REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
ECOLE NATIONALE POLYTECHNIQUE

Thèse pour obtenir

le grade de DOCTEUR D'ETAT en Electronique

Titre de la thèse

Architectures d'implémentation sur SoC d'un serveur DNS/DNSSEC

Par

Rabah SADOUN

devant le jury d'examen

Pr.	Mourad	HADDADI	Professeur, Ecole Nationale Polytechnique	Président
Pr.	Ahmed	ZERGURRAS	Professeur, Ecole Nationale Polytechnique	Directeur de thèse
Pr.	Adel	BELOUCHRANI	Professeur, Ecole Nationale Polytechnique	Co-Directeur de thèse
Pr.	Samir	BOUAZIZ	Professeur, Université d'Orsay, Paris, France	Examinateur
Pr.	Malika	BOUDRAA	Professeur, USTHB	Examinatrice
Pr.	Mohamed	TRABELSI	Professeur, Ecole Nationale Polytechnique	Examinateur
Pr.	El Bey	BOURENANE	Professeur, U. de Dijon, France	Invité
Dr.	Hakim	KHOUAS	Maître de Conférence, UMB,Boumerdes	Invité

Février 2012

Remerciements

Mes remerciements sont adressés aux membres du Jury qui m'ont fait l'honneur d'accepter de juger ce présent travail.

Que Monsieur HADDADI, Professeur à l'Ecole Nationale Polytechnique, trouve l'expression de ma gratitude pour avoir accepté de présider le jury de soutenance.

Je n'oublierai pas Monsieur ZEGUERRAS, doyen de notre école et directeur de cette thèse, pour m'avoir soutenu à maintes fois pour mener à terme ce travail de thèse.

Je ne crois pas en l'amitié absolue. Un ange en la personne de Adel m'a démontré qu'elle existe, que les anges existent et qu'on peut les côtoyer chaque jour. Que dire à quelqu'un qui recherche l'intérêt de l'autre sans contrepartie aucune tout en y mettant toute l'énergie et le dévouement nécessaire. Sans toi, cette thèse n'aurai pas vu le jour, ni soutenue. Sûrement je n'avais pas fait l'effort nécessaire pour offrir un travail à la hauteur de ton amitié, à la hauteur de ta gentillesse, de ta disponibilité et de ton talent. Simplement merci.

Mes remerciements vont aussi vers Madame BOUDRAA, Professeur à l'université HOUARI BOUMEDIENNE, qui a bien voulu examiner cette thèse.

Que le Professeur BOUAZIZ trouve l'expression de ma gratitude toute particulière. Non seulement pour avoir accepté de juger cette thèse mais aussi de m'avoir offert plus d'une fois l'hospitalité et l'aide de son laboratoire. Je n'oublierai pas les discussions plus que fructueuses que j'ai eu avec lui. Merci aussi pour ton amitié.

Le professeur BOURENANE a eu à supporter mon caractère difficile et réservé. Je ne saurai le remercier assez pour m'avoir offert l'environnement nécessaire pour mener à terme la phase de développement de ce projet. Une autre amitié tout aussi sincère.

Que le Professeur TRABELSI trouve ici l'expression de toute mon amitié et de mes remerciements pour avoir accepté de participer au jury. Qu'il trouve surtout l'expression de mon profond respect.

Un remerciement à la hauteur de son amitié et de sa discrétion va au Professeur IBTIOUEN. Merci l'ami, merci surtout mon frère. Je ne pense pas avoir été à la hauteur de tes exigences. Je dois encore progresser.

J'ai rencontré un autre ange (oui les anges existent) dont je ne tairais le nom mais qui a apporté tout son réconfort par sa présence et son soutient oh combien précieux mais surtout en O.R. Il se reconnaitra, j'en suis convaincu.

Mes remerciements vont vers ceux qui de prêt ou de loin ont apporté leur soutien.

Je finirai par une mère toute aussi attentive et aussi sensible même dans une situation des plus difficiles. A elle, je dédie toute ma reconnaissance et toute mon affection.

Résumés

ملخص
التطورات التكنولوجية الحالية في مجال إنجاز الدوائر تسمح بالنسبة لطريقة العمل، ووسائل الإنجاز والبرامج وأيضا عدد الموارد التي يمكن إدماجها في شريحة واحدة إلى تصغير عالي. اعتبار هته الحقائق DNS يعتبر إحدى نقاط الضعف لتشغيل شبكة انترنت، إذا خذنا بعين DNS/DNSSEC على دائرة إلكترونية. نموذج SDLصمم و علما أن مجال الأسماء نقترح في هده الأطروحة إنجاز موزع . لإنجاز هدا الموزع نفدت أربعة أنواع مختلفة من التصاميم على FPGA Virtex5 . دائرة التصاميم تختلف باختلاف التركيبات الممكنة الكامنة SDL باستعمال أجهزة مسرعة مادية من نموذج للبحث عن المعطيات RR ووظيفة التشفير. قمنا بدراسة مقارنة للأداء مع BIND9 الذي صمم تحت برنامج LINUX (الأكثر استعمالا) موزع

Résumé

Les évolutions technologiques actuelles dans le domaine de la conception des circuits permettent aussi bien sous les angles de la méthodologie, des plateformes de conception, des algorithmes ainsi que du nombre de ressources pouvant être intégrées sur une même puce, d'aboutir à une miniaturisation avérée. Tenant compte de ces constatations et sachant que le système de noms de domaine DNS constitue l'un des maillons faible du fonctionnement d'Internet, nous proposons dans cette présente thèse une conception d'un serveur de nom autoritaire sur un circuit. Le langage SDL a été introduit pour la modélisation de ce dernier. Quatre formes d'implémentations ont été déduites et réalisées sur un circuit FPGA Virtex 4. Ces implémentations sont différenciées par les combinaisons possibles entre les architectures déduites d'une modélisation SDL et l'utilisation d'un accélérateur matériel dédié à la recherche des ressources de données spécifiques au système de nom de domaine DNSSEC. Une étude comparative des performances a été réalisée en prenant comme référentiel de mesure un serveur Bind 9 porté par Linux. Il y apparaît que la forme MPSoC (MuliProcessor System On Chip) associée à des accélérateurs matériels aboutit à un gain de performance avoisinant les 200% par rapport au serveur de référence choisi. Tenant compte des hypothèses que nous avons formulées, l'implémentation obtenue allie performance et flexibilité à une meilleure sécurité.

Abstract

This thesis deals with the idea of implementing a complete network service on a chip. Herein, we propose an original design together with an efficient implementation of an authoritative domain name system (DNS) server on a Virtex 5 FPGA circuit. The proposed approach exploits the use of a hardware accelerator, Micro Blaze soft-processor cores and an adequate mapping between the DNS specifications and the hardware architecture leading to a Multi-Processor System on Chip (MPSoC). We propose new architectures by translating the DNS specifications to hardware form through the ‘‘Specification and Description Language’’ (SDL) tool. The proposed implementation allows significant reduction in power consumption together with significant performance and security improvement. The proposed architectures have been successfully implemented and tested on an actual network. The obtained results show a query performance improvement of around 200% with respect to the ‘‘Berkeley Internet Name Domain’’ (BIND) 9 server

Sommaire

Liste des figures

Liste des tables

CHAPITRE I

INTRODUCTION

1.1 Motivations

Dans une conférence, tenue en décembre 1959 à l'Institut californien de technologie; devenu mythique depuis, le physicien américain Feynman s'était interrogé sur la possibilité de faire tenir les collections de la Bibliothèque du Congrès, du British Muséum et de la Bibliothèque Nationale, soit 24 millions de volumes ; sur une tête d'épingle de 1,5 mm de diamètre. Cet exemple à l'appui, il avait circonscrit les principales problématiques de ce qu'est devenue la microélectronique et de ce que sont appelées à devenir les nanotechnologies.

L'appel de Feynman a eu pour écho les formulations de Moore qui, en 1965 puis en 1971, a énoncé à l'appui des travaux sur les transistors puis sur les microprocesseurs, sa célèbre loi aux termes de laquelle la puissance des composants doublerait, à taille égale, tous les dix-huit mois.

Le secteur des semi-conducteurs a répondu à cette axiomatique ambitieuse en réussissant, par sauts technologiques successifs, à organiser la réduction géométrique des composants ainsi que le développement d'outils de développement adaptés. Cette évolution va se poursuivre puisque les applications nanoélectroniques des nanotechnologies se profilent.

Ces avancées technologiques ont donc été nourries par une maturation de plusieurs décennies. Nul ne pouvait alors prévoir ces applications, pas plus que l'on ne peut totalement cerner les applications des développements technologiques actuels. Cette évolution a été rendue possible par un effort de miniaturisation qui a réduit, depuis 1960, la taille des composants d'une manière spectaculaire. Mais, tout autant que son amplitude, c'est la maîtrise de ce processus de miniaturisation qui étonne : il a été pensé, planifié et réalisé de façon constante et progressive depuis plus de quarante ans.
Des paradigmes nouveaux sont alors apparus tenant compte des possibilités d'intégration qui y sont offertes. C'est ainsi que les concepts de SoC (système sur puce), de microprocesseur réseau, de circuit reconfigurable, de NoC sont apparus. On peut être d'ailleurs de plus en plus étonné par les puissances de plus en plus spectaculaires des systèmes à usage domestique ou ceux totalement enfouis. On pourra affirmer, sans trop se tromper, que les data center de demain seront des systèmes totalement enfouis. L'ENIAC (Electronic Numerical Integrator Analyser and Computer), premier ordinateur moderne, en est une parfaite illustration si on prend le prend comme base de comparaison par rapport à une simple tablette graphique actuel.

Parallèlement à cette évolution, les technologies de la communication ont pris un essor fulgurant, aidé par celle, non moins négligeable, des machines de traitement automatique de l'information. Un effet boule de neige s'ensuit. Le tout numérique prend de plus en plus son sens et tend à se généraliser. L'interconnexion des systèmes devient une nécessité et le concept de la sécurité des systèmes s'impose de fait.

Avec cet essor des puissances de traitement s'est développé un autre paradigme non moins important : la virtualisation des systèmes; une conséquence directe du concept de couche (machine) virtuelle. Elle simplifie le test et l'exploitation de solutions de sécurité, assure la continuité d'activité en s'adossant à un stockage en réseau et réduit les coûts. On isole ainsi des services d'infrastructure (proxy, DNS, antispam, antivirus, pare-feu, annuaires) dans autant de machines virtuelles. On teste les correctifs sans risque. De plus, le contrôle des droits s'affine grâce à une dissociation par itération virtuelle, et les politiques de sécurité sont mieux ciblées et plus granulaires.

Mais l'accumulation de composants logiciels (systèmes d'exploitation, applications métiers, utilitaires, services, etc.) sur une même cible multiplie le nombre de failles potentielles et leurs interactions. Cette faiblesse a suscité des parades comme l'IPS (Intrusion prevention system) aboutissant au contrôle des échanges entre serveurs virtuels au sein d'une machine physique. Cependant, la virtualisation tire le maximum de ressources d'une machine. Les attaques par déni de service saturent alors très vite le système ainsi déployé.

Si cette solution, la virtualisation, facilite donc l'administration et l'exploitation tout en contribuant à réduire les coûts, elle peut fragiliser la sécurité des systèmes par un retour remarqué à la centralisation. L'hyperviseur devient alors le maillon faible.

C'est dans ce contexte que nous nous sommes intéressés à l'étude de la mise en œuvre des paradigmes des systèmes sur puce dans le domaine des services réseaux en vue de leur miniaturisation et de leur sécurité.

L'étude qui sera faite traitera d'une manière particulière le système de nom de domaine. L'approche restera valide pour tout service réseau aussi bien périphérique ou faisant partie du cœur du réseau.

1.2 Etat de l'art

Avec le protocole BGP (Border gateway protocol), le système de nom de domaine (DNS) met en œuvre l'un des deux protocoles les plus critiques du réseau Internet. L'une de ses particularités est son apparente simplicité fonctionnelle. Cette simplicité, outre qu'elle peut induire des négligences dans sa gestion et sa configuration conduisant ainsi à des situations pouvant être dramatiques, cache différentes difficultés non encore résolues à ce jour. Ces difficultés sont associées aussi bien à ses caractéristiques intrinsèques qu'aux implémentations utilisées. Elles dérivent d'une caractéristique essentielle liée à l'accès publique aux données qu'il gère à savoir, les adresses IP et les noms qui leurs sont associés. La classification des applications les plus vulnérables pour l'année 2006 classe le DNS parmi les vingt premières.

Le système DNS est utilisé à l'heure actuelle principalement pour assurer le service de nommage sur Internet. Toutefois, l'émergence de nouveaux protocoles, comme ENUM (telephone number mapping) par exemple, offre de nouvelles perspectives de service et donc une utilisation nouvelle des bases de données DNS avec de nouvelles contraintes de performance et de sécurité.
Avec l'évolution quantitative des nombres d'ordinateurs à interconnecter, le fichier HOSTS utilisés originellement pour résoudre le problème de nommage d'ordinateurs sur le réseau est devenu une contrainte plutôt qu'une solution. C'est ainsi qu'est né le système du nom de domaine. Ses premières définitions stables ont été décrites essentiellement par les RFC 1043 et 1034. Nul ne prédisait à l'époque les problèmes de sécurité qui en découleraient ni de la progression de la taille du réseau auquel il était destiné. On constate depuis lors plusieurs propositions pour y remédier. Les plus récentes et les plus connues sont référencées par les RFC 4033,4034 et 4034. Ce système passe alors du concept classique de DNS au concept de DNSSEC représentant la forme sécurisée de la première variante. Des outils cryptographiques ont été introduits dans le but d'assurer l'authenticité et l'intégrité des données transmises sachant qu'elles doivent garder leur caractère public. L'image qui en est donnée à cette évolution est illustrée par une enveloppe scellée (intégrité des données) mais transparente (caractère publique). La conséquence immédiate est

l'accroissement du nombre de données échangées ainsi que leurs tailles. Ce qui engendre un trafic et une latence relative encore plus importants se traduisant par la nécessité d'une bande passante et des performances des équipements à mettre en œuvre encore plus élevées.
Comme cités plus haut, les implémentations ont aussi une part importante dans les dysfonctionnements constatés. Les plus connues et les plus répondues sont BIND et Microsoft DNS. La première est traitée de peu sûre alors que la seconde est qualifiée des plus mauvaises. Dans le cas de cette dernière, sa version 2008R2 inclue la prise en charge de DNSSEC. Diverses autres implémentations existent aussi. Toutes sont sous forme logicielle.
On trouve d'autres implémentations aussi bien propriétaires que libres. Certaines se présentent comme des équipements dédiés alors que d'autres sont portées par des machines équipées de système d'exploitation POSIX (Portable Operating System Interface), LSB (Linux Standard Base) ou Microsoft. La caractéristique commune à toutes ces implémentations est l'utilisation, dans le processus de résolution, de processeurs généralistes couplée à un système d'exploitation.

Diverses recommandations pour une utilisation plus sûre ont été édictées par USA SECURITY AGENCY. L'accent est essentiellement mis sur la configuration et l'utilisation des serveurs DNS. Les failles intrinsèques sont donc essentiellement prises en charge par les normalisations édictées par l'IETF (Internet Engineering Task Force). Les implémentations quant à elles, outre qu'elles doivent assurer la transition vers ces nouvelles normalisations, doivent aussi corriger les imperfections qui y sont constatées liées aussi bien à leurs performances qu'à leurs sûretés de fonctionnement tout en garantissant une flexibilité de modification.

1.3 Contribution

C'est à travers ces deux constatations que nous avons formulé la nécessité de migrer ce service réseau vers des cibles technologiques appropriées. Nous avons proposé et mis en œuvre une méthodologie adaptée conduisant à une forme d'implémentation devant assurer de meilleures performances, une plus grande sécurité de fonctionnement et supportant toutes les normalisations édictées par l'IETF y inclus la forme sécurisée DNSSEC (Domain name system security extensions).

Cette thèse exprime notre proposition de migration des services critiques, périphériques ou faisant partie du cœur du réseau, vers des formes plus adaptées.

1.4 Plan de la thèse

Le premier chapitre traitera des spécifications du protocole DNS/DNSSEC nécessaires à sa modélisation. Tel que édicté par les spécifications du protocole objet de notre étude (DNS/DNSSEC), ce chapitre sera complété par notre motivation quant au choix des architectures d'implémentation. Le deuxième chapitre mettra en valeur et justifiera la méthodologie de développement choisie. Il en sortira que le langage d'abstraction SDL sera le choix adopté pour notre étude. Le chapitre trois est une conséquence du chapitre deux. Il traitera de la justification du flot de conception adopté tenant compte de la modélisation SDL du protocole DNS/DNSSEC. L'objectif majeur de notre étude vise la recherche de formes d'implémentation SoC les plus adaptées.
C'est ainsi que la quatrième partie et dernière partie abordera la description d'architectures d'implémentation déduites de la modélisation d'un serveur DNSSEC autoritaire. Les résultats d'implémentation obtenus seront analysés tout en concluant par des perspectives de développement.

CHAPITRE II

Le système de nommage Internet

Nous présentons dans ce qui suit les détails du Domain Name System (DNS) ou système de noms de domaine, son principe de fonctionnement et son architecture. Nous exposons ses faiblesses connues. Après avoir étudié les besoins de sécurité du DNS, nous décrivons les extensions de sécurité du système de noms de domaine ; DNSSEC. Nous formulerons par la suite notre point de vue quant à la nécessité d'une implémentation SoC d'un serveur de nom de domaine.

2.1 Concepts généraux

Le système de noms de domaine [Moc87a, Moc87b, AL021] permet d'établir, d'une manière transparente, la correspondance entre un nom de machine (niveau d'abstraction élevé),et son adresse (dite IP) formulée sous forme d'une représentation à N bits. Pour répondre au nombre grandissant de machines connectées sur le réseau Internet, le mécanisme effectuant cette correspondance doit être d'une grande efficacité. Le trafic engendré à travers les couples requêtes/réponses devient alors plus important aboutissant ainsi à d'autres mécanismes tels que leurs mémorisations au niveau des nœuds intermédiaires; le but évident étant la réduction des temps de latence observés.

2.1.1 Origines

Avec l'avènement des protocoles TCP/IP, chaque machine du réseau ARPANET se voit attribuer une adresse IP (dans sa version 4 ou 6). La mémorisation et l'utilisation de ces adresses numériques détermina rapidement la mise en place d'un système de traduction de celles-ci en objets plus compréhensibles par l'Humain.
Les ordinateurs du réseau ARPANET s'échangeaient initialement un fichier spécifié par la RFC 608 et appelé "HOSTS.TXT" (existant encore sur nos ordinateurs d'aujourd'hui). Il sert à diffuser les correspondances entre des adresses IP numériques et leurs noms en chaines alphanumériques.
Sa gestion était centralisée à l'époque par le SRI-NIC (SRI Network Information Center) et distribué à tous les hôtes du réseau via des transferts de fichiers directs et indirects.
Cependant, l'explosion du nombre de machines sur le réseau ARPANET fit apparaitre les limitations d'un tel système basé sur la mise à jour permanente de ces listes de correspondances des adresses. Plusieurs solutions ont été envisagées à l'époque pour son remplacement comme l'IEN 116 ou le système Grapevine de Xerox. Le premier était jugé trop simple, le second beaucoup trop compliqué. Les réflexions commencèrent alors sur le protocole DNS. Fait étonnant, un débutant nommé Paul Mockapetris avait hérité de cette tâche. Ce fait montre la perception qu'on en faisait de ce service (qui ressemble curieusement à celle qu'on en fait encore aujourd'hui) qui était celle d'un service qu'on considérait comme non essentiel. Le caractère transparent (mais non secondaire) peut en être la cause.

Dès 1983, Paul Mockapetris écrit la première implémentation du DNS pour le système d'exploitation "TOPS-20". Ces recherches seront publiées sous les RFC882 ("Domaine Names - Concepts and Facilities") et RFC883 ("Domain Names - Implementation and Specification") qui vont évoluer vers les plus connues, les RFC1034 et RFC1034. C'est vers 1985 que certaines machines commencèrent à n'utiliser que le DNS pour accéder au réseau. Il est intéressant de noter qu'il était encore d'usage de trouver, au début des années 1990, des références documentaires indiquant les adresses IP de serveurs en lieu et place des noms par manque de confiance dans le système DNS qui n'avait pas toujours la fiabilité souhaitée.

Le déploiement du DNS a débuté dès 1988. Dès cette année, HOSTS.TXT contenait quelques 5000 noms alors que le DNS en avait quelques 20 000 entrées. On en dénombre aujourd'hui des centaines de millions de noms.

Les serveurs de noms de la racine n'étaient que sept à l'époque. Quatre étaient sur des systèmes Unix avec comme implémentation BIND. Les trois autres des systèmes TOPS-20 dotés de la première implémentation connue qu'est Jeeves. Le trafic sur un serveur racine était très faible. Aujourd'hui, chaque machine physique racine reçoit des centaines de milliers de requêtes DNS par seconde et constitue non seulement un nœud névralgique mais aussi un élément sensible et un sujet lié à l'indépendance de l'accès libre au réseau Internet.

2.1.2 Le système de résolution de nom

Le système DNS s'articule autour d'un ensemble de serveurs disposant chacun d'un fragment d'une base de données répartie. Le modèle client/serveur est mis en œuvre.

Cette base de données à un représentation hiérarchique. Elle est modélisée sous forme d'un arbre. La figure 2.1 montre une partie de la représentation de ce dernier. La racine est représentée par «.». Chaque nœud y figurant est étiqueté. La structure arborescente de cet arbre (et donc de la représentation du système DNS) permet de joindre la racine à partir de n'importe quel nœud d'une manière unique.

L'efficacité du fonctionnement du système DNS repose sur la responsabilité d'un serveur sur une partie de la base, donc sur un sous-arbre DNS. Cette décentralisation implique une plus grande facilité de gestion des informations contenues dans ces sous-arbres. Cette approche ramène cette souplesse qui manquait à l'usage qu'on en faisait du fichier centralisé HOSTS .TXT. La tolérante aux pannes est alors grandement améliorée.

Tel que proposé par Paul Mockapetris, cette base DNS est redondante. Le but était de renforcer la tolérance aux pannes et la disponibilité du service de nommage. Les informations d'un sous arbre DNS, placées dans un fichier dit de zone, peuvent être gérées par plusieurs serveurs. Le fichier de zone se voit alors répliqué à l'identique sur chacun de ces serveurs. Chaque serveur peut être responsable de plusieurs zones différentes; donc peut contenir plusieurs fichiers de zone.

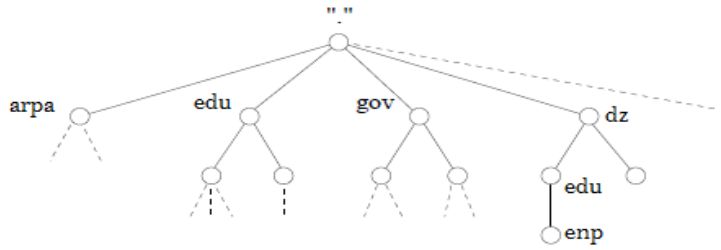

Figure 2.1 Représentation d'une partie de l'arbre DNS

2.1.3 Domaines et zones

Tel qu'introduit précédemment, le système de noms de domaine repose sur une structure arborescente. L'arbre ainsi modélisé possède deux types de constituants : les domaines et les zones.

Un domaine est représenté par un sous-arbre complet de l'arbre DNS. Le nom qui lui est rattaché est déduit par la concaténation de toutes les étiquettes de tous les nœuds en partant de la racine du sous-arbre jusqu'à la racine de l'arbre du système DNS. La construction en arbre interdit à deux nœuds frères d'avoir le même nom et de ce fait, assure l'unicité du nom dans cette base hiérarchique. Un domaine peut donc en englober un autre, comme c'est le cas par exemple sur la figure 2.2 : le domaine

dz. contient le domaine enp.dz.

Chaque domaine est constitué d'une ou plusieurs zones. L'unité administrative de l'arbre DNS est représentée par une zone. Chaque zone est la constituante élémentaire d'un domaine. Ce dernier peut contenir une ou plusieurs zones. Chaque zone est gérée par un ou plusieurs serveurs. Les administrateurs de ces derniers sont responsables sur les informations de zone, leurs mises à jour et leurs disponibilités. Lorsqu'un nœud n'a pas de fils, comme le nœud *enp* sur la figure 2.2, domaine et zone sont alors confondus (*enp.dz.)*

2.1.4 La délégation

La délégation représente la forme de gestion de l'arbre DNS. On parle plus précisément de la délégation d'autorité. Cette forme d'organisation, la délégation d'autorité, induit une souplesse de gestion avérée.

Toute zone peut être sujette à modification. Les informations qu'elle contient doivent être mises à jour . La taille peut varier d'une manière conséquente. Une nouvelle zone est créée lorsqu'une zone délègue la responsabilité d'une partie de ses noms; ce qui crée un nouveau nœud dans l'arbre DNS. Un nouveau fichier de zone est alors créée à son tour contenant les noms objet de la délégation. Il sera placé sous la responsabilité de nouveaux serveurs dits serveurs autoritaires. Ils seront dédiés à l'administration de chaque nouvelle zone. Cette dernière est généralement appelée zone fille. La zone initiale est appelée zone mère ou zone parente. Un lien est créé entre la zone mère et la zone fille lors de la création d'une délégation. Ce lien est représenté par une entrée dans le fichier de zone (de la zone mère) appelé enregistrement NS (Name Server).

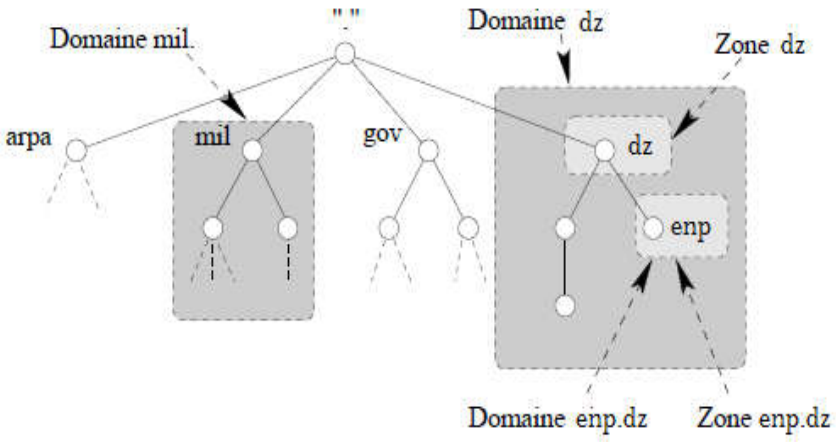

Figure 2.2 Domaine et zone du système DNS

2.1.5 Constituants de l'architecture du système DNS

Les serveurs de noms autoritaires, des serveurs caches récursifs et des résolveurs sont les constituants de l'architecture du système DNS. Ils seront dédiés à la gestion et l'exploitation des informations contenues dans chaque fichier de zone.

2.1.4.1 Serveur autoritaire (*Authoritative DNS Nameserver*)

Tel que décrit précédemment, un serveur de noms a autorité sur une ou plusieurs zones. Il est donc responsable sur les informations liées à chaque zone. Deux types de serveurs de noms sont à

considérer : les serveurs dits primaires et ceux dits secondaires. Le serveur primaire gère le fichier de zone original. C'est lui qui sera référencé par un enregistrement (ou entrée du fichier de zone) spécifique appelé SOA (Start Of Authority). Les serveurs secondaires quant à eux ne possèdent qu'une copie du fichier de zone original. Lorsque le serveur primaire est mis à jour, les serveurs secondaires sont alors informés pour répercuter les modifications ainsi effectuées. Les serveurs secondaires envoient à leurs tours une demande de mise à jour suivant un temps d'attente aléatoire pour éviter des demandes simultanées. Une copie du fichier de zone est alors transmise aux serveurs secondaires impliqués dans la requête de mise à jour (figure 2.3).

Les serveurs aussi bien primaires que secondaires répondent aux requêtes DNS en exploitant les informations contenues dans le fichier de zone approprié (original ou copie).

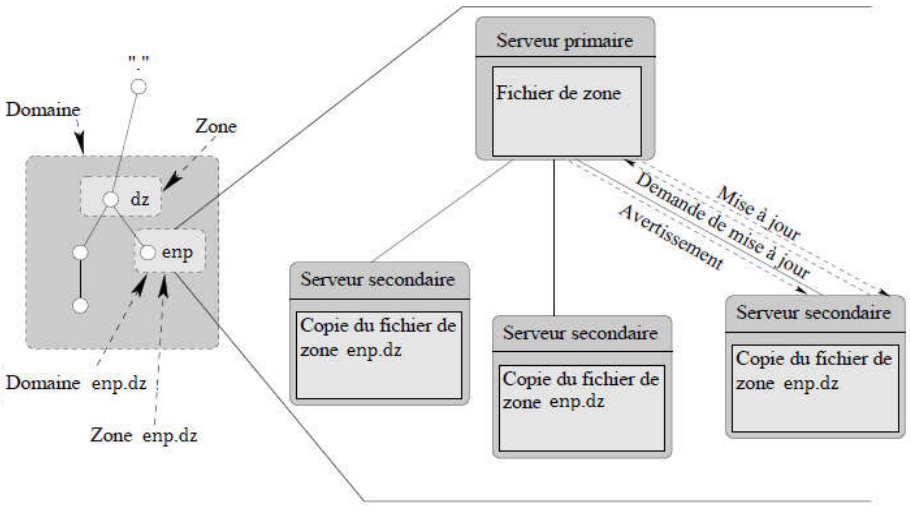

Figure 2.3 Serveur autoritaire; primaire et secondaire

2.1.4.2 Serveur récursif (*Recursive DNS Server*)

Les serveurs récursifs sont aussi appelés serveurs caches. Il est une entité cliente pour les serveurs autoritaires. De par leur nature, ils ne possèdent pas de fichier de zone. Ils ne font donc autorité sur aucune zone. Ils sont généralement positionnés en mandataires sur un réseau afin de recevoir les requêtes des résolveurs locaux. Ces serveurs auront pour rôle de répondre à des requêtes en utilisant les informations précédemment reçues et mises en mémoire durant des temps prédéterminés. Si aucune réponse n'est trouvée, les serveurs cache font suivre les requêtes au serveur autoritaire susceptibles de posséder une réponse, mettent en cache les réponses reçues et les font suivre au résolveur source de la demande. L'objectif de l'utilisation de ces serveurs vise la diminution de la charge sur les serveurs autoritaires par la mutualisation des informations au plus proche [Sit00,JSBM01,CK01].

2.1.4.2 Le résolveur (*Resolver*)

Le résolveur est l'entité cliente périphérique. C'est à lui qu'échoient les demandes en provenance d'une couche applicative à travers des requêtes DNS appropriées. Lorsque ce résolveur reçoit une réponse à une requête émise, il la redirige à l'application source.

2.1.6 L'enregistrement

Toute zone est définie par un fichier, dit de zone, localisé sur des serveurs autoritaires. Ce fichier contient un certain nombre d'entrées. Chaque entrée est représentée par un enregistrements et un seul dit enregistrement de ressource ou RR. Il constitue la brique élémentaire de ce fichier. Il a divers types selon les informations qu'ils contiennent. A chaque enregistrement est associée une durée de vie (TTL - Time To Live)) ou délai d'expiration sur laquelle se base le serveur récursif pour le rafraichissement de ses données mises en mémoire cache.
Un seul enregistrement SOA est associé un fichier de zone. Les noms des serveurs autoritaires , qu'ils soit primaires ou secondaires sont conservés dans des enregistrements NS (Naine Server). Ils créent une relation entre un nom de zone (enp.edu.dz.) et un nom de serveur (ns1.enp.edu.dz.). Le serveur primaire est déterminé par l'enregistrement SOA.

Le lien entre un nom de machine et son adresse IPv4 (resp. IPv6) est l'enregistrement A (resp. AAAA). Sur l'exemple de la figure 2.4, la noeud djurdjura.enp.edu.dz. possède comme adresse IPv4 193.194.70.1. La durée de vie (TTL) de cet enregistrement a été fixée à 60 secondes.

Un fichier de zone peut être scindée en deux parties : les informations de zone et les délégations vers des zones filles. Ces délégations sont représentées généralement par un (ou plusieurs) enregistrement(s) NS, contenant le nom d'un (ou plusieurs) serveur(s) de noms de la zone fille. A ces enregistrements sont associés un (ou plusieurs) enregistrement(s) A (ou AAAA) représentant la ou les adresse(s) IP du (ou des) serveur(s) de noms autoritaires. Ces enregistrements (NS et A) sont dupliquées sur les serveurs autoritaires de deux zones (mère(s) et fille(s)). Glue Record est la dénomination adoptée pour ce couple d'enregistrements.

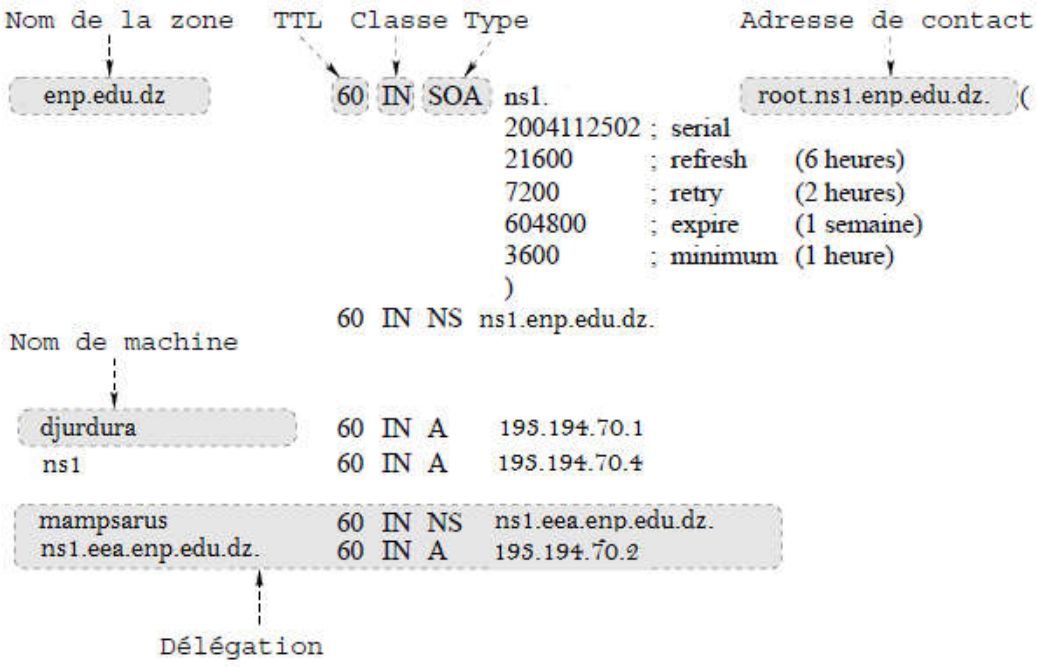

Figure 2.4 Exemple de fichier de zone

2.1.7 Différents types de résolution

Le résultat de l'association entre une adresse et un nom définit la résolution de nom. Deux modes de résolution sont à considérer; itératif et celui dit récursif. La figure 2.5 exhibe ces deux modes. En mode itératif, le serveur ne répondra qu'avec les informations dont il dispose. S'il est autoritaire sur le nom de domaine objet de la requête, il en donnera la réponse. Dans le cas contraire, la réponse produite indiquera le serveur de noms susceptible de générer une réponse la plus adaptée. Le nom de ce serveur correspondra à la délégation ayant le plus long suffixe commun avec le nom objet de la requête reçue. Par exemple, le serveur de la zone racine répondra avec le nom du serveur de la zone dz., à savoir ns1.dz., pour une requête sur www.enp.edu.dz. Cette réponse présente le plus long suffixe commun avec le nom de domaine enp.edu.dz. La figure 2.5 montre que les serveurs *Name Server* sont configurés en mode de résolution itératif.

En mode récursif, le serveur réalise la résolution complète. Il va se charger de répercuter la requête reçue sur les serveurs autoritaires en tenant compte des délégations successives jusqu'à obtenir la réponse souhaitée. La figure 2.5 montre que le serveur *Recurcive Cache* implémente le mode de résolution récursif. En général, les serveurs dits autoritaires sont dissociés des serveurs récursifs.

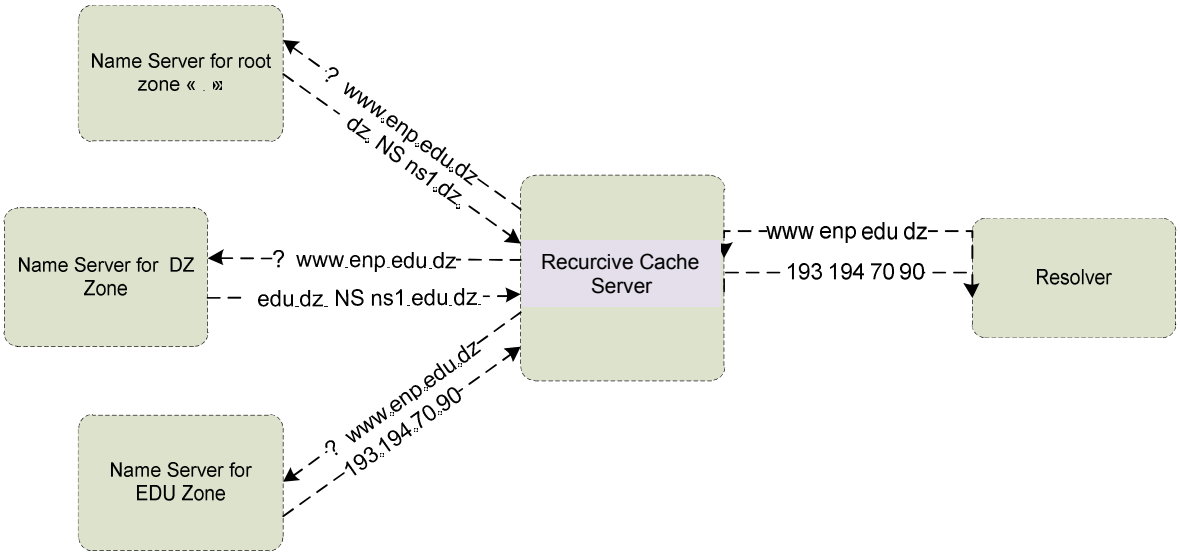

Figure 2.5 Les deux modes de résolution de noms de domaine

2.1.7.1 La résolution inverse

Le système DNS peut aussi être utilisé pour fournir le nom correspondant à une adresse IP donnée. Cette opération est dite résolution inverse. Une partie de l'arbre DNS est alors dédiée à cette résolution. Ce sous arbre est formalisé par le domaine in-addr.arpa. Ce dernier comporte quatre niveaux pour les adresses IP de version quatre. Les nœuds de chaque niveau sont étiquetés à l'aide d'un entier compris entre 0 à 255 représentant un octet de l'adresse IP cible; le niveau le plus haut représentant l'octet de poids fort de cette même adresse. Cette correspondance est représentée par un enregistrement de type PTR (PoinTeR). Chaque nœud de l'arbre reverse contient des délégations vers les serveurs de noms gérant les blocs d'adresses IP correspondants. La figure 2.6 illustre le contenu de chaque nœud du chemin 1.70.194.193 . in- addr . arpa . et représente une sous partie du domaine in-addr.arpa. pour une correspondance inverse pour la machine djurdjura.enp.edu.dz. d'adresse IP : 193.194.70.1 (86400 et 30 sont les TTL des enregistrements respectifs).

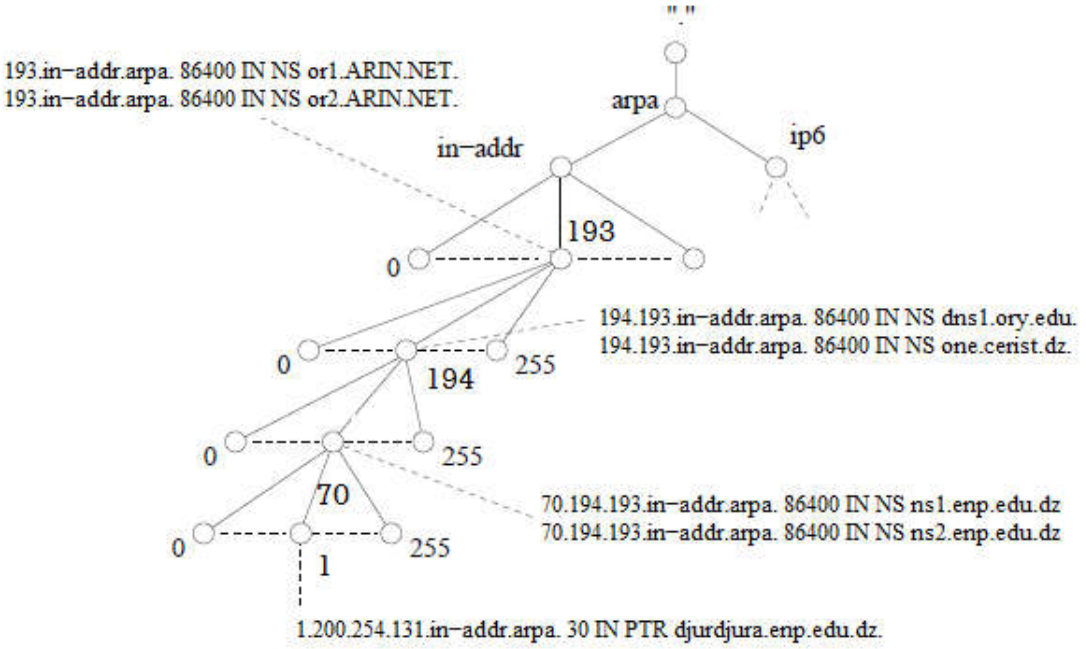

Figure 2.6 Mécanisme de résolution inverse

2.1.7.2 Mises à jour dynamiques - transfert de fichier de zone

La topologie d'un réseaux peut être dynamique. Des machines joignent ou quittent un réseau et le protocole DHCP [Dro97] peut être exploité pour une attribution dynamique d'une adresse IP à une machine rejoignant une réseau donné.

Cette dynamique doit être intégrée au niveau des mécanismes de fonctionnement des serveurs DNS afin que ces modifications soient transcrites d'une manière automatique sur le fichier de zone cible. Cette opération est représente les mises à jour dynamiques [VTRB97].

Les requêtes *Dynamic Update* permettent à un serveur autorisé, un serveur DHCP par exemple, de préciser au serveur de noms primaire de mettre à jour des enregistrements de fichier de zone en ajout, en suppression ou en modification. Le numéro de série du fichier de zone contenu dans l'enregistrement SOA se voit incrémenté induisant une notification via un message NOTIFY [Vix96] vers tous ses serveurs secondaires. Dès réception de cette notification, un serveur secondaire transmet un NOTIFY & RESPONSE pour acquitter le message NOTIFY. Il demande alors l'enregistrement SOA du serveur primaire dans le but de constater le changement effectif du numéro de série en comparaison avec celui inscrit dans son fichier de zone. Si ce numéro de série est bien supérieur à celui disponible sur le serveur secondaire et après une attente aléatoire, une demande de transfert de zone est alors générée. Cette attente aléatoire vise à répartir le temps les transferts de zone pour diminuer la charge générée sur le serveur primaire. Deux types de transfert de zone sont à considérer; le transfert de zone total précisé par A XFR [Moc87b], tout le fichier de zone est alors transmis aux serveurs secondaires, et le transfert de zone incrémental précisé par IXFR [Oht93]. Dans ce dernier cas, seules les parties du fichier de zone objet de modification sont transférées.

2.2 Les faiblesses du système de nom de domaine

Le système de résolution de nom DNS, malgré son apparente simplicité, reste une pierre angulaire dans l'architecture de fonctionnement du réseau Internet. Sa vulnérabilité reste avérée [AA04,GC03B,LMMM00]. L'attaque par déni de service [BCN01,Nar02,Thu01] dans ses deux variantes DoS ou DDoS est l'une des vulnérabilités extrinsèques des plus importantes. Les serveurs racines ont été la cible de ce type d'attaque; un disfonctionnement général du réseau Internet par la déconnection logique de tous les nœuds était l'objectif recherché.

La figure 2.7 explicite les diverses faiblesses et leurs localisations dans le flux d'échange entre les divers constituants de l'architecture du système DNS. Ce flux d'échange concerne aussi bien les échanges de base (requête-réponse) que ceux dédiés à la l'administration des fichiers de zone.

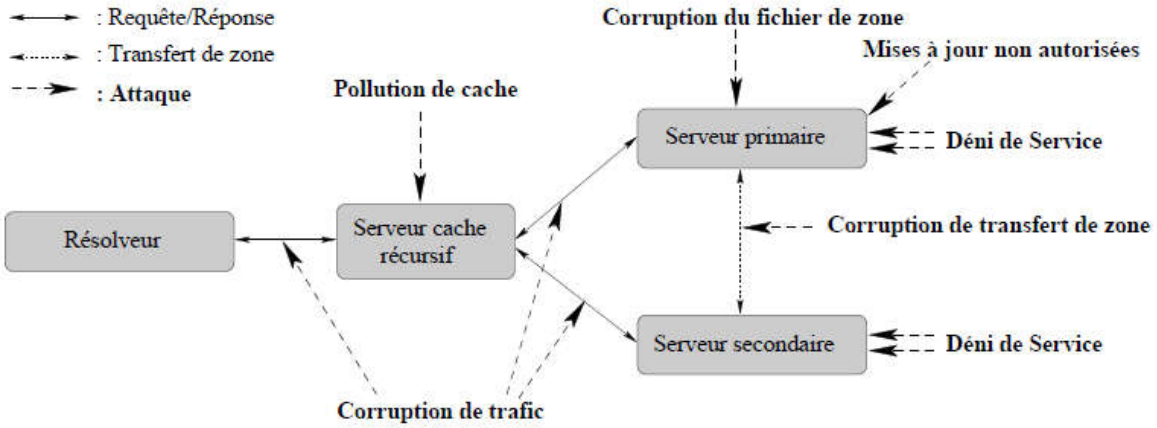

Figure 2.7 Localisation des faiblesses du DNS

2.2.1 Les attaques HDM (Homme Du Millieu) ou MITM (Man-In-The-Middle)

Les attaques traditionnelles dites attaques par l'homme du milieu (man in the middle) s'applique au système de noms de domaine. Un attaquant, positionné entre un client (la victime) et un serveur qu'il soit un serveur autoritaire ou récursif, peut intercepter le trafic les reliant. Si cet attaquant peut forger une réponse plus rapidement que le serveur cible, il conditionnera alors le comportement de la victime suivant son desiderata.

Une telle attaque peut avoir des conséquences multiples comme rediriger sa cible vers un serveur "complice" (usurpation d'identité avec une fonction à visée frauduleuse). Une attaque par déni de service peut aussi être engendrée par un signalement *"ressource ou zone demandée inexistante"* volontairement erroné.

2.2.2 Attaque par l'utilisation du paradoxe des anniversaires

Le paradoxe des anniversaires est le fait que la probabilité de trouver, au sein d'un groupe, deux personnes qui aient la même date de naissance est plus grande contrairement à ce que l'on peut intuitivement le penser. Appliqué au système DNS, cette date est remplacé par l'identifiant de la requête.

Cet identifiant porté par l'entête du message DNS a une taille de 16 bits. Il est généré aléatoirement par la machine source d'une requête. Il doit être repris dans la réponse, établissant ainsi la correspondance plus qu'étroite entre la requête et la réponse. Toute réponse qui n'a pas d'identifiant dans la liste des requêtes émises est alors rejetée. Si un attaquant n'arrive pas à intercepter un trafic DNS,

donc il n'arrive pas à connaitre les identifiants des requêtes émises, il doit trouver l'identifiant probable pour le placer dans une fausse réponse.
L'identifiant avait un contenu incrémenté de 1 à chaque nouvelle requête dans les premières implémentations du DNS. Cette approche facilitait la prévision de la valeur courante de l'identifiant de requête pour un résolveur ou un serveur donné par l'envoi d'un très petit nombre de fausses réponses. Pour résoudre cette faille, cette forme d'implémentation a évolué vers une génération d'identifiant d'une manière aléatoire. Cependant et suivant le paradoxe des anniversaires, un nombre plus important de requêtes/réponses peut aboutir à forger une mauvaise réponse avec un identifiant connu (portée par une question existante).
L'identifiant d'un message DNS étant sur 16 bits, 65535 identifiants différents sont possibles. Une attaque consistant à générer 65535 fausses réponses ne peut aboutir en raison de la limitation temporelle; toute bonne réponse reçue induit que toute "fausse" réponse sera rejetée même si son identifiant est correct. Le paradoxe de l'anniversaire résout cette autre contrainte. Si un attaquant couple n requêtes à un n fausses réponses émises vers une cible (un serveur récursif par exemple), la probabilité de prédire un identifiant augmente avec la valeur n. Le paradoxe de l'anniversaire donne la formule [GEU06] :

$$P = 1 - (1 - \frac{1}{65536})^{n(n-1)/2}$$

65536 correspond au nombre de cas possibles tenant compte de la largeur de l'identifiant.

Comme exemple, avec 302 messages, la probabilité prédire un bon identifiant est supérieure à 50%. 700 messages permettent de ramener cette probabilité à presque 1.

2.2.3 Attaque par pollution de cache (cache poisoning).

La relation de confiance entre les résolveurs et leurs serveurs récursifs fait de ces derniers des maillons faibles éligibles à la corruption de leurs caches . Cette attaque qui consiste à transmettre de fausses réponses à un serveur cache porte le nom de pollution de cache ou cache poisoning.
La première phase d'une pollution de cache est la récupération de l'identifiant de requête courant du serveur autoritaire cible. La figure 2.8 illustre une attaque par pollution de cache. Cette variante se base sur la récupération préalable de l'identifiant d'une requête courante par l'utilisation d'un serveur de noms autoritaire sous contrôle de l'attaquant.
Le déroulement de l'attaque se fait en deux phases. La première vise à récupérer un identifiant courant du serveur récursif tandis que la seconde (coeur de l'attaque) génère la corruption effective du cache.
L'hôte A transmet une requête au serveur récursif de la zone victime.dz en demandant l'adresse IP d'un hôte se trouvant dans sa propre zone (attaque.dz); l'hôte B dans notre cas. Le serveur récursif de la zone victime .dz. n'ayant pas de réponse, retransmet la requête au serveur de noms de la zone attaque.dz. qui retransmet la réponse en correspondance. Cette étape constitue le première phase de l'attaque et a pour finalité l'obtention de l'identifiant de la requête courante du serveur récursif de la zone victime.dz. (prédiction de l'identifiant courant)

L'attaque proprement dite commence à partir de la seconde phase. L'hôte A envoie une requête pour obtenir l'adresse de www.mabanque.dz.; le but étant de polluer le cache avec une fausse réponse (fausse adresse IP). Le serveur récursif de la zone victime.dz. va répercuter la requête reçue sur le serveur autoritaire de la zone mabanque.dz. En parallèle, l'hôte A inonde le serveur récursif de la zone victime.dz. avec des réponses indiquant que www.mabanque.dz. possède l'adresse IP de l'hôte A(la fausse réponse). Les réponses transmises sont identiques hormis l'identifiant associé qui se voit incrémenté à chaque réponse en partant de l'identifiant initial récupéré de la première phase citée

précédemment. Si le bon identifiant est atteint, le serveur récursif enregistre la réponse. Toute réponse qui émanerait par la suite du serveur autoritaire de la zone mabanque.dz. sera ignorée.

Cette attaque peut être couplée à un DoS sur le serveur autoritaire de la zone mabanque.dz. pour augmenter sa latence.

Tout hôte de la zone victime.dz. demandant l'adresse de www.mabanque.dz à son serveur récursif se voit obtenir l'adresse IP de l'hôte A qui va se substituer de ce fait au vrai serveur www.mabanque.dz.

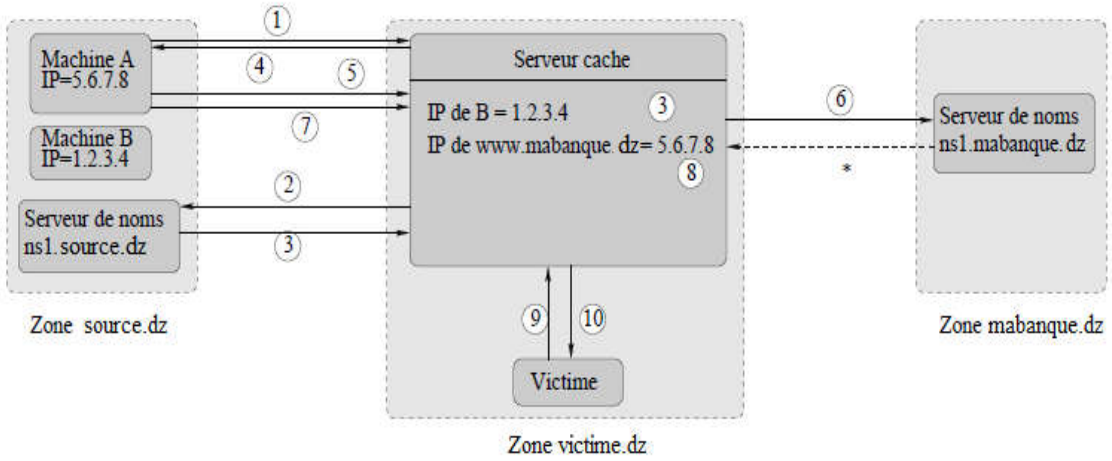

Figure 2.8 Attaque par cache poisoning

2.2.4 Corruption des données d'administration : mise à jour dynamique et transfert de zone

Lorsque une modification de topologie a lieu (nouvelle attribution d'adresse IP, présence d'un nouvel hôte, etc.), le serveur autoritaire (le primaire) associé à la zone est informé à l'aide d'un message *Dynamic Update*. Le mécanisme fonctionnel est identique aux requêtes de base; d'où sa vulnérabilité. Il existe deux manières d'exploiter cette dernière : par l'interception ou par la création d'un message de mise à jour. L'attaque par interception de messages est identique à celle présentée précédemment.

Lors de l'attaque par création de message, l'attaquant masque sa vraie adresse IP pour se faire passer pour un hôte ayant des droits pour effectuer des mises à jour dynamiques; comme un serveur DHCP par exemple.

Si cette usurpation d'adresse IP réussit, le serveur autoritaire cible met à jour son fichier de zone tenant compte des informations contenues dans message forgé. Cette attaque a pour conséquence non seulement de corrompre le serveur primaire mais de corrompre, par propagation, les serveurs autoritaires secondaires, les serveurs récursifs et les résolveurs; donc tous les constituants de l'architecture du système DNS.

Les mêmes méthodes d'attaque sont appliquées pour les messages de transfert de zone.

2.2.5 Attaques utilisant un serveur de noms

Les serveurs autoritaires qu'ils soient primaires ou secondaires sont des cibles de choix à travers la faiblesse des implémentations des commandes qui y sont disponibles. Une de ces types d'attaques a été publiée par Steve M. Bellovin [Bel95] et a été reprise par Christoph L. Schuba [Sch93]. Elle concernait la faiblesse des "r-commands" (Berkeley type rlogin, rsh, etc.). Il est à remarquer que les travaux de

Bellovin remontent à 1990 mais n'ont été publiés que bien plus tard (1995) par la non disponibilité de correctifs adéquats. Bellovin, dans son article, a montré la faiblesse du démon rlogin. Elle consistait par le fait que dernier se servait de l'adresse IP de l'hôte qui voulait s'y connecter à distance pour retrouver son nom (résolution inverse). Les droits d'accès se basait sur le nom. Un attaquant disposant d'un serveur DNS, et donc gérant son fichier de zone et son fichier de résolution inverse, peut donc modifier le nom correspondant à l'adresse IP d'un hôte distant. Le correctif du démon rlogin a consisté en l'ajout d'une double vérification; adresse IP/ nom puis nom/adresse IP; successivement.

2.3 Extensions de sécurité du DNS (DNSSEC)

Tel que pensé originellement, le système de nom de domaine ne prenait en compte le service de sécurité; probablement à cause du caractère naturellement publique des informations qu'il gérait. Tel que présenté précédemment, les failles intrinsèques liées à son fonctionnement ont induit la réflexion puis la publication des extensions de sécurité DNSSEC [AAL+05a, AAL+05b, AAL+05c, Eas99,Gun03].

Deux fonctions ont été envisagées dans le mécanisme de fonctionnement de ce système; l'intégrité et l'authentification des données DNS par l'utilisation de signatures numériques. Pour ce faire, de nouveaux enregistrements de ressource ont été créés permettant l'exploitation de clés publiques pour signer des enregistrements de ressource de base.

Ces extensions DNSSEC devait garder la compatibilité avec les anciennes implémentations; en particulier la préservation des formats messages DNS.

2.3.1 Signature du fichier de zone

Les enregistrements de ressource ayant le même nom, le même type et la même classe sont regroupés dans un Resource Record set ou RRset. Ce regroupement vise à limiter la taille du fichier de zone par la diminution du nombre de signatures. Chaque signature est liée alors à un RRset et est représentée par un enregistrement de ressource dit RRSIG. Si une zone possède plusieurs clés, chaque RRset se voit signé à l'aide de chacune des clés produisant ainsi autant d'enregistrements de ressources RRSIG. Deux exceptions à la signature existent : les enregistrements Glue Records et RRSIG ne sont pas signés. Les Glue Records (NS et A d'une délégation) étant des informations d'une zone fille dupliquée dans sa zone parente, la zone parente ne doit pas les signer car elle n'en est propriétaire. La figure 2.9 montre un exemple de fichier de zone avant et après signature. On remarque l'augmentation de le taille malgré la présence de RRset (DNSKEY RRset, NS RRset).

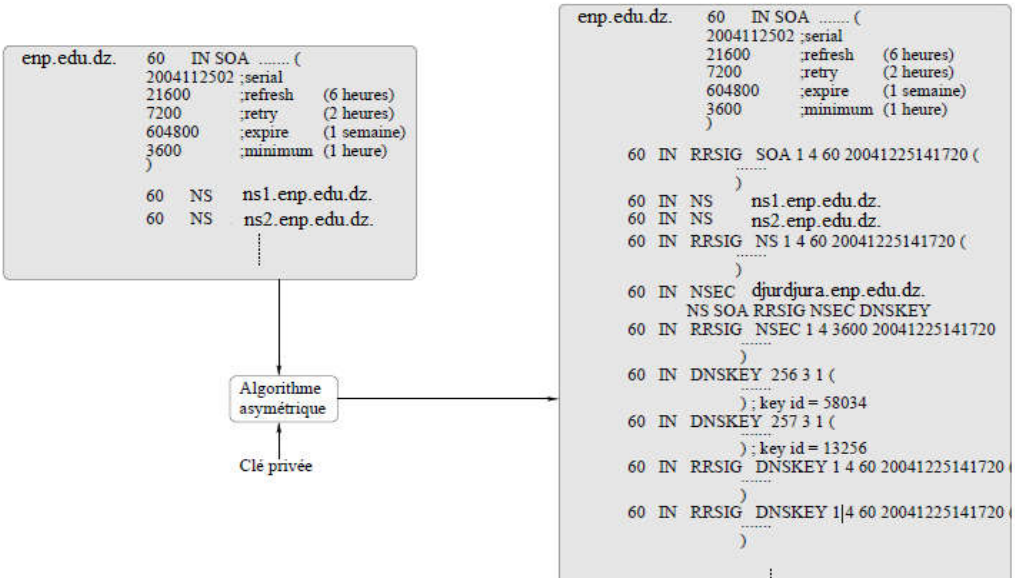

Figure 2.9 Exemple d'un fichier de zone signé

2.3.2 Les enregistrements de ressource introduit par DNSSEC

DNSSEC a introduit deux nouveaux enregistrements; DNSKEY [AAL+05a] dédié à la conservation des clés publiques (appelé anciennement KEY [Eas99]) et RRSIG [AAL+05a] pour porter les signatures numériques (appelé anciennement SIG [Eas99]).

2.3.2.1 L'enregistrement de ressource DNSKEY

A une zone DNSSEC est associée au minimum une paire de clés publique/privée. La clef privée assure la signature des enregistrements DNS contenus dans le fichier de zone. La clef publique quant à elle est disponible dans un enregistrement DNSKEY. Ce dernier contient toutes les informations nécessaires à son utilisation. Il peut alors être utilisés pour vérifier la validité de signatures des enregistrements reçus. Sa durée de vie dépendant de l'administrateur de la zone. [KG04] explicite des recommandations quant à la fréquence de renouvellement des clés de zone ainsi que leurs longueurs.
La figure 2.10 illustre le format d'un enregistrement DNSKEY. Quatre champs sont à distinguer : Flags, Protocol, Algorithm et Public Key.

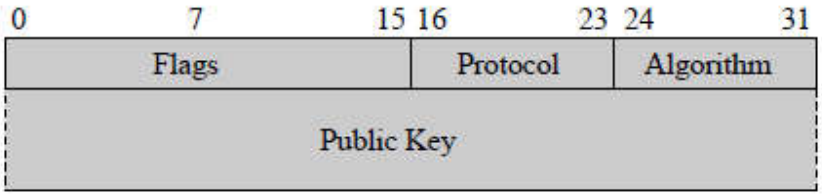

Figure 2.10 L'enregistrement de ressource DNSKEY

Seuls deux bits du champ Flags sont actuellement utilisés : il s'agit du bit 7 et du bit 14. Lorsque le bit 7 est positionné, cela signifie que l'enregistrement contient une clé de zone DNS. Le nom du propriétaire de l'enregistrement doit être le nom de la zone. Si ce bit a une valeur de zéro, l'enregistrement contient un autre type de clé publique DNS qui ne doit pas être utilisée pour vérifier les signatures des enregistrements.

Le bit 15 définit le Secure Entry Point Flag ou point d'entrée sécurisé [KSL04]. Lorsque ce bit est positionné, il signifie que la clé contenue dans l'enregistrement DNSKEY peut être utilisée comme point d'entrée sécurisé. La notion de point d'entrée sécurisé est détaillée dans la section 2.4.3 et dans l'introduction et le paragraphe 2.1 du chapitre 2.

Le champ Protocol doit toujours avoir une valeur de 3. Tout enregistrement DNSKEY ayant une valeur différente pour ce champ sera rejeté. La valeur 3 indique que la clé sera utilisée par le protocole DNSSEC. Lors de la première standardisation de DNSSEC [Eas99], cet enregistrement pouvait contenir des clés utilisables par n'importe quel protocole : IPsec, SSH, etc. Mais le RFC 3445 [MR02] a restreint l'utilisation de cet enregistrement pour les clés utilisées par le protocole DNSSEC.

Le champ Algorithm définit l'algorithme cryptographique associé à la clé publique et définit ainsi son format. Par exemple, la valeur 1 signifie que l'algorithme RSA est utilisé, la valeur 3 représente l'algorithme DSA. La clé publique est placée dans le champ Public Key.

La figure 2.11 donne un exemple d'un enregistrement DNSKEY.

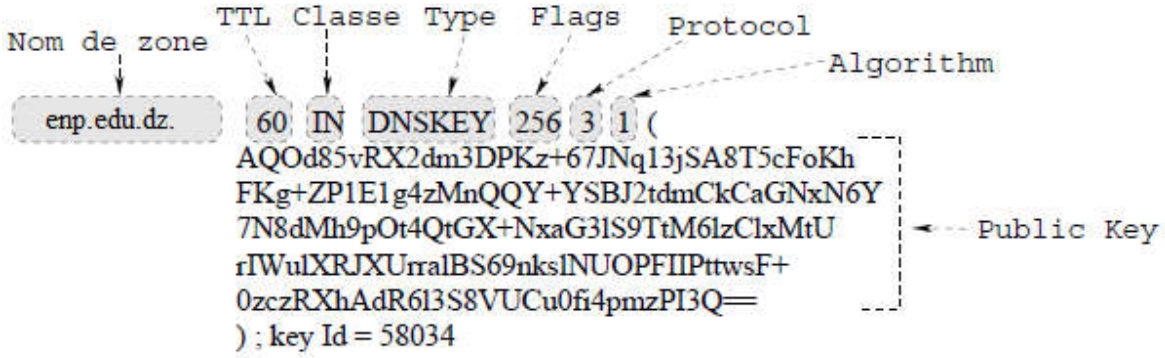

Figure 2.11 Exemple d'un enregistrement de ressource DNSKEY

Nous y trouvons le nom du propriétaire: la zone enp.edu.dz. Le TTL indique le temps pendant lequel un serveur cache qui reçoit cet enregistrement peut le conserver (60 secondes). Nous distinguons aussi les valeurs des champs présentés ci-dessus. La valeur 1 du champ Algorithm signifie que la clé contenue dans cet enregistrement est une clé RSA. L'ensemble de ces informations permet aussi de générer un identifiant de clé ou Key Id.

2.3.2.2 L'enregistrement RRSIG

Les signatures numériques générées à l'aide des clés privées de la zone doivent être présentes dans le fichier de zone pour être disponibles. C'est le rôle de l'enregistrement RRSIG.

Un enregistrement RRSIG contient les informations nécessaires à l'identification de l'enregistrement auquel il est associé, ainsi que les informations permettant d'identifier la clé ayant généré la signature numérique. La figure 2.12 présente le format d'un enregistrement RRSIG.

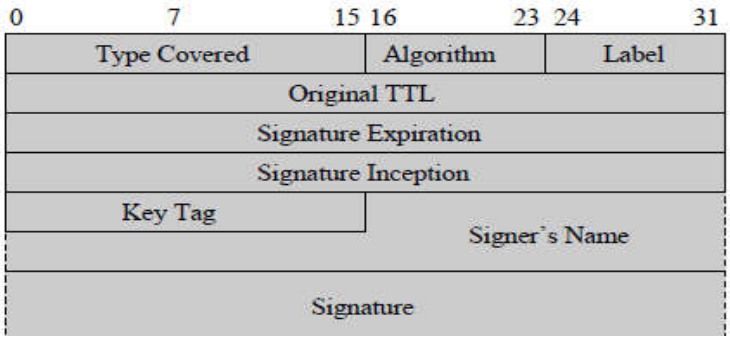

Figure 2.12 L'enregistrement RRSIG

Le champ Type Covered indique le type de l'enregistrement couvert par cette signature, s'il s'agit par exemple d'un enregistrement A ou SOA. Le champ Algorithm précise l'algorithme utilisé pour générer la signature. Les valeurs sont les mêmes que pour l'enregistrement DNSKEY : 1 représente l'algorithme RSA, etc. Le champ Label indique le nombre d'étiquettes que comportait le nom associé à l'enregistrement RRSIG lors de la création de la signature. Cette indication permet de vérifier si la réponse a été générée à partir d'un wildcard.

Un wildcard permet de définir une réponse par défaut. La figure 2.13 présente un exemple de wildcard pour un enregistrement A concernant les noms terminés par wilcard enp.edu.dz. Lorsqu'une requête parvient à un serveur de noms, celui-ci regarde si le nom demandé est présent dans son fichier de zone pour la ressource demandée. Si ce n'est pas le cas, le serveur inspecte si un wildcard correspond au nom demandé ; si un wildcard existe, le nom du wildcard est modifié dans la réponse pour correspondre au nom contenu dans la requête.

```
                                          Label
*.wilcard.enp.edu.dz.      60 IN A  131.254.200.7
*.wilcard.enp.edu.dz.      60   RRSIG  A  1  5  60  20041225141720 (
                  20041125141720 58034    enp.edu.dz.
                  d5zvOr8tfJhK7Ol//pgOsitNKTzlooqbr35q
                  216DpVwKbvnAAKfO2HEX7PuQ0wv7psySRhec
                  oVe4JunDyAnktZ5FNBHkbs7tN9jkoj+H7zEV
                  WVfIKIUPgWV4Q4XnWf6WjMoEf/Z19STcdENi
                  XFYB8SLMvcBqgLhZ7t8Y8ycKU8I=
                  )
```

Figure 2.13 Un exemple de wildcard et sa signature

Sur l'exemple donné et en supposant que le nom test.wildcard.enp.edu.dz n'existe pas dans la zone, une requête demandant l'adresse associée à ce nom aura pour réponse le wildcard dont le caractère «*» aura été remplacé par «test». Nous remarquons bien ici que le nom reçu (test.enp.edu.dz) et le nom du wildcard (*.wildcard . enp.edu.dz.) sont différents. C'est la valeur du champ Label qui va permettre la vérification de la signature. La valeur 5 de ce champ dans notre exemple indique que la signature a été générée à partir des 5 derniers labels du nom soit wildcard.enp.edu.dz. Grâce au champ label, le résolveur déduit qu'il faut vérifier la signature en

employant seulement les cinq dernières étiquettes du nom test.enp.edu.dz.

Chaque enregistrement de ressource possède un Time To Live (TTL) associé. Ce TTL est positionné par l'administrateur de la zone. Lors de la signature d'un enregistrement, son TTL est pris en compte dans la signature. Le champ Original TTL contient la valeur du TTL de l'enregistrement couvert par la signature lors de la génération de celle-ci. Dès qu'un enregistrement est placé dans une réponse ou dans la mémoire d'un serveur cache, son TTL diminue. La valeur du TTL originel contenu dans l'enregistrement RRSIG permet ainsi de pouvoir vérifier les signatures en remplaçant la valeur du TTL courant par la valeur du TTL originel lors de la vérification.

Les champs Signature Inception et Signature Expiration permettent de définir la période de validité de la signature. Ces champs contiennent une valeur sur 32 bits représentant le nombre de secondes écoulées depuis le premier janvier 1970 à minuit. Lorsqu'un résolveur reçoit une signature dont la période de validité est dépassée ou au contraire n'est pas encore atteinte, il la rejette. Le champ Key Tag contient l'identifiant de la clé (la valeur du key Id) qui a généré la signature et le champ Signer's Name identifie le propriétaire de la clé qu'un résolveur doit utiliser pour vérifier la signature. Ce champ contient le nom de la zone possédant l'enregistrement couvert par la signature.

Enfin, le champ Signature contient la signature de l'enregistrement et des données de signature, excepté le champ Signature lui-même. La figure 2.14 montre l'enregistrement RRSIG contenant la signature de l'enregistrement. SOA 58034 indique l'identifiant de la clé et enp.edu.dz. est le nom du signataire.

Figure 2.14 Un exemple d'enregistrement RRSIG

Pour vérifier cette signature, le résolveur doit donc disposer de la clé publique ayant l'identifiant 58034 ainsi que de l'enregistrement SOA de la zone enp.edu.dz
Ces deux enregistrements supplémentaires, DNSKEY et RRSIG, mettent à disposition les clés publiques de la zone ainsi que les signatures des enregistrements. Les signatures numériques protègent les enregistrements qu'elles couvrent. Elles ne protègent donc que les réponses

positives, c'est-à-dire les réponses contenant des enregistrements et leurs signatures. Mais lorsqu'il s'agit d'une réponse négative, c'est-à-dire que l'enregistrement demandé n'existe pas, la réponse envoyée par le serveur de noms est vide : elle contient juste un code d'erreur tel que NXRR (enregistrement inexistant) ou NXDOMAIN (nom inexistant). La réponse étant vide il n'est pas possible de la sécuriser grâce à des signatures numériques existant dans le fichier de zone. C'est pourquoi l'enregistrement NSEC (pour Next SECure) [Sch04] a été créé (anciennement appelé NXT [Eas99]).

2.3.2.3 L'enregistrement NSEC

Dans le protocole DNS, lorsqu'une requête porte sur une ressource ou un nom de domaine qui n'existe pas, la réponse contient uniquement un code d'erreur dans son en-tête et cet en-tête n'est pas protégé.

L'enregistrement NSEC a été créé pour pouvoir vérifier les réponses négatives. Un enregistrement NSEC contient les informations nécessaires à l'identification des enregistrements existants pour un nom donné, ainsi que le prochain nom existant dans la zone (dans l'ordre lexicographique). Ces deux informations suffisent pour prouver qu'un enregistrement ou qu'un domaine n'existe pas. La figure 2.15 montre le format d'un enregistrement NSEC.

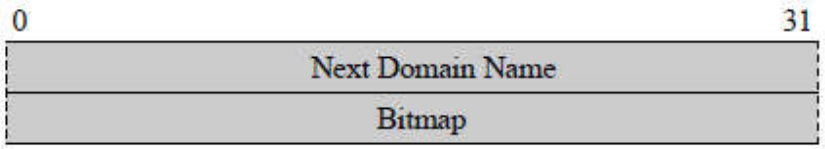

Figure 2.15 L'enregistrement NSEC

Le champ Next Domain contient le prochain nom dans la zone. Le champ Bitmap représente chacun des types d'enregistrements présents dans le fichier de zone pour le nom associé à l'enregistrement NSEC.
Lorsqu'un résolveur reçoit un enregistrement NSEC, il vérifie sa signature, puis regarde le nom associé. Si ce n'est pas le nom qu'il a demandé, mais que le nom demandé est dans l'intervalle [Nom associé,..,Next Domain Name] alors le résolveur déduit que le nom demandé n'existe pas. L'enregistrement NSEC permettant de vérifier qu'un domaine donné n'existe pas est appelé le NSEC couvrant. Il permet de prouver les réponses contenant le code d'erreur NXDOMAIN.
Les enregistrements NSEC permettent aussi de prouver qu'un enregistrement donné n'existe pas. Lorsqu'un résolveur reçoit un enregistrement NSEC pour le nom qu'il a demandé, il regarde dans le champ Bitmap si le type d'enregistrement demandé est présent. Si ce type ne s'y trouve pas alors l'enregistrement demandé n'existe pas. L'enregistrement NSEC étant protégé par une signature numérique, nous pouvons prouver que les données qu'il contient n'ont pas été modifiées.
Un exemple d'enregistrement NSEC est présenté sur la figure 2.16.

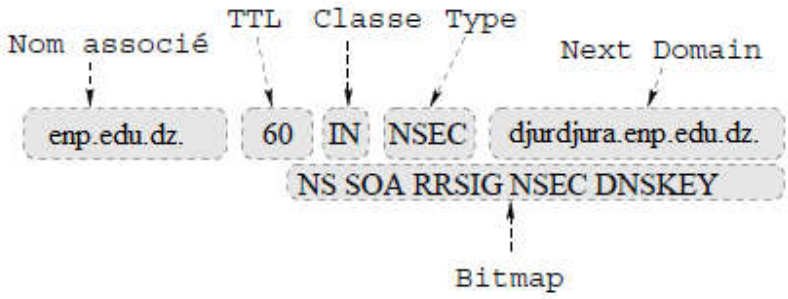

Figure 2.16 Un exemple d'enregistrement NSEC

Cet enregistrement spécifie qu'il n'existe pas de nom entre enp.edu.dz et djurdjura.enp.edu.dz et que les types d'enregistrements existants pour le nom enp.edu.dz sont NS, SOA, RRSIG, NSEC, DNSKEY.

Ainsi un résolveur demandant l'adresse de a.enp.edu.dz recevra cet enregistrement NSEC signé. Après avoir vérifié la signature, le résolveur vérifiera que le nom demandé est dans l'intervalle [enp.edu.dz ,djurdjura.enp.edu.dz].

2.3.3 Chaîne de confiance et authentification de clé en cascade

Dans le processus de vérification des signatures numériques présenté sur la figure 2.11, Bob croit la signature du message envoyé par Alice car il a confiance dans la clé publique d'Alice, cette clé ayant été obtenue via une infrastructure de gestion de clés ou par un processus sécurisé.

Dans DNSSEC, avoir confiance en une clé est ce que nous appelons posséder un point d'entrée sécurisé (SEP) [KSL04] ou une clé de confiance. Cette clé de confiance est une clé publique d'une zone et est configurée par l'administrateur du résolveur. Le résolveur fait alors confiance à cette clé et il peut ainsi vérifier la signature du DNSKEY RRset de la zone grâce à sa clé de confiance. Une fois la signature vérifiée, le résolveur fait confiance aux clés se trouvant dans le DNSKEY RRset et peut ainsi vérifier les signatures des autres enregistrements de la zone grâce aux clés contenues dans le DNSKEY RRset. Ainsi avec une clé de zone configurée comme clé de confiance, un résolveur peut, après avoir vérifié les signatures, étendre sa confiance à toutes les clés de la zone.

En suivant ce modèle, il faudrait configurer dans un résolveur une clé de confiance pour chaque zone, ce qui est impossible car il existe plusieurs millions de zones dans l'arbre DNS (plus de 30 millions au quatrième trimestre 2004). Pour résoudre ce problème, DNSSEC tire partie de la structure arborescente de l'architecture DNS. Un nouvel enregistrement, l'enregistrement Delegation Signer (DS) [Gun03], est placé au point de délégation. Cet enregistrement, conservé dans le fichier de zone de la zone parente, permet d'identifier une clé de sa zone fille.

2.3.3.1 L'enregistrement DS

Un enregistrement DS référence un enregistrement DNSKEY, il est utilisé pour authentifier une clé. L'enregistrement DS contient les informations nécessaires à l'identification

d'une clé, telles que le Key Tag, l'algorithme utilisé et un haché de la clé concernée. En obtenant l'enregistrement DNSKEY référencé par un enregistrement DS, puis en vérifiant la signature de cet enregistrement DS, un résolveur peut authentifier la clé contenue dans l'enregistrement DNSKEY.

L'enregistrement DS et l'enregistrement DNSKEY possèdent le même nom de propriétaire, mais l'enregistrement DS est toujours conservé dans la zone parente au point de délégation. Par exemple, l'enregistrement DS pour enp.edu.dz est conservé dans la zone enp.edu.dz. tandis que l'enregistrement DNSKEY est conservé dans la zone fille test.enp.edu.dz La figure 2.17 décrit le format d'un enregistrement DS.

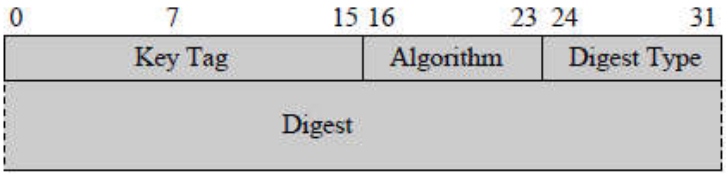

Figure 2.17 L'enregistrement DS

Le champ Key Tag contient l'identifiant de la clé référencée dans l'enregistrement DS. Le champ Algorithm est identique au champ Algorithm de l'enregistrement DNSKEY. Le champ Digest Type contient l'algorithme utilisé pour générer le haché de l'enregistrement DNSKEY. Ce haché est placé dans le champ Digest. Il s'agit en fait d'un haché de la concaténation du nom du propriétaire de l'enregistrement DNSKEY et des champs Flags, Protocol, Algorithm et Public Key de cet enregistrement.

La figure 2.18 montre un enregistrement DNSKEY et son enregistrement DS associé. L'enregistrement DNSKEY se trouve dans la zone fille test.enp.edu.dz et l'enregistrement DS se trouve dans la zone enp.edu.dz.

Lorsqu'une zone DNS déploie DNSSEC, elle doit dans un premier temps créer ses clés de zone et signer son fichier de zone. Puis, l'administrateur de cette nouvelle zone sécurisée doit contacter l'administrateur de sa zone parente afin de créer les enregistrements DS associés aux clés de la nouvelle zone sécurisée.

Le problème du démarrage est qu'il n'existe pas encore de lien de confiance entre la zone parente et la zone fille. Les deux administrateurs doivent alors utiliser un autre moyen pour s'authentifier mutuellement (certificats X.509, formulaires administratifs, etc.).

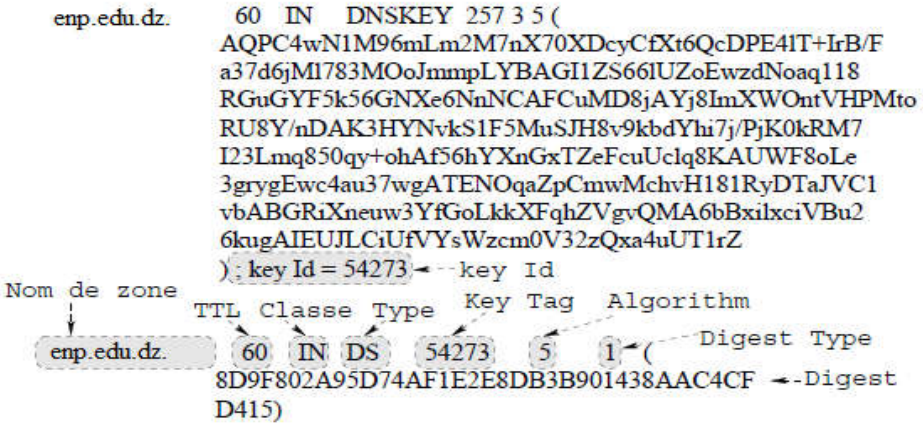

Figure 2.18 Un exemple d'enregistrement DS

2.3.3.2 Les deux types de clés ZSK et KSK

L'enregistrement DS crée un lien entre une zone parente et une zone fille en référençant une de ses clés. Le modèle Delegation Signer introduit donc une distinction entre les clés : les clés ayant un enregistrement DS associé, appelées Key Signing Key (KSK) et les clés n'ayant pas d'enregistrement DS associé, appelées Zone Signing Key (ZSK).

La création d'un enregistrement DS nécessitant une interaction entre les deux zones (échange de clés de la zone fille vers la zone parente, puis création de l'enregistrement DS et de sa signature), la distinction entre ces deux types de clés a été faite pour minimiser la charge de travail induite par la création et la mise à jour des enregistrements DS.

En effet, les ZSK signent tous les enregistrements d'une zone. Comme les ZSK n'ont pas d'enregistrement DS associé, lorsque une zone décide de changer une de ses ZSK, il n'y a aucun travail supplémentaire pour sa zone parente, ni d'échange entre les deux zones.

Les KSK ne signent que les enregistrements DNSKEY. Comme une KSK possède un enregistrement DS associé, lorsqu'une zone change une de ses KSK, il doit y avoir un échange d'informations entre cette zone et sa zone parente afin de mettre à jour l'enregistrement DS correspondant.

Le document [KG04] compile un certain nombre d'informations issues de différentes expérimentations et fournit un certain nombre de conseils, notamment sur la période de renouvellement des clés, le nombre de clés à utiliser, leur taille, la fréquence de renouvellement de ces clés en fonction de leur taille, etc. [KG04] conseille d'utiliser une KSK pendant quelques mois avant de la renouveler et de renouveler une ZSK après un mois d'utilisation.

2.3.3.3 L'authentification de clés en cascade

L'ensemble des éléments présentés ci-dessus (un point d'entrée sécurisé (SEP), un enregistrement DS authentifiant une clé de la zone fille qui est elle-même signée, etc.), permet de construire une chaîne de confiance [Gie01]. Le début de la chaîne est le SEP, chaque maillon est constitué d'un DNSKEY RRset, dont au moins une signature générée par une KSK est vérifiée et d'un enregistrement DS dont au moins une signature générée par une ZSK est vérifiée.

Le lien entre les maillons est représenté par un enregistrement DS authentifiant une KSK qui a

signé le DNSKEY RRset. Le dernier maillon de la chaîne contient la ressource demandée et son enregistrement RRSIG associé. La figure 2.22 représente cette authentification en cascade.

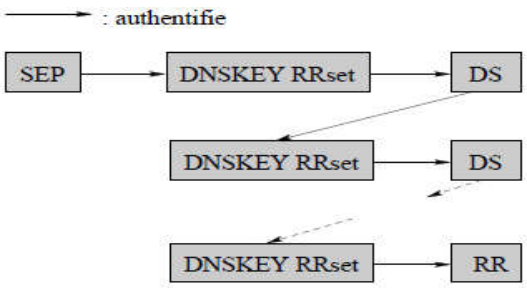

Figure 2.19 L'authentification de clés en cascade

Dans chaque maillon de la chaîne, nous retrouvons le même schéma : un enregistrement DS permet d'avoir confiance en une KSK, une KSK permet d'avoir confiance en l'ensemble des clés de la zone (KSK et ZSK) et les ZSK permettent alors d'avoir confiance dans les autres enregistrements.

Bien que le modèle Delegation Signer introduise une distinction au niveau des clés, il n'y a pas d'obligation au niveau des zones sur l'utilisation de ces deux types de clés. Nous pouvons donc nous attendre à trouver, dans les petites zones, une clé ayant les deux fonctions (KSK et ZSK), c'est-à-dire que cette clé signe tous les enregistrements de la zone et aura un DS associé dans sa zone parente.

2.3.4 La résolution de noms DNSSEC

Le principe de résolution de noms reste le même pour DNSSEC. Une des contraintes lors de la conception des extensions de sécurité du DNS était la compatibilité entre DNS et DNSSEC: un équipement ne comprenant pas DNSSEC doit être en mesure d'effectuer des résolutions de noms DNS sans avoir de problèmes. Le comportement d'un ancien équipement, face à des enregistrements qu'il ne connaît pas, est tout simplement de les ignorer.
Le schéma d'échange des messages reste donc le même, les entités communicantes sont les mêmes que sur la figure 2.4. Le contenu des réponses est plus important car il contient des enregistrements supplémentaires.
Un serveur de noms sécurisé (possédant un fichier de zone signé) va inclure dans ses réponses le matériel cryptographique nécessaire, tels que ses enregistrements DNSKEY. De plus, un enregistrement est toujours envoyé avec ses signatures associées.

De la même manière, si la réponse est une délégation, le serveur sécurisé enverra en réponse les enregistrements DS adéquats et leurs signatures. Si à un moment de la résolution sécurisée, il manque au serveur cache ou au résolveur certains enregistrements de sécurité, une requête spécifique sera envoyée pour récupérer les enregistrements manquants.

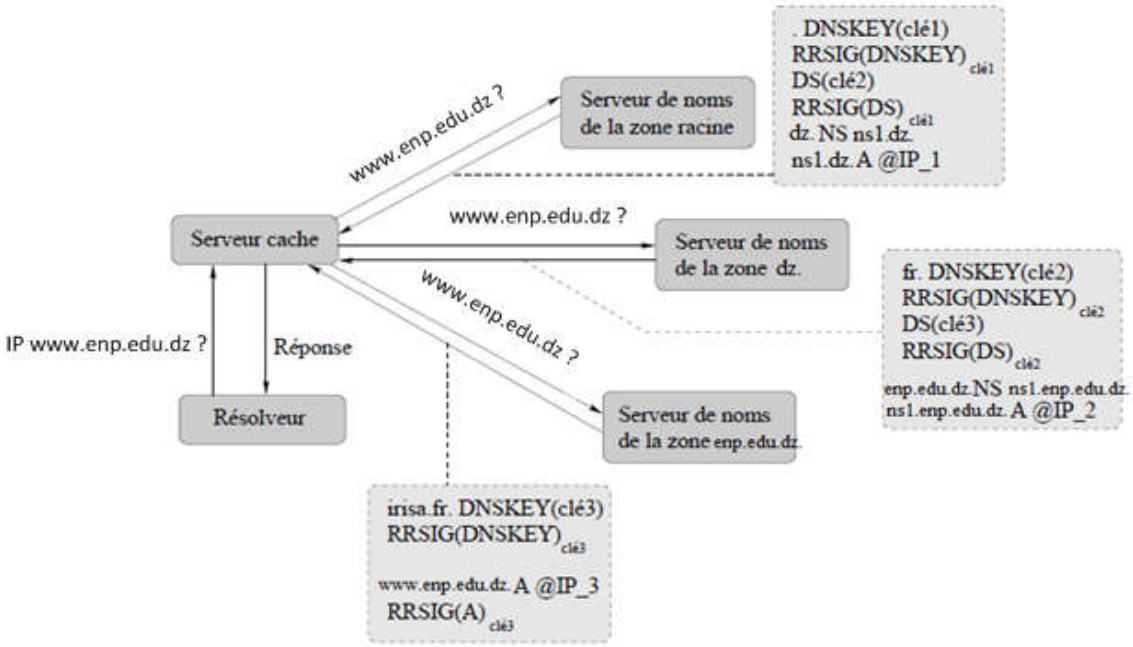

Figure 2.20 La résolution de noms sécurisée

La figure 2.20 montre une résolution de noms sécurisée. Les interactions entre les entités sont les mêmes que lors d'une résolution de noms DNS, mais cette fois-ci les messages contiennent le matériel cryptographique nécessaire à la vérification de signatures et à l'établissement d'une chaîne de confiance.

Deux algorithmes peuvent être utilisés pour effectuer la résolution de noms sécurisée : il s'agit des algorithmes Top-Down et Bottom-Up.

La première étape de ces deux algorithmes est la même : récupérer la ressource demandée, comme présenté sur la figure 2.5, sans se soucier du matériel cryptographique pour le moment. Si la ressource se trouve dans une zone sécurisée, alors la réponse contiendra la ressource et sa signature. Nous montrons dans les deux paragraphes suivants le fonctionnement de ces deux algorithmes dans l'exemple présenté sur la figure 2.23. Nous voulons obtenir l'adresse IP associée à www.enp.edu.dz

2.3.4.1 L'algorithme Top-Down

1. Nous obtenons la ressource demandée grâce à une requête DNS. La ressource est dans une zone sécurisée, sa signature est donc présente dans la réponse. Nous avons donc sur notre exemple, un enregistrement A et un enregistrement RRSIG pour www.enp.edu.dz
2. L'algorithme regarde le nom du signataire de l'enregistrement, présent dans le champ Signer's Name de l'enregistrement RRSIG, il s'agit de enp.edu.dz.. Puis, il cherche la clé de confiance la plus proche de ce nom, qu'il possède. Cette recherche est effectuée en ôtant une

par une les étiquettes de gauche du nom du signataire, jusqu'à obtenir le nom associé à une clé de confiance, ici nous obtenons clél comme clé de confiance la plus proche.

3. clél est la clé de la zone racine, nous demandons donc au serveur de noms de la zone racine ses enregistrements DNSKEY. L'algorithme utilise alors clél pour vérifier les signatures. Il fait désormais confiance à toutes les clés contenues dans le DNSKEY RRset de la zone racine.
4. L'algorithme demande les clés de la zone enp.edu.dz.. La racine ne possède pas cette information, elle transmet donc la délégation appropriée, les enregistrements DS de dz. et leurs signatures, ainsi que les Glue Records. L'algorithme utilise alors le DNSKEY RRset obtenu précédemment pour vérifier les signatures des enregistrements DS.
5. Puis l'algorithme interroge le serveur de noms de dz. et obtient de la même manière le DNSKEY RRset de dz. et la délégation enp.edu.dz.. L'algorithme vérifie qu'un enregistrement DS de dz. authentifie une clé du DNSKEY RRset de la zone dz., puis que cette clé vérifie une signature du DNSKEY RRset de la zone dz..
6. L'algorithme vérifie ensuite les signatures du DS de enp.edu.dz.et interroge le serveur de noms de enp.edu.dz. fourni dans la délégation.
7. L'algorithme obtient et vérifie le lien entre une clé et un enregistrement DS, et vérifie alors les signatures du DNSKEY RRset de la zone enp.edu.dz.avec cette clé.
8. Et finalement, ayant obtenu les clés des signataires de la ressource, l'algorithme vérifie les signatures de la ressource demandée.

À chaque étape cet algorithme valide une ressource qui vient augmenter l'ensemble des ressources en lesquelles il a confiance.

2.3.4.2 L'algorithme Bottom-Up

L'algorithme Bottom-Up est orienté signature, c'est-à-dire qu'au lieu de partir d'une clé de confiance pour arriver à la ressource, il va regarder qui a signé la ressource et remonter jusqu'à une clé de confiance.

Nous obtenons la ressource demandée grâce à une requête DNS. La ressource est dans une zone sécurisée, sa signature est donc présente dans la réponse. Nous avons donc sur notre exemple, un enregistrement A et un enregistrement RRSIG de www.enp.edu.dz

L'algorithme regarde le nom du signataire de l'enregistrement présent dans le champ Signer's Name de l'enregistrement RRSIG; il s'agit de enp.edu.dz. Il demande donc les clés de enp.edu.dz et obtient le DNSKEY RRset de la zone enp.edu.dz. et ses signatures.

L'algorithme regarde si une des clés reçues est une clé de confiance, ce n'est pas le cas. Il demande donc les enregistrements DS existant pour la zone enp.edu.dz. La zone dz. lui répond avec les enregistrements DS et leurs signatures.
L'algorithme regarde le nom du signataire des enregistrements DS, il s'agit de dz. il demande donc les clés de dz.

Il compare ces clés avec ses clés de confiance, aucune ne correspond il continue donc et demande

les enregistrements DS pour dz. La zone racine lui répond avec les enregistrements DS signés. L'algorithme regarde le nom du signataire des enregistrements DS, il s'agit de la racine, il demande donc les clés de la zone racine.

Il compare ces clés avec ses clés de confiance ; une clé correspond : il s'agit de clé1. La vérification des signatures peut donc commencer.

L'algorithme vérifie les signatures des clés de la zone racine, puis les signatures du DS de dz. Il vérifie l'existence d'un lien entre un enregistrement DS et une clé de dz.. Puis il vérifie les signatures des clés de dz. et les signatures des enregistrements DS de enp.edu.dz. Il vérifie l'existence d'un lien entre un enregistrement DS et une clé de enp.edu.dz. Puis il vérifie les signatures des clés de enp.edu.dz.. Finalement, il vérifie les signatures de la ressource obtenue à l'étape 1.

Ce qui différencie ces deux algorithmes est le sens de récupération des informations. La chaîne de confiance est toujours établie de la même manière. Nous pouvons remarquer que l'algorithme Top-Down se prête bien au comportement d'un serveur cache qui valide à la volée les enregistrements. De plus, lors de la première requête pour obtenir la ressource demandée, les serveurs autoritaires envoient généralement le matériel cryptographique nécessaire. Ceci implique donc que, lors de la première requête pour obtenir la ressource, notre serveur cache ayant suivi les délégations, il a déjà obtenu la majorité des informations nécessaires.
En revanche, pour un résolveur situé derrière son serveur cache récursif, lors de la première requête il obtient ce qu'il a demandé, même si son serveur cache a obtenu beaucoup plus d'informations. Pour ce type de résolveur [Pfl03,Pel04], l'utilisation de l'un ou l'autre algorithme est identique. Le trafic pour obtenir les enregistrements nécessaires se faisant entre le résolveur et le serveur cache qui normalement les possède déjà. Nous pouvons aussi remarquer une similarité entre l'utilisation de ces algorithmes et les résolutions de noms récursive ou itérative.

Dans les trois paragraphes suivants, nous présentons quelques éléments spécifiques utilisés durant une résolution de noms sécurisée : le bit Authenticated Data [VG03], le bit Checking Disable [Eas99] et l'extension EDNSO [Vix99].

2.3.4.3 Le bit Authenticated Data (AD)

Le bit AD [WG03] est un bit de l'en-tête des messages DNS. Ce bit est positionné dans une réponse uniquement si tous les enregistrements contenus dans cette réponse ont été cryptographiquement vérifiés.

Un résolveur DNSSEC appelé par une application peut lui retourner, grâce à la vérification des signatures, le statut de sécurité du RRset reçu. Néanmoins, certains résolveurs ne possèdent pas les ressources suffisantes pour effectuer la vérification cryptographique des enregistrements. Ces résolveurs font généralement confiance à leur serveur cache récursif qui effectue la vérification pour eux.

Le serveur cache utilise alors le bit AD dans l'en-tête de ses réponses pour indiquer que les enregistrements envoyés ont été vérifiés.
Un résolveur ne doit faire confiance au bit AD, positionné par son serveur cache, que s'il possède un canal d'échange sécurisé avec son serveur cache. Il faut aussi que la réponse soit passée par ce canal et que le résolveur soit explicitement configuré pour faire confiance au serveur cache. Ce canal sécurisé permet d'empêcher un attaquant de placer de faux enregistrements dans une réponse dont le bit AD est à 1, ce qui aurait pour conséquence que le résolveur fasse confiance à cette fausse réponse.

2.3.4.4 Le bit Checking Disable (CD)

Un autre bit de l'en-tête des messages DNS peut être utilisé par un résolveur pour spécifier un comportement particulier de la part du serveur qui reçoit la requête : il s'agit du bit Checking Disable (CD) [Eas99].
Le bit CD signifie que l'entité ayant envoyé la requête accepte des enregistrements même si ceux-ci n'ont pas été cryptographiquement vérifiés par l'émetteur de la réponse. Ce bit est généralement utilisé par les résolveurs dont la politique de sécurité locale est d'effectuer eux-mêmes la vérification des signatures.

2.3.4.5 La taille des messages DNSSEC et l'extension EDNSO

Historiquement, le protocole DNS fonctionne au-dessus du protocole UDP avec une taille de paquet de 512 octets. Cette limite était largement suffisante pour contenir une réponse DNS.
Néanmoins, pour une réponse devant contenir tout le matériel cryptographique nécessaire à DNSSEC, cette limite est très souvent dépassée. Pour pallier ce problème, DNSSEC possède un pseudo-enregistrement, il s'agit de EDNSO qui n'est pas conservé en cache. Ce pseudo-enregistrement est ajouté dans la partie optionnelle d'une requête : il permet de spécifier entre autre la taille de message que supporte l'émetteur de la requête et s'il supporte DNSSEC. L'entité recevant la requête peut alors adapter la taille de la réponse en fonction de cette information et ainsi placer dans sa réponse des enregistrements supplémentaires. Un cas de dysfonctionnement assez subtil peut apparaître avec une mauvaise configuration de EDNSO, comme le montre la figure 2.21.

Prenons un résolveur sur un brin de réseau dont le Maximum Transfert Unit (MTU) vaut 1280 octets et son serveur cache récursif sur un brin de réseau dont le MTU vaut 1500 octets. Un équipement filtrant le trafic ICMP existe entre le résolveur et le serveur cache. Si le résolveur indique une taille de 1500 octets en utilisant EDNSO alors sa requête, de petite taille, parvient au serveur cache.

Nous avons vu que cette résolution de noms nécessite un point d'entrée sécurisé configuré dans les résolveurs. Il y a ainsi une duplication des informations DNSSEC dans les clients et dans les serveurs. Aucun mécanisme n'est intégré à DNSSEC pour garder la cohérence de ces clés dupliquées, lorsqu'une zone décide de les renouveler. Dans les deux chapitres suivants nous présentons la problématique d'un tel renouvellement de clés dans DNSSEC pour les clients et pour les serveurs. Dans le premier cas, nous abordons le problème de cohérence entre les clés dupliquées, présentes dans les résolveurs et dans les serveurs de noms ; dans le second cas, le problème de cohérence lors d'un renouvellement de clé entre les enregistrements DNSKEY de la zone fille et les enregistrements DS de la zone mère

2.4 Motivations pour un serveur DNS sur SoC

Une implémentation DNS ou DNSSEC efficace doit offrir :

- Une meilleure fiabilité: le système DNS peut être comme un interrupteur logique. Il suffit de le couper et l'accès haut niveau au réseau Internet est suspendu. L'équipement utilisé doit être fiable épaulé par des mécanismes de redondance.
- Une meilleure performance : plus la dimension du réseau est importante, plus le nombre de requêtes l'est aussi. Ce qui peut engendrer, si la performance de l'équipement n'est pas suffisante, une latence plus importante. Autrement exprimé, le

temps d'accès à Internet peut dépendre du temps de réponse du serveur DNS. Cette nécessité de performance est encore plus présente sachant le recours aux outils de chiffrement pour garantir l'authenticité et l'intégrité des données dans le cas de DNSSEC.

- Une meilleure sécurité : l'accumulation de composants logiciels (systèmes d'exploitation, applications métiers, utilitaires, services, etc…) sur un même serveur multiplie le nombre de failles potentielles et leurs interactions.
- Si l'on revient au problème de performance, plus on utilise de couches virtuelles, plus la latence devient importante. Souvent, sur les systèmes d'exploitation Unix, les services sont isolés entre eux (chroot) pour éviter des influences mutuelles négatives (sécurité). Une isolation physique serait meilleure qu'une isolation logique.
- Un gain d'espace : Comme cité plus haut, la fiabilité implique des mécanismes de redondance et donc, une multiplicité de serveurs à mettre en œuvre. Les recommandations de séparer les types de serveurs DNS, de consacrer une machine par service ou d'introduire des systèmes d'équilibrage et de répartition de charges engendrent encore d'avantage de ressources et d'espace. Le critère espace peut paraître donc important. Une conséquence directe peut être une réduction de la consommation d'énergie. La miniaturisation deviendrait alors une réponse à cette contrainte.

Nous déduisons trois arguments majeurs qui ont motivé notre idée de développer un serveur DNS sur un circuit:

- La virtualisation est une source de latence et de multiplication de sources de failles de sécurité.
- L'accumulation de composants logiciels (systèmes d'exploitation, applications métiers, utilitaires, services, etc.) sur un même serveur multiplie le nombre de failles potentielles et leurs interactions. Les négligences citées ci-dessus sont dues en grande partie à cette flexibilité qui facilite la coexistence d'applications et de services variés sur une même machine (ou un ensemble de machines). Par ailleurs, la multiplication des couches virtuelles engendrée par cette flexibilité [Tan06] est une source de latence évidente.
- Les serveurs d'infrastructures réseaux ne sont pas optimisés
- Si l'on tient compte des analyses de performances publiées [Kol05], il y apparaît que les ressources de calcul des serveurs utilisés sont faiblement utilisées en particulier dans le cas des serveurs autoritaires. La recherche et le transfert de données sont les fonctions majoritairement utilisées. Un serveur généraliste ne peut être optimisé pour tous les services impliqués dans le fonctionnement d'un réseau.
- Le progrès considérable dans le domaine de la miniaturisation des circuits et les outils de conception associés, nouveaux paradigmes dans le domaine des processeurs réseaux. Il est plus facile aujourd'hui d'intégrer sur un même circuit plusieurs fonctions complexes. On y remarque d'ailleurs une incursion de ce type de circuits dans l'implémentation de certains services réseau [CYC03, KHF00+,KH02,SF03]

Ainsi optimisée, cette implémentation conduit à un serveur DNS qui devient alors :

- dédié. Il sera plus fiable qu'un système généraliste composé (meilleure fiabilité).
- dépouillé des ressources inutiles ou pouvant nuire à un service donné (meilleure performance). La performance peut être vue comme une protection accrue contre le

déni de service.
- physiquement cloisonné. Cela rappelle la fonction « chroot » des systèmes POSIX permettant un cloisonnement logique du service cible (meilleure sécurité).
- de taille plus réduite. La redondance et la séparation des fonctions nécessaires au fonctionnement du système ne seront pas une gêne quant à la multiplicité des machines (gain d'espace et d'énergie consommée).

Par une analyse fine aussi bien des structures de données manipulées que des algorithmes mis en jeux, on peut rechercher une architecture dédiée forcément optimisée qui doit conduire aux performances recherchées.
Les évolutions technologiques actuelles dans le domaine de la conception des circuits permet aussi bien sous les angles de la méthodologie, des plateformes de conception, des algorithmes ainsi que du nombre de ressources pouvant être intégrées sur une même puce, d'aboutir à une miniaturisation avérée.

2.5 Conclusion

La migration du DNS vers des plateformes dédiées fait partie d'une progressive mais inexorable migration des services de réseau critique vers des formes d'implémentation dédiées. Sans trop se tromper, on peut tenter un parallèle qui nous paraît évident : le premier service à tendre vers cette migration a été le routeur en tant qu'équipement de la périphérie ou du cœur du réseau. Il était autrefois implémenté sur une plateforme polyvalente autour d'un système d'exploitation donné. Le routeur qui en résultait, malheureusement, n'a pas été particulièrement rapide, fiable, ou sécurisé. Son évolution dictée par les besoins, a atteint, comme on peut le voir aujourd'hui, des niveaux spectaculaires de performance, de fiabilité et de sécurité. Peu de monde réalisent que ces routeurs ont été implémentés d'abord sur des ordinateurs d'usage général; des PDP 11 en l'occurrence.

CHAPITRE III

Le langage de spécification SDL

Nous avons donné précédemment un aperçu sur la pratique actuelle dans l'ingénierie des protocoles et sur les approches de conception mixte matérielle et logiciel pour les systèmes sur puce. Nous poursuivons dans le présent développement avec la problématique de la combinaison des techniques existantes de l'ingénierie des protocoles avec les approches de conception des systèmes sur puce. Notre objectif est de développer une méthodologie pour la mise en œuvre efficace de protocole de communication sur des systèmes sur puce et l'appliquer à l'implémentation du serveur autoritaire DNS/DNSSEC.

3.1 Préambule

Lors de la conception des systèmes de communication sur puce, les objectifs à atteindre se résument au développement ou (et) au choix de protocoles efficaces, l'implémentation de ces derniers et leur intégration dans un système complet. Une méthodologie de conception intégrée devrait soutenir toutes les étapes du flux de conception et doit permettre des temps de développement courts.

Nous avons conclu notre précédent chapitre sur notre choix du langage SDL (Spécification et Description Language) comme outil clef de notre méthodologie de développement. Comme rapporté, il est devenu un langage de développement populaire pour la conception de protocoles de communication. Les outils de développement, tels que la suite Telelogic TAU SDL [Tel06], fournissent les outils de conception, de simulation et de validation. Cet ensemble de fonctions est mis en œuvre sous forme logicielle. Des recherches ont été menées afin d'étendre le flux de conception dans l'optique, par exemple, de préciser les contraintes temps réel, le développement de modèles de mise en œuvre plus efficaces ou la génération automatique d'implémentations matérielles. Un certain nombre d'implémentations de protocole dérivé de spécifications formelles et utilisant SDL a été décrit dans la littérature [Dmj98,Dmv01].

Les méthodologies de conception qui ne sont pas basées sur SDL sont également envisageables. Nous pouvons citer par exemple, SystemC [Sys05] avec son processus de raffinement potentiellement simple pour aboutir à des implémentations mixtes matérielles et logicielles, Simulink [Mat07] avec son environnement de simulation intégré avec d'autres couches de protocole y compris les canaux physiques, ou UML (Unified Modeling Language) qui une technique de modélisation très populaire.

Toutefois, les avantages d'un flux de conception basé sur SDL sont le niveau d'abstraction élevé, l'indépendance de mise en œuvre des spécifications par rapport au matériel et sa sémantique formelle permettant la vérification et la validation du protocole. Par conséquent, on ne s'étendra pas sur les approches alternatives.

3.2 Conception de système avec SDL

3.2.1 Spécification et vérification

Les concepts les plus importants associés au langage SDL ont été présentés au chapitre précédent. Grâce à sa capacité de spécifier formellement les types de données tels que les unités de protocole de données et le comportement du système d'une manière indépendante de l'implémentation, ce langage a été utilisé dans bon nombre de standardisation de protocoles de communication. On cite pour exemple les normes IEEE 802.14.1 ou 802.14.4.

La vérification formelle de spécifications SDL a été décrite par un certain nombre de travaux ([MIJ03], [BDHS00], [SS01], [RB98], [JG01]). Dans cette dernière publication par exemple, les auteurs ont présenté leurs expériences de la vérification et de la validation du protocole industriel MASCARA (Mobile Access Scheme based on Contention and Reservation for ATM) orienté vers le

contrôle d'accès au support sans fil. Il a été spécifié en utilisant SDL sur un total d'environ 300 pages. Cette spécification a été traduite automatiquement en un équivalent en langage de spécification IF. Le langage IF [BFG+99] a été défini comme une représentation intermédiaire pour les systèmes asynchrones temporisés. Les principaux défis pour la vérification de ce système étaient la réduction de la complexité du modèle et l'analyse séparée des sous systèmes. Les auteurs ont appliqué une combinaison de techniques statiques et dynamiques pour la vérification du modèle et la réduction de sa complexité. La figure 4.1 illustre la chaîne d'outils utilisée pour la vérification telle que présentée dans [JG01].

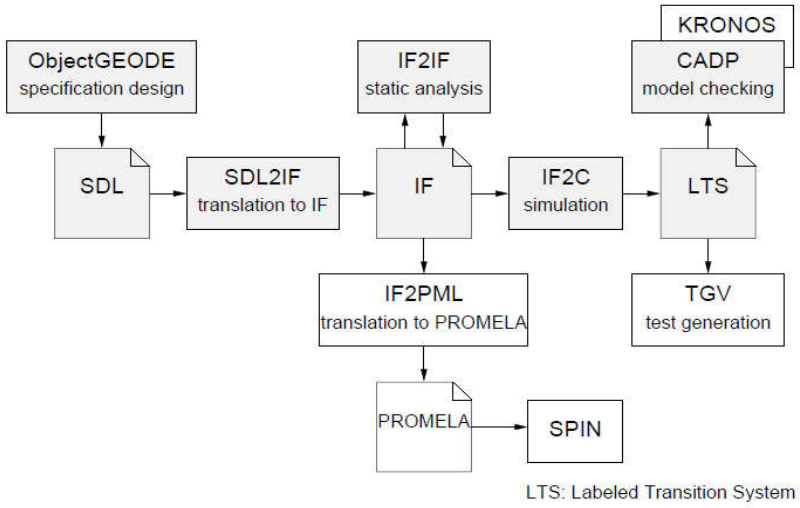

Figure 3.1 Outils et langages utilisés pour la vérification expérimentale du protocole MASCARA [JG01]

Le model checker intégré à CADP [GLM02] a été utilisé pour analyser le système de transition étiqueté. Pour démontrer l'utilité de cette approche, un certain nombre de propriétés génériques, tel que l'absence d'inter blocages et le comportement attendu du protocole a été vérifiée. Pour spécifier ces propriétés, elles doivent être décrites par des formules en logique temporelle, ou encore par le biais d'automates finis pour exprimer le système de transitions étiqueté. Un comportement du protocole jugé contraire à une propriété spécifiée pourrait être visualisé à l'aide du diagramme de séquence. La spécification de ces propriétés directement sur le modèle SDL n'était pas possible.

De nombreux articles ([Leu95], [FL98], [dW04], [Bou07]) soulignent que les concepts du langage SDL pour spécifier le comportement temporel sont insuffisants pour modéliser des systèmes temps réel. A cet égard, les limitations les plus sévères de ce langage sont relatives à l'absence de concepts pour préciser les délais de réactivité ; c'est à dire imposer un temps de réaction au système à un événement, des retards de communication entre les canaux, des horloges pour des systèmes différents. SDL a une sémantique de temporisation abstraite qui se traduit par un temps d'activité du système sans aucun déphasage. Ces activités sont effectuées sans aucun retard. Il en est de même pour les changements d'état quand une transition se produit via une temporisation ou suite au déclenchement d'un événement extérieur.

Plusieurs groupes de recherche ont proposé des extensions à SDL pour exprimer les contraintes temps réel ainsi que des délais de traitement et de communication.

Fischer et Leue dans [FL98] ont exprimé l'insuffisance de SDL pour la spécification de l'exigence de la qualité de service. Ils notent, qu'en raison de la caractéristique asynchrone du langage, les systèmes décrits en SDL peuvent répondre aux spécifications même s'ils dépassent les limites d'une valeur non spécifiées et même encore sans la contrainte de temps. Le maximum qui peut être exprimée est qu'il y ait un minimum de temps qui s'écoule entre le réglage de la temporisation et la reconnaissance de son expiration par un processus temporisé. Comme solution, les auteurs introduisent le concept de complémentarité des spécifications à travers les modèles sémantiques de la logique temporelle (Temporal Logic metric) [RU71] et SDL. À cette fin, un système global de transition d'état a été défini et sert de modèle formel commun pour l'interprétation des spécifications SDL et les formules de logique temporelle. Une présentation détaillée de cette approche est donnée par [FL98].

La conception et la vérification des propriétés temps réel des spécifications SDL a été étudiée par Bourgeois [Bou07]. L'auteur a introduit des annotations temporelles pour exprimer des hypothèses sur le système d'exécution, sur l'environnement ainsi que sur les exigences temporelles. Les annotations sont des commentaires dans le modèle SDL. Les fonctionnalités de modélisation introduites pour décrire le comportement temporel de la spécification comprend la spécification d'horloges, la durée du processus SDL, une notion de priorité des transitions, la définition des événements et le maximum de temps disponible entre deux événements, ainsi que le comportement des canaux de communication et des évenements extérieurs. Le temps de traitement est une estimation par le concepteur et n'a aucune relation avec la mise en œuvre réelle. Malheureusement, l'approche ne tient pas compte de l'influence du moteur d'exécution sur les temps et les retards d'attente. La spécification annotée SDL est analysée et cartographiée par un automate temporisé. Cet automate peut alors être analysée et vérifiée avec l'outil UPPAAL [LPY97].

Dans [dW04], une méthode de simulation s'appuyant sur l'analyse des performances des protocoles de communication et sur une combinaison SDL et UML 2.0 est présentée. SDL est utilisé comme un moyen pour spécifier le comportement du protocole. A partir des propriétés temporelles de l'environnement d'exécution et des détails d'implémentation, ses caractéristiques sont modélisées et raffinées avec l'aide de diagrammes UML. La méthodologie proposée a été validée expérimentalement.

3.2.2 Les outils de conception mixte matérielle logicielle

La conception d'un système englobe sa spécification comportementale et le développement d'une architecture système sur lequel la fonctionnalité est transposée. Déjà, dans les années 1990, un certain nombre d'outils ont été développés utilisant les spécifications SDL comme une description de haut niveau du système et comme support au processus de conception et en effectuant une exploration de différentes architectures et projections du modèle SDL sur ses dernières. Comme exemples de cette génération d'outils, on cite COSMOS [IAJ94] et Corsair [DMTS00].
COSMOS est un outil de codesign développé par les laboratoires TIMA. Un produit commercial, ArchiMate [MdCP00], en a été déduit sur la base des travaux initiaux. COSMOS utilise SDL comme un langage d'entrée. Le partitionnement matériel logiciel peut être effectué manuellement. Cela

signifie que le concepteur a le choix d'automatiser ou pas de la répartition des processus SDL sur une architecture multi processeurs [ZMDJ98].

Des accélérateurs matériels sont alors automatiquement générés et décrits en VHDL, tandis que les autres processus SDL sont convertis en programmes C et exécutés sur des processeurs génériques. En outre, la génération de composants d'interface pour connecter le matériel et les processus logiciels est prise en charge par l'outil.

Une co simulation du code VHDL et du code C obtenus est réalisée comme dernière étape afin de valider les propriétés temporelles de la conception. Les trois outils utilisés dans le flux de codesign sont illustrés par la figure 3.2. Un inconvénient de cette approche est le temps de simulation qui reste assez long qui peut atteindre plusieurs heures dans le cas de systèmes complexes tels que décrits par [ZMDJ98].

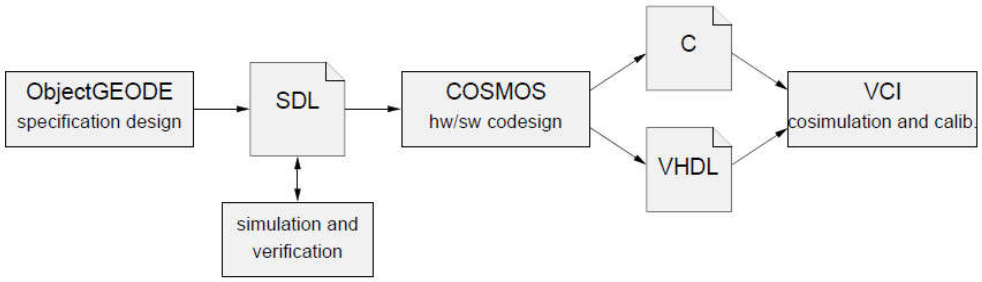

Figure 3.2 Flot de conception hardware/software de l'outil COSMOS

COSMOS ne permet pas le partitionnement avec une granularité plus fine que les processus SDL. Cela signifie que pour profiter pleinement du gain en efficacité des réalisations matérielles, le concepteur doit structurer la spécification de haut niveau avec des détails de mise en œuvre préalables. L'efficacité de la conception matérielle générée a été signalée comme pauvre [MHA+02] bien que les résultats aient été obtenus avec l'outil ArchiMate.

Les limitations de l'outil de COSMOS ont été levée par l'outil CORSAIR (Codesign and Rapid Prototyping System for Applications with Realtime Constraints), développé par l'Université d'Erlangen-Nuremberg. Comme COSMOS, c'est un outil de conception intégré qui traite des systèmes mixtes matériel/logiciel. Il réalise aussi la synthèse automatique. Toutefois, une extension du langage SDL, dénommé SDL*, a été utilisé pour la spécification du système. Cette extension intègre les spécifications des aspects temporels du système. Un exemple qui illustre une annotation dans un processus de synchronisation SDL* est illustré par la figure 3.3.

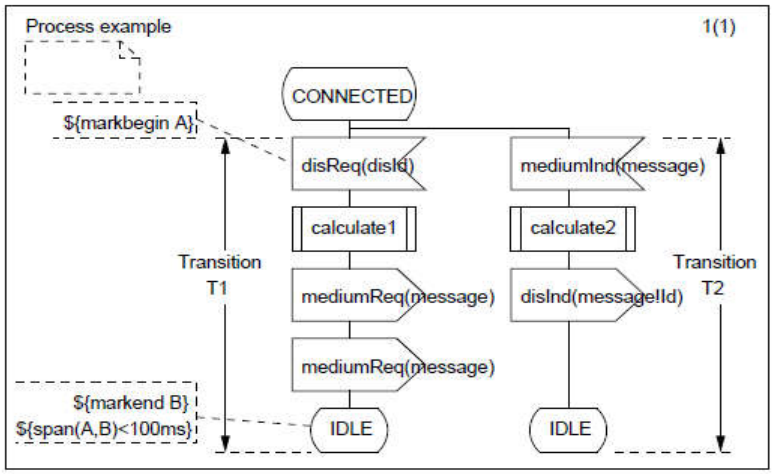

Figure 3.3 Exemple d'annotation temporelle supportée par SDL* [DMTS00]

Ces annotations de synchronisation dans la spécification sont prisent en compte par l'outil lors de la génération d'une application prototype répondant aux contraintes temps réel. L'outil CORSAIR crée d'abord un graphe de la spécification SDL* et par la suite, des cartes composées de nœuds de traitement et d'interconnexions construites à partir de composants de ses bibliothèques. Ce schéma est représenté par la figure 3.4. Les objectifs d'optimisation de ce processus de cartographie sont une utilisation raisonnable des ressources pour répondre ainsi aux contraintes de temps. La spécification d'architecture système est également exprimée en SDL*. Le flot de conception complet, y compris la spécification du système, l'architecture et la synthèse de mise en œuvre, est schématisé par la figure 3.4. La synthèse des logiciels, du matériel et des interfaces est également prise en charge par l'outil. L'objectif du processus de conception avec CORSAIR est un système prototype réalisé sur une plateforme multi processeurs interconnectée à des circuits programmables.

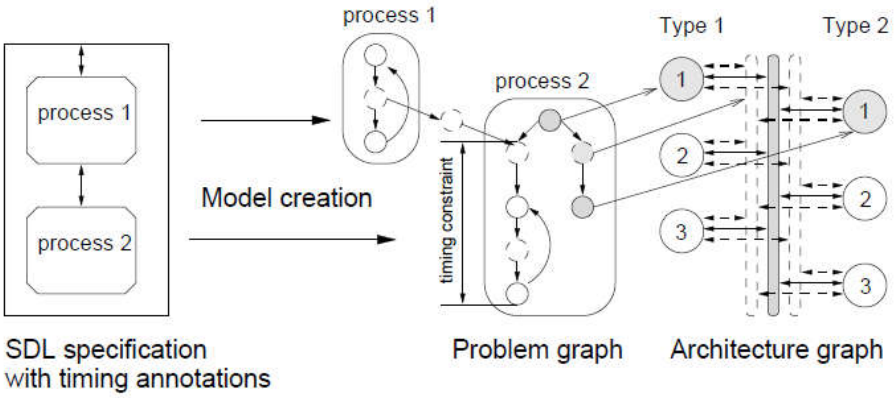

Figure 3.4 Modélisation SDL* avec prise en charge des annotations temporelles [DMTS00]

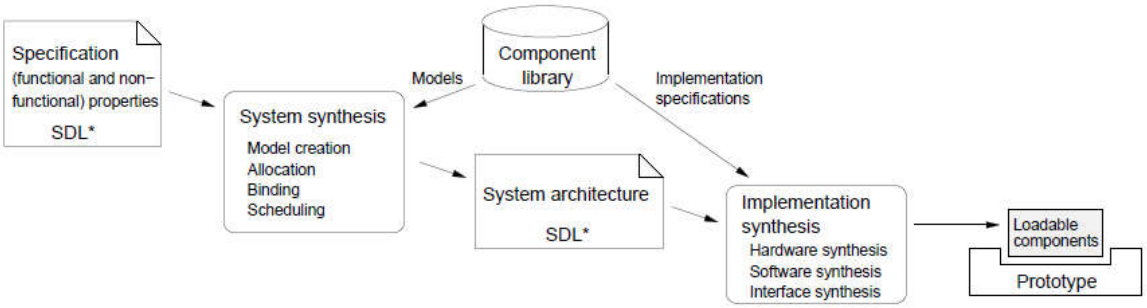

Figure 3.5 Flot de conception mis en œuvre par l'outil CORSAIR [DMTS00]

3.2.3 Synthèse d'implémentations

Nous présentons les travaux de recherche qui traite de la transformation des spécifications SDL abstraites vers des implémentations dans des environnements d'exécution donnés.
L'équivalence sémantique de la spécification et du modèle de mise en œuvre doit être garantie lors de ce processus de transformation. Cependant, le fait que les modèles abstraits SDL fonctionnent avec des files d'attente infinies et une capacité mémoire illimitée contredit la réalité physique et montre qu'il y a des limites en allant d'un modèle théorique vers une application réelle. Une extension de la sémantique de SDL vers l'utilisation de ports d'entrée limités en taille a été proposée par Gotzhein et al [GGK07]. L'étude approfondie de la problématique de la génération d'implémentations efficaces a conduit à un grand nombre de techniques d'optimisation.

La génération des implémentations logicielles était le but de la majorité des outils SDL commerciaux. A partir des années 1990, la traduction automatique vers des descriptions de matériel, à savoir vers du code VHDL, a été étudiée par un certain nombre de groupes de recherche. Outre la génération de code, l'intégration du système SDL dans un environnement d'exécution comme les systèmes d'exploitation et la conception des interfaces entre le matériel et les logiciels est prise en compte. Les outils pour la génération des implémentations matérielles prototypiques ont été présentés. Ils n'ont pas cependant trouvé le chemin du succès dans le domaine commercial.

Un modèle de mise en œuvre simple pour les spécifications SDL et qui maintient la sémantique du langage est le modèle serveur présenté au chapitre précédent. Chaque processus SDL est réalisé par un serveur asynchrone; une entité avec sa file d'attente propre de signaux. Il traite les signaux d'entrée un par un et communique avec d'autres processus en plaçant des signaux asynchrones dans leurs files d'attente d'entrée. Ce modèle peut être appliqué aux logiciels et aux implémentations matérielles similaires. Dans le premier cas, tous les services sont exécutés simultanément sous le contrôle de l'ordonnanceur du système d'exploitation, tandis que dans le dernier cas, le parallélisme matériel réel peut être exploité. Le modèle serveur est utilisé, par exemple, par le générateur de code de Telelogic TAU pour les implémentations de logiciels ou par l'outil COSMOS [ZMDJ98] dans le cas de la génération de code VHDL. Les inconvénients majeurs de cette approche sont les coûts généraux de changement de contexte ainsi que les ressources requises pour maintenir les files d'attente.

Une optimisation qui supprime la surcharge mentionnée en rapport avec le modèle serveur est le modèle de thread. Dans ce modèle, la communication entre les processus SDL est synchrone à travers les appels de procédure. Au lieu d'envoyer un message vers un autre processus, la transition correspondant au processus récepteur est appelée. L'exécution reste dans le même thread, aucun changement de contexte et de tampons de signal n'est nécessaire. La spécification SDL doit être analysée afin d'identifier toutes les chaînes possibles de transitions déclenchées par un événement extérieur et se terminent par une transition sans sortie de signal interne ou vers l'environnement. Une telle chaîne est appelée fil d'exécution.

Le modèle de thread peut être efficacement appliqué à la mise en œuvre des piles de protocoles simples où, fondamentalement, les signaux sont échangés seulement du haut vers le bas lors de la transmission, ou dans la direction opposée après la réception d'un paquet. Des modes d'interaction plus complexes dans ce modèle peut créer d'autres coûts généraux puisque les transitions, qui font partie de l'activité de plusieurs échanges sont exécutées suivants différents événements extérieurs, seront multipliées si une sérialisation est introduite [MF00]. Pour les implémentations de logiciels utilisant le modèle de thread, une attention particulière doit être prise quand il y a des interdépendances entre les processus ou les sorties de signaux multiples au sein d'une transition afin de préserver la sémantique du modèle [HMTKL96], [Kon03]. Une telle analyse de la dépendance a été présentée par Leue et Oechslin [LO96] qui ont proposé des optimisations pour améliorer le parallélisme au sein d'une application.

L'amélioration de l'efficacité a été obtenue également par la réduction du nombre d'opérations de copie de tampons volumineux tels que les unités de protocole de données, ainsi que par une technique appelée cadrage au niveau de l'application. La première technique utilise des références à des tampons. Le cadrage au niveau des applications est une technique de traitement des unités de protocole de données à travers les couches protocolaires.

Les implémentations matérielles découlant de spécifications SDL a été au centre des travaux de recherche depuis les années 1990. Les tentatives d'utiliser SDL comme un langage de description des systèmes numériques synchrones, de façon similaire à celle de SystemC ont été infructueuses en raison de l'absence de concepts pour exprimer un comportement synchrone et des types de données pour les opérations de manipulation de bits [MHA+02].

Parmi ces travaux, ont cite ceux de Muth [Mut02] (qui a également contribué au système de CORSAIR). Le flux de prototypage rapide présenté commence avec une spécification initiale en SDL avec un ensemble d'annotations de synchronisation qui expriment les contraintes temps réel du système. L'objectif est de générer un matériel compatible sur une plate-forme de prototypage rapide composée de plusieurs processeurs généralistes de haute avec des E/S configurables (CIOP). Le CIOP se compose de matériel programmable (FPGA Xilinx) et est connecté via un bus PCI avec les autres processeurs de la plateforme.

Une analyse en temps réel de la spécification SDL utilise le modèle de flux d'événements pour modéliser l'apparition possible d'événements externes et des temporisations. Les délais pour le traitement des événements peuvent être spécifiés. Dans le cadre de l'analyse en temps réel, est prise en compte également une estimation dans les pires des cas de la profondeur de la file d'attente. Ceci est important afin de préserver la sémantique du modèle SDL originel et d'allouer efficacement des ressources.

Les processus SDL sont manuellement mappés sur les unités de transformation de l'architecture cible. Le partitionnement dépend de leur chronologie et des exigences de complexité de calcul [Mut02]. Pour la partie logicielle, le générateur de code de Telelogic CAdvanced est utilisé. Ce générateur de code C crée des implémentations basées sur le modèle serveur. Un moteur d'exécution est tenu de fournir toutes les fonctions complémentaires nécessaires telles que timers, les communications inter processus et l'interface avec l'environnement qui font implicitement partie de la spécification SDL. Dans l'approche décrite, le moteur d'exécution est le système d'exploitation RTEMS (Real Time Executive for Multiprocessor Systems).

Pour la génération du code VHDL, les techniques de mise en œuvre différentes peuvent être choisies: le modèle serveur ou les fils d'exécution parallèles. Pour le modèle serveur, le processus spécifique à la machine à états finis étendus est modélisé en VHDL et complété avec des composants préconçus pour les files d'attente du signal et des ports de sortie.

Dans le cas d'une implémentation sous forme de fil d'exécution, la chaîne des transitions en provenance d'un événement initial externe est déterminée. Chaque thread est mis en œuvre comme un processus séparé VHDL. Puisque des threads multiples peuvent accéder aux mêmes variables partagées d'un processus, des verrous sont insérés afin de protéger ces variables d'accès simultanés lors d'une mise en œuvre parallèle.

Le code VHDL résultant est synthétisé et associé à un circuit FPGA Xilinx. Le processus de conception est illustré par la figure 3.6. L'éditeur de lien m4 est utilisé pour traiter les annotations temporelles.

Comme déjà mentionné précédemment, un moteur d'exécution pour les spécifications SDL doit fournir des services et des mécanismes qui font partie du modèle sémantique du langage. Parmi eux, on cite la manipulation de l'horloge du processus d'ordonnancement ainsi que les files d'attente de messages asynchrones. Pour des raisons de performance, il est également possible de transposer les services susmentionnés vers des mécanismes du système d'exploitation.

Un aspect qui est souvent négligé est le fait que les moteurs d'exécution dont les systèmes d'exploitation sont sujets à des erreurs de conception. Donc, même si le modèle SDL est formellement vérifié, son environnement d'exécution peut ne pas l'être. Cela peut compromettre la confiance dans le fonctionnement du système.

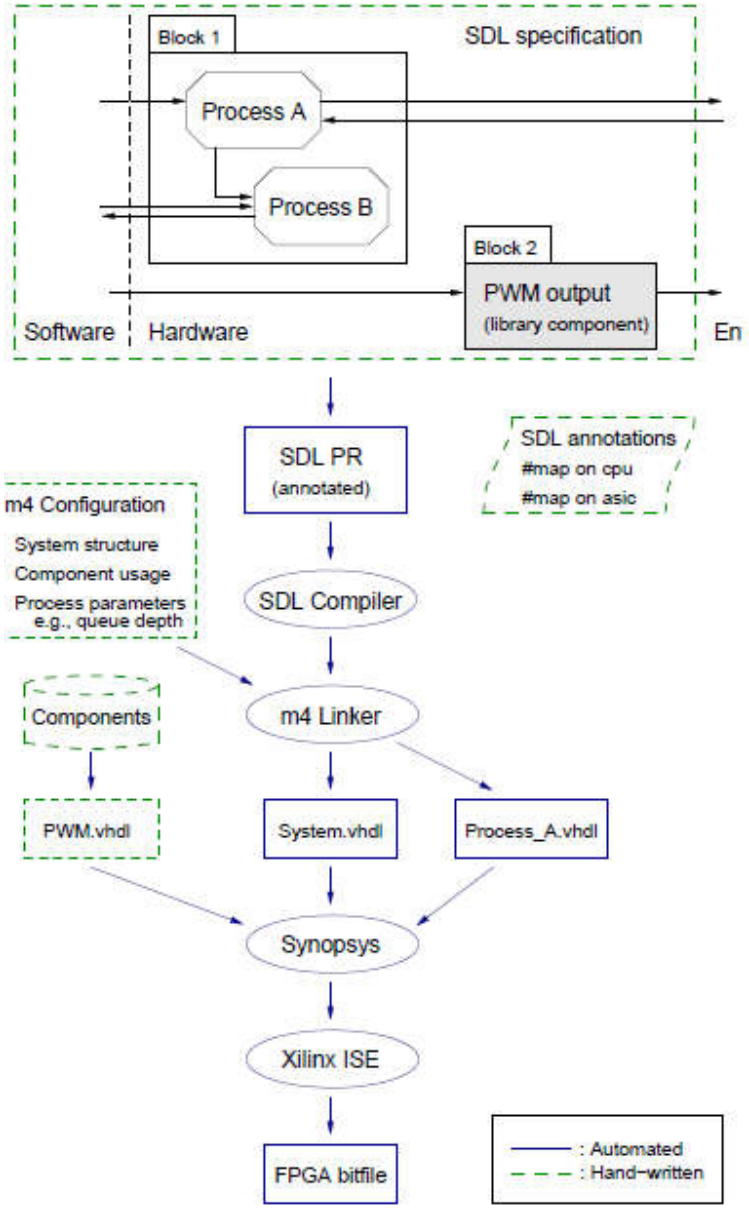

Figure 3.6 Prototypage rapide tel que décrit par [Mut02]

Une approche récente vers un système d'exploitation temps réel conçu en utilisant des méthodes formelles a été rapportée dans [VdJ07]. Ce système d'exploitation, OpenComRTOS, peut mapper des spécifications SDL sur celle des ressources cibles choisies.

Toutes les implémentations d'autres logiciels de spécifications SDL exigent une confiance sur les moteurs d'exécution, c'est-à-dire qu'ils soient soigneusement testés et que la plupart des erreurs trouvées. Pour un certain nombre de différents systèmes d'exploitation, les mappages spécifiques - basé sur la sortie du générateur de code - ont été créés. Telelogic, par exemple, offre des modèles d'intégration pour Neutrino, VxWorks, OSE Delta, et Nucleus+. Les modèles d'intégration pour les

autres plateformes sont développées d'une manière spécifique par divers groupes de recherche.

Les générateurs de code à partir de spécifications SDL sont des outils matures et des standards de l'industrie. Le code généré automatiquement à partir CAdvanced C contient de nombreuses macros destinées au préprocesseur pour faciliter une intégration d'un moteur d'exécution. Chaque environnement d'exécution doit définir ces macros telles que décrits par les concepts SDL (les signaux, par exemple, les processus et timers) et peuvent être mappées sur des ressources de l'environnement d'exécution.

Malheureusement, Telelogic ne fournit pas un cadre générique pour l'intégration des systèmes d'exploitation temps réel, mais recommande simplement d'utiliser les intégrations existantes comme exemples lors de la conception d'une intégration étroite avec un nouvel système d'exploitation.

3.3 Implémentation de protocole de communication basée sur SDL

Dans notre développement, nous présentons non seulement une méthodologie de conception des systèmes de communication pour SoC, mais nous montrons comment elle a été appliquée pour une mise en œuvre mono puce.

Des systèmes de communication développés à partir de spécifications SDL ont été rapportés par la littérature. Afin d'examiner notre approche dans le contexte des travaux antérieurs, nous nous intéresserons à l'étude de quelques conceptions comparables déjà publiées. Il est à noter que des systèmes propriétaires ont été certainement conçus selon des processus basés sur SDL mais les résultats n'ont pas été publiés.

Drosos et al décrivent dans [DZM01] le processus de conception pour le protocole MAC de la norme DECT. Ce protocole MAC a été mis en œuvre entièrement sous forme logiciel. Le développement a commencé à partir d'un modèle SDL traduit automatiquement en C par le générateur de code CAdvanced Telelogic. Afin de valider le protocole sur la plateforme matérielle cible, deux étapes ont été réalisées à savoir une intégration forte du modèle SDL avec le système d'exploitation virtuose et un modèle de simulation pour le système de débogage ARMulator.
L'approche d'intégration forte (tight integration) a été choisie du fait que le moteur d'exécution de Telelogic ne permet pas le mode de fonctionnement préemptif. L'approche choisie permet à chaque processus SDL d'être mappé sur une tâche telle que les processus de priorité supérieure peuvent préempter des processus de faible priorité. Une bibliothèque pour l'intégration de l'OS virtuose devait être développée par Telelogic. Comme il n'existe pas de modèle générique pour cibler d'autres systèmes d'exploitation que ceux fournis par la suite TAU, cette adaptation a été considérée par l'auteur comme un résultat majeur [DZM01]. Les détails sur la conception de cette bibliothèque n'ont pas été publiés.

L'environnement ARMulator permet de simuler un processeur ARM et ses périphériques. Le modèle mémoire d'ARMulator a été étendu pour intégrer un modèle de comportement de l'interface radio. Le protocole MAC implémenté sous forme logicielle interagit via un certain nombre de commandes avec l'émetteur-récepteur radio. Les événements du récepteur interrompent le processeur de déclenchement. Une tâche dédiée à l'environnement est chargée de générer des signaux SDL adaptés et de les envoyer au processus périphérique.

L'intégration sur un système cible matériel semble ne pas avoir nécessité beaucoup d'effort. Ce gain est dû à la phase validation par simulation des spécifications des interfaces matérielles et des

comportements [DZM01].
Des travaux similaires ont été menés par Marko Hannikainen [Han02]. L'auteur a traité de la conception d'un système de communication sans fil (TUTWLAN) avec une prise en charge de la QoS. Le partionnement a été réalisé sur deux circuit ; FPGA et DSP. Ce dernier a pris en charge la partie logicielle.
Dans [HKHS00], une application développée à base du langage SDL est expliquée en détail. Comme dans des développements précédents, Tau Telelogic a été utilisé pour l'édition, la simulation et la génération de code. Les spécifications complètes modélisées peuvent être considérées comme complexes dans ce sens qu'elle a mis en œuvre 24 processus, 75 procédures et un total 96 différents types de signaux internes. L'auteur rapporte que la simulation au début des phases de conception a contribué à l'accroissement de la qualité de la conception et au gain en temps et en coût de réalisation [HKHS00].

Le modèle d'intégration léger (light integration) a été choisi pour cibler une spécification SDL sur un DSP. Aucun système d'exploitation n'a été intégré. Le code C compilé consomme plus de 400ko de mémoire. Des fonctions associées à l'interaction avec l'environnement ont été ajoutées pour fournir des interfaces pour le module radio et l'ordinateur hôte. Le modèle SDL mis en œuvre était indépendant de la plateforme cible.
Un certain nombre d'optimisations pour améliorer les performances de la mise en œuvre SDL ont été proposées et leurs effets évalués. Ces optimisations comprennent l'introduction de processus de priorités, l'adressage explicite des signaux, l'utilisation de types de données efficaces en évitant les types tableau et chaîne. Les implémentations algorithmes pour les fonctions externes ont été codées en C. Les performances ont été mesurées en simulant le modèle complet SDL sur un ordinateur hôte. Il n'a pas été rapporté que la simulation portait sur le processeur cible ou qu'une analyse temps réel n'ait été effectuée.

Un petit nombre de tâches ont été identifié pour être câblées. Il semble que la décision a été prise sur la base de l'intuition du concepteur associée à sa connaissance des contraintes de temps relative au protocole. Aucune méthode de validation de ces décisions de conception n'a été signalée. Les tâches sélectionnées pour la mise en œuvre du matériel sont la synchronisation de la trame TDMA, le cryptage des données et de calcul du CRC [SHH02].

Un accélérateur matériel a été conçu pour fournit une interface pour la communication avec le DSP. L'accélérateur matériel se compose de blocs de transmission et de réception de données, un bloc de contrôle de transmission et un registre d'état avec des informations sur la trame reçue. La plateforme de démonstration pour le système WLAN avec support de la QoS est illustrée par la figure 3.7.

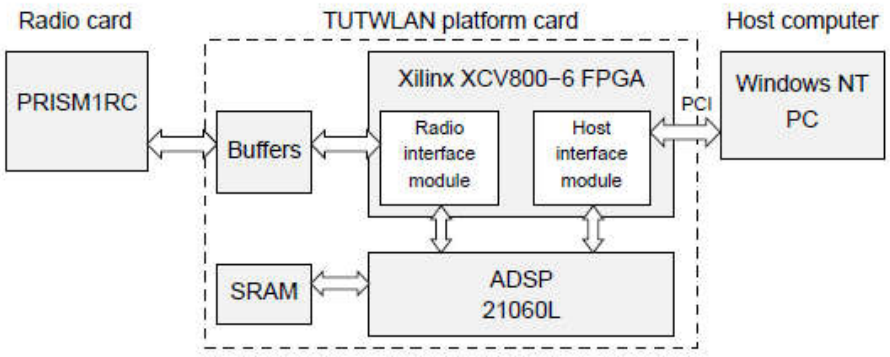

Figure 3.7 Plateforme matérielle pour la prise en charge de la QoS [SHH02]

Une mise en œuvre matérielle / logicielle du protocole IEEE 802.14.3 MAC a été présenté dans [HBB04]. Comme les deux précédentes implémentations du protocole, la méthodologie de conception utilisée par les auteurs commence par une spécification de haut niveau en SDL. Cependant, aucune transformation automatique en un matériel ou un logiciel de mise en œuvre n'est appliquée. Au lieu de cela, le système est à nouveau implémenté en SystemC. Cette nouvelle représentation est synthétisable sur du matériel et peut également être simulée. Le code SystemC obtenu a été enrichi par des sorties texte afin de retracer l'échange de signaux à la lisière de l'environnement externe.

Par le biais de l'analyse comparée des diagrammes de séquence de message tracés et ceux générés par la simulation du modèle abstrait, il a été montré que la mise en œuvre matérielle reflète correctement le modèle originel. Cette approche ne peut pas donner une preuve de fonctionnement sans erreurs à partir du moment qu'elle ne repose que sur les cas de tests préalablement sélectionnés.

Les fonctions critiques ont été identifiées pour être implémentées sous forme matérielle. On cite la prise en charge des acquittements et le décodage. Dans la perspective de l'émission d'un acquittement, l'intégrité de la trame de réception doit être vérifiée. Le décodage de cette dernière est requis pour extraire l'information des positions temporelles pour la réception ou l'émission.
Le bloc de contrôle appelé SuperFrame décode la trame beacon (balise) et maintient un temporisateur comme indicateur des opportunités de transmission ou de réception vers le bloc de contrôle principal. Ce dernier peut initier un processus de transmission ou de réception. Le traitement de la trame en cours est traitée par les blocs de coordination TX et RX. Additionnellement, une interface vers la couche physique et vers la mémoire tampon d'entrée existe.

Le partitionnement matériel / logiciel ne faisait pas partie de la méthodologie de conception présentée parce que le modèle complet a été traduit en matériel. Toutefois et chose intéressante, elle propose une nouvelle approche : les spécifications SDL peuvent être simulées et vérifiées avec des outils disponibles et ensuite traduites manuellement en un modèle SystemC. Dans un processus de raffinement progressif, des parties du modèle qui doivent être réalisées sous forme matériel sont traduites manuellement en utilisant le sous ensemble SystemC synthétisable. Cette approche de codesign bénéficierait grandement de la disponibilité de modèles de systèmes d'exploitation dans le contexte de SystemC. Cette fonctionnalité a été annoncée pour les nouvelles versions de SystemC.

3.4 Conclusion

La modélisation des spécifications des normes RFC 1034,1035 ainsi que les RFC 4033,4035 et 4035, bases des définitions du système DNS/DNSSEC n'ont pas fait l'objet de travaux utilisant le langage SDL et encore moins leurs translations dérivées vers des formes d'implémentations logiciels ou mixtes matérielles/logicielles.

Les exemples d'implémentations présentés illustrent l'intérêt qu'on peut porter au langage SDL pour permettre de passer de formes de descriptions informelles vers celles qui sont formelles. Ils illustrent l'utilisation de SDL comme outils de conception et de mise en œuvre de systèmes de communication et montrent les flots de conception typiquement employés. On y remarque que le générateur de code CAdvanced est couramment utilisé pour la synthèse des parties logicielles à partir de modèles SDL.

Toutefois, ces exemples montrent également l'absence d'un modèle général et de concepts pour la création efficace de moteurs d'exécution pour les systèmes d'exploitation autres que ceux déjà pris en charge par Tau Telelogic. Si aucun système d'exploitation n'est utilisé pour l'implémentation, le mode light est utilisé. Ce dernier implémente des processus non préemptifs. La taille du code exécutable produit est plus grande et sa performance peut être moindre par rapport aux autres modes fournis par Tau Teleogic. L'ordonnanceur est réalisé comme une boucle infinie. Si un processus SDL doit être exécuté pour traiter un signal d'entrée ou de temporisation, la fonction d'activation contenant la mise en œuvre de machine d'état est appelée. La réalisation de l'ordonnanceur comme une boucle infinie présente l'inconvénient que le processeur ne peut être mis dans un mode veille à faible consommation quand il n'y a pas de processus actif.

La synthèse automatique de matériel à partir du niveau de spécification SDL est une option que nous avons exclue de notre flux de conception en raison de la complexité accrue du matériel. La traduction automatique des machines à états finis et les types de données SDL sur du matériel est peu efficace et reste loin d'être optimale ; preuve en est le manque de succès commerciaux des produits développés. L'efficacité des compilateurs pour ces langages de haut niveau sont encore un sujet de recherche fondamentale d'aujourd'hui.

Dans notre approche, la décision de partitionnement est faite sur la base de simulations du système matériel / logiciel et de profiling. Nous ne considérons pas qu'il soit nécessaire d'utiliser des outils pour l'exploration de conception automatique car ces outils ne sont pas forcément optimisés.

Nous soutenons que le concepteur a généralement une bonne connaissance du système et ses goulots d'étranglement potentiels, et donc permet de sélectionner facilement les architectures éligibles et les partitionnements adaptés. Cependant, le concepteur a besoin d'être guidés par les résultats des simulations et d'évaluation de performance afin d'estimer l'impact sur les décisions des choix architecturaux, d'identifier les goulots d'étranglement dans la conception, et donc d'être en mesure d'améliorer le partitionnement. La synthèse du matériel à partir du modèle SDL est effectuée manuellement dans notre approche. Ceci inclut également la conception de blocs optimisés couplés à des interfaces de communication adaptées. Le processus de conception du matériel est assuré par des outils EDA (Electronic design automation).

CHAPITRE IV

DNS/DNSSEC sur SoC

Nous abordons dans le présente partie la synthèse de nos travaux d'implémentation d'un serveur autoritaire sur SoC. Après l'analyse des spécifications du système DNS/DNSSEC, après avoir posé nos motivations quand à une implémentation sur SoC de ce dernier et après avoir abordé les voies et moyens pour atteindre l'objectif assigné, nous nous proposons dans cette dernière partie de synthétiser les résultats de notre implémentation.

4.1 Préambule

Le système de nom de domaine (DNS) met en œuvre l'un des deux protocoles les plus critiques du réseau Internet [Nssc00]. L'une de ses particularités est son apparente simplicité fonctionnelle. Cette simplicité, outre qu'elle peut induire des négligences dans sa gestion et sa configuration conduisant ainsi à des situations pouvant être dramatiques, cache différentes difficultés non encore résolues à ce jour [Wes06]. Ces difficultés sont associées aussi bien à ses caractéristiques intrinsèques qu'aux implémentations utilisées. Elles dérivent d'une caractéristique essentielle liée à l'accès publique aux données qu'il gère à savoir, les adresses IP et les noms qui leurs sont associés.

Nous avons abordé dans cette présente partie une forme d'implémentation d'un serveur autoritaire devant assurer de meilleures performances, une plus grande sécurité de fonctionnement et supportant toutes les normalisations édictées par IETF (Internet Engineering Task Force) y inclus, la forme sécurisée DNSSEC (Domain name system security extensions).

4.2 Conception d'un circuit dédié à un serveur autoritaire

4.2.1 Méthodologie de développement adoptée

La figure 4.1 met en valeur notre approche de conception; elle sera nécessairement descendante car l'optimisation étant l'un des arguments à satisfaire. Les spécifications sont déduites à partir des recommandations édictées par les autorités de normalisation [Sf03]. La modélisation de ces spécifications en s'aidant d'une traduction formelle sera un atout; l'utilisation d'un langage en conséquence sera recherchée. L'analyse fine des spécifications citées mettra plus facilement en valeur aussi bien l'aspect comportemental que la forme des structures de données mises en jeu. De cette modélisation sera déduite la validation de notre système ainsi que le choix de son partitionnement. Les chemins critiques seront plus facilement mis en valeur en vue de l'optimisation de leur implémentation.
Le nombre de couches virtuelles impliquées dans le processus de résolution doit être réduit au maximum. Ce qui nous a conduit à faire le choix d'éviter le plus possible, dans le cas de ce processus, de faire appel à un système d'exploitation (figure 4.2). Les mises à jour des recommandations relatives aux protocoles à mettre en œuvre dans le processus de résolution de nom impliquent nécessairement une flexibilité d'adaptation. Ce qui peut être satisfait, dans notre cas, à travers le choix de la cible technologique et d'une architecture qui doivent allier performance, adaptabilité et sécurité.

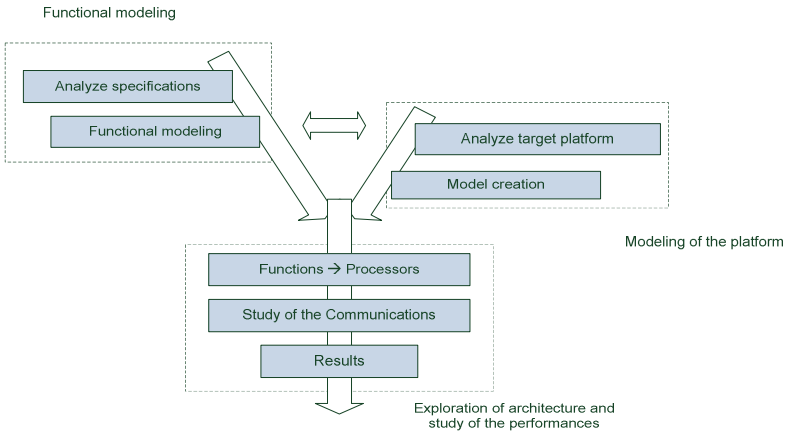

Figure 4.1 Approche de conception

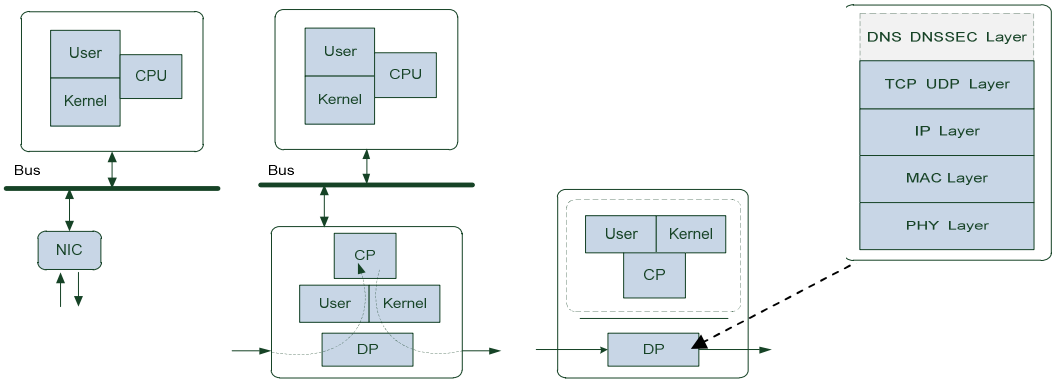

Figure 4.2 Formes d'implémentations possibles du modèle OSI

4.2.2 Choix de la cible technologique

Les applications réseaux deviennent de plus en plus complexes et variées. Le trafic associé s'est intensifié en conséquence et apparaît de plus en plus lourd. Le modèle classique des équipements réseaux mono ou multiprocesseurs généraliste devient inadapté en raison de l'inadéquation entre l'architecture des processeurs cités et la nature des données ainsi que celle des traitements que ces données doivent subir.

Une nouvelle génération de processeurs dédiés est apparue cette dernière décennie [IETF] [LEK04] [Fh05] [Zlb+06]. Leurs caractéristiques intrinsèques découlent de leur spécialisation dans le traitement de paquets dans les réseaux haut débit à des vitesses de plus en plus proches de celles de la couche physique. Leurs dénominateurs communs résident aussi bien dans leur flexibilité que dans leur performance, leur coût, leur temps de développement et leur puissance consommée.

Ces processeurs dits réseaux peuvent être classés selon trois architectures principales :
- architecture à base de processeurs RISC déployés en parallèle sur un seul circuit. Le nombre de processeur ne peut être augmenté sans une augmentation excessive de la complexité et de la taille du circuit
- architecture à base de processeurs RISC couplés à des accélérateurs matériels. Garde la même contrainte que la première architecture avec en sus une limitation quant à l'adaptabilité et à la flexibilité des accélérateurs matériels.
- processeur réseau spécifique. Il intègre plusieurs cœurs de processeurs légers et rapides taillés pour réaliser des tâches réseaux spécifiques. Ces cœurs de processeurs sont optimisés pour réaliser des traitements sur des paquets. Ils utilisent un jeu d'instructions concis pour accomplir une tâche donnée. Ils nécessitent l'équivalent d'un dixième du nombre d'instructions utilisé par un processeur RISC classique pour réaliser une tâche équivalente [Zlb+06].

La charge de traitement de ces processeurs est intrinsèquement parallèle. Les paquets voire les messages, qui constituent les entités de base des applications réseaux, sont souvent indépendants et peuvent être ainsi traités concurremment. En parallèle, les applications réseaux demandent de plus en plus une forte reprogrammabilité car sujettes à de constantes mises à jour et de mise à niveau des spécifications des protocoles mis en jeu.
On voit donc que le caractère commun est l'optimisation pour de meilleures vitesses de traitement des informations élémentaires, couplé à une flexibilité plus que nécessaire pour appliquer de nouvelles adaptations.
La figure 4.3 [XIL06] montre une nette évolution future des formes d'implémentation sur des circuits reprogrammables particulièrement le modèle Multi-Processor System On Chip (MPSoC) [Yg06] par rapport aux processeurs réseaux ASIC aussi bien en terme de performance que de flexibilité. Ces deux caractéristiques sont dans notre cas étroitement corrélées.
La technologie FPGA se présente comme une solution de choix tant elle allie de plus en plus [XIL06] performance, intégration et adaptabilité.

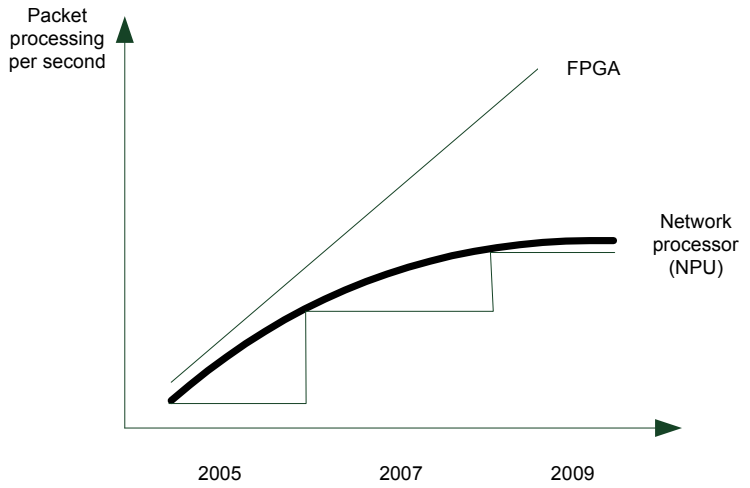

Figure 4.3 Comparatif NPU - FPGA

4.2.3 Choix du langage de modélisation

Ce choix a fait l'objet de notre développement du chapitre II. On rappelle qu'il est de plus en plus difficile d'écrire manuellement des implantations efficaces qui respectent les contraintes de l'application et qui réalisent sans ambiguïté les spécifications du protocole à implémenter. L'utilisation de techniques formelles de spécification des protocoles et l'intégration de l'étape d'implémentation dans le processus général de mise en œuvre d'un protocole sont alors indispensables.

Les langages de spécification au niveau système [EZC] [XIL06] [JN04] (ESTELLE, SDL, LOTOS, ESTEREL) offrent alors des concepts et des méthodes adaptées à de telles modélisations. Ils peuvent faire appel à des méthodes formelles pour la modélisation, la vérification et la validation du comportement des systèmes à décrire. Ils servent aussi de base pour dériver une forme d'implémentation car conçus pour offrir un niveau d'abstraction élevé sans relation à priori avec les modes de réalisation.

La caractéristique de fonctionnement asynchrone du système DNS, sécurisé ou non, nous a conduits à choisir le langage SDL. Outre qu'il assure l'approche formelle de modélisation, il est pourvu d'outils de développement et de validation matures offrant en plus une translation automatique vers d'autres langages de plus bas niveau aussi bien impératifs (C/C++) que descriptif (VHDL) [EZC].

4.2.4 Modélisation SDL et validation du modèle du serveur autoritaire

Les serveurs autoritaires se distinguent par rapport aux serveurs récursifs par le fait qu'ils sont dédiés à la gestion d'un ou plusieurs domaines donnés. On peut trouver sur un même serveur la coexistence de ces deux services. Cependant, les recommandations de sécurité préconisent leur séparation [Hu06].

L'analyse des spécifications citées au paragraphe II met en valeur trois fonctions essentielles rentrant dans le processus de résolution au niveau d'un serveur autoritaire : le traitement de l'entête du message, la résolution associée à la requête reçue, la composition et la compression éventuelle du message réponse à émettre.

La modélisation SDL que nous avons obtenue (figure 4.4) et validée nous décrit cinq blocs pouvant fonctionner en parallèle sur des messages en entrée disjoints :

Header_block : traite l'entête du message d'entrée en vue de le valider ou de le filtrer, d'extraire la ou les questions et d'aiguiller les informations valides vers le bloc résolution.

Resolve_Block : il est en liaison directe avec la base de données de ou des zones à gérer. Il n'a aucune influence sur cette dernière. La seule fonction qu'il implémente est la recherche de RR tenant compte des requêtes qu'il a reçues et de la normalisation [Moc87].

Compress_Block : son rôle est de composer la réponse à émettre. Dans le cas d'une réponse valide, la fonction compression est activée pour réduire la taille du message en sortie [Moc87].

Transfert_Block : concrétise les transactions entre serveur primaire et ses serveurs secondaires. Il peut générer une fonction d'écriture sur la base de données dans le cas des serveurs secondaires.

Data_Base_Block : Constitue l'interface nécessaire pour la gestion des données de zones. Il est à remarquer que les fonctions cryptographiques ne sont aucunement mises en œuvre lors du processus de résolution dans le cas de ce type de serveur. La figure 4.5 montre le processus de résolution associé à Resolve_Block.

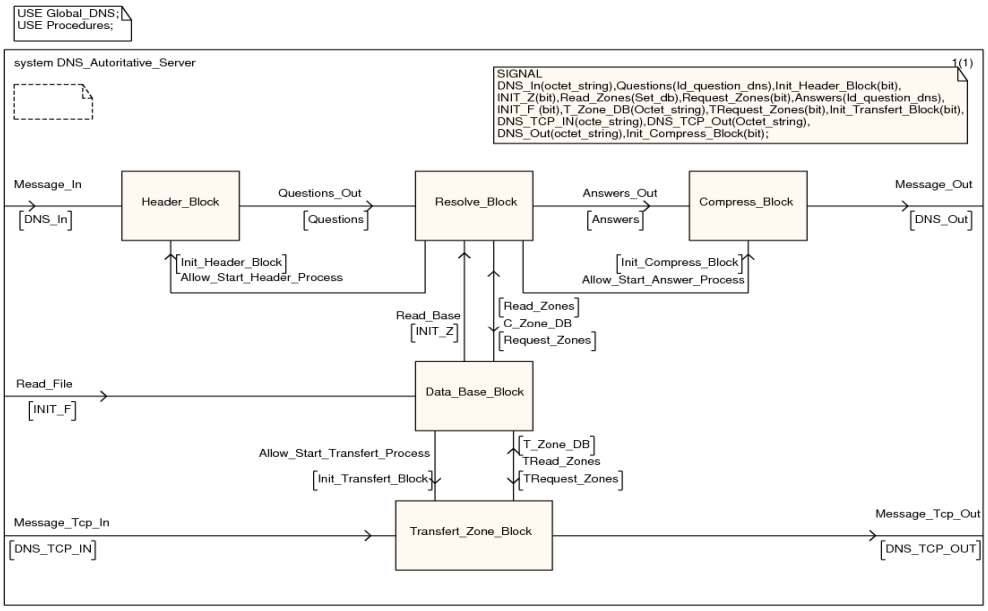

Figure 4.4 Notre modélisation SDL d'un serveur autoritaire

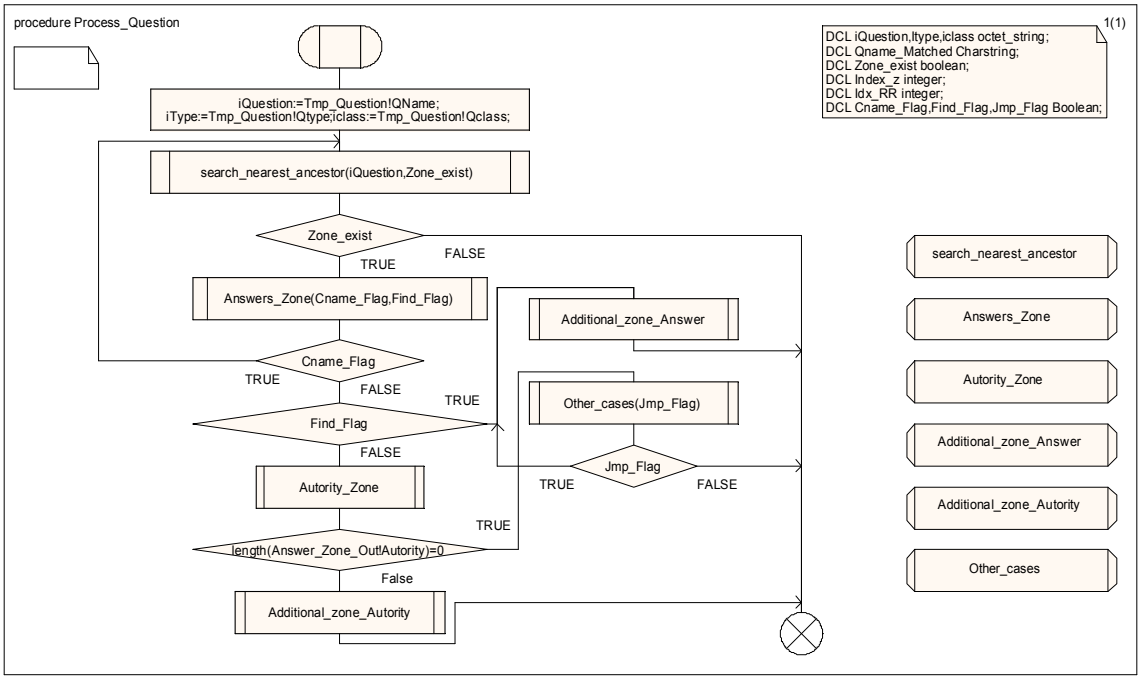

Figure 4.5 : Processus de résolution dans le cas d'un serveur autoritaire

La recherche de ressources RR constitue la fonction critique dans le processus de résolution. Elle l'est d'autant plus que la taille de la base de données est importante.

4.2.5 La recherche de RR

L'étude de divers travaux visant à augmenter les débits de traitements aussi bien pour les couches réseaux de niveau du premier au troisième niveau voire du septième niveau du modèle OSI, met en valeur le recours systématique à la fonction recherche. On cite les équipements haut débit tel que les routeurs, les sondes de détection d'intrusion, les filtres réseau et les commutateurs web par exemple [KOL05] [Khf+00]. Elles font toutes appel soit aux méthodes d'indexation par l'optimisation de l'arbre de recherche soit aux fonctions de hachage. On y décèle une similitude par rapport aux deux premières approches citées plus haut. Elles se distinguent cependant par leurs formes d'implémentation qui visent souvent la transposition des algorithmes de recherche vers des formes hardware pour atteindre des débits de plus en plus élevés. On y remarque aussi que les structures de données manipulées ainsi que la taille de la base de données conditionnent dans une grande part l'approche d'implémentation.

La recherche de préfixes dans une table nécessite le réarrangement des clefs de recherche dans des structures de données bien définies. Des algorithmes de recherche adaptés vont être déduits en conséquence. Le choix sera conditionné aussi bien par les temps d'exécution que par la taille mémoire à utiliser.
L'une des solutions hardware les plus naturelles revient à utiliser des mémoires associatives. Ces mémoires dites CAM [Dmv+01] [Dmj98] réalise l'adressage par le contenu. Le contenu, dans notre cas le préfixe à chercher, est appliqué à l'entrée de la mémoire.
A cette entrée sera déduit, s'il existe, un index vers la donnée préfixée. Cet index est souvent une adresse pointant une mémoire classique. La recherche se fait à l'identique excluant de facto la recherche par intervalle. Les TCAM offrent une réponse à ces deux contraintes [SED83] [KNU73].
Aussi bien dans le cas des CAM que des TCAM, la recherche de préfixe est optimale (O(1)). Elle est obtenue au prix d'une recherche parallèle sur l'ensemble du contenu. Ce qui génère des circuits plus complexes, plus chers, impliquant des consommations de puissance accrues [KNU73] limitant du coup les tailles maximales disponibles.
Si des tailles mémoires plus importantes sont nécessaires, une orientation vers une autre alternative sera indispensable. Les solutions découlant des techniques algorithmiques offrent alors un compromis. Elles peuvent conduire à des tailles mémoires plus grandes et des temps d'exécution des fonctions à implémenter acceptables.
Ces solutions algorithmiques sont classées selon deux types. Le premier type fait appel aux structures de données en arbre; les temps de réponse ainsi que la taille mémoire dépendent alors de l'organisation de l'arbre utilisée. Le second utilise des fonctions de hachage. Si leur performance en O(1) rappelle celle des CAM, leur utilisation implique souvent l'ajout de la résolution des collisions de données généré par leur concept [DHA06] [CC94].
Les deux types de serveurs à implémenter se différencient par les caractéristiques des bases de données à implémenter. L'une est quasi statique dont la taille augmente de plus en plus qu'on se rapproche de la racine de l'arbre de nommage. C'est le cas des serveurs autoritaires. Son contenu, globalement structuré sous forme d'arbre, n'évolue que lentement. La recherche d'un nœud père peut être indispensable (NSEC).
Le serveur récursif est caractérisé par un contenu dynamique structuré localement. Les ajouts ainsi que les mises à jour peuvent être fréquentes. Le nombre de RR manipulés et la taille de la mémoire utilisée sont conditionnés aussi bien par la taille du réseau d'hôtes qui lui font appel que

par la durée de vie des données qu'il reçoit. Pour les différentes raisons qu'on vient de citer, nous avons opté pour l'utilisation des structures de données en arbre pour le serveur autoritaire tandis que l'utilisation des fonctions de hachage sera privilégiée pour les serveurs récursifs.

4.2.6 Implémentation du module de Recherche dans le cas d'un serveur autoritaire

Nous avons implémenté la base de données locale sous forme d'arbre. Comme les fonctions de mise à jour de cette base ont une fréquence d'utilisation faible, nous avons opté pour leur implémentation sur un hôte externe et garder seulement la fonction recherche dans le but de simplifier la complexité de notre implémentation.

Diverses structures d'arbre de recherche existent [DHA06] [CC94] [Sz05] [Ms93] [Ps06]. Elles sont souvent l'amélioration de l'arbre binaire de recherche en ciblant les critères temps de recherche et taille de la mémoire utilisée. La conséquence est souvent la complexité des fonctions de mise à jour.

Parmi ces structures en arbres, nous nous sommes intéressés à la structure 256-way radix trie. Outre qu'elle constitue une représentation fidèle de l'arbre de nommage, ses principaux avantages [DHA06] résident dans des performances acceptables sans une grande complexité de mise en œuvre comparativement aux arbres équilibrés. Elle peut aussi utiliser des clefs de recherche de longueurs variables. Une amélioration possible vise la réduction du nombre de nœuds par concaténation des caractères constituant un chemin vers un nœud valide.

La figure 4.6 donne une représentation de l'arbre de nommage en utilisant la structure en arbre 256-way radix trie. Quelques constatations ont conduit au modèle de nœud donné par la figure 4.6-c.

Un nœud quelconque est identifié par :

- une entrée. Elle correspond à la chaîne de caractères portée par une branche d'un des nœuds précédents.
- plusieurs sorties. Elles peuvent être modélisées par des pointeurs vers les entrées des nœuds fils.
- des données spécifiques à un nœud. Ce dernier est similaire à un nœud de l'arbre de nommage. Il peut être la racine d'une zone ou un des constituants de cette dernière. Dans l'un ou dans l'autre cas, il représente un ensemble de données RR. Cet ensemble peut être indexé pour résider dans une zone mémoire donnée.
- un index propre permettant son identification.

Ce qui donne la structure de données simplifiée suivante.

```
Node
{
        Index_of_node          : adress
        Index_of_RR  : adress
        Edge_In                : input string
        P0,…,Pk        : childs
}
```

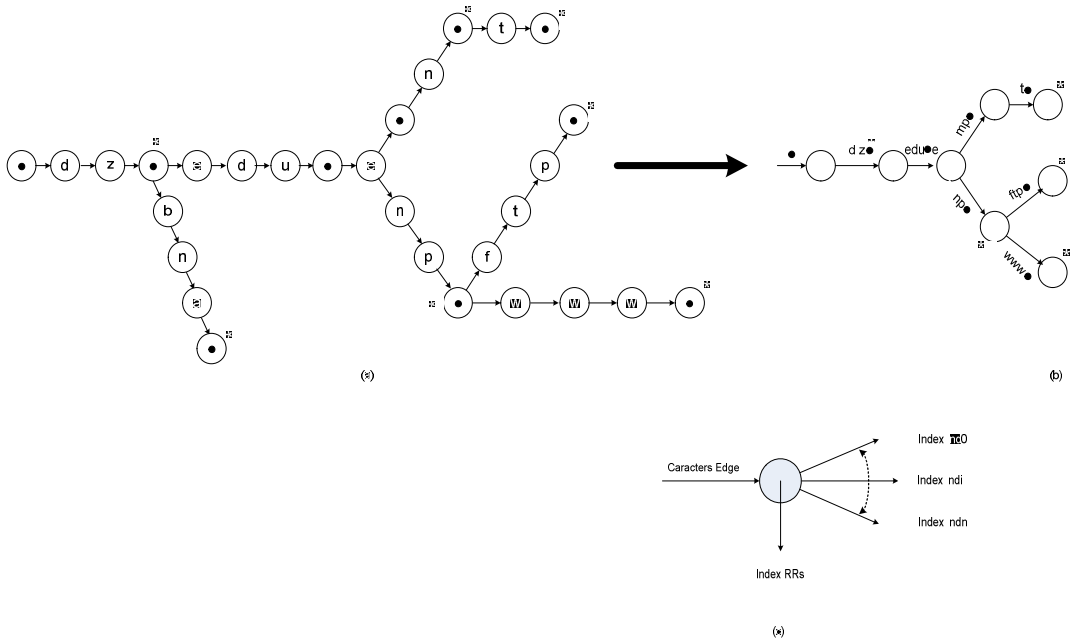

Figure 4.6 Arbre de nommage et 256-way Radix trie
(a) Arbre sans compression (b) compression du même arbre (c) modélisation d'un nœud.

Cette modélisation a conduit à l'implémentation donnée par la figure 4.8. Nous avons utilisé trois mémoires que nous avons nommées respectivement Memory_Index_node ou L0, Memory_Node ou L1 et Memoy_Edge ou L2. La première mémoire servira d'espace conservant l'ensemble des index des nœuds de l'arbre.
Elle correspondra à un empilement d'adresses pointant la seconde mémoire. La première position sera synonyme de nœud racine et prendra tout le temps la valeur zéro. La seconde mémoire servira à mémoriser les données locales d'un nœud à savoir le nombre de nœuds fils, l'adresse du premier fils, la chaîne de caractères du nœud parent et l'adresse du RR cible. Les adresses des nœuds fils sont mises dans un espace contigu. La dernière mémoire sera dédiée à la sauvegarde des chaînes de caractères identifiant l'entrée des nœuds fils. La recherche d'un nœud consistera alors à un parcours sélectif de l'arbre par lecture chaînée de ces trois mémoires. La localisation positive d'un nœud équivaudra à un parcours sans rupture par des comparaisons successives réussies de l'ensemble des caractères constituants le chemin vers la cible avec la clef de recherche ; cette dernière correspond au champ Qname de la requête DNS.
L'algorithme de parcours de l'arbre associé au schéma de principe du chemin de données du module recherche est donné ci-dessous.
Il se base sur les principaux signaux d'état à savoir Fifo_End, Node_End et Edge_End. Ils représentent respectivement l'état de la pile fifo synonyme de Qname, la fin du parcours de l'arbe et la fin du parcours d'une branche d'un nœud.
Une recherche sera positive si Node_End et Node_Mach seront tous les deux positionnés à un.

Algorithme de recherche d'un nœud

```
Node := root
Fifo_end := False
Node_End := False
Node_Match:=False
Wait for Qname
While (Node_End<>True) and ((Node_End and Node_Mach)<>True)
        Case (Fifo_End,Edge_End)
                (0,0) :  if (Read_Fifo !=   Read_Edge) then
                                        Node_Match:= False
                                        Node_end := True
                                        Break
                (0,1) :   Node := Next_Node
                          Break
                (1,0) :   Node_Match := False
                          Node_end := True
                          Break
                (1,1) :  Node_Match := True
                          Node_end := True
                          Break
        End Case
End While
Return (Node, Node_Match, Index_RR)
```

La figure 4.7 représente la machine d'état relative à l'algorithme proposé.
Nous distinguons sept états. L'état initial est atteint soit par le signal *init*, soit par le résultat de la recherche (négatif ou positif). Ces deux derniers états sont à leur tour conditionnés par quatre signaux (*ne,fe,ee et eq*) représentant respectivement les états de la fifo d'entrée, du nœud à analyser, de la chaîne de caractères portée par le nœud et du résultat de comparaison entre le contenu de la fifo et la chaîne portée par le nœud pointé. La recherche renvoyant un résultat positif est conditionnée par une fifo d'entrée vide, une égalité entre le dernier contenu de la fifo et celui de la chaîne portée par le nœud cible. Le changement de nœud courant est obtenu quand celui ci n'est pas vide (pointe des RR). Cet état est exprimé par le signal *nne* positionné à zéro. On doit tenir compte en parallèle de l'état de la fifo qui doit être non vide, de l'égalité entre les contenus à comparer et de l'état de la fin de la chaîne pointée. Deux autres états sont alors possibles ; ou bien aucun nœud ne peut être ciblé ; dans ce cas, on génère des signaux d'erreur et on doit rejoindre alors l'état initial. Dans le cas contraire, la lecture des caractères à comparer devient effective et le cycle de comparaison recommence.

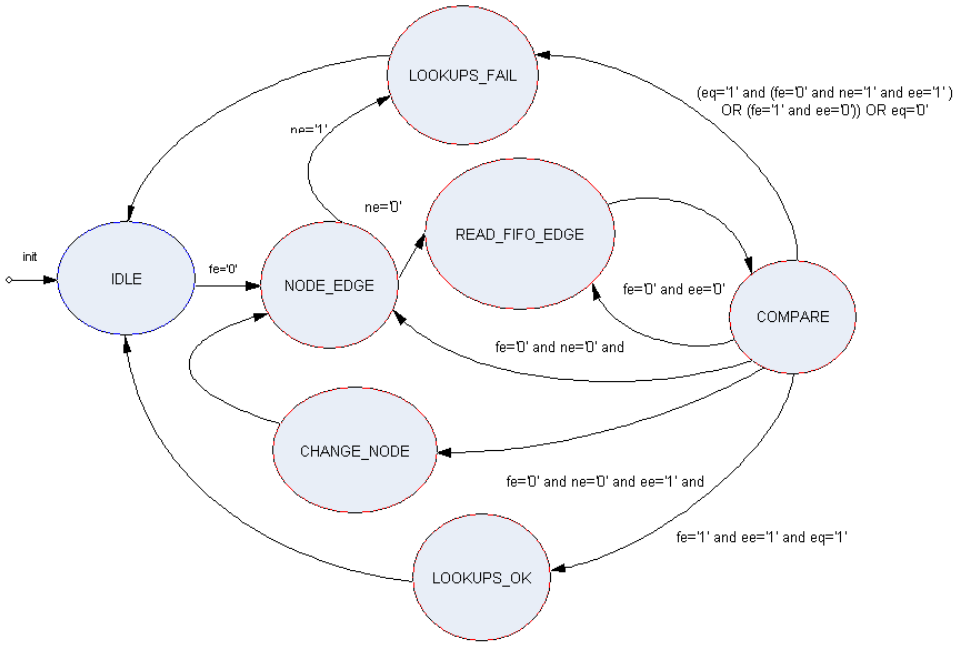

Figure 4.7 Machine d'état du module de recherche

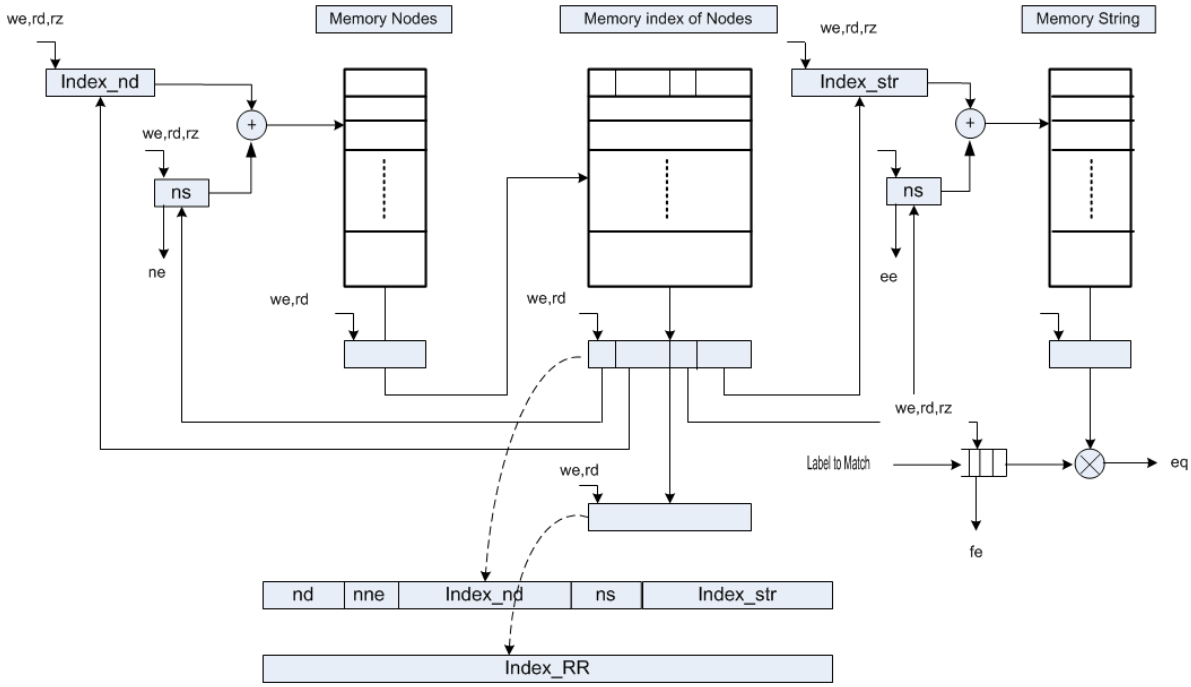

Figure 4.8 Principe du chemin de données

La figure 4.9 met en valeur le module de recherche tenant compte de la machine d'état et du chemin de données qu'elle contrôle. Le module mémoire L2 peut être interne ou externe. Si la taille de la base de données est faible, la première forme est préférée conduisant ainsi à la performance maximale. Dans le cas contraire, celle-ci sera mise à l'extérieur du circuit programmable et la performance restera meilleure par rapport à une implémentation logicielle. Le passage de l'une à l'autre forme pourra se faire par reconfiguration dynamique par exemple.

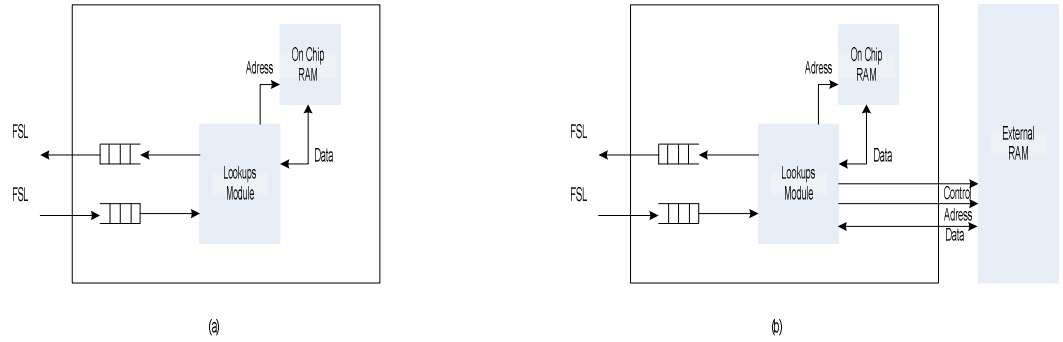

Figure 4.9 Module de recherche (a) : mémoire L2 interne (b) mémoire L2 externe

4.2.7 Architectures du serveur à implémenter

Les architectures possibles ont été obtenues par une combinaison entre la modélisation réalisée sous SDL et la technologie cible choisie, associée à son environnement de développement. Les implémentations ont tenu compte de modules accélérateurs matériels. Nous citons le module de recherche que nous venons de décrire ainsi qu'un module de vérification et de génération de code de contrôle d'erreur Cyclic Redundancy Check (CRC) nécessaire aux couches réseau et transport.

Les environnements de développement utilisés sont EDK 8.2 et ISE 8.2 de Xilinx. Les ressources nécessaires à notre implémentation ont conditionné notre choix quant à l'utilisation du circuit FPGA Virtex 4. La plateforme d'évaluation et de développement Virtex-5 ML501 de Xilinx a été utilisée à cet effet.

Le passage de la modélisation SDL vers l'implémentation a été facilité par l'environnement de modélisation TAU SDL de Telelogic à travers son outil Cmicro Targeting. Cet environnement a été utilisé pour adapter l'implémentation à la cible technologique choisie. Le tableau 4.1 donne les formes d'implémentation que nous avons envisagées.

Table 4.1 : les divers partitionnements envisagés

Implémentation	Dénomination
SDL Microblaze	1MB
SDL Microblaze + IP core Lookups	1MB_2048
MPSoC with microblaze	3MB
MPSoC with microblaze + IP core lookups	3MB_2048

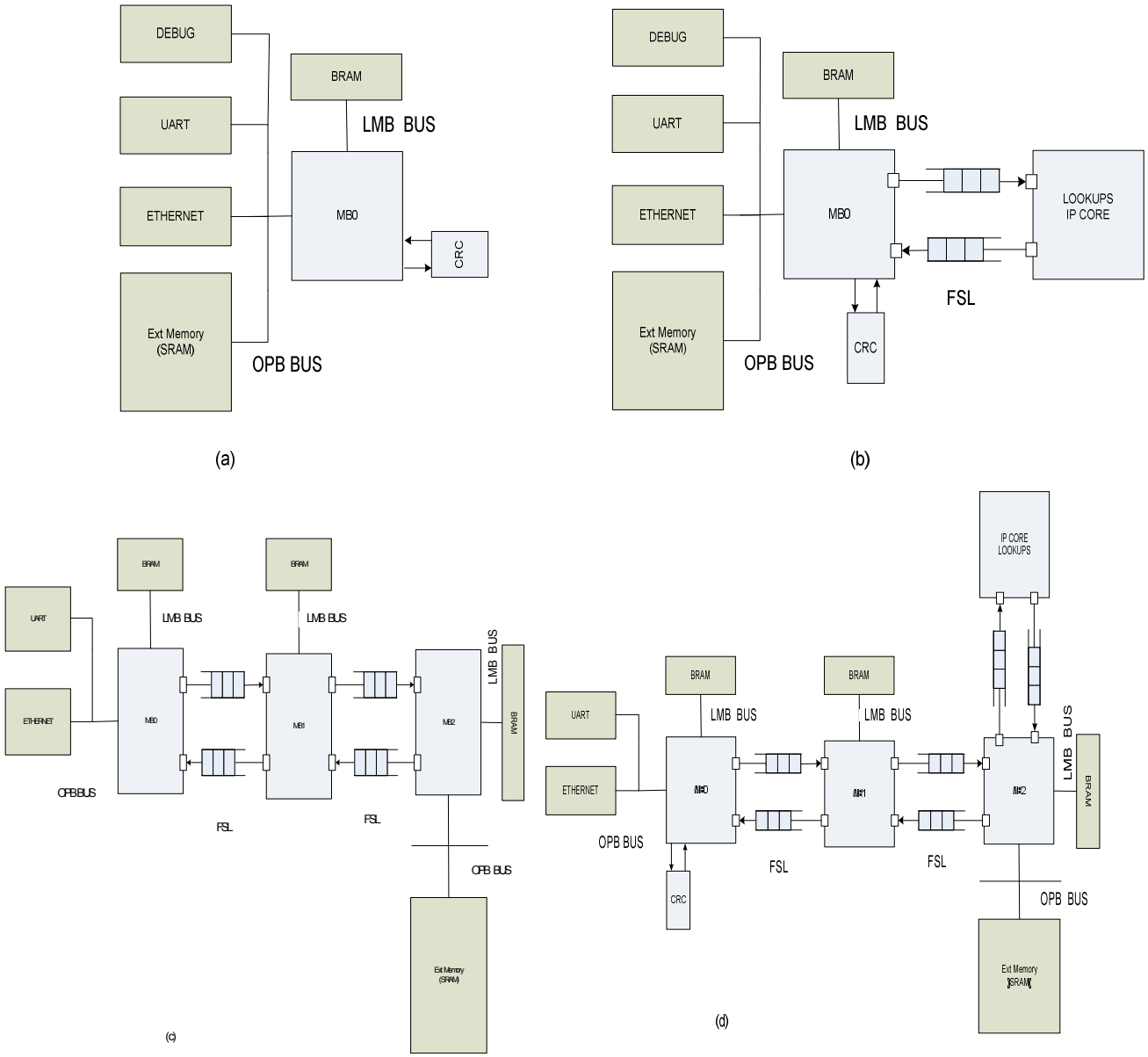

Figure 4.10 Les diverses formes de partitionnement envisagées

L'architecture de la figure 4.10-a représente le passage de la modélisation SDL vers son implémentation sur un soft processeur microblaze de Xilinx. Le seul accélérateur matériel utilisé dans ce cas est le module CRC nécessaire pour les couches réseau IP et UDP. La couche ethernet a été implémentée quant à elle à l'aide du module OPB Ethernet Media Access Controller (EMAC) v1.04a. Les figures 4.10-c et 4.10-d représentent l'adéquation des architectures matérielles que nous avons envisagées à la modélisation SDL introduite. On y distingue le même flot de traitement (traitement de l'entête, processus de résolution et construction du message réponse incluant la compression). A chaque processus cité, nous avons dédié un processeur avec son environnement spécifique. Les formes d'implémentation données par les figures 4.10-d et 4.10-b se distinguent des autres formes par l'utilisation de l'accélérateur matériel dédié à l'indexation. Le tableau 4.2 met en valeur les tailles des mémoires internes possibles en fonction du nombre de nœuds du ou des domaines à gérer si l'on décide d'utiliser exclusivement les mémoires BRAM du Virtex 4. Dans ce dernier cas, la carte que nous avons choisie pour le prototypage ne permettra au maximum que 2048 nœuds aussi bien avec un ou trois microprocesseurs microblaze; le Virtex 5LX50 ne disposant que de 216 Ko de mémoire BRAM. Avec un Virtex 5 lx330, on peut atteindre 32k nœud en gardant les mêmes performances. Pour rappel, on peut gérer des bases plus importantes en implémentant la mémoire L2 à l'extérieur du circuit FPGA au prix d'une perte relative de performance.

Table 4.2 Taille des BRAM en fonction du nombre de nœuds envisageables

Nbre Nœuds	L0 (bits)	Taille M0 (ko)	L1 (bits)	Taille M1(ko)	L2 (bits)	Taille M1(ko)	Total Mémoire BRAM estimée
1024	10	1,25	39	9,75	8	32	43
2048	11	2,75	41	20,5	8	64	87,25
4096	12	6	43	43	8	128	177
8192	13	13	45	90	8	256	359
16384	14	28	47	188	8	512	728
32768	15	60	49	392	8	1024	1476
65536	16	128	51	816	8	2048	2992

4.3 Mise en oeuvre et analyse de performances

Le tableau 4.3 montre les ressources consommées dans le cas d'un Virtex 5 LX50. Hormis les mémoires BRAM qui ont atteint les 100% d'utilisation pour l'architecture 3MB_2048, l'utilisation de ces ressources reste faible pour les quatre types d'architectures; ce qui peut permettre d'autres combinaisons pour atteindre des performances encore plus élevées.

Concernant la puissance consommée, nous l'avons évaluée en utilisant l'outil fourni par Xilinx (Virtex5_XPE_9_1.xls disponible sur le site de Xilinx). Le tableau 4.4 montre que la puissance consommée reste inférieure à 1W pour les quatre architectures pour une fréquence de fonctionnement de 116 Mhz. La distribution des puissances consommées est donnée par la figure 4.11. Pour faire une première évaluation de performance de nos implémentations, nous avons procédé à la connexion directe, à travers un câble croisé, de notre prototype à un client PC (Pentium IV 2.4 Ghz – 1024Mo -100Mb/s – Linux Fedora V) sur lequel a été compilé l'outil de mesure de performance queryperf . Un autre serveur (Pentium III 800Mhz – 512Mo -100Mb/s – Linux Redhat Entreprise Server III) a été utilisé comme témoin pour réaliser les mêmes mesures et constituer

ainsi un référentiel pour les mesures obtenues avec notre carte. Nous avons installé et configuré sur ce serveur le package bind 9 [Ps06] (serveur DNS le plus utilisé au monde). Nous avons opté pour la version la plus performante 9.4.0 associée au package OpenSSL 0.9.5a [Ps06] nécessaire aux fonctions de cryptage DNSSEC. Le serveur a été configuré pour permettre l'activation de DNS ou DNSSEC en fonction des tests envisagés.
Les tailles des bases de données utilisées ont été limitées par celles des mémoires disponibles sur la carte d'évaluation ML501 (256Mo) pour les types d'architecture 1MB et 3MB (au maximum 1 million de RR). Celles relatives aux types 1M_2048 et 3M_2048 ont été limitées par la taille des BRAM disponibles sur le Virtex 5 (tableau 4.4).
Dans les deux cas et pour le type mesure de performance que nous avons effectué, nous avons constitué une base structurée en une seule zone (un seul SOA). Dans le cas de la configuration DNSSEC, toutes les clefs ont été générées sur le serveur Linux.
La figure 4.12 montre des gains de performance obtenus par rapport à un serveur DNS BIND 9 pris comme référence. Les gains sont successivement de 42%,141%,70% et 214% pour les architectures 1MB, 3MB, 1MB_2048 et 3MB_2048 dans le cas d'un fonctionnement en mode DNS. Dans le cas d'un fonctionnement DNSSEC, les gains sont respectivement de 31%,114%,69% et 194%. On déduit que l'accélérateur matériel pour la recherche de RR et l'architecture MPSoC participent d'une manière cumulée à l'accroissement de performance du prototype obtenu.

Table 4.3 Ressources consommées par chaque implémentation

	Virtex5LX50	1MB		3MB		1MB_2048		3MB_2048	
Logic Utilization	**Available**	**Used**	**Utilization %**	**Used**	**Utilization %**	**Used**	**Utilization %**	**Used**	**Utilization %**
Number of Slice Registers	28800	3168	11	5472	19	3173	11	5467	19
Number of Slice LUTs	28800	3456	12	8928	31	3463	12	8941	31
Number of bonded IOBs	440	88	20	101	23	118	27	121	28
Number of BlockRAM/FIFO	48	18	37	26	54	40	83	48	100

Table 4.4 Distribution de la consommation de puissance pour chaque type d'architecture

Function\Type	1MB	3MB	1MB_2048	3MB_2048
Logic (W)	0,041	0,076	0,047	0,081
IO (W)	0,074	0,086	0,132	0,146
BRAM (W)	0,028	0,041	0,046	0,075
Quietescence (W)	0,463	0,464	0,465	0,466
Dynamic (W)	0,143	0,203	0,225	0,302
Total (W)	0,606	0,667	0,690	0,768

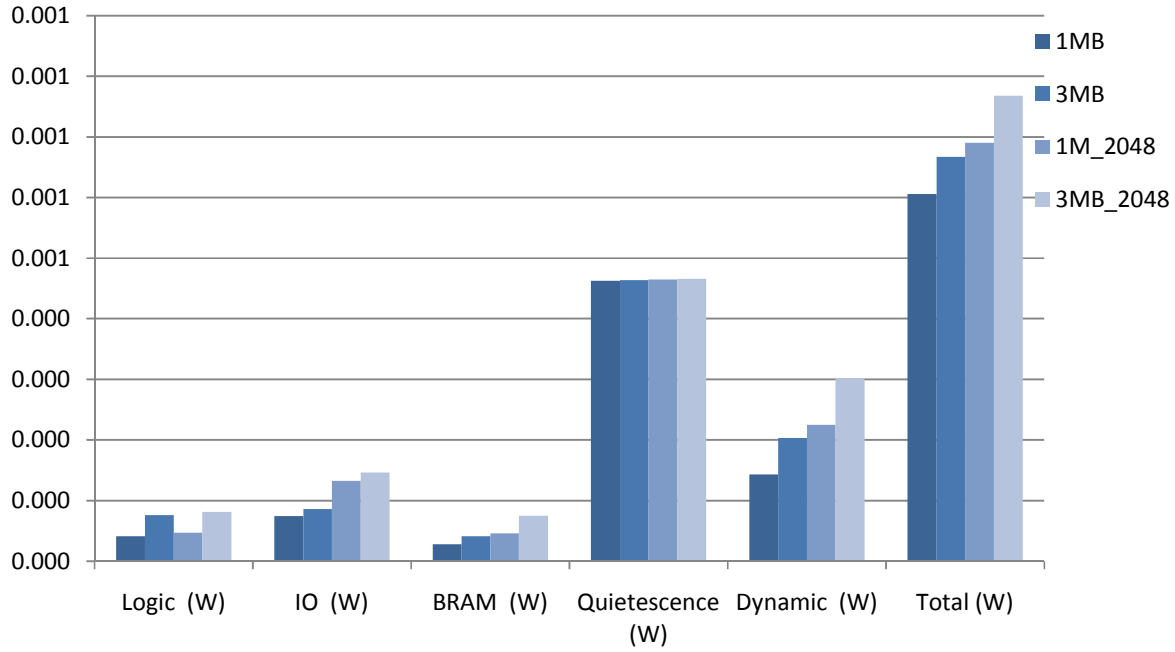

Figure 4.11 Distribution comparée de la puissance consommée par nos quatre implémentations

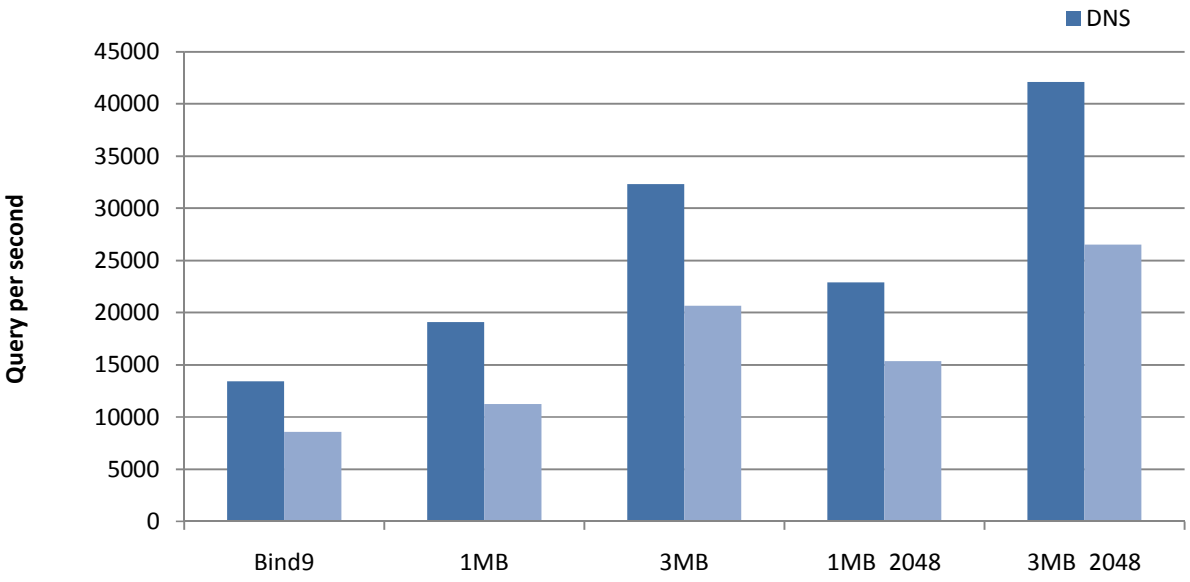

Figure 4.12 Performances comparées des quatre formes de partitionnement

4.3 Conclusion

Quatre formes d'implémentations ont été déduites et réalisées sur un circuit FPGA Virtex 4. Ces implémentations sont différenciées par les combinaisons possibles entre les architectures déduites d'une modélisation SDL et l'utilisation d'un accélérateur matériel dédié à la recherche des ressources de données spécifiques au serveur DNSSEC. Une étude comparative des performances a été réalisée en prenant comme référentiel de mesure un serveur Bind 9 fonctionnant sous Linux. Il y apparaît que la forme MPSoC associée à l'accélérateur matériel cité aboutit un gain de performance avoisinant les 200% par rapport au serveur de référence choisi. Tenant compte des hypothèses que nous avons formulées, l'implémentation obtenue allie performance et flexibilité avec une meilleure sécurité.

CONCLUSION

Comme introduit au début de notre développement, l'une des particularités du système DNS est son apparente simplicité fonctionnelle. Cette simplicité, outre qu'elle peut induire des négligences dans sa gestion et sa configuration conduisant ainsi à des situations pouvant être dramatiques, cache différentes difficultés non encore résolues à ce jour. Ces difficultés sont associées aussi bien à ses caractéristiques intrinsèques qu'aux implémentations utilisées. Elles dérivent d'une caractéristique essentielle liée à l'accès publique aux données qu'il gère à savoir, les adresses IP et les noms qui leurs sont associés.
Il est facile de tromper un serveur DNS en lui injectant de fausses informations. Souvent, c'est le serveur DNS mis à la disposition de l'utilisateur par le FAI ou par le service informatique local qui devient un DNS « menteur ». DNSSEC était une réponse visant à mieux sécuriser ce système de résolution de noms.
La révélation de la "Faille Kaminsky" en 2008 a mobilisé plusieurs acteurs entraînant une prise de conscience sur la nécessité de déployer le protocole DNSSEC. La vulnérabilité était susceptible de faire tomber le Web en menant une attaque par empoisonnement de cache. De nombreux registrer (ou bureaux d'enregistrement) ont décidé d'accélérer les travaux déjà en cours sur la mise en œuvre de DNSSEC. C'est ainsi que la racine du DNS a été signée entièrement en juillet 2010.
Si pour un PC moderne, les calculs cryptographiques nécessaires ne représentent qu'une tâche bien légère, la généralisation de cette extension de sécurité induirait une charge accrue non supportable aisément par les serveurs DNS autoritaires. En effet, comme il n'y aurait plus de cache partagé (rôle que jouent aujourd'hui les serveurs résolveurs des FAI, qui gardent en mémoire la réponse aux questions déjà posées), les serveurs des différentes zones DNS recevraient bien plus de requêtes. Ce qui engendrerait des latences beaucoup plus importantes au niveau des serveurs autoritaires déployés à cet effet.
Comme on l'a exprimé au début de notre développement, nous nous sommes inscrits dans cette problématique pour proposer une forme d'implémentation pouvant répondre, en autre, à ce coût dû à cette extension de sécurité. Ce qui nous a conduits à formuler nos motivations quant à la nécessité d'implémenter un serveur autoritaire sur Soc.
L'étude préalable des spécifications informelles de ce système ainsi que ses extensions de sécurité était un préalable. La problématique qui en a découlée a résidée dans le comment traduire ces spécifications informelles en un en système devant répondre aux postulats que nous nous sommes imposés.
C'est ainsi que les approches méthodologiques dans le contexte mixte développement de protocoles et développement de circuits ont été passées en revue. C'est à travers cette étude et à travers la spécificité du système DNS (sous sa forme de base ou celle sécurisée) que le langage SDL a été choisi aboutissant à la traduction de spécifications informelles en des spécifications formelles. Un flot de conception en a découlé et a été appliqué facilitant la recherche d'une architecture adaptée.
Le partitionnement matériel/logiciel a été fait manuellement et a tenu compte de notre expérience tout en exploitant le profiling et l'évaluation de performance comparée à travers des outils adaptés. C'est ainsi qu'une que la forme MPSoC couplée à des accélérateurs matériels a conduit à de meilleures performances. Bind9, implémentation la plus répondue dans le monde du service DNS, a été prise comme référence.
A notre avis, notre premier résultat majeur réside dans la justification des formulations que nous avons annoncées quant à l'intérêt de s'orienter dans ce genre de système (les services réseaux critiques) vers des formes d'implémentation sur circuit conduisant à des performances accrues par rapport à des formes généralistes. On peut rappeler comme repère le rendement comparé entre l'ENIAC et le plus banal des ordinateurs d'aujourd'hui qui prouve que la miniaturisation est un

voie inéluctable en raison des évolution technologiques en cours et futures. L'autre repère le plus expressif et bien adapté à notre cas est le routeur d'aujourd'hui qui naguère était un service pris en charge par l'ordinateur central d'un centre de calcul.
L'autre résultat majeur de notre développement est la justification de notre proposition de migrer les services réseaux les plus sensibles vers des formes d'implémentation que nous avons proposées. La méthodologie que nous avons adoptée et les architectures que nous avons proposées peuvent être appliquées à cet effet d'une manière aisée.
Les perspectives qui peuvent être offertes par notre étude peuvent être nombreuses. Nous pouvons par exemple citer l'optimisation de la performance de la pile protocolaire à tous les niveaux en partant de la couche liaison. La méthodologie telle que nous l'avions présentée peut être appliquée. Le caractère asynchrone du service DNS offre l'opportunité de la mise en œuvre d'un parallélisme aisé. Des modules tels que GMAC ou une peuvent être intégrés pour atteindre des performances se chiffrant en gigabits. Nous pouvons envisager dans ce sens l'étude de l'aspect adaptatif de l'architecture proposée à travers la mise en œuvre de la reconfigurabilité en vue de d'optimisation du rapport consommation/performance.

BIBLIOGRAPHIE

[AA04] Atkins (D.) et Austein (R.). Threat Analysis of the Domain Name System. RFC 3833, août 2004.

[AAL 05b] A.rends (R.), Austein (R.), Larson (M.), Massey (D.) et Rose (S.). - Protocol Modifications for the DNS Security Extensions. - RFC 4035, mars 2004.

[AAL+05a] Arends (R.), Austein (R.), Larson (M.), Massey (D.) et Rose (S.). DNS Security Introduction and Requirements. RFC 4033, mars 2004.

[Aal+05a] Arends (R.), Austein (R.), Larson (M.), Massey (D.), and Rose (S.). DNS Security Introduction and Requirements. RFC 4033, Mars 2005

[Aal+05b] Arends (R.), Austein (R.), Larson (M.), Massey (D.), and Rose (S.). Resource Records for the DNS Security Extensions. RFC 4034, Mars 2005

[Aal+05c] Arends (R.), Austein (R.), Larson (M.), Massey (D.), and Rose (S.). Protocol Modifications for the DNS Security Extensions. RFC 4035, Mars 2005

[AAL05c] A.rends (R.), Austein (R.), Larson (M.), Massey (D.) et Rose (S.). – Resource Records for the DNS Security Extensions. - RFC 4034, mars 2004.

[AL02] Albitz (P.) et Liu (C.). DNS and BIND. Sebastopol,Californie,O'Reilly Associates, Inc., janvier 2002, 4ème édition.

[Be195] Bellovin (S. M.). - Using the Domain Name System for System Break-Ins. In: Proceedings of the 5th Usenix UNIX Security Symposium, pp. 199-208. - Salt Lake City, Utah, juin 1994.

[BeN01] Brownlee (N.), Claffy (K.) et Nemeth (E.). DNS Measurements at a Root Server. In: IEEE GLOBECOM 2001. - San Antonio, Texas, novembre 2001.

[CC94] Clarke (N.),Cantoni (A.), “Implementation of dynamic look-up tables”, IEEE Proc.-Comput. Digit. Tech., vol. 141, No. 6, Nov. 1994, pp. 391-397.

[Cha03] Chan (B.). - Identity-Based PKI for DNSSEC. Thèse de master, Royal Holloway University of London, 2003.

[CK01] Cohen (E.) et Kaplan (H.). Proactive Caehing of DNS Records: Addressing a Performance Bottleneck. In: Symposium on Applications and the Internet. - San Diego, Californie, janvier 2001.

[Coo99] Cooper (D. A.). - A model of Certificate Revocation. In: 15th Annual Computer Security Applications Conference. - décembre 1999.

[Cr06] Chandramouli (R.), Rose (S.), Secure Domain Name System (DNS) Deployment Guide. Special Publication 800-81. Mai 2006

[CYC03] Chu (J.), Yu (G.), Chong (E.). Smart gateway systems for Internet security for broadband communication networks: SoC solutions and FPGA demonstrations. ASIC, 2003. Proceedings. 5th International. Conference On Vol. 2, pp 1317-1320, 2003.

[Cyc03] Chu (J.), Yu (G.), Chong (E.). Smart gateway systems for Internet security for broadband communication networks: SoC solutions and FPGA demonstrations. ASIC, 2003. Proceedings. 5th International Conference On. Vol. 2, pp 1317-1320, 2003

[DH76] Diffie (W.) et Hellman (M. E.). - New Directions in Cryptography. IEEE Transactions of Information Theory, vol. 22, n6, novembre 1976, pp. 644- 654.

[DHA06] Dharmapurikar (S.). Algorithms And Architectures For Network Search Processors. Ph.D. thesis, Department Of Computer Science And Engineering, Washington University, August 2006.

[Dif88] Diffie (W.). - The First Ten Years of Public-Key Cryptography. Proceedings of the IEEE, vol. 76, n5, mai 1988, pp. 560-577.

[Dmj98] DAVEAU (J.), MARCHIORO (G.), JERRAYA (A.), Hardware/Software Codesign of an ATM Network Interface Card: A Case Study, CODES/CASHE'98, Seattle, WA, USA, 15-18 March 1998

[Dmv+01] DAVEAU (J.), MARCHIORO (G.), VALDERRAMA (C.), JERRAYA (A.), VHDL Generation from SDL Specification, Republished in Readings in Hardware/Software Co-Design, pp. 125-134, Ed. by G. De Micheli, R. Ernst, and W. Wolf, Morgan Kaufmann Publishers, 2001.

[Dro97] Droms (R.). Dynamic Host Configuration Protocol. RFC 2131, mars 1997.

[Eas00a] Eastlake (D.). - DNS Request and Transaction Signatures (SIG(0)s). RFC 2931, septembre 2000.

[Eas00b] Eastlake (D.). - Secret Key Establishment for DNS (TKEY RR). RFC 2930, septembre 2000

[Eas99] Eastlake (D.). - Domain Name System Security Extensions. - RFC 2535, mars 1999.

[EZC] Network Processor Designs for Next-Generation Networking Equipment http://www.ezchip.com

[Fed95] Federal Information Processing Standards (FIPS). Secure Hasts Standards. Publication n180-1, U.S. DoC and NIST, avril 1994.

[Fh05] Fu (J.), Hagsand (O.),Designing and Evaluating Network Processor Applications, in Proc. of the IEEE Workshop on High Performance Switching and Routing (HPSR), Hong-Kong, May, 2005"

[FL98] Fox (B.) et LaMacchia (B.). - Certificate Revocation: Mechanics and Meaning. In : International Conference on Financial Cryptography (FC). - février 1998.

[FS87] Fiat (A.) et Shamir (A.). - How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In : Advances in Cryptology CRYPTO 86. pp. 186-194. Springer, août 1987.

[GCF05] Guette (G.), Cousin (B.) et Fort (D.). - Le roulement des clés de confiance dans les résolveurs DNSSEC. In: 4ème rencontre francophone sur la Sécurité et Architecture Réseaux (SAR), pp. 3-12. - juin 2004.

[GCF05a] Guette (G.), Cousin (B.) et Fort (D.). - Algorithm for DNSSEC Trusted Key Rollover. In: The International Conference on Information Networking (ICOIN). pp. 679-688. Springer, janvier 2005

[GCF05b] Guette (G.), Cousin (B.) et Fort (D.). GDS Resource Record: Generalization of the Delegation Signer Model. In: 4th International Conference on Networking (W N). pp. 844-851. Springer, avril 2004.

[GCO3a] Guette (G.) et Courtay (O.). - KRO: A Key RollOver Algorithm for DNSSEC. In International Conference on Information and Communication Technology (ICICT), pp. 217-228. - novembre 2003.

[GCO3b] Guette (G.) et Cousin (B.). - Les faiblesses du DNS. In : sème rencontre francophone sur la Sécurité et Architecture Réseaux (SAR), pp. 235-244. - juillet 2003.

[GCO5] Guette (G.) et Courtay (O.). Requirements for Automated Key Rollover in DNSSEC. - Draft IETF, travail en cours, janvier 2004.

[GEU06] (G.) Guette. Les extensions de sécurité DNS (DNSSEC). In Techniques de l'ingénieur. TE 7553. November 2006

[GFC04] Guette (G.), Fort (D.) et Cousin (B.). - Generalization of Delegation Signer Resource Record (DS RR) use. - Draft IETF, travail en cours, août 2004.

[Gie01] Gieben (R.). - Chain of Trust. - Thèse de master, NLnet Labs, 2001.
[GMR88] Goldwasser (S.), Micali (S.) et Rivest (R. L.). - A Digital Signature Scheme Secure Against Adaptative Chosen Message Attacks. SIAM Journal of Computing, vol. 17, n2, avril 1988, pp. 281-308.
[Gun03] Gundmundsson (O.). Delegation Signer Resource Record. - RFC 3658, déc-03
[GVE00] Gulbrandsen (A.), Vixie (P.) et Esibov (L.). - A DNS RR for specifying the location of services (DNS SRV). RFC 2782, février 2000.
[Hu06] Hubert (B.). PowerDNS Update. In RIPE-53 Meeting, Oct 2006
[IDs04] IDsA. - Infrastructure DNSSEC et ses Applications. - http://www.idsa.prd.fr, 2004.
[IETF] The Internet Engineering Task Force http://www.ietf.org
[IKM04] Ihren (J.), Kolkman (O.) et Manning (B.). - An In-Band Rollover Algorithm and a Out-Of-Band Priming Method for DNS Trust Anchors. - Draft IETF, travail en cours, juillet 2004.
[INFO00] DNS/DHCP: Why Users Prefer Appliances http://www.infoblox.com/library/whitepapers.cfm
[ISC] ISC http://www.isc.org
[ISC04] ISC. - Berkeley Internet Naming Daemon. http://www.isc.org, 2004.
[JN04] JERRAYA (A.), NICOLESCU (G.), La spécification et la validation des systèmes monopuces, HERMES Science Publications, 2004.
[JSBM01] Jung (J.), Sit (E.), Balakrishan (H.) et Morris (R.). DNS Performance and th Effectiveness of Caching. In : ACM SIGCOMM Internet Measurement Workshop '01. - San Francisco, Californie, novembre 2001.
[KBC97] Krawczyk (H.), Bellare (M.) et Canetti (R.). HMAC: Keyed-Hashing for Message Authentication. RFC 2104, février 1997.
[KGO4] Kolkman (O.) et Gieben (R.). DNSSEC operational practices. Draft IETF, travail en cours, avril 2004.
[KH02] Kayssi (A.), Harik (L.). FPGA-Based Load Balancer for Internet Servers. In Proc.14th International. Conference on Microelectronics (ICM 2002), pp. 190 - 193, Beirut, Lebanon, December 2002.
[KH02] Kayssi (A.), Harik (L.). FPGA-Based Load Balancer for Internet Servers. In Proc.14th International Conference on Microelectronics (ICM 2002), pp. 190 - 193, Beirut, Lebanon, December 2002
[Khf+00] Kayssi (A.), Harik (L.), Ferzli (R.), Fawaz (M.). FPGA-based Internet protocol firewall chip. Electronics, Circuits and Systems, 2000. ICECS 2000. Volume 1, 17-20. pp 316 – 319. Dec. 2000
[KHF00+] Kayssi (A.), Harik (L.), Ferzli (R.), Fawaz (M.). FPGA-based Internet protocol. firewall chip. Electronics, Circuits and Systems, 2000. ICECS 2000. Volume1, 17-20. pp 316 – 319. Dec. 2000.
[KNU73] Knuth (D.) – The art of computer programming. Volume 3, Sorting and searching. Computer Science and Information Processing, Addison-Wesley, 1973.
[Kol05] Kolkman (O.). - Measuring the resource requirements of DNSSEC. RIPE NCC / NLnet Labs.Sept 2005
[KOL05] Kolkman (O.). Measuring the resource requirements of DNSSEC. RIPE NCC / NLnet Labs.Sept 2005
[KON87] Kohonen (T.), Content-Addressable Memories, 2nd ed. New Jersey:

Springer-Verlag, 1987.

[KSL04] Kolkman (O.), Schlyter (J.) et Lewis (E.). Domain Name System KEY(DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag. RFC 3757, avril 2004.

[LEK04] Lekkas (P.), Network Processors—Architectures, Protocols, and Platforms. McGraw Hill, 2004.

[LM04] Laurent-Maknavicius (M.). – Les annuaires LDAP et DNSSEC au service d'une PKI globale. In: Sème rencontre francophone sur Sécurité et Architecture Réseaux (SAR). - Juin 2004.

[LMMMOO] Lioy (A.), Maino (F.), Marian (M.) et Mazzocchi (D.). DNS Security. In: Terena Networking Conference. - mai 2000

[LS.A.05] Laurie (B.), Sisson (G.) et Arends (R.). DNSSEC Hash Authenticated Denial of Existence. - Draft IETF, travail en cours, février 2004.

[Mil85] Mills Network Time Protocol (NTP). RFC 958, septembre 1984.

[Moc83a] Mockapetris (P.). - Domain Names Concept and Facilities. RFC 882, novembre 1983.

[Moc83b] Mockapetris (P.).Domain Names - Implementation and Specification.RFC 883, novembre 1983.

[Moc83c] Mockapetris (P.).Domain Names - Concept and Facilities. RFC 1034, novembre 1987.

[Moc83d] Mockapetris (P.).Domain Naines - Implementation and Specification.RFC 1035, novembre 1987.

[Moc87] Mockapetris (P.). Domain names - Concept and facilities. Implementation and Specification RFC1034, RFC 1035 Nov. 1987.

[MR02] Massey (D.) et Rose (S.). Limiting the Scope of the KEY Resource Record (RR). RFC 3445, décembre 2002.

[Ms93] MERRETT (T.), SHANG (H.). Trie Methods for representing text.Proceedings of Fourth international Conference,FoDo'93,LNCS 730.Chicago: Spriner-Verlag,pp 130-145, 1993.

[MVV96] Menezes (A.), Van Oorschot (P.) et Vanstone (S.). Handbook of Applied Cryptography. CRC Press, 1996.

[Nar02] Naraine (R.). - Massive DDoS Attack Hit DNS Root Servers, octobre 2002.

[NN98] Naor (M.) et Nissim (K.). Certificate Re-vocation and Certificate Update. In: 7th USENIX Security Symposium. - janvier 1998.

[Nssc00] The National Strategy to Secure Cyberspace. http://www.whitehouse.gov/pcipb

[ORC06] 2nd DNS-OARC Workshop http://public.oarci.net/oarc/workshop-2006

[Pel04] Pels (M.). - DNSSEC Validator. - Thèse de master, Hogenschool van Amsterdam, juin 2004.

[Pfi96] Pfitzmann (B.). - Digital Signature &liernes General Framework and Failstop Signatures. - Springer, août 1996, LNCS, volume 1100.

[Pfl03] Pfleger - DNSSEC Resolver Algorithm. - Thèse de master, Fachhochschulen, Autriche, avril 2003.

[Ps02] Panigrahy (R.), Sharma (S.). Sorting and searching using ternary CAMs - in Symposium on High Performance Interconnects, pp. 101–106, 2002.

[Ps06] Pagiamtzis (K.), Sheikholeslami (A.). Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. IEEE Journal of Solid-State

Circuits, vol. 41, no. 3, pp. 712–727, March 2006
[SANS06] Top-20 Internet Security Attack Targets (2006 Annual Update) http://www.sans.org/top20/2006
[Sch04] Schlyter (J.). - DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format. - RFC 3845, août 2004.
[Sch93] Schuba (C. L.). - Addressing Weaknesses in the Domain Name System. - Department of Computer Sciences, Thèse de master, Purdue University, août 1993.
[Sch961] Schneier (B.). - Applied cryptography. - New York, New York, John Wiley&Sons, Inc., 1996, seconde édition.
[SED83] Sedgewick (R.). Algorithms. Addison-Wesley. 1983.
[SF03] Sato (T.),Fukase (M.). Reconfigurable Hardware Implementation of Host-Based IDS. The 9th Asia-Pacific. Conference on Communication, Vol. 2, pp. 849-853, Penang, Malaysia, Sept. 2003
[Sf03] Sato (T.), Fukase (M.). Reconfigurable Hardware Implementation of Host-Based IDS. The 9th Asia-Pacific Conference on Communication, Vol. 2, pp. 849-853, Penang, Malaysia, Sept. 2003
[Sit00] Sit (E.). - A Study of Caching in the Internet Domain Name System. - Department of Electrical Engineering and Computer Sciences, Thèse de master, Massachusetts Institute of Technology, mai 2000.
[StJ03] StJohns (M.). - Using DNSSEC-secured NOTIFY to Trigger Parent Zone Updating. - Draft IETF, travail en cours, juin 2003.
[StJ04] StJohns (M.). - Automated Updates of DNSSEC Trust Anchors. - Draft IETF, travail en cours, mars 2004.
[Sz05] Sun (X.), Zhao (Y.), An on-chip IP address lookup algorithm. IEEE Transactions on Computers, 54(7), pp. 873-884. 2004.
[Tan06] Tanenbaum (A.). - Structured Computer Organization, 5th Edition, Prentice Hall, 2006
[TAN06] Tanenbaum (A.), Structured Computer Organization, 5th Edition, Prentice Hall, 2006
[Thu01] Thurrott (P.). - Microsoft Suffers Another DoS Attack, janvier 2001.
[VGEW00] Vixie (P.), Gudmundsson (O.), Eastlake (D.) et Wellington (B.). - Secret Key Transaction Authentication for DNS (TSIG). RFC 2845, mai 2000.
[Vix96] Vixie (P.). - A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY). RFC 1996, août 1996.
[Vix99] Vixie (P.). - Extension Mechanisms for DNS (EDNSO). RFC 2671, août 1999.
[VTRB97] Vixie (P.), Thomson (S.), Rekhter (Y.) et Bound (J.). Dynamic Updates in the Domain Name System (DNS UPDATE). - RFC 2136, avril 1997.
[Wel001] Wellington (B.). Secure Domain Name System (DNS) Dynamic Update. - RFC 3007, novembre 2000.
[Wes06] Wessels (D.). What is Wrong with the DNS. In RIPE-53 Meeting, Oct 2006
[WG031] Wellington (B.) et Gundmundsson (O.). Redefinition of DNS AD bit. RFC 3655, novembre 2003.
[XIL06] Programming modern FPGA http://www.Xilinx.com/univ/mpsoc2006keynote.pdf
[Yg06] Yi (K.), Gaudiot (J.) . Features of Future Network Processor Architectures.

Proceedings of the 2006 IEEE John Vincent Atanasoff International Symposium on Modern Computing (JVA 2006), Sofia, Bulgaria, October 3-6, 2006

[ZHG04] K. Zheng et al., "An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine," Proc. IEEE INFOCOM, Mar. 2004

[Zlb+06] Zhao (L.), Luo (Y.), Bhuyan (L.), Iyer (R.), A Network Processor Based Content Aware Switch, IEEE Micro Special Issue on High Performance Interconnect, May/June 2006.

Publication