, à ce prix, il était inimaginable de construire un matériel embarqué basé sur un processeur généraliste, capable de traiter du trafic réseau sur une interface gigabit. Nous avons donc imaginé que nos sondes pourraient être basées sur la technologie FPGA [7] [75] qui permet de « fabriquer » un circuit matériel dédié à une tâche. De plus, contrairement aux circuits ASIC [8], les FPGA sont reprogrammables et peuvent donc être mis à jour. Cette fonctionnalité est bien entendu indispensable dans notre cas d'application car de nouvelles attaques, et donc de nouvelles contre mesures, sont mises au point très régulièrement.

Cependant, installer des sondes embarquées ne nous affranchit pas de chercher à bénéficier des équipements déjà existants. En effet, les gros pare-feux de cœur de réseaux sont toujours à même de fournir des informations pertinentes. De même, même si une sonde embarquée peut être installée sur un serveur physique, ce serveur physique héberge généralement plusieurs serveurs virtuels. Il semblerait intéressant de pouvoir mettre une sonde, cette fois virtuelle, sur chaque serveur virtuel. L'architecture complète de DISCUS se présente donc comme indiqué figure 3.5. Les éléments constitutifs de DISCUS sont les sondes embarquées, les pare-feux ainsi que les sondes logicielles qui pourront être soit directement implémentées au sein de l'hyperviseur ou dans chaque système invité.

Nous sommes donc confrontés à au moins trois cibles distinctes : les sondes embarquées, les IDS de cœur et les hyperviseurs. Développer du logiciel pour ces trois cibles demande trois expertises différentes qui n'est généralement pas celle d'un administrateur système.

Une solution permettant de gérer l'hétérogénéité des cibles ainsi que l'adaptation du système à un utilisateur ne possédant pas particulièrement de connaissance en développement est l'utilisation d'un langage dédié.

Les langages dédiés, ou *DSL* [9], sont une alternative à l'utilisation de langage générique comme le C ou le Java. Comme leur nom l'indique, les langages dédiés sont conçus pour s'adapter à une situation particulière. Leur expressivité offre une abstraction de haut niveau, adapté à un domaine d'application. Ceci leur permet :

— d'exprimer des problématiques spécifiques et donc d'être facilement utilisés par des spécialistes du domaine d'application, qui ne sont pas nécessairement à l'aise avec des langages génériques, ni avec les spécificités des cibles visées ;
— d'offrir un ensemble de garanties sur la correction du logiciel produit par construction qui permettent de s'affranchir d'erreurs lors de l'exécution.

7. Field Programmable Gate Array
8. Application Specific Integrated Circuit
9. Domain Specific Langage

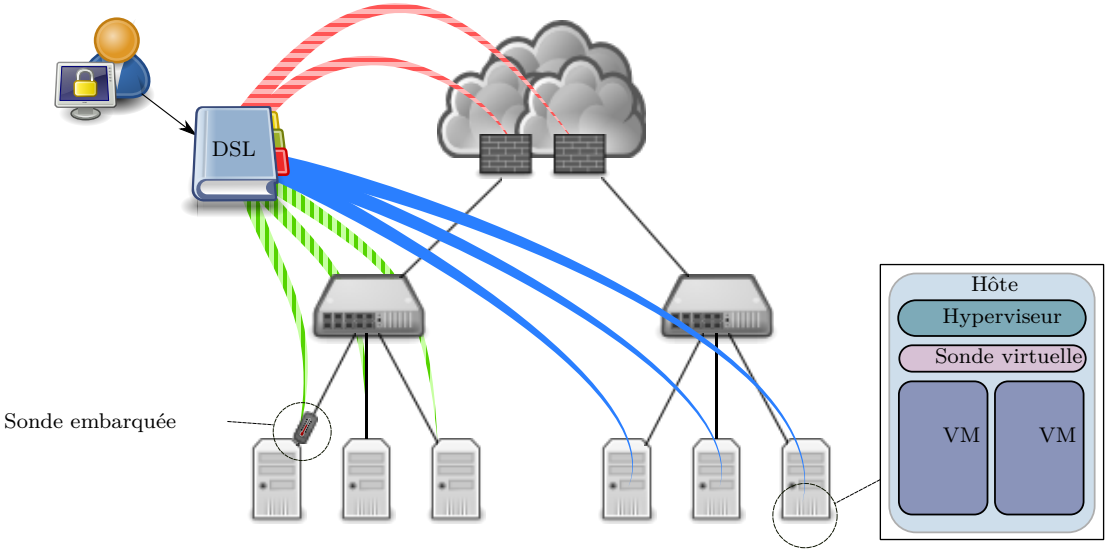


FIGURE 3.5 – Architecture de DISCUS.

Un nombre important de langages dédiés existent déjà pour des applications aussi nombreuses que variées comme l'écriture de pilotes de périphériques [66], l'implémentation de politiques d'ordonnancement [40], le développement de protocoles réseau [12, 11] ou la description d'attaques réseau [64].

La contribution principale de ce chapitre est DISCUS, une architecture de détection d'intrusions distribuée. Pour gérer le caractère hétérogène et la complexité de l'écriture de règles de génération d'alerte, nous proposons un langage dédié pour la génération des images des composants de l'architecture.

3.2.3 Un DSL pour la définition de CIDS

Pour configurer l'architecture de sécurité, nous proposons DISCUS Script, un langage dédié au domaine de la détection d'intrusion. Ce langage doit permettre à un expert de la sécurité réseau de décrire les événements qu'il souhaite détecter et pour lesquels le CIDS DISCUS doit lever une alerte.

Il est important à ce stade que l'expert sécurité n'ait pas besoin d'avoir une connaissance spécifique sur la programmation des différentes sondes. Savoir développer un circuit matériel FPGA ou un module d'analyse de paquets dans un hyperviseur n'est pas de son ressort. Le langage que nous proposons est donc agnostique des spécificités des cibles. Il fournit les moyens d'exprimer des analyses de paquets et de retenir l'état du système dans son ensemble, de façon distribuée.

L'analyse de paquets est intrinsèquement événementielle. L'analyse commence à l'arrivée d'un paquet. Le langage permet donc d'écrire des règles qui décrivent comment réagir à un événement particulier. Ce modèle de programmation, événementiel, permet de tirer parti du parallélisme naturel que l'on peut mettre en œuvre avec un circuit

électronique. Encore une fois, la cible FPGA a été en grande partie responsable des choix faits pour DISCUS.

Le langage est basé sur deux éléments clés : les tables et les règles. Les tables sont le moyen fourni par DISCUS pour que les sondes puissent collaborer. Elle sont le cœur d'une base de données distribuée que les sondes peuvent interroger. Les règles sont l'élément qui permet d'exprimer « l'intelligence » de l'IDS. Les algorithmes de détection d'intrusion sont implémentés comme une suite de règles qui se déclenchent à l'occurrence d'un événement.

3.2.3.1 Types

Le langage de DISCUS est un langage statiquement typé. Un script subi une vérification stricte au moment de la compilation pour vérifier la compatibilité des types employés dans les diverses expressions du langage. Les principaux types de bases du langage sont :

intN le langage de DISCUS manipule des entiers dont la taille, exprimée en bits, est toujours connue. Ainsi, on pourra utiliser des `int1, int2, ..., intN` pour des entiers de 1, 2, ..., `N` bits, respectivement. Le type de retour des différents opérateurs arithmétiques sera susceptible d'être plus grand (au sens du nombre de bits) que celui des opérandes ;

enum DISCUS peut utiliser un type énuméré. Les variables d'un type énuméré ne pourront prendre qu'un ensemble de valeur connues à l'avance ;

float des nombres réels. Contrairement aux entiers, la taille des **float** est fixée. En fonction des cibles, le compilateur DISCUS produira du code utilisant les unités de calcul flottantes, du calcul en virgule fixe à l'aide de nombres entiers ou un circuit dédié ;

ipaddr une variable désignant une adresse de machine ou de réseau (IPv4 ou IPv6). L'opérateur **in** du langage permet de tester l'appartenance d'une adresse d'hôte à un réseau ;

time une variable pouvant contenir une estampille de temps. Le mot clé **now** du langage permet de connaître l'heure actuelle ;

bitstream ce type représente un flux de bits, dont la taille peut être connue ou non à l'avance. Il est utilisé principalement pour représenter les données contenues dans un paquet. C'est un paramètre de type `bitstream` qui est fourni aux règles qui traitent l'événement élémentaire `packet`. Le langage fournit un ensemble d'opération sur les flux de bits comme l'extraction d'un entier de N bits, ou d'une sous partie du flux, la concaténation de deux flux ainsi que l'extraction de motifs [10] ;

3.2.3.2 Tables

Les tables sont au cœur de la collaboration des sondes de l'architecture. Elles permettent de stocker des informations relatives à l'état du réseau et peuvent être consul-

10. *au sens des expressions régulière*

tées par les sondes. Le terme *table* est ici emprunté au vocabulaire des systèmes de gestion de bases de données. Au même titre que les bases de données, les tables contiennent un ensemble d'enregistrements qui ont chacun une structure identique, des champs typés par un type de base du langage.

```
table connexions_actives {
  ipaddr hote_distant;
  ipaddr hote_local;
  time heure;
};
```

FIGURE 3.6 – Déclaration d'une table

La figure 3.6 donne un exemple de déclaration de table. Ici, cette table peut par exemple servir à retenir les connexions TCP établies avec l'heure à laquelle elles l'ont été. Les insertions, mises à jour ou suppressions dans ces tables, ainsi que leur consultation, seront les actions principales que pourront effectuer les règles de sécurité.

3.2.3.3 Règles de sécurité événementielles

L'élément basique de DISCUS est l'événement. Un script DISCUS est constitué d'un ensemble de règles, chacune de ces règles réagissant à un événement donné. L'événement élémentaire, celui à partir duquel on peut construire une analyse est la réception d'un paquet.

La syntaxe de déclaration d'une règle est donnée figure 3.7. L'exécution d'une règle est conditionnée par la réception d'un événement et d'une condition optionelle que l'on peut indiquer par la clause **where**. En particulier, cette clause permet de lire l'état des tables et de les comparer aux paramètres de la règle. Les clauses **insert** et **update** permettent de faire des modifications des tables. L'action **alert** permet de lever une alerte. La façon de réagir à une alerte est découplée de la description des règles et sera présentée section 3.2.4. Cela permet d'écrire des librairies de détection et de laisser les responsables sécurité décider du comportement à adopter en cas d'alerte (envoi d'un mail, d'un SMS, exécution d'un script, reconfiguration d'un pare-feu, etc.). Enfin, l'expression **raise** permet de créer un nouvel événement. Cet événement déclenchera à son tour d'autres règles. L'événement peut être levé immédiatement ou après un délai.

```
on <event_name>(<arguments>)
   [ where <conditions> ]
   [ insert <table entries>, ]
   [ update <table entries>, ]
   [ alert <alertName>, ]
   [ raise <events> [in X]];
```

FIGURE 3.7 – Déclaration d'une règle

```
on packet(bitstream b)
   where b[96:111] == 0x800 /* Champ EtherType == IPv4 */
   raise ipv4_packet(b[112:]);

on ipv4_packet(bitstream b)
   raise ipv4_payload(
              b[4:7], /* IHL */
              b[8:13], /* DSCP */
              b[14:15], /* ECN */
              b[16:31], /* Total length */
              (...)
              b[(b[4:7] * 32): ] /* payload */
   );
```

FIGURE 3.8 – Exemple de règles

Un exemple de règles est donné figure 3.8. Dans cet exemple, la première règle est déclenchée sur l'événement élémentaire `packet`. L'argument de cet événement est un `bitstream` qui contient l'entête ethernet et les données de la trame ethernet. La clause **where** permet de ne déclencher le corps de l'événement que dans le cas d'un paquet IPv4. Il déclenche alors un autre événement, `ipv4_packet`, à l'aide de l'instruction **raise**. La règle qui traite le paquet `ipv4_packet` se contente de découper l'entête IPv4 et de déclencher un nouvel événement, `ipv4_payload`, qui reçoit en paramètre les valeurs des champs de l'entête IPv4 et les données du paquet.

3.2.3.4 Exemple d'utilisation de DISCUS Script

Nous allons ici donner un exemple complet d'utilisation du langage de DISCUS. L'attaque que nous cherchons à détecter avec ce script est une attaque par inondation de paquet SYN. L'idée générale de l'attaque et d'ouvrir partiellement une multitude de connexion TCP sans jamais terminer la procédure de poignée de main (*i.e.* sans envoyer de paquet ACK). Comme la cible de l'attaque doit retenir toutes les tentatives de connexion TCP, elle peut vite manquer de mémoire et ne plus pouvoir accepter de connexion TCP légitimes, et donc ne plus être en mesure d'assurer son service.

De nombreuses méthodes existent pour détecter une telle attaque [33, 44, 57]. Pour cet exemple, nous avons choisi d'implémenter une version très simplifiée des filtres de complétions partiels (*Partial Completion Filter (PCF)* [44]).

L'idée des PCF est de maintenir un compteur pour chaque cible dont la valeur du couple $(ipDestination, portDestination)$ vu au travers d'une fonction de hachage est équivalente. Ce compteur est incrémenté à chaque paquet SYN et décrémenté à chaque paquet FIN. Une valeur proche de 0 sur un compteur correspond au comportement normal (toutes les connexions TCP légitimes s'établissent correctement et finissent par se terminer). Si la valeur dépasse un certain seuil, alors on peut supposer qu'une attaque est en cours sur la cible correspondante au compteur. Pour des raisons de simplicité

de présentation, la version que nous décrivons n'utilise pas de fonction de hachage et maintient un compteur par couple $(ipDestination, portDestination)$.

Pour stocker les compteurs, la table `tcp_table` donnée figure 3.9 est utilisée. Cette table permet d'associer le compteur à chaque couple $(ipDestination, portDestination)$. La mise à jour de cette table est gérée par les règles données figure 3.10. La première de ces règles est déclenchée si le paquet SYN est le premier reçu par le service et insère une entrée dans la table pour ce service. La deuxième règle est exécutée si l'entrée existe déjà dans la table, auquel cas elle met à jour le compteur. De plus, elle lève l'événement `check_flood_counter` qui sera capturé par la règle de la figure 3.12.

```
table tcp_table {
    ipaddr dst;
    int16 p_dst;
    int32 counter;
};
```

FIGURE 3.9 – Table pour la détection d'attaque en inondation.

```
/* Règle déclenchée si le compteur n'est pas encore créé */
on tcp_packet(ipaddr dst, int16 p_dst, int9 flags)
      where flags == SYN
      and not exists t in tcp_table
         with t.dst == dst,
              t.p_dst == p_dst
      insert into tcp_table {
          dst = dst;
          p_dst = p_dst;
          counter = 1;
      };

/* Règle déclenchée si le compteur existe */
on tcp_packet(ipaddr dst, int16 p_dst, int9 flags)
    where flags == SYN
    and exists t in tcp_table
        with t.dst == dst,
             t.p_dst == p_dst
    update t.counter += 1
    raise check_flood_counter(dst, p_dst);
```

FIGURE 3.10 – Détection d'une nouvelle connexion TCP.

Cette règle va comparer la valeur du compteur mis à jour avec un seuil et déclencher une alerte s'il le dépasse.

Enfin, la règle de la figure 3.11 décrémente le compteur à la fermeture d'une connexion TCP.

```
on tcp_packet(ipaddr dst, int16 p_dst, int9 flags)
    where flags == FIN
    and exists t in tcp_table
        with t.dst == dst,
             t.p_dst == p_dst,
             t.counter > 0
    update t.counter -= 1
```

FIGURE 3.11 – Fermeture d'une connexion TCP.

```
on check_flood_counter(ipaddr dst, int16 p_dst)
    where exists t in tcp_table
        with t.dst == dst,
             t.p_dst == p_dst,
             t.counter >= FLOOD_THRESHOLD
        alert flood_detected(dst, p_dst)
```

FIGURE 3.12 – Vérification du seuil et alerte.

Une fois les règles écrites, la création des images des différentes sondes de l'architecture DISCUS sera gérée par les outils de compilation et de déploiement.

3.2.4 Compilation et déploiement

Le langage DISCUS script que nous venons de présenter permet l'écriture de mécanismes de détection distribués tout en s'affranchissant des spécificités des différentes cibles matérielles.

La liaison entre les scripts et les images binaires finales sera effectuée par les différents outils de DISCUS. L'organisation des différents outils est donné figure 3.13. L'élément principal est le compilateur qui est en charge de transformer les fichiers textuels en un graphe de règles. Cette représentation est alors sauvegardée et constitue la base de travail de tous les autres outils.

Les outils d'analyse vont effectuer des actions de marquage sur le graphe afin d'en détecter les cycles, les nœuds inutiles (qui ne mènent pas à une règle qui effectue une action concrète, qui ne sont pas atteignables, etc.) ou tout autre élément pouvant mener à une optimisation ou à une déduction de propriété de correction du graphe de règle.

Les optimiseurs vont être chargés de réduire le graphe en supprimant les règles marquées inutiles, en fusionnant les règles qui peuvent l'être (pour limiter le nombre d'événement généré), etc. Par exemple, un optimiseur pourra fusionner les deux règles de la figure 3.8. En effet, comme la règle `ipv4_packet` ne dépend d'aucune condition et que la seule action de `packet` et de lever l'événement `ipv4_packet`, il est facile de construire une règle unique qui réagit à l'événement `packet` et lève l'événement `ipv4_payload`.

Le « *mapper* » est chargé de l'instanciation des règles en fonction de la topologie finale. Il prend en entrée l'ensemble des règles ainsi qu'une descriptions des cibles (capacités matérielle, description d'architecture, ...) et de la topologie (ensemble des

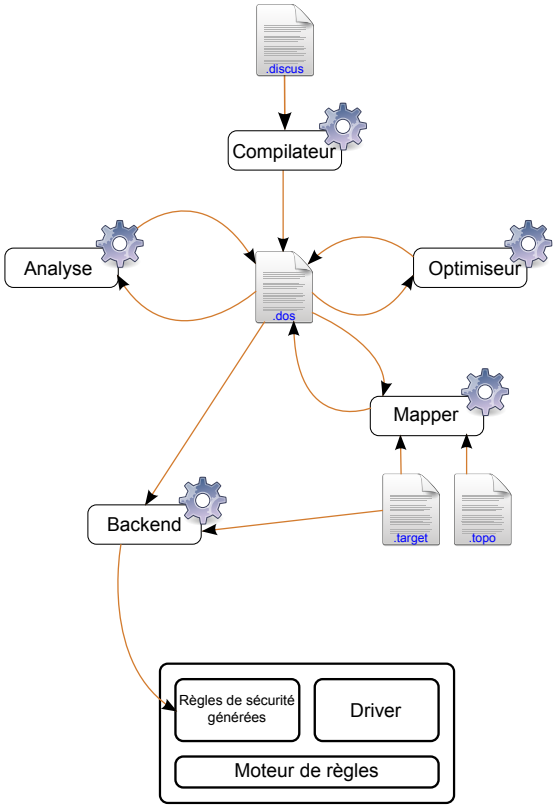


FIGURE 3.13 – Les outils DISCUS.

équipements, connexion entre ces équipements, adresses des différents réseaux). Cet outil génère alors pour chaque cible un fichier décrivant les règles qui tiennent compte de la topologie et les outils d'analyse et d'optimisation peuvent alors optimiser chacun de ces ensembles de règles dédiées afin de nettoyer encore plus spécifiquement les règles en fonction de la cible sur laquelle elle vont s'exécuter. L'analyse et l'optimisation seront exécutées jusqu'à l'obtention d'un point fixe (*i.e.* l'optimisation ne parvient plus à améliorer l'ensemble de règles).

Enfin, les différents « *backends* » vont générer l'image binaire spécifique à chaque cible à partir de la base de règles dédiées et optimisées pour celle ci. Les images binaires pourront alors être déployées sur chacun des éléments de l'architecture. La procédure exacte de déploiement dépendra alors de chacune des cibles.

La description de la topologie contient également les réactions à avoir en fonction des alertes générés par les scripts DISCUS.

En situation réelle, les règles seront généralement écrites par des spécialistes en sécurité, des chercheurs dont le domaine est la détection d'attaques particulières et la définition des contres-mesures associées.

La topologie sera elle donnée par le responsable de l'infrastructure surveillée. Ces deux acteurs, même s'ils peuvent dans certains cas être les mêmes personnes, seront généralement deux ensembles de personnes physiques distincts. Les premiers seront

généralement les fabriquants des solutions de sécurité ou une communauté de développeurs de mécanismes de détection. Les seconds seront les utilisateurs de la solution. Cette distinction importante ouvre la possibilité d'un développement communautaire de règles de sécurité, à la manière d'IDS comme Snort par exemple.

3.3 Conclusion

La protection des architectures complexes telles qu'utilisées pour la création de services cloud nécessite la surveillance du trafic réseau dans sa globalité. Développer le logiciel qui équipera un nombre important d'éléments hétérogènes dont la puissance et même le modèle de programmation varie énormément relève de la gageure.

Nous avons ici fait le choix de nous concentrer sur l'application visée et de développer une architecture basée sur un langage dédié de façon à résoudre les problèmes d'hétérogénéité par les outils de compilation et d'optimisation.

Cette approche nous permet de fournir un langage qui peut être utilisé par les experts en sécurité, même s'ils ne sont pas à même de développer du logiciel embarqué. Les outils permettent de limiter les erreurs de développement en effectuant des vérifications fortes sur un ensemble de propriétés du logiciel produit comme la garantie de terminaison, l'élimination de code mort ou la vérification de la cohérence des types.

La thèse de Damien Riquet est en cours de finition et ses travaux posent les fondations de la solution DISCUS : la définition de l'architecture, du langage et du modèle de déploiement. Il a, à l'heure actuelle, montré une partie des possibilité de DISCUS en développant un backend qui génère des règles comprises par Snort et est donc capable de générer la configuration de Snort à partir des description de règles DISCUS. Pour montrer l'expressivité du langage de DISCUS, un outil de transformation de règles Snort en règles DISCUS a été développé. Cet outil permet une conversion de 98% des règles Snort dans le langage DISCUS. Une campagne de mesure qui permettra d'évaluer les apports de DISCUS en terme de performances, de détection d'erreur et de fiabilité sur différentes architectures cibles est en cours.

Les travaux de Damien ont pour l'instant mené à quatre publications [C3, C4, C7, C8] dans des conférences ou ateliers avec comité de lectures ainsi qu'à un poster dans une conférence majeure du domaine [M2].

Plusieurs verrous scientifiques sont encore à étudier pour que DISCUS devienne une réalité. En particulier, la distribution des tables est pour l'instant réalisée de façon naïve et il faudra s'atteler à des méthodes efficaces pour partager l'information entre les différents éléments de l'architecture. Nous pensons notamment à l'utilisation d'architecture de type pair à pair, mais la quantité d'informations qui devront être traitées par la solution demande une analyse poussée. Sur cet aspect, il faudra également adresser les problématiques de fiabilité et de sécurité. Il faudra que les protocoles permettant la cohérence des tables soient capable de gérer l'éventuelle faute d'une ou plusieurs sondes, qu'elle soit causée par une panne directe ou par une compromission. Les protocoles de consensus tolérant aux fautes, et en particulier aux fautes byzantines [74, 35, 30], offrent de nombreuses pistes pour gérer cet aspect, mais il faudra bien évidemment en étudier l'application en environnement embarqué fortement contraint. De plus, les

protocoles de déploiement et de mise à jour des sondes pendant leur fonctionnement devront garantir qu'un attaquant n'aura pas la possibilité de les compromettre en y injectant du code malicieux.

Enfin, la cible qui est à l'origine de l'élaboration de DISCUS, les circuits FPGA n'est pas encore développée. S'il peut sembler naturel de générer un circuit qui exécute les règles DISCUS, nous pensons qu'il n'en est rien et que réaliser le backend FPGA se heurtera à de nombreux obstacles scientifiques et technologiques qu'il faudra adresser dans nos futurs travaux.

L'encadrement de la thèse de Damien m'aura montré à quel point il est difficile pour un doctorant de démarrer sur une page vierge sans aide pour le développement des prototypes nécessaires à la production et l'évaluation des résultats. Le thème est nouveau dans l'équipe et, faute de financement, Damien n'a pu avoir l'appui d'un ingénieur pour développer les parties « non scientifiques » mais indispensables à l'évaluation de ses travaux. Je compte apporter une attention particulière à ce point pour les futurs travaux que j'encadrerai.

Chapitre 4

Mise en réseau d'équipements embarqués dans un contexte de villes intelligentes

Les travaux présentés dans ce chapitre sont le fruit de la thèse de Tony Ducrocq [T3] que j'ai co-encadré avec Nathalie Mitton.

Cette thèse a été effectuée dans le cadre du projet ANR *BinThatThink* dont le but était de réfléchir à l'apport des sciences informatiques sur l'environnement et en particulier sur l'optimisation de la collecte de déchets. L'idée générale du projet était de proposer des moyens d'améliorer à la fois la qualité de la collecte en terme de tri, mais également les conditions de ramassage.

Il fallait alors être en mesure d'élaborer des conteneurs capables de prévenir les équipes de ramassage de la qualité du tri ainsi que de l'éventuelle présence d'éléments dangereux (explosifs, corrosifs...) dans ces derniers.

Nous avons travaillé sur la mise en réseau de ces conteneurs, effectuée à l'aide d'équipements embarqués, fonctionnant sur batterie. Les principales contributions de ces travaux sont donc des protocoles de communication et d'organisation des éléments du réseau économes en énergie.

4.1 Motivations

4.1.1 Réseau et ville « intelligente »

La définition exacte du terme « ville intelligentes », bien qu'il soit fréquemment utilisé, reste assez flou. Néanmoins, un critère que doit respecter une ville qui se dit intelligente semble faire l'unanimité. Une ville intelligente doit être connectée, elle doit utiliser les nouvelles technologies pour améliorer le confort des habitants et l'empreinte écologique de la ville.

Pour augmenter le confort des habitants ou l'attrait de la ville, il faut pouvoir mesurer la qualité de l'air, de l'eau, sa consommation électrique, en gaz ou en pétrole et utiliser ces mesures pour optimiser, si possible de façon automatique, l'utilisation de

toutes ces ressources. Pour effectuer les mesures, on doit disposer de capteurs disséminés partout dans la ville et, pour relever les informations de ceux-ci, ils se doivent d'être connectés pour que ce relevé soit automatique. Le concept de ville intelligente devient donc par cet aspect assez indissociable de la notion d'internet des objets.

Si l'on veut disposer un nombre important de capteurs, il paraît délicat de leur apporter de l'énergie ou un lien de communication via un fil, ce qui nécessiterait de lourds investissements d'infrastructure. Les technologies permettant ce déploiement se basent donc sur de la communication sans-fil et sur une alimentation autonome par une batterie, si possible rechargée par des moyens de collecte de l'énergie offerte par l'environnement (comme l'énergie solaire par exemple).

La problématique majeure d'un déploiement de ce type devient alors le coût unitaire de chaque objet et son efficacité énergétique. Les composants que l'on va mettre en place seront donc nécessairement fortement contraints en terme de puissance de calcul ou de capacité mémoire. En plus de ces contraintes, et contrairement aux équipements de types cartes à puce ou sondes de détection d'intrusion dont nous avons discuté dans les chapitres précédents, ces objets devront également être très économes en énergie. Il est bien évidement impensable de changer les piles de capteurs qui auraient été coulés dans le béton à la construction d'un bâtiment pour en surveiller la structure.

C'est donc dans une optique d'économie d'énergie qu'ont été conduits les travaux présentés dans ce chapitre. L'application visée était le cas particulier de la collecte de déchets, et donc la mise en réseau des conteneurs de tri et des camions de ramassage, mais les algorithmes présentés ici restent pertinents dès que l'on souhaite établir un réseau de capteurs embarqués qui doit effectuer de la surveillance et de la remontée d'alerte.

Pour la communication entre les conteneurs et les équipes de ramassage, nous nous sommes intéressés à deux scénarios :

— les conteneurs ne disposent que d'une interface radio, et certains d'entre eux connaissent leur position précise. Nous avons ici étudié les protocoles de routage dit « géographiques » à connaissance de position partielle.
— les conteneurs disposent de deux interfaces radio, une à longue distance qui consomme une quantité d'énergie importante et une à courte distance qui utilise moins d'énergie. Il s'agissait donc ici d'organiser la communication des équipements pour balancer les communications à courte et longue distance pour allonger au maximum la période durant laquelle tous les éléments du réseau peuvent continuer à fonctionner grâce à leur batterie.

4.1.2 Collecte d'informations dans un réseau sans-fil multi-saut

Le routage des données dans un réseau sans-fil multi-sauts est un sujet de recherche très actif depuis les années 1990 qui a mené à la standardisation d'algorithmes comme AODV [51, 71] ou OLSR [47, 59]. Ces deux algorithmes illustrent bien les deux grandes familles de protocoles de routages, respectivement réactifs (la route est construite à la demande) et pro-actifs (des tables de routages sont maintenues régulièrement). Dans les deux cas, des informations sur la construction des routes, c'est à dire sur la liste des

identifiants des nœuds [1] à contacter pour envoyer un paquet d'un point à un autre du réseau, doit être « mémorisée » [2].

Pour des réseaux très larges, à l'échelle d'une ville, il en résulte alors une consommation mémoire importante ou une taille de paquets importante au regard de la taille des données à transmettre. Pour répondre à cette problématique, deux familles d'algorithmes ont été proposées : l'organisation du réseau en groupe ou « clusters » et les algorithmes de routage géographique.

Le but d'un algorithme de clustering [24, 34] est d'organiser une hiérarchie au sein du réseau. Certains éléments seront désignés *chef* ou *cluster heads* et seront chargés de collecter les informations des autres éléments placés sous leur responsabilité. Le chef pourra alors traiter les données, les agréger, les réduire ou les compresser. Les données peuvent alors être collectées soit par action physique [3], soit par exemple en utilisant une interface de communication longue distance. La formation des groupes peut être déterminée statiquement, au moment du déploiement du réseau, ou dynamiquement, au cours de l'évolution du réseau. La détermination statique est utilisée généralement si l'ont peut distinguer des éléments particuliers. Ces éléments peuvent être fixes, avoir une énergie illimitée ou d'importantes capacités de communication. Le recours à la détermination dynamique sera plutôt utilisé si un ensemble important d'éléments peuvent endosser le rôle de chef et si ce rôle impose une contrainte à l'équipement comme une consommation accrue d'énergie par exemple.

Les protocoles de routage géographique [56, 82] utilisent quand à eux les informations de position des éléments du réseau. Un nœud connaît sa position et la position de la destination du paquet. Ces algorithmes seront alors plutôt utilisés quand les nœuds connaissent à l'avance la position du ou des sites de collecte. À la réception d'un paquet, un nœud choisi parmi ses voisins la suite de la route en fonction de leurs deux positions et de la position de la cible.

La façon de faire ce choix va varier en fonction des différents protocoles. Par exemple :

- MFR [85] choisi celui qui minimise la distance euclidienne entre la projection orthogonale du voisin sur la droite portée par la source et la destination et cette dernière (nœud a figure 4.1) ;
- GREEDY [82] choisi le nœud qui minimise la distance euclidienne avec la destination (nœud b figure 4.1) ;
- NFP [83] choisi le nœud le plus proche de la source qui permet un progrès vers la destination ((nœud c figure 4.1) ;
- COMPAS[70] choisi le nœud qui minimise l'angle voisin-source-destination ($\widehat{eSD}$ figure 4.1).

Le choix du protocole à utiliser sera alors dicté principalement par l'application visée. Dans le cadre du projet *BinThatThinks*, l'application pour laquelle nous avons dû

1. On parle ici de nœuds par analogie avec la théorie des graphes où un réseau serait modélisé par un graphe dont les nœuds sont les éléments du réseau. Une arête existe alors entre deux nœuds s'ils sont à portée radio.

2. Pour OLSR et AODV les informations sont effectivement stockées en mémoire, mais certains protocoles réactifs utilisent les entêtes des paquets pour mémoriser la route.

3. Comme la récupération manuelle d'une carte mémoire sur laquelle sont stockées les données

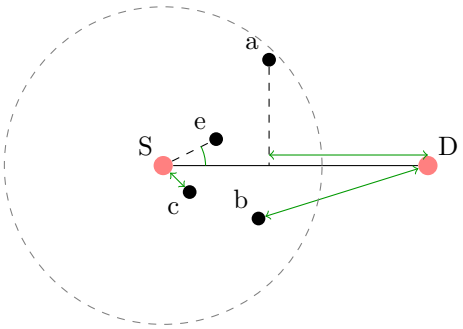


FIGURE 4.1 – Illustrations de différents algorithmes de routage géographique. Le nœud a est choisi par *MFR*, le nœud b par *Greedy*, le nœud c par *NFP* et le nœud e par *Compass*.

proposer des solutions est celle de la collecte des données générées par les conteneurs de déchets vers un site de collecte ou un camion de ramassage. Connaître les informations sur les taux de remplissage des conteneurs, sur la possibilité de leur ramassage (accessible par le camion ou non) permet d'optimiser les collectes, d'améliorer le tri et contribue donc à une meilleure gestion, un meilleur tri et à une collecte plus efficace des déchets ménagers et industriels.

Nous avons donc envisagé des solutions répondant aux deux scénarios de collecte de données.

4.2 Collecte de données pour réseaux urbains

Nous avons ici cherché à proposer des solutions de collecte de données pour les deux scénarios imposés par le projet. Ces deux scénarios appellent naturellement à l'utilisation des deux familles de protocoles de collecte que nous avons présenté, à savoir les protocoles de routage géographique et les protocoles d'auto organisation sous forme de cluster.

4.2.1 Routage géographique à connaissance partielle

S'ils sont considérés comme efficaces et ne demandant que peu de ressources (en particulier mémoire), les algorithmes de routage géographique sont tout de même fondés sur une hypothèse forte : tous les nœuds doivent connaître leur position. Cette position peut être obtenue par configuration statique ou par positionnement GPS mais le coût de l'équipement augmente.

Des algorithmes ont été proposés pour déduire la position de nœuds qui ne connaissent pas leur position grâce à ceux qui la connaissent [37, 54], mais, faute de dispositif de mesure de distance précise entre les nœuds, leur performance est faible. Une étude sur l'impact de l'erreur de positionnement sur le routage géographique [45] montre que celle ci a un impact important sur les taux de livraison des algorithmes.

D'autres solutions, à base de coordonnées virtuelles, ont été proposées [36]. L'inconvénient majeur est que ce système de coordonnées virtuelles nécessite de nombreux échanges de messages et donc consomme beaucoup d'énergie et diminue la bande passante utile du réseau.

Nous avons fait le choix de considérer que seul un sous ensemble des nœuds connaît la position précise. Ceci peut bien évidemment être obtenu à l'aide d'un GPS, mais dans le contexte d'une ville intelligente, on peut plus facilement envisager que les nœuds qui connaissent leur position seront des nœuds fixes, par exemple attachés sur les éclairages publics ou les feux de circulation. La position de ces derniers sera donc facilement connue au moment du déploiement et donc par configuration.

À partir de cette hypothèse, nous avons proposé l'algorithme HGA [4] [C1] qui combine les avantages des algorithmes géographiques et réactifs.

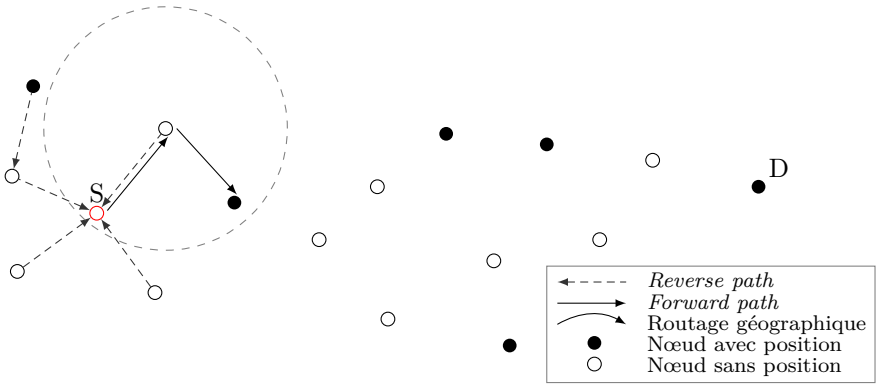
4.2.1.1 Algorithme HGA

Le principe général d'HGA est d'utiliser un routage géographique tant que cela est possible, c'est à dire tant que les positions des nœuds nous permettent de suivre la direction vers le puits. Quand la position n'est pas disponible, on effectue une recherche de route à courte distance jusqu'à trouver le prochain nœud qui connaît sa position. Cette recherche de route est similaire à celle d'AODV [71]. La distance de recherche est un paramètre de l'algorithme, que l'on notera k est qui est la profondeur de recherche de route maximale en nombre de sauts. L'algorithme HGA qui effectue une recherche à profondeur k sera nommé HGA-k.

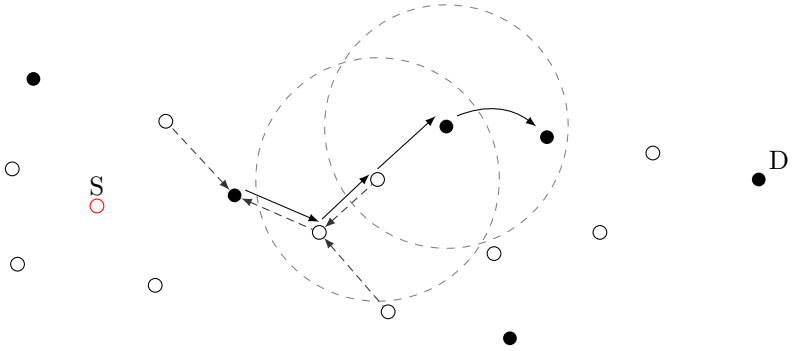
L'entête d'un paquet de données routé par l'algorithme HGA contient la position de la destination et la dernière position connue par laquelle le paquet a transité. Pour router un paquet de données, un nœud u vérifie si l'un de ses voisins permet un progrès vers la destination (c'est la partie géographique de l'algorithme). Si u connaît sa propre position, le progrès dépendra de cette dernière. Sinon, le progrès dépend de la dernière position connue qui est contenue dans l'entête du paquet. Si aucune position n'est contenue dans l'entête, un voisin connaissant sa position est choisi comme prochain saut pour la route.

Si aucun des voisins de u ne permet de progrès vers la destination, l'algorithme bascule en mode réactif. Il va tout d'abord rechercher dans sa table de routage une entrée connue pour proposer un progrès et transmettre le paquet au voisin permettant de joindre cette entrée. Si aucune route offrant un progrès n'est disponible, l'algorithme effectue une recherche de route par un message RREQ [5]. Ce message est transmis de proche en proche (par un algorithme d'inondation) jusqu'à ce qu'un nœud v connaissant une route proposant un progrès soit trouvé ou que le message RREQ ait parcouru k sauts. Le nœud v répond alors au nœud u à l'aide d'un message RREP [6] et u met à jour sa table de routage et transmet le paquet par la route qu'il vient de découvrir. Si le message RREQ est transmis sur k sauts sans succès, le nœud u met le paquet

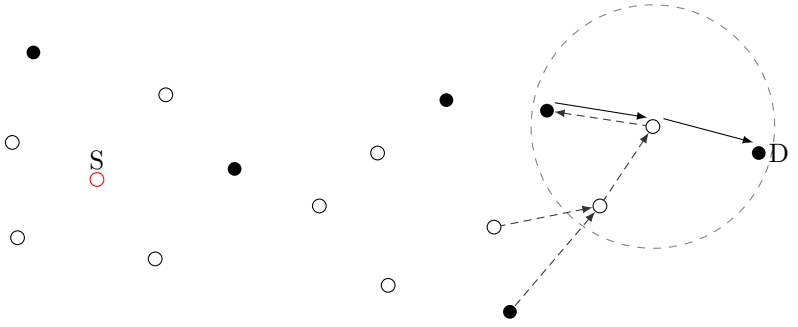
4. Hybrid Greedy-AODV
5. RREQ : Route REQuest
6. Route REPly



(a) Première RREQ.



(b) Deuxième RREQ et routage géographique.



(c) Troisième RREQ.

FIGURE 4.2 – Exemple de routage dans HGA.

de données en attente et retente une recherche après un délai d'attente. La figure 4.2 montre un exemple de paquet routé par HGA.

4.2.1.2 Évaluation

Nous avons évalué l'algorithme HGA en le comparant à l'algorithme VCAP [36] qui utilise un système de coordonnées virtuelles pour effectuer un routage proche d'un algorithme géographique. Les coordonnées d'un nœud sont exprimées par une distance (en nombre de sauts) entre ce nœud et des ancres fixes. La mise à jour des coordonnées est effectuée par l'annonce par chaque nœud de sa distance aux ancres à ses voisins. Les nœuds considèrent dans un premier temps qu'ils sont à une distance infinie des ancres et mettent à jour leur distance à chaque fois qu'ils reçoivent un message leur annonçant une distance inférieure.

Nous avons également mesuré les performances de l'algorithme Greedy [82] afin d'illustrer l'impact du manque de position sur un algorithme de routage géographique classique.

L'algorithme HGA a été évalué avec des profondeurs de recherche de route de un à cinq sauts. VCAP a été utilisé dans deux variantes, l'une à trois ancres, l'autre à cinq ancres. Le simulateur WSNET [29] a été utilisé pour les expériences. Les métriques observées sont le taux de livraison des paquets, le nombre de message de contrôles (et donc le trafic non lié directement à des données) ainsi que l'utilisation mémoire des différents algorithmes (occupés par les tables de voisinage et de routage).

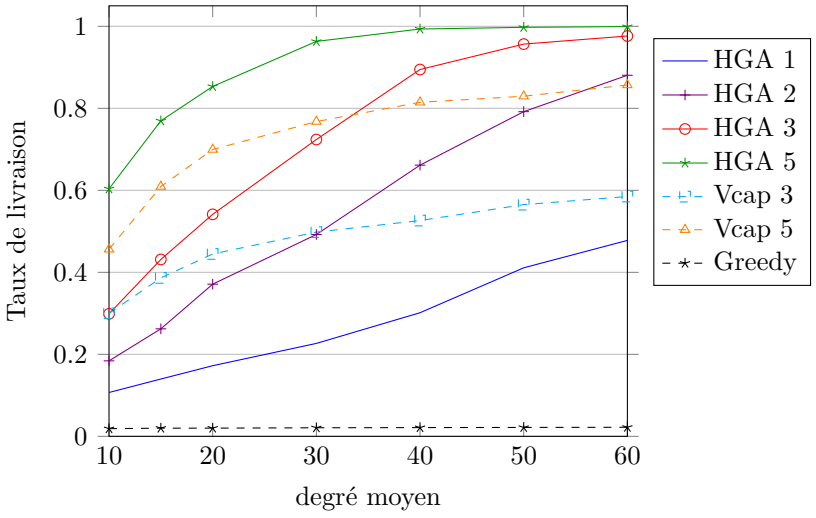


FIGURE 4.3 – Taux de livraison des différents algorithmes de routage avec **1%** des nœuds ayant leur information de position pour HGA et Greedy.

Les figures 4.3 et 4.4 montrent une comparaison des taux de livraison des algorithmes HGA-k, Vcap et Greedy avec respectivement 1% et 10% de nœuds ayant connaissance de leur position. Ce paramètre n'influe pas sur les algorithmes Vcap qui utilisent un positionnement virtuel, mais les résultats le concernant sont repris sur les deux figures pour faciliter la comparaison. Un taux de livraison de 1 représente un routage parfait, sans perte de paquet. Ces courbes sont données en fonction du degré moyen des nœuds.

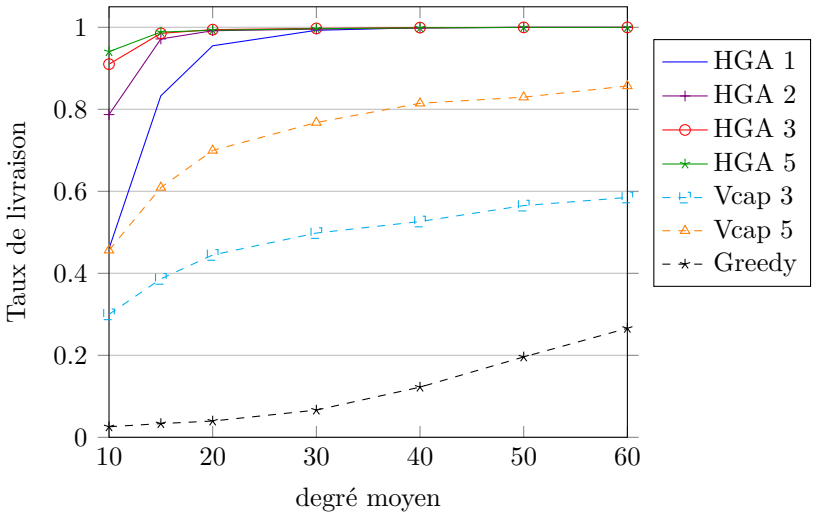


FIGURE 4.4 – Taux de livraison des différents algorithmes de routage avec **10%** des nœuds ayant leur information de position pour HGA et Greedy.

On observe, comme prévu, que l'algorithme Greedy s'effondre quand l'information de position n'est que partielle. Dès 10% de nœuds connaissant leur position, les algorithmes HGA-k se comportent bien mieux que les variantes de Vcap. Ce gain important de fiabilité dès une densité de 20 se paye néanmoins par le coût en messages de contrôle comme le montre la figure 4.5. Cependant, on peut voir que, si la densité le permet (au delà de 20), l'algorithme HGA-1 permet un taux de livraison de plus de 95% tout en ayant un surplus de message de contrôle très faible par rapport à Vcap.

Enfin, la consommation mémoire de l'algorithme HGA, donnée en octets figure 4.6 [7], si elle est importante à densité faible, décroît significativement quand la densité augmente ce qui confirme les possibilités de passage à l'échelle en terme de mémoire pour HGA. Cette diminution s'explique par la plus grande probabilité d'avoir un voisin connaissant sa position ce qui a pour effet de ne pas avoir besoin de constituer une table de routage.

Fonder les hypothèses de fonctionnement du protocole sur l'application nous a ici permis de proposer une solution efficace, tant en terme d'efficacité qu'en terme de consommation mémoire ou d'utilisation de l'interface radio (et donc en congestion du réseau et en énergie consommée). En effet, c'est cette spécificité qui permet à HGA de se comporter de bien meilleure façon que les différents algorithmes auxquels nous l'avons comparé, qui ne sont pas adaptés à une connaissance partielle de la position. Il existe encore un certain nombre d'évolutions pouvant être apportées à l'algorithme. En particulier, l'étude d'hybridation entre d'autres algorithmes géographiques et « classiques » est une piste à explorer pour améliorer cette proposition.

Dans le contexte qui nous intéresse, à savoir les villes intelligentes et connectées,

7. HGA utilise 8 octets par entrée dans la table de routage, vcap3 10 octets par voisin, vcap5 14 octets et Greedy 8 octets par voisin

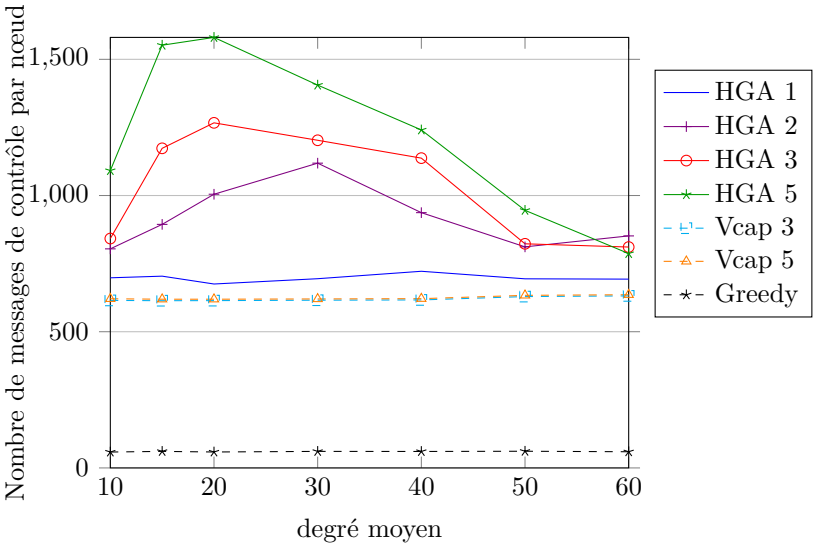


FIGURE 4.5 – Nombre de messages de contrôle des différents algorithmes de routage avec **10%** des nœuds ayant leur information de position pour HGA.

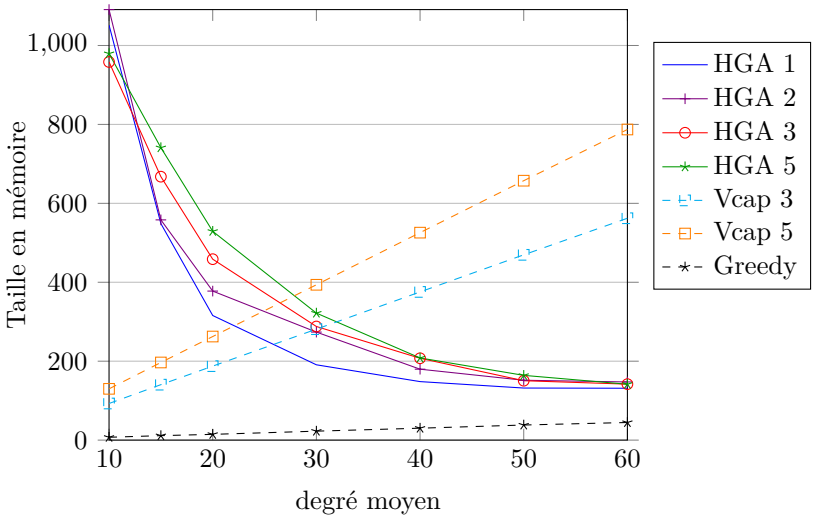


FIGURE 4.6 – Consommation mémoire moyenne pour les différents algorithmes de routage avec **10%** des nœuds ayant leur information de position pour HGA.

le routage géographique devient utilisable sans avoir besoin d'un quelconque recours à un système de positionnement car les équipements dont la position peut-être connue à l'avance sont légion. Ce type d'algorithmes permet donc de concevoir des systèmes embarqués à un coût bien plus faible et qui consommeront une quantité d'énergie bien moins importante. Il sera alors plus raisonnable dans ces conditions d'envisager un

déploiement massif au sein d'une ville.

Dans le cas de densité moins importante, pour des villes de taille plus modeste en milieu rural par exemple, nous avons envisagé un deuxième scénario. Dans ce dernier, les nœuds n'utilisent pas de routage géographique mais sont équipés de deux interfaces de communication. L'une à courte distance, économe en énergie, l'autre à longue distance, qui consomme beaucoup plus d'énergie. Dans ces conditions, le réseau doit proposer une organisation qui va lui permettre de prolonger sa durée de vie.

4.2.2 Auto-organisation pour l'optimisation de la durée de vie

Nous nous intéressons ici au cas où les éléments du réseau disposent tous de deux interfaces de communication :

— un module sans-fil basse consommation IEEE 802.15.4™ pour les communications pair à pair entre les nœuds ;
— un module sans-fil GPRS pour les communications longue distance avec le site de collecte ou le camion de ramassage.

Ce scénario est directement issu des discussions avec Véolia Propreté au sein du projet *BinThatThinks*. Les conteneurs peuvent communiquer directement entre eux via une interface basse consommation et remonter les données et les alertes via une interface longue distance, mais qui consomme beaucoup plus d'énergie. Les conteneurs peuvent contacter le centre de tri via l'interface GPRS et communiquer entre eux via l'interface IEEE 802.15.4™.

Nous donnons table 4.1 une comparaison des deux interfaces en terme de consommation et de débit. On voit aisément que la communication GPRS, si elle permet de communiquer à longue distance car c'est un réseau cellulaire, va coûter beaucoup plus cher en terme d'énergie non seulement car sa consommation instantanée est 40 fois importante mais également car le débit atteignable est 10 fois moindre.

	IEEE 802.15.4	GPRS
Débit max	250 kbit/s	26.8 kbit/s
Consommation au repos	0.77 mW	6.4 mW
Consommation de transmission	31.32 mW	1.25 W
Consommation de réception	35.46 mW	1.25 W

TABLE 4.1 – Comparaison des interfaces radio employées.

Pour que la durée de vie du réseau soit la plus grande possible il faut donc limiter au maximum l'utilisation de l'interface GPRS. De plus, de part la nature de l'application visée, il faut absolument que l'intégralité des éléments du réseau soient en état de fonctionner (et donc n'épuisent pas leur batterie) le plus longtemps possible. Il est donc nécessaire d'équilibrer la consommation d'énergie et donc l'utilisation de l'interface GPRS. Les algorithmes de *clustering* [34] nous ont servis de base de départ pour répondre à cette problématique.

4.2.2.1 Algorithmes proposés

Nous avons proposé la famille d'algorithmes BLAC[8] [C5, C6, J1] qui répond à notre problème en améliorant significativement les propositions de la littérature [15, 23, 41, 42, 55, 77] sur les points suivants :

— BLAC est complètement décentralisé et n'utilise que des informations locales (sur les éléments située à portée radio) ;
— BLAC considère la durée de vie du réseau comme étant celle durant laquelle **tous** les éléments restent en fonction et fournissent leur service et cherche donc à conserver tous les nœuds actifs le plus longtemps possible.

La construction des clusters de l'algorithme BLAC est dérivée de l'algorithme de Mitton *et al.* [39]. Cette construction est basée sur la notion de densité d'un nœud. La densité d'un nœud u, notée $\rho(u)$, est définie comme le rapport entre le nombre de liens qui existent entre les voisins de u et le nombre de voisins de u, soit :

$$\rho(u) = \frac{|(v,w) \in E | v \in \{u, \mathcal{N}(u)\}, w \in \mathcal{N}(u)|}{\delta(u)},$$

avec E l'ensemble des arrêtes du graphe modélisant le réseau, $\mathcal{N}(u)$, l'ensemble des voisins de u (c'est à dire l'ensemble des nœuds pouvant communiquer directement avec u) et $\delta(u)$ le degré de u (le nombre de voisins de u).

Le choix de chef de cluster est alors effectué en choisissant le nœud qui a la plus forte densité dans son voisinage. Ce choix s'effectue localement en échangeant les valeurs de densité avec ses voisins. En cas d'égalité, l'ambiguïté est levée en choisissant le chef de cluster comme le nœud ayant le plus petit identifiant (qui peut être l'adresse MAC de l'interface réseau par exemple).

L'idée de l'algorithme BLAC est de considérer en plus de la notion de densité ou de degré la charge restante de la batterie. Le calcul de clustering se fait donc maintenant avec une nouvelle métrique :

$$c(u) = B(u) \times h(u)$$

avec

$$B(u) = \left\lfloor \frac{batt(u) \times 10}{battcap} \right\rfloor,$$

où $batt(u)$ et le niveau de batterie restant et $battcap$ la capacité totale de la batterie. Le fait de discrétiser le niveau de batterie entre 0 et 10 permet à la valeur de moins fluctuer et contribue à la stabilité de l'algorithme.

L'algorithme BLAC est proposé en utilisant deux métriques possibles en remplaçant $h(u)$ soit par $\delta(u)$ soit par $\rho(u)$.

Une variante de l'algorithme a également été proposée qui effectue dans un premier temps une réduction du graphe d'origine en un graphe RNG [88] qui peut être construit localement [53, 61]. En combinant les deux métriques et les deux constructions, on obtient alors les quatre variantes de l'algorithme de la table 4.2.

8. **B**attery-**L**evel **A**ware **C**lustering

	Sans réduction de graphe	Réduction RNG
métrique $\rho(u)$	BLAC-bs	BLAC-rs
métrique $\delta(u)$	BLAC-bg	BLAC-rg

TABLE 4.2 – Variantes de l'algorithme BLAC.

4.2.2.2 Évaluation des algorithmes BLAC-*

L'évaluation des algorithmes a été réalisée à l'aide du simulateur WSNET en intégrant les paramètres de consommation énergétique des équipements réels donnés précédemment table 4.1. Chaque nœud du réseau génère 16 kbits de données périodiquement (toutes les cinq secondes) et les transmet au chef de cluster à l'aide de son interface radio basse consommation. Le chef de cluster transmet alors ces données au site de collecte à l'aide de la liaison GPRS. Afin que le temps de simulation soit raisonnable, la batterie des nœuds est choisie avec une capacité de 32 mWh, ce qui explique que la durée de vie du réseau n'est que d'une poignée d'heures.

Les topologies utilisées sont créées en utilisant des cartes réelles issues de quartiers de la ville de Lille et de petites villes du nord de la France. Les nœuds sont placés dans les rues, à proximité des bâtiments. Les données du projet *OpenStreetMap* [7] ont été utilisées pour générer ces topologies. Trois exemples de topologies sont donnés figure 4.7. Chaque point rouge représente ici un équipement embarqué.

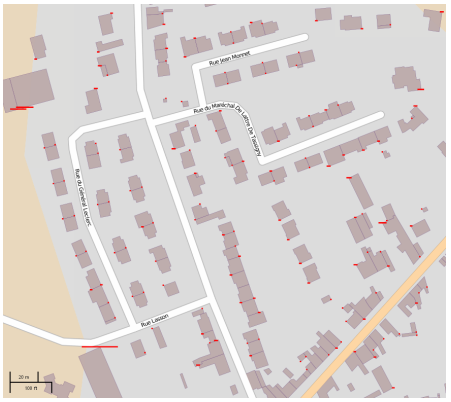
Nos algorithmes sont comparés à DDR [68] et l'algorithme basé uniquement sur la densité présenté plus haut [39] en raison de leur principe approchant celui de BLAC. Nous avons également comparé nos algorithmes à LEACH [65], qui considère également l'économie d'énergie avec pour valeur de son paramètre p : 5, 10 et 20%.

Nous donnons ici les résultats concernant la durée de vie du réseau (figure 4.8) et le taux de livraison de l'algorithme (figure 4.9). Le temps est exprimé en heures. Comme on peut le voir, les variantes de nos algorithmes permettent de maintenir l'intégralité des nœuds du réseau en vie pour une durée quasiment trois fois supérieure à celle obtenue par les autres algorithmes. Ceci est obtenu grâce au meilleur équilibrage des rôles de chef de cluster en fonction de la quantité d'énergie disponible. Par contre, et pour la même raison, tous les nœuds arrêtent de fonctionner presque tous en même temps. Dans un scénario dans lequel il est primordial que tous les nœuds restent fonctionnels, nos algorithmes offrent donc la meilleur alternative. Si l'on imagine que l'on adjoint aux nœuds un moyen de collecte d'énergie (solaire, vibration, etc.), notre solution permet d'amortir les écarts de génération d'énergie en assurant un fonctionnement normal du réseau.

En terme de taux de livraison, nos algorithmes ont une performance similaire aux autres algorithmes testés.

Le lecteur intéressé par les autres propriétés de l'algorithme (stabilité, configuration des clusters, etc.) est invité à se référer au document de thèse de Tony Ducrocq [T3] ou à la publication annexée au document [J1].

(a) 250 nœuds en environnement urbain



(b) 250 nœuds en environnement rural

(c) 100 nœuds en environnement urbain

FIGURE 4.7 – Exemple de topologies.

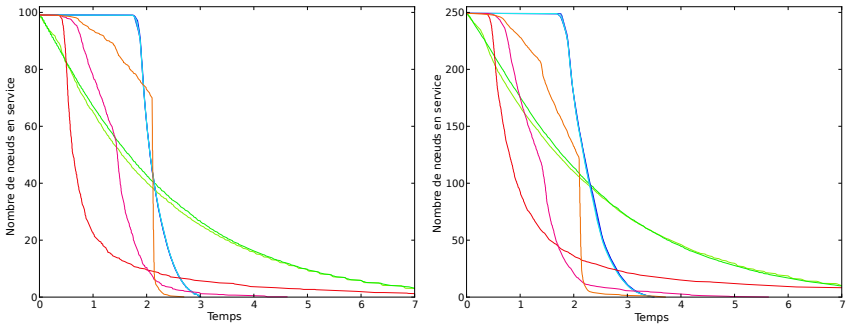
4.3 Conclusion

Dans ces travaux, nous nous sommes principalement intéressés à la façon de mettre en réseau des équipements embarqués dans un contexte de ville intelligente et connectée. Les problématiques premières qui ont été adressées sont le coût des équipements et les besoins en énergie de ces derniers. Nous avons proposé des solutions à deux scénarios particuliers, scénarios issus de souhaits industriels pour le développement durable dans les villes. Ces protocoles, en plus d'avoir été évalués sur simulateur ont également été éprouvés en environnement réel à l'aide de la plateforme IoT-LAB [9].

J'ai souhaité encourager Tony à ne pas offrir des solutions « fourre tout » qui se proposent, de façon utopique, d'être adaptées à tous les scénarios possibles. L'efficacité de nos propositions s'explique en partie par cette bonne adaptation aux scénarios visés.

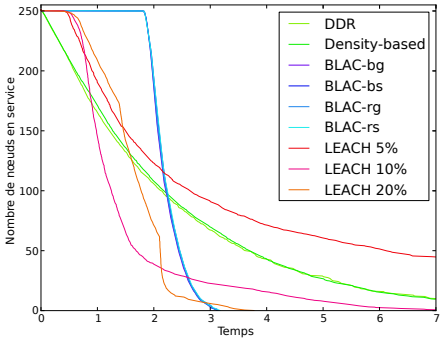
La thèse de Tony Ducrocq [T3] a été soutenue le 15 novembre 2013 et a mené à une publication dans une revue [J1], trois publications dans des conférences ou ateliers

9. Ex Senslab



(a) 100 nœuds en environnement urbain.

(b) 250 nœuds en environnement urbain.



(c) 250 nœuds en environnement rural.

FIGURE 4.8 – Durée de vie du réseau.

internationaux [C1, C2, C5], une publication dans une conférence française [C6] et un poster dans une conférence internationale [M1].

Les aspects liés aux recherches de ce chapitre que je pourrais souhaiter poursuivre sont principalement liés aux architectures système [10] qui permettent la mise en œuvre de ces algorithmes dont le but est de fonctionner sur des équipements fortement contraints en énergie, en puissance de calcul et en mémoire. Dans le cadre d'un projet précédent, le projet Européen FP6 WASP, nous avons proposé une architecture de développement de pile de communication [M3, M4] qui a permis de mettre en œuvre plusieurs protocoles réseaux, tant au niveau MAC qu'au niveau transport, de façon efficace, cette fois d'un point de vue système. Les travaux de ce projet mériteraient d'être rapprochés de ceux présentés dans ce chapitre pour leur implémentation en environnement réel.

Enfin, pour pouvoir être réellement déployés dans un contexte de ville intelligente, les protocoles de mise en réseaux se devront d'assurer la sécurité des transmissions, la continuité de services et le respect de la vie privée des utilisateurs. J'ai déjà étudié certains de ces aspects peu après ma thèse [C13, C12] mais de nombreuses voies restent ouvertes. Si tous ces problèmes sont déjà présents et difficiles dans l'univers des ordinateurs personnels, le caractère embarqué, omniprésent et sensible des équipements qui

10. au sens système d'exploitation

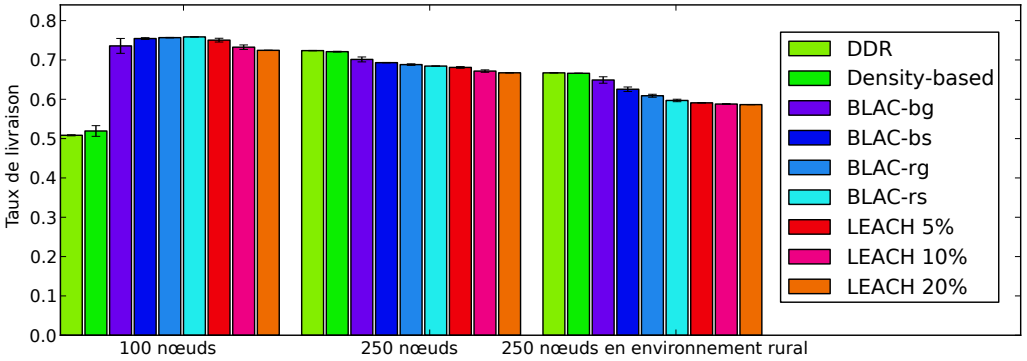


FIGURE 4.9 – Taux de livraison des données (1 = 100%).

nous intéresse empire la situation [9] et offre des perspectives de défis très stimulantes.

Chapitre 5

Conclusion

Ce document présente un extrait des recherches que j'ai menées dans les cinq dernières années. Durant cette période, j'ai pu encadrer quatre étudiants, d'abord en master pour trois d'entre eux, puis en thèse. Parmi ces thèses, trois sont co-encadrées en collaboration avec Gilles Grimaud et une avec Nathalie Mitton. Deux étudiants, Geoffroy Cogniaux et Tony Ducrocq, ont obtenu leur doctorat et ont pu le faire valoir pour leur vie professionnelle en intégrant des équipes de recherche et développement de grandes entreprises (Gemalto et Worldline). Damien Riquet et François Serman sont toujours en thèse dans l'équipe à l'heure où ces lignes sont écrites et je n'ai aucun doute quant à l'issue de leur travail.

L'écriture de ce document aura été pour moi l'occasion de dresser un bilan de ces années et du chemin parcouru entre l'étude de protocoles réseau, avec la théorie des graphes comme outil de modélisation et la simulation comme moyen de validation, et l'étude de la production concrète de logiciel pour des systèmes embarqués fortement contraints qui constituent une cible privilégiée pour l'utilisation des résultats de la communauté liée à mes travaux de thèse.

Une des originalités de mon travail est certainement dans la diversité des problèmes abordés et dans les différents outils scientifiques utilisés pour apporter des solutions à ces problèmes. Chacune des contributions que j'ai apportées utilisent des outils différents, comme l'adaptation d'environnement d'exécution ou l'utilisation de langage dédié, à une même problématique de fond : comment rendre plus abordable la conception d'un logiciel fiable et performant quand celui-ci est destiné à des cibles très contraintes et/ou très hétérogènes ?

Même si, parfois, cette volonté d'utiliser de nombreux outils, tant scientifiques que techniques, peut rendre un peu délicate la réutilisation des travaux précédents dans un nouveau contexte, c'est cette diversité et l'appréhension de ce vaste choix d'approches qui me motive et m'intéresse dans la recherche.

Le point commun de tous ces travaux, en dehors de leur applications à la production de systèmes embarqués, est la volonté de pousser une méthodologie basée sur l'observation du système et de l'état de l'art, leur compréhension et la recherche d'une bonne voie pouvant mener à des résultats probants.

5.1 Méthodologie

Dans les trois travaux présentés, j'ai poussé les étudiants à obtenir une compréhension fine des différents aspects liés à la problématique de leur thèse.

Concernant la thèse de Geoffroy Cogniaux, discutée dans le chapitre 2, il nous fallait comprendre à minima deux choses fondamentales :

— quel est le schéma d'accès mémoire d'un micro-contrôleur pour l'exécution de code embarqué dans le cas du code natif ou du code interprété ? En particulier, comment sont utilisées les méta-données associées au bytecode interprété ?

— comment se comporte une mémoire cache quand on ne dispose pas de support matériel à son implémentation et que la mémoire disponible pour le cache est si faible que les structures de données liées à sa gestion ne sont pas de taille négligeable par rapport aux données gérées par cette mémoire cache ?

Les réponses apportées à ces questions par les études poussées menées par Geoffroy, et surtout la surprise apportées pas ces réponses, lui ont permis de concevoir son adaptation de la machine virtuelle Java et JavaCard et d'offrir une solution à l'exécution depuis une mémoire série qui est non seulement possible, mais également efficace. Ce n'était absolument pas la voie que nous avions imaginé prendre au début de sa thèse et l'issue en est d'autant plus gratifiante.

Dans le cas de la thèse de Tony Ducrocq, c'est l'étude des particularités de l'application, et de pourquoi elle rendait les solutions de la littérature inefficaces, qui lui a permis de proposer des solutions adaptées. C'est également grâce à la mise en relation de simulation et de l'expérimentation sur plateforme réelle que ces résultats ont trouvé leur justification.

Enfin, Damien Riquet a cherché à comprendre les spécificités des nouvelles architectures des centres de données et quelles en étaient les possibilités d'attaque pour proposer DISCUS. Les possibilités offertes par son langage ont été élaborées à partir d'une analyse des besoins des acteurs qui définissent des politiques de sécurité et des algorithmes de détection d'intrusions.

5.2 Perspectives

Les perspectives spécifiques, associées à chacune des contributions, ont déjà été évoquées dans les chapitres consacrés. Cependant, ces perspectives, si elles sont intéressantes, ne reflètent pas nécessairement les voies que je souhaite poursuivre directement. Je présente ici, les deux pistes qui me tiennent vraiment à cœur et pour lesquelles je compte continuer ou lancer de nouveaux travaux.

DISCUS la thèse de Damien Riquet nous a permis de définir une architecture de sondes de détection d'intrusions distribuées ainsi qu'un langage et une suite d'outils permettant la production du logiciel de ces sondes. Le mécanisme support de collaboration des sondes est une base de données distribuée leur permettant de partager l'état du système global. L'implémentation actuelle de cette base, qui sera utilisée pour le premier prototype et la validation de la démarche, est encore naïve. La distribution de données

à une multitude de sondes doit être tolérante aux fautes et aux attaques essayant d'induire le système en erreur mais doit également assurer la cohérence des données et être efficace et utilisable malgré les faibles ressources d'une partie des équipements de l'infrastructure de détection. Si les technologies pair à pair, comme les tables de hachages distribuées, peuvent constituer une première piste, la forte hétérogénéité des sondes, le volume de données qu'elles seront amenées à échanger et la nécessité d'être robuste à la compromission d'une partie des sondes empêche la simple réutilisation de la littérature.

L'élaboration des sondes matérielles à base de FPGA représente également un défi important. Plus qu'une simple traduction du langage vers un pseudo processeur générique, c'est bien un nouveau type d'architecture dédié qu'il faudra mettre en place si l'on veut pouvoir traiter les données à une vitesse de l'ordre du gigabit par seconde à l'aide d'une sonde de moins de cent euros. La mise en œuvre de la base de données distribuée à l'aide d'un circuit dédié est également à penser de façon transversale. Le système produit devra être capable de se mettre à jour partiellement afin de mettre à jour les politiques de sécurité en limitant l'interruption de surveillance au maximum. Cette mise à jour devra aussi se faire de façon sûre.

Il est donc aisé de voir que la thèse de Damien Riquet pose des bases solides à DISCUS mais que de nombreuses problématiques de recherches sont encore en suspens. Je suis donc très confiant sur la possibilité d'encadrer de nouveaux étudiants sur ces sujets et de poursuivre le travail commencé avec Damien.

Sécurité des systèmes contraints J'ai choisi délibérément de ne pas présenter de façon détaillée dans ce document le travail effectué avec François Serman qui en est encore à ses débuts. Cependant, il serait bien malvenu de ma part de l'occulter complètement tant il m'emmène vers un monde fascinant à la frontière entre la conception d'hyperviseur et la preuve de programme.

La thèse de François est effectuée en contrat CIFRE avec l'entreprise Prove & Run. L'objectif de cette thèse est de rapprocher le monde des méthodes formelles, et en particulier de la preuve de programme, avec celui de la conception de système. La multiplication de failles dans les différents systèmes traditionnels (qu'ils soient de bureau ou destinés à être support d'exécution de machines virtuelles) ont encouragé le développement de systèmes cherchant à valider des niveaux de certification de plus en plus élevés. Le niveau ultime étant la production d'un système entièrement développé et prouvé à l'aide de méthodes formelles.

La récente mise en open source de seL4 [4] a permis de montrer que la production d'un micro-noyau dont certaines garanties sont appuyées par une preuve formelle était possible. Cependant, ces garanties sont encore limitées. Par exemple, seL4 apporte la preuve que la mémoire utilisée par le noyau du système est isolé est donc inaccessible depuis les processus exécutés. Aucune garantie n'est apportée sur l'isolation des processus entre eux. Ceci s'explique principalement par la difficulté de raisonner et d'établir des preuves, même sur des modèles simples. Ce constat est également vrai en ce qui concerne le matériel. Les preuves d'isolation s'appuient sur des mécanismes matériels comme la MMU, mais ce matériel n'est pas prouvé au sens où l'on a pas de preuve formelle du bon fonctionnement de ce dernier.

Pour cette raison, et également car une partie de nos cibles privilégiées ne disposent même pas de ce support matériel, nous avons choisi de proposer un modèle d'hypervision original dans lequel l'hyperviseur et les systèmes invités fonctionnent avec le même niveau de privilège matériel. L'hyperviseur fonctionne par une pré-analyse du code invité permettant de détecter les instructions susceptibles de compromettre l'hyperviseur (accès mémoire, configuration de périphérique,...). Outre le fait de s'affranchir du besoin de support matériel à la virtualisation (ni MMU, ni instruction spécifiques), cette technique permet de laisser s'exécuter le code du monde utilisateur (les processus non-privilégiés des systèmes invités) sans aucun contrôle par l'hyperviseur, et donc avec des performances quasi équivalentes à un système natif.

Nous nous proposons alors d'établir des preuves de bon fonctionnement de notre mécanisme d'hypervision, et en particulier que sa pré-analyse n'omet aucune instruction succeptibles d'altérer son fonctionnement.

5.3 Épilogue

Dix années se sont écoulées depuis que j'ai terminé ma thèse. J'ai passé une bonne partie de ces dix années à essayer de transmettre ma passion de notre domaine, de notre métier, tant aux doctorants que j'ai eu le plaisir d'encadrer, qu'aux nombreux étudiants de premier et deuxième cycle que j'ai vu passer dans mes cours.

La thèse (et la recherche en général) est, et sera toujours, une aventure, différente pour chacun des doctorants. On peut cependant dégager un caractère commun. On sait au départ où on veut aller et on pense connaître le chemin à emprunter. Ce qu'on sait moins, et qu'on mesure avec l'expérience, c'est qu'on prend rarement le chemin prévu et qu'il arrive même fréquemment qu'on n'arrive pas non plus à la destination escomptée, bien qu'également intéressante. J'aime insister sur ce point quand je parle de la recherche aux étudiants afin qu'ils mesurent la difficulté de s'engager dans la préparation d'une thèse.

Heureusement, plus que le résultat en lui même, c'est justement le chemin qui rend l'aventure passionnante. Une carrière de chercheur, c'est avant tout des discussions, des réflexions, des doutes, des échecs et, parfois, de jolis succès.

L'important, à mon sens, dans l'encadrement d'une thèse et donc bien évidemment d'insuffler la direction à suivre et d'apporter un regard critique sur les idées, mais surtout d'accompagner le doctorant sur le plan professionnel, mais aussi personnel, pour qu'il puisse développer le recul et les pratiques nécessaires à l'élaboration de solutions qui lui sont propres.

Bibliographie personnelle

Cette section reprend les références bibliographiques des travaux effectués après ma thèse. Les entrées concernant les thèses encadrées indiquent si la thèse est soutenue ou en cours.

Thèses encadrées

[T1] D. Riquet. « DISCUS : une architecture dédiée à la protection réseau ». Co-encadrée avec le Pr. Gilles Grimaud. En cours. Thèse de doct. Université Lille 1.

[T2] F. Serman. « Études des élements architecturaux d'un micro-noyeau formellement prouvé ». Co-encadrée avec le Pr. Gilles Grimaud. En cours. Thèse de doct. Université Lille 1.

[T3] T. Ducrocq. « Auto organisation des réseaux sans-fil multi-sauts dans les villes intelligentes ». Co-encadrée avec Nathalie Mitton. Soutenue. Thèse de doct. Université Lille 1, nov. 2013.

[T4] G. Cogniaux. « Exécution d'applications stockées dans la mémoire non-adressable d'une carte à puce ». Co-encadrée avec le Pr. Gilles Grimaud. Soutenue. Thèse de doct. Université Lille 1, déc. 2012.

Journal

[J1] T. Ducrocq, M. Hauspie et N. Mitton. « Balancing energy consumption in clustered wireless sensor networks ». In : *ISRN Sensor Networks* (oct. 2013). url : `http://hal.inria.fr/hal-00863123`.

Conférences et ateliers avec comité de lecture

[C1] T. Ducrocq, M. Hauspie et N. Mitton. « Geographic Routing with Partial Position Information ». In : *Proceedings of the third internation conference on sensor networks (SENSORNETS)*. Lisbon, Portugal, jan. 2014.

[C2] T. Ducrocq, M. Hauspie, N. Mitton et S. Pizzi. « On the impact of network topology on wireless sensor networks performances ». In : *Proceedings of The nineth Workshop on Performance Analysis and Enhancement of Wireless Networks (PAEWN'2014)*. Victoria, Canada, mai 2014.

[C3] D. Riquet, G. Grimaud et M. Hauspie. « DISCUS : A massively distributed IDS architecture using a DSL-based configuration ». In : *Proceedings of the 2014 International Conference on Information Science, Electronics and Electrical Engineering (ISEEE'2014)*. Sapporo City, Hokkaido, Japan, 2014.

[C4] D. Riquet, G. Grimaud et M. Hauspie. « Un langage pour la configuration de DISCUS, une architecture de solutions de sécurité ». In : *Conférence en Parallélisme, Architecture et Systèmes (ComPAS)*. Neuchâtel, Suisse, avr. 2014.

[C5] T. Ducrocq, N. Mitton et M. Hauspie. « Energy-based Clustering for Wireless Sensor Network Lifetime Optimization ». In : *Proceedings of IEEE WCNC - Wireless Communications and Networking Conference*. Shanghai, China, avr. 2013. URL : http://hal.inria.fr/hal-00767690.

[C6] T. Ducrocq, N. Mitton et M. Hauspie. « Clustering pour l'optimisation de la durée de vie des réseaux de capteurs sans fil ». In : *14èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications (AlgoTel)*. La Grande Motte, France, 2012.

[C7] D. Riquet, G. Grimaud et M. Hauspie. « Etude de l'impact des attaques distribuées et multi-chemins sur les solutions de sécurité réseaux ». In : *MajecSTIC 2012, 9ème Conférence Internationale Jeunes Chercheurs*. 2012. URL : http://hal.archives-ouvertes.fr/hal-00746991.

[C8] D. Riquet, G. Grimaud et M. Hauspie. « Large-scale coordinated attacks : Impact on the cloud security ». In : *Proceedings of The Second International Workshop on Mobile Commerce, Cloud Computing, Network and Communication Security 2012 (MCNCS 2012)*. Palermo, Italy, juin 2012.

[C9] G. Cogniaux, M. Hauspie et F.-X. Marseille. « Étude Préliminaire À Une Utilisation De Mémoires Secondaires Pour Le Stockage Des Métadonnées Java Dans Des Systèmes Contraints ». In : *Conférence Française en Système d'Exploitations*. Saint-Malo, France, mai 2011.

[C10] G. Cogniaux et G. Grimaud. « Impact of Pages Sizes to Execute Code Using Demand Paging and NAND Flash at Smart Card Scale ». In : *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010)*. CISIS '10. Washington, DC, USA : IEEE Computer Society, 2010, p. 794–799. ISBN : 978-0-7695-3967-6. DOI : 10.1109/CISIS.2010.175.

[C11] Geoffroy Cogniaux et Gilles Grimaud. « Key-study to Execute Code Using Demand Paging and NAND Flash at Smart Card Scale ». Anglais. In : *The ninth Smart Card Research and Advanced Application IFIP Conference - CARDIS 2010*. Sous la dir. de Springer) Lecture Notes of Computer Science

(LNCS 6035. XLIM. Passau, Allemagne, avr. 2010. URL : `http://hal.inria.fr/inria-00529489`.

[C12] M. Hauspie et I. Simplot-Ryl. « Enhancing nodes cooperation in ad hoc networks ». In : *Proceedings of the 4th IEEE/IFIP Annual Conference on Wireless On demand Network Systems and Services (WONS 2007)*. Obergurgl, Austria : IEEE Press, jan. 2007.

[C13] M. Hauspie et I. Simplot-Ryl. « Cooperation in ad hoc networks : Enhancing the virtual currency based models ». In : *Proceedings of the 1st ACM International Conference on Integrated Internet Ad hoc and Sensor Networks (InterSense 2006)*. Nice, France, mai 2006.

Chapitres de livres

[B1] G. Grimaud et M. Hauspie. *Les cartes à puce*. Sous la dir. de S. Bouzefrane et P. Paradinas. Hermes, 2013. Chap. Les systèmes d'exploitation de la carte à microprocesseur. URL : `http://editions.lavoisier.fr/notice.asp?ouvrage=2745320`.

[B2] J. Carle, M. Hauspie, N. Mitton, T. Razafindralambo et D. Simplot-Ryl. *Informatique et intelligence ambiante : des capteurs aux applications*. Sous la dir. de G. Calvary, T. Delot, F. Sèdes et J.-Y. Tigli. Hermes, 2012. Chap. Les réseaux de capteurs. URL : `http://www.lavoisier.fr/livre/notice.asp?ouvrage=2139894`.

Autres

[M1] T. Ducrocq, M. Hauspie et N. Mitton. *On the Impact of Network Topology on Wireless Sensor Networks Performances - Illustration with Geographic Routing*. Poster in the Tenth ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2013). Nov. 2013. URL : `http://hal.inria.fr/hal-00850924`.

[M2] D. Riquet. *Massively Distributed Intrusion Detection Systems*. Poster session, The European Professional Society on Computer Science, EuroSys 2012. 2012.

[M3] M. Hellenschmidt, C. Stocklöw, R. Serna Olivera, I. Shcherbakov, A. Lachenmann, L. Gomez, M. Hauspie, D. Blasi, M. Schuster, A. Kovacevic, A. Fraboulet, M. Aoun, M. Bennebroek, J.-D. Decotignie, L. von Allmen et R. Verhoeven. *WASP Deliverable D7.4 : WASP development handbook*. EU FP6 IST, 2010.

[M4] M. Andree, H. Karl, M. Herlich, J. Catalano, A. Schoofs, P. van der Stok, L. Vanzago, L. von Allmen, R. Serna Olivera, G. Fohler, C. Brandolese, M. Hauspie, G. Grimaud, S. Buisine, E. Fleury, A. Fraboulet, A. Picu et F. Bouwens. *WASP Deliverable D3.2 : Core hardware abstraction and programming model*. EU FP6 IST, 2008.

[M5] M. Hauspie et I. Simplot-Ryl. *How to negociate the price of packet forwarding in credit-based systems for ad-hoc networks.* Invited talk. Mexico-City, Mexico, 2005.

Références

[1] Internet Security Systems' X-Force. « Backdoors and Trojan Horses ». In : *Information Security Technical Report* 6.4 (2001), p. 31–57. ISSN : 1363-4127. DOI : http://dx.doi.org/10.1016/S1363-4127(01)00405-8. URL : http://www.sciencedirect.com/science/article/pii/S1363412701004058.

[2] CVE Editorial board, éd. *CVE-2014-0160 : Heartbleed bug*. 2014. URL : http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[3] Andrei Costin, Jonas Zaddach, Aurélien Francillon et Davide Balzarotti. « A Large-Scale Analysis of the Security of Embedded Firmwares ». In : *Proceedings of the 23rd USENIX Security Symposium*. Sous la dir. d'USENIX. San Diego, CA, 2014, p. 95–110.

[4] Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski et Gernot Heiser. « Comprehensive Formal Verification of an OS Microkernel ». In : *ACM Transactions on Computer Systems* 32.1 (fév. 2014), 2 :1–2 :70. DOI : 10.1145/2560537.

[5] Slashdot. *Reported iCloud Hack Leaks Hundreds of Private Celebrity Photos*. Sept. 2014. URL : http://apple.slashdot.org/story/14/09/01/1515236/reported-icloud-hack-leaks-hundreds-of-private-celebrity-photos.

[6] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin et Kuang-Yuan Tung. « Intrusion detection system : A comprehensive review ». In : *Journal of Network and Computer Applications* 36.1 (2013), p. 16–24. ISSN : 1084-8045. DOI : http://dx.doi.org/10.1016/j.jnca.2012.09.004. URL : http://www.sciencedirect.com/science/article/pii/S1084804512001944.

[7] Open Street Map Foundation. *Open Street Map*. 2013. URL : http://www.openstreetmap.org.

[8] Ahmed Patel, Mona Taghavi, Kaveh Bakhtiyari et Joaquim Celestino Júnior. « An intrusion detection and prevention system in cloud computing : A systematic review ». In : *Journal of Network and Computer Applications* 36.1 (2013), p. 25–41. ISSN : 1084-8045. DOI : http://dx.doi.org/10.1016/j.jnca.2012.08.007. URL : http://www.sciencedirect.com/science/article/pii/S108480451200183X.

[9] Dimitrios N Serpanos et Artemios G Voyiatzis. « Security challenges in embedded systems ». In : *ACM Transactions on Embedded Computing Systems (TECS)* 12.1s (2013), p. 66.

[10] Carlos A. Catania et Carlos García Garino. « Automatic network intrusion detection : Current techniques and open issues ». In : *Computers & Electrical Engineering* 38.5 (2012). Special issue on Recent Advances in Security and Privacy in Distributed Communications and Image processing, p. 1062–1072. issn : 0045-7906. doi : http://dx.doi.org/10.1016/j.compeleceng.2012.05.013. url : http://www.sciencedirect.com/science/article/pii/S0045790612001073.

[11] Julien Mercadal, Laurent Réveillère, Yérom-David Bromberg, Bertrand Le Gal, Tegawendé F. Bissyandé et Jigar Solanki. « Zebra : Building Efficient Network Message Parsers for Embedded Systems ». In : *Embedded Systems Letters* PP.99 (juil. 2012). 4 pages, p. 1–4. doi : 10.1109/LES.2012.2208617. url : http://hal.archives-ouvertes.fr/hal-00730930.

[12] Laurent Burgy, Laurent Réveillère, Julia Lawall L. et G. Muller. « Zebu : A Language-Based Approach for Network Protocol Message Processing ». In : *IEEE Transactions on Software Engineering* 37.4 (2011), p. 575–591. url : http://hal.archives-ouvertes.fr/hal-00814448.

[13] ORACLE. *Java Card 3 Platform*. 2011.

[14] D. N. M. Bryan et M. Anderson. *Cloud Computing, a Weapon of Mass Destruction ?* 2010. url : https://www.defcon.org/html/links/dc-archives/dc-18-archive.html.

[15] Xiang Min, Shi Wei-ren, Jiang Chang-jiang et Zhang Ying. « Energy efficient clustering algorithm for maximizing lifetime of wireless sensor networks ». In : *{AEU} - International Journal of Electronics and Communications* 64.4 (2010), p. 289–298. issn : 1434-8411. doi : http://dx.doi.org/10.1016/j.aeue.2009.01.004. url : http://www.sciencedirect.com/science/article/pii/S1434841109000454.

[16] Spiceworks. *New Study Sees Rise in Cloud Services Adoption Among Small and Medium Businesses in First Half of 2010*. 2010. url : http://www.spiceworks.com/news/press-release/2010/07-28.php.

[17] Chenfeng Vincent Zhou, Christopher Leckie et Shanika Karunasekera. « A survey of coordinated attacks and collaborative intrusion detection ». In : *Computers & Security* 29.1 (2010), p. 124–140. issn : 0167-4048. doi : http://dx.doi.org/10.1016/j.cose.2009.06.008. url : http://www.sciencedirect.com/science/article/pii/S016740480900073X.

[18] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin et Matei Zaharia. *Above the Clouds : A Berkeley View of Cloud Computing*. Rapp. tech. 2009.

[19] European Network and Information Security Agency. *Cloud Computing Risk Assessment*. Rapp. tech. 2009.

[20] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández et E. Vázquez. « Anomaly-based network intrusion detection : Techniques, systems and challenges ». In : *Computers & Security* 28.1–2 (2009), p. 18–28. ISSN : 0167-4048. DOI : http://dx.doi.org/10.1016/j.cose.2008.08.003. URL : http://www.sciencedirect.com/science/article/pii/S0167404808000692.

[21] Stephen Shankland. *HP's Hurd dings cloud computing, IBM*. 2009. URL : http://news.cnet.com/8301-30685_3-10378781-264.html.

[22] Georgios Kambourakis, Tassos Moschos, Dimitris Geneiatakis et Stefanos Gritzalis. « Detecting DNS Amplification Attacks ». English. In : *Critical Information Infrastructures Security*. Sous la dir. de Javier Lopez et BernhardM. Hämmerli. T. 5141. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, p. 185–196. ISBN : 978-3-540-89095-9. DOI : 10.1007/978-3-540-89173-4_16. URL : http://dx.doi.org/10.1007/978-3-540-89173-4_16.

[23] Chor Ping Low, Can Fang, Jim Mee Ng et Yew Hock Ang. « Efficient Load-Balanced Clustering Algorithms for wireless sensor networks ». In : *Computer Communications* 31.4 (2008). Algorithmic and Theoretical Aspects of Wireless ad hoc and Sensor Networks, p. 750–759. ISSN : 0140-3664. DOI : http://dx.doi.org/10.1016/j.comcom.2007.10.020. URL : http://www.sciencedirect.com/science/article/pii/S0140366407004264.

[24] Ameer Ahmed Abbasi et Mohamed Younis. « A survey on clustering algorithms for wireless sensor networks ». In : *Computer Communications* 30.14–15 (2007). Network Coverage and Routing Schemes for Wireless Sensor Networks, p. 2826–2841. ISSN : 0140-3664. DOI : http://dx.doi.org/10.1016/j.comcom.2007.05.024. URL : http://www.sciencedirect.com/science/article/pii/S0140366407002162.

[25] Mark Allman, Vern Paxson et Jeff Terrell. « A brief history of scanning ». In : *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. IMC '07. San Diego, California, USA : ACM, 2007, p. 77–82. ISBN : 978-1-59593-908-1. DOI : 10.1145/1298306.1298316. URL : http://doi.acm.org/10.1145/1298306.1298316.

[26] J. B. Grizzard, V. Sharma, C. Nunnery et B. B. Kang. « Peer-to-Peer Botnets : Overview and Case Study ». In : *Proceedings of the first Workshop on Hot Topics in Understanding Botnets (HotBots07)*. Sous la dir. d'USENIX. 2007.

[27] J. Grossman, R. Hansen, P. Petkov, A. Rager et S. Fogie. *XSS Attacks : Cross-site scripting exploits and defense*. Sous la dir. d'Elsevier. Syngress, 2007.

[28] Andreas LACHENMANN, Pedro José MARRÓN, Matthias GAUGER, Daniel MINDER, Olga SAUKH et Kurt ROTHERMEL. « Removing the memory limitations of sensor networks with flash-based virtual memory ». In : *Proceedings of the 2nd ACM SIGOPS EuroSys 2007*. Lisbon, Portugal, 2007, p. 131–144.

[29] Guillaume CHELIUS, Antoine FRABOULET et Eric FLEURY. « WorldSens : system tools for embedded sensor networks ». Anglais. In : *Real-Time Systems Symposium (RTSS 2006)*. Brésil, 2006, 4, WIP Session. URL : http://hal.archives-ouvertes.fr/hal-00399617.

[30] James COWLING, Daniel MYERS, Barbara LISKOV, Rodrigo RODRIGUES et Liuba SHRIRA. « HQ replication : A hybrid quorum protocol for Byzantine fault tolerance ». In : *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association. 2006, p. 177–190.

[31] D. SIMON, C. CIFUENTES, D. CLEAL, J. DANIELS et D. WHITE. « Java™ on the Bare Metal of Wireless Sensor Devices : The Squawk Java Virtual Machine ». In : *Proceedings of the 2Nd International Conference on Virtual Execution Environments*. VEE '06. Ottawa, Ontario, Canada : ACM, 2006, p. 78–88. ISBN : 1-59593-332-8. DOI : 10.1145/1134760.1134773. URL : http://doi.acm.org/10.1145/1134760.1134773.

[32] Doug SIMON, Cristina CIFUENTES, Dave CLEAL, John DANIELS et Derek WHITE. « Java on the bare metal of wireless sensor devices : the squawk Java virtual machine ». In : *Proceedings of the 2nd international conference on Virtual execution environments*. VEE '06. Ottawa, Ontario, Canada : ACM, 2006, p. 78–88. ISBN : 1-59593-332-8. DOI : 10.1145/1134760.1134773. URL : http://doi.acm.org/10.1145/1134760.1134773.

[33] Vasilios A. SIRIS et Fotini PAPAGALOU. « Application of anomaly detection algorithms for detecting {SYN} flooding attacks ». In : *Computer Communications* 29.9 (2006). {ICON} 2004 12th {IEEE} International Conference on Network 2004, p. 1433–1442. ISSN : 0140-3664. DOI : http://dx.doi.org/10.1016/j.comcom.2005.09.008. URL : http://www.sciencedirect.com/science/article/pii/S0140366405003531.

[34] O. YOUNIS, M. KRUNZ et S. RAMASUBRAMANIAN. « Node clustering in wireless sensor networks : recent developments and deployment challenges ». In : *Network, IEEE* 20.3 (mai 2006), p. 20–25. ISSN : 0890-8044. DOI : 10.1109/MNET.2006.1637928.

[35] Michael ABD-EL-MALEK, Gregory R GANGER, Garth R GOODSON, Michael K REITER et Jay J WYLIE. « Fault-scalable Byzantine fault-tolerant services ». In : *ACM SIGOPS Operating Systems Review* 39.5 (2005), p. 59–74.

[36] A CARUSO, S. CHESSA, S. DE et A URPI. « GPS free coordinate assignment and routing in wireless sensor networks ». In : *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*. T. 1. Mar. 2005, 150–160 vol. 1. DOI : 10.1109/INFCOM.2005.1497887.

[37] Erwan Ermel, Anne Fladenmuller, Guy Pujolle et André Cotton. « On Selecting Nodes to Improve Estimated Positions ». English. In : *Mobile and Wireless Communication Networks*. Sous la dir. d'ElizabethM. Belding-Royer, Khaldoun Al Agha et Guy Pujolle. T. 162. IFIP International Federation for Information Processing. Springer US, 2005, p. 449–460. isbn : 978-0-387-23148-8. doi : 10.1007/0-387-23150-1_38. url : http://dx.doi.org/10.1007/0-387-23150-1_38.

[38] Michael E Locasto, Janak J Parekh, Angelos D Keromytis et Salvatore J Stolfo. « Towards collaborative security and p2p intrusion detection ». In : *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*. IEEE. 2005, p. 333–339.

[39] N. Mitton, E. Fleury, IG. Lassous et S. Tixeuil. « Self-stabilization in self-organized multihop wireless networks ». In : *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*. Juin 2005, p. 909–915. doi : 10.1109/ICDCSW.2005.122.

[40] G. Muller, J. L. Lawall et H. Duchesne. « A framework for simplifying the development of kernel schedulers : design and performance evaluation ». In : *Proceedings of Hgih Assurance Systems Engineering Conference*. Heidelberg, Germany, 2005.

[41] Tao Shu, Marwan Krunz et Sarma Vrudhula. « Power Balanced Coverage-time Optimization for Clustered Wireless Sensor Networks ». In : *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. MobiHoc '05. Urbana-Champaign, IL, USA : ACM, 2005, p. 111–120. isbn : 1-59593-004-3. doi : 10.1145/1062689.1062705. url : http://doi.acm.org/10.1145/1062689.1062705.

[42] Yingwei Yao et G.B. Giannakis. « Energy-Efficient Scheduling for Wireless Sensor Networks ». In : *Communications, IEEE Transactions on* 53.8 (août 2005), p. 1333–1342. issn : 0090-6778. doi : 10.1109/TCOMM.2005.852834.

[43] Sorav Bansal et Dharmendra S. Modha. « CAR : Clock with Adaptive Replacement ». In : *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. FAST '04. San Francisco, CA : USENIX Association, 2004, p. 187–200. url : http://dl.acm.org/citation.cfm?id=1096673.1096699.

[44] Ramana Rao Kompella, Sumeet Singh et George Varghese. « On Scalable Attack Detection in the Network ». In : *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*. IMC '04. Taormina, Sicily, Italy : ACM, 2004, p. 187–200. isbn : 1-58113-821-0. doi : 10.1145/1028788.1028812. url : http://doi.acm.org/10.1145/1028788.1028812.

[45] Karim Seada, Ahmed Helmy et Ramesh Govindan. « On the Effect of Localization Errors on Geographic Face Routing in Sensor Networks ». In : *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks*. IPSN '04. Berkeley, California, USA : ACM, 2004, p. 71–80. isbn :

1-58113-846-6. DOI : 10.1145/984622.984633. URL : http://doi.acm.org/10.1145/984622.984633.

[46] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph et John D Kubiatowicz. « Tapestry : A resilient global-scale overlay for service deployment ». In : *Selected Areas in Communications, IEEE Journal on* 22.1 (2004), p. 41–53.

[47] T. Clausen et P. Jacquet. *RFC 3626 : Optimized Link State Routing Protocol (OLSR)*. IETF. 2003. URL : http://www.ietf.org/rfc/rfc3626.txt.

[48] JCRE.2.2.1. *JAVA CARD Runtime Environment Specification. Sun Microsystems, Inc.* 2003.

[49] JCVM.2.2.1. *JAVA CARD 2.2.1 Virtual Machine Specification. Sun Microsystems, Inc.* 2003.

[50] Nimrod Megiddo et Dharmendra S. Modha. « ARC : A Self-Tuning, Low Overhead Replacement Cache ». In : *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*. FAST '03. San Francisco, CA : USENIX Association, 2003, p. 115–130. URL : http://dl.acm.org/citation.cfm?id=1090694.1090708.

[51] C. Perkins, E. Belding-Royer et S. Das. *RFC 3561 : Ad hoc On-Demand Distance Vector (AODV) Routing*. IETF. 2003. URL : http://www.ietf.org/rfc/rfc3561.txt.

[52] Nicholas Weaver, Vern Paxson, Stuart Staniford et Robert Cunningham. « A Taxonomy of Computer Worms ». In : *Proceedings of the 2003 ACM Workshop on Rapid Malcode*. WORM '03. Washington, DC, USA : ACM, 2003, p. 11–18. ISBN : 1-58113-785-0. DOI : 10.1145/948187.948190. URL : http://doi.acm.org/10.1145/948187.948190.

[53] S.A Borbash et E.H. Jennings. « Distributed topology control algorithm for multihop wireless networks ». In : *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*. T. 1. 2002, p. 355–360. DOI : 10.1109/IJCNN.2002.1005497.

[54] Srdjan Čapkun, Maher Hamdi et Jean-Pierre Hubaux. « GPS-free Positioning in Mobile Ad Hoc Networks ». English. In : *Cluster Computing* 5.2 (2002), p. 157–167. ISSN : 1386-7857. DOI : 10.1023/A:1013933626682. URL : http://dx.doi.org/10.1023/A:1013933626682.

[55] Mainak Chatterjee, SajalK. Das et Damla Turgut. « WCA : A Weighted Clustering Algorithm for Mobile Ad Hoc Networks ». English. In : *Cluster Computing* 5.2 (2002), p. 193–204. ISSN : 1386-7857. DOI : 10.1023/A:1013941929408. URL : http://dx.doi.org/10.1023/A:1013941929408.

[56] I Stojmenovic. « Position-based routing in ad hoc networks ». In : *Communications Magazine, IEEE* 40.7 (juil. 2002), p. 128–134. ISSN : 0163-6804. DOI : 10.1109/MCOM.2002.1018018.

[57] Haining Wang, Danlu Zhang et K.G. Shin. « Detecting SYN flooding attacks ». In : *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. T. 3. Juin 2002, p. 1530–1539. DOI : 10.1109/INFCOM.2002.1019404.

[58] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge et R. B. Brown. « MiBench : A free, commercially representative embedded benchmark suite ». In : *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization.* Austin, TX, USA : IEEE, 2001, p. 3–14. ISBN : 0-7803-7315-4. DOI : 10.1109/WWC.2001.990739. URL : http://dx.doi.org/10.1109/WWC.2001.990739.

[59] P. Jacquet, P. Muhlethaler, T. Clausen, A Laouiti, A Qayyum et L. Viennot. « Optimized link state routing protocol for ad hoc networks ». In : *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*. 2001, p. 62–68. DOI : 10.1109/INMIC.2001.995315.

[60] Antony Rowstron et Peter Druschel. « Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems ». In : *Middleware 2001*. Springer. 2001, p. 329–350.

[61] Mahtab Seddigh, Julio Solano González et Ivan Stojmenovic. « RNG and Internal Node Based Broadcasting Algorithms for Wireless One-to-one Networks ». In : *SIGMOBILE Mob. Comput. Commun. Rev.* 5.2 (avr. 2001), p. 37–44. ISSN : 1559-1662. DOI : 10.1145/584066.584069. URL : http://doi.acm.org/10.1145/584066.584069.

[62] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek et Hari Balakrishnan. « Chord : A scalable peer-to-peer lookup service for internet applications ». In : *ACM SIGCOMM Computer Communication Review* 31.4 (2001), p. 149–160.

[63] Andrew S. Tanenbaum. *Modern Operating Systems*. 2nd. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2001. ISBN : 0130313580.

[64] Frédéric Cuppens et Rodolphe Ortalo. « LAMBDA : A Language to Model a Database for Detection of Attacks ». English. In : *Recent Advances in Intrusion Detection*. Sous la dir. d'Hervé Debar, Ludovic Mé et S.Felix Wu. T. 1907. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, p. 197–216. ISBN : 978-3-540-41085-0. DOI : 10.1007/3-540-39945-3_13. URL : http://dx.doi.org/10.1007/3-540-39945-3_13.

[65] W.R. Heinzelman, A. Chandrakasan et H. Balakrishnan. « Energy-efficient communication protocol for wireless microsensor networks ». In : *Proceedings of the 33rd Hawaii International Conference on System Sciences*. HICSS. Maui, HI, USA : Computer Society Press, 2000, p. 3005–3014. DOI : 10.1109/HICSS.2000.926982.

[66] F. Mérillon, L. Réveillère, C. Consel, R. Marlet et G. Muller. « Devil : And IDL for hardware programming ». In : *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*. San Diego, California, 2000, p. 17–30.

[67] Motorola, éd. *SPI Block Guide*. 2000.

[68] N. Nikaein, H. Labiod et C. Bonnet. « DDR-Distributed Dynamic Routing algorithm for mobile ad hoc networks ». In : *Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*. MobiHoc. Boston, MA, USA : ACM, 2000, p. 19–27. doi : 10.1109/MOBHOC.2000.869209.

[69] H. Debar, M. Dacier et A. Wespi. « Towards a taxonomy of intrusion-detection systems ». In : *Computer Networks* (1999).

[70] Evangelos Kranakis, Harvinder Singh et Jorge Urrutia. « Compass Routing on Geometric Networks ». In : *IN PROC. 11 TH CANADIAN CONFERENCE ON COMPUTATIONAL GEOMETRY*. 1999, p. 51–54.

[71] C.E. Perkins et E.M. Royer. « Ad-hoc on-demand distance vector routing ». In : *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*. Fév. 1999, p. 90–100. doi : 10.1109/MCSA.1999.749281.

[72] M. Roesch et al. « Snort-lightweight intrusion detection for networks ». In : *Proceedings of the 13th USENIX conference on System administration*. Seattle, Washington. 1999, p. 229–238.

[73] Doug Simon, Doug Simon, Antero Taivalsaari, Antero Taivalsaari, Antero Taivalsaari, Bill Bush, Bill Bush et Bill Bush. *The Spotless System : Implementing a Java System for the Palm Connected Organizer*. Rapp. tech. Sun Microsystems, Inc, 1999.

[74] Leslie Lamport. « The part-time parliament ». In : *ACM Transaction on Computer Systems* 16.2 (1998), p. 133–169.

[75] S. D. Brown, R. J. Francis, J. Rose et Z. G. Vranesic. *Field-Programmable Gate Arrays*. 1997. isbn : 9780792392484.

[76] Stuart Staniford-Chen, Steven Cheung, Richard Crawford, Mark Dilger, Jeremy Frank, James Hoagland, Karl Levitt, Christopher Wee, Raymond Yip et Dan Zerkle. « GrIDS-a graph based intrusion detection system for large networks ». In : *Proceedings of the 19th national information systems security conference*. T. 1. Baltimore. 1996, p. 361–370.

[77] Mario Gerla et Jack Tzu-Chieh Tsai. « Multicluster, mobile, multimedia radio network ». English. In : *Wireless Networks* 1.3 (1995), p. 255–265. issn : 1022-0038. doi : 10.1007/BF01200845. url : http://dx.doi.org/10.1007/BF01200845.

[78] Elizabeth J. O'Neil, Patrick E. O'Neil et Gerhard Weikum. « The LRU-K page replacement algorithm for database disk buffering ». In : *SIGMOD Rec.* 22.2 (juin 1993), p. 297–306. ISSN : 0163-5808. DOI : 10.1145/170036.170081. URL : http://doi.acm.org/10.1145/170036.170081.

[79] Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, L Todd Heberlein, Che-Lin Ho, Karl N Levitt, Biswanath Mukherjee, Stephen E Smaha, Tim Grance et al. « DIDS (distributed intrusion detection system)-motivation, architecture, and an early prototype ». In : *Proceedings of the 14th national computer security conference*. T. 1. 1991, p. 167–176.

[80] Fred Cohen. « Computer viruses : Theory and experiments ». In : *Computers & Security* 6.1 (1987), p. 22–35. ISSN : 0167-4048. DOI : http://dx.doi.org/10.1016/0167-4048(87)90122-2. URL : http://www.sciencedirect.com/science/article/pii/0167404887901222.

[81] D. E. Denning. « An Intrusion-Detection Model ». In : *IEEE Transactions on Software Engineering* SE-13.2 (fév. 1987), p. 222–232.

[82] Gregory G Finn. *Routing and addressing problems in large metropolitan-scale internetworks*. Rapp. tech. DTIC Document, 1987.

[83] Ting-Chao Hou et V.O.K. Li. « Transmission Range Control in Multihop Packet Radio Networks ». In : *Communications, IEEE Transactions on* 34.1 (jan. 1986), p. 38–44. ISSN : 0090-6778. DOI : 10.1109/TCOM.1986.1096436.

[84] F. Masuoka., M. Asano, H. Iwahashi, T. Komuro et S. Tanaka. « A new flash E^2PROM cell using triple polysilicon technology ». In : *Proceedings of IEEE International Electron Devices Meeting*. T. 30. 1984, p. 464–467. DOI : 10.1109/IEDM.1984.190752.

[85] H. Takagi et L. Kleinrock. « Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals ». In : *Communications, IEEE Transactions on* 32.3 (mar. 1984), p. 246–257. ISSN : 0090-6778. DOI : 10.1109/TCOM.1984.1096061.

[86] Alan J. Smith. « Cache memories ». In : *ACM Computer surveys* 14.3 (sept. 1982), p. 473–530.

[87] Richard W. Carr et John L. Hennessy. « WSCLOCK a simple and effective algorithm for virtual memory management ». In : *SIGOPS Oper. Syst. Rev.* 15.5 (déc. 1981), p. 87–95. ISSN : 0163-5980. DOI : 10.1145/1067627.806596. URL : http://doi.acm.org/10.1145/1067627.806596.

[88] Godfried T. Toussaint. « The relative neighbourhood graph of a finite planar set ». In : *Pattern Recognition* 12.4 (1980), p. 261–268. ISSN : 0031-3203. DOI : http://dx.doi.org/10.1016/0031-3203(80)90066-7. URL : http://www.sciencedirect.com/science/article/pii/0031320380900667.

[89] Rudolf Bayer. « Symmetric binary B-Trees : Data structure and maintenance algorithms ». English. In : *Acta Informatica* 1 (4 1972), p. 290–306. issn : 0001-5903. doi : 10.1007/BF00289509. url : http://dx.doi.org/10.1007/BF00289509.

[90] L. A. Belady. « A study of replacement algorithms for a virtual-storage computer ». In : *IBM Syst. J.* 5.2 (juin 1966), p. 78–101. issn : 0018-8670. doi : 10.1147/sj.52.0078. url : http://dx.doi.org/10.1147/sj.52.0078.

Annexe

Une annexe à ce document inclu une sélection de publications qui précisent des résultats ou mettent en lumière des résultats volontairement omis pour conserver le caractère synthétique du document. Cette annexe contient les publications suivantes :

[J1] T. Ducrocq, M. Hauspie et N. Mitton. « Balancing energy consumption in clustered wireless sensor networks ». In : *ISRN Sensor Networks* (oct. 2013). url : `http://hal.inria.fr/hal-00863123`.

[C2] T. Ducrocq, M. Hauspie, N. Mitton et S. Pizzi. « On the impact of network topology on wireless sensor networks performances ». In : *Proceedings of The nineth Workshop on Performance Analysis and Enhancement of Wireless Networks (PAEWN'2014)*. Victoria, Canada, mai 2014.

[C3] D. Riquet, G. Grimaud et M. Hauspie. « DISCUS : A massively distributed IDS architecture using a DSL-based configuration ». In : *Proceedings of the 2014 International Conference on Information Science, Electronics and Electrical Engineering (ISEEE'2014)*. Sapporo City, Hokkaido, Japan, 2014.

[C8] D. Riquet, G. Grimaud et M. Hauspie. « Large-scale coordinated attacks : Impact on the cloud security ». In : *Proceedings of The Second International Workshop on Mobile Commerce, Cloud Computing, Network and Communication Security 2012 (MCNCS 2012)*. Palermo, Italy, juin 2012.

[C9] G. Cogniaux, M. Hauspie et F.-X. Marseille. « Étude Préliminaire À Une Utilisation De Mémoires Secondaires Pour Le Stockage Des Métadonnées Java Dans Des Systèmes Contraints ». In : *Conférence Française en Système d'Exploitations*. Saint-Malo, France, mai 2011.

[C12] M. Hauspie et I. Simplot-Ryl. « Enhancing nodes cooperation in ad hoc networks ». In : *Proceedings of the 4th IEEE/IFIP Annual Conference on Wireless On demand Network Systems and Services (WONS 2007)*. Obergurgl, Austria : IEEE Press, jan. 2007.