Titre: Title:	Implémentation d'une mémoire cache supportant la recherche IP
Auteur: Author:	Bachir Fradj
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Fradj, B. (2019). Implémentation d'une mémoire cache supportant la recherche IP [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4178/

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Implémentation d'une mémoire cache supportant la recherche IP

BACHIR FRADJ

Département de génie électrique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie électrique

Décembre 2019

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

Implémentation d'une mémoire cache supportant la recherche IP

présenté par **B**achir FRADJ

en vue de l'obtention du diplôme de Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

Jean-Jules BRAULT, président
Yvon SAVARIA, membre et directeur
Pierre LANGLOIS, membre

REMERCIEMENTS

Tout d'abord, je tiens à remercier mon directeur de recherche, le Professeur Yvon Savaria, de m’avoir encouragé à réaliser une maîtrise en génie électrique et d’avoir dirigé mes travaux de recherche. Je le remercie également pour les ressources qu'il a mises à ma disposition et du cadre de travail très stimulant qui m’a été offert ; en m’intégrant au sein d’un laboratoire de recherche compétent et en créant des opportunités tout au long de ma maîtrise comme celle d’être en stage chez Kaloom, une entreprise très prometteuse.

Aussi, j’aimerai remercier mes collègues de laboratoire, Jeferson Santiago da Silva, Thibaut Stimpfling, Thomas Luinaud et Imad Benacer, de m’avoir intégré facilement à l’équipe, d’avoir développé mon sens de la réflexion scientifique et d’avoir été des exemples de travailleurs acharnés et passionnés.

Mais surtout, j’aimerai remercier Monsieur Normand Bélanger, assistant de recherche, bras droit de Monsieur Savaria, de m’avoir encadré et suivi durant mon apprentissage. Je le remercie pour tout le temps et l'énergie qu'il a investis pour suivre mon avancement, corriger mes lacunes et me guider face aux différents problèmes rencontrés. Je tiens à lui exprimer ma profonde gratitude pour la patience et la bienveillance dont il a fait part.

Mes remerciements sont également adressés à l'ensemble du corps enseignant ainsi qu'à tout le personnel de l'École Polytechnique de Montréal et tous ceux qui ont participé de près ou de loin au bon déroulement de cette maîtrise.

Enfin, je présente mes remerciements aux membres du jury pour leur participation à l'évaluation de ce mémoire.

RÉSUMÉ

La croissance explosive du trafic Internet a été accompagnée d'une croissance exponentielle de la bande passante des liens de transmission des équipements de traitement de données, à savoir les routeurs, rendue possible par le déploiement de la fibre optique. Les routeurs sont devenus le principal goulot d'étranglement du traitement des paquets circulant dans le réseau. L'implémentation matérielle permet aux routeurs de satisfaire les exigences de performance d'un routeur à haute vitesse en exploitant l'abondant parallélisme disponible dans le traitement des paquets. Les ASIC (« Application-Specific Integrated Circuits »), qui sont des circuits intégrés spécialisés, sont alors utilisés pour leur performance. Ces circuits induisent cependant des coûts : comme les routeurs sont construits à partir de matériel spécialisé, ce sont des périphériques à fonction fixe qui ne peuvent pas être programmés.

Beaucoup d'efforts et de travaux ont été réalisés afin de rendre les ASIC plus programmables, cependant, cela est encore insuffisant pour exprimer de nombreux algorithmes. Dû à la nécessité de mieux contrôler les opérations du réseau et à la demande constante d'offrir de nouvelles fonctionnalités, la contrainte de la programmabilité des routeurs est devenue aussi importante que la performance.

Les routeurs logiciels sont considérés plus appropriés dans le contexte où la programmabilité prime sur la performance et ils peuvent bénéficier d'une performance intéressante, comme l'incarnent les processeurs de réseau. Les processeurs réseau utilisent des accélérateurs matériels pour implémenter des fonctions spécifiques comme l'utilisation des mémoires TCAM (« Ternary Content Addressable Memory ») pour effectuer des recherches de types LPM (« Longest Prefix Match »), nécessaires pour la transmission des paquets. La TCAM satisfait le requis de débit exigé par le LPM, qui est le facteur de performance le plus limitant dans la transmission de paquets au vu de sa complexité, et elle s'est imposée comme la solution standard dans l'industrie. Cependant, elle présente des inconvénients graves : sa consommation d'énergie élevée, sa faible flexibilité (opérations de mise à jour lente) et son coût financier (coût par bit plus élevé par rapport aux autres types de mémoire). La consommation d'énergie de la TCAM est critique dans les routeurs, qui ont des budgets de puissance énergétique limités.

La motivation de notre travail est d'explorer le concept d'une mémoire cache généralisée pouvant soutenir le LPM. La mémoire sur puce d'un processeur réseau joue le rôle de mémoire cache et est

implémentée en utilisant la technologie SRAM (« Static Random Access Memory) qui est une mémoire beaucoup moins couteuse que la mémoire TCAM. Afin d'accélérer le LPM, la mémoire cache enregistre les préfixes consultés récemment, afin de diminuer le temps d'accès à la table de routage.

Dans ce mémoire, une architecture de mémoires associatives-LPM est proposée. Elle repose sur des mémoires associatives (mémoire cache partiellement associative) implémentées grâce à des fonctions de hachage. Ces mémoires permettent d'associer des préfixes de tailles différentes à des adresses de la mémoire cache, s'affranchissant de la nature pleinement associative de la mémoire TCAM.

Puisque la transition d'IPv4 à IPv6 prend une importance croissante, ce travail cible le LPM pour IPv6. Nous avons utilisé un outil de synthèse de haut niveau (« High-Level Synthesis » - HLS) ciblant une carte FPGA (« Field-Programmable Gate Array ») ZC706 de Xilinx pour émuler la fonctionnalité de la mémoire cache. Nous montrons qu'une table de 26 000 préfixes de 128 bits s'insère dans une mémoire cache de 1 Mo et que le débit atteint permet de transmettre des paquets à un débit de 160 Gbps. La carte FPGA a servi de support pour concevoir une plateforme de prototypage configurable pour la conception et le test de fonctions de réseau.

ABSTRACT

The Internet's traffic growth has been accompanied by an exponential growth in the bandwidth of the data processing equipment transmission links, made possible by the deployment of optical fiber. Routers have, therefore, become the main bottleneck in the packets processing speed across the network. The hardware implementation allows routers to meet performance requirements of a high-speed router by exploiting the abundant parallelism available in packet processing. ASICs (Application-Specific Integrated Circuit) are specialized integrated circuits used for their performance, but they have a cost: as routers are built from specialized hardware, they are fixed-function devices that do not cannot be programmed.

Much effort and work has been done to make ASICs more programmable, however, this is still insufficient to express many algorithms. Due to the need to better control network operations and the constant demand to support new features, the constraint of router programmability has become as important as performance. Hardware specification is the only way to achieve performance requirements for high-speed routers, however some contexts involve high computing requirements with lower link speeds. Software routers are considered more appropriate in the context where programmability takes precedence over performance and can benefit from interesting performance, as incarnated by network processors. Network processors use hardware accelerators to implement specific functions like TCAM (Ternary Content Addressable Memory) memories to perform LPM (Longest Prefix Match) lookup, required for packet transmission. The TCAM meets the speed required by the LPM, which is the most limiting performance factor in packet transmission due to its complexity, and has been an effective standard in the industry. However, TCAMs have some serious disadvantages: their high-power consumption, their poor scalability and their higher cost per bit compared to other memory types.

The motivation of our work is to explore the concept of a generalized cache memory that can support the LPM. The on-chip memory of a network processor acts as a cache memory and is implemented using SRAM technology (Static Random-Access Memory) which is a much less expensive memory than TCAM memory. In order to speed up LPM lookup, the cache memory stores the prefixes consulted recently, in order to reduce the access time to the routing table.

In this thesis, an architecture is proposed which relies on associative memories implemented by hash functions. As the transition from IPv4 to IPv6 becomes primordial, this work targets the LPM

for IPv6. We used HLS (High-Level Synthesis) tools targeting a Xilinx ZC706 FPGA (Field-Programmable Gate Array) to emulate the functionality of the cache. We show that a table of 26,000 prefixes is inserted in a 1 MB cache and the throughput achieved allows processing packets at wire speed over four 40Gb links. The FPGA is used as prototyping platform for designing and testing network functions, including our solution.

TABLE DES MATIÈRES

REMERCIEMENTS III

RÉSUMÉ IV

ABSTRACT VI

TABLE DES MATIÈRES VIII

LISTE DES TABLEAUX XI

LISTE DES FIGURES XII

LISTE DES SIGLES ET ABRÉVIATIONS XIV

CHAPITRE 1 INTRODUCTION 1

CHAPITRE 2 NOTIONS FONDAMENTALES 4

2.1 Introduction 4

2.2 Modèles 4

2.2.1 Modèle TCP/IP 5

2.2.2 Encapsulation 6

2.3 Réseaux de communication de données 7

2.3.1 Ethernet / Réseau local 8

2.3.2 Internet / Réseau mondial 9

2.4 Conclusion 12

CHAPITRE 3 LES ROUTEURS 14

3.1 Introduction 14

3.2 Fonctionnalités d'un routeur 14

3.3 Plan de données, de contrôle et de gestion 16

3.4 Architecture 18

3.4.1 Compromis entre performance et programmabilité 18

3.4.2 Révolution SDN 21
3.4.3 Travaux de recherche courants 24
3.5 Conclusion 27
CHAPITRE 4 RECHERCHE IP 29
4.1 La recherche (« lookup ») 29
4.2 LPM 30
4.2.1 Recherche IP —Contexte 30
4.2.2 Adressage IPv6, routage et transmission de paquets 31
CHAPITRE 5 SYNTHÈSE DES SOLUTIONS DISPONIBLES 41
5.1 Structures de données et algorithmes 41
5.1.1 Listes et tables 41
5.1.2 Structure arborescentes 45
5.2 TCAM 47
5.3 Conclusion 52
CHAPITRE 6 MÉMOIRE CACHE SUPPORTANT LA RECHERCHE IP 53
6.1 Mémoire cache de processeur 53
6.1.1 Architecture de base de la mémoire cache 54
6.1.2 Opération du contrôleur de cache 54
6.1.3 Types de correspondance 55
6.1.4 Approche de conception 58
6.2 Mémoire cache soutenant les recherches IP 60
6.2.1 Hachage groupé 60
6.2.2 Fonctions de hachage 63
6.2.3 Architecture 67

CHAPITRE 7 ÉVALUATION DE LA CACHE 74
7.1 Résultats 74
7.1.1 Structure de données 74
7.1.2 Latence et débit 77
7.1.3 Ressources physiques 78
7.2 Plateforme de prototypage 79
CHAPITRE 8 CONCLUSION 84
RÉFÉRENCES 85

LISTE DES TABLEAUX

Tableau 4-1 : Format d’une adresse « unicast » globale 38

Tableau 6-1 : Nombre de collisions maximal dans une case de hachage 61

Tableau 6-2 : Nombres de cases de hachage par groupe de préfixes 63

Tableau 6-3 : Matrice H1 66

Tableau 7-1 : Résultats temporels 78

Tableau 7-2 : Consommation des ressources physiques 78

Tableau 7-3 : Signaux de l’interface AXI4-Stream 80

LISTE DES FIGURES

Figure 2-2: Couches du modèle ISO/OSI 5
Figure 2-3 : Couches du modèle TCP/IP 6
Figure 2-4 : Mécanisme d'encapsulation 7
Figure 2-5 : Transmission d'un paquet IP 11
Figure 3-1 : Plan de données, de contrôle et de gestion [13] 17
Figure 3-2 : Architecture typique d'un NP [12] 19
Figure 3-3 : Évolution de la performance des routeurs [1] 20
Figure 3-4 : Algorithmes de routeur [1] 21
Figure 3-5 : Architecture SDN [13] 22
Figure 3-6: Routeur ASIC [21] 25
Figure 4-1 : En-tête IPv6 [9] 32
Figure 4-2 : Types d'adresse IPv6 36
Figure 5-1 : Correspondance du hachage 42
Figure 5-2: Hachage parallèle [6] 43
Figure 5-3: Implémentation du hachage parallèle grâce à l'extension des préfixes [7] 44
Figure 5-4: Structure d'arborescence binaire [35] 46
Figure 5-5 : Structure de la mémoire CAM [9] 48
Figure 5-6: Implémentation basée sur une TCAM pour le LPM [8] 49
Figure 5-7: Représentation fonctionnelle d'un cache pour la recherche IP [36] 50
Figure 5-8: Routeur basé sur des processeurs réseau [39] 51
Figure 6-1: Hiérarchie mémoire 53
Figure 6-2: Architecture d'une mémoire cache [47] 55
Figure 6-3: Mémoire cache à correspondance préétablie [47] 57

Figure 6-4: Mémoire cache partiellement associative [47] 58
Figure 6-5: Schéma fonctionnel d'un cache ARM940T [47] 59
Figure 6-6: Groupes de préfixes 62
Figure 6-7: Registre à décalage à rétroaction linéaire [44] 64
Figure 6-8: Correspondance d'un préfixe 67
Figure 6-9: Recherche IP dans un ensemble de cache 70
Figure 6-10: Synthèse par défaut d'une boucle *for* [48] 71
Figure 6-11: Optimisation d'une boucle *for* [48] 71
Figure 6-12: Optimisation de la latence et du débit [49] 72
Figure 6-13: Partitionnement de la mémoire [49] 72
Figure 6-14 : code original [50] 73
Figure 6-15 : code optimisé [50] 73
Figure 7-1 : Collisions des fonctions de hachage universelles 75
Figure 7-2: Distribution de la fonction de hachage universelle la plus performante 75
Figure 7-3: Distribution de nos fonctions de hachage 76
Figure 7-4: Architecture de la plateforme de prototypage 79
Figure 7-5: Signaux du protocole ap_ctrl_hs 81
Figure 7-6: Interface entre le FPGA et la DDR3 82
Figure 7-7: Architecture du banc d'essai fonctionnel 83

LISTE DES SIGLES ET ABRÉVIATIONS

ARP	Address Resolution Protocol
AS	Autonomous Systems
ASIC	Application Specifie lntegrated Circuits
BGP	Border Gateway Protocol
BRAM	Block Random Access Memory
CAM	Content Addressable Memory
CIDR	Classless Inter-Domain Routing
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DoD	Department of Defense
DRAM	Dynamic Random Access Memory
DHCP	Dynamic Host Configuration Protocol
DWDM	Dense Wavelength Division Multiplexing
EGP	External Gateway Protocol
FIB	Forwarding Information Base
FPGA	Field-Programmable Gate Array
GPP	General Purpose Processor
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers
ICMP	Internet Control Message Protocol
IGP	Interior Gateway Protocol
IoT	Internet of Things

IP	Internet Protocol
IPSec	Internet Protocol Security
ISO	International Standard Organization
LAN	Local Area Networks
LFSR	Linear Feedback Shift Register
LIR	Local Internet Registry
LPM	Longest Prefix Match
MAC	Media Access Control
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NHI	Next Hop Information
NP	Network Processor
NVGRE	Network Virtualization using Generic Routing Encapsulation
OSI	Open Systems Interconnection
OSPF	Open Shortest Path Bridging
OTV	Overlay Transport Virtualization
P4	Programming Protocol-Independent Packet Processors
PBB	Provider Backbone Bridge
PCI-E	Peripheral Component Interconnect Express
PDU	Protocol Data Unit
RAM	Random-Access Memory
RFC	Requests For Comments
RIR	Regional Internet Registry
RIP	Routing Information Protocol

RT	Routing Table
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SoC	System on a Chip
SRAM	Static Random-Access Memory
STT	Stateless Transport Tunnelling
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
TTL	Time-To-Live
UDP	User Datagram Protocol
VHDL	VHSIC Hardware Description Language
VoIP	Voice over Internet Protocol
VxLAN	Virtual Extensible LAN

CHAPITRE 1 INTRODUCTION

Les systèmes réseau (« network systems ») sont conçus pour répondre aux exigences fonctionnelles spécifiées par les protocoles. Ils doivent également satisfaire des exigences de performance constamment croissantes; dues à l'augmentation du nombre d'utilisateurs d'Internet et l'émergence de nouvelles applications de plus en plus gourmandes en bande passante. Le déploiement de la fibre optique a entraîné une explosion des débits, soutenus par les liens de transmission, ce qui a conduit les routeurs à devenir le principal goulot d'étranglement de la performance actuelle d'Internet.

Un routeur implémente de nombreuses fonctionnalités, autres que la transmission de paquets, relatives à la sécurité, à la surveillance et à la performance. L'évolution des routeurs était principalement motivée par la performance des fonctions de traitement de paquets. En raison de la nécessité d'offrir de nouvelles fonctionnalités et le besoin constant d'évolutivité, la programmabilité des routeurs est devenue aussi importante que leur performance. L'équilibre entre la performance et la programmabilité imposent des compromis majeurs lors de la conception de systèmes réseau, car ces caractéristiques exigent des contraintes d'implémentation différentes.

Les routeurs basés sur des ASIC représentent la meilleure solution pour satisfaire les besoins de performance, car ils utilisent des fonctions matérielles dédiées et exploitent l'abondant parallélisme disponible dans le traitement des paquets. Cependant, ils présentent l'inconvénient d'avoir un temps de commercialisation élevé (« time-to-market »), d'une part, et d'être rigide d'autre part. Les routeurs logiciels [1], basés sur des plateformes programmables, sont plus adaptés aux applications nécessitant plus de flexibilité. Ces derniers ont évolué au cours des années pour offrir une performance intéressante, comme l'incarnent très bien les processeurs de réseau (« Network Processors »). Les processeurs de réseau implémentent certaines fonctions en logiciel et utilisent des accélérateurs matériels pour les fonctions dont les requis de performance sont importants.

La recherche IP [2] est souvent le goulot d'étranglement de la transmission de paquets. Le routeur doit faire correspondre l'adresse IP de destination de chaque paquet à une table de routage, qui peut contenir des millions d'entrées [3], pour déterminer le prochain saut du paquet vers sa destination finale. Les réseaux IP sont identifiés en utilisant la notation CIDR (« Classless Inter-Domain Routing »). Le routage inter-domaine sans classe permet de créer une hiérarchie d'adresses réseau

en définissant des sous-réseaux IP, où la partie de l'adresse IP utilisée pour le routage est le préfixe. La notation CIDR a permis de réduire la taille des tables de routage en spécifiant une large plage d'adresses dans une entrée de la table de routage grâce à un préfixe. Cependant, la recherche IP devient plus complexe ; le CIDR nécessite que les routeurs traitent des préfixes de taille variable, où plusieurs préfixes peuvent correspondre à l'adresse IP à rechercher. Le LPM est le critère de priorisation pour retourner la meilleure correspondance, où le préfixe le plus long représente le meilleur résultat de recherche. Puisque la transition d'IPv4 à IPv6 devient primordiale, nous ciblons le LPM pour IPv6.

Les mémoires TCAM [4] sont une solution standard et répandue en industrie pour effectuer la recherche IP de manière performante. Chaque cellule mémoire peut stocker un bit valant '0', '1' ou 'X' (« wildcard » ou entrées indifférentes) ; les entrées indifférentes permettent de faire des recherches masquées, où certains bits de la clé de recherche peuvent être ignorés. Chaque préfixe de la table de routage est stocké dans un mot complété par des « wildcards » et, lors d'une opération de recherche, la mémoire adressable par contenu compare simultanément l'adresse IP à tous les mots stockés. Parce qu'une TCAM effectue une recherche dans sa mémoire en un seul cycle d'horloge, elle est plus rapide que les autres types de mémoire dans toutes applications de recherche d'un contenu. Les TCAM constituent des accélérateurs matériels et elles peuvent s'interfacer avec un processeur de réseau comme une mémoire ordinaire. Elles sont également utilisées dans la conception de mémoire cache.

Certains routeurs utilisent un système de mise en cache [5] (« caching ») pour accélérer la recherche IP. La mémoire cache stocke les préfixes consultés récemment, afin de diminuer le temps d'accès à la table de routage.

En dépit de certaines propriétés fort intéressante, les TCAM présentent certains inconvénients graves : leur consommation d'énergie élevée, leur faible flexibilité et leur coût par bit supérieur par rapport aux autres types de mémoire. La motivation de notre travail est de concevoir une mémoire cache, pouvant supporter la recherche IP, en proposant une solution matérielle alternative à la TCAM et capable de satisfaire les besoins de performance.

Les solutions basées sur le hachage [6] sont utilisées pour effectuer une recherche IP dans un sous-ensemble de la table de routage au lieu de la totalité de la table, afin de réduire le nombre de préfixes à analyser. L'accès s'effectue par une fonction de hachage, qui, à partir de l'adresse IP à rechercher,

calcule une valeur qui indexe les éléments de la table de hachage (« buckets ») qui, contient les préfixes. Cependant, le hachage présente des défis majeurs. Le premier défi consiste à gérer les collisions générées par le hachage qui dégradent la performance. Une collision est produite quand deux clés de hachage différentes produisent le même résultat de hachage (pointent vers la même case de hachage). Ces collisions conduisent à des opérations de recherche supplémentaires sont nécessaires et une mauvaise distribution des préfixes conduit à une utilisation partielle et inefficace de la mémoire. Le deuxième défi concerne la taille variable des préfixes, qui pose des problèmes lors de la définition des clés de hachage (les clés doivent avoir une taille définie). Le hachage avec les « wildcards » peut être résolu par le hachage restreint (« restricted hashing ») ou le hachage groupé (« grouped hashing ») [7]. Dans le hachage groupé, les préfixes sont regroupés en fonction de leurs longueurs et une fonction de hachage différente est utilisée pour chaque groupe.

Le but de notre travail est de proposer une architecture qui repose sur des mémoires associatives (mémoire cache partiellement associative), implémentée grâce au hachage groupé, et qui permet d'associer des préfixes de tailles différentes à des adresses de la mémoire cache. Cette solution exploite l'implémentation des mémoires caches de processeurs, qui utilisent des mémoires SRAM, six fois moins couteuses que les mémoires TCAM [8]. Dans ce mémoire, nous visons à prouver la faisabilité d'une telle approche et de présenter des résultats de performance satisfaisants.

Ce mémoire est constitué de huit chapitres dont le premier représente l'introduction. Le deuxième chapitre vise à présenter les notions fondamentales sur lesquelles reposent les réseaux de communication. Le troisième chapitre présente les fonctionnalités d'un routeur, ses exigences de performance, ainsi que les architectures et les implémentations dominantes. Le chapitre 4 est consacré à l'adressage IP pour permettre de comprendre la complexité et le défi que présente la recherche IP. Le chapitre 5 présente les différentes solutions matérielles et logicielles dominantes dans la littérature. Le chapitre 6 détaille la méthodologie et la conception de notre solution. Dans le chapitre 7, nous discutons des résultats obtenus et nous présentons une implémentation concrète de notre solution. Enfin, le chapitre 8 clos ce mémoire.

CHAPITRE 2 NOTIONS FONDAMENTALES

2.1 Introduction

Les réseaux de communication (« communication networks ») sont apparus sous forme de connexions physiques entre pairs (« peer-to-peer »), transportant d'abord des données en flux analogique, puis évoluant vers des connexions acheminant des données en flux numérique: les données de communications sont segmentées en paquets. Le paquet est l'unité de données qui est acheminée entre deux hôtes dans un réseau.

La croissance rapide des applications et des réseaux de communications est accompagnée d'une croissance exponentielle du nombre et du débit des paquets circulant dans le réseau, qui doivent être traités. Le traitement du flot des flots des données circulant sur le réseau concerne le traitement de paquets en fonction des protocoles, des demandes et du comportement du réseau. Il est donc essentiel de comprendre les notions fondamentales sur lesquelles reposent les réseaux de communication pour savoir comment un paquet est créé pour pouvoir l'analyser et le traiter.

2.2 Modèles

Au fur et à mesure que la programmation des réseaux de communication, des interfaces et des équipements réseaux devenaient de plus en plus sophistiqués, l'organisation internationale de normalisation (« International Standard Organization », ISO) proposait une architecture de réseau structurée en couches abstraites appelée OSI (Open Systems Interconnection). Ce modèle a pour but de permettre l'interconnexion de systèmes hétérogènes et de s'adapter à la croissance des flux de données à traiter en normalisant les communications. Le modèle ISO/OSI comporte sept couches, représentées à la figure 1-1, où chaque couche se rapporte à une fonction réseau bien précise, qu'on appelle *service*.

Les données à transmettre entre deux nœuds suivent un chemin défini : les données du nœud émetteur traversent d'abord la couche la plus abstraite (couche application, L7), avant de passer par les couches successives jusqu'à la couche la moins abstraite (couche physique, L1), pour être transmises (sous la forme de trames) au support de communication. Au niveau du nœud destinataire, les données traversent le chemin inverse (de la couche physique vers la couche application).

L'activité de chaque couche est codifiée selon un certain *protocole*. Un protocole est un ensemble de règles qui détermine le format et la transmission des données. Chaque couche prépare les données pour la couche suivante : deux couches superposées communiquent donc entre elles et partagent une *interface*. Enfin, la fonctionnalité d'une couche peut être réalisée par un équipement réseau différent de celui des autres couches.

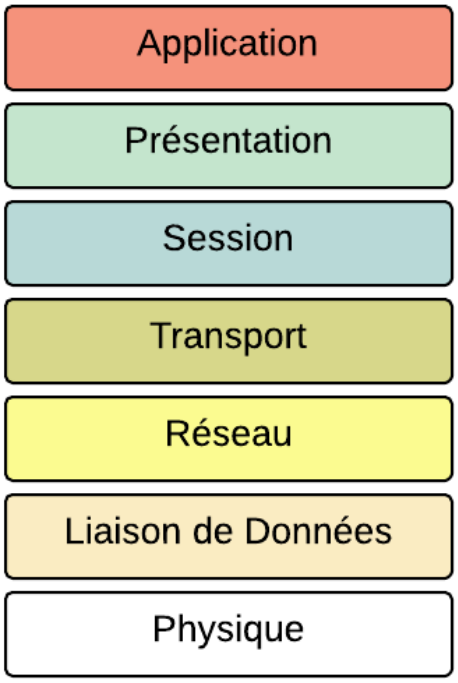

Figure 2-1: Couches du modèle ISO/OSI

2.2.1 Modèle TCP/IP

Le département américain de la Défense (« Department of Defense ») a proposé un autre modèle en couches axé sur la modélisation de réseau de données, plus simple et contenant moins de couches ; comme représenté dans la figure 1-2. Ce modèle, le TCP/IP, est fondé, tout comme le modèle ISO/OSI, sur le concept de **pile de protocoles indépendants**. Néanmoins, on remarque que les couches présentation et session ne sont plus présentes dans le modèle TCP/IP car elles semblaient peu utiles (peu utilisées).

La révolution du modèle TCP/IP réside dans la couche Internet (L3). Cette couche réalise l'interconnexion des réseaux (hétérogènes) distants **sans connexion**. Comme aucune connexion n'est présente, l'acheminement des paquets se fait de manière indépendante, ainsi les paquets

peuvent arriver dans le désordre. La couche Internet a une implémentation officielle : le protocole Internet (« Internet Protocol »). En effet, contrairement au modèle ISO/OSI, le modèle TCP/IP est né d'une implémentation : les protocoles TCP et IP. La normalisation est venue par la suite. Les couches situées au-dessus de l'IP sont responsables de la création de divers types de connexion pour de nombreuses applications (par exemple, l'application HTTP, pour la navigation sur le Web). Les deux types de protocoles définis dans la couche de transport (L4) sont les protocoles TCP et UDP.

Aujourd'hui, le modèle TCP/IP, plus souple, s'est imposé comme modèle de référence. Le modèle OSI, plus rigoureux, est principalement utilisé pour certaines applications critiques, ou pour garantir une qualité de service.

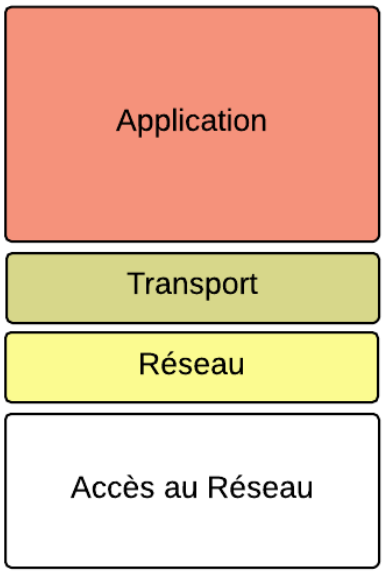

Figure 2-2 : Couches du modèle TCP/IP

2.2.2 Encapsulation

Les deux modèles reposent sur le concept de pile de protocoles, grâce à un mécanisme d'*encapsulation*. Chaque couche ajoute son propre en-tête au paquet, avec des informations spécifiques à la couche en question (définies par le protocole), ce qui permet aux couches homologues du nœud récepteur de pouvoir interpéter correctement le paquet. Les données sont ainsi enveloppées à chaque couche et portent le nom de PDU et sont constituées de l'en-tête

spécifique à la couche courante et des données initiales auxquelles on a ajouté les en-têtes des couches qui la précèdent.

La figure 1-3 illustre ce mécanisme d'encapsulation et représente les PDU associés aux couches application, réseau et physique. Ainsi, on définit une trame comme étant la structure de données transportée à travers un lien physique. Le paquet représente l'unité d'encapsulation transmise à travers l'interface entre la couche réseau et la couche liaison, représentant ainsi des données qui voyagent indépendamment sur le réseau et dont la taille est limitée. Enfin, un datagramme représente l'unité de donnée utilisée par les applications.

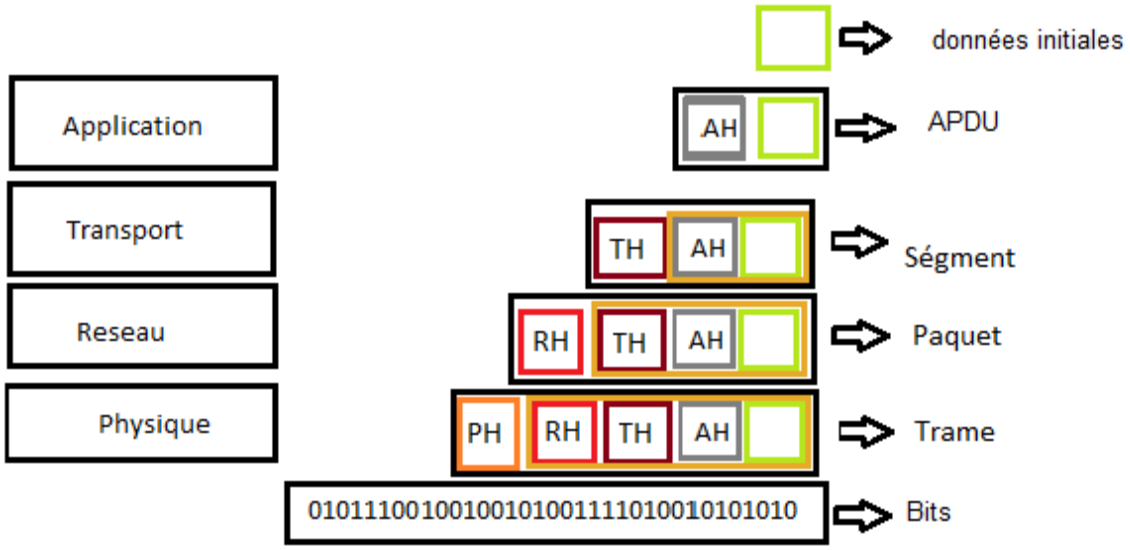

Figure 2-3 : Mécanisme d'encapsulation

Lors de la réception, l'en-tête spécifique à chaque couche est enlevé : les couches supérieures n'ont pas besoin des en-têtes des couches inférieures.

2.3 Réseaux de communication de données

Il existe de nombreux types de réseaux de données, ainsi que plusieurs topologies pour les catégoriser, utilisant des technologies différentes. Cependant, on distingue deux technologies très répandues : Ethernet et Internet.

2.3.1 Ethernet / Réseau local

La technologie Ethernet fut la première technologie à haut débit pour les réseaux locaux (« Local Area Networks », LAN). Elle a été conçue originalement pour une topologie physique et logique en bus, où les nœuds d'un réseau LAN sont reliés les uns aux autres par un canal de diffusion commun. Chaque machine a une adresse distincte et propre, permettant de l'identifier, appelée adresse MAC. Elle représente l'adresse physique d'un périphérique et est sensée être unique au monde. Cette topologie reposait sur le protocole CSMA/CD, qui permet d'accéder au support de transmission et de gérer les collisions de communications.

Ethernet fut ensuite adapté (norme 10BASE-T) pour utiliser des topologies physiques en étoile sur câbles à paires torsadées (équipés de connecteurs RJ45), les pairs étant rattachés à des concentrateurs (« hubs »). Néanmoins, la topologie logique reste le bus et le médium reste partagé (tout le monde reçoit toutes les trames), provocant des collisions potentielles. Pour résoudre les problèmes liés aux collisions, les commutateurs (« switches ») ont été développés afin de maximiser la bande passante disponible, en reprenant les câbles à paires torsadées (puis la fibre optique), permettant d'avoir des topologies plus souples : topologie logiques et physiques en étoile où les communications entre deux pairs donnés sont isolées.

Un commutateur transporte les trames qu'il reçoit vers un de ses ports, grâce à une table qui contient les adresses MAC de ses interfaces. L'usage de commutateurs permet de ne plus avoir de collisions et de construire des réseaux plus étendus géographiquement. Le commutateur est également utilisé pour relier des segments de réseaux subdivisés (un grand réseau peut être subdivisé pour une meilleure gestion du réseau).

Le protocole Ethernet intervient dans les couches liaison de données (niveau 2) et physique (niveau 1) du modèle ISO/OSI et il permet à deux (ou plus) hôtes de communiquer dans un réseau local en utilisant les adresses MAC associées à chaque hôte. Cependant, la mise en place d'un réseau constitué de commutateurs à l'échelle mondiale n'est pas réalisable, il est nécessaire d'utiliser des systèmes spécialement conçus pour assurer une connectivité globale. De plus, Ethernet est peu flexible, par exemple, à chaque fois qu'une application serveur ou une application client modifie son adresse MAC (par exemple, on remplace une plate-forme matérielle), une nouvelle adresse MAC doit être associée et distribuée à tous les pairs qui veulent communiquer avec elle. Pour résoudre ce problème, le protocole Internet (couche L3) utilise des adresses logiques IP qui peuvent

être facilement configurées. La couche L3 est efficace pour les réseaux locaux, mais elle est surtout nécessaire pour l'interconnexion de réseaux.

2.3.2 Internet / Réseau mondial

2.3.2.1 Vue d'ensemble

Internet est un réseau informatique couvrant l'ensemble du globe, composé de milliers de réseaux LAN interconnectés (aussi bien publics que privés). Étant donné qu'Internet est en perpétuelle croissance et inclut des centaines de milliers de réseaux, une approche hiérarchique est utilisée pour découper le réseau en domaines appelés systèmes autonomes (« Autonomous Systems », AS), représentant ainsi des agrégations de réseaux. Chaque AS est identifié par un numéro unique (16 bits) alloué par la Société pour l'attribution des noms de domaine et des numéros sur Internet (« Internet Corporation for Assigned Names and Numbers », ICANN) et le registre Internet régional (« Regional Internet Registry », RIR). Le nombre de systèmes autonomes enregistrés est passé d'environ 10 000 en 2000 à plus de 60 000 en 2013 [10], ce qui est un bon indicateur de l'augmentation substantielle de la complexité d'Internet.

Internet s'appuie sur le modèle TCP/IP et repose sur un ensemble standardisé de protocoles. Le protocole IP est l'implémentation officielle de la couche réseau et est devenue la plateforme la plus commune pour les applications de communications de données et, également, pour plusieurs services de télécommunications comme la tété (IPTV). La quatrième génération du système de téléphonie cellulaire (4G) est également basée sur une infrastructure IP.

La fonctionnalité principale de la couche réseau consiste à fournir une connectivité de bout en bout entre deux hôtes, se trouvant dans le même réseau ou deux réseaux différents. Cette fonctionnalité est la base de la conception d'Internet. Les réseaux IP utilisent un mécanisme de routage par saut (« hop-by-hop ») à commutation de paquets et sont indépendants du matériel. Le routage est l'une des fonctionnalités de protocole importantes qui doivent être implémentées dans la couche réseau afin d'obtenir une connectivité de bout en bout. Le routage consiste à déterminer comment atteindre une interface réseau donnée et représente, avec l'adressage et la transmission de paquets, les trois notions importances pour comprendre la fonctionnalité de la couche réseau.

2.3.2.2 Adressage, routage et transmissions de paquets

L'abstraction des réseaux IP du matériel découle du mode d'adressage choisi : on affecte à chaque interface une adresse logique, qu'on appelle adresse IP, découlant de l'adresse du réseau auquel elle est connectée. Le protocole IP utilise des champs d'adresses de 32 bits pour la version IPv4 et 128 bits pour la version IPv6 et chaque hôte peut avoir une ou plusieurs adresses IP. La motivation d'utiliser des adresses IP de 128 bits est d'étendre l'espace d'adressage, qui s'épuise [11], afin d'assurer les besoins futurs. Dans cet ouvrage, nous nous intéressons à la version 6 du protocole IP. L'adressage IP sera discuté plus en détail au chapitre 3. Notons que d'autres mécanismes sont utilisés pour pallier les problèmes de la pénurie des adresses IP comme la traduction d'adresses réseau (« Network Address Translation », NAT) qui permet de faire correspondre une seule adresse externe publique visible sur Internet (adresse routable) à toutes les adresses d'un réseau privé (réseau intranet).

Le protocole Internet lie les adresses IP aux adresses MAC L2 (pour la transmission de paquets sur Ethernet L2) à l'aide du protocole ARP. L'hôte émetteur diffuse un paquet ARP en demandant l'adresse MAC L2 de l'hôte possédant l'adresse IP de destination. Si l'hôte se trouve dans le même réseau IP, l'adresse MAC de l'hôte destinataire est renvoyée, sinon, l'adresse MAC du routeur par lequel il est doit passer pour sortir du réseau est renvoyée. Le routeur est le dispositif responsable de faire transiter les paquets d'un réseau vers une autre car il dispose de plus d'une carte réseau dont chacune est reliée à un réseau différent.

Il est important de distinguer la notion de routage de paquets à celle de la transmission de paquet. Le routage détermine la route à suivre pour chaque paquet pour arriver à sa destination, alors que la transmission correspond à l'action d'acheminer un paquet d'une interface à une autre. Quand le routeur reçoit une trame provenant d'une machine connectée à un des réseaux auquel il est rattaché, il extrait l'adresse IP de destination de l'en-tête IP du paquet et une décision de transmission est prise en fonction de l'adresse IP de destination et de sa table de routage. La table de routage contient les adresses des réseaux attachés au routeur ainsi que les interfaces (adresses des cartes réseau) par lesquelles les paquets doivent être acheminés. Les tables de routage sont construites et maintenus notamment grâce au protocole de routage. Sur la base de cette décision, le paquet est acheminé vers le routeur du saut suivant ou vers sa destination. La figure 1-4 montre comment un paquet est acheminé entre deux nœuds dans deux réseaux différents.

La topologie hiérarchique présentée précédemment (systèmes autonomes) permet de gérer le routage de manière interne et indépendante : en utilisant des protocoles de routage appelés IGP, opérant dans un seul AS pour optimiser des routes utilisées dans sa propre structure (exemple : le protocole RIP) et ; en utilisant des protocoles de routage EGP pour échanger des informations de routages entre deux AS (exemple : le protocole BGP).

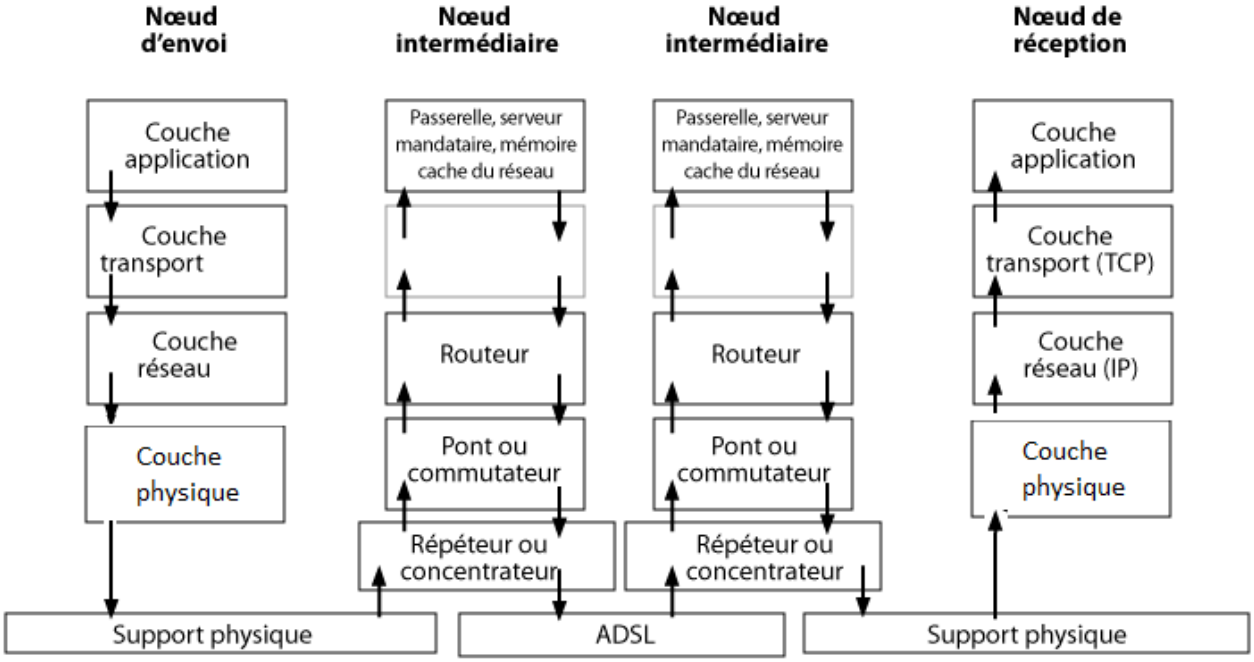

Figure 2-4 : Transmission d'un paquet IP

Cependant, à mesure que les routeurs deviennent plus centraux et ont de nombreuses adresses de destination (i.e. de nombreuses interfaces de sortie) à gérer, il peut s'avérer difficile de choisir parmi les interfaces possibles pour l'adresse de destination appropriée (correspondance optimale). En effet, un des mécanismes les plus répondus est le routage inter-domaines sans classe (« Classless Inter-Domain Routing », CIDR), qui permet à un réseau IP de se décomposer en plusieurs sous-réseaux (subdivision logique) pour divers intérêts comme la séparation des services, rendre le réseau plus flexible et limiter la propagation des « broadcast » (opération coûteuse qui consiste à diffuser les données à toutes les machines du réseau) ; il ajoute de la complexité et influence l'adressage, le routage et la transmission des paquets. Comme chaque entrée de la table de routage peut spécifier un sous-réseau, une adresse de destination peut correspondre à plusieurs entrées de table. Une stratégie de recherche appelée « Longest Prefix Match » (LPM) est utilisée pour trouver le port de sortie représentant la meilleure correspondance.

La transmission des paquets (« packets forwarding ») représente l'une des principales tâches de traitement de paquets. La transmission de paquets et ses exigences de performance et fonctionnelles seront couvertes dans les chapitres suivants. L'adressage, ainsi que d'autres sujets liés à la transmission de paquets seront discutés ; néanmoins, nous n'allons pas développer le sujet du routage. En effet, un algorithme de routage spécifique mis en œuvre ne prend pas en compte les exigences de transmission [12], tant que les paquets sont transférés vers les ports de sortie appropriés, tout algorithme de routage possible (y compris une table configurée de manière statique) est acceptable. Il s'agit donc de deux fonctions ayant deux implémentations propres et distinctes.

2.3.2.3 Évolution des routeurs

Historiquement, l'objectif principal d'Internet consistait à interconnecter facilement une grande variété de réseaux existants (par exemple, réseaux longue distance, réseaux locaux, réseaux de satellites et réseaux radio) via un ensemble de routeurs entre ces réseaux. De ce fait, la plus grande partie de la complexité résidait dans les hôtes finaux, alors que les routeurs fonctionnaient selon le principe de « best-effort » [1] (faire au mieux) pour transmettre des paquets, n'assurant aucune fiabilité et limitant beaucoup de services comme : la hiérarchisation des paquets, les fonctions de sécurité permettant de détecter les défaillances du réseau, etc. Les premiers routeurs étaient donc singulièrement dédiés à la transmission de paquets. Aujourd'hui, quatre décennies après la publication des idées sous-jacentes à Internet, il est clair que les routeurs doivent faire beaucoup plus que la transmission de paquets et d'autres objectifs tels que la performance, la sécurité et la surveillance gagnent en importance. En conséquence, un routeur typique implémente aujourd'hui de nombreuses fonctionnalités au-delà de la transmission de paquets, relatives à la sécurité (par exemple, contrôle d'accès), à la surveillance (par exemple, compter le nombre de paquets appartenant à chaque flux) et à la performance (par exemple, files d'attente prioritaires). Nous discuterons dans le chapitre suivant de l'évolution des routeurs (architectures et implémentations), ainsi que des exigences de performance et fonctionnelles auxquels ils sont confrontés.

2.4 Conclusion

Ce chapitre avait pour but de définir les concepts fondamentaux sur lesquels reposent les réseaux de communication de données. Un routeur possède plusieurs interfaces réseaux par lesquelles il

fait transiter les paquets IP qu'il reçoit. Un paquet IP est transporté de routeur en routeur, potentiellement, jusqu'à arriver à sa destination. Le routeur représente ainsi la pierre angulaire d'Internet. Cependant, aux vues des exigences exponentielles que nécessite le traitement de paquets, d'autre concepts et technologies ont vues le jour et on fait évoluer la manière dont on traite le réseau.

CHAPITRE 3 LES ROUTEURS

3.1 Introduction

On distingue deux catégories de fonctions de traitement de paquets : les fonctions du plan de contrôle (« control plane ») et les fonctions du plan de données (« data plane » ou « forwarding plane »). Les fonctions de contrôle sont destinées à configurer l'équipement réseau alors que les fonctions du plan de données sont destinées à traiter les paquets et transporter le trafic en se basant sur la configuration de l'équipement réseau. Les fonctions du plan de contrôle s'exécutent relativement peu souvent en comparaison avec les fonctions du plan de données : de l'ordre d'une fois toutes les millisecondes, par opposition à une fois toutes les nanosecondes. Ce qui se traduit par des requis et des implémentations fort différentes.

Le traitement réseau désigne ainsi le traitement des paquets entrants dans le réseau selon un ensemble de règles qui gèrent leur transmission sur un lien de communication sortant. Les progrès récents en matière de technologie de réseau ont entraîné une explosion des débits soutenus par les équipements réseau. Disposant de la technologie de fibre optique rapide pour la transmission de données, les éléments de traitement de données, à savoir les routeurs, sont devenus le principal goulot d'étranglement [3] de la performance actuelle d'Internet.

Dans ce chapitre, nous allons présenter les fonctionnalités d'un routeur, ses exigences de performance, ainsi que les architectures et les implémentations dominantes. Ce chapitre a également pour but d'introduire l'objectif et l'intérêt du projet de recherche dans le cadre duquel cette recherche s'inscrit.

3.2 Fonctionnalités d'un routeur

Les exigences fonctionnelles pour un routeur IP sont déterminées par une série de documents officiels décrivant les aspects et spécifications techniques d'Internet (« Requests For Comments, RFC »). Un routeur implémente plusieurs tâches qui nécessitent différentes fonctions. Le processus de traitement de paquets peut se résumer à trois étapes [12] : le traitement d'entrée (opérant le traitement de la couche liaison pour recevoir un paquet IP), la transmission de paquet et le traitement de sortie (mise en file d'attente et ordonnancement des paquets ainsi que le traitement de la couche de liaison pour la transmission).

Les tâches de traitement de paquets impliquent essentiellement [9] :

- La mise en trame (« framing »)
- L'analyse et l'extraction des champs d'entête du paquet (« parsing »)
- La classification
- La recherche (« lookup »)
- La modification
- La compression et le cryptage
- La gestion du trafic (« traffic management » et « queueing »)
- La transmission des paquets (« forwarding »)

Le traitement des paquets commence par l'entrée du paquet sur un port d'entrée associé à une des interfaces de l'équipement réseau et sa mise en trame. Le « framing » garanti que les paquets sont reçus correctement et corrige, dans certain cas, les bits incorrects du paquet. Par la suite, les paquets sont analysés et classés, en fonction des exigences de l'application. Ce tri est basé sur plusieurs champ de paquets, extraits durant le « parsing », qui seront comparés à un ensemble de règles contenues dans une ou plusieurs tables de classification. Traditionnellement, cinq champs (« 5-uplets ») sont utilisés : l'adresse IP Source, l'adresse IP Destination, le Protocole de transport, le Port Source et le Port de Destination. Une table de classification consiste, généralement, en une table de recherche qui exploite plusieurs stratégies de recherche (par exemple, la recherche de type « exact match »). Chaque règle a une action associée, qui sera effectuée si les champs utilisés pour la classification correspondent à cette règle. Une action consiste à un traitement appliqué au paquet.

La tâche de recherche (« lookup table») est une opération atomique impliquée dans d'autres tâches, comme la classification et la transmission de paquets. Puisque toutes les activités de traitement de paquets commencent par une recherche IP, le « lookup » est donc le traitement le plus fréquent et important dans le traitement de paquets. Étant donné son importance, le chapitre 3 sera dédié à la tâche de recherche (particulièrement, la recherche IP, « IP lookup »).

Lors de la phase de modification du paquet, le routeur peut supprimer ou modifier le paquet en cours de traitement, ou générer éventuellement de nouveaux paquets, selon l'application (une application de type « multicast », par exemple, duplique le paquet en cours de traitement). Certains

systèmes de traitement de paquets compressent et chiffrent les paquets avant qu'ils ne quittent le système, mais souvent les paquets transitent sans modification. Enfin, la transmission du paquet peut impliquer d'autres fonctions de mise en queue, de priorisation et de gestion de trafic pour s'assurer que le récepteur va recevoir le paquet selon le modèle de trafic attendu.

Afin de respecter les exigences fonctionnelles, différentes fonctions sont nécessaires [12] telles que :

- Le calcul de la somme de contrôle (« checksum ») : l'en-tête IP contient un « checksum » qui couvre les champs de l'en-tête pour s'assurer que ces champs importants n'ont pas été corrompus. Lors de la réception d'un paquet, tout système doit vérifier que cette somme de contrôle est toujours correcte.
- TTL (« Time-To-Live ») : il s'agit d'un compteur indiquant le nombre maximal de routeurs par lesquels le paquet peut transiter. Le compteur est décrémenté à chaque saut et le paquet est rejeté si ce compteur atteint la valeur zéro. Cette fonctionnalité garantit que les paquets qui ne peuvent atteindre leur destination seront rejetés pour ne pas traverser le réseau de manière indéterminée et infinie.
- Autres fonctions : un certain nombre d'autres opérations peuvent être effectuées (non standard, dépendamment de l'application). Un exemple de telles opérations est la fragmentation IP. Lorsque les datagrammes sont plus longs que ce que le réseau peut transférer directement, le datagramme doit être envoyé en plusieurs parties. Cela se produit lorsque le datagramme dépasse l'unité de transmission maximale (MTU). Une fois que le datagramme original a été fragmenté ou segmenté, il doit être réassemblé à la destination.

3.3 Plan de données, de contrôle et de gestion

Les réseaux informatiques peuvent être divisés en trois plans de fonctionnalités [13] : les plans de données, de contrôle et de gestion (voir Figure 2-1). Le plan de contrôle est responsable de l'exécution de protocoles de routage, tels que RIP, OSPF ou BGP, afin de peupler et de maintenir à jour les tables de transmission (« forwarding tables »).

Le plan de données est responsable de l'acheminement des paquets, conformément aux tables de routage, et de diverses fonctions de traitement de paquets telles que la mise en mémoire tampon des paquets (« packet buffering »), l'ordonnancement des paquets (« packet scheduling ») et la

modification des en-têtes. Le plan de gestion inclut les services logiciels utilisés pour surveiller et configurer à distance la fonctionnalité de contrôle, tel que le protocole SNMP. Les administrateurs réseau définissent la politique du réseau via le plan de gestion alors que le plan de contrôle l'applique et que le plan de données l'exécute en transmettant les données en conséquence.

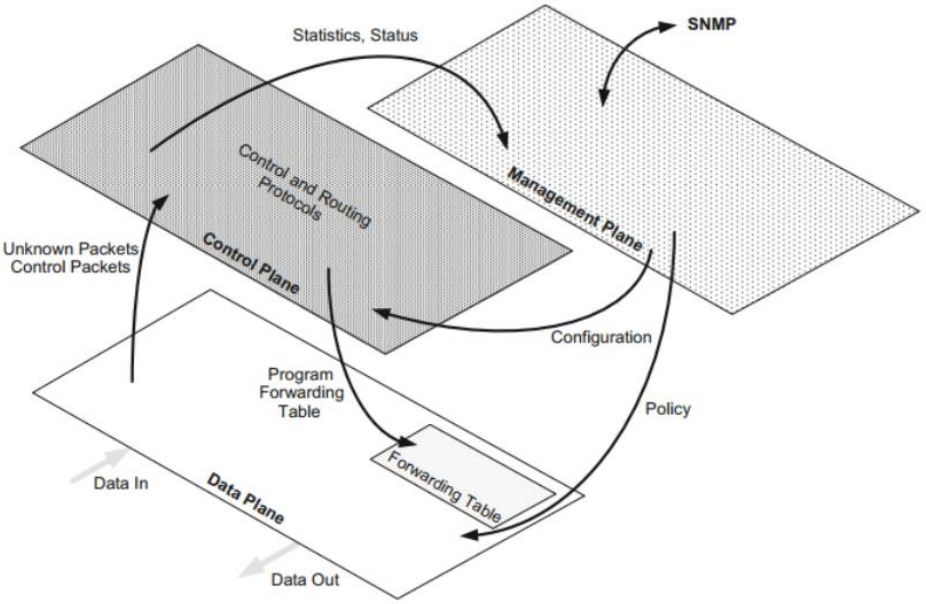

Figure 3-1 : Plan de données, de contrôle et de gestion [13]

Le plan de contrôle s'exécute généralement lorsque la topologie du réseau ou la stratégie du réseau change. Par conséquent, les fonctions de contrôle ne sont pas effectuées à l'arrivée de chaque paquet ; elles n'ont pas donc besoin d'autant de performance de la part du matériel. Cependant, les fonctions de « data plane » sont fréquentes, effectuées sur chaque paquet qui passe par le routeur, et nécessitent ainsi une implémentation performante.

Dans les réseaux IP traditionnels, les plans de contrôle et de données sont étroitement couplés et intégrés dans les mêmes dispositifs de réseau. Ce couplage était considéré comme important pour la conception d'Internet : c'était le meilleur moyen de garantir la résilience des réseaux [13], qui était l'objectif crucial de conception. Cependant, cela a rendu extrêmement difficiles diverses tâches de gestion du réseau [14], telles que déboguer les problèmes de configuration et prévoir ou contrôler le comportement de routage. C'est la raison fondamentale pour laquelle les réseaux traditionnels sont rigides et complexes à gérer et à contrôler, rendant l'innovation difficile. Au début des années 2000, le plan de contrôle des routeurs s'est vu donc séparé du plan de données.

Nous présentons, dans la section suivante, l'évolution des architectures des routeurs pour rencontrer les différents besoins de traitement de réseau. Nous discuterons également du concept SDN (« Software-Defined Networking »), qui est un modèle d'architecture réseau devenu un paradigme puissant et révolutionnaire. Avec SDN, il est possible de résoudre divers problèmes persistants dans le domaine du réseau en améliorant la séparation entre le plan de contrôle et le plan de données, permettant aussi d'introduire des technologies comme la virtualisation [15].

3.4 Architecture

3.4.1 Compromis entre performance et programmabilité

Historiquement, l'évolution des routeurs de réseau était principalement motivée par la performance des fonctions de traitement de paquets.

La performance d'un routeur peut être exprimées selon plusieurs dimensions [12] :

- Le débit de transmission : qui déterminent la quantité de données pouvant être transférée via le système par unité de temps.
- Le taux de paquets : étant donné que la plupart des opérations doivent être effectuées pour chaque paquet, les petits paquets entraînent une charge de travail plus importante pour un débit de données donné fixé.
- La consommation d'énergie électrique.

L'exigence de la vitesse de traitement de paquets peut être illustrée par un simple exemple : un routeur à haute vitesse actuel [1] est censé soutenir un débit de l'ordre du térabit par seconde ; supposons que ce dernier traite des paquets de 1000 bits et qu'il effectue 10 opérations par paquet, alors le routeur doit prendre en charge environ 10 milliards d'opérations par seconde.

En raison de la nécessité de mieux contrôler les opérations du réseau et de la demande constante de nouvelles fonctionnalités, la programmabilité des routeurs est devenue aussi importante que la performance. Elle peut être exprimée en fonction des algorithmes soutenus, en utilisant un modèle de programmation, et l'extensibilité. Cet équilibre entre la performance et la programmabilité représente un compromis majeur que rencontrent les opérateurs de réseau.

3.4.1.1 Routeurs logiciels

Initialement les routeurs étaient basés sur des processeurs à usage général (« General Purpose Processor », GPP), où les fonctions de réseaux étaient réalisées en logiciel. Ces derniers ont évolué au cours des années ; par exemple, en 2000, Click [16] utilisait un processeur avec un seul cœur. Au début des années 2000, Intel a lancé une gamme de processeurs spécifiques pour les applications réseaux, appelés processeurs de réseau, tels que le IXP2800 [17] en 2002. Il existe également des routeurs basés sur des processeurs graphiques (PacketShader [18], proposé en 2010) ou encore des circuits logiques programmables comme le FPGA dans l'implémentation présentée par NetFPGA- SUME [19] en 2014.

Cette évolution a permis aux routeurs logiciels d'être plus performant ; par exemple, les processeurs de réseaux bénéficient d'une performance intéressante grâce à l'utilisation de processeurs multicœurs, de modules matériels dédiés pour les opérations réseaux courantes (accélérateurs), d'interfaces mémoire à haute vitesse et d'interfaces d'entrée/sortie à haut débit. Par exemple, le processeur de réseau NP-5 [20] de Mellanox soutient un débit de 240 Gb/s. La figure 2-2 illustre l'architecture typique d'un processeur de réseau. Ce dernier possède une interface avec un « switch fabric », responsable de déplacer les données entrant dans un nœud du réseau vers le nœud suivant.

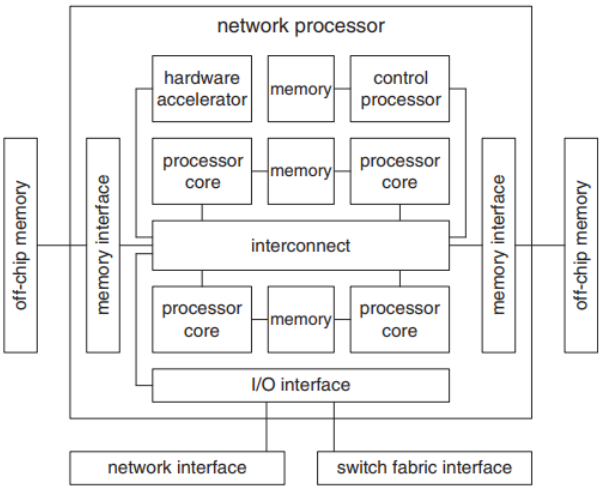

Figure 3-2 : Architecture typique d'un NP [12]

L'avantage principale des routeurs logiciels reste la flexibilité : l'ajout de nouvelles fonctionnalités se fait facilement, permettant ainsi aux routeurs de prendre en charge différents services (pare-feu,

détection d'intrusion, proxy …). D'ailleurs, la demande de soutenir différents services ne cesse de croître. Cependant, la croissance de la quantité de données à traiter, due à la croissance du débit des liens et des débits de transmission, exige des équipements plus performants. En effet, les routeurs logiciels ne sont pas en mesure de satisfaire les exigences de performance d'un routeur à haute vitesse (i.e. 1 Tb/s) [1].

3.4.1.2 Routeurs basés sur des ASIC

Comme mentionné précédemment, Le plan de contrôle des routeurs s'est vu séparé du plan de données. Utilisé moins fréquemment, le plan de contrôle est alors implémenté en logiciel, sur des processeurs à usage général pour être plus flexible ; alors que le plan de données est implémenté avec des entités matérielles plus performantes.

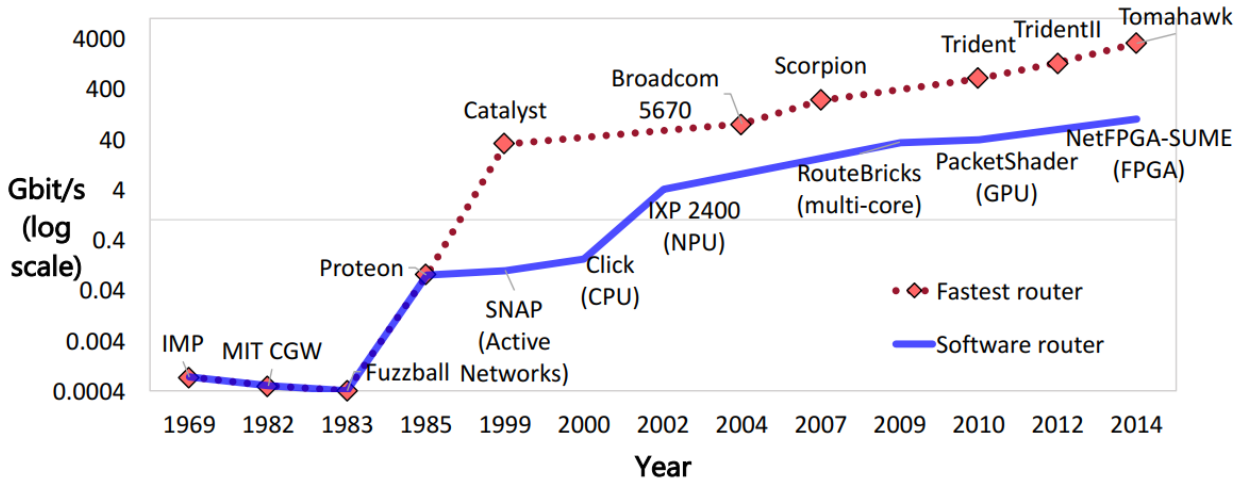

Figure 3-3 : Évolution de la performance des routeurs [1]

L'implémentation matérielle offre aux routeurs une amélioration de la performance (10 à 100 fois plus élevée par rapport aux routeurs logiciels, voir Figure 2-3) qui résulte de l'exploitation complète de l'abondant parallélisme disponible dans le traitement des paquets. La spécification matérielle permet de traiter simultanément différentes parties d'un même paquet (ou des paquets appartenant à des ports différents) et d'effectuer simultanément différentes opérations sur différents paquets.

Les ASIC, qui sont des circuits intégrés spécialisés, sont utilisés pour leur performance, mais ils possèdent un coût : comme les routeurs sont construits à partir de matériel spécialisé, ce sont des périphériques à fonction fixe qui ne peuvent normalement pas être programmés. Pour modifier une

fonctionnalité, l'opérateur de réseau doit attendre, de 2 à 3 ans, la prochaine génération de routeur, ce qui crée un décalage entre la standardisation et la disponibilité de nouveaux protocoles ou de certaines nouvelles applications. En conséquence, les routeurs à haute vitesse sont limités dans leur capacité à intégrer des nouveaux algorithmes. La figure 2-4 présente une chronologie des principaux algorithmes de routeur mis au point depuis les années 1980. Seule une poignée (représentée en bleu) est disponible dans les routeurs les plus rapides du moment.

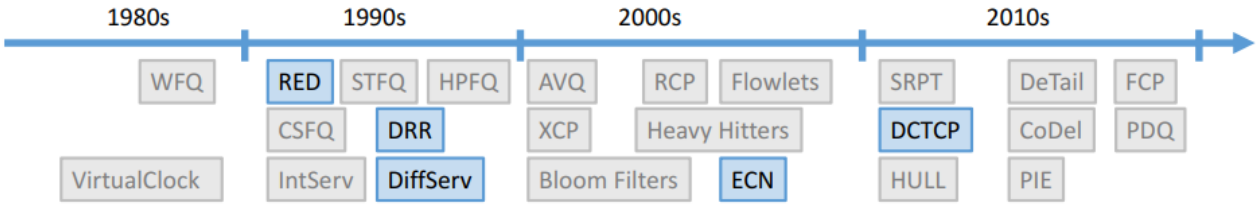

Figure 3-4 : Algorithmes de routeur [1]

3.4.2 Révolution SDN

Contrairement aux autres domaines de l'informatique, les réseaux informatiques sont difficiles à gérer et à faire évoluer. Pour prendre en charge de nouveaux services, les gestionnaires devaient configurer des milliers de périphériques, augmentant considérablement la complexité du réseau [22]. Puisque les équipements réseaux diffèrent d'un fournisseur à l'autre, seul un petit nombre d'interfaces externes est normalisé (par exemple, la transmission de paquets), ce qui limite la capacité des opérateurs de réseaux à adapter les réseaux à leurs environnements individuels et à améliorer leur matériel ou leurs logiciels.

Le plus gros inconvénient des approches traditionnelles est le faible niveau d'abstraction [15]. La réduction de la complexité des architectures exige certainement un niveau d'abstraction plus élevé. L'abstraction ne supprime pas toute la complexité du réseau, mais elle donne la possibilité d'encapsuler ou de masquer un certain degré de complexité. De plus, l'abstraction est l'une des conditions essentielles d'une conception modulaire efficace : elle offre la souplesse nécessaire pour effectuer le traitement dans différentes parties du réseau. Même si l'abstraction du plan de données est basée sur les couches du modèle TCP/IP (présenté précédemment), les interfaces sont mal implémentées et violent les principes de la modularité [15]. Ce constat a poussé les chercheurs et les industriels à penser le traitement du réseau différemment, faisant apparaître un nouveau paradigme, qui a défini les abstractions dont avait besoin le réseau, le SDN.

Le SDN repose sur plusieurs travaux antérieurs, notamment ceux qui avaient pour but la séparation entre le plan de contrôle et le plan de données, l'apprivoisement des réseaux de centre de données et la réorganisation de la gestion des réseaux. Le SDN permet aux opérateurs de programmer le plan de contrôle du réseau, en introduisant l'idée du contrôle centralisé : en déplaçant le plan de contrôle hors des équipements réseaux, le plan de contrôle entier pour le réseau pourrait être centralisé sur quelques serveurs (exécuté sur des processeurs performants). Ceci permet à ces serveurs de calculer des itinéraires pour l'ensemble du réseau avec l'avantage de la visibilité globale du réseau. Pour ce faire, le SDN avait besoin d'un mécanisme permettant au plan de contrôle d'alimenter le contenu des tables de routage, une fois que le plan de contrôle a calculé les itinéraires pour chaque routeur. Ces tables sont ensuite consultées par le plan de données lors de la transmission des paquets. Le plus connu de ces mécanismes était l'API OpenFlow, qui a exposé une interface entre le plan de contrôle et le plan de données.

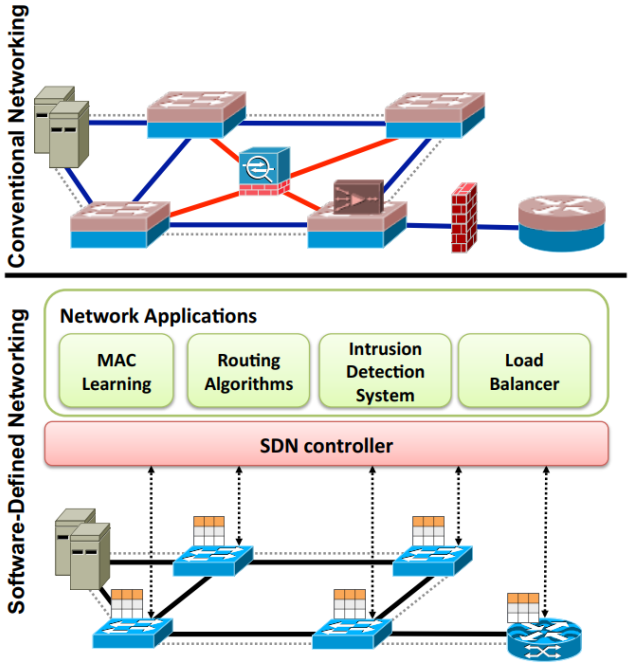

Figure 3-5 : Architecture SDN [13]

La figure 2-5 illustre la différence architecturale entre les réseaux traditionnels et les réseaux SDN. Avec SDN, la gestion devient plus simple. À noter que, puisque le plan de données est responsable principalement de transiter des paquets vers les interfaces appropriées, il est plus juste d'appeler la plateforme qui l'implémente un commutateur (« switch »); par exemple, un commutateur ASIC. Cependant, le terme routeur est souvent utilisé par abus de langage.

3.4.2.1 Traitement basé sur le flux

Avec SDN, le traitement du plan de données devient basé sur les flots (« flow-based »). Un flot est défini de manière générale par un ensemble de valeurs de champs de l'en-tête des paquets agissant comme critère de correspondance et auquel on associe un ensemble d'actions.

Dans le cas d'une implémentation OpenFlow, un commutateur a une ou plusieurs tables de flots, représentant les règles de transmission de paquets. Chaque règle est associée à une action spécifique que le routeur doit exécuter sur un paquet entrant si des champs d'en-tête particuliers du paquet correspondent à cette entrée (par exemple, 15 champs, allant de la couche 2 à la couche 4, pour un routeur supportant la version 1.1.0 d'OpenFlow). Par exemple, une règle pourrait indiquer au routeur de transmettre un paquet sur un port de sortie particulier (action) si le paquet a une certaine adresse IP de destination (correspondance). Le plan de données est chargé de lire les tables de correspondance et d'exécuter l'action appropriée sur chaque paquet (si le paquet correspond à une règle particulière). L'action [14] à exécuter peut consister, par exemple, à supprimer le paquet, le transmettre à une interface particulière, à modifier un champ d'en-tête etc. Ces tables de correspondance contiennent également un ensemble de compteurs (pour calculer le nombre d'octets et de paquets) et un mécanisme de priorité (pour lever toute ambiguïté si deux règles venaient à correspondre à un même paquet).

L'abstraction de flot permet d'unifier le comportement de différents types de périphériques réseau où tous les paquets d'un flot reçoivent un traitement identique.

3.4.2.2 Fonctionnalités clés de l'architecture SDN

Le SDN présente une abstraction pour la définition d'une architecture pour le plan de contrôle centralisé, pour découpler la topologie, le trafic et les dépendances entre les couches, pour avoir un réseau multicouche dynamique etc.

Mais surtout, SDN et OpenFlow ont révolutionné la manière dont on traite le réseau, en s'appuyant sur l'approche dite de « Match-Action » [24], qui consiste à exécuter une action en fonction de l'entrée des tables de correspondance, qui convient au paquet entrant. OpenFlow propose différents types d'instruction comme appliquer, effacer et écrire des actions, permettant d'avoir une flexibilité sans précédent (limitée néanmoins) pour les routeurs basés sur les ASIC (dans un routeur statique, toutes les tables de flots sont remplies manuellement).

3.4.3 Travaux de recherche courants

Comme vu précédemment, le seul moyen de satisfaire les besoins de performance élevée est d'utiliser un routeur basé sur un ASIC. La figure 2-6 représente un routeur typique de centre de données. Dans ces routeurs, les tables de flux sont souvent implémentées à partir de TCAM et SRAM, qui sont des mémoires rapides mais de capacité limitée (on discutera plus en détails de ces mémoires dans le chapitre 4). Le plan de contrôle, quant à lui, s'exécute sur une carte CPU, qui roule un micro-serveur. En plus de ces composants standards (RAM, support de stockage, etc.), une carte CPU dispose d'une interconnexion PCI-E au commutateur ASIC et la présence d'un processeur x86 permet l'installation de Linux, offrant les fonctionnalités générales d'un système d'exploitation.

Puisque l'implémentation ASIC est requise pour obtenir une grande performance, beaucoup de travaux de recherche se sont focalisé à rendre ces routeurs plus programmables.

3.4.3.1 Évolution d'OpenFlow

Initialement, avec OpenFlow, l'ensemble des champs sur lesquels des correspondances pouvaient être effectuées était fixe et limité à un petit nombre. De même, les actions autorisées sur les en-têtes de paquet étaient limitées à un ensemble fixe d'actions. Les routeurs les plus récents (également les plus performants) utilisent un traitement de type « match-action » plus souple. Dans ces routeurs, un « parser programmable » [1] permet à l'utilisateur de spécifier un nouveau format de protocole car l'utilisateur est capable de spécifier les champs à extraire du paquet sans être contraint à un ensemble fixe de champs disponibles à des emplacements fixes dans l'en-tête du paquet. Également, pour surmonter la limite de la capacité mémoire des TCAM (en raison de leur coût), le traitement est fait à l'aide de plusieurs tables de flots en pipeline.

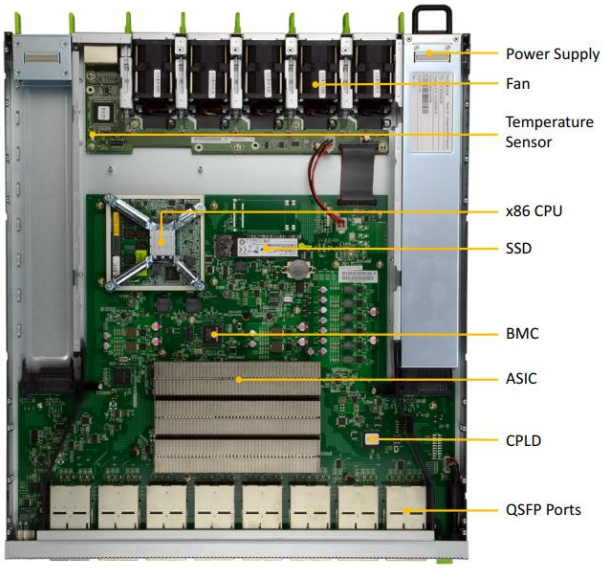

Figure 3-6: Routeur ASIC [21]

3.4.3.2 Routeur “programmable”

La communauté SDN a consacré beaucoup de travail de recherche à la programmation du plan de contrôle et elle cible aujourd’hui la programmation dans le plan de données. De ce fait, des travaux universitaires et industriels ont abouti au développement de routeurs dits programmables [1]. Ces routeurs ont un jeu d’instructions minimal qui permet à un opérateur de réseau d’exprimer des règles de transmission de type OpenFlow pour n’importe quel protocole. Cela contraste avec OpenFlow, qui ne prend en charge qu’un ensemble fixe d’actions (suppression, décrémentation, durée de vie, transfert, etc.)

Le langage de programmation P4 a émergé d’un effort de l’industrie et du milieu académique visant à mettre au point un langage de programmation standard pour ces routeurs. Les compilateurs P4 visent des architectures de routeurs “programmable” telles que l’architecture RMT et FlexPipe. P4 [25] permet de programmer le plan de données et les programmeurs utilisent des abstractions spécifiques à un domaine (« domain-specific abstractions »), et faisant partie du langage, pour mettre en œuvre diverses applications innovantes : notamment les outils de diagnostic réseau et de télémétrie, des systèmes d’équilibrage de charge (« load balancing »), et même des protocoles de consensus optimisés.

L’architecture RMT [24] (« Reconfigurable Match Tables ») permet la définition d’en-têtes et de séquences d’en-tête arbitraires, une correspondance arbitraire (de champs) par un nombre arbitraire

de tables (règles), une écriture arbitraire de champs d'en-tête de paquet (mais pas le corps du paquet) et une mise à jour de l'état du paquet, permettant ainsi de soutenir de nouveaux protocoles tels que PBB, VxLAN, NVGRE, STT et OTV [24].

Le Tofino [26] (produit de Barefoot Networks, en décembre 2016) est un des fruits de ces travaux. Il est présenté comme le commutateur programmable le plus rapide au monde, supportant 11 instructions et une vitesse atteignant les 6.5 Tb/s. PISA [26] (« Protocol Independent Switch Architecture ») est l'architecture du Tofino et elle découle du concept de RMT.

3.4.3.3 Limites des routeurs ASIC

3.4.3.3.1 Flexibilité limitée

Beaucoup d'efforts et de travaux ont été réalisés afin de rendre les ASIC plus programmables. Cependant cela est encore insuffisant pour exprimer certains des algorithmes illustrés à la figure 2-4. En effet, les auteurs du papier introduisant l'architecture RMT [24] ont défini des restrictions de reconfigurabilité nécessaires afin de soutenir une vitesse de l'ordre du térabit par seconde, comme des restrictions matérielles (par exemple, le nombre d'étages de pipeline), la taille maximale d'un paquet pouvant être traité mais surtout des restrictions au niveau des actions possibles. Les instructions sont limitées à des opérations arithmétiques et logiques simples, permettant l'implémentation de certains protocoles, mais n'autorisant pas le cryptage des paquets ou le traitement des expressions régulières, par exemple. Également, ces instructions ne sont pas en mesure d'implémenter la fonctionnalité d'une machine à état, ce qui constitue un handicap quand il s'agit d'offrir de multiples algorithmes.

3.4.3.3.2 Capacité mémoire limitée

Une autre limitation des ASIC (dans le domaine des réseaux) est la capacité mémoire. En effet, les exigences de performance nécessitent l'utilisation de mémoire TCAM ou SRAM pour implémenter les tables de correspondance (« match tables »), qui sont des mémoires rapides mais coûteuses. Puisque ces mémoires sont intégrées sur puce (« on-chip »), elles sont limitées à quelques dizaines de mégaoctets [27]. Par ailleurs, les tables de flots peuvent contenir la description des millions de flots [28] ce qui dépasse la capacité de ces mémoires.

3.5 Conclusion

Une mise en œuvre matérielle est le seul moyen d'atteindre les requis de performance pour les routeurs à haut débit. Néanmoins, certains scénarios ont des exigences en calculs élevées mais pour des débits plus faibles, par exemple, pour l'implémentation des algorithmes de couche MAC en Wifi et des algorithmes de traitement du signal dans la couche physique sans fil[1].

Dans le but de répondre au mieux aux besoins des opérateurs de réseau, les routeurs ont été divisés en deux classes [1] : les routeurs de périphérie (« edge routers »), qui sont localisés à l'entrée ou à la périphérie d'un réseau et les routeurs qui sont localisés au centre du réseau (« core routers »). Les « core routers » ont pour rôle la transmission de paquets au cœur du réseau alors que les « edge routers » gèrent uniquement une petite portion du trafic global du réseau, puisqu'ils sont spatialement distribués. Les « edge routers » sont responsables de la redistribution et les agrégations des routes et de l'échange d'information de routage. En conséquence, les « edge routers » ont des exigences de performances beaucoup moins strictes que pour les « core routers ». Les routeurs logiciels sont plus appropriés dans le contexte où la programmabilité prime sur la performance. Aussi, l'augmentation de la puissance de traitement des processeurs a rendu les routeurs logiciels plus attrayants [23] pour les chercheurs et les entreprises.

Les commutateurs logiciels apparaissent comme l'une des solutions les plus prometteuses pour les centres de données et les infrastructures de réseau virtualisées [13]. Switch Light, ofsoftswitch13, Open vSwitch, OpenFlow Reference, Pica8, Pantou et XorPlus sont des exemples [13] d'implémentation de commutateurs logiciels supportant OpenFlow. De ce fait, mon projet de recherche s'est focalisé sur les processeurs de réseau, qui présentent un bon compromis entre la performance des ASIC et la flexibilité des GPP.

Le choix des processeurs de réseau n'est pas anodin. D'une part, leur flexibilité permet naturellement de supporter SDN [29] et, d'autre part, nous pensons que les processeurs de réseau peuvent étendre la capacité mémoire des commutateurs ASIC. En effet, une solution hybride pourrait consister à interfacer un ASIC avec un processeur réseau : le ASIC contiendrait des petites tables de flux contenant la description des flux qui exigent un traitement à haute vitesse, alors que le processeur réseau contiendrait des tables plus volumineuses mais nécessitant un traitement moins rapide. Il s'agit donc d'un système basé sur la mise en cache (garder les données les plus fréquemment utilisées proches de l'unité de traitement principale) et qui est cohérent avec notre

époque où le trafic vidéo, dont la demande est amplifiée par des plateformes comme Netflix et YouTube, qui représente une partie significative et croissante du trafic Internet (58 % en 2018 [30]). La localité temporelle qui découle de la transmission de données structurées en flots, suggère de développer des solutions basées sur la mise en cache. Le choix d'orienter notre projet de recherche vers les processeurs de réseau au lieu des FPGA (les FPGA modernes, tels que le Xilinx Virtex-7 [24], peuvent transférer le trafic à près de 1 Tb/s) est justifié par le fait que les FPGA consomment plus d'énergie et coûtent beaucoup plus cher.

CHAPITRE 4 RECHERCHE IP

4.1 La recherche (« lookup »)

Comme vu précédemment, le plan de données SDN repose sur le paradigme « match-action » dans lequel les programmeurs peuvent spécifier un flot via une règle de correspondance de champs d'en-tête, ainsi que des actions de traitement appliquées aux paquets correspondants. Le « lookup » consiste à faire une recherche dans une table de correspondance (« lookup table »), contenant une liste d'association de valeurs, à partir d'une clé de recherche pour avoir la valeur associée à cette clé, représentant le résultat de la recherche. Puisque chaque paquet est soumis à des opérations de recherche, la recherche de correspondance représente ainsi l'une des opérations les plus importantes dans le traitement des paquets, où les modules de recherche (« search engine ») doivent faire face à des exigences de performance pouvant nécessiter des millions d'opérations de recherche par seconde [31].

Le « lookup » peut paraître simple à première vue (retourner une valeur à partir d'une clé), mais il est en réalité complexe. La recherche doit se faire sur un nombre d'entrées important et en constante croissante, par exemple, la transmission de paquet repose sur la recherche d'adresse IP où une adresse IP doit correspondre à une entrée de la table de routage du routeur qui peut contenir des millions d'entrées [3] [32]. Aussi due à la croissance d'Internet et du nombre de réseaux IP, la taille des tables de routage ne cesse d'augmenter (au niveau des « edge routers », leur taille augmente au rythme de 25 à 50 000 entrées par an [31]). De plus, toujours dans le contexte de la transmission de paquets, plusieurs entrées peuvent correspondre à la clé de recherche et l'entrée avec la plus grande priorité doit être retournée. Le critère de priorisation sur lequel repose la transmission de paquet est le LPM. L'adressage IP et la transmission de paquets ont été brièvement introduits dans le chapitre 1 mais nous allons apporter plus de détails au cours du présent chapitre.

Il est important de noter que le terme table de routage est souvent utilisé pour désigner la table de recherche sur laquelle repose la transmission des paquets mais en réalité on utilise la table FIB. Elle conserve les informations d'adresse des interfaces selon les informations contenues dans la table de routage IP.

La vitesse est le facteur le plus important dans une solution de recherche (module de recherche). Cependant, la mise à jour des données (insérer, modifier ou supprimer une entrée) revêt également

une importance de performance capitale, de même que la taille de la structure de données et des mémoires utilisées pour stocker toutes les entrées.

Différents types de recherche sont nécessaire en fonction des besoins. La clé de recherche peut être utilisée entièrement ou partiellement. Le « Full Matching » signifie que la totalité de la clé de recherche doit correspondre aux données requises alors que « Partial Match » signifie que la correspondance partielle de la clé de recherchée suffit pour considérer qu'il y a correspondance avec les données. Les données de la table de correspondance peuvent être également utilisées totalement ou partiellement. Le LPM, la correspondance exacte (« exact match ») et la correspondance ternaire (« ternary match ») représentent les principaux types de recherche effectués dans le domaine du réseau [33]. L'« exact match » requiert que la totalité de la clé de recherche corresponde à la totalité d'une donnée de la table de recherche alors que le « ternary match » permet d'émuler une variété de types de correspondance car il permet d'effectuer des recherches masquées où un masque est appliqué à la clé de recherche de façon à ce que certain bits ne soient pas pris en compte.

Ce chapitre est consacré au LPM, qui est un facteur de performance de poids dans la transmission de paquets et qui est aussi le sujet majeur de mon travail de recherche. Nous expliquerons comment la transmission de paquets s'effectue grâce à la recherche LPM, ainsi que les exigences et les défis qu'elle présente.

4.2 LPM

4.2.1 Recherche IP —Contexte

Le LPM est un critère de priorisation sur lequel repose la recherche d'adresse IP (« IP address lookup »), connue aussi comme la recherche IP (« IP lookup »), pour la transmission de paquets.

La recherche IP est requise pour la classification, l'attribution de priorités (à des fins de qualité de service), les tâches de gestion de réseau (à des fins de sécurité), etc. Mais surtout, la recherche IP est nécessaire pour effectuer une décision de transmission, à savoir identifier les informations du prochain saut (NHI). Presque chaque activité de traitement de paquets [9] commence par une recherche IP et puisque l'exigence de vitesse de traitement des routeurs ne fait que croitre, la vitesse de la recherche IP est essentielle.

En effet, la croissance d'Internet a été accompagnée par une croissance explosive de la bande passante des liens de transmission. Après le déploiement du DWDM, technique utilisée en communication optique, la vitesse de liaison a doublé tous les sept mois [34]. Aujourd'hui, la norme Ethernet 100 gigabits (IEEE P802.3ba) a été adoptée et la norme Ethernet 400 gigabits (IEEE P802.3bs) est visée. La croissance de la bande passante des liens de transmission conduit les routeurs à traiter les paquets de plus en plus vite et, puisque le traitement le plus récurent qu'effectue un routeur est l'acheminement de paquet, la recherche IP, et particulièrement le LPM, devient le goulot d'étranglement de la transmission de paquets. Par exemple, une liaison de 100 Gbps nécessite un débit de plus de 195 millions de recherche par seconde pour des paquets IPv6 de taille 64 octets.

Afin de comprendre le LPM, nous allons décrire dans la section suivante le principe de l'adressage IP et son fonctionnement.

4.2.2 Adressage IPv6, routage et transmission de paquets

La recherche d'adresse IP est naturellement basée sur l'adresse IP, c'est-à-dire une clé de 32 bits ou 128 bits selon qu'on utilise IPv4 ou IPv6 respectivement. IPv6 est né du besoin d'étendre l'espace d'adressage, ce qui est nécessaire étant donné la croissance du nombre de réseaux IP, même si l'épuisement de l'espace d'adressage IPv4 a été compensé par l'utilisation des services tels que NAT et DHCP. Rappelons que le NAT permet de faire correspondre une seule adresse IP publique et routable à toutes les adresses d'un réseau privé (non routable) alors que le DHCP configure automatiquement des paramètres IP d'une station ou d'une machine, en attribuant une adresse IP limitée dans le temps (cela permet d'avoir des adresses tournantes, permettant au réseau d'accueillir plus de stations si ces dernières ne sont pas connectées simultanément). En raison du besoin de transiter vers IPv6, nous nous sommes focalisés sur cette version du protocole IP dans cet ouvrage. Nous présenterons, dans la section suivante, les principales fonctionnalités de même que nous décrirons le format de l'entête et de l'adresse IPv6 ainsi que la manière dont la transmission de paquets est faite. Notons que beaucoup des fonctionnalités d'IPv6 ont été rétroportées dans IPv4 et qu'IPv6 utilise des concepts introduits pour IPv4 comme l'adresse sans classes (CIDR).

En offrant un espace d'adressage de l'ordre du sextillion ($2^{128} \approx 340$ sextillions d'adresses **publiques**), IPv6 est en mesure de répondre aux besoins futurs comme ceux de l'internet des objets

(IoT) qui impliquent que le nombre d'appareils connectés à Internet et qui doivent être clairement identifiés augmentera de manière significative [35] dans les prochaines années.

Aujourd'hui [11], la pression pour la transition d'IPv4 à IPv6 provient principalement des fournisseurs de services et opérateurs réseau et d'autres groupes dotés de grands réseaux internes tels que les opérateurs de réseaux de téléphonie cellulaire.

4.2.2.1 En-tête IPv6

Avec IPv6, la taille de l'en-tête IP est passée de 24 octets (192 bits) à 40 octets (320 bits). Cependant, l'en-tête IPv6 est beaucoup plus simple que l'en-tête IPv4 (mis à part les champs d'adresse, l'en-tête est en réalité plus petit que dans IPv4.). La figure 3-1 présente l'en-tête IPv6.

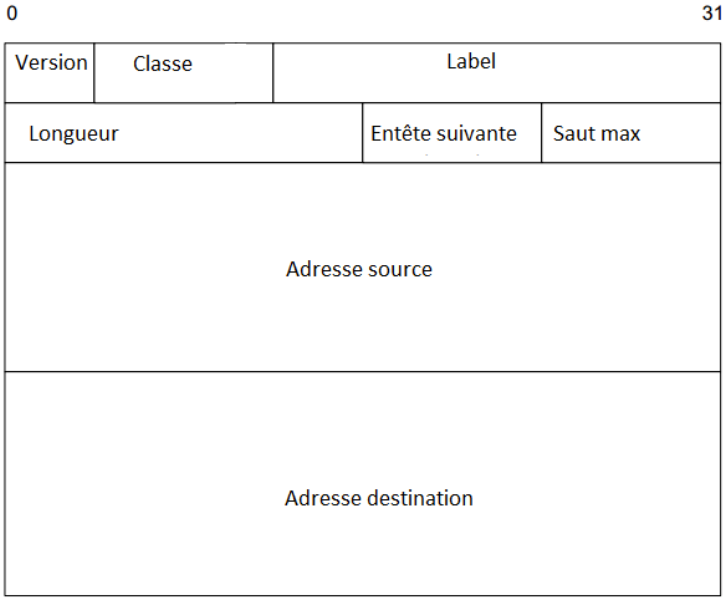

Figure 4-1 : En-tête IPv6 [9]

Les champs de l'en-tête sont décrits comme suit [36] :

- Version : ce champ est codé sur 4 bits et il donne le numéro de version du Protocole Internet.
- Classe (« Traffic Class ») : est codé sur 8 bits et définit la priorité du paquet pour des fins de qualité de service.
- Label (« Flow Label ») : est codé sur 20 bits et il est utilisé pour indiquer les instructions de traitement d'un flot de paquets (par exemple pour un service temps réel).

- Longueur (« Payload Length ») : indique le nombre d'octet des données qui suivent l'entête IPv6 dans le paquet.
- Entête suivante (« Next Header ») : est un champ de 8 bits indiquant le type de l'en-tête qui succède l'entête IPv6, par exemple la valeur 06 correspond à un en-tête TCP alors que la valeur 17 correspond à un en-tête UDP.
- Saut maximum (« Hop Limit ») : est codé sur 8 bits et indique le nombre de routeur maximum que le paquet peut traverser, en effet, la valeur de ce champ est décrémentée de 1 à chaque saut du paquet pour éviter d'avoir des boucles dans une situation de routage où le paquet n'arrive pas à atteindre sa destination et on rejette le paquet si la valeur du champ « Saut maximum » atteint zéro (ce champ est identique au champ « TTL » de l'en-tête IPv4).
- Adresse source (« Source Address ») : représente l'adresse IP de l'émetteur.
- Adresse destination (« Destination Address ») : représente l'adresse IP du destinataire.

Il est possible de supporter un ou plusieurs en-têtes d'extension, vus comme un prolongement de l'en-tête d'IPv6, afin de fournir des informations complémentaires et des fonctionnalités spéciales comme l'en-tête de routage qui permet de spécifier un chemin déterminé à suivre.

4.2.2.2 Fonctionnalités IPv6

Mis à part l'extension de l'espace d'adressage, IPv6 offre également un certain nombre de fonctions qui permettent de pallier certaines limites d'IPv4. Avec le format d'en-tête IPv6, certains champs d'en-tête IPv4 ne sont plus utilisés tandis que d'autres deviennent facultatifs. Cette simplification permet de maintenir les coûts de bande passante de l'en-tête IPv6 aussi bas que possible, malgré l'augmentation de la taille de l'adresse.

Avec IPv6, on pourrait également se passer de NAT car chaque périphérique aurait une adresse unique au monde, offrant une **connexion de bout en bout réelle**. En effet, l'emploi des NAT a conduit à une complexification [36] de la gestion des réseaux et à un alourdissement des mécanismes de routage. Cela nuit également au développement d'application « Peer to Peer » en temps réel comme la VoIP.

Aussi, IPv6 permet d'améliorer la capacité d'Internet dans divers domaines comme la sécurité avec l'intégration d'IPSec qui est un ensemble de protocoles permettant des communications privées et protégées sur des réseaux IP, par l'utilisation des services de sécurité cryptographiques. L'implémentation d'IPSec est notamment possible grâce à la possibilité du chiffrement de bout en bout. De plus, les périphériques IPv6 peuvent se configurer automatiquement via le protocole « Neighbor Discovery » qui est responsable entre autres de la découverte des adresses L2 (MAC) des nœuds et des routeurs voisins, fournissant ainsi des services similaires à ARP et ICMP pour IPv4. Aussi, le protocole « Neighbor Discovery » fournit certaines améliorations comme le « Neighbor Unreachability Detection » (NUD) qui permet de détecter des systèmes inaccessibles.

Finalement, les champs « Label » et « Class » permettent à IPv6 de disposer d'un mécanisme intégré pour la sécurisation de la qualité des services.

4.2.2.3 Adresse IPv6

Une adresse IPv6 a une taille de 128 bits. Elle est décrite grâce à huit champs de 16 bits, exprimés en hexadécimal et délimité par deux-points (:). Par exemple [37]: 2001:0db8:0000:85a3:0000:0000:ac1f:8001. Il est permis d'omettre de 1 à 3 zéros non significatifs dans chaque champ de 16 bits. Ainsi, l'adresse IPv6 ci-dessus est équivalente à : 2001:db8:0:85a3:0:0:ac1f:8001. De plus, une unique suite d'un ou plusieurs groupes consécutifs de 16 bits valant zéro peut être omise (en conservant toutefois les signes deux-points). Ainsi, l'adresse IPv6 ci-dessus peut être abrégée en : 2001:db8:0:85a3::ac1f:8001.

4.2.2.3.1 Le routage inter-domaines sans classe (CIDR)

Les réseaux IP sont identifiés en utilisant la notation CIDR. Le routage inter-domaine sans classe a été introduit pour IPv4 et permet de créer une hiérarchie d'adresses réseau en définissant des sous-réseaux (un sous-réseau représente une subdivision logique d'un réseau de taille plus importante) avec des préfixes.

Un préfixe réseau, ou sous-réseau, est représenté par la première adresse du réseau suivie par une barre oblique « / » et de la longueur du préfixe, qui est comprise entre 0 et 128. La longueur du préfixe définit le nombre de bits de poids fort de l'adresse IP (la première adresse du réseau) devant être considérés comme identifiant du réseau, représentant ainsi la partie commune des adresses

déterminées par ledit réseau. La liste suivante résume quelques points clés sur la façon d'écrire des préfixes IPv6 :

- Le préfixe a la même valeur que les *n* bits de poids forts des adresses IP du réseau, où *n* représente la longueur du préfixe.
- Tous les bits qui suivent le nombre de bits de la longueur du préfixe sont des 0s binaires.
- Le préfixe peut être abrégé avec les mêmes règles que les adresses IPv6.

La notation 2001:0820:9511::/48 correspond par exemple à un sous-réseau qui comprend les adresses de :

2001:0820:9511:0000:0000:0000:0000:0000 à 2001:0820:9511:FFFF:FFFF:FFFF:FFFF:FFFF.

4.2.2.3.2 Types d'adresses IPv6

Il existe trois principaux types d'adresses IPv6 :

- Les adresses d'envoi individuel (monodiffusion, « unicast ») : ce type d'adresse IP est utilisé pour identifier une seule interface réseau (une source ou une destination du réseau) de manière unique. Il existe plusieurs types d'adresses « unicast », on distingue :
    - Les adresses internes : ces adresses sont utilisées au sein d'un réseau local et ne sont pas routable sur internet. On distingue deux types d'adresses internes : les adresses lien-local (« Link local ») dont le préfixe est FE80::/10 et les adresses unique-local (« Unique Local ») dont le préfixe est FC00::/7. Les adresses unique-local sont routables en interne.
    - Les adresses « unicast » globales : une adresse « unicast » global est unique au monde sur Internet ; il s'agit donc d'une adresse routable. Ces adresses représentent 1/8e de l'espace d'adressage total d'IPv6, comprenant les adresses avec le préfixe 2000::/3 qui commencent par :

      2000:: jusqu'à 3FFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF.
- Les adresses de multidiffusion (« multicast ») : ce type d'adresse IP est utilisé pour identifier un groupe d'interfaces (relation un-à-N), en règle générale sur des nœuds différents. Ainsi, les paquets sont dupliqués par le routeur pour que ces derniers atteignent

les différentes destinations qui font partie du groupe « multicast ». Le préfixe FF00::/8 est appliqué pour les adresses « multicast ».

- Les adresses d'envoi à la cantonade (« anycast ») : ce type d'adresse IP est utilisé pour identifier un ensemble d'interfaces défini, généralement, sur des périphériques différents. Les adresses « anycast » sont utilisées pour livrer des paquets à l'interface la plus proche selon la politique de routage (contrairement aux adresses multicast où les paquets de données sont envoyés à tous les membres du groupe de diffusion). Les adresses « anycast » sont principalement utilisées pour permettre une répartition de charge et pour des raisons de sécurité.

IPv6 n'utilise pas la méthode « broadcast » (utilisé dans IPv4 pour diffuser les données à toutes les machines d'un réseau). En IPv6, les adresses de multidiffusion ont la même fonction que les adresses broadcast en IPv4.

Le type d'une adresse IPv6 est identifié par les bits de poids fort de l'adresse (déterminé par son préfixe), la figure 3-2 représente les types d'adresses IPv6. Il n'y a pas de préfixe spécial pour une adresse « anycast IPv6 », elle utilise la même plage d'adresses que les adresses unicast globales.

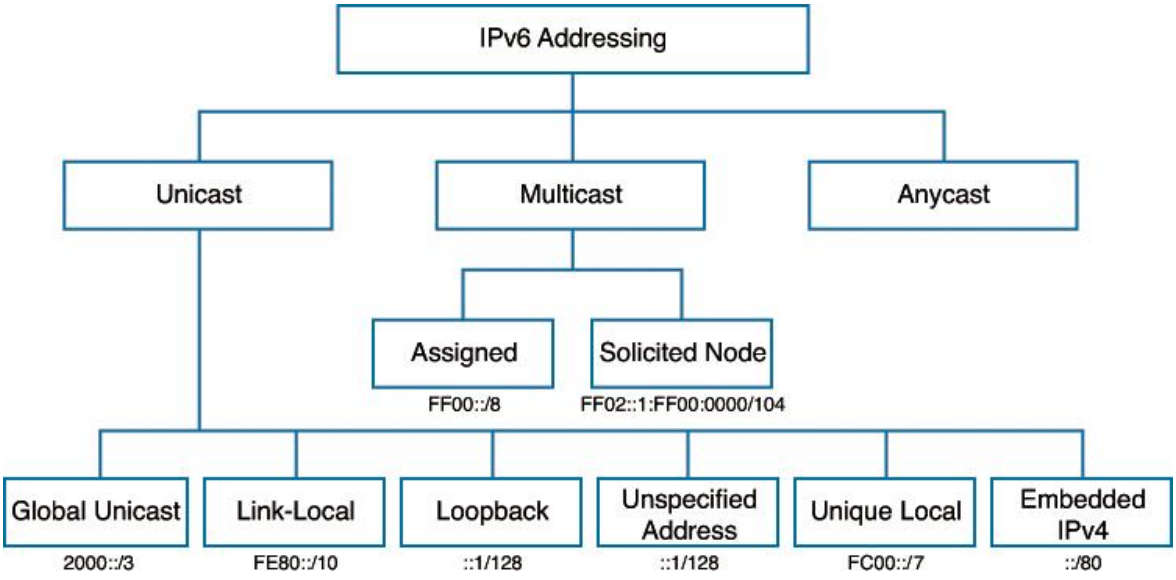

Figure 4-2 : Types d'adresse IPv6

Dans le contexte de la recherche IP pour la transmission de paquets (recherche LPM), nous nous sommes intéressés aux adresses « unicast » globales et « anycast » car elles sont routables sur Internet. Les adresses IP Unicast sont assignées par l'IANA aux registres Internet régionaux (RIR).

L'IANA est le département de l'ICANN, qui est l'autorité de régulation de l'Internet, responsable de superviser l'allocation globale des adresses IP et des numéros de systèmes autonomes (AS, dont on a parlé dans le chapitre 1). Les RIR, quant à eux, gèrent les ressources d'adressage IPv4 et IPv6 dans leur région (chaque registre est attaché à une zone géographique, par exemple, le RIPE-NCC pour l'Europe et le Moyen-Orient). L'IANA alloue des blocs de taille /23 à /12 dans l'espace unicast global (2000::/3) aux cinq RIR. Ces derniers les allouent à leur tour aux fournisseurs d'accès à internet (LIR) sous forme de blocs de taille minimale de /32. Les RIR peuvent choisir de subdiviser leur bloc de taille /23 en 512 blocs de taille /32 ($2^{32-23} = 512$), typiquement un par LIR. Le LIR peut à son tour assigner 65536 blocs de taille /48 ($2^{48-32} = 65536$) aux sites Internet (topologie publique), qui disposent alors chacun de 65536 sous-réseaux LAN (topologie privé) de taille /64 ($2^{64-48} = 65536$).

Les 128 bits d'une adresse IP spécifient l'hôte sur le sous-réseau et le préfixe identifie le sous-réseau. Le préfixe de pour les adresses « unicast » et « anycast » est contenu dans les 64 bits les plus significatifs de l'adresse, alors que les 64 bits les moins significatifs de l'adresse représentent l'identificateur d'interface réseau (« interface identifier », IDD). Les hôtes d'un même sous-réseau doivent avoir le même préfixe (partie réseau) IP.

La RFC 4291 [38] indique que toutes les adresses « unicast » globales doivent avoir une taille d'identificateur d'interface (IID) égale à 64 bits. L'adresse « unicast » établit ainsi une relation un-à-un entre l'adresse de sous-réseau et l'extrémité du réseau (interface). L'ID d'interface est construite à partir de l'adresse MAC de l'interface réseau (serveur DHCPv6), dans un format nommé EUI-64.

Le tableau 3-1 présente le format d'une adresse « unicast » globale. La taille du préfixe de routage peut varier; une taille de préfixe plus grande signifie une taille d'ID de sous-réseau plus petite. Les bits du champ ID de sous-réseau sont disponibles pour l'administrateur du réseau afin de définir les sous-réseaux au sein du réseau donné alors que le préfixe de routage est attribué par un fournisseur de services Internet.

Tableau 4-1 : Format d'une adresse « unicast » globale

Préfixe de sous-réseau		ID d'interface
Préfixe de routage / Topologie publique	Identifiant de sous-réseau / Topologie privée	
64 bits		64 bits

Les adresses « anycast » ont la même structure que les adresses « unicast » (préfixe de sous-réseau + ID « anycast »). Cependant, la longueur du préfixe de sous-réseau n'est pas spécifiée, car une adresse « anycast » doit s'adapter aussi bien aux plans d'adressage actuels (où la longueur est généralement de 64 bits) qu'aux futurs plans qui pourraient avoir des tailles différentes.

4.2.2.4 Routage et transmission de parquets

Les routeurs acheminent les paquets entre les réseaux qui constituent Internet **par la partie réseau de l'adresse IP,** représentée par **le préfixe**. Un préfixe n'a pas de longueur prédéfinie, elle doit donc être spécifiée grâce à la notation CIDR. La notation CIDR a permis de réduire la taille des tables de routage en spécifiant une large plage d'adresses dans une seule entrée de la table de routage. En stockant des informations de préfixes de réseau IPv6, la table de routage représente ainsi un ensemble de routes (itinéraires), accessibles directement ou indirectement (elle permet également de spécifier la manière de joindre ces réseaux). Chaque nœud IPv6 (hôte ou routeur) possède sa propre table de routage IPv6. Comme vu précédemment, l'en-tête d'un paquet IPv6 contient à la fois l'adresse de l'hôte émetteur et l'adresse de l'hôte destinataire. Lorsqu'un paquet IPv6 arrive sur un routeur, sur une interface réseau physique ou logique, celui-ci utilise le processus suivant pour déterminer comment transférer le paquet vers la destination souhaitée :

a) Le routeur vérifie dans sa mémoire cache de destination (« destination cache ») s'il existe une entrée correspondant à l'adresse de destination dans l'en-tête du paquet. Cette mémoire contient les adresses IP recherchées récemment pour réduire le temps de recherche. Si une telle entrée est trouvée, le routeur transfère le paquet directement à l'adresse spécifiée dans l'entrée de sa mémoire cache de destination et le processus de routage se termine.

b) Si la mémoire cache de destination ne contient pas d'entrées correspondantes à l'adresse de destination dans l'en-tête du paquet ou que le routeur ne fait pas de mise en cache (« caching »), le routeur utilise sa table de routage pour déterminer comment transférer le paquet. Pour transmettre le paquet, il a besoin des informations du prochain saut où il doit envoyer le paquet, à savoir :

   - L'adresse du prochain saut (« Next-hop address »): si l'hôte de destination se trouve dans le réseau local, l'adresse du prochain saut est simplement l'adresse IP de destination dans l'en-tête du paquet. Si l'hôte de destination se trouve dans un réseau distant, l'adresse du saut suivant est l'adresse d'un routeur connecté à ce réseau local.
   - L'interface de saut suivant: il s'agit de l'interface réseau physique ou logique du routeur qui doit être utilisée pour transférer le paquet.

   Le routeur détermine l'adresse et l'interface du prochain saut comme suit :

   - Pour chaque entrée de la table de routage, les *n* bits du préfixe réseau de la route sont comparés aux mêmes bits de l'adresse de destination dans l'en-tête du paquet, *n* étant la longueur du préfixe de la route. Si ces bits correspondent, il est déterminé que l'itinéraire correspond à la destination.
   - Si plusieurs itinéraires correspondants sont trouvés, l'itinéraire correspondant ayant le **préfixe le plus long** est choisi et le processus de détermination d'itinéraire est terminé. Il s'agit du concept du LPM.
   - Si aucune route n'est trouvée, le paquet est transféré à l'aide de la route par défaut (avec le préfixe réseau :: / 0).

   Il est possible ainsi de définir des réseaux qui font exception à une plage d'adresses, représentée par un préfixe, par des entrées supplémentaires dans la table de routage (fournissant des règles de routage spécifiques).

c) Le routeur transmet ensuite le paquet à l'adresse du saut suivant à l'aide de l'interface correspondante. Il met également à jour sa mémoire cache de destination avec ces informations afin que les paquets ultérieurs envoyés à la même adresse de destination

puissent être transférés plus rapidement, à l'aide de sa mémoire cache au lieu d'utiliser sa table de routage.

Avec l'émergence d'Internet, de nombreux chercheurs du monde universitaire et de l'industrie se sont penchés sur le problème de la recherche IP, essentiellement depuis l'adoption du CIDR, où cette dernière est devenue plus complexe (basée sur le LPM).

Nous discuterons dans le chapitre suivant des approches et des solutions les plus populaires pour effectuer des recherches de type LPM. Notons que les solutions décrites sont utilisées pour tous types de données et toutes exigences de recherche, mais le LPM représente un plus grand défi en raison de sa complexité.

CHAPITRE 5 SYNTHÈSE DES SOLUTIONS DISPONIBLES

5.1 Structures de données et algorithmes

Les structures de données et les algorithmes sont utilisés pour tous types de données et types de recherche. Afin d'évaluer une structure des données, trois mesures principales sont prises en compte : la complexité temporelle pour effectuer une opération (recherche ou mise à jour), l'efficacité de la taille (« size efficiency ») et la flexibilité. L'efficacité de la taille est simplement le rapport entre l'espace requis pour maintenir la structure de données et la taille des données en elles-mêmes. Habituellement, les complexités temporelles et spatiales (taille) forment un compromis: l'efficacité d'un paramètre provoque une déficience de l'autre. La flexibilité réfère à la capacité d'augmenter la structure de données pour répondre à des futures exigences.

5.1.1 Listes et tables

La liste chaînée est l'une des structures de données les plus simples où les préfixes sont ordonnés linéairement. Les algorithmes d'insertion sont très efficaces (complexité temporelle O (1)). Cependant, toutes les autres opérations (recherche, suppression ou modification) sont très lentes. Une opération de recherche peut avec une complexité temporelle de O (N), avec N représentant le nombre de préfixes contenus dans la table.

Les tables à accès direct (« direct address tables ») sont des tableaux d'éléments adressés par leurs clés. Ces structures sont les plus efficaces pour des recherches de type « exact match » parce que toutes les opérations ont une complexité temporelle O (1). Le problème de ce type de structure est qu'elle nécessite un espace (taille de la mémoire) de la taille de la plage des clés ; par exemple, pour une clé de 64 bits, une table avec 2^{64} entrées doit être conservée. Afin de palier à ce problème, on utilise les tables de hachage qui permettent de compresser la table. En effet, la table de données est presque vide lorsque l'espace d'adressage est très vaste donc réduire l'espace d'adressage via une fonction de hachage permet donc d'augmenter le taux de remplissage de la taille et de diminuer la taille de la mémoire utilisée.

- Hachage

Une table de hachage est une structure de données qui permet une association clé–élément. Tout comme pour les tables directement adressables, on accède à chaque élément de la table par sa clé.

Néanmoins, l'accès s'effectue par une ou plusieurs fonctions de hachage qui transforme la clé en une valeur de hachage, qui représente un pointeur vers l'élément stocké dans la table de hachage, comme illustré dans la figure 4-1. Une entrée de la table de hachage, appelée case ou alvéole (« bucket »), contient la clé de hachage et les données associées à celle-ci. La fonction de hachage vise à distribuer uniformément un grand nombre de valeurs de clé sur un plus petit nombre de cases de hachage, indépendamment de la structure des données stockées. Cette solution permet d'avoir une table relativement compacte et offre une excellente flexibilité.

Cependant, des collisions sont possibles : c'est-à-dire que deux clés peuvent produire la même valeur de hachage, ce qui affecte la complexité temporelle des opérations de recherche. La qualité de distribution d'une fonction de hachage (nombre de collisions générées) représente un facteur de performance important. Une fonction de hachage imparfaite entraîne des collisions et ne peut pas fournir une complexité temporelle fixe car des opérations de recherche supplémentaires sont nécessaires (opération de recherche pour chaque élément de la case), réduisant considérablement la performance de la solution. En revanche, le nombre de collisions peut être réduit en augmentant le nombre de cases de hachage (compromis) : une mémoire plus importante (moins compacte) entraine naturellement moins de collision. Aussi, L'utilisation de fonctions de hachage dites universelles permet de minimiser la probabilité de collision de hachage, ces fonctions permettent de définir la probabilité d'une collision pour deux clés distinctes.

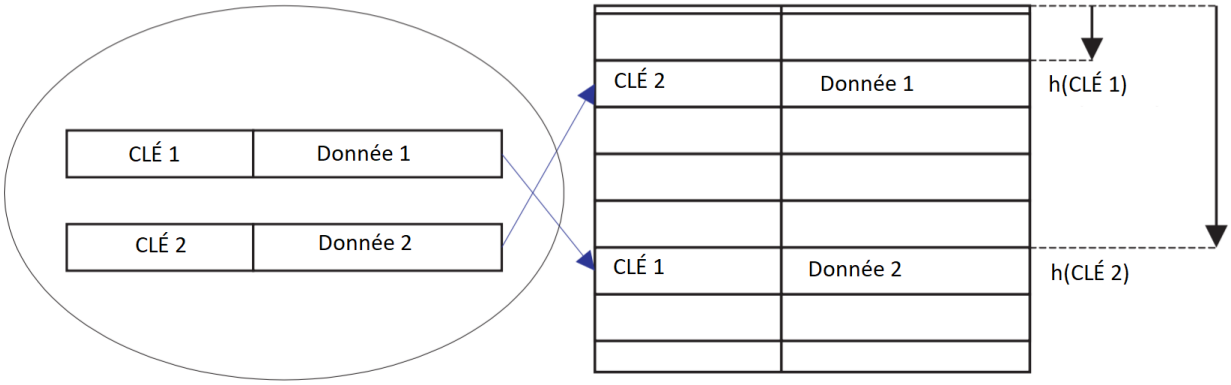

Figure 5-1 : Correspondance du hachage

Par définition, une fonction de hachage prend en entrée une clé de taille fixe et retourne en sortie une adresse ayant également une taille prédéfinie. Par conséquent le hachage n'est pas directement

adapté pour la recherche LPM car le CIDR nécessite que les routeurs traitent des préfixes de taille variable. En effet, la recherche d'une interface se fait intuitivement par le préfixe associé, qui joue donc le rôle de la clé de hachage. Nous allons présenter quelques solutions permettant de relever ce défi.

Afin de supporter le LPM, Lim et al. [6] proposent une architecture parallèle exploitant la correspondance directe. Comme le montre la figure 4-2, chaque plan est dédié à des préfixes de même taille (un plan par taille de préfixe) et comprend une fonction de hachage propre, une table de hachage principale et une sous-table qui gère les collisions. Quand le moteur de recherche reçoit une adresse IP, elle est transmise à tous les plans pour un traitement parallèle. Chaque plan extrait les *n* bits les plus significatifs de l'adresse IP, où *n* représente la taille du préfixe traité par le plan, pour former la clé de hachage. Cette dernière est transmise à la fonction de hachage appropriée pour produire le résultat de hachage, qui localise une entrée dans la table principale. Si le préfixe utilisé comme clé de hachage correspond au préfixe stocké dans l'entrée pointée, alors l'adresse IP correspond au préfixe stocké et un pointeur vers la mémoire RAM, qui contient les informations des prochains sauts, est transmis à l'encodeur à priorité. Si la clé de hachage ne correspond pas au préfixe stocké dans cette entrée, une recherche binaire est effectuée dans la sous-table pour les préfixes ayant provoqués des collisions. L'encodeur prend en entrée les adresses des préfixes qui correspondent à l'adresse IP à rechercher et sélectionne celle qui correspond au plus long préfixe.

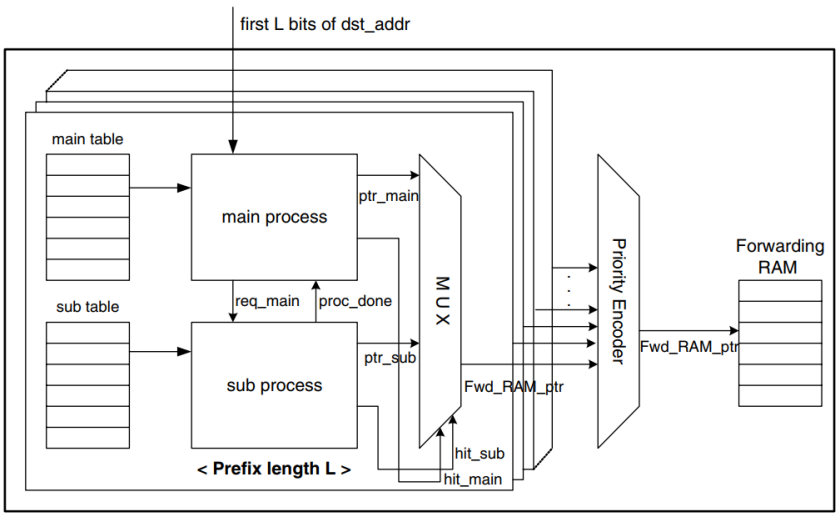

Figure 5-2: Hachage parallèle [6]

Cette architecture exploite le hachage parallèle et exige un plan de traitement (table et fonction de hachage) par taille de préfixe et devient par conséquent couteuse pour les adresses Ipv6, dont la taille du préfixe peut atteindre 128 bits.

Une solution pour réduire le nombre de plans de traitement consiste à utiliser l'extension du préfixe [32] [33] (« Prefix Expansion »), qui consiste à étendre la taille d'un préfixe. L'extension de préfixe remplace un préfixe court par un ensemble de préfixes plus longs permettant d'avoir des préfixes de même longueur (permettant de s'affranchir du LPM si tous les préfixes ont la même longueur). Par exemple, un préfixe 001/3 peut être étendu aux préfixes 0010/4 et 0011/4 ou encore aux préfixes 00100/5, 00101/5, 00110/5 et 00111/5.

CA-RAM [31] est une structure de mémoire spécialisée proposée pour accélérer des opérations de recherche et qui étend la clé de hachage pour prendre en charge la recherche de type LPM. CA-RAM utilise une mémoire classique à haute densité (SRAM) et un certain nombre de modules matériels pour offrir une capacité de recherche en parallèle, comme illustré dans la figure 4-3. Chaque ligne mémoire contient un « bucket » qui peut contenir plusieurs préfixes (dû aux collisions) et chaque bloc logique compare un préfixe avec la clé de recherche.

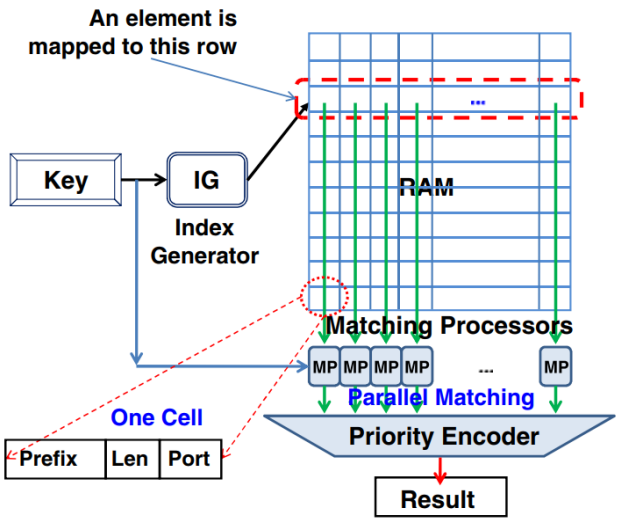

Figure 5-3: Implémentation du hachage parallèle grâce à l'extension des préfixes [7]

Cependant, l'extension du préfixe nécessite une mémoire plus grande pour stocker les préfixes de substitution, puisque chaque bit d'extension nécessite deux préfixes de remplacement. Le hachage groupé [7] [34] (« Grouped Hashing ») représente l'autre solution majeur pour supporter la taille variable des préfixes. Dans le hachage groupé, les préfixes sont regroupés en fonction de leur

longueur, et une fonction de hachage est associée à chaque groupe. Par exemple, des préfixes « unicast » peuvent être répartis comme suit :

- Groupe S64 qui contient des préfixes de longueur 48 et 64 bits
- Groupe S32 qui contient les préfixes de longueur 32 et 48 bits
- Groupe S23 qui contient des préfixes de longueur comprise entre 23 et 32

La clé d'une fonction de hachage doit avoir une taille définie. Par conséquent, les préfixes de tailles différentes d'un même groupe doivent être étendus ou raccourcis. Par exemple, une fonction de hachage qui prend une clé de 23 bits peut être appliquée au groupe S23. L'efficacité du hachage groupé dépend principalement de la façon dont nous sélectionnons les groupes et les fonctions de hachage associées. Raccourcir la taille d'un préfixe pour former une clé de hachage n'entraine pas une table de recherche plus grande, contrairement à l'extension d'un préfixe. Par exemple, on peut former une clé de hachage pour la fonction S23 en sélectionnant les 23 premiers bits d'un préfixe de 26 bits. Le résultat de hachage servira de pointeur vers un « bucket » où le préfixe de 26 bits sera stocké. Cependant, cette approche entraine naturellement plus de collisions.

Le hachage représente une solution bien répondue pour le LPM, permettant d'implémenter une structure de données abstraite où une relation 1 : 1 entre une clé et une valeur de la mémoire existe. Ce type de mémoire est mieux connu sous le nom de **mémoire associative**. Ces solutions peuvent s'interfacer avec le processeur autant qu'avec des accélérateurs matériels (voir Figure 2-2). Ces deniers implémentent des tâches de traitement spécifiques au réseau dans une logique personnalisée et offrent des performances bien supérieures aux implémentations logicielles classiques.

5.1.2 Structure arborescentes

Un moyen efficace et naturel de faire du LPM est d'utiliser des structures de données arborescentes (« tries »). Il s'agit d'une solution très bien connue dans la littérature [9] [12] et, au vu de sa complexité, nous allons uniquement nous attarder sur la solution la plus basique : la structure arborescente binaire.

La figure 4-4 montre une structure arborescente binaire représentant un ensemble de préfixes d'une table de transmission, où un préfixe P est stocké dans un nœud. Chaque arc (« edge ») contient un bit d'information : Les arcs gauches signifient un bit valant zéro et les arcs droits signifient un bit

valant un. Un chemin de la racine à un nœud représente une succession de bits qui correspond à un préfixe et la profondeur d'une structure est égale à la longueur du préfixe le plus long qui existe dans la table. Par exemple, dans la figure 4-4, le préfixe P_3 est au niveau 5 (représentant sa longueur) et représente toutes les adresses commençant par la séquence 01011 (valeur du préfixe).

La recherche est guidée par les bits de l'adresse IP de destination à rechercher. À chaque nœud, la recherche s'effectue à gauche ou à droite en fonction de l'inspection séquentielle des bits d'adresse. Si nous visitons un nœud marqué comme préfixe, ce dernier correspond alors à l'adresse transmise en entrée. Certaines adresses peuvent correspondre à plusieurs préfixes. Par exemple, les adresses commençant par 01011 correspondront aux deux préfixes P_6 et P_3. Néanmoins, le préfixe P_3 doit être préféré car il est plus spécifique (règle de correspondance la plus longue, LPM). La recherche se termine quand il n'y a plus de nœuds à parcourir et, parmi les préfixes qui correspondent à l'adresse IP, le préfixe le plus long doit être retenu.

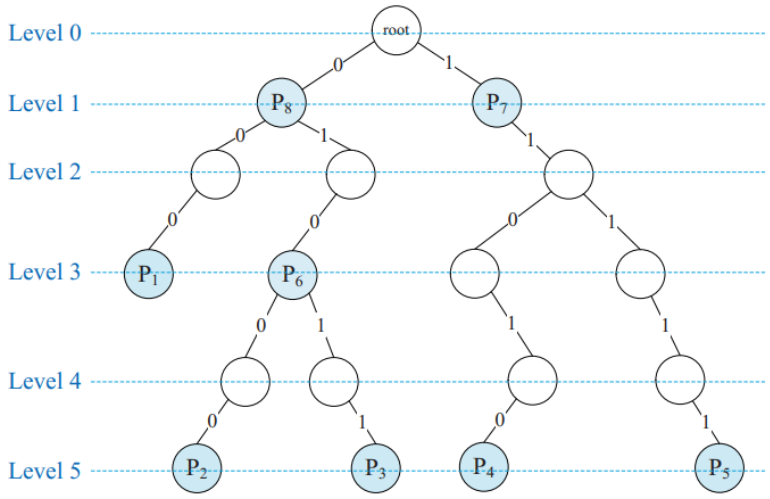

Figure 5-4: Structure d'arborescence binaire [35]

Pour faire du LPM, nous effectuons donc une recherche séquentielle de préfixes par longueur, en essayant à chaque niveau de trouver une meilleure correspondance. Par conséquent, une structure de données pour des préfixes de longueur maximal N, nécessite une mémoire O (NM) (où M est le nombre de préfixes), et possède une complexité temporelle de O (N). Pour réduire la complexité matérielle et temporelle, il existe une variante de cette structure de données appelée **arbre binaire** [3], où les informations sont contenues explicitement dans les nœuds (le nombre de nœuds est donc égal au nombre de préfixes de réseau).

5.2 TCAM

Un moteur de recherche dans un traitement de type Match-Action a besoin de l'abstraction suivante : retourner une adresse à partir d'une donnée (voir Figure 2-6). Pour une recherche de type LPM, la donnée en entrée est l'adresse IPv6 de destination extraite du paquet et l'adresse en sortie représente un pointeur vers l'action à effectuer, qui consiste à transférer le paquet vers une interface.

Une mémoire ordinaire renvoie une valeur à partir de son adresse. L'opération inverse se produit avec un type de mémoire spécifique, appelé mémoire adressable par contenu (CAM), qui renvoie l'adresse de la valeur présentée en entrée. Un schéma simplifié d'une CAM à *m* mots de *n* bits est représenté dans la figure 4-5. C'est une mémoire composée d'une mémoire à semi-conducteur conventionnelle, où les mots sont stockés horizontalement et chaque bit représente une cellule qui contient un espace de stockage et un circuit de comparaison. La clé (mot à rechercher) est diffusée verticalement sur les cellules de la CAM à travers les « Search Lines » et comparée avec les mots stockés. Les « Match Lines » indiquent si la clé correspond au mot stocké en devenant actives ; si la ligne est active, on a une concordance (« match »), sinon une non-concordance (« mismatch »). Cette indication est le résultat d'un "et" booléen entre toutes les comparaisons des bits du mot à rechercher avec les cellules d'une ligne CAM. Toute non-concordance dans l'une des cellules d'une ligne entraîne un « mismatch ». Les « Match Lines » connectées à un encodeur qui calcule l'adresse de la clé de recherche.

La valeur stockée dans une cellule CAM est binaire, soit '0' ou '1'. Il existe des CAM ternaires, appelées TCAM, capables de supporter un type additionnel de bit qui est le « wildcard » (ou « don't care »), représenté par un 'X'. Lors d'une recherche, les cellules qui contiennent les « wildcards » permettent de ne pas affecter la « Match Line », indépendamment du bit recherché. Les « wildcards » permettent donc de faire des recherches masquées où certains bits d'une clé de recherche peuvent ne pas être pris en considération.

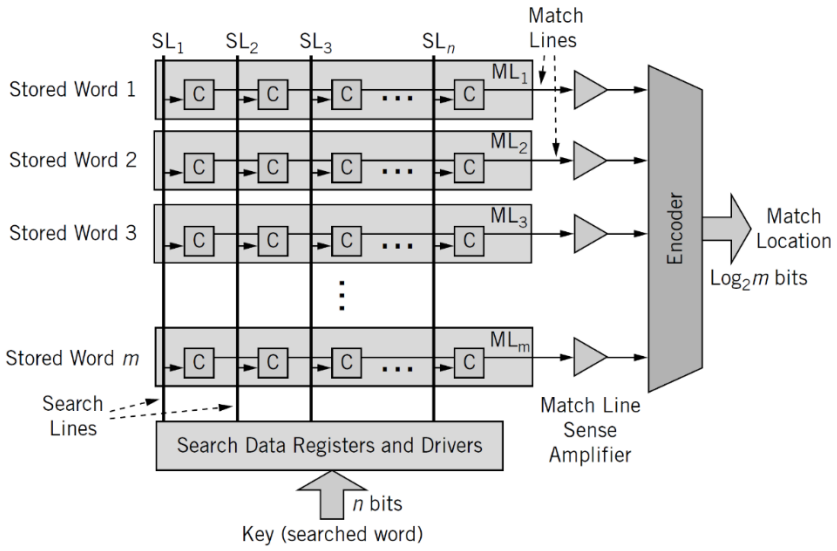

Figure 5-5 : Structure de la mémoire CAM [9]

Il est possible d'avoir plusieurs « Match Lines » activées lors d'une recherche dans une CAM binaire, cette probabilité est plus importante avec une TCAM car les bits à faire correspondre aux bits stockés sont moins nombreux. Si plusieurs « matches » sont possibles, un encodeur à priorité est utilisé pour retourner l'adresse de l'entrée avec la plus grande priorité ; par exemple, l'une des stratégies consiste à retourner la plus petite adresse parmi les lignes actives (« First Matching »). Avec la TCAM, des mécanismes de priorisation plus complexes existent qui permettent de faire des recherches de type LPM.

Dans la plupart des implémentations, la sortie de la TCAM est utilisée pour piloter une mémoire RAM qui retourne la valeur attachée à la clé de recherche passée à la TCAM, comme illustré dans la figure 4.6 (avec une adresse de 4 bits). Lors d'une recherche LPM, l'adresse IP de destination est utilisée comme clé de recherche dans la TCAM et la sortie de l'encodeur représente l'adresse de la mémoire RAM qui contient le port et l'adresse du prochain saut pour acheminer le paquet.

Le LPM se fait avec l'utilisation de TCAM en stockant chaque préfixe dans les bits de gauche d'un mot et de compléter les bits restants du mot avec des « wildcards ». La taille des mots est celle de la taille maximum que peut avoir un préfixe (128 bits pour IPv6). Puisque plusieurs « matchs » sont possibles, les préfixes sont ordonnés en fonction de leur taille pour que l'encodeur puisse retourner le plus long préfixe, en retournant la plus grande adresse de la ligne active. Dans notre exemple, Figure 4-6, les lignes stockant les mots 101X, 10XX et 1XXX qui correspondent aux

préfixes 101/1, 10/2 et 1/3 respectivement, sont actives car les mots concordent avec la clé de recherche 1010. L'encodeur reçoit trois correspondances et retourne la plus grande adresse de la ligne active. L'adresse permet à la mémoire RAM de retourner les informations du prochain saut.

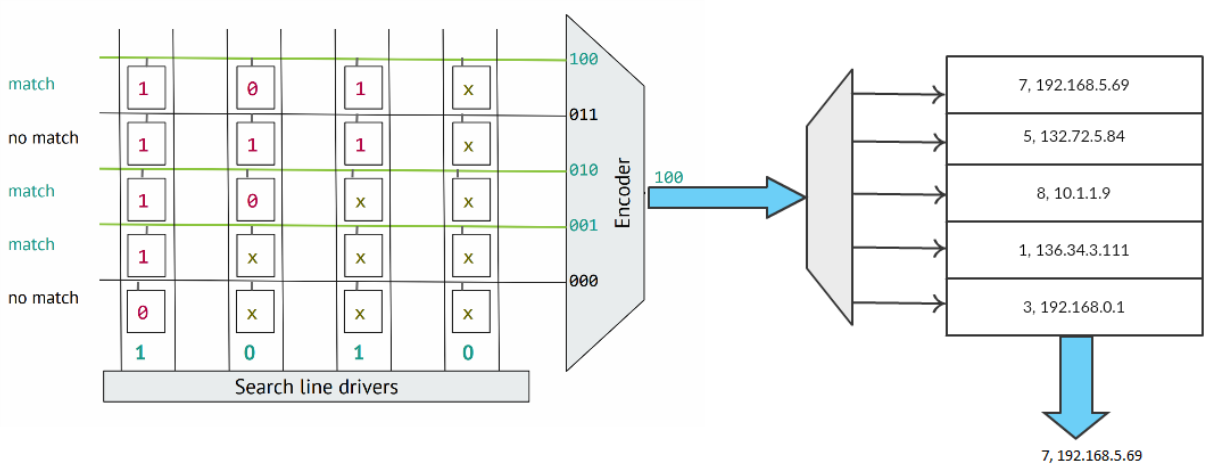

Figure 5-6: Implémentation basée sur une TCAM pour le LPM [8]

Parce qu'une CAM effectue une recherche dans sa mémoire en un seul cycle, il s'agit d'une mémoire plus rapide que les autres mémoires dans toutes les applications de recherche d'un contenu. La TCAM est une CAM ternaire qui permet de supporter des recherches masquées et s'est imposée comme un standard *de facto* dans l'industrie pour effectuer des recherches de type LPM.

Ces moteurs de recherche constituent des accélérateurs matériels pouvant s'interfacer avec le processeur de réseau comme une mémoire ordinaire (Figure 2-2) et travailler de manière transparente avec le processeur de réseau ou en utilisant l'interface standard LA-1 [9]. L'interface LA-1 est essentiellement une interface SRAM, avec certaines modifications destinées à intégrer des coprocesseurs, telles que la latence d'accès variable, les réponses en désordre (« out-of-order responses »), etc.

- Solutions basées sur la mise en cache

Comme mentionné précédemment, certains routeurs utilisent un système de « caching » pour accélérer le LPM. La mémoire cache est une mémoire bien connue et répandue en informatique (architecture des ordinateurs) qui enregistre temporairement une copie de données provenant d'une autre mémoire plus lente, afin de réduire le temps d'accès à cette dernière.

La mémoire sur puce (« on-chip memory », voir figure 2-2) d'un processeur réseau joue le rôle, dans certaines implémentations, de mémoire cache entre le processeur et la mémoire externe. Dans la plupart des cas, la mémoire sur puce utilise la technologie SRAM, qui est une mémoire de faible capacité mais très rapide, alors que la mémoire externe utilise la mémoire DRAM pour avoir une plus grande capacité (au détriment de la vitesse).

Afin d'accélérer le LPM, la mémoire cache enregistre les adresses IP recherchées récemment, afin de diminuer le temps d'accès à la table de routage. Si l'adresse IP à rechercher ne correspond à aucune entrée de la mémoire cache, un échec en cache (« cache miss ») se produit et elle doit être mise à jour en conséquence à partir de la table de routage. Ainsi, le taux de « cache miss » représente un facteur limitant de performance.

Les caches sont aussi utilisés pour contenir des préfixes [5] plutôt que des adresses, une seule entrée de cache peut ainsi couvrir une plage numérique (plusieurs adresses) et peu réduire le taux de « cache miss ». Une description fonctionnelle d'un cache [37] utilisé pour la recherche IP conçue par Berube et al. est montrée à la figure 4-7. Un tableau d'adresses de destination (« Destination Address Array », DAA) stocke les adresses IP où les préfixes en utilisant une mémoire CAM ou TCAM respectivement, et un tableau de sauts suivant (« Next-Hop Array », NHA) stocke les informations des nœuds et qui peut être mis en œuvre en utilisant la technologie SRAM CMOS standard.

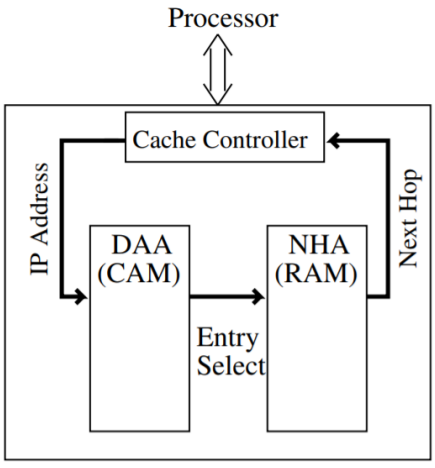

Figure 5-7: Représentation fonctionnelle d'un cache pour la recherche IP [36]

- Inconvénient des TCAM

Les TCAM sont des mémoires rapides mais présentent certains inconvénients graves [38] : leur consommation d'énergie élevée, leur faible flexibilité (opérations de mise à jour lente) et leur coût financier (coût par bit plus élevé par rapport aux autres types de mémoire).

La TCAM effectue des comparaisons sur l'ensemble de ses entrées (on dit qu'elle est de nature totalement associative), ce qui la rend gourmande en énergie : une TCAM classique à 18 Mbits (512 000 entrées de préfixes IPv4) peut consommer jusqu'à 15 watts lorsque d'une opération de recherche [39].

Conventionnellement, on utilise des TCAM avec des mots de 144 bits pour stocker les préfixes IPv6, ce qui cause une efficacité d'espace relativement faible et une augmentation considérablement de la consommation énergétique.

La consommation d'énergie de la TCAM est critique dans les applications de routeur. Un routeur basé sur des processeurs réseaux contient principalement plusieurs cartes de ligne (« line cards »), comme illustré dans la figure 4-8. Ces « line cards » sont des cartes électroniques qui présentent les points d'entrée et de sortie des paquets et ont des budgets de puissance fixes en raison de contraintes de refroidissement et de répartition de l'alimentation [40].

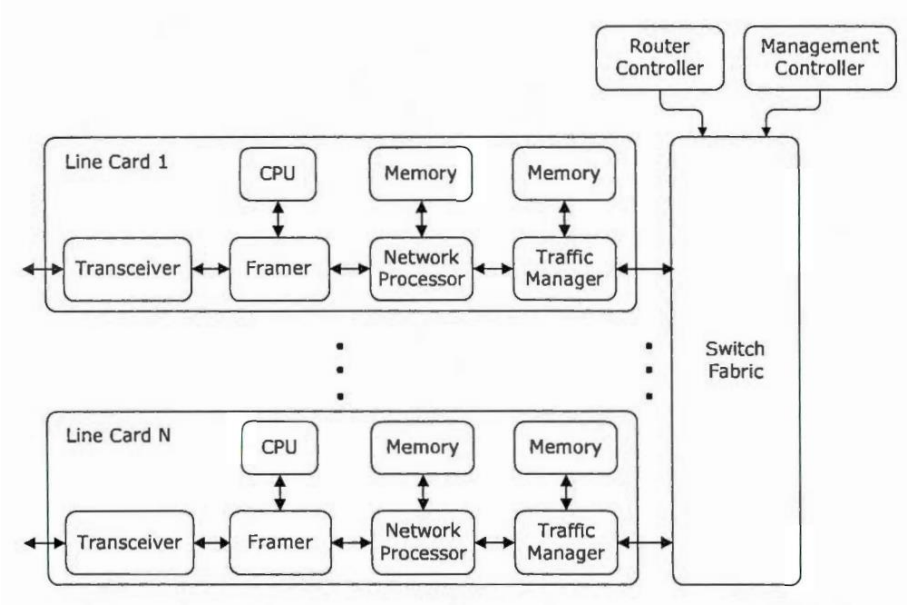

Figure 5-8: Routeur basé sur des processeurs réseau [39]

5.3 Conclusion

Les solutions présentées peuvent avoir des implémentations différentes (matérielle, logicielle ou hybride). Pour répondre aux exigences de performance, nous nous sommes intéressés à concevoir une solution matérielle spécialisée (accélérateur matériel). L'implémentation d'une tâche spécifique dans un matériel personnalisé présente un avantage évident : elle est plus efficace de deux à trois ordres de grandeur qu'une solution purement logicielle [31].

Aussi, beaucoup de travaux se sont focalisés sur la localité temporelle du trafic Internet [41] pour concevoir des systèmes basés sur la mise en cache pour accélérer le LPM. Ces mémoires caches utilisent souvent des mémoires TCAM couteuses pour supporter les « wildcards ».

Le but de notre travail est de concevoir une mémoire cache capable de supporter le LPM à l'aide de modules matériels spécifiques. Nous exploitons les propriétés du hachage pour pouvoir implémenter des mémoires associatives et nous utilisons le hachage groupé pour réduire la complexité matérielle de la logique (due à l'exploitation parallèle) et de la mémoire (déjà limitée dans un processeur réseau). Notre solution vise une implémentation utilisant des mémoires SRAM, six fois moins couteuses que les mémoires TCAM [8].

CHAPITRE 6 MÉMOIRE CACHE SUPPORTANT LA RECHERCHE IP

6.1 Mémoire cache de processeur

La mémoire cache est le niveau de mémoire intermédiaire entre le CPU et la mémoire principale dans la hiérarchie de la mémoire [46] (voir figure 5-1).

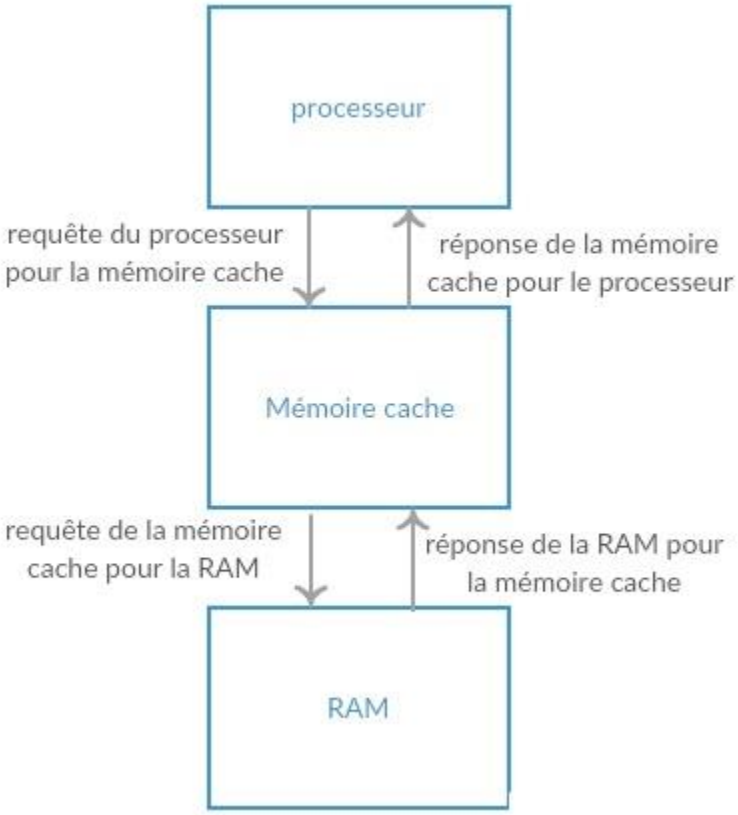

Figure 6-1: Hiérarchie mémoire

La mémoire cache peut-être intégrée au processeur ou elle peut être hors puce (« off-chip ») et elle est souvent implémentée en utilisant la mémoire SRAM. Très performante mais onéreuse, la mémoire cache est organisée en niveaux (souvent au nombre de trois : L1, L2 et L3), en fonction de sa proximité avec le microprocesseur et de sa capacité ; la mémoire cache L1 étant la plus rapide mais la plus petite. L'utilisation de mémoires caches est devenue populaire pour les systèmes affichant une bonne localité temporelle ou spatiale, ce qui signifie que ces systèmes, au cours d'une courte période de temps donnée, accèdent des données qui sont, pour la plupart, situées dans une petite partie de la mémoire. Le cache est utilisé pour stocker un sous-ensemble de la mémoire

principale accédé fréquemment dans une courte période de temps. Nous nous intéressons à l'architecture de cette mémoire pour mettre en œuvre une architecture supportant les recherches de types LPM de manière performante. La politique de cache (caractérisée par les données à mettre en cache et les à mettre à jour) n'est pas traitée dans ce mémoire. Une telle technique a été proposée dans [45].

6.1.1 Architecture de base de la mémoire cache

On distingue trois parties principales dans une ligne de mémoire cache : une étiquette (« tag »), un bloc de données et des drapeaux (« flags »). Un bloc est le plus petit élément de données qui peut être transféré entre la mémoire cache et la mémoire de niveau supérieur. Un bloc de données est constitué de mots qui représentent le plus petit élément de données qui peut être transféré entre le processeur et la mémoire cache. La taille de la mémoire cache est donnée par la taille d'un bloc de données multiplié par le nombre de ligne de la mémoire cache.

La mémoire cache a besoin de connaitre la provenance du bloc de données (son adresse dans la mémoire principale), cette information se trouve dans l'étiquette. Les drapeaux d'une ligne de cache représentent des bits d'information d'états. Les deux bits d'états les plus répandus sont le bit de validité et le « dirty bit ». Le bit de validité permet d'indiquer que le bloc de données dans la ligne de cache est valide et peut être accédé par le processeur. Le « dirty bit » permet d'indiquer si les données d'une ligne de mémoire cache sont identiques ou différentes de celles présentes dans la ligne de la mémoire principale. La figure 5-2 représente l'architecture de base d'une mémoire cache de 4 KB de 256 lignes de mémoire cache contenant 4 mots de 32 bits (on remarque la correspondance avec l'adresse transmise par le processeur, par exemple 8 bits d'adresse pour adresser 256 lignes).

6.1.2 Opération du contrôleur de cache

Le contrôleur de cache est un module matériel qui dicte les opérations d'écritures et de lectures dans la mémoire cache. Quand le processeur veut effectuer l'écriture ou la lecture d'une donnée, il envoie une requête vers le contrôleur de cache. Ce dernier décompose l'adresse en trois parties : l'étiquette, l'index du groupe (« set index ») et l'index de la donnée (« data index » ou « offset »), comme illustré dans la figure 5-2.

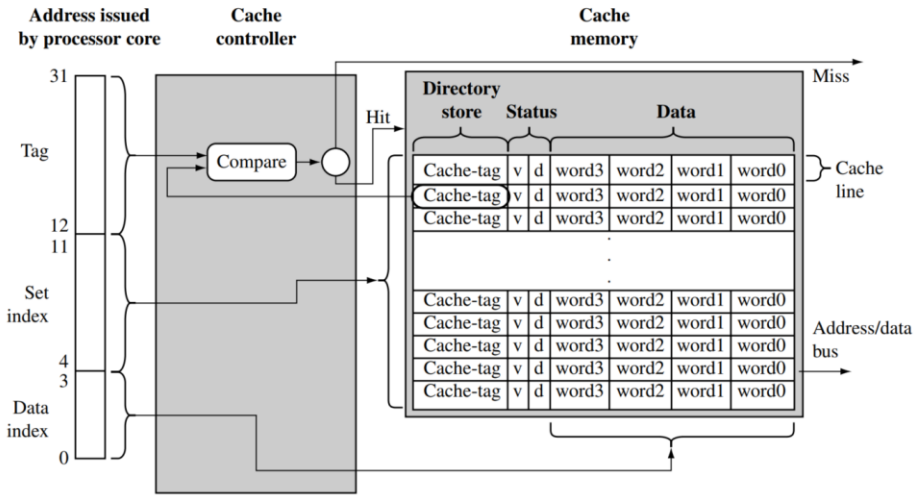

Figure 6-2: Architecture d'une mémoire cache [47]

L'index du groupe permet au contrôleur d'identifier la ou les lignes (selon le niveau d'associativité) susceptibles de contenir la donnée recherchée. Par la suite, le contrôleur vérifie les bits d'états des lignes identifiées précédemment et si l'une d'elles contient l'étiquette qui correspond à l'étiquette de l'adresse transmise par le processeur. Si le contrôleur trouve le « tag » dans l'une des lignes de mémoire cache, l'opération se complète en un « cache hit » sinon en un « cache miss ». Dans le cas d'un « cache hit », le contrôleur identifie le mot recherché dans le bloc de donnée grâce au champ « offset » de l'adresse. S'il s'agit d'une lecture, le contrôleur transmet la donnée directement du cache au processeur. Dans le cas d'une écriture, la politique d'écriture définit comment les données sont mise à jour dans la mémoire principale.

S'il s'agit d'un « cache miss », le contrôleur alloue une ligne de cache où il copie le bloc de donnée de la mémoire principale, et complète la requête du processeur par la suite. Si toutes les lignes de la mémoire cache sont occupées, alors on remplace une des lignes par la nouvelle entrée en fonction de la correspondance et la politique de remplacement ; le contenu de la ligne est écrit en mémoire avant de charger le contenu de la mémoire désiré.

6.1.3 Types de correspondance

La mémoire cache ne pouvant pas mémoriser tout le contenu de la mémoire principale, une méthode doit être définie pour indiquer à quelle adresse de la mémoire cache doit être écrite une ligne de la mémoire principale. Cette méthode s'appelle la correspondance (mapping). Il existe trois types de correspondances répandus :

- Mémoire cache à correspondance préétablie (« direct-mapped cache »)

Dans une mémoire cache à correspondance préétablie, l'adresse d'une ligne de la mémoire principale est associée à une adresse de la mémoire cache ; elle peut être copiée uniquement à cette adresse. La mémoire principale étant plus grande que la mémoire cache, plusieurs adresses de la mémoire principale pointent à la même adresse de la mémoire cache. Un exemple d'implémentation de la correspondance préétablie est la fonction modulo : la ligne d'adresse j en mémoire principale correspond à la ligne d'adresse i en mémoire cache avec i = j % L (L qui correspond au nombre de lignes de cache). La figure 5-3 montre comment un bloc de la mémoire principale est copiée dans la mémoire cache à correspondance préétablie. Dans l'exemple, toutes les adresses se terminant par 0x824 pointent vers la même adresse de la mémoire cache.

Le champ « set index » de l'adresse transmise par le processeur permet d'identifier ligne de cache où se trouve le bloc recherché. Vu qu'une ligne de cache est partagée par de nombreuses lignes de la mémoire de niveau supérieur, il faut savoir quel bloc est actuellement dans le cache. À cet égard, le « tag » de l'adresse est comparé au « tag » présent dans la ligne mémoire cache. Le champ « data index », quant à lui, permet finalement de localiser le mot recherché.

L'avantage de cette méthode est la consommation d'énergie réduite car on sait où se trouve la donnée recherchée. Aussi, la complexité matérielle est faible car une seule comparaison de « tag » est suffisante (contrairement à une TCAM qui fait une comparaison pour chacune de ses lignes comme dans le cas de la cache totalement associative – voir section suivante). La politique de remplacement est un autre avantage car elle est plutôt simple, en effet, en cas de « cache miss », on remplace la ligne cache directement par la nouvelle entrée puisqu'une ligne de la mémoire principale ne peut être copiée qu'à cette adresse. Cependant, le taux de « cache hit » est faible car à chaque fois qu'on recherche un bloc avec le même « index set », on doit remplacer la ligne de cache.

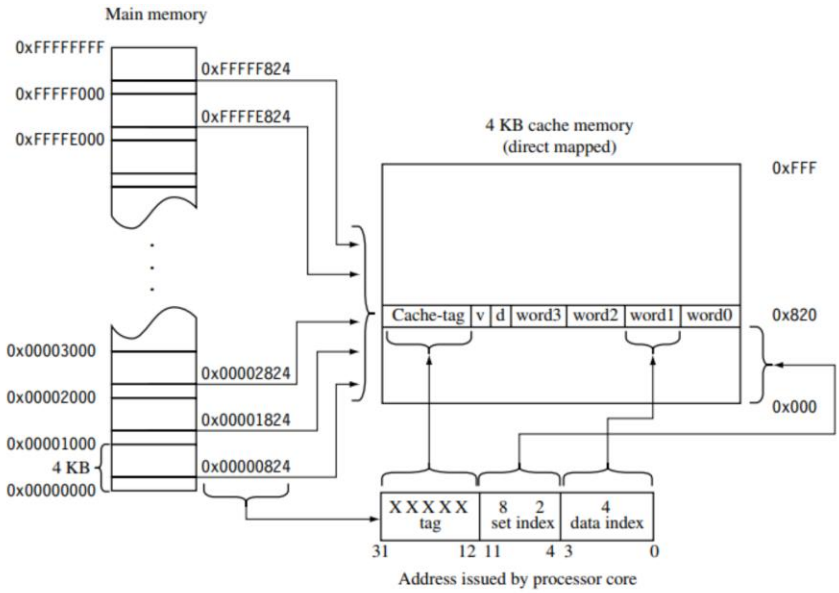

Figure 6-3: Mémoire cache à correspondance préétablie [47]

- Mémoire cache totalement associative (« fully associative cache »)

Dans une cache totalement associative, une ligne de la mémoire principale peut être copiée dans n'importe quelle ligne de la mémoire cache. L'adresse envoyée au contrôleur de cache pour rechercher un mot ne contient que l'étiquette et l'index de la donnée. Le champ « set index » n'est pas utile pour le processeur, car le bloc de la mémoire principale peut se trouver n'importe où dans le cache. Le « tag » de l'adresse est comparé au « tag » de toutes les lignes de la mémoire cache. En cas de « hit », le mot est retourné grâce à son index dans le bloc de données.

Cette méthode a pour avantage d'offrir une flexibilité pour placer et remplacer (large variété d'algorithme de remplacement) un bloc de données dans la mémoire cache. Elle offre également un meilleur taux de « cache hit ». Cependant, elle requiert beaucoup de logique et donc consomme beaucoup d'énergie, car la cache effectue une recherche dans toute sa mémoire pour localiser un bloc.

- Mémoire cache partiellement associative

Elle représente une méthode intermédiaire entre les deux méthodes précédentes. La cache est divisée en ensembles (« ways ») qui contiennent le même nombre de ligne de mémoire cache. Le nombre d'ensembles définit l'associativité (« associativity ») de la cache. Contrairement à la cache à correspondance préétablie, le « set index » n'adresse pas une seule ligne cache mais plutôt N

lignes où N représente l'associativité de la cache. En effet, les lignes de caches de chaque ensemble avec la même adresse font partie du même groupe (« set ») et ce dernier est pointé par le « set index ». Une ligne de la mémoire principale est affectée donc à un groupe. La figure 5-4 illustre un cache avec une associativité de 4 où les lignes de la mémoire cache sont séparées en 4 ensembles.

Quand le processeur transmet une requête de lecture ou d'écriture, le « set index » de l'adresse transmise au contrôleur permet d'identifier le groupe de cache et donc les lignes avec la même adresse dans chaque ensemble. Le « tag » est comparé par la suite aux étiquettes de chaque ligne de cache pointée par le « set index ». Si une des étiquettes correspond au « tag », on obtient un « cache hit » et on retourne le mot recherché d'un bloc grâce au champ « offset ». Sinon, on charge le bloc de données à partir de la mémoire principale. Si toutes les lignes du groupe ont un bit de validité à 1, on remplace une des lignes en fonction de la politique de remplacement.

Cette méthode essaie d'allier la vitesse de la cache à correspondance directe et l'efficacité de la cache totalement associative (taux de « miss cache » relativement faible). La consommation d'énergie est moins importante également par rapport à la cache totalement associative (comparaison de quelques étiquettes uniquement au lieu de toutes les étiquettes).

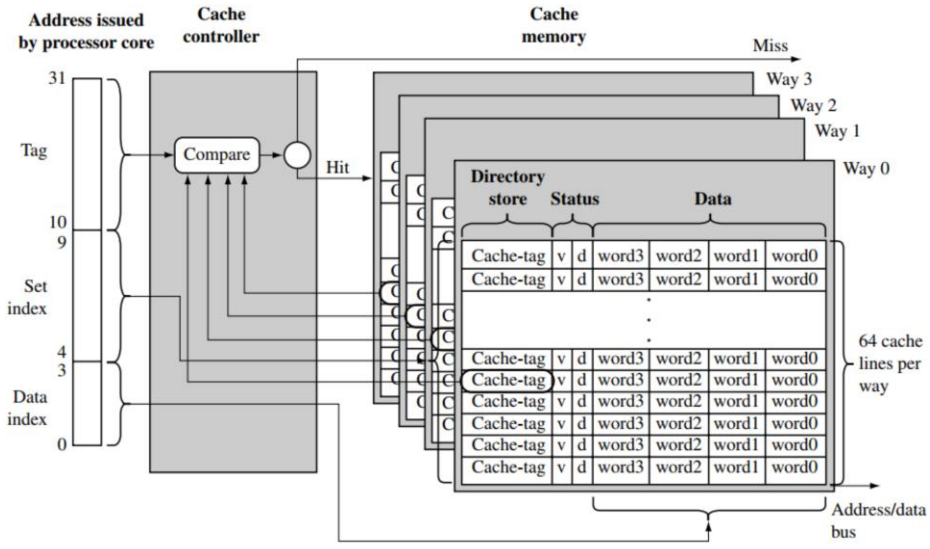

Figure 6-4: Mémoire cache partiellement associative [47]

6.1.4 Approche de conception

La mémoire cache partiellement associative essaie d'allier la performance de la cache à correspondance directe et l'efficacité de la cache totalement associative (taux de « miss cache »

relativement faible). La consommation d'énergie est moins importante également par rapport à la cache totalement associative (comparaison de quelques étiquettes uniquement au lieu de toutes les étiquettes). Par conséquent, nous choisissons cette architecture, d'autant plus que le hachage est une solution bien connue pour implémenter des mémoires associatives.

Plus l'associativité de la cache est grande, plus la complexité matérielle est grande car plusieurs comparaisons d'étiquettes sont nécessaires. La comparaison de « tag » est une opération de recherche de type « exact match ». L'utilisation de CAM pour comparer les étiquettes est le choix de conception que ARM a fait dans ses cœurs de processeur ARM920T et ARM940T, qui utilisent une mémoire cache avec une associativité de 64. La figure 5-5 montre un schéma fonctionnel d'un cache ARM940T.

La mémoire cache est divisée en 64 ensembles et chaque ensemble contient 4 lignes de mémoire cache. Le « set index » permet de sélectionner la mémoire CAM à utiliser parmi les 4 qui sont implémentées et chaque CAM contient les 64 étiquettes des lignes caches d'un même groupe. Le « tag » transmis par le processeur est utilisé comme entrée pour la CAM qui le compare à ses entrées.

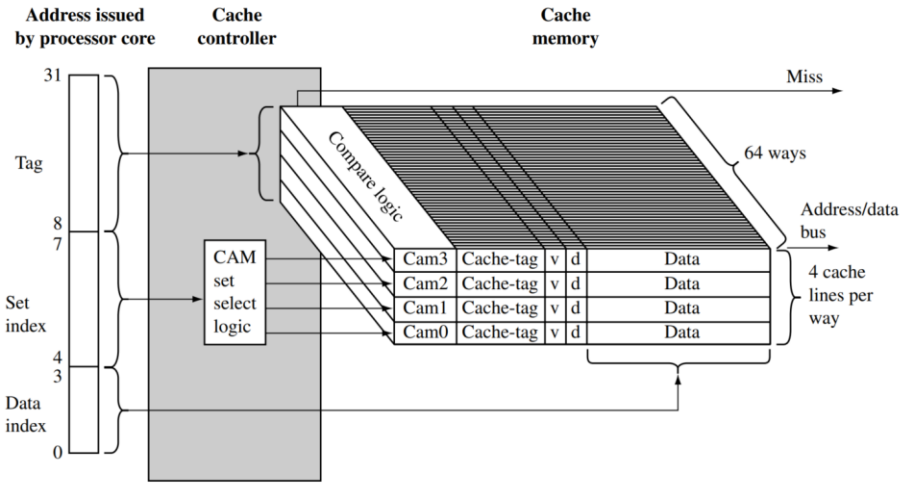

Figure 6-5: Schéma fonctionnel d'un cache ARM940T [47]

Intuitivement, nous pouvons penser à utiliser des TCAM au lieu de mémoires CAM pour pouvoir effectuer des recherches de types LPM, mais, en raison de leur coût élevé, nous proposons dans ce mémoire d'implémenter notre propre solution matérielle.

6.2 Mémoire cache soutenant les recherches IP

6.2.1 Hachage groupé

Le groupement de préfixe est la méthode utilisée pour répondre à la problématique liée aux « wildcards », à savoir la taille variable des préfixes. Dans le hachage groupé, les préfixes sont regroupés en fonction de leurs longueurs et chaque groupe utilise une fonction de hachage différente. La clé de hachage d'une fonction est définie à partir du préfixe à hacher, plus précisément ses bits tels que définis par sa longueur (non- « wildcards »). Un groupe est représenté par un intervalle qui correspond aux tailles de préfixes contenus dans celui-ci.

Les préfixes d'un même groupe ont une longueur différente et la fonction de hachage affiliée doit hacher une clé de taille définie. Étant donné que la mémoire coûte cher (mémoire rapide) et que la taille de la mémoire est limitée physiquement dans un processeur réseau, il est important d'utiliser des structures de données très efficaces sans compromettre le temps de recherche. Puisque l'extension du préfixe nécessite une mémoire plus grande pour stocker les préfixes de substitution, la clé d'une fonction de hachage, dans la solution proposée, a une taille inférieure ou égale à l'ensemble des préfixes d'un groupe : elle représente une sélection de bits du préfixe associé. Étant donné que certains préfixes d'un même groupe partagent les mêmes bits sur cette longueur (taille de la clé), différents préfixes du même groupe peuvent générer la même clé, ce qui augmente le nombre de collisions. En effet, par définition, deux clés identiques passées par la même fonction de hachage induisent le même résultat. Le but de cette section est de présenter les groupes de préfixes utilisés pour ce travail et comment la clé de hachage est construite à partir d'un préfixe donné.

Notre travail est axé sur les préfixes IPv6 globaux « unicast » et les préfixes « anycast », dont la taille est comprise entre 23 et 64 bits (comme discuté dans le chapitre 3). Le table de préfixes de test, rrc00 [42], a été extraite de la base de données ouverte du RIPE NCC. Le RIPE (Réseaux IP Européens) est un forum ouvert à toutes les parties ayant un intérêt dans le développement de l'Internet en Europe.

Nous avons évalué différents nombres et intervalles de groupes, et plusieurs tailles de clé possibles. Par la suite, nous avons examiné les collisions générées uniquement par les clés, sans prendre en compte les fonctions de hachage. Pour exprimer les collisions, chaque clé représente un pointeur

vers un « bucket », cette clé représente donc directement le résultat de hachage. Chaque clé pointe vers une table de hachage propre au groupe auquel son préfixe est associé. Ces évaluations sont présentées dans le tableau 5-1, où les paramètres I, B et M correspondent respectivement à l'intervalle du groupe, les bits sélectionnés à partir du préfixe pour former la clé de hachage et le nombre maximal de collisions générées dans une case de hachage uniquement à partir de ces clés.

Tableau 6-1 : Nombre de collisions maximal dans une case de hachage

Évaluations	**Paramètres**	**Gr0**	**Gr1**	**Gr2**	**Gr3**	**Gr4**	**Gr5**	**Gr6**	**Gr7**
1	I	24-31	32	33-40	41-47	48	49-64		
	B	0-23	0-31	1-32	9-40	16-47	17-48		
	M	25	1	39	32	4	55		
2	I	26-30	31-35	36-39	40-43	44-47	48-63		
	B	2-25	0-30	4-35	8-39	12-43	16-47		
	M	9	9	8	6	8	25		
3	I	26-30	31-35	36-39	40-43	44-47	48	49-63	64
	B	0-25	0-30	0-35	0-39	0-43	0-47	0-48	0-63
	M	9	9	8	6	8	1	18	1
4	I	26-30	31-35	36-39	40-43	44-47	48-50	51-63	64
	B	0-25	0-30	0-35	0-39	0-43	0-47	0-50	0-63
	M	9	9	8	6	8	3	6	1

Durant nos évaluations, tous les préfixes dont la taille est comprise entre 23 à 26 bits ont été écartés. Ces préfixes représentent une partie insignifiante de notre échantillon (0,18%) mais provoquent beaucoup de collisions car beaucoup d'entre eux partagent les mêmes 23 premiers bits. Dans une implémentation concrète, nous postulons que ces préfixes peuvent être stockés dans une petite mémoire TCAM. Puisque les collisions ont une répercussion importante sur la performance, le but de ces évaluations est d'avoir le moins de collisions engendrées par les clés de hachage pour anticiper et réduire les collisions finales (générées par les différentes fonctions de hachage).

L'évaluation 4 donne les paramètres qui causent le moins de collisions produites par les clés : 9 collisions, au maximum, dans une case, pour toutes tables de hachage confondus.

La clé est définie en sélectionnant les N premiers bits du préfixe associé, avec N représentants la taille du plus petit préfixe du groupe. Par exemple, pour un préfixe de 56 bits appartenant au groupe 6 (Gr6), qui contient les préfixes dont la taille est comprise entre 51 et 63 bits, la clé de hachage associé utilise les 51 bits de poids forts du préfixe. Évidemment, le préfixe de 56 bits et sa taille seront stockés en mémoire (et non sa clé de hachage) pour pouvoir effectuer la recherche IP.

Intervalle	Nombre de préfixes par groupe
26-30	1025
31-35	7714
36-39	1286
40-43	1680
44-47	1716
48-50	11507
51-63	253
64	693
Nombre total de préfixes	**25874**

Figure 6-6: Groupes de préfixes

Pour réduire les collisions, le nombre de groupes et l'intervalle de chaque groupe sont étudiés. En effet, si on a un groupe avec des préfixes de longueur minimal /x, ce dernier peut contenir des préfixes de longueur / y variable (où y> x) qui possèdent les mêmes x bits. Ainsi, si z est la limite maximale de la longueur d'un préfixe, plus z - x est grand, plus le nombre de clés égales est important et plus le nombre de collisions produites est grand. Aussi, si nous définissons plus de groupes avec des intervalles plus petit, le nombre de fonctions et tables de hachage sera plus important.

La figure 5-6 représente la définition de nos groupes de préfixes et le tableau 5-2 révélé le nombre de « buckets » et de collisions. Par exemple, le groupe 0 (Gr0) contient les préfixes dont la taille est comprise entre 26 et 30 bits, les clés générées (les 26 premiers bits des préfixes) génèrent 843 clés uniques, 43 clés en double, 18 clés en triple etc.

Tableau 6-2 : Nombres de cases de hachage par groupe de préfixes

	1	2	3	4	5	6	7	8	9
Gr0	843	43	18	5	1	0	0	1	1
Gr1	6489	344	91	16	15	5	3	5	6
Gr2	1018	44	21	22	0	1	1	2	0
Gr3	1376	81	15	19	3	1	0	0	0
Gr4	1254	68	19	64	1	0	0	1	0
Gr5	11477	12	2	0	0	0	0	0	0
Gr6	107	37	11	7	1	1	0	0	0
Gr7	693	0	0	0	0	0	0	0	0

6.2.2 Fonctions de hachage

Puisque nous utilisons le groupement de préfixes, nous devons définir une fonction de hachage pour chaque groupe de préfixes. Une bonne fonction de hachage se caractérise par une bonne distribution des préfixes, qui se traduit par un nombre et une taille d'alvéoles générées raisonnable et un taux de remplissage de la mémoire satisfaisant. En effet, nous souhaitons implémenter une solution matérielle, où toutes les cases d'une table de hachage ont la même taille. Si une case en particulier est beaucoup plus grande que les autres, due à une mauvaise distribution, elle oblige toutes les autres à avoir la même taille, et la mémoire implémentée sera en grande partie inutilisée.

La fonction de hachage est une fonction qui associe chaque clé à son résultat de hachage de manière la plus aléatoire possible [43]. Cette définition nous a orienté vers une approche qui permet de générer des nombres pseudo-aléatoires et qui s'implémente bien en matériel.

Le registre à décalage à rétroaction linéaire, ou LFSR, est reconnu comme générateur de nombres pseudo-aléatoires. Il est composé d'un registre à décalage de type SIPO (« Serial In Parallel Out ») et d'une fonction de rétroaction linéaire. Le registre à décalage est un registre, c'est-à-dire un ensemble de bascules synchrones qui sont reliées en une chaine unique. Le registre à décalage de type SIPO permet, à chaque cycle d'horloge, d'insérer un bit en entrée ou de lire les valeurs de

toutes les bascules. La fonction de rétroaction linéaire, implémentée grâce à des portes XOR, permet de générer une suite récurrente linéaire donnée. La figure 5-7 représente un LFSR pour l'implémentation d'un générateur CRC (« Cyclic Redundancy Check »), utilisé pour détecter des erreurs de transmission.

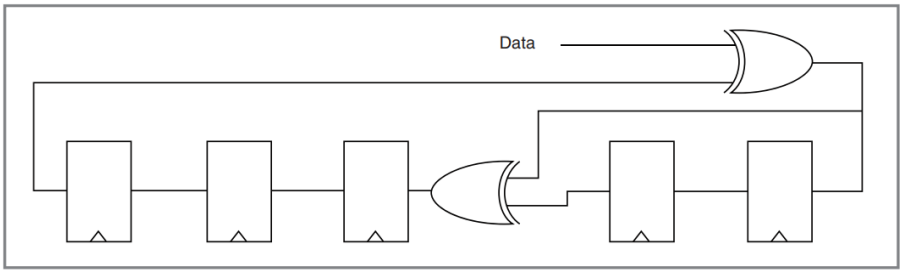

Figure 6-7: Registre à décalage à rétroaction linéaire [44]

Les propriétés du LFSR le rendent intéressant pour implémenter des fonctions de hachage. Le but est de transmettre la clé de hachage à l'entrée du LFSR et d'utiliser son état final, qui correspond aux valeurs de ses bascules après la transmission de la clé (représentant le nombre pseudo-aléatoire généré), comme résultat de hachage. La suite récurrente produite par un LFSR est nécessairement répétitive à partir d'une certaine période. La fonction de rétroaction implémentée détermine cette période et, plus la période est grande plus la génération pseudo-aléatoire est bonne.

Cependant, le LFSR fonctionne habituellement de manière séquentielle, on transmet un seul bit de la clé de hachage, à partir du bit du poids fort, par coup d'horloge. Transmettre une clé de 64 bits prendrait 64 cycles d'horloge, ce qui est évidemment beaucoup trop long dans un contexte où la vitesse de traitement est primordiale. La propriété du LFSR à générer des nombres pseudo-aléatoires repose sur des algorithmes mathématiquement déterministes. Nous nous sommes donc intéressés à paralléliser le LFSR, de manière à prendre N bits en entrée, avec N représentant la taille de la clé. Pour ce faire, nous nous sommes inspirés de la méthodologie décrite dans le *white paper* [44]. Ce papier a pour but d'implémenter un CRC avec un LFSR qui prend une donnée en entrée de N bits, avec N défini, en un cycle d'horloge. Le calcul d'un CRC consiste à générer un nombre pseudo-aléatoire en transmettant une donnée au LFSR, l'état final du LFSR représente la valeur du CRC. Cette méthodologie a été adaptée à nos besoins pour implémenter nos fonctions de hachage. Dans ce qui suit, nous décrivons la méthodologie en question et nous l'illustrons à l'aide de l'exemple de la figure 5-7.

La première étape de la méthodologie consiste à définir le LFSR à utiliser. Un LFSR se caractérise par son polynôme : on définit le polynôme grâce à la taille du LFSR (nombre de bascules), qui représente le degré le plus haut du polynôme, et la position des portes XOR, qui vont déterminer les coefficients binaires (0 ou 1) des monômes. Par exemple le degré du polynôme du LFSR de la figure 5-7 est 5. Nous avons donc le polynôme $P(x)=a_0 X^0+a_1X^1 +a_2X^2 + a_3X^3 + a_4X^4+ X^5$, avec a_0 et a_2 qui valent 1 car les portes XOR se trouve à l'entrée des bascules 1 et 3 (de gauche à droite). Le polynôme du LFSR représenté est donc $P(x)= X^5 + X^2 +1$.

Le degré du polynôme va déterminer le nombre de cases de la table de hachage, car l'état du LFSR représente le résultat de hachage. Ce dernier sera donc compris entre 1 et 2^M-1, avec M représentant la taille du LFSR. Aussi, la méthodologie suivie pour construire nos fonctions de hachage nécessite de définir la taille de la clé d'entrée du LFSR. Cette dernière correspond à la clé de hachage et puisque nous avons défini 8 groupes de préfixes, nécessitant 8 fonctions de hachage, nous devons reproduire cette méthodologie autant de fois.

La deuxième étape consiste à implémenter un générateur de CRC sériel avec ce LFSR pour construire une matrice, H1, qui sera utilisée par la suite pour construire notre fonction de hachage. Notons M le degré du polynôme et N la taille de la donnée en entrée. La matrice H1 décrit l'état de sortie du LSFR, M_{out}, en fonction de données en entrée, N_{in}, avec M_{in}=0 (état initiale du LFSR représenté par les valeurs initiales des bascules). La matrice a une taille de N*M. Nous allons prendre le LFSR illustré dans la figure 5-7 comme exemple, avec une entrée de 4 bits. Nous avons donc M=5 et N=4. Le tableau 5-3 représente le résultat de la construction de la matrice H1 avec un LFSR dont l'état initial est à 0 (M_{in}=0). Nous avons N valeurs N_{in} qui suivent un codage 1 parmi N (« One-Hot »), ce qui veut dire qu'un seul bit est à 1 pour chaque valeur N_{in} en entrée. Pour N=4, nous avons les valeurs N_{in} qui valent 0x1 (0b0001), 0x2 (0b0010), 0x4 (0b0100) et 0x8 (0b1000). Les résultats du tableau sont obtenus grâce à l'implémentation du générateur de CRC sériel : chaque ligne correspond à l'état final du LFSR (la valeur d'une case équivaut à la valeur finale d'une bascule), après avoir transmis la donnée séquentiellement (bit par bit) à l'entrée du LFSR.

Après la construction de la matrice, nous allons définir les équations du LSFR parallèle, avec M=5 et N=4. Chaque case *i* valant 1 d'une colonne *j* de la matrice H1 participe à l'équation du bit M_{out} [*j*] en tant que N_{in} [*i*]. On applique une porte XOR à tous les participants de l'équation pour avoir :

M_{out} [0] = N_{in} [0] ^ N_{in} [3]

$M_{out}[1] = N_{in}[1]$

$M_{out}[2] = N_{in}[0] \wedge N_{in}[2] \wedge N_{in}[3]$

$M_{out}[3] = N_{in}[1] \wedge N_{in}[3]$

$M_{out}[4] = N_{in}[2]$

Le vecteur M_{out} représente ainsi l'état final du LFSR. Nos fonctions de hachage reposent ainsi sur des opérations XOR entre des bits spécifiques de la clé de hachage, définis par un traitement antérieur. Par conséquent, ces fonctions ont l'avantage de bien s'implémenter en matériel, contrairement aux fonctions de hachage universelles, qui utilisent des opérations modulo (%), qui sont très couteuses en matérielle (modèle d'une fonction de hachage universelle [54]: h (clé) = [(A × clé + B) % P] % M) .

Tableau 6-3 : Matrice H1

M_{in}= 0	M_{out} [4]	M_{out} [3]	M_{out} [2]	M_{out} [1]	M_{out} [0]
N_{in}= 0x1	0	0	1	0	1
N_{in}= 0x2	0	1	0	1	0
N_{in}= 0x4	1	0	1	0	0
N_{in}= 0x8	0	1	1	0	1

Les tailles des clés des différentes fonctions de hachage ont été définies dans la section précédente. Cependant la définition des polynômes des différents LFSR dépend de l'implémentation visée. Le résultat de hachage pointe vers une mémoire (table de hachage), et puisque nous utilisons le hachage groupé, nous avons le choix d'implémenter une ou plusieurs tables de hachage (une par groupe).

L'utilisation de différentes tables de hachage entraîne davantage de collisions et dégrade la qualité de la distribution des préfixes [7], particulièrement si les groupes de préfixes contiennent un nombre de préfixes très différents. En conséquence, nous avons fait le choix d'utiliser qu'une seule table de hachage, qui sera stockée dans la mémoire cache. Les LFSR utilisés pour construire les

fonctions de hachage peuvent donc avoir le même polynôme. Avant de définir ce dernier, nous devons définir la correspondance entre la table de hachage et la mémoire cache. Nous discuterons des résultats obtenus dans le chapitre suivant.

6.2.3 Architecture

6.2.3.1 Correspondance

Dans l'architecture proposée, le cache stocke la totalité de la table de hachage. Chaque case de hachage est associée à un groupe (« set ») et chaque élément d'une case est stocké dans une ligne cache (« row »). La table de hachage définit ainsi la taille et l'associativité de la mémoire cache : le nombre maximal de collisions par « bucket » représente l'associativité de la mémoire cache et le nombre de cases de la table de hachage représente le nombre de groupes. La taille de la mémoire cache est alors l'associativité de la cache multipliée par le nombre de groupes, multipliés par la taille d'une ligne cache.

La figure 5-8 illustre comment un préfixe est stocké dans une ligne de mémoire cache. Pour un préfixe donné, on extrait la clé de hachage en fonction du groupe auquel il appartient et on la transmet à la fonction de hachage pertinente. La fonction de hachage produit le numéro du groupe (« set number ») où le préfixe sera stocké dans une de ses lignes. Chaque ligne de cache a un bit de validité (« V ») pour indiquer si la ligne est vide ou non, et contient le préfixe stocké, ainsi que sa taille et l'interface réseau associée. Le préfixe peut être stocké dans n'importe quelle ligne, tant que celle-ci est disponible (bit de validité à 0).

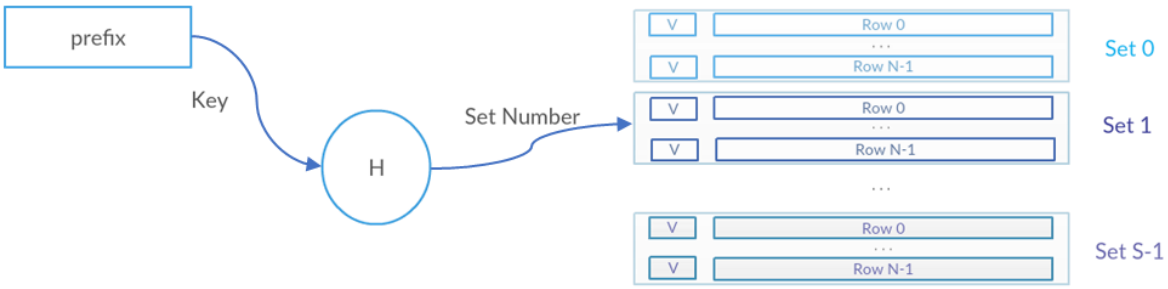

Figure 6-8: Correspondance d'un préfixe

Puisque les groupes de cache contiennent le même nombre de ligne de mémoire et que les groupes de préfixes possèdent un nombre de préfixes différents, il serait inadéquat d'associer un groupe de préfixes à un ensemble ; car cela entrainera une mauvaise distribution causée par les collisions. Chaque groupe de cache contient donc des préfixes de longueurs différentes.

6.2.3.2 Conception

- Implémentation

L'algorithme 5-1 décrit les différentes étapes pour effectuer la recherche IP. Pour chaque adresse IPv6 reçue, on extrait une clé pour chaque fonction de hachage, puisque nous utilisons le hachage groupé (nous avons défini 8 groupes de préfixes).

Pour chaque groupe, la fonction « Key_Generator » extrait une clé à partir de l'adresse IP, en sélectionnant les x bits les plus significatifs de l'adresse, où x est la limite inférieure de l'intervalle de taille d'un groupe de préfixes. Chaque clé est ensuite hachée par la fonction appropriée (« Lfsr_Index_Generator »). Pour chaque index généré, une recherche parallèle est effectuée dans toutes les lignes du groupe correspondant. Le masque de chaque préfixe (qui correspond à sa longueur) est appliqué à l'adresse IPv6 reçue, et une comparaison avec le préfixe est faite pour détecter si ce dernier correspond à l'entrée. Puisqu'un groupe de cache contient des préfixes de longueur différente, plusieurs correspondances sont possibles au sein d'un groupe ; la fonction « Look_For_Hit » retourne la meilleure correspondance.

Après avoir itéré au travers ces opérations pour chaque groupe de préfixe, la fonction « RechercheIP » retourne le préfixe le plus long parmi ceux retournés par la fonction « Look_For_Hit » durant le processus de recherche dans un groupe de cache.

La figure 5-9 montre comment la recherche IP est effectuée au sein d'un groupe pour le cas d'une mémoire cache dont l'associativité est de 4. Une fois qu'un ensemble de cache a été identifié grâce à la fonction de hachage, la recherche est effectuée dans les 4 lignes de cache. Cette implémentation correspond à la fonction « Look_For_Hit » de L'algorithme 5-1. Pour sélectionner le préfixe le plus long dans un groupe parmi ceux qui correspondent à l'adresse IP à rechercher, nous effectuons une comparaison binaire. Nous utilisons des encodeurs à priorité qui sélectionnent le meilleur résultat en comparant la taille des préfixes en entrée, par paires.

```
1: N indique le nombre de groupes de préfixes
2: index désigne l'index généré par la fonction de hachage
3: key indique la clé de hachage
4: match représente la structure de donnée qui contient la meilleure correspondance dans un
   ensemble
5: lpm indique le plus long préfixe
6: mask défini la taille d'un préfixe
7: result correspond au NHI retourné
8: function RECHERCHEIP(A)
9:     lpm = 0
10:    for i = 0 to N − 1 do
11:        key = Key_Generator(A, i);
12:        index = Lfsr_Index_Generator(key, i);
13:        match = Look_For_Hit(A, index);
14:        if match.hit then
15:            if lpm < match.mask then
16:                result = match.nhi;
17:                lpm = match.mask;
18:            end if
19:        end if
20:    end for
21: end function
```

Algorithme 6-1 : Recherche IP

Nous avons implémenté la fonctionnalité de la mémoire cache sur une carte FPGA ZC706 de Xilinx. L'implémentation sur FPGA nous permet de simuler un système (processeur-mémoire cache) de manière réaliste. En effet, le FPGA utilise des blocs BRAM pour stocker des données à l'intérieur du FPGA (« on-chip ») et elles sont accessibles en un cycle d'horloge, soit la latence habituelle entre un processeur est une mémoire cache.

Le langage VHDL est le langage de description matérielle majoritairement utilisé (avec le Verilog) dans l'industrie pour décrire la fonctionnalité des circuits intégrés. Cependant, nous avons utilisé le langage C pour concevoir notre solution, qui est un langage à plus haut niveau. La synthèse de la configuration FPGA du design a été faite avec l'outil « Vivado High-Level Synthesis » (Vivado HLS), de Xilinx. L'objectif de Vivado HLS est de permettre aux développeurs de concevoir, vérifier et optimiser leurs solutions, en bénéficiant d'un niveau d'abstraction plus élevé puisque l'outil génère automatiquement l'implémentation RTL (« Register Transfer Level ») du design. Les outils HLS gagnent du terrain dans l'industrie et dans le milieu académique car ils visent à réduire le temps de conception (via une description algorithmique).

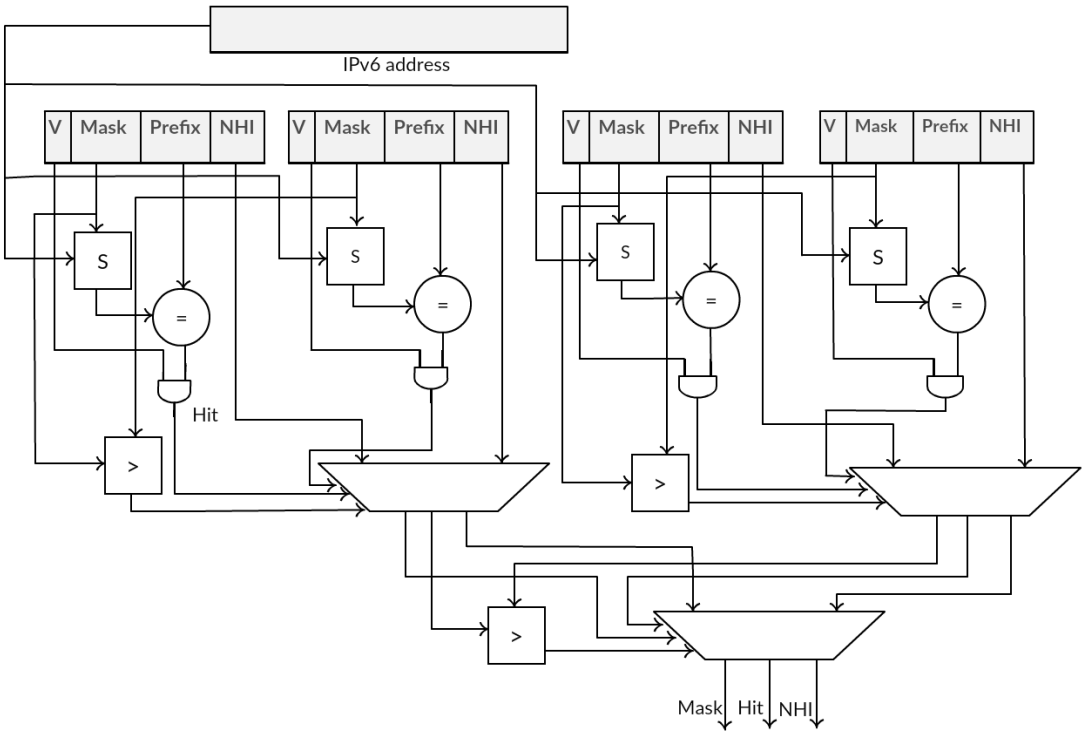

Figure 6-9: Recherche IP dans un ensemble de cache

- Optimisation

L'optimisation du « design » comprend la réduction de la latence, l'amélioration du débit (« throughput ») et la réduction des ressources physiques utilisées. La latence est le nombre de cycle nécessaire pour générer un résultat, alors que le débit est le nombre de cycles entre deux nouveaux résultats. Néanmoins, la maîtrise de l'outil HLS nécessite un temps d'apprentissage important. Un style de codage approprié est nécessaire pour bénéficier des avantages de la synthèse HLS, et la vérification et l'analyse du RTL généré est une étape indispensable pour avoir un résultat performant (évidement, l'utilisation d'un langage comme le VHDL permet d'être moins dépendant de l'outil de synthèse). Nous allons présenter quelques optimisations importantes pour améliorer la performance de n'importe quel design.

L'outil Vivado HLS exploite de lui-même le parallélisme des fonctions et des opérations. Néanmoins, nous devons optimiser nous-même le « design » en adoptant un style de codage permettant de limiter la dépendance des données, et en donnant à l'outil des directives pour produire une microarchitecture qui réponde aux objectifs de performance souhaités. Par exemple,

la directive #pragma HLS latency min= max=8 permet de définir la latence minimal et maximal souhaitées (réalisable dans la mesure du possible).

Les boucles (« loops ») sont des bons candidats à optimiser. Par défaut, elles sont roulées (« rolled »), comme illustré dans la figure 5-10.

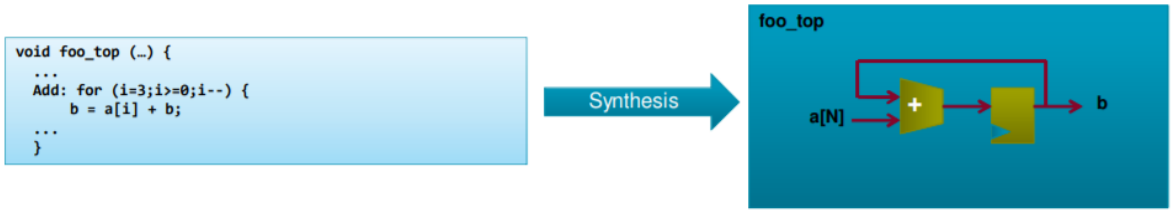

Figure 6-10: Synthèse par défaut d'une boucle *for* [48]

Cette boucle a une latence de 4 cycles. Si le délai de l'additionneur est petit comparé à la fréquence d'horloge, nous pouvons réduire la latence en déroulant la bouche, à l'aide de la directive : #pragma HLS unroll. Ainsi, nous aurons le RTL illustré dans la figure 5-11:

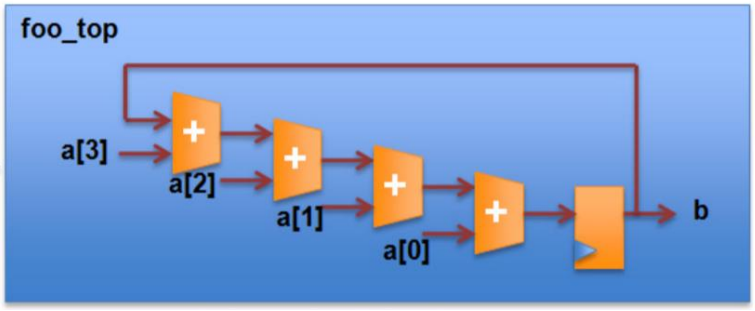

Figure 6-11: Optimisation d'une boucle *for* [48]

La boucle peut donc être exécutée en un seul cycle (si la contrainte de temps est raisonnable). Cependant les éléments du tableau doivent être accessibles en parallèle, sinon on ne peut tirer aucun avantage à dérouler une boucle.

La directive #pragma HLS pipeline permet l'exécution d'instructions découpées en plusieurs étapes. Elle est appliquée aux fonctions et aux boucles pour améliorer le débit et la latence, comme le montre la figure 5-12. Dans l'exemple donné, la boucle comprend trois instructions (op_READ, op_COMPUTE, op_WRITE). À l'aide de la direction #pragma HLS pipeline, l'architecture est décrite en pipeline où chaque étage (étape) exécute une instruction. Ainsi, lors du deuxième cycle d'horloge, la première instruction de la deuxième itération de la boucle (op_READ) pourra être

exécutée pendant l'exécution de la deuxième instruction de la première itération de la boucle (op_COMPUTE).

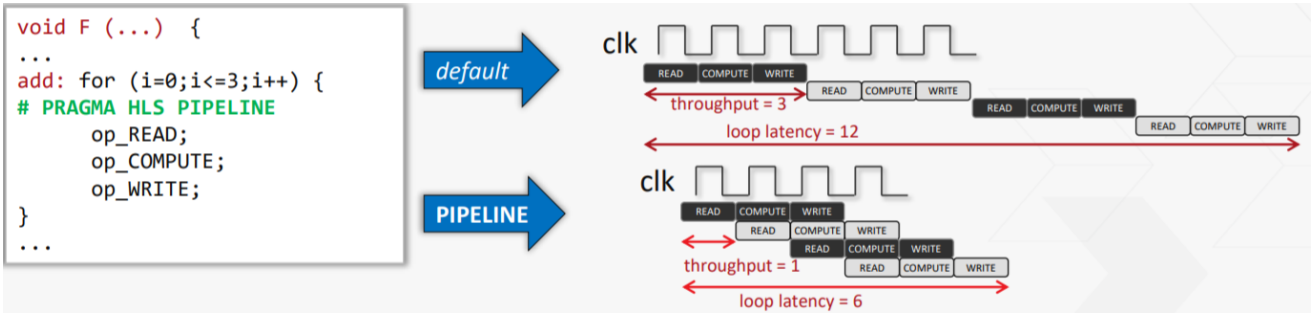

Figure 6-12: Optimisation de la latence et du débit [49]

Les accès aux tableaux sont souvent le goulot d'étranglement de performance. En effet, les tableaux sont implémentés en mémoire (exemple : BRAM) où une donnée est accédée en un cycle d'horloge. Vivado HLS permet de partitionner un tableau en sous-tableaux indépendants, afin de pouvoir accéder aux données de manière parallèle. Plusieurs types de partition sont possibles, comme démontré dans la figure 5-13:

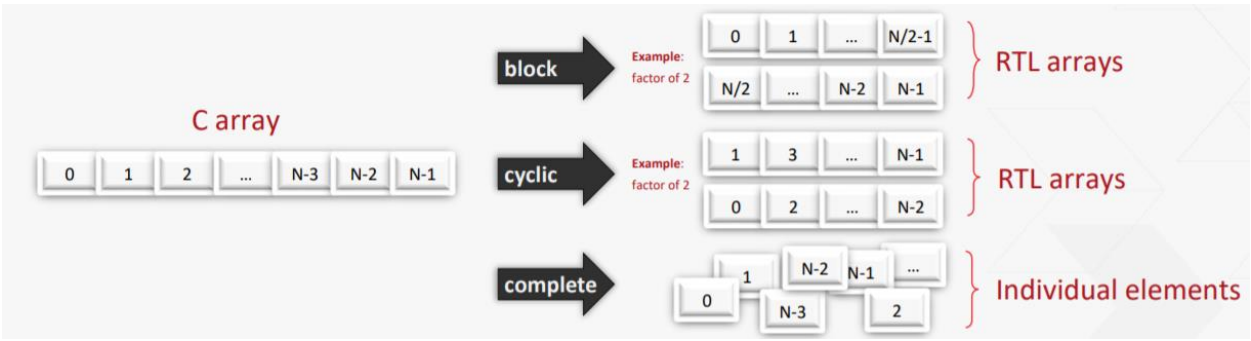

Figure 6-13: Partitionnement de la mémoire [49]

Les matrices aussi peuvent être partitionnées, par exemple, la matrice tab[2][3][4] peut être partitionné en :

- 4 matrices (dimension de la partition égale à 3) : tab_0[2][3], tab_1[2][3], tab_2[2][3] et tab_3[2][3]
- 3 matrices (dimension de la partition égale à 2) : tab_0[2][4], tab_1[2][4] et tab_2[2][4]
- 2 matrices (dimension de la partition égale à 1) : tab_0[3][4] et tab_1[3][4]

- Ou encore en 2*3*4= 24 registres (dimension de la partition égale à 0)

Néanmoins, la synthèse HLS a des limitations comme : les pointeurs de fonctions qui ne sont pas supportés, les fonctions récursives qui ne sont pas synthétisables, les boucles à itérations variable qui ne peuvent pas être optimisées etc. Aussi, un mauvais codage peut entrainer des situations où l'optimisation est compliquée à réaliser comme avec le code suivant (figure 5-14) :

```
#include "array_mem_bottleneck.h"

dout_t array_mem_bottleneck(din_t mem[N]) {

      dout_t sum=0;
      int i;

      SUM_LOOP:for(i=2;i<N;++i)
        sum += mem[i] + mem[i-1] + mem[i-2];

      return sum;
}
```

Figure 6-14 : code original [50]

Même si la directive #pragma HLS pipeline est appliquée à la boucle pour avoir un débit à 1, cette optimisation n'est pas réalisable car on doit faire 3 accès mémoire à chaque itération. Ce code doit être exprimé comme à la figure 5-15 pour atteindre un débit à 1 suite à l'utilisation de la directive #pragma HLS pipeline :

```
#include "array_mem_perform.h"

dout_t array_mem_perform(din_t mem[N]) {

      din_t tmp0, tmp1, tmp2;
      dout_t sum=0;
      int i;

      tmp0 = mem[0];
      tmp1 = mem[1];
      SUM_LOOP:for (i = 2; i < N; i++) {
             tmp2 = mem[i];
             sum += tmp2 + tmp1 + tmp0;
             tmp0 = tmp1;
             tmp1 = tmp2;
      }

      return sum;
}
```

Figure 6-15 : code optimisé [50]

CHAPITRE 7 ÉVALUATION DE LA CACHE

7.1 Résultats

7.1.1 Structure de données

Une distribution équilibrée des fonctions de hachage, traduite par des tailles de « buckets » raisonnablement égales, est nécessaire pour avoir un taux d'utilisation de la mémoire satisfaisant. Nos fonctions de hachages ont été implémentées grâce à des opérations XOR entre des bits spécifiques de la clé de hachage (définis par un prétraitement, présenté dans le chapitre 5). Pour évaluer la distribution de nos fonctions de hachage, nous avons implémenté en logiciel des fonctions de hachage universelles, réputées pour générer peu de collisions, et nous avons comparé la distribution de notre solution avec celles des fonctions de hachage universelles.

Une fonction de hachage universelle [51] est décrite comme suit : h (*clé*) = [(A × *clé* + B) % P] % M, où A, B, P et M sont des nombres entiers positifs constants, respectant les conditions suivantes :

A ∈ [1, P – 1]

B ∈ [0, P – 1]

M ≡ taille de la table de hachage

P : Nombre premier et P > M

La fonction de hachage universelle garantit que, pour deux clés différentes x et y, la probabilité d'une collision est inférieure ou égale à 1 / M, permettant ainsi de contrôler le nombre de collisions. Les fonctions de hachage universelles ne s'implémentent pas bien en matériel car elles reposent sur des opérations de base modulo ; on les utilise ici à titre de comparaison.

En utilisant la même base de données (rrc00), nous avons généré 100 000 fonctions de hachage (combinaisons de paramètres A, B et P) de manière pseudo-aléatoire, et nous les avons évaluées en fonction de leurs distributions. Le but est de sélectionner la fonction de hachage la plus performante pour l'utiliser comme élément de comparaison. La figure 6-1 représente le nombre de fonctions ayant produit le même nombre maximal de collision dans un « bucket ».

N.B. Puisque nous utilisons le hachage groupé, une fonction de hachage comprend en réalité 8 fonctions de hachage.

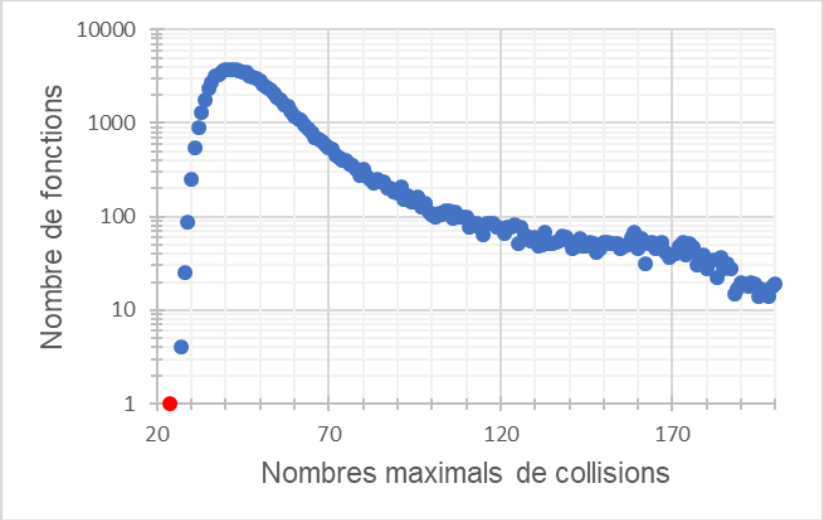

Figure 7-1 : Collisions des fonctions de hachage universelles

La figure 6-2 illustre la meilleure distribution obtenue avec nos évaluations. Avec M constant et égal à 2048, la fonction de hachage la plus performante génère 24 collisions au maximum dans une case de la table de hachage. En plus de la génération pseudo-aléatoire, nous avons utilisé NOMAD [52], un algorithme d'optimisation conçu par des professeurs de Polytechnique Montréal pour optimiser les fonctions de hachage universelles. NOMAD permet de définir les paramètres A, B et P afin de minimiser les collisions.

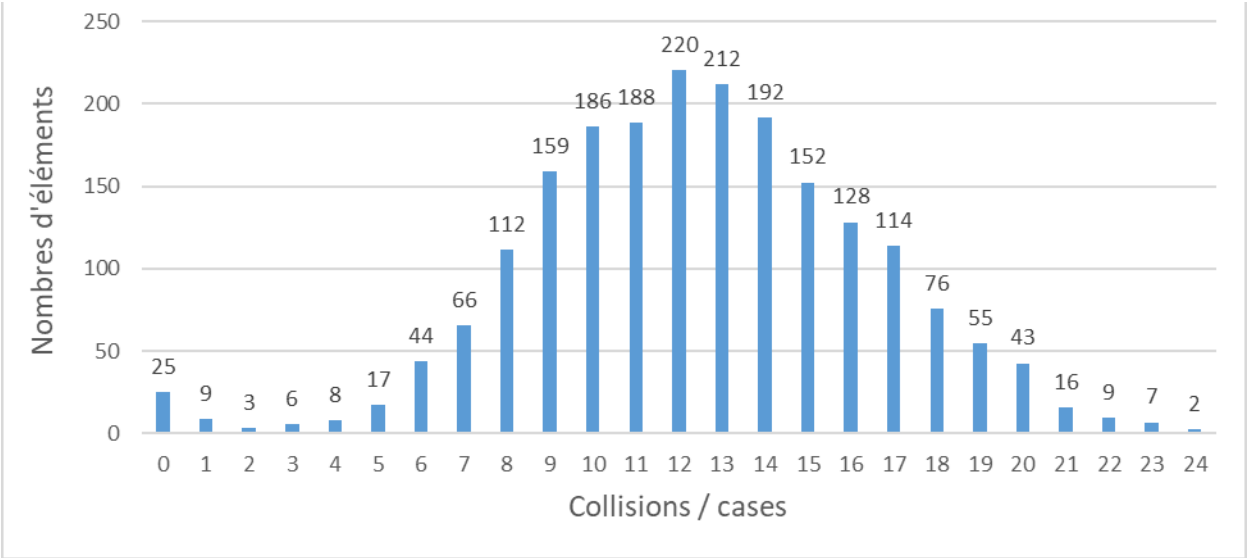

Figure 7-2: Distribution de la fonction de hachage universelle la plus performante

Nous avons construit nos fonctions de hachage à l'aide d'un LFSR de 11 bits pour pouvoir implémenter une table de hachage avec 2048 « buckets » (équivalent à la constante M de nos fonctions de hachage universelles). Le polynôme $P(x)= X^{11} + X^9 + X^8 + X^7 + X^2 +1$ est utilisé pour la génération de CRC-11 (standard pour la génération d'une valeur CRC de 11 bits). Nous avons utilisé ce dernier pour construire nos fonctions de hachage. La figure 6-3 représente la distribution des préfixes obtenue avec nos fonctions de hachage. Nous obtenons 28 collisions au maximum dans un « bucket », et puisque le meilleur résultat obtenu avec nos évaluations (génération aléatoire et optimisation NOMAD) est de 24, nous considérons le résultat obtenu satisfaisant. La distribution obtenue suit une courbe gaussienne, synonyme d'une distribution aléatoire [53], ce qui représente le but primaire d'une fonction de hachage.

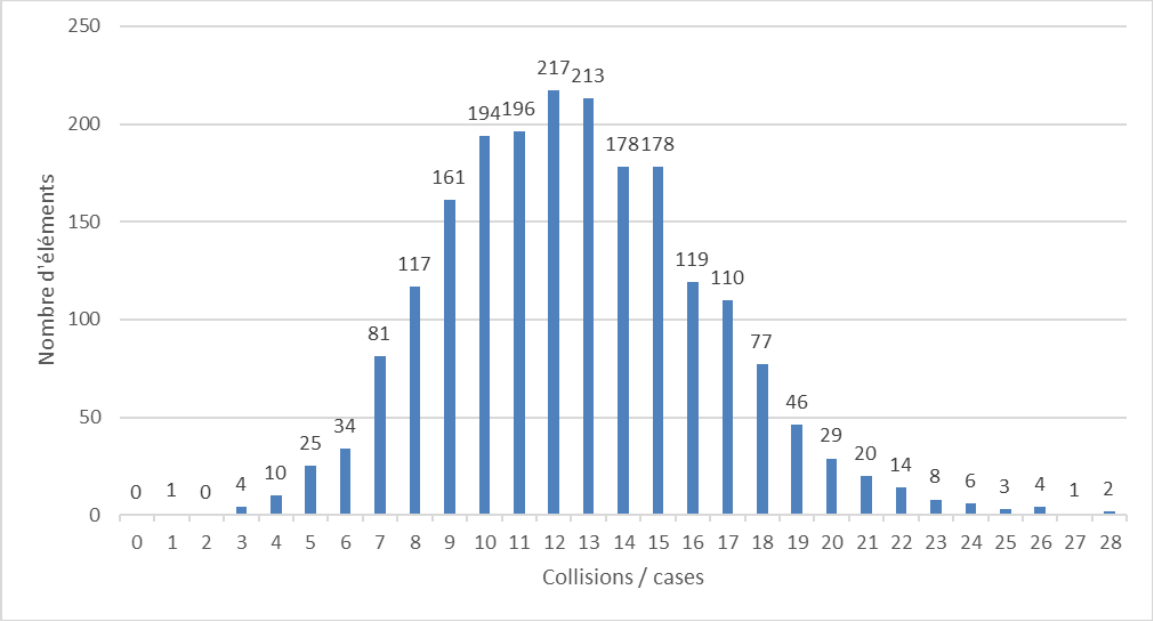

Figure 7-3: Distribution de nos fonctions de hachage

Chaque case de la table de hachage est stockée dans un groupe cache de 32 lignes de 128 bits (associativité de 32). En cas de collision dans un « bucket », nous stockons le préfixe dans une des lignes de cache disponibles du groupe indexé par la fonction de hachage. Une associativité de 32 permet de soutenir le nombre maximal de collisions et une taille de 128 bits permet de stocker le préfixe, sa longueur et l'interface réseau associée (la taille d'un bloc mémoire doit être une puissance de 2). Pour une table de 25874 préfixes, la taille de cache requise est de 1 Mo. Étant donné que nous avons besoin de 128 bits pour stocker une entrée, le meilleur résultat possible est une taille mémoire de 512 Ko, ce qui correspond à la moitié de la taille atteinte par l'architecture

proposée. La taille mémoire pour stocker notre structure de données est raisonnable car une fonction de hachage entraîne nécessairement des « buckets » de tailles différentes, réduisant la proportion de l'utilisation mémoire.

7.1.2 Latence et débit

Une lecture mémoire est requise pour chaque itération de l'algorithme 5.1 (pour lire les préfixes stockés dans une ligne de cache). De ce fait, les itérations de l'algorithme ne peuvent se faire en parallèle, car on ne peut effectuer qu'une seule lecture en mémoire BRAM par cycle d'horloge. Rappelons que l'exploitation du parallélisme des données est le bénéfice majeur espéré des implémentations matérielles.

La boucle de l'algorithme est mise en pipeline afin d'avoir une architecture permettant de faire la lecture mémoire dans le premier étage de pipeline et le reste du traitement sur un nombre d'étages calculé par l'outil. Le nombre d'étage de pipeline représente la latence de la boucle. De ce fait, la lecture mémoire de la deuxième itération peut se dérouler dès le deuxième cycle d'horloge. La mise en pipeline de la boucle de l'algorithme permet de réduire la latence de ce dernier. La latence de notre mémoire cache pour effectuer une opération de recherche IP est de 11 cycles à une vitesse d'horloge de 156.25 MHz.

Nous avons défini la mémoire comme une mémoire BRAM à double port pour effectuer deux accès mémoire par cycle d'horloge. Puisque nous devons faire huit lectures pour une seule recherche IP et que nous ne pouvons effectuer que deux lectures par cycle, l'intervalle entre deux opérations de recherche est de 4 cycles au minimum. Ainsi, lorsqu'une adresse IP est reçue, il faut 11 cycles à notre cache pour transmettre le paquet IP, et 4 cycles pour transmettre chacun des suivants. La mémoire cache est donc capable de soutenir 39 millions d'opérations de recherche par seconde (« million lookups per second », Mlps).

Puisqu'un paquet IPv6 a une taille minimale de 84 octets, le débit maximum correspondant est de 26.25 Gbps. Cependant, un tel système (processeur et cache) est normalement implémenté dans un ASIC avec une fréquence d'horloge d'au moins 1 GHz. Une version ASIC de cette architecture pourrait ainsi être au moins 6.4 fois plus rapide, auquel cas elle prendrait en charge une bande passante de 168 Gbps, suffisante pour quatre liaisons physiques de 40 Gbps ou 16 liaisons de 10 Gbps. Le Tableau 6-1 résume les résultats temporels.

Tableau 7-1 : Résultats temporels

Fréquence (Mhz)	Latence (Cycles)	Intervalle (Cycles)	Recherche (Mlps)	Débit (Gbps)
156.25	11	4	39	26.25

Les résultats obtenus permettent de satisfaire les besoins des routeurs de périphérie (« edge routers ») qui ont des exigences de performance beaucoup moins strictes par rapport à celles des « core routers ».

7.1.3 Ressources physiques

Afin d'évaluer la consommation des ressources physiques de notre solution, nous avons implémenté une mémoire cache classique (tel qu'utilisé par les processeurs) et nous avons comparé les ressources consommées par la cache classique et notre solution de cache, pour estimer le supplément de logique nécessaire pour supporter la recherche IP. Le tableau 6-2 présente ces résultats.

Tableau 7-2 : Consommation des ressources physiques

	LUT	FF	BRAM (18 Kb)
Cache classique	12294	3995	512
Cache supportant la recherche IP	22809	8856	512

Dans un FPGA, un bloc logique est de manière générale constitué d'une table de correspondance (« Look-Up-Table », LUT), qui sert à implémenter des équations logiques, et d'une bascule (« Flip-Flop », FF), permettant de mémoriser un état ou de synchroniser un signal. Le surplus de logique requise nous semble raisonnable, car nous avons décrit une architecture pipelinée, et le traitement LPM est **beaucoup** plus complexe que l'« exact match » (une cache classique effectue uniquement

une comparaison de « tag »). De plus, les blocs mémoires (BRAM) occupent une grande part des ressources physiques et leur nombre reste inchangé en fonction des implémentations, ce qui réduit la proportion du surplus de la logique nécessaire pour supporter la recherche IP.

7.2 Plateforme de prototypage

La fonctionnalité de notre solution a été testée sur une plateforme de prototypage conçue par notre laboratoire de recherche, à savoir : Jeferson Santiago da Silva, Thibaut Stimpfling, Thomas Luinaud, Imad Benacer et moi-même Bachir Fradj. Le but de notre collaboration était la mise en œuvre d'une plateforme flexible et programmable pour la conception et le test de futures fonctionnalités réseaux. En fournissant une bibliothèque de modules, comprenant des fonctions de réseau, des interfaces et un générateur de paquets, nous pouvons combiner des modules pour simuler des fonctionnalités spécifiques. La carte FPGA ZC706 de Xilinx a servi de support physique, où la partie logique programmable (« Programmable Logic ») implémente des fonctions types d'un plan de données, qui nécessitent un traitement rapide (« Fast Path »). La carte ZC706 contient également deux processeurs ARM Cortex-A9 embarqués, qui permettent la configuration des modules physiques implémentés et la récolte de statistiques (pas d'exigence de vitesse, « Slow Path »). La figure 6-4 illustre l'architecture de notre plateforme.

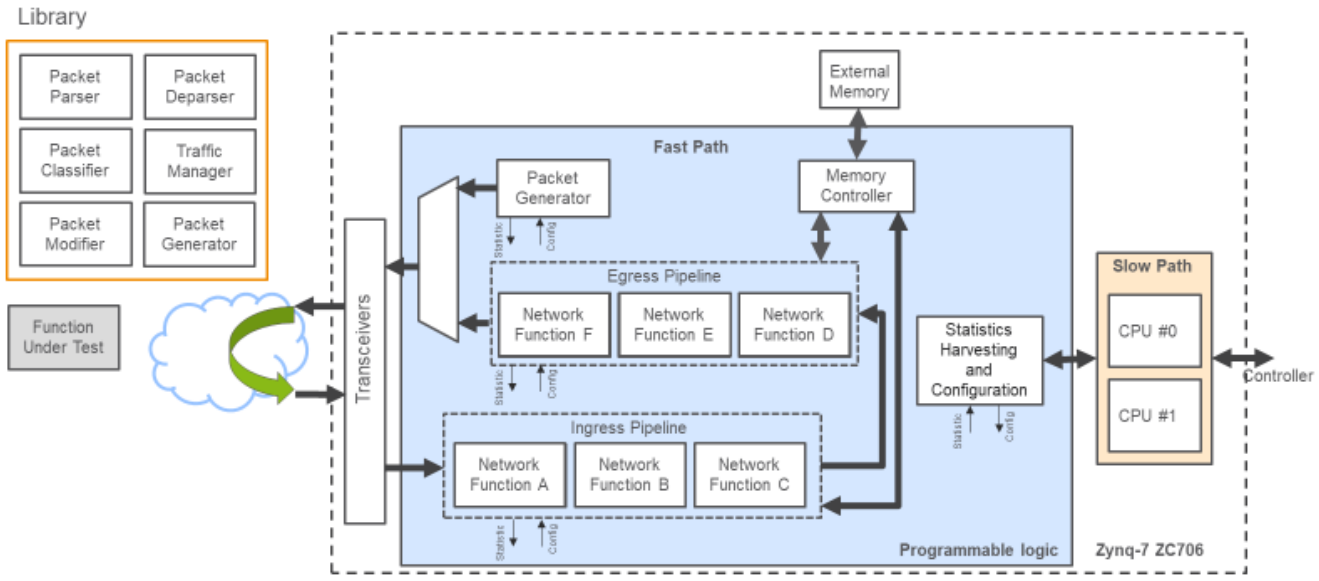

Figure 7-4: Architecture de la plateforme de prototypage

- Banc d'essai

Nous avons implémenté les modules nécessaires pour tester la fonctionnalité de notre mémoire cache, comme montré dans la figure 6-7, à savoir :

- Un générateur de paquets configurables (« Packet Generator ») : qui permet de générer divers types de flux. Puisque nous ne traitons pas les cas de « cache miss », nous générons uniquement des adresses IPv6 qui correspondent forcement à une entrée de cache. Le générateur de paquets a été conçu par l'outil de synthèse à haut niveau (HLS) de Xilinx, et il génère des paquets IP avec des adresses IP pseudo-aléatoires grâce à des LFSR. Le module transmet les paquets par segments de 64 bits par cycle d'horloge de 156.25 MHz. Ce débit a été calculé pour correspondre à celui du « transceiver » implémenté. L'interface du module a aussi été définie pour correspondre à celle du « transceiver ».
- Un « Transceiver » : qui représente l'interface d'entrée et de sortie des paquets, permettant ainsi de communiquer avec le monde extérieur. Le « transceiver » transmet les paquets à 10 Gb/s avec un bus de 64 bits, définissant ainsi notre domaine d'horloge : $10^9/64$ = 156.25 MHz. Cette interface a été implémentée en utilisant l'IP de générique de Xilinx. Cette IP fournit une interface avec le contrôleur qui permet au concepteur d'envoyer des requêtes vers le « transceiver ». Le contrôleur possède une interface de type AXI4-Stream. Le tableau 6-3 présente les signaux de synchronisation d'une interface AXI4-Stream.

  Tableau 7-3 : Signaux de l'interface AXI4-Stream

Signal	Description
TVALID	Signal à '1' lorsque la donnée est valide
TREADY	Signal à '1' pour indiquer que le « transceiver » est opérationnel
TKEEP	Masque de données
TLAST	Dernier 64 bits du transfert courant
TUSER	Bus permettant de transmettre des signaux spécifiques à l'application

- Un « Parser » : dont le rôle est d'extraire les adresses IPv6 de destination à partir des paquets qui le traversent, sous forme de segments de 64 bits. Aussi connu sous le nom d'analyseur de paquets, le « Parser » supporte les protocoles de transport UDP et TCP et possède une interface de type AXI4-Stream pour être synchronisé avec le « transceiver ». Le « Parser » représente ainsi une machine à état, capable de détecter les champs des différents entêtes et retourne ceux déterminés par le concepteur. Dans notre cas, nous extrayons les champs pour une classification traditionnelle ; à savoir les adresses IP, les ports et le protocole de transmission.
- Une table de recherche « LPM LUT » : qui est notre mémoire cache. Elle joue le rôle d'une table de recherche ; elle retourne l'interface associée à l'adresse IPv6 de destination du paquet. Rappelons que notre solution a été conçue en utilisant Vivado HLS, qui permet de générer une IP avec une interface standard de type ap_ctrl_hs, dont les signaux sont représentés dans la figure 6-5. Le module commence son opération lorsque le signal ap_start est à '1'. Le signal ap_ready indique que le module est prêt à recevoir de nouvelles entrées au prochain cycle d'horloge et le signal ip_idle indique que le module est en cours de traitement. Finalement, le signal ap_done indique que la sortie est valide. C'est le « Parser » qui indique à la mémoire cache de commencer son traitement (après avoir extrait l'adresse IP de destination).

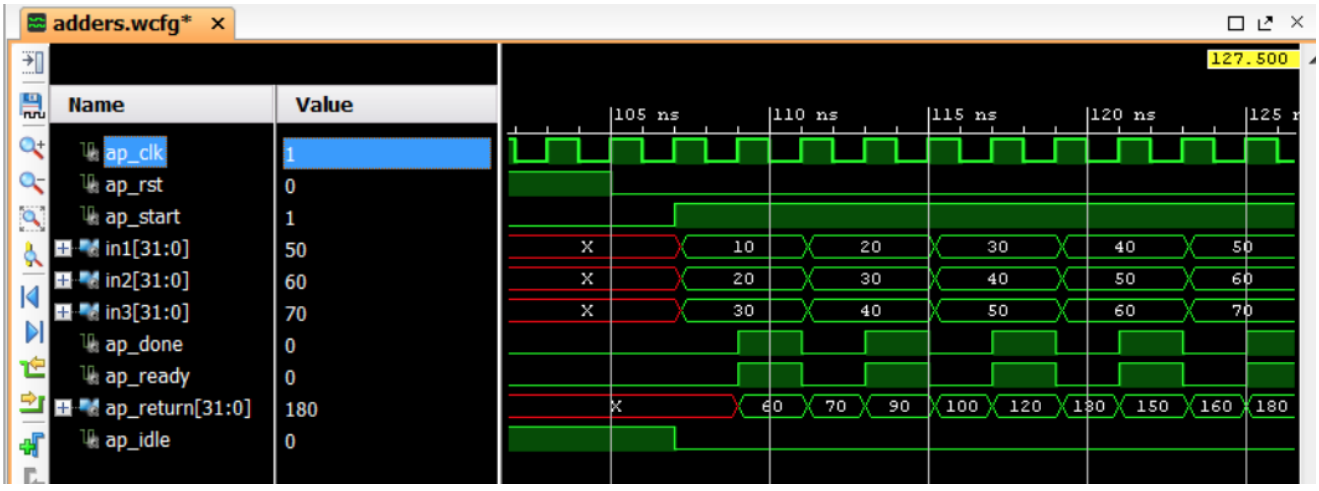

Figure 7-5: Signaux du protocole ap_ctrl_hs

- Un contrôleur de mémoire externe (« Memory Controller ») : Il s'agit de l'interface avec la mémoire DDR3 à grande capacité, implémentée physiquement sur la carte (hors du FPGA). La figure 6-6 illustre l'interface entre la DDR3 et le FPGA.

  Nous chargeons notre mémoire cache à partir de la DDR et pour respecter le concept du « caching », nous chargeons les adresses IP les plus fréquentes.

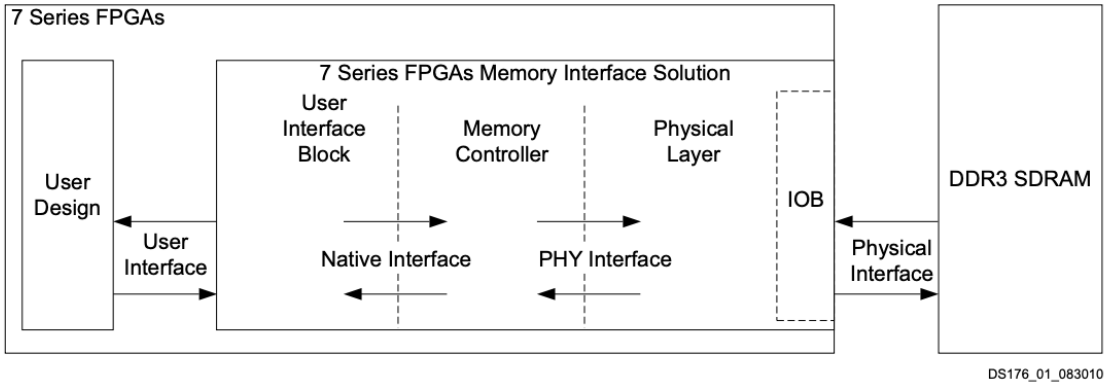

Figure 7-6: Interface entre le FPGA et la DDR3

- Un CPU : dont le rôle est de contrôler la génération des paquets et colleter les statistiques afin d'analyser nos résultats. Les statistiques sont générées à l'aide de compteurs implémentés dans le FPGA. Le processeur possède une interface de type AXI-MM (« Memory Mapped ») avec le FPGA, cela signifie que le FPGA voit le processeur comme une mémoire, où il envoie des requêtes de type lecture/écriture.

Nous commençons par charger une partie de la table de routage dans la mémoire cache à partir de la mémoire externe. Une machine à état est responsable d'envoyer des requêtes de lecture vers la DDR3 pour écrire les préfixes lus dans la mémoire cache. Nous envoyons par la suite une requête vers le générateur de paquets pour qu'il commence sa génération de paquets avec des adresses IP pseudo-aléatoires, mais qui ne provoqueraient pas tout de même de défauts de cache. Toutes interactions avec les modules implémentés se fait grâce au processeur embarqué, qui offre une certaine souplesse. Cette flexibilité et la séparation entre le « control plan » et le « data plan » permettent d'être en adéquation avec les principes majeurs de SDN.

N.B : Nous générons des paquets de taille minimale (84 bytes) pour couvrir le « worst case », ce qui impacte négativement notre débit.

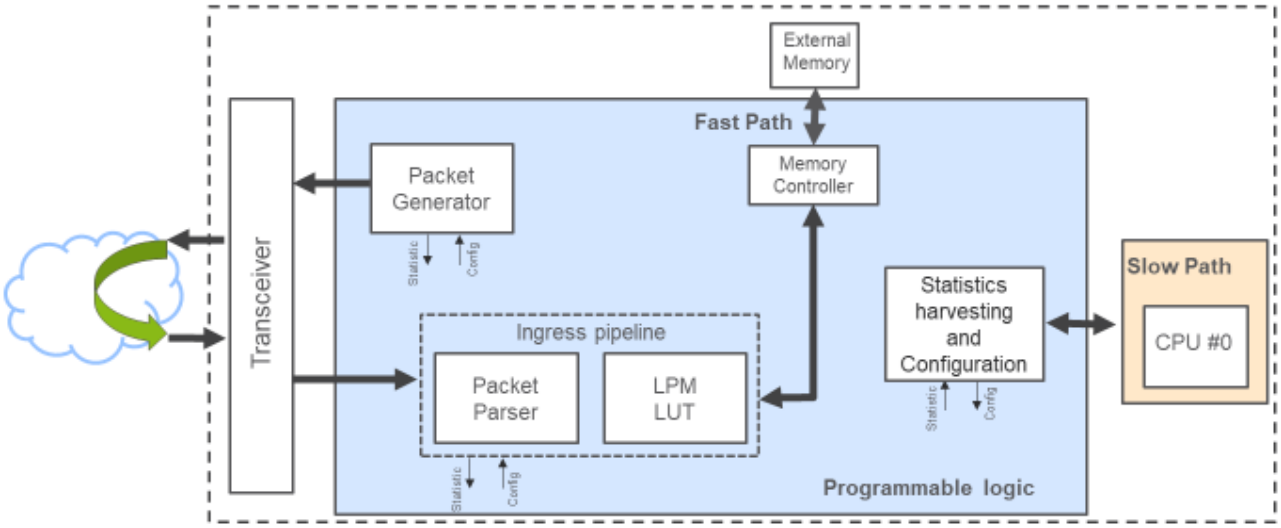

Figure 7-7: Architecture du banc d'essai fonctionnel

Les paquets traversent le « Transceiver » en effectuant un « loop back » physique (cela permet de simuler la sortie puis l'entrée du paquet du FPGA) pour arriver au « Parser » qui extrait l'adresse IP de destination du paquet. Notre cache (« LPM LUT ») prend cette dernière en entrée puis retourne le NHI associé. Si l'adresse IP correspond au minimum à un préfixe stocké, on incrémente le compteur des succès de cache (« cache hit »).

Le processeur CPU récolte les compteurs pour avoir les résultats de recherche. Nous n'obtenons aucun défaut de cache et nous analysons les NHI retournés afin de vérifier qu'ils correspondent bien au plus long préfixe. Pour cela, on émule une TCAM (on implémente une « lookup talbe » en logiciel) et pour chaque adresse IP, on compare le résultat retourné par notre cache avec celui calculé au en logiciel.

Ce banc de test permet de valider la fonctionnalité de notre solution et de démontrer de manière évidente que nous sommes en mesure de supporter un trafic de 10 Gb/s, en simulant au mieux un système de réseau. L'implémentation de la figure 6-7 permet de charger la mémoire cache à partir d'une mémoire à plus grande capacité mais plus lente (la DDR a une latence importante et variable), puis elle réalise des recherches IP pour retourner le NHI, qui peut être utilisé pour divers application comme l'acheminement de paquets ou la classification.

CHAPITRE 8 CONCLUSION

Aujourd'hui, les réseaux informatiques ont des besoins croissants de performance, dus à l'émergence d'applications de plus en plus gourmandes en bande passante, et des besoins de programmabilité, afin de supporter davantage de services. Les processeurs réseau fournissent la flexibilité désirée et offrent une performance intéressante.

Dans ce mémoire nous avons proposé une solution permettant aux processeurs de réseau de s'affranchir de l'utilisation de mémoires coûteuses, les TCAM, responsables de la recherche IP. Nous présentons une mémoire cache basée sur le hachage permettant d'exploiter les ressources physiques des NP et de la localité temporelle présente dans un trafic réseau.

Dans l'architecture proposée, nous utilisons le hachage groupé pour résoudre le problème des « wildcard » et nous définissons des fonctions de hachage efficaces pouvant être facilement implémentées en matériel. La méthode proposée produit une bonne distribution de préfixes permettant de charger une partie de la table de routage dans une mémoire cache. Atteignant une performance de recherche avoisinant les 40 Mlps, nous avons montré la capacité d'étendre la fonctionnalité d'une mémoire cache, pouvant être utilisée comme solution de recherche IP capable de transmettre des paquets à une vitesse de 160 Gbps.

Même si nous avons obtenu des résultats de transmission convenables, l'objectif principal de notre travail n'est pas orienté vers le critère de performance. En définissant une correspondance de données (« mapping ») appropriée et en étendant la fonctionnalité de recherche, nous avons montré que notre cache représente une solution de recherche IP. Notre travail permet de prouver le concept et l'intérêt d'avoir une mémoire cache capable d'offrir un type de traitement différent. Ceci représente un réel défi, car la mémoire cache est un paradigme bien ancré et difficile à faire évoluer.

La performance de notre solution peut évidemment être améliorée en maîtrisant mieux l'outil de synthèse Vivado HLS ou en décrivant notre architecture avec un langage plus axé vers la performance comme le VHDL. Mais surtout, notre travail doit être complété en proposant des politiques de cache permettant de charger dynamiquement la mémoire pour permettre notamment les mise à jour.

RÉFÉRENCES

[1] A. Sivarman Kaushalram, “Designing fast and programmable routers”, Doctoral dissertation, Massachusetts Institute of Technology, 2017.

[2] M.A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous, “Survey and taxonomy of IP address lookup algorithms”, IEEE network, vol. 15, no.2, pp. 8-23, 2001.

[3] M. Tahir and S. Ahmed, “Tree-Combined Trie: A Compressed Data Structure for Fast IP Address Lookup”, International Journal of Advanced Computer Science and Applications, vol. 6, no. 12, 2015.

[4] D. E. Turner, “Survey and Taxonomy of Packet Classification Techniques”, ACM Computing Surveys, vol. 37, no. 3, pp. 238–275, 2005.

[5] H. Liu, “Routing prefix caching in network processor design”, International Conference on Computer Communications and Networks, October 2001.

[6] Hyesook Lim, Ji-Hyun Seo and Yeo-Jin Jung, “High speed IP address lookup architecture using hashing”, IEEE Communications Letters, vol. 7, no. 10, pp. 502-504, 2003.

[7] M. Hanna, S. Demetriades, S. Cho, and R. Melhem, “Advanced hashing schemes for packet forwarding using set associative memory architectures”, Journal of Parallel and Distributed Computing, vol. 71, no. 1, pp. 1-15, 2011.

[8] P. Kumar, “Match-Action Tables and P4 Runtime Control”, Cornell-pl.github.io, 2018. [Online]. Available: https://cornell-pl.github.io/cs6114/download/lecture10.pdf. [Accessed: 24- Jun- 2019].

[9] R. Giladi, Network Processors. 2008.

[10] A. Baumann and B. Fabian, “Who Runs the Internet? Classifying Autonomous Systems into Industries”, International Conference on Web Information Systems and Technologies · WEBIST, 2014.

[11] W. Goralski, Illustrated Network. Elsevier Science, 2009.

[12] D. Serpanos and T. Wolf, Architecture of Network Systems. Saint Louis: Elsevier Science, 2014.

[13] D. Kreutz, F. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky and S. Uhlig, “Software-Defined Networking: A Comprehensive Survey”, Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, 2015.

[14] N. Feamster, J. Rexford and E. Zegura, “The road to SDN”, ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 87-98, 2014.

[15] J.Wolfgang, et al, “Split architecture for large scale wide area networks”, 2014.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, “The Click Modular Router”, ACM Transactions on Computer Systems (TOCS), 2000, vol. 18, no 3, p. 263-297.

[17] Intel IXP2800 Network Processor. [Online]. Available: http://www.ic72.com/pdf_file/i/587106.pdf. [Accessed: 24- Jun- 2019].

[18] S. Han, K. Jang, K. Park, and S. Moon, “PacketShader: A GPU-accelerated Software Router”, SIGCOMM, 2010.

[19] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, “NetFPGA SUME: Toward Research Commodity 100Gb/s”, IEEE Micro, 2014.

[20] NP-5 Network Processor. [Online]. Available: http://www.mellanox.com/related-docs/prod_npu/PB_NP-5.pdf. [Accessed: 24- Jun- 2019].

[21] Choi, Sean, et al, “FBOSS: building switch software at scale”, ACM, 2018.

[22] A. Hakiri, A. Gokhale, P. Berthou, D. Schmidt and T. Gayraud, “Software-Defined Networking: Challenges and research opportunities for Future Internet”, Computer Networks, vol. 75, pp. 453-471, 2014.

[23] E. Kaljic, A. Maric, P. Njemcevic and M. Hadzialic, “A Survey on Data Plane Flexibility and Programmability in Software-Defined Networking”, IEEE Access, vol. 7, pp. 47804-47840, 2019.

[24] P. Bosshart et al., “Forwarding metamorphosis”, ACM SIGCOMM Computer Communication Review, vol. 43, no. 4, pp. 99-110, 2013.

[25] Wang, Han, et al., “P4fpga: A rapid prototyping framework for p4”, ACM, 2017.

[26] C. Kim, “Programming The Network Data Plane”, Qconsf.com, 2017. [Online]. Available: https://qconsf.com/sf2017/system/files/presentation-slides/qcon_-_2017_11_15_-_chang_kim_-_public.pdf. [Accessed: 24- Jun- 2019].

[27] Kim, Daehyeok, et al., “Generic External Memory for Switch Data Planes”, ACM, 2018.

[28] M. Ring, D. Schlör, D. Landes and A. Hotho, “Flow-based network traffic generation using Generative Adversarial Networks”, Computers & Security, vol. 82, pp. 156-172, 2019.

[29] W. Wang, L. Dong, B. Zhuge, M. Gao, F. Jia, R. Jin, J. Yu, and X. Wu, “Design and implementation of an open programmable router compliant to IETF ForCES specifications,” in Networking, 2007. ICN ’07. Sixth International Conference on, April 2007, pp. 82–82.

[30] “The Global Internet Phenomena Report”, Sandvine.com, 2018. [Online]. Available: https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf. [Accessed: 12- Jan- 2019].

[31] S. Cho, J. R. Martin, R. Xu, M. H. Hammoud, and R. Melhem, “CA-RAM: A High-Performance Memory Substrate for Search-Intensive Applications”, 2007 IEEE International Symposium on Performance Analysis of Systems & Software, 2007.

[32] M. Bando, N. S. Artan, and H. J. Chao, “FlashLook: 100-Gbps hashtuned route lookup architecture”, International Conference on High Performance Switching and Routing, pp. 1–8, June 2009.

[33] S. Ghosh and M. Baliyan, “A hash based architecture of longest prefix matching for fast IP processing”, 2016 IEEE Region 10 Conference (TENCON), 2016.

[34] Y. Sun and M. S. Kim, “A Hybrid Approach to CAM-Based Longest Prefix Matching for IP Route Lookup”, 2010 IEEE Global Telecommunications Conference GLOBECOM, 2010.

[35] H. Le and V. K. Prasanna, “Scalable Tree-Based Architectures for IPv4/v6 Lookup Using Prefix Partitioning”, IEEE Transactions on Computers, vol. 61, no. 7, pp. 1026–1039, 2012.

[36] S. Kasnavi, P. Berube, V. Gaudet, and J. N. Amaral, “A cache-based internet protocol address lookup architecture”, Computer Networks, vol. 52, no. 2, pp. 303–326, 2008.

[37] P. Berube, A. Zinyk, J. N. Amaral, and M. Macgregor, “The Bank Nth Chance Replacement Policy for FPGA-Based CAMs”, Field Programmable Logic and Application Lecture Notes in Computer Science, pp. 648–660, 2003.

[38] F.-C. Kuo, Y.-K. Chang, and C.-C. Su, “A Memory-Efficient TCAM Coprocessor for IPv4/IPv6 Routing Table Update”, IEEE Transactions on Computers, vol. 63, no. 9, pp. 2110–2121, 2014.

[39] H. J. Chao and B. Liu, High performance switches and routers. Hoboken, NJ: Wiley-Interscience, 2007.

[40] S. Kaxiras and G. Keramidas, “IPStash: a power-efficient memory architecture for IP-lookup”, 22nd Digital Avionics Systems Conference.

[41] W. Wu, Packet forwarding technologies. Boca Raton, FL: Auerbach Publications, 2008.

[42] “RIS Raw Data — RIPE Network Coordination Centre”, Ripe.net, 2016, [Online].

[43] Eecs.harvard.edu, 2018. [Online]. Available: https://www.eecs.harvard.edu/~michaelm/postscripts/dimacs-chapter-08.pdf. [Accessed: 19- Aug- 2018].

[44] E. Stavinov, “A practical parallel CRC generation method”, Circuit Cellar-The Magazine For Computer Applications, 2010, vol. 31, no 234, p. 38.

[45] B. Wolff, B. Fradj, N. Belanger, and Y. Savaria, “Extending a CPU Cache for Efficient IPv6 Lookup”, 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), 2018.

[46] T. Noergaard, Embedded systems architecture: a comprehensive guide for engineers and programmers. Amsterdan: Newnes, 2013.

[47] “ARM Developer Suite Developer Guide - ARM architecture.” [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dui0056d/DUI0056.pdf. [Accessed: 12-Aug-2018].

[48] “Improving Performance”, Users.ece.utexas.edu, 2018. [Online]. Available: http://users.ece.utexas.edu/~gerstl/ee382v_f14/soc/vivado_hls/VivadoHLS_Improving_Performance.pdf. [Accessed: 12- April- 2017].

[49] “Vivado HLS –Tips and Tricks”, Xilinx.com, 2018. [Online]. Available: https://www.xilinx.com/publications/events/developer-forum/2018-frankfurt/hls-tips-and-tricks.pdf. [Accessed: 19- Mar- 2018].

[50] Xilinx.com, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf. [Accessed: 20- Apr- 2017].

[51] T. Cormen, C. Leiserson, R. Rivest and C. Stein, Introduction to algorithms.

[52] C. Audet, S. Le Digabel, and C. Tribes “NOMAD user guide”, Tech. Rep. G-2009-37, Les Cahiers du GERAD, 2009.

[53] “Loi normale”, Fr.wikipedia.org, 2019. [Online]. Available: https://fr.wikipedia.org/wiki/Loi_normale. [Accessed: 17- Nov- 2018].

[54] “Hachage universel”. [Online]. Available: https://fr.wikipedia.org/wiki/Hachage_universel. [Accessed: 17- Nov- 2018].