UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MODÉLISATION D'INSTANCES VIRTUALISÉES D'ACCÉLÉRATION DE TRAITEMENT DE PAQUET SUR UNE ARCHITECTURE MULTICŒURS

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

KHALIL BLAIECH

JUILLET 2016

REMERCIEMENTS

Le travail présenté dans ce mémoire a été réalisé au sein du Laboratoire de recherche en Téléinformatique et Réseaux (LTIR). Je remercie le Professeur Omar Cherkaoui, mon directeur de recherche et directeur du laboratoire LTIR de m'avoir accueilli, dirigé et donné les moyens pour effectuer mon travail de recherche.

Je tiens tout particulièrement à exprimer ma profonde gratitude au Professeur Omar Cherkaoui pour son encadrement et son suivi. Ses conseils, ses directives, son soutien, ses remarques pertinentes, ses encouragements permanents, sa confiance qu'il m'a témoignée tout au long de ces années de recherche, ses qualités d'enseignant, ses qualités humaines, sa générosité, son dynamisme et sa disponibilité ont joué un rôle déterminant dans l'accomplissement de ce travail. Qu'il trouve ici, l'expression de ma profonde reconnaissance et de ma grande estime.

Je remercie très chaleureusement tous mes anciens et actuels collègues du laboratoire LTIR de leur collaboration et de leur support dont ils m'ont fait bénéficier durant ce travail. Qu'ils trouvent ici l'expression de ma reconnaissance pour leur témoignage de sympathie et d'amitié. Ce fut, en effet, un plaisir de partager quotidiennement de si bons moments avec eux tout le long de ces trois années.

Tous mes remerciements à l'ensemble du corps enseignant ainsi qu'à tout le personnel de l'Université du Québec à Montréal (UQAM) et tous ceux qui ont participé de prés ou de loin au bon déroulement de cette maîtrise.

Je présente, enfin, mes remerciements aux membres du jury pour leur participation à l'évaluation de ce travail.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LEXIQUE

- **Access Control List (ACL) :** liste de contrôle d'accès
- **Application Programming Interface (API) :** interface de programmation
- **Application-Specific Integrated Circuit (ASIC) :** circuit intégré spécialisé
- **Bare-metal :** machine sans système d'exploitation
- **Bid :** mise
- **Bundle :** collection de ressources
- **Backbone Network :** réseau fédérateur
- **Bridging :** un service qui permet de passer d'un réseau à un autre
- **Central Processing Unit (CPU) :** unité de traitement centrale
- **Checksum :** empreinte qui permet de vérifier qu'un paquet est reçu sans erreurs
- **Classifier :** classificateur de flux
- **Cloud Computing :** informatique nuagique
- **Cluster :** collection ou groupement
- **Control Plane :** plan de contrôle ou plan de commande
- **Constraints :** contraintes
- **Core :** cœur dans un processeur multi-cœurs
- **Counters :** compteurs
- **Crypto :** chiffrement des données des paquets
- **Data Path :** circuit de données
- **Data Plane :** plan de données
- **Deadline :** délai limite
- **Deep Packet Inspection (DPI) :** une application réseau qui vise à faire une analyse approfondie du paquet, généralement appliquée pour des fins de sécurité

informatique

- **Direct Memory Access :** accès direct à la mémoire
- **Dynamic Random Access Memory (DRAM) :** mémoire vive dynamique
- **Egress :** à la sortie
- **Exact Match :** correspondance exacte
- **Fast Path :** chemin de traitement de données
- **Field :** champ
- **Field Programmable Gate Array (FPGA) :** circuit logique programmable
- **Filtering :** filtrage ou fonction de filtrage des flux de paquets
- **Firewall :** pare-feu
- **Flow :** flux de paquets
- **Flow Table :** table de flux, table à plusieurs entrées où chaque entrée est associé à un flux de paquet
- **Forwarding :** transmission de paquets, un seul terme qui nous permet de désigner une commutation au niveau 2 ou un routage au niveau 3.
- **General Purpose Processors (GPP) :** processeur à usage général
- **Header :** en-tête de paquet, un champ ayant une valeur significative
- **Hypervisor :** hyperviseur
- **Ingress :** à l'entrée
- **Input/Output (I/O) :** entrée/sortie (E/S)
- **Internet Protocol (IP) :** protocole internet
- **Inter-Process Communication (IPC) :** mécanismes de communication inter-processus
- **Kernel space :** espace noyau
- **Kit :** trousse ou ensemble d'outils
- **Latency :** latence ou délai de traitement
- **Longest Prefix Match (LPM) :** le préfixe de correspondance le plus long
- **Lookup :** recherche de données dans une table

- **Mapping :** association
- **Match :** correspondance
- **Matchfield :** champ de correspondance
- **Measures :** mesures prélevées à partir de compteurs
- **Medium Access Control :** contrôle d'accès au support
- **Microengine :** micro-moteur, l'équivalent d'un cœur dans un processeur multi-cœurs
- **Multi-cores :** processeur multi-cœurs
- **Multi Purpose Processor Array (MPPA) :** réseau de processeurs massivement parallèles
- **Multithreading :** exploitation de plusieurs instances de petit processus s'exécutant de manière asynchrone et partagent les ressources du processus hôte
- **Network Flow Processor (NFP) :** processeur des flux réseau
- **Network Function Virtualization (NFV) :** virtualisation de fonction réseau
- **Network Interface Controller (NIC) :** contrôleur d'interface réseau
- **Network on Chip (NoC) :** système de communication entre les cœurs d'un processeur multi-cœurs
- **Network Processor (NP) :** processeur réseau
- **Operating System (OS) :** système d'exploitation
- **Packet Processing :** traitement de paquets
- **Packet Processing Path :** circuit de traitement de paquet
- **Parse :** analyse d'un paquet
- **Parsing :** analyse du contenu d'un paquet
- **Path :** circuit
- **Performance metric :** métrique de performance
- **Performance threshold :** seuil toléré de performance
- **Peripheral Component Interconnect Express (PCIe) :** bus local série pour connecter l'interface réseau au processeur hôte

- **Poll :** sonder ou obtenir
- **Processing :** traitement
- **Real-Time Executive for Multiprocessor Systems (RTEMS) :** système d'exploitation temps réel pour processeur multi-cœurs
- **Requirements :** exigences
- **Ressource Controller (RC) :** contrôleur de ressources
- **Ressource Manager (RM) :** gestionnaire de ressources
- **Routing :** routage
- **Search :** recherche
- **Search Structure :** structure de recherche
- **Settings :** paramètres
- **Slice :** partition de réseau logique
- **Slicing :** partitionner
- **Slow Path :** chemin de contrôle de données
- **Software Defined Networks (SDN) :** réseau définis de manière logicielle
- **Software Development Kit (SDK) :** trousse de développement logiciel
- **Specific Purpose Processors (SPP) :** processeur à usage spécifique
- **Static Random Access Memory (SRAM) :** mémoire vive statique
- **Statistics :** mesures prélevées et traitées à partir de compteurs
- **Switch :** commutateur, équipement réseau de niveau 2.
- **Task :** tâche
- **Tag :** étiquette
- **Tagging :** ajout d'étiquettes ou encapsulation
- **Task Optimized Processor (TOP) :** processeur à tâches spécialisés
- **Ternary Content-Addressable Memory (TCAM) :** mémoire ternaire adressable par contenu
- **Testbed :** banc de test
- **Thread :** petit processus s'exécutant de manière asynchrone et partagent les

ressources du processus hôte

- **Throughput :** débit ou nombre de paquet reçus à l'entrée du système
- **Time-To-Live (TTL) :** champs d'entête d'un paquet IP qui indique le nombre de saut restant pour le paquet avant qu'il soit supprimé du réseau dans lequel il circule
- **Traffic generator :** générateur de trafic
- **Traffic Manager (TM) :** gestionnaire de trafic
- **Transmission Control Protocol (TCP) :** protocole de contrôle de transmissions
- **Trigger :** déclencher
- **Tunneling :** service qui permet de créer des tunnel en isolant un trafic allant d'un réseau à un autre
- **User space :** espace utilisateur
- **Very Long Instruction Word (VLIW) :** architecture de processeur à mot d'instruction très long

RÉSUMÉ

La convergence vers des solutions qui reposent sur le logiciel porte à croire que les opérateurs cherchent plus de flexibilité pour programmer le réseau et moins d'effort pour s'adapter aux nouvelles contraintes et aux changements brusques des technologies réseau. La virtualisation des fonctions réseau nécessite des accélérations de traitement de paquet qui sont, généralement, supportées au niveau matériel, en particulier au niveau des interfaces réseau. Dans un contexte où le matériel est partagé et les besoins en accélération de traitement de paquet sont requis, l'utilisation des interfaces réseau basées sur des processeurs multi-cœurs s'avère très pratique. À l'instar des processeurs dédiés au traitement de paquets tels que les NPs (*Network Processors*) ou les ASICs (*Application Specific Integrated Circuits*), les processeurs multi-cœurs sont considérés comme la solution la plus adéquate pour supporter un traitement de paquet modulable tout en garantissant de très hautes performances.

Dans ce mémoire, nous proposons un Packet Processing-Aware Hypervisor (*P2AH*) pour déterminer des patrons d'accélération de traitement de paquet en utilisant les ressources modulables et programmables du processeur multi-cœurs virtualisé. *P2AH* est un hyperviseur qui interagit avec les ressources matérielles pour offrir des accélérations de traitement de paquets aux applications et aux fonctions de réseau telles que les routeurs, les commutateurs, les pare-feux, etc. *P2AH* a été conçu pour les architectures à processeur multi-cœurs sur lesquelles des instances d'accélérations seront générées et adaptées afin de satisfaire les besoins et les contraintes strictes du traitement de paquet. Dans l'objectif de modéliser des accélérations de traitement de paquet, *P2AH* introduit une stratégie de gestion dynamique des ressources virtualisées garantissant *(i)* l'optimisation de l'utilisation des ressources, *(ii)* le partage équitable des ressources et *(iii)* la minimisation des coûts liés au partitionnement des ressources et à l'isolation du trafic.

À l'issue de ce travail, nous avons démontré que les architectures à processeur multi-cœurs sont capables de supporter des instances virtualisées pour offrir une accélération matérielle de traitement de paquet. De même, nous avons prouvé la robustesse et l'efficacité de la stratégie de gestion des ressources employée par *P2AH* pour ajuster le traitement de paquet aux exigences du trafic et aux contraintes matérielles.

Mots clés : Virtualisation ; Gestion des ressources ; Traitement de paquets ;

INTRODUCTION

Les nouveaux paradigmes réseau, principalement le SDN (*Software-Defined Networking*) et le NFV (*Network Function Virtualization*), mettent à l'épreuve l'infrastructure réseau sous-jacente et n'en finissent pas de repousser les limites des équipements réseau. Généralement, ces équipements intègrent une multitude de technologies de processeurs. Partant des processeurs à usage général (GPP) et arrivant aux processeurs à usage spécifique (SPP), chaque architecture a ses propres caractéristiques et présente ses propres limites. Dans une ère régie par le besoin de faciliter la programmation et la gestion des réseaux, la diminution des coûts liés à l'infrastructure et la rapidité de déploiement de nouveaux services, la virtualisation est reine, plus précisément celle des fonctions réseau dans les centres de données. Ainsi, les opérateurs réseaux et fournisseurs de services ont tendance à favoriser les architectures à processeur multi-cœurs. À l'instar des processeurs dédiés au traitement de paquets tels que les NPs (*Network Processors*) ou les ASICs (*Application Specific Integrated Circuits*), les processeurs multi-cœurs sont considérés comme la solution la plus adéquate pour satisfaire aux exigences des tendances émergentes du réseau.

Bien que les processeurs ASICs répondent entièrement aux besoins critiques de performances, ils présentent l'inconvénient d'être rigide. Ces derniers présentent, d'une part, une architecture figée conçue pour supporter un modèle de traitement de paquet fixe (*pipeline*), et d'autre part, d'avoir un temps de mise en marché très lent, ce qui est contraignant vu la vitesse à laquelle évolue le réseau et les exigences strictes des applications auxquelles il faut répondre. Néanmoins, les processeurs

NPs qui ont connu une très grande popularité, étaient considérés comme une solution miracle pouvant rallier à la fois, flexibilité et performance. En effet, leur architecture intègre aussi bien des composants programmables (TOP, MicroEngine) et des accélérateurs matériels configurables (TCAM, TM, Crypto). Certes, le modèle de traitement de paquet démontre certaines limites ; le *pipeline* utilisé est souvent statique, en d'autres termes, le traitement doit impérativement suivre un cheminement prédéfini à l'avance. De ce fait, il est plus difficile d'adapter ou d'arranger le *pipeline* pour répondre aux besoins et aux spécifications nouvellement introduites par la vague technologique que connait actuellement le réseau.

La convergence vers des solutions qui reposent sur le logiciel porte à croire que les opérateurs cherchent plus de flexibilité pour programmer le réseau et moins d'effort pour s'adapter aux nouvelles contraintes et aux changements brusques du réseau. En effet, dans un environnement où le matériel est partagé et les besoins en accélération de traitement de paquet sont requis, il est indispensable de disposer de techniques avancées et de solutions optimales pour permettre de générer rapidement des instances de traitement spécifiques d'une part et d'optimiser l'utilisation des ressources matérielles d'une autre part. L'utilisation des interfaces réseau basées sur des processeurs multi-cœurs s'avère ainsi très pratique. Grâce à leur architecture qui incorpore une centaine de processeurs, parfois certains accélérateurs matériels, les multi-cœurs créent un terrain fortement favorable pour supporter un traitement de paquet modulable et garantissant plus de facilité en matière de programmabilité.

Dans cette optique, nous proposons d'exploiter une architecture de processeur multi-cœurs pour mettre en place un environnement qui supporte des modèles d'accélérations de traitement de paquet spécifiques tout en bénéficiant de la virtualisation et du partage des ressources. Ainsi, nous avons conçu, implémenté et évalué un hyperviseur pour le processeur Andey MPPA-256 de Kalray. L'hyper-

viseur tient compte des besoins en accélérations pour générer des instances de traitement de paquets et partage les ressources en utilisant une stratégie d'allocation dynamique. Cette solution vise à offrir des patrons d'accélérations de traitement de paquet en tirant parti des ressources modulables et programmables du processeur multi-cœur pour un environnement virtualisé, dans l'objectif (1) d'assurer une isolation entre les différentes instances, (2) d'optimiser l'utilisation des ressources, (3) de garantir un partage équitable des ressources, et (4) de prévenir la dégradation du traitement de paquet.

Ce mémoire est constitué de quatre chapitres. Le premier chapitre présente les récents travaux réalisés dans le cadre de la virtualisation des architectures multi-cœurs ainsi que les mécanismes utilisés pour supporter les opérations de traitement de paquet.

Ensuite, le deuxième chapitre présente la solution proposée qui s'articule autour de *P2AH* (Packet Processing-Aware Hypervisor) pour arranger les ressources modulables des architectures multi-cœurs afin de satisfaire les besoins en accélération de traitement de paquet et de respecter les contraintes liées aux ressources. Nous allons décrire la conception et les fonctionnalités de *P2AH* ainsi que les mécanismes utilisés pour la gestion des ressources du multi-cœurs.

Le troisième chapitre présente l'implémentation de *P2AH* sur le processeur Andey MPPA-256 de Kalray. Nous allons décrire l'architecture matérielle du processeur multi-cœurs et détailler les différentes approches pour la mise en place de *P2AH*.

Avant de donner la conclusion générale de ce mémoire, nous allons évaluer *P2AH* comparativement aux approches existantes afin d'identifier les points forts et les limites de la modélisation des accélérations de traitement de paquet au niveau des architectures matérielles et en particulier le processeur MPPA-256.

CHAPITRE I

VIRTUALISATION ET TRAITEMENT DE PAQUET POUR LES ARCHITECTURES MULTI-CŒURS

Introduction

L'objectif de ce premier chapitre est d'analyser l'état de l'art de la virtualisation et du traitement de paquet au niveau des architectures multi-cœurs. Ces architectures présentent une grande flexibilité et offrent de bonnes performances. Avec l'émergence de la virtualisation et de la programmabilité des réseaux, les fonctions réseau généralement supportées par des solutions matérielles ont tendance à converger vers des solutions logicielles. Ainsi les plateformes multi-cœurs sont mises en avant pour supporter une telle transition. Dans un premier temps, nous allons présenter les travaux et les mécanismes existants qui ont été mis en œuvre afin d'assurer la virtualisation des architectures multi-cœurs ainsi que les outils qui ont permis de réaliser du traitement de paquet sur ce type de plateforme. Ensuite, nous allons nous intéresser à une approche existante similaire à notre proposition. Cette approche sera décrite et servira de modèle de comparaison dans la suite de ce mémoire.

1.1 Virtualisation des architectures multi-cœurs

La virtualisation est un mouvement en plein essor qui est en train de révolutionner l'industrie des réseaux (Chowdhury et Boutaba, 2010). En effet, c'est l'un des concepts centraux autour duquel les réseaux d'entreprises et les centres de données sont construits. La virtualisation est une couche d'abstraction qui supprime l'adhérence entre les systèmes logiques et les systèmes physiques. Même s'il existe plusieurs techniques de virtualisation, elles s'accordent toutes sur le même objectif qui est de permettre à plusieurs systèmes de fonctionner sur le même matériel. Dans le cas de la virtualisation des réseaux, les routeurs physiques sont principalement utilisés pour acheminer les paquets. Toutes les règles de configuration du réseau, de gestion de la sécurité, de la haute disponibilité, de mise en œuvre des protocoles de routage, sont gérées par un hyperviseur réseau. L'hyperviseur réseau permet l'instanciation de plusieurs nœuds virtuelle sur un seul nœud physique. Cette technologie offre plusieurs avantages par rapport aux implantations physiques, dont la flexibilité et le partage des ressources.

1.1.1 Besoin de virtualisation des architectures multi-cœurs

Avec l'émergence des technologies tels que le NFV (*Network Function Virtualization*) (Mijumbi *et al.*, 2015) et le SDN (*Software Defined Networking*) (Kreutz *et al.*, 2015), les défis sont de taille et sont de plus en plus difficiles à relever. Les technologies NFV et SDN suscitent un intérêt croissant et les fournisseurs de services sont nombreux à s'interroger sur les conditions de migration vers une infrastructure réseau virtuelle. Cela implique le déploiement de différents réseaux logiques définis et régis par une entité centrale sur le même réseau physique. Tandis que le SDN consiste à séparer le plan de contrôle et le plan des données pour consolider et automatiser les réseaux distribués à grande échelle, le NFV vise à virtualiser les fonctions du réseau et promet la réduction des coûts liés à l'infra-

structure et la maintenance du réseau. D'un point de vue purement technique, l'infrastructure matérielle doit s'adapter à ce changement brusque imposé par ces nouvelles technologies. Les fournisseurs de services et les opérateurs optent ainsi pour l'utilisation d'équipements génériques capable de réduire les délais de déploiement et de supporter de nouvelles fonctionnalités.

Pour illustrer cela, les fournisseurs de services implémentent les fonctionnalités réseaux, qu'un équipement dédié soit supposé supporter, au niveau des machines virtuelles installées sur un ou plusieurs serveurs. Ces fonctions réseau ont besoin d'accélération, dans ce contexte une accélération de traitement de paquet, pour acheminer le trafic et offrir les hautes performances qu'un équipement dédié était en mesure de présenter. Dans cette optique, les accélérations de traitement de paquet seront réalisées directement au niveau de l'interface réseau et seront contrôlées par la machine virtuelle associée à la fonction réseau. Les constructeurs ont donc introduit des interfaces réseau "intelligentes" capables de supporter ce type d'accélération. Ainsi, les plateformes multi-cœurs de par leur architecture flexible et programmable représentent une solution pratique et économique pour ce genre d'interface réseau. Ces architectures présentent plusieurs modèles de programmation et outils associés qui permettent de construire des programmes parallèles offrant de bonnes performances. En effet, la multiplication des cœurs accentue les gains de performance et de capacité de traitement.

1.1.2 Approches de virtualisation des architectures multi-cœurs

Les constructeurs ont suivi un processus de normalisation des implémentations pour faciliter la création de nouvelles opportunités pour concevoir, développer et déployer des fonctionnalités réseau personnalisées. L'une des réalisations majeures

qui ont été mises au point est l'architecture d'OpenvSwitch [1]. Cette initiative définit la virtualisation des fonctions de commutation au niveau de processeurs génériques qui supportent un système d'exploitation Linux. La conception et l'implémentation est décrite dans (Pfaff *et al.*, 2015). L'OpenvSwitch exploite des modules qui s'exécutent au niveau de l'usager (*user space*) pour les fonctions de contrôle et d'autres au niveau du noyau (*kernel space*) pour l'acheminement des paquets. Ces commutateurs virtuels sont régis et contrôlés par l'hyperviseur réseau. Des améliorations visant à exploiter les ressources des interfaces réseau et à décharger une partie du trafic ont été récemment apportées au modèle.

D'autres travaux abordent la virtualisation des multi-cœurs en s'intéressant au partitionnement des ressources pour les dédier à différentes tâches de traitement. Dans (Wells *et al.*, 2009), une technique consistant à affecter dynamiquement des processeurs virtuels aux systèmes qui s'exécutent en parallèle sur la même plateforme matérielle est proposée. Il s'agit d'associer des unités de traitement, particulièrement les *cores* virtuels, aux *cores* physiques qui conviennent le plus au traitement des machines virtuelles s'exécutant au niveau applicatif.

Malgré les avantages offerts par la couche de virtualisation de l'hyperviseur, les différents appels aux services de l'hyperviseur augmentent la charge et le délai des traitements. Certaines propositions tentent de pallier ce problème par l'emploi de stratégie qui minimise ces appels. Dans (Hu *et al.*, 2010), des stratégies d'ordonnancement au niveau des interfaces d'entrée/sortie sont proposées pour éliminer les blocages et améliorer davantage le partitionnement dynamique au niveau des multi-cœurs. La proposition consiste à avoir plusieurs groupes de *cores* dédiés pour un ordonnancement spécifique des tâches de traitement. Cette stratégie a permis de réduire les délais d'ordonnancement habituels réalisés au niveau de

1. http ://openvswitch.org/

l'hyperviseur. D'autres technologies utilisées dans (Dong *et al.*, 2012) permettent de contourner l'hyperviseur en virtualisant les bus de communications entre les interfaces réseaux et le processeur hôte. Cela assure un accès direct aux interfaces. Cette technologie permet de partager les interfaces réseau en donnant l'impression que chaque machine virtuelle bénéficie d'une interface réseau dédiée.

1.2 Traitement de paquet sur les architectures multi-cœurs

L'industrie des réseaux a été parmi les premiers à adopter les processeurs multi-cœurs, aussi bien pour la couche d'acheminement des paquets que pour la couche de contrôle. Ces processeurs supportent généralement un traitement de paquet logiciel, c'est-à-dire un traitement supervisé par un système d'exploitation contrairement aux traitements de paquet exécuté directement au niveau matériel (*bare-metal*). Ce traitement de paquet logiciel nécessite de nombreux appels système ce qui est contraignant pour un traitement de paquet à haut débit.

1.2.1 Outils d'accélération du traitement de paquet

Netmap (Rizzo *et al.*, 2012) propose une accélération pour le traitement de paquet pour atteindre de bonnes performances pouvant supporter jusqu'à 10Gbit/s. Cette approche consiste à apporter des optimisations au niveau de l'interfaçage avec la pile TCP/IP du système d'exploitation, plus spécifiquement les systèmes Linux. Netmap utilise ces moyens d'interfaçage pour exposer à l'espace usager (*user space*) une copie des anneaux des NICs (*Network Interface Controller*). Cette approche minimise la charge liée à la copie des paquets de l'interface physique vers la mémoire du système et permet ainsi de bénéficier d'une synchronisation rapide.

D'autres démarches proposent de contourner le système d'exploitation et d'offrir

des accélérations matérielles. C'est le cas de la solution d'Intel, le DPDK [2] (*Data Plane Development Kit*). Cette solution vise à faciliter le développement d'applications de traitement de paquets à haut débit destinées à rouler, à la base, sur les processeurs multi-coeurs d'Intel. Par la suite, différents constructeurs ont adapté cette solution à leurs architectures matérielles. L'outil consiste en un ensemble de bibliothèques de traitement de paquet visant à accélérer le traitement et améliorer les performances. DPDK incorpore plusieurs composants pour l'accès aux ressources matérielles. Aussi, l'outil offre des pilotes pour l'interfaçage avec des cartes réseau Ethernet haut débit.

En outre, ODP [3] (OpenDataPlane) propose une bibliothèque pour programmer des applications de plan de données sur diverses plateformes en particulier les architectures multi-processeurs. L'ODP offre une abstraction par rapport au matériel, ce qui facilite la portabilité des applications de traitement de paquet. L'outil permet de manipuler différentes ressources hétérogènes, ce qui permet d'avoir, à titre d'exemple, une partie du traitement de paquet supportée au niveau logiciel et une autre partie supportée au niveau matériel. Ainsi, ODP vise à contourner les limites imposées par les équipements spécialisés et les architectures spécifiques pour consolider la programmation du traitement de paquet.

1.2.2 Mécanismes d'optimisation du traitement de paquet

Des travaux récents s'intéressent à l'optimisation de certaines tâches de traitement de paquet jugées contraignantes au niveau logiciel. En effet, le traitement de paquet doit respecter certaines exigences liées à la capacité et à la latence de traitement. Le traitement de paquet repose essentiellement sur la recherche et la

---

2. http ://dpdk.org/doc/guides/

3. http ://www.opendataplane.org/

classification des flux. Cette tâche s'avère complexe et coûteuse au niveau logiciel. Généralement réalisée au niveau d'accélérateurs matériels, la classification des flux au niveau des processeurs à usage général ralentit considérablement le traitement de paquet.

AgreeCuts (Qi *et al.*, 2007) présente un algorithme pour la classification des flux proposé pour les processeurs réseau à base de multi-cœurs. Cet algorithme se base essentiellement sur l'algorithme de classification de paquet à haut débit HiCuts (Gupta et McKeown, 1999). L'approche utilisée améliore le débit de classification et réduit l'espace mémoire utilisé à l'aide d'un mécanisme de gestion des états de flux (SigHash) qui peut supporter jusqu'à 10M (Million) de flux. D'un autre côté, les réseaux actuels présentent des liaisons pouvant dépasser les 10Gbit/s d'où la nécessité de garantir une capacité de traitement élevée.

Différents travaux proposent ainsi de paralléliser les tâches de traitement de paquet en particulier la classification des paquets (Qu *et al.*, 2015). Une recherche indépendante sur chaque champ de l'entête de paquet est réalisée par l'emploi d'un *thread* par recherche. À la fin de la recherche, on emploie plusieurs *threads* pour accélérer la fusion des différents résultats de recherche. Cette approche supporte une classification de 33Mpps (Million de paquets par seconde) avec des règles utilisant 15 champs d'entête de paquet et assure un délai de 2 microsecondes (Qu *et al.*, 2014).

Par ailleurs, des travaux s'intéressent à l'adaptabilité du traitement de paquet aux processeurs multi-cœurs (Kokku *et al.*, 2004). La méthode employée détermine le traitement requis pour chaque module de routage, applique une duplication des tâches pour satisfaire les besoins en accélération du traitement de paquet et introduit un schéma d'association entre les tâches et les ressources de traitements (Wu et Wolf, 2012).

1.3 Network-Aware Hypervisor

Dans cette section, nous allons présenter *NAH* (Network-Aware Hypervisor), une solution existante, initialement proposée dans le cadre du projet NetVirt [4], qui incorpore un mécanisme de gestion dynamique de ressources. Ce mécanisme a été tout d'abord introduit dans les travaux de (Blaiech *et al.*, 2014). Par la suite, la stratégie d'allocation des ressources a connu des améliorations au niveau conceptuel qui ont été décrites dans (Blaiech et Cherkaoui, 2015) et détaillées dans (Snaiki, 2014). *NAH* a été conçu pour des plateformes matérielles virtualisées et a été testé principalement sur des processeurs réseaux tels que Netronome NFP-3200 (nfp, 2010) et EZchip NP-4 (ezc, 2015). *NAH* supervise un ensemble de ressources de routeurs virtualisés. Il se base sur des observations pour un ensemble de flux qui caractérise une *slice* réseau afin d'allouer les ressources nécessaires pour le traitement de ces *slices*. *NAH* vise à maintenir (i) l'équité entre les instances virtuelles, (ii) la stabilité des traitements des instances et (iii) la prévention de congestions au niveau des ressources partagées.

NAH considère un ensemble de ressources interdépendantes qui supporte un traitement de paquet spécifique. Cet ensemble de ressources caractérise les instances de traitement de paquet. Grâce aux mécanismes de gestion des ressources implémentés par *NAH*, les ressources sont allouées suivant le besoin en traitement des différentes *slices* réseau. Ainsi, *NAH* partage ces ressources entre les différentes *slices* afin de traiter les différents flux qui leur sont associés.

1.3.1 Processus de fonctionnement de NAH

NAH présente une série de contrôleurs des ressources matérielles et de contrôleurs de *slices* réseaux. Ces contrôleurs collectent des mesures de performances et des

4. http ://www.netvirt.ca/

statistiques relatives aux ressources et aux *slices* puis les transmettent au gestionnaire des ressources pour allouer les ressources correctement et ainsi répondre aux exigences du trafic et aux contraintes des ressources.

Le mécanisme d'allocation incorpore trois niveaux de contrôle : l'équité, la stabilité et la prévention des blocages. Chaque niveau s'exécute périodiquement sur un intervalle de contrôle différent. En effet, chaque niveau a un intervalle de contrôle qui le caractérise et au bout duquel il s'exécute pour déterminer un schéma optimal d'allocation de ressources. Également, chaque niveau agit sur un critère de performance différent. Par exemple, un niveau peut contrôler le débit correspondant aux *slices* tandis qu'un autre niveau peut contrôler la latence de traitement des *slices*. Le mécanisme d'allocation des ressources procure des vecteurs d'allocation pour chaque *slice*. Chaque vecteur indique les proportions de ressources auxquelles le traitement des flux de la *slice* correspondante sera associé. Ces vecteurs seront pris en compte par les contrôleurs et appliqués directement aux ressources contrôlées.

1.3.2 Architecture de gestion des ressources de NAH

NAH présente une hiérarchie de niveaux de contrôle qui sont directement reliés, chaque niveau dépend du niveau qui le précède ou le succède. Chaque niveau de contrôle vise à déterminer une allocation de ressources suivant l'objectif qui lui a été fixé. Nous distinguons trois principaux niveaux de contrôle d'allocation des ressources :

- Niveau d'équité : présente le premier niveau de contrôle, celui-ci vise à allouer les ressources équitablement entre les *slices*. Ce niveau implémente un algorithme basé sur la théorie des jeux pour ajuster l'allocation afin d'uniformiser la fonction d'utilité ou le profit de chacune des *slices*.
- Niveau de stabilité : présente le deuxième niveau de contrôle, celui-ci est responsable de la stabilité du traitement de paquet de chacune des *slices*. Ce niveau

se base sur la théorie de contrôle pour stabiliser l'utilité de chaque slice et pour minimiser les variations et les fluctuations des allocations des ressources.

- Niveau de prévention de blocage : présente le troisième et dernier niveau. Il vise à prévenir les congestions au niveau des ressources en détectant les éventuels points de blocages. Ce niveau est le gestionnaire le plus dynamique qui s'exécute fréquemment afin d'ajuster les allocations des ressources pour les différentes *slices*.

1.3.3 Mécanismes de gestion des ressources de NAH

Les trois niveaux reçoivent les vecteurs de mesures de performance des contrôleurs de *slices* pour générer éventuellement un nouveau vecteur d'allocation. Suivant la vue qu'il a sur les ressources du système, chaque niveau communique avec le niveau qui le précède ou le succède à travers deux différents signaux :

- *Resource Threshold* : Ce signal est envoyé d'un niveau supérieur à un niveau inférieur pour déclencher la transmission des seuils d'allocation des ressources que le niveau suivant doit respecter dans son processus de gestion des ressources. Les seuils d'allocation sont déterminés par chaque niveau suite à l'exécution des calculs pour la gestion des ressources.
- *Violation Event* : Ce signal est envoyé d'un niveau inférieur à un niveau supérieur pour signaler le dépassement des seuils d'allocations. Le niveau précédent doit ainsi calculer un nouveau seuil d'allocation de ressources et le transmettre au niveau suivant ayant émis le signal.

Conclusion

À travers ce chapitre, nous avons étudié certaines approches de virtualisation et différents mécanismes pour le traitement de paquets au niveau des plateformes multi-cœurs. Il est important de relever la nécessité de réaliser des accélérations

de traitement de paquet au niveau des interfaces réseau (NIC) et de minimiser les délais dus aux appels au système hôte (hyperviseur). Il est ainsi indispensable de gérer les ressources matérielles afin de bénéficier des bonnes performances de traitement. Aussi, nous avons vu à travers une proposition existante (NAH) les mécanismes de virtualisation et de gestion des ressources pour supporter des instances d'accélérations de traitement de paquet qui répondent au mieux aux exigences du trafic et qui s'adaptent aux contraintes matérielles. Ces différentes stratégies et approches étudiées vont guider nos choix de conception et de mise en place d'un hyperviseur capable de gérer des instances d'accélération modulables dynamiquement dans un environnement multi-cœurs virtualisé.

CHAPITRE II

PACKET PROCESSING-AWARE HYPERVISOR

Introduction

Dans ce chapitre, nous allons présenter notre hyperviseur *P2AH* (*Packet Processing-Aware Hypervisor*) pour une architecture à processeur multi-cœurs. *P2AH* supporte des modèles d'accélérations de traitement de paquet spécifiques dans un environnement virtualisé. La solution proposée vise à offrir des patrons d'accélérations de traitement de paquet en bénéficiant des ressources modulables et programmables du processeur multi-cœurs. L'objectif de ce chapitre est de présenter la conception, les propriétés et les caractéristiques de *P2AH*. Tout d'abord, nous présenterons la conception de l'hyperviseur où nous détaillerons son architecture et ses principales fonctionnalités. Ensuite, nous expliquerons ses concepts de base que nous avons adaptés pour assurer ses fonctionnalités.

2.1 Conception de P2AH

P2AH est un hyperviseur qui interagit avec les ressources matérielles pour offrir des accélérations de traitement de paquets aux applications et aux fonctions réseau telles que les routeurs, les commutateurs, les pare-feux, etc. *P2AH* a été conçu pour les architectures à processeur multi-cœurs sur lesquelles des instances d'accélérations seront générées et adaptées afin de satisfaire les besoins et les

contraintes strictes du traitement de paquet. Il s'agit d'un hyperviseur spécifique pour le traitement de paquet et dédié aux plateformes multi-cœurs. En d'autres termes, *P2AH* présente une abstraction des accélérateurs matériels au niveau des interfaces réseau en exploitant les ressources modulables des multi-cœurs.

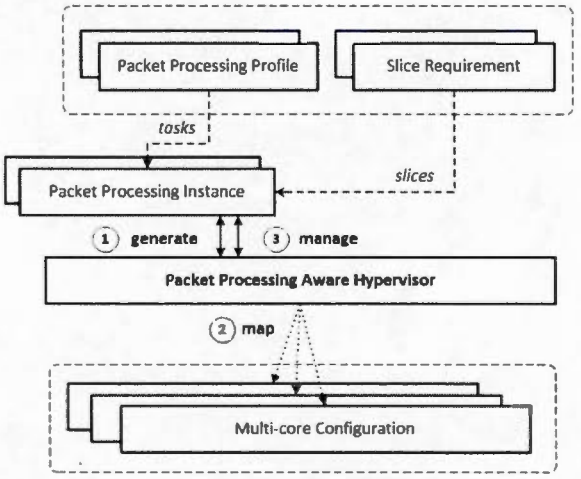


Figure 2.1: Vue générale de *P2AH*

Dans un environnement où les *slices* réseau, caractérisées par un ensemble de flux de trafic, sont imprédictibles et nécessitent différents traitements, il est nécessaire de trouver le groupe de ressources adéquates pour satisfaire ces besoins. À partir des profils d'accélération de traitement de paquet et des exigences du trafic, *P2AH* génère des modèles d'instances d'accélération de traitement de paquet en associant les tâches de traitement aux différentes ressources sous-jacentes (voir figure 2.1). Les instances d'accélération de traitement de paquet seront, par la suite, dimensionnées et ajustées dynamiquement par rapport aux besoins des *slices*. Il s'agit d'adapter les ressources, correspondant à une configuration du processeur multi-cœurs, pour supporter des accélérations de traitement de paquet requises.

À travers ces opérations, la conception de l'hyperviseur vise principalement à identifier des "patrons" ou "modèles" de virtualisation de fonctions d'accélération

de traitement de paquet dans le contexte prédéfini des plateformes à base de processeurs multi-cœurs. Ces modèles seront déterminés en fonction de : *(i) la nature du traitement*, soit les tâches requises pour acheminer les paquets ; *(ii) les contraintes du traitement*, soit le temps nécessaire pour appliquer l'ensemble des opérations aux paquets ainsi que le coût lié à ces opérations ; *(iii) la capacité du traitement*, soit l'ensemble des flux supportés pour un ensemble d'accélérations spécifiques de traitement paquet.

Dans la perspective de modélisation des fonctions d'accélérations de traitement de paquet, la conception de *P2AH* introduit une stratégie de gestion des ressources virtualisées garantissant *(i)* l'optimisation de l'utilisation des ressources, *(ii)* le partage équitable des ressources et *(iii)* la minimisation des coûts liés au partitionnement des ressources et à l'isolation du trafic.

2.1.1 Architecture de P2AH

L'architecture de *P2AH* s'articule autour de deux principales entités : (1) l' *Allocator* et (2) le *Slicer*, qui interagissent avec une suite de contrôleurs (voir figure 2.2). Les blocs fonctionnels qui constituent l'architecture de *P2AH* permettent de déterminer une configuration multi-cœurs adaptée pour chaque instance de traitement de paquet suivant les besoins en accélération.

- **L' *Allocator*** : Cette entité présente un gestionnaire d'allocation responsable de l'instanciation, de la modélisation et du dimensionnement des accélérations de traitement de paquet. En fonction des besoins des *slices* et des ressources disponibles, le gestionnaire d'allocation détermine une combinaison de ressources pour satisfaire le traitement de paquet correspondant.
- **Le *Slicer*** : Cette entité présente un gestionnaire de partage responsable du partitionnement des instances générées entre les *slices* ayant des besoins d'accélération spécifiques. Le gestionnaire de partage des instances ajuste dynami-

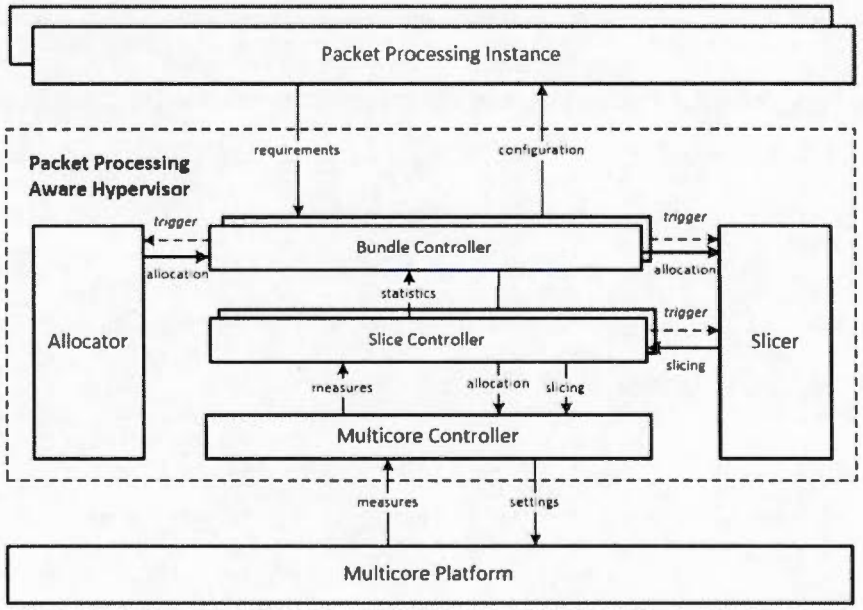


Figure 2.2: Architecture de *P2AH*

quement la portion de ressources nécessaires pour traiter les flux relatifs à une *slice*.

- **Les contrôleurs** : Cet ensemble se décompose autour d'une série (i) de contrôleurs décentralisés, gérant les instances de traitement et les *slices*, et (ii) d'un contrôleur centralisé, gérant les ressources matérielles. Les contrôleurs sont responsables de recueillir des mesures et des statistiques relatives aux entités et aux ressources qu'ils contrôlent, et d'appliquer les changements requis à celles-ci.

Les différentes entités constituant les blocs fonctionnels de l'hyperviseur seront décrites explicitement dans la suite de ce mémoire.

2.1.2 Fonctionnalités de P2AH

Une instance d'accélération de traitement de paquet est caractérisée par un profil qui décrit principalement le *pipeline* de traitement. Le *pipeline* regroupe l'ensemble des tâches et des ressources nécessaires pour acheminer les paquets. Par ailleurs,

le profil doit prendre en considération le trafic entrant. En effet, le trafic caractérisé par des ensembles de flux correspondants aux *slices* réseau, requiert différents types d'accélérations. À titre d'exemple certains flux doivent être filtrés avant d'être transmis à travers le port de sortie. Le filtrage nécessite éventuellement une analyse de plusieurs entêtes du paquet et une classification par rapport aux champs d'entêtes. Généralement, des algorithmes de recherche et des accélérateurs matériels peuvent être utilisés pour la classification des paquets. Aussi, il faut noter que les flux varient au cours d'une période de temps donnée, par conséquent, la capacité de traitement ne sera pas la même pour les instances d'accélération. Dépendamment du contexte, parfois on observe un trafic dense, d'où un besoin de filtrer plusieurs flux, et d'autres fois le trafic est moins dense et le besoin en filtrage diminue. L'*Allocator* considère ces critères de traitement définis au niveau des profils ainsi que les exigences du trafic pour déterminer une combinaison de ressources optimales pour une accélération donnée. Vu que les ressources matérielles disponibles sont limitées, il est indispensable de gérer les ressources tout en respectant les exigences de chacune des instances d'accélération.

Une fois les ressources sont allouées aux instances, le *Bundle Controller* met à jour la configuration de l'instance qui lui est directement rattachée en fonction de l'allocation définie par l'*Allocator* puis déclenche l'exécution du *Slicer*. Ce dernier prend la relève pour partager les instances d'accélérations en fonction des besoins des *slices* réseau. L'objectif principal du *Slicer* est de déterminer la portion des ressources associée aux instances à allouer aux traitements des *slices*.

Les réseaux actuels sont partagés entre différentes entités et plusieurs applications. Ces entités et ces applications qui partagent la même infrastructure matérielle correspondent à un ensemble de flux, en d'autres termes une *slice* réseau. Une *slice* réseau est acheminée de différentes manières à travers le réseau. En effet, les flux relatifs à la *slice* donnée peuvent être sujets à une commutation de paquet, une

inspection des données, un changement d'entête et un routage tout au long de son acheminement à travers le réseau. De ce fait, différents types d'accélération s'appliquent aux flux d'une même *slice*. Le *Slicer* assure le partage des instances d'accélérations afin que chaque *slice* bénéficie du traitement de paquet requis. Par la suite, le schéma de partage est pris en compte par chaque *Slice Controller* associé à une *slice*. Enfin, les contrôleurs décentralisés relatifs aux instances et aux *slices* communiquent les changements relatifs à l'allocation et au partage des ressources au contrôleur de la plateforme matérielle, le *Multicore Controller*. Ce dernier programme et configure les ressources de la plateforme en conséquence. Il agit directement sur les ressources matérielles pour générer les instances d'accélération et instaurer l'isolation et le partage de ces ressources.

Les processus d'allocation et de partage visent ainsi à offrir un environnement optimal pour les instances d'accélération de traitement de paquet. À travers ces processus, on distingue principalement deux types de partage de ressources. D'une part, nous avons (i) un *partage spatial* des ressources caractérisé par l'allocation des ressources aux instances. Ce mécanisme offre des ressources dédiées à chacune des instances, ces ressources ne peuvent être sollicitées que par les traitements d'une seule instance. Le partage spatial nécessite des opérations d'isolation afin de garantir que le trafic traité au niveau d'une instance donnée ne perturbe pas le traitement du trafic des autres instances ; d'autre part, nous avons (ii) un *partage temporel* des ressources caractérisé par le partage des instances entre plusieurs *slices*. Ce mécanisme partage les ressources dédiées pour traiter les flux appartenant aux différentes *slices*. Les ressources sont ainsi sollicitées pour le traitement de plusieurs *slices* au niveau d'une instance donnée. Afin de garantir un tel partage, des opérations d'ordonnancement sont requises. En effet, les *slices* sont en concurrence pour le traitement au niveau d'une instance. Par conséquent, un temps de traitement est réservé à chacune des *slices* pour satisfaire ses besoins.

Le mécanisme employé par *P2AH* présente un moyen de virtualisation, d'isolation des fonctions d'accélération de traitement de paquet et de gestion des ressources matérielles disponibles selon la dynamicité du trafic. Ce mécanisme ajuste ainsi selon le besoin les paramètres d'allocation et de partage. En effet, à partir des mesures et des statistiques récoltées par les différents contrôleurs des ressources (i.e., *Bundle Controller*, *Slice Controller* et *Multicore Controller*), les gestionnaires des ressources (i.e., l'*Allocator* et le *Slicer*) s'exécutent pour adapter l'allocation et le partage des ressources. Pour cela, des *seuils de performances* sont utilisés pour vérifier si les paramètres actuels correspondent aux exigences actuelles. Le dépassement des seuils déclenche impérativement l'un des deux gestionnaires de ressources, dans certains cas les deux gestionnaires sont activés successivement (voir figure 2.3).

L'*Allocator* qui bénéficie d'une vue globale sur le système, ce qui veut dire qu'il supervise les instances d'accélération suivant toutes les ressources disponibles, utilise une métrique et un seuil différent que ceux utilisés au niveau du *Slicer*. Contrairement à l'*Allocator*, le *Slicer* n'agit qu'au niveau d'une instance donnée, ce qui revient à dire qu'il bénéficie d'une vue restreinte sur les ressources, d'où l'utilisation d'un seuil différent. Ainsi le *Slicer* ne peut surveiller que les métriques se rapportant aux différentes *slices*. Dans le cas où les seuils pour le partage des ressources sont dépassés, le *Slicer* est déclenché. Celui-ci calcule un nouveau schéma de partage pour les *slices* et leurs les instances. Par contre, si les seuils réservés pour l'allocation ne sont plus respectés, l'*Allocator* puis le *Slicer* s'exécutent pour réguler le système. La surveillance se fait très régulièrement au cours du temps afin de sensibiliser le système pour répondre rapidement face aux différents changements et perturbations qui s'y produisent.

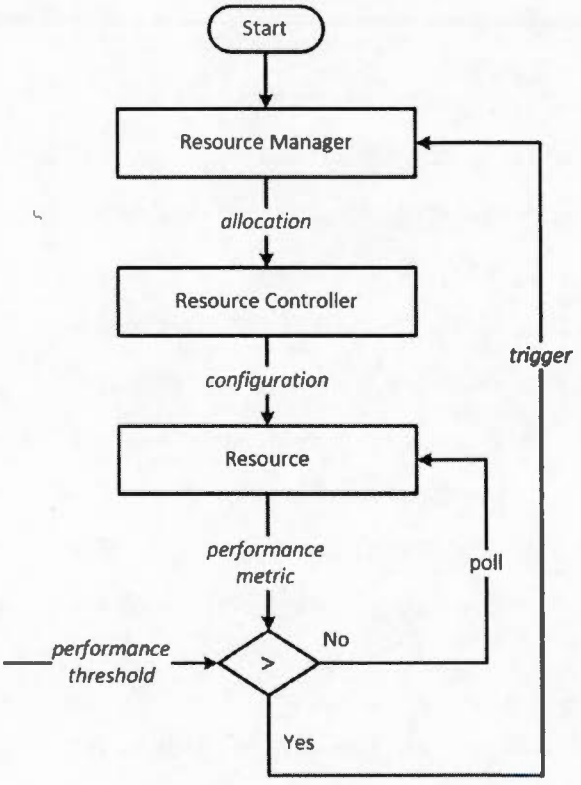


Figure 2.3: Processus de gestion des ressources de *P2AH*

2.2 Gestion des ressources

P2AH implémente des gestionnaires de ressources qui appliquent une stratégie de gestion de ressource dynamique selon les besoins en accélération de traitement de paquet et selon la dynamicité du trafic. Dans cette optique, suivant les besoins en accélération, les ressources disponibles seront libérées et utilisées par les gestionnaires pour être allouées aux autres instances ayant besoin de plus de ressources.

2.2.1 Gestionnaire d'allocation

L'objectif principal du gestionnaire d'allocation (*Allocator*) est de trouver l'association optimale entre les tâches de traitement de paquets et les ressources. L'en-

semble de ces associations constituera, par la suite, une instance d'accélération de traitement de paquets.

Tandis que les stratégies d'allocation conventionnelles reposent sur la ressource comme étant une unité partageable ou non partageable, la stratégie proposée considère un ensemble de ressources dépendantes qu'on caractérisera par *bundle* de ressources. Lorsqu'un paquet est reçu, une suite de tâches lui seront appliquées, ces tâches peuvent être associées à une ou plusieurs ressources. Par exemple, un routage nécessite principalement une extraction de l'adresse IP destination, une recherche dans la table de routage et une modification de l'entête du paquet avant de le transmettre à travers l'interface de sortie. Ces opérations forment un chemin de traitement de paquet et mettent en jeu un ensemble limité de ressources telles que les unités de traitement et les structures de recherches au niveau de la mémoire. Ainsi, une dépendance existe entre les ressources constituant un unique chemin de traitement. Ignorer cette dépendance aura une incidence négative sur le traitement de paquet. Un *bundle* peut être alors assimilé à une configuration de ressources de la plateforme multi-cœurs. Par conséquent, une instance représente le *bundle* le plus adéquat pour satisfaire les contraintes et les exigences d'une accélération de traitement de paquet.

Le gestionnaire d'allocation vise ainsi à déterminer la configuration la plus adéquate parmi les configurations existantes pour répondre au mieux aux besoins en accélération des *slices* réseau. Dans cette perspective, nous proposons une stratégie d'allocation des ressources aux instances qui s'articulent autour de trois phases importantes (voir figure 2.4) :

1. *Phase de modélisation* : Association des tâches d'accélération du traitement de paquets aux ressources.
2. *Phase d'évaluation* : Évaluation des différentes configurations multi-cœurs selon les besoins en accélération.

3. *Phase d'allocation* : Allocation optimale des ressources aux différentes instances.

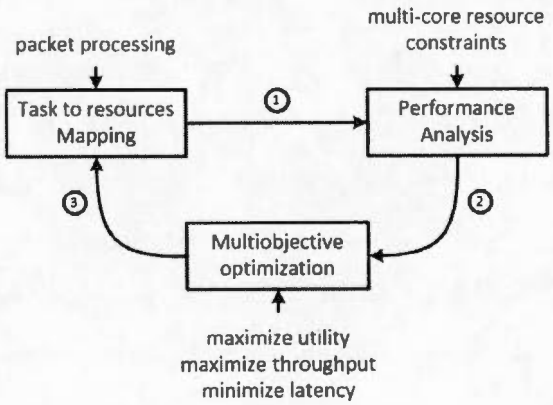


Figure 2.4: Stratégie d'allocation des ressources

Cette stratégie vise à offrir une allocation de ressources qui garantit principalement les propriétés suivantes :

- Maximiser le gain d'allocation pour chaque instance ; une instance doit bénéficier des ressources nécessaires afin de répondre aux besoins en accélération de traitement de paquet. L'allocation doit garantir le même profit en termes d'affectation des ressources pour chaque instance. Le minimum requis doit être assuré dans un premier temps, puis les ressources disponibles peuvent être allouées pour équilibrer le partage des ressources de la plateforme multi-cœurs.
- Maximiser la capacité de traitement de chaque instance d'accélération ; une instance doit traiter le trafic entrant, il est important de bénéficier d'une capacité de traitement de paquet permettant de supporter plusieurs flux. Les ressources allouées doivent garantir une capacité optimale afin de prévenir les blocages et de répondre aux exigences du trafic.
- Minimiser le délai de traitement de chaque instance d'accélération ; Une instance doit assurer une accélération qui respecte les contraintes du traitement de paquet. Certaines fonctions réseau présentent des exigences strictes en termes

de latence de traitement. L'allocation doit garantir un délai de traitement minimum qui respecte ces exigences, dépendamment du trafic entrant.

Phase de modélisation

Durant cette phase, il s'agit de trouver une combinaison de ressources qui reflète une configuration de la plateforme multi-cœurs pour une instance d'accélération. Considérant un traitement de paquet défini par le profil de l'instance d'accélération, nous pouvons le décomposer en plusieurs tâches, chaque tâche peut solliciter une ou plusieurs ressources. Par exemple, pour une classification suivant un ou plusieurs entêtes d'un paquet, nous pouvons solliciter la mémoire de recherche interne de la plateforme multi-cœurs ou un classificateur dédié. Aussi, il est important de mentionner que plusieurs tâches peuvent solliciter la même ressource. Par exemple, l'extraction des champs d'entête et la modification des entêtes peuvent être réalisées par un même *core* (unité de traitement) du processeur multi-cœurs. Ainsi, il est possible d'avoir différentes associations entre les tâches et les ressources de traitement de paquet (voir figure 2.5).

À travers les associations, nous constatons qu'il est possible d'avoir un ensemble de configurations pour supporter une accélération du traitement de paquet d'une instance. Toutefois, il est évident que chaque configuration présente des spécifications différentes en termes de capacité et de délai. En effet, une classification réalisée au niveau logiciel (mémoire DRAM) présente un délai plus important qu'une classification réalisée au niveau matériel (classificateur dédié). De plus, une tâche de traitement réalisée par plusieurs *cores* en parallèle offre une capacité de traitement très supérieur comparée à une tâche réalisée au niveau d'un seul *core*. Cependant, nous avons précisé qu'un chemin de traitement de paquet rassemble plusieurs tâches et ressources dépendantes l'une des autres. Par conséquent, il est indispensable d'évaluer les configurations déduites pour chacune des

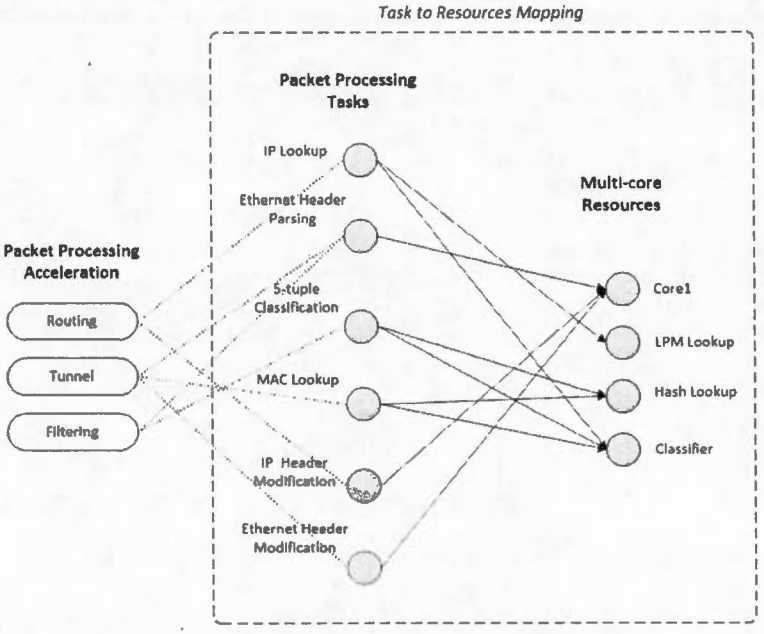


Figure 2.5: Schéma d'association des tâches aux ressources

instances d'accélération afin de déterminer la configuration la plus optimale, qui satisfait au mieux les propriétés citées auparavant.

Phase d'évaluation

Durant cette phase, il s'agit de déterminer les besoins en accélération des différentes instances en vue de sélectionner la configuration la plus optimale. En effet, les *slices* réseau renferment plusieurs flux qui nécessitent différentes accélérations de traitement de paquet. En fonction de ces flux, nous allons définir des contraintes en termes de capacité de traitement et de latence de traitement. La capacité de traitement fait référence au nombre de paquets qu'une instance donnée peut supporter pendant une période de temps; dans ce qui suit, nous allons considérer le nombre de paquets par seconde. La latence de traitement fait référence au délai de traitement d'un paquet de bout-en-bout, de la réception à la transmission. Dans ce qui suit, le délai sera mesuré au niveau de l'ensemble des

ressources qui constitue un chemin de traitement. Dans cette optique, nous allons utiliser la théorie du *Network Calculus* pour calculer les bornes déterministes (ou pire cas) sur la capacité et le délai de traitement (Le Boudec et Thiran, 2001). Ces calculs nous permettront d'analyser les performances pire-cas d'une instance d'accélération de traitement de paquet afin de déterminer ses besoins ; en fonction des *slices* partageant l'instance et de leurs flux respectifs, nous allons repérer les limites sur la capacité et sur le délai maximal pour le traitement. À partir de cette analyse, nous pouvons déduire la configuration nécessaire pour chaque instance de traitement. Les informations sur les limites du traitement sont modélisées grâce à des fonctions, telles que les *courbes d'arrivée* qui modélisent le trafic et les *courbes de service* qui quantifient le service garanti au niveau de chaque ressource tout au long du chemin de traitement (Thiele *et al.*, 2002).

On définit par f un flux dont l'ensemble des paquets est traité par la même instance d'accélération. Les paquets sont traités par les mêmes ressources suivant le même ordre au niveau d'un chemin de traitement. On note $A_f(t)$ le nombre de paquets du flux f qui arrivent durant un intervalle de temps $[0, t]$. Toute courbe de trafic $A_f(t)$ admet des courbes d'arrivée, quantité maximale et minimale de paquets pouvant arriver dans un intervalle de temps de durée t. La courbe de trafic $A_f(t)$ est ainsi contrainte par les courbes d'arrivée α_f^l et α_f^u qui vérifient la relation suivante :

$$\alpha_f^l(t-s) \leqslant A_f(t) - A_f(s) \leqslant \alpha_f^u(t-s) \qquad \forall 0 \leqslant s \leqslant t \tag{2.1}$$

$\alpha_f^l(\triangle)$ et $\alpha_f^u(\triangle)$ présentent respectivement une borne inférieure et une borne supérieure pour le nombre de paquets reçus d'un flux f pendant un intervalle de temps de durée $\triangle$. Ainsi, pour tout $\triangle > 0$:

$$\alpha_f^l(\triangle) \leqslant \alpha_f^u(\triangle) \quad et \quad \alpha_f^l(0) = \alpha_f^u(0) = 0 \tag{2.2}$$

Par ailleurs, nous admettons que le système, c'est-à-dire le chemin de traitement

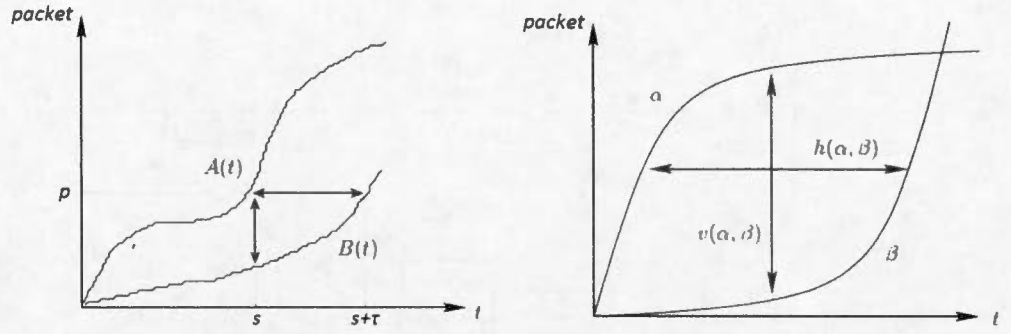
correspondant à une configuration de la plateforme multi-cœurs, doit garantir un délai limite (*deadline*) de traitement de bout-en-bout d_f pour chaque flux f. Généralement le délai limite est défini en fonction de l'application de traitement de paquet, il reflète les requis à respecter pour maintenir les performances du traitement de paquet. Ainsi, le délai limite de bout-en-bout présente le délai maximal durant lequel tous les paquets d'un flux f sont traités après leurs arrivées respectives.

Les ressources constituant le chemin de traitement de paquets supportent une ou plusieurs opérations de traitement. Ces opérations sont caractérisées par une courbe de service. Chaque ressource de traitement garantit une courbe de service minimum et maximum durant une période de temps donnée. Par exemple, on suppose que $C(t)$ est le nombre de paquets qui doivent être filtrés au niveau d'un classificateur c pendant un intervalle de temps $[0, t]$. Les courbes de service minimum et maximum β_c^l et β_c^u correspondant au classificateur c respectent l'inégalité suivante :

$$\beta_c^l(t-s) \leqslant C(t) - C(s) \leqslant \beta_c^u(t-s) \tag{2.3}$$

$$\forall 0 \leqslant s \leqslant t \quad et \quad \beta_c^l(0) = \beta_c^u(0) = 0$$

Dans l'objectif d'évaluer une configuration de la plateforme multi-cœurs en fonction des *slices* réseau et des flux correspondants, on considère les limites bornées pour la capacité et pour le délai au niveau de chaque ressource de traitement (voir figure 2.6a et 2.6b). Les contraintes sont exprimées en fonction des distances entre la courbe de service minimum $\beta(t)$ et la courbe d'arrivée $\alpha(t)$ correspondantes aux courbes de trafic $A(t)$ et de service $B(t)$; la distance horizontale $h(\alpha, \beta)$ signifie que le paquet qui arrive à l'instant s est traité à l'instant $s + \tau$. La distance verticale $v(\alpha, \beta)$ illustre le nombre de paquets p à traiter à l'instant s. On définit le délai maximum pour un flux f et la charge maximale au niveau d'une ressource



(a) Courbe de trafic et courbe de service (b) Contraintes

Figure 2.6: Évaluation des performances en fonction des contraintes

c par les inégalités suivantes :

$$delai_{max} \leqslant h(\alpha, \beta) = \sup_{u \geqslant 0}\{inf\{\tau \geqslant 0 : \alpha^u(u) \leqslant \beta^l(u+\tau)\}\} \tag{2.4}$$

$$charge_{max} \leqslant h(\alpha, \beta) = \sup_{u \geqslant 0}\{\alpha^u(u) - \beta^l(u)\} \tag{2.5}$$

En outre, nous avons stipulé qu'un chemin de traitement est constitué par un ensemble de ressources. Généralement, un paquet reçu doit enchainer les traitements au niveau de l'ensemble des ressources d'un chemin de traitement pour être transmis. Ainsi, il est important de pouvoir déterminer le délai de bout-en-bout ainsi que la charge au niveau de chaque ressource pour pouvoir évaluer une configuration donnée de la plateforme multi-cœurs. Dans cette perspective, nous nous basant sur une méthode du *Network Calculus* qui utilise une algèbre $(min, +)$, connue pour leurs applications aux systèmes à événements discrets (Baccelli *et al.*, 1992). Dans cette algèbre, les définitions de courbe d'arrivée et de courbe de service minimum s'expriment à l'aide de la convolution. Le *Network Calculus* permet de propager ces contraintes au niveau du chemin de traitement afin de calculer les bornes sur les performances, c'est à dire le délai de bout-en-bout et la charge au niveau de chaque ressource (voir figure 2.7).

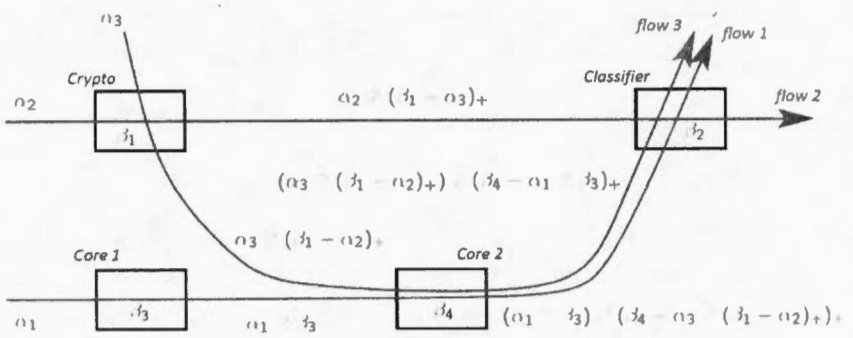


Figure 2.7: Exemple de propagation des contraintes au niveau des chemins de traitement de flux pour une configuration de ressources

Un flux, contraint par α, qui est traité par une ressource caractérisée par un service minimum β, ressort contraint par $\alpha \oslash \beta$ (déconvolution). La concaténation de deux ressources de services minimums respectivement β_1 et β_2 offre un service minimum de $\beta_1 * \beta_2$ au flux (convolution). Enfin, si deux flux f_1 et f_2 de courbes d'arrivée respectivement α_1 et α_2 sont traités par une ressource de service minimum β, alors la courbe de service pour le flux f_1 est $\beta_1 = (\beta - \alpha_2)_+$ [1]. À partir de ces règles de combinaison et de propagation, il est possible de déterminer le délai de bout-en-bout ainsi que la charge nécessaire au niveau de chaque ressource de la configuration pour traiter l'ensemble des flux.

Pour chaque instance, nous réalisons une analyse de flux pour déterminer la configuration la mieux adaptée pour répondre aux besoins d'accélération de traitement de paquet. Nous proposons les algorithmes de l'outil DISCO Network Calculator (Schmitt et Zdarsky, 2006) pour déterminer les besoins en accélération de traitement de paquet, plus particulièrement les bornes pour le délai et la charge. En fonction des bornes, nous déterminant la quantité de ressources qu'il faut

1. Notation : $x_+ = max(x, 0)$

allouées pour l'instance donnée. Pour illustrer cette phase d'évaluation, on considère un profil d'instance présentant un simple routage de paquets qui consiste principalement à extraire l'adresse IP destination du paquet et à exécuter une recherche *lpm* (*longest prefix match*). On suppose une association de tâches et des ressources qui est décrite par une configuration C={*cores*, mémoire}, le *core* se charge des opérations d'extraction et la recherche s'effectue au niveau de la mémoire qui implémente un algorithme de LPM. On considère que cette instance doit traiter plusieurs flux de différentes *slices* et que l'analyse de flux révèle qu'il est nécessaire d'avoir une capacité d'extraction de 1000 paquets/s et d'exécuter 1500 recherches/s. Si on suppose qu'un *core* est capable de traiter 500 paquets/s et que la mémoire assure 1500 recherches/s, alors nous devons allouer au minimum deux *cores*. Ainsi, la configuration minimale dont a besoin l'instance pour assurer son accélération de traitement de paquet est B_c={2 *cores*, mémoire}. Dans ce qui suit, on propose l'appellation *bundle* pour faire référence à une configuration minimale qui satisfait les contraintes et les exigences du traitement de paquet.

Phase d'allocation

La configuration déterminée à partir de l'évaluation des performances correspond au *bundle* minimum pour satisfaire les contraintes et les exigences d'une accélération de traitement de paquet. À travers l'évaluation, nous avons constaté qu'il peut exister un ou plusieurs *bundle* pour répondre aux besoins d'accélération d'une instance donnée. En effet, une capacité supérieure au niveau d'un *bundle* peut offrir de meilleures performances qu'un *bundle* avec une capacité moindre. Durant cette phase, il s'agit de déterminer le *bundle* le plus optimal pour chaque instance par rapport aux autres instances d'accélération de traitement de paquet. Dans cette perspective il faut résoudre le problème d'allocation de ressources équitable et efficace. La complexité d'une telle allocation réside dans le fait que les besoins en accélération de traitement sont variables dépendamment du trafic et que les res-

sources sont limitées. On propose ainsi de nous baser sur la théorie des jeux pour remédier au problème d'allocation des ressources entre les différentes instances. On considère alors le contexte où chaque instance possède un budget prédéfini et doit miser sur un ensemble de ressources (Feldman *et al.*, 2005). On propose une simple enchère où le plus offrant remporte la mise (Koutsopoulos et Iosifidis, 2010).

On suppose que notre schéma d'allocation s'articule autour de n instances d'accélération et m ressources de traitement. Chaque ressource de traitement à un prix et peut être partagée entre différentes instances. Un *bid* (mise) est calculé suivant les différents *bundles* relatifs aux différentes instances. Si une instance i a besoin du *bundle* B_1={*cores*, mémoire LPM} et du B_2={*cores*}, son *bid* sera plus important pour les *cores* et la mémoire LPM que pour les autres ressources disponibles, par exemple. L'allocation déterminera ainsi combien de *cores* on va lui associer. Les instances sont stratégiques, chacune recherchant son "intérêt" en terme de traitement de paquet. Le prix d'une ressource est déterminé à partir de la totalité des *bids* que les instances ont placés. Si une instance i soumet un *bid* $b_{ij}|b_{ij} \geq 0$ pour obtenir une ressource j, le prix Y_j de la ressource j est alors le total des *bids* pour la ressource j :

$$Y_j = \sum_{i=1}^{n} b_{ij} \tag{2.6}$$

Par ailleurs, on suppose le vecteur de *bids* de l'instance i est $b_i = (b_{i1}, ..., b_{in})$. Le budget de l'instance i est alors, le total de *bids* placés sur m ressources :

$$X_i = \sum_{j=1}^{m} b_{ij} \tag{2.7}$$

Le schéma d'allocation se présente comme $a = (r_1, ..., r_n)$, où $r_i = (r_{i1}, ..., r_{im})$ avec r_{ij} la fraction de partage de la ressource j allouée à l'instance i. La fonction d'utilité de l'instance i est caractérisée par la fonction U_i des fractions $(r_{i1}, ..., r_{im})$

de chaque ressource reçue par l'instance i. On en déduit alors la fonction linéaire de profit présenté comme suit :

$$U_i(r_{i1}, ..., r_{in}) = q_{i1}w_{i1}r_{i1} + ... + q_{im}w_{im}r_{im} \quad (2.8)$$

La fonction de profit s'exprime en fonction des fractions r_{ij}, $w_{ij} \geq 0$ qui réfère à la préférence de l'instance i pour la ressource j et de $q_{ij} \in 1, 0$ qui décrit si la ressource j est demandée ou pas. Dans notre contexte, la préférence pour une ressource est exprimée en fonction des différents *bundles* requis pour l'instance d'accélération. Les ressources appartenant au *bundle* qui offre les meilleures performances seront favorisées par rapport aux autres, d'où une préférence importante.

Comme nous avons supposé qu'une instance est stratégique, les instances cherchent ainsi à obtenir l'allocation qui maximise leurs fonctions d'utilité respectives. Par conséquent, nous devons déterminer les *bids* équitablement et efficacement. Nous proposons alors l'utilisation de l'algorithme 1 de *best response* (León et Navarro, 2011) adapté pour la détermination des *bids* pour le schéma d'allocation décrit auparavant. Du point de vue d'une instance i, si le total des *bids* des autres instances placés en chacune des ressources j est y_j, alors la "meilleure réponse" d'une slice i au système est la solution du problème d'optimisation suivant :

$$maximiser \quad U_i(\frac{b_{ij}}{b_{ij} + y_j}) = \sum_{j=1}^{m} q_{ij}w_{ij}\frac{b_{ij}}{b_{ij} + y_j} \quad (2.9)$$

$$avec \quad \sum_{j=1}^{m}(b_{ij}) \geq Xi \quad et \quad b_{ij} \geq 0$$

On considère que chaque instance i a besoin d'un minimum d'allocation ϕ_i au niveau de chaque ressource. Ainsi, le *bid* minimum à placer est :

$$b_{ij} \geq \frac{y_j\phi_i}{1 - \phi_i} \qquad avec \quad \phi_i \in (0, 1] \quad (2.10)$$

Algorithme 1 : Algorithme d'allocation des ressources

Données : ϕ : *bid* minimum de l'instance i.

Résultat : $\{b_{1:m}\}$: vecteur d'allocation pour l'instance i.

début

$X \longleftarrow \sum_{j=1}^{m} b_{ij}$ /* budget de l'instance i */

$M \longleftarrow \{y_j : q_j = 1\}$ /* prix des ressources demandées */

$M \longleftarrow triCroissant(M)$ /* tri croissant des y_j de M */

/* calcul du plus grand k */

tant que $\frac{\sqrt{y_k}}{\sum_{i=1}^{k}\sqrt{y_i}}(X + \sum_{i=1}^{k} y_i) - y_k \geqslant \frac{y_j\phi}{1-\phi}$ **faire**

pour $b_j \longleftarrow 0 for j > k, and 1 \leqslant j \leqslant k)$ **faire**

$b_j \longleftarrow \frac{\sqrt{y_j}}{\sum_{i=1}^{k}\sqrt{y_i}}(X + \sum_{i=1}^{k} y_i) - y_j$

retourner $(b_1, ..., b_j)$

Par ailleurs, il est important de mentionner un concept important dans la théorie des jeux, ce concept est connu sous le nom de *Nash equilibrium* (équilibre de Nash). L'équilibre de Nash fait référence à un état d'équilibre et de stabilité où personne ne souhaite changer de stratégie (allocation). Dans notre contexte et suivant le schéma d'allocation proposé, pour atteindre cet état d'équilibre, il faut calculer un vecteur de *bids* $(b_1, ..., b_m)$ où pour tout $1 \leq i \leq m$, b_i est la meilleure réponse au système, et pour tout autre vecteur de *bids* b'_i :

$$U_i(b_1, ..., b_i, ..., b_m) \geq U_i(b_1, ..., b'_i, ..., b_m) \tag{2.11}$$

Pour évaluer l'équilibre de Nash, nous considérons l'écart entre le vecteur de *bids* initial qui formalise le besoin ou la demande de chaque instance, par rapport au vecteur de *bids* déterminé. Plus l'écart est faible et plus l'allocation est à la fois

équitable et efficace.

La phase d'allocation vise ainsi à déterminer le *bundle* le plus optimal suivant les besoins des instances et les exigences du traitement de paquet. Ainsi, les fractions de ressources allouées aux différentes instances d'accélérations seront considérées comme dédiées au traitement de paquet relatif à l'instance correspondante. Une fois l'allocation réalisée, les tâches de traitement de paquet sont associées aux différentes ressources pour constituer le *bundle* qui caractérisera l'instance d'accélération.

2.2.2 Gestionnaire de partage

L'objectif principal du gestionnaire de partage (*Slicer*) est de déterminer les proportions de ressources nécessaires aux *slices* pour satisfaire leurs besoins en accélération de traitement de paquet. En effet, les instances doivent traiter les flux correspondants aux *slices*. Le gestionnaire de partage veille à garantir aux flux un traitement de paquet "équitable" au niveau des instances. Le partage des instances se traduit principalement par un ordonnancement des paquets au niveau des ressources de traitement (*bundle*). Dans le cas où plusieurs *slices* sont en concurrences pour le même *bundle*, il faut fournir un moyen pour prévenir les blocages et de garantir une affectation optimale des ressources (voir figure 2.8).

Une *slice* regroupe un ensemble de flux. Chaque flux nécessite un traitement particulier, par exemple, un routage, un filtrage ou une encapsulation. Chaque *bundle* reflète une instance responsable d'une accélération de traitement de paquet. Plusieurs flux associés à différentes *slices* peuvent solliciter les ressources du même *bundle*. Pour mieux illustrer le problème de partage des ressources, nous supposons un *bundle* ayant une capacité de traitement de 1 Gb/s et deux principaux flux associés à deux *slices* distinctes qui transmettent des paquets à des débits, respectivement de 0.6 Gb/s et 0.9 Gb/s. Il est évident que le débit appliqué à

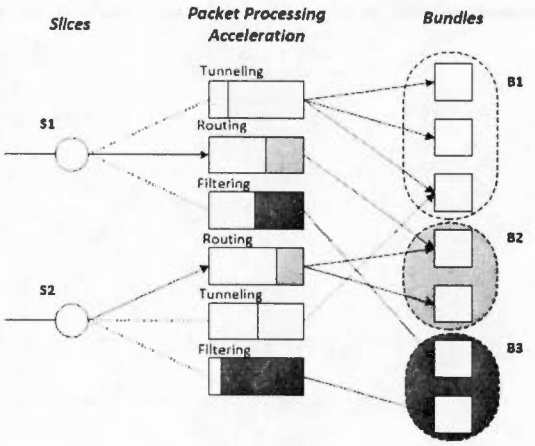


Figure 2.8: Schéma de partage des instances

l'entrée du *bundle* est nettement supérieur à la capacité. Dans ce cas de figure, comment partager les ressources d'une manière "équitable" ? Le moyen le plus évident serait de réserver 0.5 Gb/s à chacun des flux. Dans ce cas, le traitement du flux le plus important est affecté. Si au contraire, on réserve, respectivement aux flux mentionnés précédemment, 0.4 Gb/s et 0.6 Gb/s, nous allons affecter le traitement des deux flux. Un partage "équitable" doit ainsi être défini en fonction d'un contexte particulier. Dans notre contexte, nous jugeons qu'un partage est équitable s'il respecte les propriétés suivantes :

- Minimiser la congestion au niveau des ressources ; un volume de trafic important peut causer des blocages (*bottlenecks*) au niveau des ressources. Ce blocage cause la perte de paquets durant le traitement des flux. Les flux correspondant à une slice donnée ne doivent pas connaitre une forte perte de paquets afin de maintenir la stabilité du traitement de paquet.
- Maximiser le minimum de partage pour chaque *slice* ; les flux respectant les contraintes liées aux ressources ne doivent pas subir de détérioration du traitement en faveur des flux importants qui peuvent éventuellement provoquer une congestion au niveau des ressources.

Dans cette optique, nous proposons une stratégie de partage de ressources basé sur le concept du Max-Min *fairness* (Marbach, 2002). Chaque instance d'accélération de traitement de paquet est gérée indépendamment. Le gestionnaire de partage reçoit périodiquement les mesures correspondant aux nombres de paquets entrants pour chaque *slice*. En fonction de ces mesures, le gestionnaire ajuste dynamiquement le partage des ressources entre les différentes *slices*. Cette stratégie vise à prévenir les blocages au niveau des ressources des instances afin de maintenir les performances du traitement de paquet.

On considère N *slices* qui partagent une instance i ayant une capacité C_i. La capacité reflète la capacité de traitement du *bundle* alloué par le gestionnaire d'allocation pour l'instance i. Une *slice* k, où $1 \leqslant k \leqslant N$, x_k le nombre de paquets que la *slice* peut traiter au cours d'une période donnée au niveau de l'instance i, et s_k le nombre de paquets à traiter qui lui est alloué. Dans notre contexte, nous nous intéressons au nombre de paquets à traiter par seconde. Soit $(s_1, ..., s_N)$ le vecteur de partage possible où :

$$\sum_{k=1}^{N} s_k \leq C_i \tag{2.12}$$

Le vecteur de partage est dit max-min équitable, s'il est possible d'augmenter le partage d'une slice p sans réduire le partage d'une autre slice q avec un partage de $s_p \leqslant s_q$. Ainsi, la *slice* qui demande le minimum obtient le partage le plus grand possible sans pour autant gaspiller les ressources. Dans l'objectif de déterminer ce vecteur de partage équitable, nous calculons l'allocation des ressources selon la formulation suivante :

$$\begin{cases} si & x_k > \frac{C_i}{N} \quad alors \quad r_k = \frac{C_i}{N} \\ sinon & r_k = x_k \end{cases} \tag{2.13}$$

Cette formulation conduit à un partage équitable entre les différentes *slices* au niveau d'une instance i comme définie précédemment. Pour le gestionnaire de

Algorithme 2 : Algorithme de partage des ressources

Données : x_k : plus petite demande dans $\{x_{1:N}\}$ correspondante à la *slice* k.

Résultat : s_k : partage possible pour la *slice* k.

début

tant que $N \geq 0$ **faire**

si $x_k \leqslant \frac{C_i}{N}$ **alors**

$s_k \longleftarrow x_k$

si $x_k \geq \frac{C_i}{N}$ **alors**

$s_k \longleftarrow \frac{C_i}{N}$

$N \longleftarrow N - 1$

$C \longleftarrow C - s_k$

$\{x_{1:N}\} \leftarrow \{\{x_{1:N}\} \backslash x_k\}$

partage, nous nous sommes inspirés de l'algorithme Max-Min décrit dans (Radunović et Le Boudec, 2007) pour proposer l'algorithme 2.

Conclusion

P2AH est un hyperviseur conçu principalement pour les architectures à processeur multi-cœurs sur lesquelles des instances d'accélérations sont générées et adaptées de façon à satisfaire les besoins et les contraintes strictes du traitement de paquet. Dans ce chapitre, nous avons présenté la conception et les fonctionnalités de *P2AH*. Par la suite, nous avons décrit les concepts de base utilisés par les principaux composants de *P2AH* pour la gestion des ressources. Nous avons décrit en premier, le gestionnaire d'allocation qui utilise la théorie du *Network Calculus* pour déterminer les besoins en accélération de traitement de paquet en fonction des exigences du trafic, puis qui applique la théorie des jeux pour trouver l'allocation de ressources optimale pour chaque *slice* réseau. Ensuite, nous avons décrit le

gestionnaire de partage qui se sert du concept de Max-Min *fairness* pour partager les ressources allouées à une instance entre les différentes *slices* qui la partagent. L'adaptabilité des concepts de base de *P2AH* se reflète à travers son implémentation sur une architecture multi-cœurs et la définition des paramètres de contrôle nécessaire pour assurer son propre fonctionnement.

CHAPITRE III

IMPLÉMENTATION DE P2AH

Introduction

Dans ce chapitre, nous allons nous intéresser à l'implémentation de *P2AH* sur une plateforme multi-cœurs basée sur le processeur Andey MPPA-256 de Kalray [1]. L'objectif de ce chapitre est de présenter les mécanismes servant à l'intégration de *P2AH* pour gérer les ressources du processeur multi-cœurs. Nous présenterons l'architecture de traitement de paquet déployé sur le MPPA-256, nous justifierons et nous analyserons les différents choix techniques qui nous ont aidés à virtualiser le processeur et à mettre en place notre hyperviseur. L'implémentation réalisée à ce niveau servira ultérieurement pour les évaluations.

3.1 Description du MPPA AccessCore Evaluation Kit

L'environnement de développement MPPA AccessCore Evaluation Kit se compose d'une machine virtuelle exécutant un système d'exploitation hôte CentOS7 64 bits et du MPPA Accesscore SDK version 1.4 avec un simulateur du MPPA. Cet environnement nous permet de développer et d'exécuter des applications de traitement de paquet qui serviront pour la génération et la gestion des instances d'accélérations de traitement de paquet. Dans cette section, nous présenterons

1. http ://www.kalrayinc.com/

l'architecture du processeur MPPA-256 ainsi que ces principales caractéristiques.

3.1.1 Architecture du processeur Andey MPPA-256

Le MPPA-256 est un processeur multi-cœurs faisant partie de la famille des *many-cores* (voir figure 3.1). Comme son nom l'indique, le MPPA-256 comporte 256 processeurs à usage général de type VLIW (*Very Long Instruction Word*) réparties sur 16 *Compute Clusters*. Le *Compute Cluster* est l'élément clé du MPPA-256, il inclue 16 *cores* pour les traitements des données, une mémoire partagée, un *core* système (RM) pour superviser l'exécution des tâches et les opérations de transfert de données, des interfaces NoC (Network on Chip) pour les communications avec les *clusters* voisins et une DMA pour le transfert des données entre la mémoire partagée et le NoC (ou au niveau de la mémoire partagée). Les *clusters* sont interconnectés par la technologie NoC qui véhicule les données (D-NoC) et les informations de contrôle (C-NoC) entre les *clusters* et le système d'entrée/sortie (*I/O subsystem*). Le MPPA-256 incorpore 4 *Clusters* d'entrées/sorties basés sur un processeur *Quad-Cores* (RM) qui servent d'interfaçage entre les *clusters* et les autres périphériques externes principalement la mémoire DRAM externe, le bus PCIe, les interfaces 1/10 Gigabit Ethernet.

3.1.2 Programmation du processeur Andey MPPA-256

Le SDK du processeur MPPA-256 inclut différents outils de développement notamment la programmation *C/C++* et *SigmaC*, une suite de simulateurs, de débogueurs, de *profilers* et d'outils de traces ainsi que plusieurs systèmes d'exploitation et de pilotes de périphériques. Le MPPA-256 supporte principalement deux modèles de programmation :

- *Dataflow Programming Model* : L'application est décrite par un graphe d'agents qui communiquent à travers des canaux de communication. Ce modèle offre un

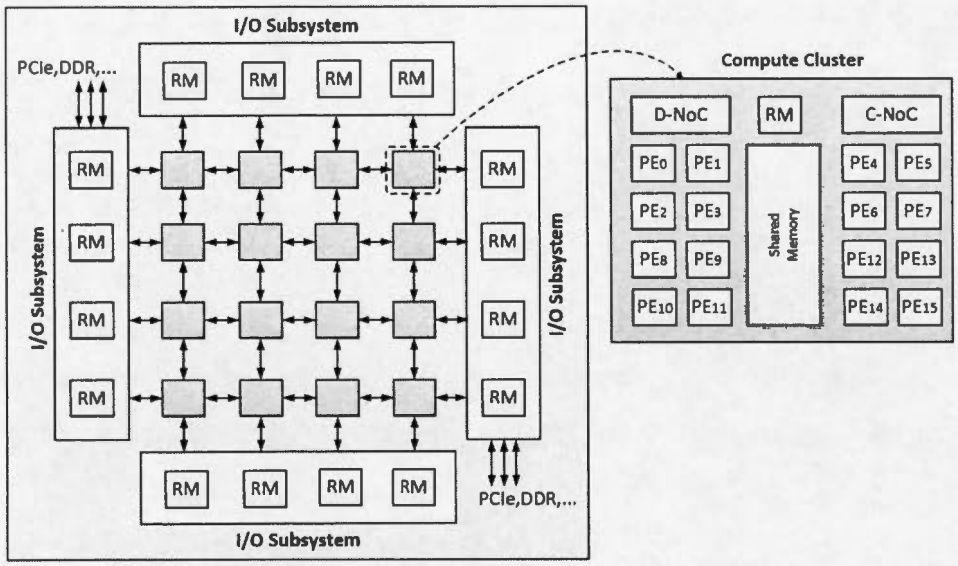


Figure 3.1: Architecture du processeur Andey MPPA-256

haut niveau d'abstraction au développeur et lui permet de programmer indépendamment des détails matériels. Une allocation automatique des ressources est appliquée lors de la compilation de l'application (*cluster*, mémoire, NoC).

- *POSIX Programming Model* : L'application est décrite par un ensemble de processus échangeant des données à travers des interfaces de communications. Ce modèle implique que le développeur gère la majorité des détails matériels comme la communication entre les processus, les *threads* ainsi que leurs assignations aux ressources physiques.

Suivant le modèle de programmation POSIX, l'application est développée en *C* en utilisant la librairie standard POSIX pour la gestion des processus et la librairie IPC pour assurer la communication inter-processus. L'application peut s'exécuter au niveau des *clusters* d'entrée/sortie qui supportent le système d'exploitation RTEMS et au niveau des *clusters* de traitement qui supportent un système d'exploitation spécialisé, le NodeOS. Par ailleurs, les applications qui utilisent le modèle de programmation *Dataflow* sont développées grâce au langage

de programmation fonctionnel propriétaire *SigmaC*.

Dans la perspective d'implémenter des applications de traitement de paquet, nous allons utiliser l'API OpenDataPlane supportée par Kalray [2]. L'API offerte utilise un code source qui se base sur le modèle de programmation POSIX. En effet, ce modèle de programmation offre plus de flexibilité et de liberté pour développer les applications de traitements de paquets. Toutefois, il faut gérer minutieusement tous les aspects liés à l'interconnexion et à l'échange de données entre les différentes ressources ou blocs de traitement afin de garantir les meilleures performances pour notre application de traitement de paquet.

3.2 Traitement de paquet sur le processeur MPPA-256

Le traitement de paquet conventionnel réalisé au niveau des processeurs réseau et des processeurs ASICs nécessite l'exécution d'une série de tâches prédéfinies lors de la réception des paquets à travers l'interface réseau. Les tâches consistent principalement à charger le paquet au niveau de la mémoire, analyser l'entête du paquet pour extraire les champs requis pour la classification au niveau des structures de recherches, la modification de l'entête du paquet (s'il y a lieu) et enfin la transmission du paquet à travers le port de sortie. Ces tâches sont généralement réalisées en parallèle pour un ensemble de paquets reçus.

Dans l'optique de générer des instances d'accélération de traitement de paquet sur le processeur MPPA-256, nous nous inspirons des *pipelines* de traitement de paquets des accélérateurs matériels cités auparavant pour mettre en place un modèle de traitement de paquet modulable, flexible et performant. De ce fait, il faut prendre en considération certains critères jugés primordiaux pour le traitement de paquet :

---

2. https ://github.com/kalray/odp-mppa.git

- La gestion des accès rapides à la mémoire. Le traitement de paquet repose entièrement sur des lectures et des écritures à partir des paquets chargés au niveau de la mémoire. Par conséquent, il est indispensable d'offrir des mécanismes accélérant les opérations de lecture et d'écriture au niveau de la hiérarchie de mémoire offerte par le processeur MPPA-256 ;
- Le support de composantes matérielles notamment pour les opérations de réception des paquets, de distribution des paquets entre les unités de traitement (*cores*), la classification des paquets et la transmission des paquets. Cela est avantageux pour avoir un traitement de paquet à haute performance. Ce critère dépend entièrement des caractéristiques de la plateforme matérielle utilisée ;
- Le parallélisme du traitement de paquet. Exécuter un ensemble de tâches en parallèle pour traiter plusieurs paquets permet d'accentuer la capacité de traitement et d'offrir de meilleures performances. Ce critère est fortement dépendant de l'architecture matérielle ;

Face aux exigences et aux contraintes du traitement de paquet, il est important d'examiner les fonctionnalités des ressources matérielles du processeur MPPA-256 pour concevoir une structure fonctionnelle flexible pour supporter les besoins en accélération des différentes instances.

3.2.1 Programmation du traitement de paquet

Il est vrai que les processeurs multi-cœurs sont moins performants que les processeurs dédiés. Toutefois, le degré de flexibilité offert au niveau des architectures multi-cœurs permet d'optimiser les performances et de répondre aux besoins des réseaux actuels. En effet, la technologie d'interconnexion (NoC) des différents *clusters* brise les contraintes liées à la rigidité des *pipelines* de traitement prédéfinis au niveau des accélérateurs matériels. Le traitement de paquet peut être alors arrangé de différentes façons selon les besoins.

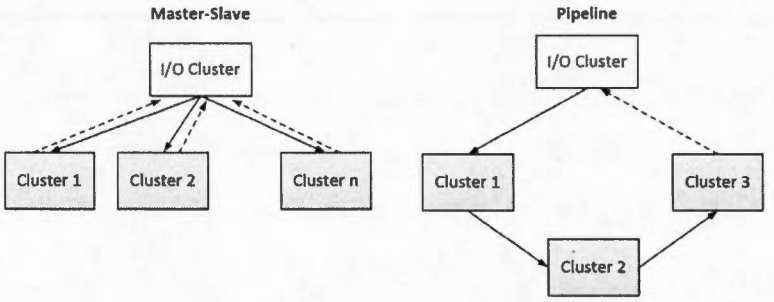


Figure 3.2: Modes de programmation du traitement de paquet

Grâce à son architecture, le processeur MPPA-256 supporte deux approches de programmation (voir figure 3.2). Une approche basée sur une architecture maitre-esclave et une autre basée sur une architecture en *pipeline* :

- Maître-esclave : Le traitement de paquet est contrôlé d'une manière centralisée, un processeur faisant office de maître distribue le travail entre plusieurs processeurs esclaves. Le même traitement est exécuté au niveau de tous les processeurs esclaves. Le traitement est ainsi réalisé de bout en bout par la même ressource. Ce mécanisme vise à réduire la communication entre les ressources et ne nécessite pas de synchronisation. L'avantage de cette approche est de maximiser la capacité du traitement.
- Pipeline : Le traitement de paquet est distribué sur plusieurs ressources, généralement des processeurs et des accélérateurs matériels. Le traitement est ainsi réalisé sur plusieurs étapes par différentes ressources. Toutefois, ce mécanisme nécessite une synchronisation entre les différentes ressources d'où un échange important entre les ressources, ce qui accentue la sollicitation de la mémoire. Par conséquent, le traitement peut être lourdement affecté si on utilise des ressources homogènes. L'avantage d'une telle approche est d'optimiser l'utilisation des ressources.

Il est aussi important de mentionner que ces deux approches peuvent être com-

binées et appliquées à différents ensembles de ressources. Par exemple, si nous disposons en plus des processeurs génériques, d'un pré-classificateur de paquets à l'entrée et d'un engin de hachage, il serait possible de réaliser un traitement en parallèle tout en sollicitant des composants matériels pour le traitement de paquet. Cela améliore considérablement les performances du traitement de paquet. Néanmoins, certaines architectures matérielles présentent des contraintes qui limitent la combinaison de ces deux approches. Dans notre contexte, nous utilisons des ressources homogènes représentées par l'ensemble des *cores* des *clusters*. De ce fait, nous optons pour une implémentation d'un algorithme maître-esclave au niveau du processeur MPPA-256.

3.2.2 Modélisation du traitement de paquet

Dans cette section, nous allons présenter et décrire un modèle flexible pour le traitement de paquet au niveau du processeur MPPA-256. Nous nous sommes inspirés de certaines architectures d'accélérateurs matériels pour proposer des blocs fonctionnels pour le traitement de paquet (voir figure 3.3). L'architecture en bloc servira de référence pour l'association des tâches aux ressources de la plateforme multi-cœurs (voir section 2.2.1 pour la description de la phase de modélisation).

L'architecture pour l'accélération de traitement de paquet s'articule essentiellement autour de quatre entités principales :

- ***Input Data Path*** : Entité responsable de la réception des paquets. Cette entité se constitue des blocs de traitement suivants :
  - *Input Interface Unit (IFU)* : présente les interfaces MAC pour la réception des paquets et se charge de la classification des paquets à l'entrée.
  - *Input Packet Manager Unit (IPMU)* : charge les paquets en mémoire, ordonnance et supervise les différentes opérations liées au traitement des paquets.
- ***Data Path Processing*** : Entité responsable des traitements des paquets,

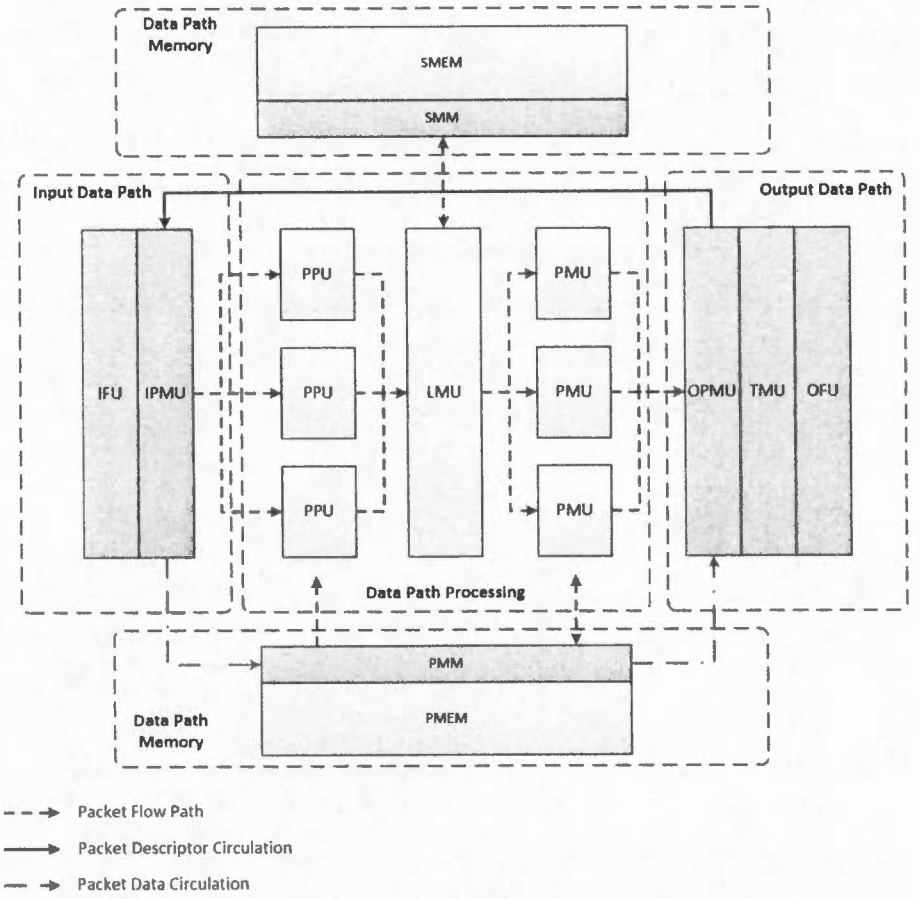


Figure 3.3: Architecture d'accélération de traitement de paquet pour le MPPA-256

notamment l'extraction des champs d'entêtes des paquets, l'exécution des recherches et la modification des entêtes des paquets. Cette entité se constitue des blocs de traitement suivants :

- *Packet Parsing Unit (PPU)* : analyse l'entête du paquet et extrait les champs pour la construction des clés de recherche.
- *Lookup Manager Unit (LMU)* : charge les clés, ordonnance et supervise les requêtes et les résultats des recherches.
- *Packet Modification Unit (PMU)* : applique les changements adéquats aux entêtes des paquets.

- ***Output Data Path*** : Entité responsable de la transmission des paquets. Cette entité se constitue des blocs de traitement suivants :
  - *Output Packet Manager Unit (OPMU)* : supervise l'état du traitement et se charge de l'ordonnancement des paquets à la sortie.
  - *Traffic Manager Unit (TMU)* : se charge de l'ordonnancement, la classification et le lissage des flux lors de la transmission des paquets.
  - *Output Interface Unit (OFU)* : présente les interfaces MAC pour la transmission des paquets.
- ***Data Path Memory*** : Entité responsable de la gestion des mémoires, notamment la mémoire des paquets et la mémoire de recherche. Cette entité se constitue des blocs de traitement suivants :
  - *Search Memory Manager (SMM)* : gère les structures de recherches, exécute les requêtes de recherche puis transmet les résultats.
  - *Search Memory (SMEM)* : Présente la mémoire de recherche incluant les différentes structures de recherche.
  - *Packet Memory Manager (PMM)* : gère le chargement et le déchargement des paquets, et supervise les accès aux données des paquets.
  - *Packet Memory (PMEM)* : Présente la mémoire des paquets. À la réception du paquet, son contenu est sauvegardé au niveau de cette mémoire.

Pour l'architecture décrite auparavant, nous proposons d'associer les entités ainsi que les blocs fonctionnels aux ressources du processeur MPPA-256 (voir figure 3.4). Nous avons donc le *cluster* d'E/S qui joue le rôle de maître et l'ensemble des *clusters* de traitement qui jouent le rôle des esclaves. Les NoCs sont utilisés pour assurer la communication entre le maître et ses esclaves.

Afin d'illustrer le mécanisme de traitement de paquet, le *cluster* d'E/S reçoit les paquets à travers les interfaces réseaux et les distribue entre les différents *clusters* de traitement. Toutes les tâches liées à l'analyse de l'entête, les recherches et la

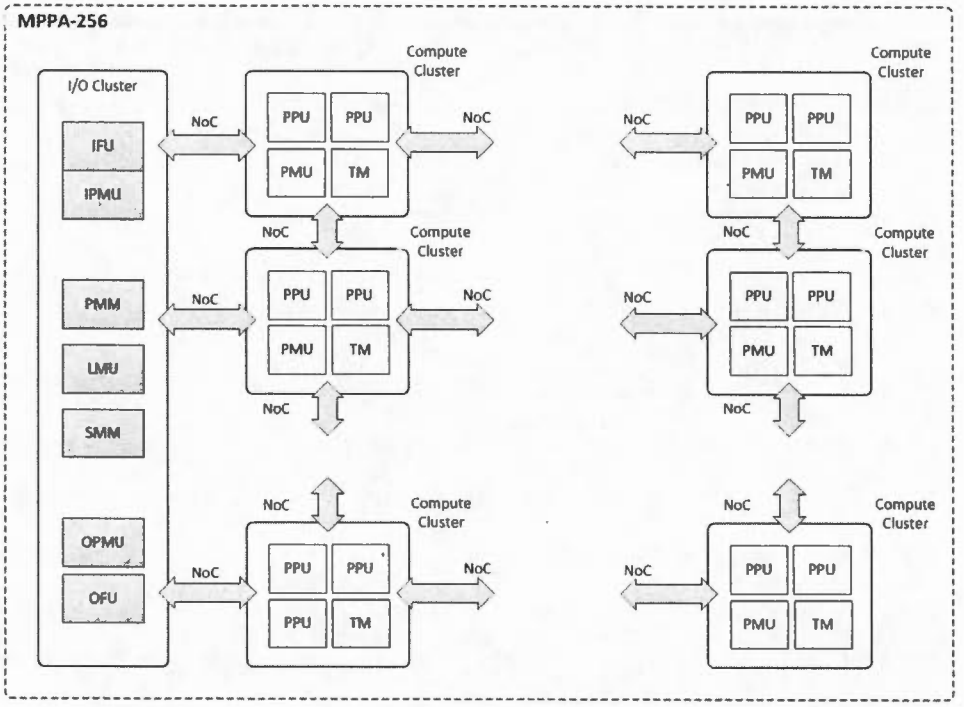


Figure 3.4: Association des blocs de traitement aux ressources du MPPA-256

modification de l'entête sont exécutées au niveau du *cluster* de traitement. Les recherches sont déclenchées par les *cores* des *clusters* de traitement et traités au niveau des *cores* du *cluster* d'E/S. Ce dernier sert d'interfaçage avec les mémoires de recherches où sont implémentés les algorithmes de recherches et où sont stockées les structures de recherches. Certes, ce mécanisme ralentit le traitement de paquet, car cela nécessite de nombreux passages par les *clusters* d'E/S pour solliciter les périphériques externes, mais reste néanmoins indispensable. D'un autre côté, pour réduire les accès mémoire durant les traitements qui impliquent l'accès au contenu du paquet, nous avons opté pour une stratégie qui consiste à charger les entêtes des paquets au niveau de la mémoire partagée au niveau du *cluster*. Ainsi, on réduit considérablement les passages par les *cluster* d'E/S pour accéder à la mémoire des paquets. Une fois le traitement des paquets est achevé au niveau du *cluster* de traitement, les paquets sont transmis à travers l'interface de sortie via le *cluster*

d'E/S.

La recherche et la classification des paquets réalisées au niveau des mémoires DRAM et SRAM (éventuellement) doivent satisfaire certaines exigences afin d'assurer un traitement de paquet à haut débit. En effet, les algorithmes de recherche et de classification jouent un rôle très important et sont considérés comme les points critiques du traitement de paquet. Ainsi, l'implémentation des algorithmes peut impacter considérablement les délais du traitement. Par conséquent, il est indispensable de prendre en considération les paramètres suivants : le type de recherche utilisée (*exact match*, *longest prefix match*) ; le nombre de champs utilisés pour la recherche (*Ntuple*) ; l'algorithme de recherche utilisé (*trie*, *hash*) ; le nombre de recherches réalisées simultanément (*Nlookup*) ; pour notre implémentation, nous proposons divers mécanismes de recherche et de classification dépendamment des exigences des applications.

Au niveau de l'implémentation, nous utiliserons l'OpenDataPlane pour déployer divers mécanismes nécessaires à la mise en place de cette architecture de traitement de paquet. Nous avons implémenté une couche d'abstraction pour la gestion des différentes entités d'accélération du traitement de paquet. Cette couche est responsable de la création, de l'initialisation et de l'exécution des blocs de traitements. Par ailleurs, une application de traitement de paquet utilise une ou plusieurs tables pour la classification des paquets. Nous avons alors développé des librairies pour supporter différents types de classification et de recherche au niveau des mémoires :

- *Longest Prefix Match (LPM)* : Algorithme de LPM qui associe chaque route à un préfixe. Pour chaque préfixe on associe un résultat qui est renvoyé lors de l'opération de recherche. L'algorithme implémenté est basé sur le *binary trie*. Chaque route est divisée sur 32 groupes, chaque groupe correspond à une lon-

gueur spécifique du préfixe. La recherche s'effectue en parcourant ces différents groupes.

- *Exact Match* : Un algorithme de *Hash* est implémenté pour associer chaque route à une clé enregistrée au niveau de la mémoire. La fonction de hachage utilisée est basée sur la fonction de *Jenkins*[3].
- *NTuples Classification* : La recherche est réalisée sur *Ntuple* sur plusieurs règles ACL et renvoie le résultat correspondant à la règle de plus haute priorité. L'implémentation se base sur une structure de *multibit tries* avec 8 niveaux. Suivant l'ensemble des règles un arbre est construit et divise l'ensemble de règles en plusieurs sous ensembles. Chaque sous-ensemble est identifié par un *bit mask* et possède son propre *trie*.

Les structures de recherches sont implémentées au niveau des mémoires externes. Pour la mise en place des algorithmes et des mécanismes de recherche, nous nous sommes basés principalement sur les algorithmes de recherches et de classification décrits par (Chao et Liu, 2007) ainsi que sur les modèles développés au niveau du DPDK[4].

3.3 Virtualisation du processeur MPPA-256

Suivant le modèle de traitement de paquet susmentionné, le processeur MPPA-256 doit offrir des mécanismes pour la virtualisation afin de supporter plusieurs instances d'accélération de traitement de paquet. Dans cette section, il s'agit de décrire les différents profils des instances d'accélération de traitement de paquet ainsi que les mécanismes d'isolation et de gestion des ressources utilisés.

---

3. http ://www.burtleburtle.net/bob/hash/doobs.html

4. http ://dpdk.org/doc/api/

3.3.1 Profils d'accélération de traitement de paquet

Les instances d'accélération générées au niveau du processeur MPPA-256 décrivent un traitement de paquet particulier. Pour notre implémentation, nous proposons trois profils d'accélération de traitement de paquet. L'objectif est d'avoir plusieurs combinaisons de ressources de traitement pour déterminer les modèles d'accélération de traitement de paquet qui peuvent être supportés au niveau d'une plateforme multi-cœurs virtualisée. Les profils de traitement de paquet proposés reflètent des fonctions réseau généralement utilisées dans les réseaux de centre de données notamment les fonctions de routage, de filtrage et de tunnel (encapsulation/décapsulation) :

- Routage : Cette fonction achemine les paquets en fonction de l'adresse IP de destination. Il s'agit d'un traitement réalisé particulièrement au niveau des routeurs dans les réseaux *backbones* et qui nécessite des performances élevées. Il s'agit de chercher le réseau de destination dans la table de routage d'appliquer les changements nécessaires à l'entête Ethernet du paquet pour le transmettre à travers la bonne interface de sortie. Généralement les tales de routage contiennent de nombreuses entrées ce qui complique la recherche. L'algorithme le plus utilisé pour le routage est le LPM. Les ressources sollicitées sont particulièrement les *cores*, les tables LPM, les tables de hachages et les interfaces réseaux.
- Filtrage : Cette fonction applique plusieurs filtres aux paquets avant d'autoriser ou refuser l'accès au flux correspondant. Il s'agit d'un traitement de paquet réalisé au niveau des pare-feux pour prévenir l'accès aux flux jugés malicieux. Le traitement nécessite l'analyse de plusieurs entêtes du paquet d'où l'utilisation des classificateurs *NTuples*. Les ressources sollicitées sont particulièrement les *cores*, les tables ACL et les interfaces réseau.
- Tunnel : Cette fonction applique une encapsulation (ou une décapsulation) au

paquet en *ingress* (ou en egress) pour les adapter au réseau de destination. Il s'agit de traitements réalisés au niveau des routeurs *edge* qui sont de plus en plus supportés au niveau logiciel. Le traitement nécessite principalement une recherche *Hash* pour déterminer en fonction de l'entête Ethernet les modifications nécessaires à apporter aux paquets (encapsulation/décapsulation). Les ressources sollicitées sont particulièrement les *cores*, les tables de hachages et les interfaces réseau.

Nous notons que les accélérations de traitement de paquets (voir annexe A et B pour la description des chemins de traitement et des structures de recherche) utilisées pour bâtir nos cas d'utilisations pour les trois profils d'instances sont en partie présentées par le document de spécification (Bro, 2014).

3.3.2 Isolation et gestion des ressources

Dans notre contexte, nous considérons un ensemble de ressources à allouer aux différentes instances d'accélération de traitement de paquet. Les interfaces réseaux ainsi que les partitions mémoires seront dédiés. En effet, il n'est pas logique d'allouer la mémoire dynamiquement vu que le traitement de paquet repose essentiellement sur les entrées au niveau des structures de recherches. Par conséquent, nous ne pouvons pas priver une instance d'espace mémoire durant les allocations de ressources. Aussi, nous proposons de dédier des interfaces virtuelles à chaque instance réseau afin de recevoir et de transmettre des paquets. Par ailleurs, nous allons allouer dynamiquement les *cores* des *clusters* de traitement. Vu que ces derniers prennent en charge la majorité des tâches de traitement, nous allons assigner un ensemble de *cores* aux traitements de paquet d'une instance d'accélération. Il s'agit d'associer dynamiquement les tâches de traitement de paquet aux *cores* du processeur MPPA-256. Nous définissons alors, un *Data Path Element* (DPE) comme l'unité de traitement minimale nécessaire pour prendre en charge

une accélération de traitement de paquet. La capacité de traitement d'un DPE varie selon les tâches qui lui seront associées. Un DPE sera en premier lieu dédié (partage spatial) à une instance donnée par le gestionnaire d'allocation, puis partagé (partage temporel) entre les différentes *slices* par le gestionnaire de partage (voir section 2.1.2). Ainsi une configuration multi-cœurs inclut principalement des interfaces réseau virtuelles, un espace mémoire pour les structures de recherches et un ensemble de DPEs. Un *bundle* caractérisera principalement la configuration multi-cœurs ayant le nombre optimal de DPEs (en terme de capacité de traitement) pour supporter l'instance d'accélération de traitement de paquet.

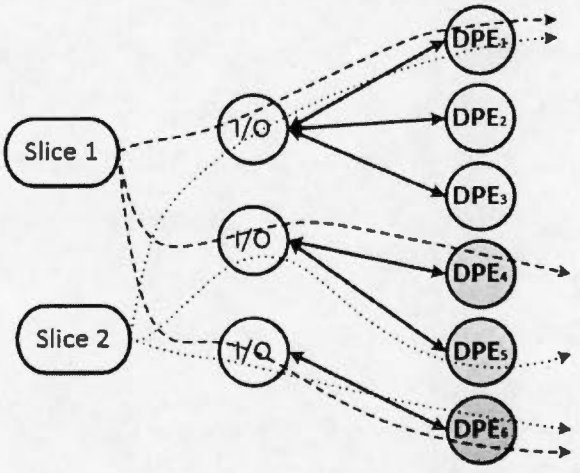


Figure 3.5: Mécanisme d'isolation des instances et des *slices* réseau

Afin d'assurer la virtualisation du processeur MPPA-256, il va falloir isoler les ressources relatives à chacune des instances en plus du partitionnement des ressources. En effet, il faut instaurer un mécanisme d'isolation efficace afin de prévenir qu'une instance n'affecte pas le traitement des autres instances. De même, il ne faut pas que les flux d'une *slice* interfèrent avec les flux des autres *slices*. Pour cela, nous avons implémenté une stratégie d'isolation qui consiste à associer les DPEs (caractérisés par un ensemble de *cores*) à une interface virtuelle correspondante à l'instance de traitement en question (voir figure 3.5). En d'autres termes, un

core associé au *cluster* d'E/S sera connecté directement à une interface virtuelle. Les paquets reçus à travers l'interface seront distribués entre les différents DPEs dédiés à l'instance d'accélération. Le *core* d'E/S maintient une table de correspondance entre les instances et la liste des DPEs associés. Lorsque le gestionnaire d'allocation modifie l'allocation pour les instances, la table de correspondance sera mise à jour. En outre, pour distribuer les paquets entre les différents DPEs, le *core* d'E/S implémente un algorithme d'ordonnancement qui se base sur un *modulo* (un algorithme de *Round Robin* classique). Enfin, pour isoler les flux des différentes *slices* nous avons incorporé un mécanisme de pré-classification au niveau du *core* d'E/S. Cette pré-classification se base sur les descripteurs de *slices* pour assurer la différentiation des flux. Pour l'implémentation, nous avons supposé que les flux appartenant aux *slices* sont identifiés à partir de l'interface de réception (port d'entrée) et un ensemble d'octets de l'adresse MAC source. Par exemple, une *slice S* définie par les interfaces (1,2,3) et les trois premiers octets 01 :CD :89 génère un ensemble de flux possédant les mêmes caractéristiques. Par exemple, les paquets reçus par l'interface 1 et ayant une adresse SMAC=01 :CD :89 :6A :23 :BE seront considérés comme des paquets appartenant à un flux correspondant à la *slice S*.

3.4 Intégration de P2AH

Nous sommes conscients que ces mécanismes d'isolation et de gestion des ressources introduisent un coût additionnel, ce qui peut impacter le délai de traitement de bout-en-bout. À travers les choix de conception de P2AH et les choix techniques de l'implémentation, nous faisons en sorte de minimiser le coût tout en assurant une accélération de traitement de paquet optimale satisfaisant les contraintes des ressources et les exigences du trafic. Dans cette partie, nous présentant notre environnement au complet (voir figure 3.6).

P2AH s'exécute sur un processeur hôte, nous avons dédié un processeur de la

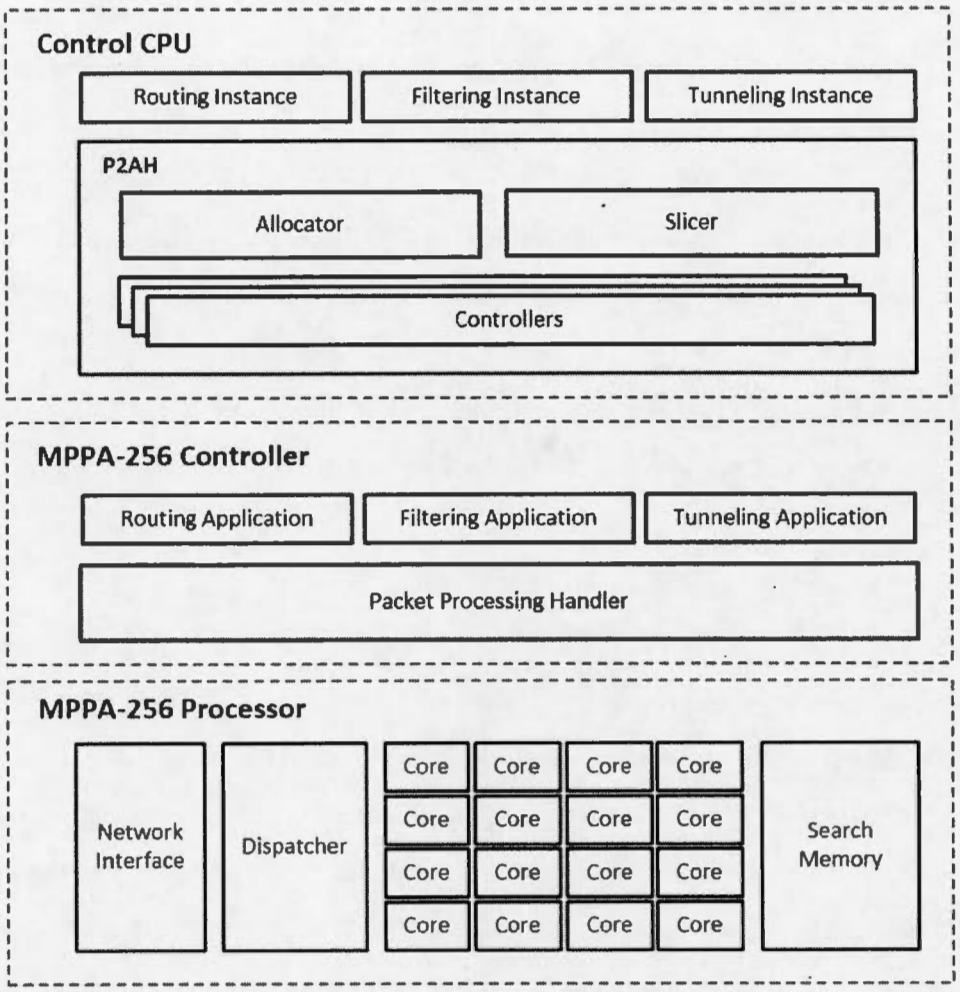


Figure 3.6: Architecture du système

machine virtuelle de l'environnement de développement MPPA AccessCore Evaluation Kit (voir section 3.1) pour exécuter les blocs fonctionnels de *P2AH* et générer les instances d'accélérations de traitement de paquet. Nous avons implémenté la couche d'abstraction particulièrement le *Packet Processing Handler* au niveau du processeur MPPA-256. Celui-ci implémente le modèle de traitement de paquet proposé pour le multi-cœurs (voir section 3.2.2). Aussi, le *handler* permet de contrôler les ressources sous-jacentes spécialement les interfaces réseau, le *dispatcher*, les *cores* et la mémoire de recherche. Le *dispacther* assure l'isola-

tion des instances et les flux correspondants aux *slices* réseau (voir section 3.3.2). Ce dernier est contrôlé par *P2AH* à travers les contrôleurs qui sont directement connectés aux gestionnaires d'allocation (*Allocator*) et de partage (*Slicer*). *P2AH* communique les décisions d'allocation et de partage des ressources au *handler* en utilisant un mécanisme de communication inter-processus basé sur une structure de mémoire partagée. De même, le *handler* renvoi les mesures et les statistiques capturées par les compteurs en éditant l'espace mémoire partagée. La lecture de ce bloc mémoire s'effectue très régulièrement par l'hyperviseur et par le contrôleur du processeur MPPA-256.

Conclusion

Dans ce chapitre, nous avons décrit l'implémentation et l'intégration de *P2AH* au niveau du processeur Andey MPPA-256 de Kalray. Comme nous pouvons le constater, il est fortement recommandé de développer un modèle flexible pour le traitement de paquet d'une part, et d'instaurer des mécanismes d'isolation et de partitionnement d'une autre part. En effet, *P2AH* vise particulièrement à adapter le traitement de paquet aux contraintes des ressources et aux exigences du trafic. En étudiant l'architecture du processeur MPPA-256, nous avons pu mettre en place une abstraction qui offre des moyens efficaces pour le contrôle des ressources sous-jacentes tout en garantissant la flexibilité du traitement de paquet. Grâce à cette implémentation, *P2AH* est capable d'associer les tâches aux ressources de traitement pour générer des instances d'accélération de traitement de paquet qui par la suite seront subdivisées entre les différentes *slices* réseau. Cette implémentation sera utilisée à des fins d'évaluation, de validation et d'études de modèles d'accélérations de traitement de paquet sur les plateformes multi-cœurs.

CHAPITRE IV

EVALUATION DU PACKET PROCESSING-AWARE HYPERVISOR

Introduction

Dans ce chapitre, nous allons procéder à l'évaluation de *P2AH* en termes de modélisation de traitement de paquet et de gestion des ressources. À travers l'évaluation de *P2AH*, nous voulons déterminer des modèles d'accélération de traitement de paquet à la fois flexible et performant pour les plateformes multi-cœurs virtualisées. En effet, nous considérons que cela permettra d'adapter les infrastructures réseaux actuelles aux nouvelles tendances de virtualisation et de programmabilité des réseaux. Nous allons ainsi décrire notre environnement de test. Ensuite, nous allons présenter et analyser les résultats relatifs aux tests. Durant l'évaluation nous allons comparer *P2AH* à d'autres approches existantes, principalement le *Network-Aware Hypervisor* (NAH).

4.1 Environnement de tests

Pour l'évaluation de *P2AH* nous utiliserons l'implémentation décrite dans le chapitre 3. Les tests sont réalisés sur le processeur Andey MPPA-256. Nous avons utilisé une configuration du processeur qui se compose principalement d'un *cluster* avec 16 *cores* et 3 interfaces pour la réception et la transmission des flux. Nous avons initialisé les tables au niveau de la mémoire (voir annexe B pour la descrip-

tion des structures de recherches), nous avons installé les entrées nécessaires pour l'acheminement des flux au niveau de chaque table. Pour générer le trafic correspondant aux *slices*, nous avons configuré des interfaces virtuelles connectées directement à l'environnement de la machine virtuelle Andey MPPA-256. Nous avons ainsi associé une interface virtuelle à chacune des trois instances d'accélération de traitement de paquet. La génération du trafic se fait grâce à l'outil TCPreplay [1] et Netmap (Rizzo, 2012).

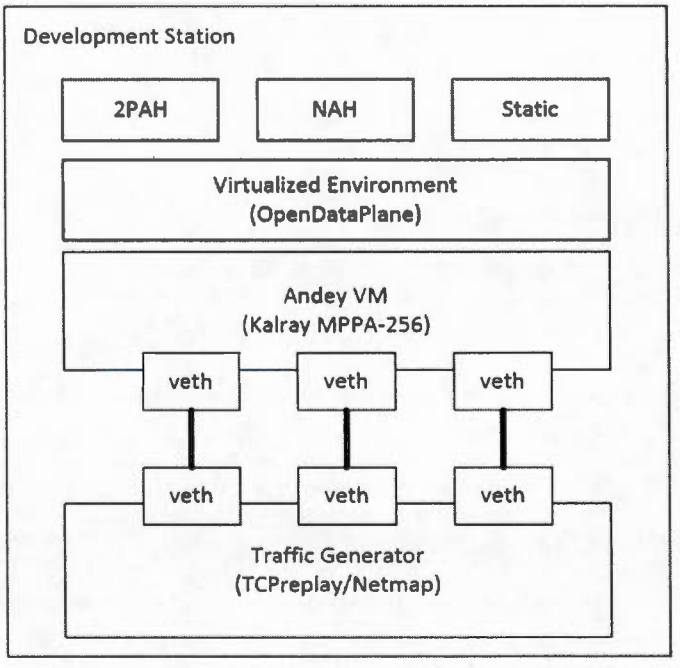


Figure 4.1: Banc de tests

Pour pouvoir étudier les points forts et les points faibles de *P2AH*, nous proposons de tester dans le même contexte et sous les mêmes conditions *NAH* ainsi qu'une approche statique (voir figure 4.1). *NAH* offre un mécanisme d'allocation de ressources avec trois principaux objectifs : (1) l'équité entre les *slices*, (2) la stabilité du traitement de paquet et (3) la prévention des congestions (voir section 1.3).

1. http ://tcpreplay.appneta.com/wiki/installation.html

NAH assure la gestion et le partage de *bundles* prédéfinis à l'avance et qui ne peuvent pas être réarrangés. Par contre, *NAH* offre un mécanisme dynamique pour le partage des *bundles* correspondants aux instances d'accélération de traitement de paquet dans notre contexte. En outre, l'approche statique consiste à fixer les ressources allouées aux *bundles* ainsi que leurs partages entre les différentes *slices*. Cette configuration statique reste inchangée tout au long du fonctionnement du système. Généralement, les ressources sont allouées de façon à satisfaire les exigences du trafic autant durant les conditions normales que durant les conditions critiques. Cette stratégie nécessite une étude au préalable, mais présente un risque important de gaspillage des ressources.

4.1.1 Paramètres de tests

Dans cette section, nous allons présenter les paramètres de tests relatifs aux gestionnaires de ressources, au trafic et à l'environnement virtualisé. Durant les tests, certains paramètres seront fixés, certains contrôlés et d'autres observés.

Pour l'évaluation, nous avons généré 3 différentes instances qui s'exécutent simultanément sur le processeur MPPA-256. Chaque instance présente une accélération de traitement de paquet avec une capacité de traitement en termes de paquets par seconde (voir tableau 4.1).

Tableau 4.1: Paramètres des instances

Instance	Traitement de paquet	Capacité par DPE
1	Routage	990pps
2	Filtrage	1100pps
3	Tunnel	1320pps

Ces instances seront partagées entre 2 à 5 *slices* réseau. Chacune des *slices* sol-

licite un chemin de traitement de paquet à travers l'instance d'accélération. Le profil d'une *slice* est défini en fonction des perturbations suivant X qui réfère à la demande en accélération pour le tunnel, Y qui réfère à la demande en accélération pour le filtrage et Z qui réfère à la demande en accélération pour le routage. Pour assurer la stabilité du système et prévenir les débordements importants quant à la capacité des instances, nous posons $(X, Y, Z) \in [0 - 40\%]$. Ainsi une *slice* peut avoir un profil $P = (X, Y, Z) = (20\%, 40\%, 30\%)$. Il est à noter aussi que le profil de la *slice* varie durant les tests en fonction du trafic entrant.

Tableau 4.2: Paramètres des gestionnaires de ressources

	NAH	**P2AH**
Période de prélèvement des mesures	1sec	1sec
Intervalle de contrôle	*Équité* : 10sec *Stabilité* : 5sec *Congestion* : 1sec	- - -
Taux de pertes de paquet tolérées	10%	10%
Seuils de contrôle	- -	*Partage* : 5% *Allocation* : 9%
Période de gestion globale	-	20sec

Les algorithmes sont configurés avec des paramètres fixes qui ont été vérifiés lors des expérimentations et considérés comme les paramètres optimaux (voir section 2.2 et la section 1.3 pour la description des paramètres). En ce qui concerne les seuils de contrôle pour les métriques de performances, nous les avons choisis suivant les paramètres souhaitables et requis pour maintenir un traitement de pa-

quet précis et performant (voir tableau 4.2). Les mesures quant aux nombres de paquets reçus par seconde par instance et par *slice* sont prélevés chaque période d'une seconde (1sec). Les mécanismes de contrôles pour *NAH* sont déclenchés sur des intervalles de temps différents. Le critère de performance choisi est le taux de pertes de paquet, on considère qu'une valeur de 10% permet d'assurer un traitement de paquet stable et performant. Cette valeur est utilisée par le niveau de stabilité de *NAH*. Aussi cette valeur est utilisée par *P2AH* pour fixer les seuils de contrôle. On propose ainsi de déclencher le gestionnaire de partage à chaque fois qu'on atteint les 5% de taux de pertes. Le gestionnaire d'allocation sera quant à lui déclenché lorsque le taux de pertes atteint les 9%. Cette stratégie vise à prévenir la dégradation du traitement de paquet et assurer l'ajustement rapide de l'allocation des ressources.

En outre, nous assignerons le même nombre de DPEs ($\frac{16}{3} \sim 5$ DPEs) par instance lors de l'initialisation. Pour le gestionnaire de ressource *NAH* et l'approche statique, l'assignation ne change pas au cours du temps et caractérisera les *bundles* pour les instances. Par contre, l'*Allocator* au niveau du *P2AH* gère dynamiquement l'allocation des DPEs aux instances d'accélération de traitement de paquet.

Le trafic est généré à partir des fichiers *PCAP*[2] personnalisés suivant les flux des *slices*. Pour nos tests, nous avons créé plusieurs fichiers, chacun contient 12,000 flux. La taille des paquets est de 78 octets transmis à un débit de 62,4kb/s pendant une durée de 2 minutes. Au cours des évaluations, nous allons faire varier le débit ainsi que le nombre de flux transmis pour créer des perturbations de trafic (voir tableau 4.3). Ces perturbations sont importantes pour l'évaluation des mécanismes d'allocation des ressources. En effet, les variations de trafic impactent le traitement de paquet et par conséquent les mécanismes d'allocation sont en-

2. http ://www.tcpdump.org/

clenchés pour réguler le système suivant les nouvelles exigences et l'adapter aux nouvelles contraintes.

Tableau 4.3: Description des perturbations du trafic généré

Perturbations	**Description**
Très faible (0%)	10-50% de la capacité maximale
Faible (10%)	10-60% de la capacité maximale
Moyenne (20%)	10-70% de la capacité maximale
Forte (30%)	10-80% de la capacité maximale
Très forte (40%)	10-90% de la capacité maximale

Nous faisons ainsi varier le trafic en transmettant un ensemble de flux variable en fonction du type de perturbation. Par exemple, pour une perturbation moyenne (20%), les DPEs au niveau du processeur MPPA-256 peuvent être chargés de 10% jusqu'à 70% de leurs capacités de traitement maximales. Ainsi, pour un nombre de flux F transmis par seconde et une perturbation P, nous avons un débit d, c'est-à-dire un nombre total de flux émis par seconde à un instant t de :

$$d(t) = F.(1 + 2.P) \qquad P \in [10\%..90\%]$$

Sur un intervalle $[0..t]$, $d(t)$ constitue la courbe de trafic de notre système. Les flux F caractérisent l'ensemble des flux des *slices* réseaux, qui représentent le trafic minimal qui doit être accéléré. Ce trafic varie de 10% à 50% de la capacité maximale d'une instance d'accélération de traitement de paquet. Les perturbations P caractérise des flux additionnels des *slices* qui s'ajoute au trafic existant. Ce trafic fait ainsi varier le besoin en accélération des différentes *slices*. Ainsi, pour une perturbation forte (30%) et un ensemble de *slices* émettant à 50% de la capacité maximale d'une instance, nous calculons, au niveau de l'interface réseau correspondante, un débit variable entre 10% et 80% de la capacité maximale.

4.1.2 Profils de tests

Au cours des tests, nous faisons varier les profils des différentes *slices* ainsi que les perturbations de trafic dans le but de déstabiliser le système et observer le fonctionnement des mécanismes de gestion de ressources. Aussi, nous allons faire varier le nombre de *slices* qui partagent la même instance d'accélération de traitement de paquet. Nous faisons varier tous les paramètres mentionnés auparavant pour les trois approches ; les mécanismes de gestions de ressources P2AH et NAH, ainsi que l'approche statique. Nous allons nous intéresser aux évaluations suivantes :

- La modélisation du traitement de paquet : Nous allons évaluer l'allocation des ressources en fonction des différentes accélérations de traitement de paquet. Nous allons aussi étudier son impact sur les performances du traitement de paquet.
- L'optimisation des ressources : Nous allons évaluer la gestion des ressources par rapport aux exigences du trafic et aux contraintes matérielles. Nous allons étudier le partage des ressources et ses effets sur les ressources de traitements.
- Coût de la virtualisation : Nous allons évaluer le temps de réponse des mécanismes de gestion des ressources et étudier leur fonctionnement en fonction de la variation des exigences et des contraintes.

4.2 Résultats de tests

Dans cette section, nous allons observer différents paramètres et analyser les résultats des différentes évaluations pour pouvoir en tirer des conclusions concrètes.

4.2.1 Modélisation du traitement de paquet

En variant le besoin en accélération de traitement de paquet, nous allons comparer l'allocation des ressources pour les trois instances (voir figure 4.2). On observe

que le gestionnaire des ressources, plus particulièrement, le gestionnaire d'allocation (*Allocator*) offre plus de ressources aux instances de routage et de filtrage qui obtiennent respectivement 38% et 35% contre 27% pour l'instance de tunnel (encapsulation). D'un autre côté, un mécanisme d'allocation de ressources naïve aurait distribué les ressources équitablement soit 33% à chacune des instances. Plus le besoin en accélération de traitement de paquet augmente, plus l'allocation des ressources tend vers une allocation naïve. Toutefois, les ressources allouées aux instances de routage et de filtrage restent nettement supérieures à celles allouées à l'instance de tunnel.

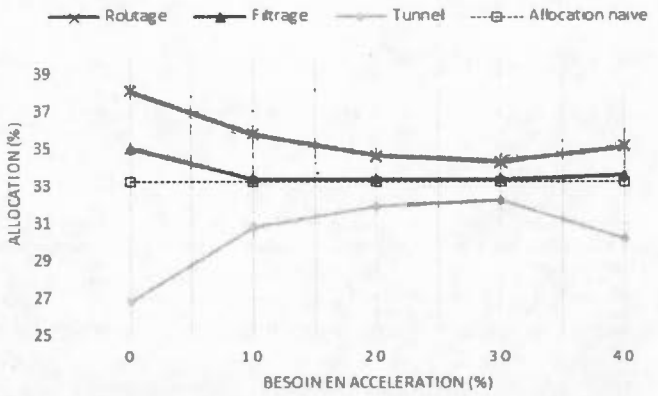


Figure 4.2: Modélisation des instances d'accélération de traitement de paquet

Pour des demandes en accélérations très faibles, *P2AH* dispose de la majorité des ressources, toutes les ressources sont disponibles. Ce dernier alloue ces ressources en fonction des requis pour le traitement de paquet. L'instance de routage utilise une recherche LPM qui est plus ou moins coûteuse. En effet, cette recherche est considérée comme complexe vu qu'elle est réalisée au niveau logiciel. Par conséquent, pour garantir de bonnes performances de traitement *P2AH* lui alloue plus de ressources de traitement. Ceci grâce au mécanisme d'évaluation introduit lors de l'allocation (voir section 2.2.1), qui à travers une analyse des flux détermine la

quantité de ressources nécessaire afin de répondre aux besoins du traitement de paquets (délai et charge de traitement). De même pour l'instance de filtrage. Par ailleurs, dans le cas de fortes demandes, les ressources disponibles sont minimales. Dans ce cas, *P2AH* tend à limiter l'allocation des ressources pour les instances.

Pour des demandes d'accélération de traitement de paquet qui atteignent les 40%, il est difficile de restreindre encore plus les ressources pour l'instance de routage, car le traitement de paquet ne sera plus stable. Par conséquent, on lui alloue plus de ressources pour l'adapter aux nouvelles exigences, d'où la croissance de la courbe d'allocation et la décroissance des autres courbes. Pour l'instance de routage, on constate que l'allocation ne doit pas descendre en dessous des 35%. Celle du filtrage se stabilise autour des 33% et celle du tunnel varie entre 27% et 31,5%. Ainsi, *P2AH* tient compte de la nature du traitement de paquet et ses requis pour allouer convenablement les ressources aux instances, contrairement aux mécanismes naïfs d'allocation de ressources.

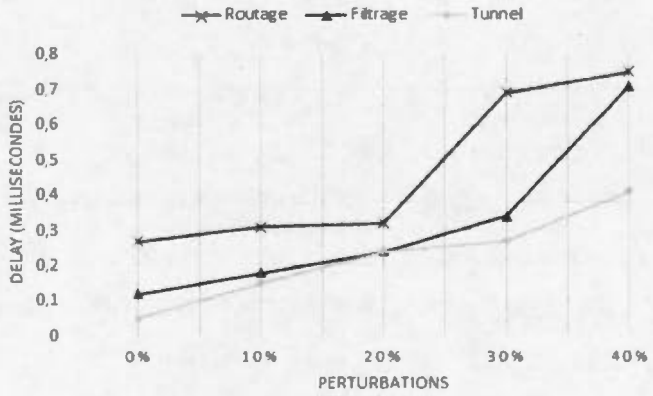


Figure 4.3: Performances des instances de traitement de paquet

En outre, on évalue le délai de traitement de paquet de bout-en-bout en fonction des perturbations de trafic (voir figure 4.3). On calcule le temps moyen par instance

durant les tests. On observe que plus les perturbations s'intensifient plus le délai de traitement augmente. Le délai correspondant à l'instance de routage passe approximativement de 0,28ms à 0,75ms. Celui de l'instance de filtrage croît de 0,1ms jusqu'à 0,7ms. Toutefois, on observe une légère augmentation du délai pour l'instance de tunnel, le temps de traitement passera de 0,05ms à 0,4ms.

Le délai moyen de traitement des instances de routage et de filtrage est très supérieur à celui de l'instance de tunnel. On constate qu'à partir une perturbation de trafic supérieure à 20%, le délai de traitement pour le routage est très important et est jugé insatisfaisant pour répondre aux exigences du trafic. L'instance de filtrage quant à elle peut supporter des perturbations allant jusqu'à 30%. Pour des perturbations très élevées, on constate que le délai de traitement de paquet pour les deux instances n'est plus satisfaisant. En effet, les algorithmes de recherche et de classification utilisés par les deux instances et qui sont implémentés au niveau logiciel ralentissent le traitement de paquet contrairement à la fonction de hachage utilisé par l'instance de tunnel. Ces algorithmes coûteux en termes de calculs limitent l'accélération de traitement de paquet des instances de routage et de filtrage et par conséquent la virtualisation de ces fonctions. Le processeur MPPA-256 ne dispose pas d'accélérateurs matériels pour effectuer ces types de classification ce qui diminuerait considérablement le temps de traitement. De ce fait, on considère que les fonctions de routage et de filtrage ne sont plus stables ou virtualisables au niveau du MPPA-256 sous des perturbations qui dépassent les 20% pour le routage et les 30% pour le filtrage. L'utilisation d'accélérateurs matériels pour les tâches de classification est fortement requise.

Ainsi, nous pouvons conclure sur les différents modèles d'accélération de traitement de paquet. Le traitement de paquet qui sollicite des tâches de classification complexes nécessite plus de ressources, 33-38% de l'allocation des ressources. Cependant, les accélérations ne sont plus suffisantes sous des perturbations dépassant

les 20-30% et nécessitent des classificateurs matériels pour répondre aux besoins du traitement de paquet.

4.2.2 Optimisation des ressources

Dans un premier temps, nous évaluons le partage des ressources en mesurant le gain moyen par *slice* réseau. Nous faisons varier le nombre de *slice* et on calcule le gain pour les deux mécanismes de gestion des ressources, *P2AH* et *NAH*. Le gain caractérise la fonction d'utilité pour chaque *slice* (voir section 2.2.1) et représente spécifiquement le rapport entre les ressources allouées et les ressources demandées. Plus le gain est proche de 1 et plus le partage est équitable.

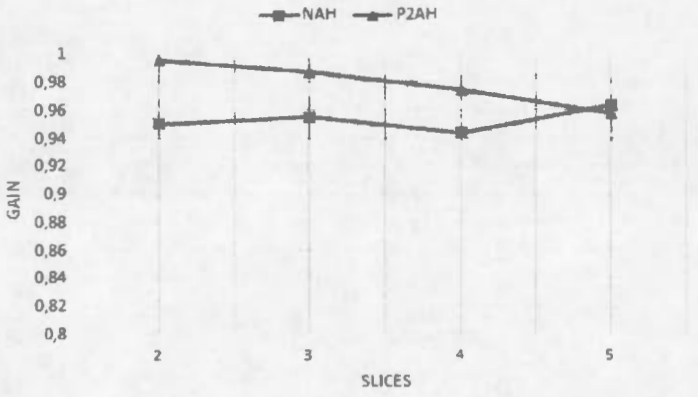
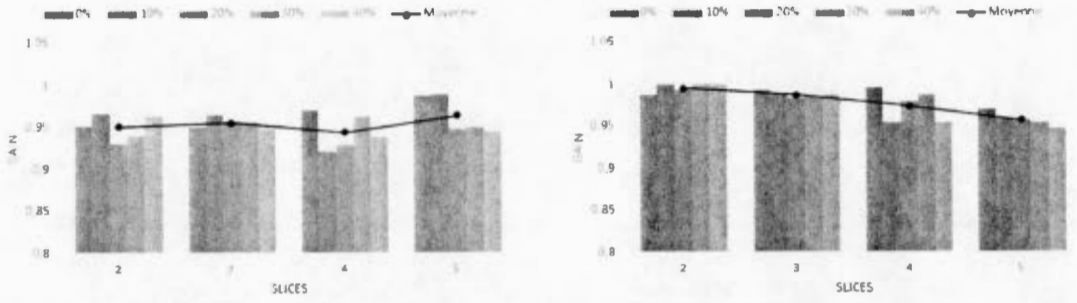


Figure 4.4: Gain moyen de gestion des ressources

On observe que le gain présenté par *P2AH* est supérieur au gain présenté par *NAH*, plus que 0,97 pour un nombre de *slices* inférieur ou égal à 4. *NAH* présente un gain moyen par slice avoisinant les 0,95. Toutefois, dans le cas où nous avons 5 *slices* qui partagent les instances d'accélération de traitement de paquet, P2AH et NAH convergent approximativement vers le même gain qui est de 0,96.

Les deux gestionnaires de ressources garantissent un gain élevé, ce qui prouve que

l'allocation des ressources satisfait la demande des différentes *slices*. Toutefois, *P2AH* se démarque en affichant un gain strictement supérieur, car celui-ci, en plus du partage des instances entre les différentes *slices*, il arrive à ajuster les instances en fonction des besoins des *slices*. En effet, *P2AH* redimensionne dynamiquement les ressources pour les instances et ainsi les ressources qui seront partagées entre les *slices* pour cette même instance correspondent pratiquement à leurs demandes respectives. Par contre, *NAH* présente des instances dont les ressources sont prédéfinies ce qui contraint le partage des ressources entre les *slices*, d'où un gain plus faible par rapport à celui de *P2AH*.



(a) NAH (b) P2AH

Figure 4.5: Gain de gestion des ressources en fonction des perturbations du trafic

Lorsqu'on observe le gain de gestion des ressources par rapport aux perturbations de trafic (voir figure 4.5), on remarque que *P2AH*, contrairement à *NAH*, maintient systématiquement un gain stable (avoisinant la moyenne) quelles que soient les perturbations du trafic. Ainsi, on considère que *P2AH* est robuste et qu'il assure un partage équitable des ressources. Toutefois, le gain a tendance à diminuer lorsque le nombre de *slices* qui partagent les mêmes instances augmente. Nous jugeons que le gain garantit par *P2AH* reste raisonnable et satisfaisant indépendamment des perturbations.

Par ailleurs, nous évaluons la stratégie de gestion des ressources en termes d'optimisation des ressources en fonction des perturbations de trafic (voir figure 4.6). Nous comparons les résultats relatifs aux gestionnaires des ressources à ceux d'une allocation statique. On remarque que pour des perturbations inférieures à 20%, les mécanismes d'allocation des ressources permettent d'avoir plus de 30% des ressources de disponibles contrairement à une allocation statique qui utilise la grande majorité des ressources. Au-delà d'une perturbation de 20%, on constate que l'écart se creuse et atteint approximativement les 30% pour une perturbation de 40%.

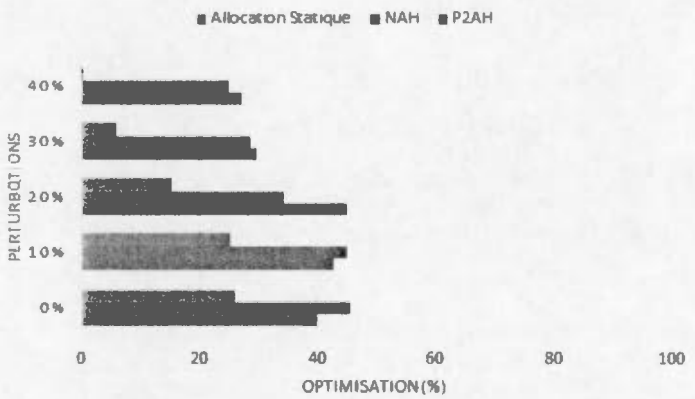
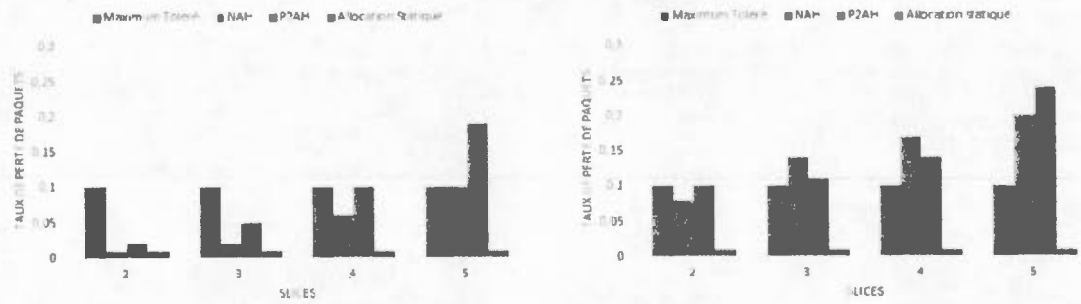


Figure 4.6: Optimisation des ressources

Une allocation statique réserve toutes les ressources disponibles pour le traitement de paquet d'où un gaspillage de ressources. Par contre, les gestionnaires des ressources allouent uniquement une partie des ressources qui seront requises pour le traitement de paquet d'où plus de ressources de disponible. Toutefois, on constate que *P2AH* est plus efficace sous de fortes perturbations, mais affiche globalement le même taux d'optimisation que celui assuré par *NAH*. En effet, le gestionnaire d'allocation (*Allocator*) détermine approximativement les quantités de ressources nécessaires pour satisfaire les besoins des instances d'accélération de traitement

de paquet. Cette optimisation peut aller de 27% à 47% dépendamment des perturbations. *NAH*, quant à lui, se base sur des *bundles* de ressources prédéterminés ce qui limite l'optimisation des ressources. Ainsi les mécanismes pour moduler les ressources en fonction des besoins d'accélération du traitement de paquet jouent un rôle très important pour l'utilisation efficace des ressources disponibles.

Cependant, si on observe le taux de pertes sous des perturbations faibles (10%) et élevées (30%) en fonction du nombre des *slices* (voir figure 4.7), on remarque que les gestionnaires des ressources n'arrivent plus à maintenir le taux de pertes sous le seuil minimal toléré lorsque les perturbations sont fortes et que le nombre de *slices* est supérieur à 2. Pour des perturbations élevées (30%) le taux de pertes dépasse rapidement le seuil toléré des 10%. Pour des perturbations faibles et un nombre de *slices* inférieur ou égal à 4, le taux de pertes ne dépasse jamais le seuil. Au-delà de 4 *slices*, *P2AH* présente un taux de perte de 0.2%, supérieur à celui de *NAH*. Toutefois, *NAH* est plus instable sous des perturbations fortes.



(a) Perturbations faibles (10%)

(b) Perturbations fortes (30%)

Figure 4.7: Taux de perte de paquets

Le taux de perte élevé observé pour *P2AH* est causé par le mécanisme de redimensionnement, implémenté par l'*Allocator*. En effet durant le redimensionnement des instances, le traitement de paquet s'interrompt pendant un bref instant pour dé-

dier et isoler les ressources des instances d'accélération. Plus le débit est grand plus la perte de paquet s'accentue, d'où un taux de pertes qui peut dépasser le double de ce qui est toléré. En effet, lorsque le redimensionnement des instances nécessite un certain délai, tous les paquets qui arrivent durant l'opération de réallocation seront perdus. Ainsi, le redimensionnement des instances à la volée introduit par *P2AH* n'est plus performant pour des perturbations importantes et un nombre de *slices* supérieur à 3.

En outre, si on évalue l'impact de la gestion d'allocation des ressources sur le traitement de paquet, en moyenne pour toutes les instances (voir figure 4.8), on constate *P2AH* affecte moins le traitement de paquet par rapport à *NAH*. La détérioration du traitement de paquet est mesurée en fonction du taux de pertes par rapport à toute la durée des tests.

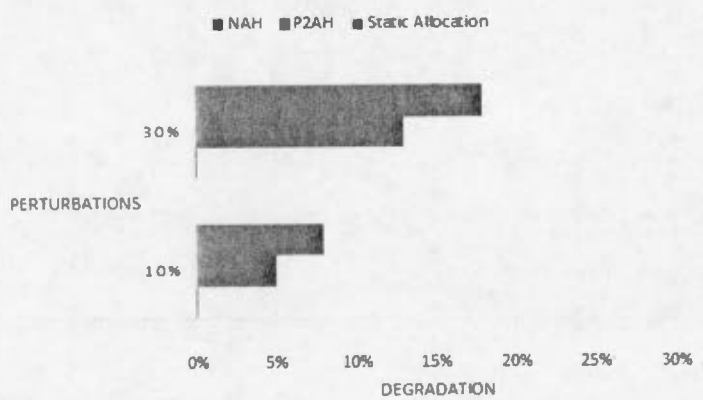


Figure 4.8: Dégradation du traitement de paquet

On constate que *P2AH* affecte le traitement de paquet de 13% comparé à *NAH*, qui lui, provoque une dégradation dépassant les 18% sous de fortes perturbations. En effet, *NAH* réalloue les ressources périodiquement suivant les intervalles de contrôle (voir section 1.3). Cette réallocation fréquente provoque une perturbation

importante du traitement de paquet. De son côté, *P2AH* ne réalloue les ressources que lorsque c'est nécessaire, c'est à dire lors d'un dépassement de seuil. Toutefois, la dégradation du traitement de paquet provoquée par les deux gestionnaires de ressources est jugée très faible.

En conclusion, nous considérons que la gestion dynamique des ressources introduite par *P2AH* tout comme *NAH*, garantit une efficacité en termes d'utilisation des ressources en réservant les ressources requises et en libérant le reste, et perturbe faiblement le traitement de paquet.

4.2.3 Coût de la virtualisation

En fonction des perturbations, nous mesurons la fréquence d'exécution des deux gestionnaires de ressources, *P2AH* et *NAH* (voir figure 4.9). On remarque que *P2AH* s'exécute moins souvent que *NAH*. En effet, *NAH* se base sur des intervalles de contrôle, ce qui fait qu'il s'exécute régulièrement. *NAH* s'exécute aux alentours de 100 fois indépendamment des perturbations. Grâce au mécanisme de fonctionnement de *P2AH* qui se base sur des seuils de contrôle, nous avons réduit le nombre d'exécutions du gestionnaire des ressources de plus de 40% pour de faibles perturbations. Sous de fortes perturbations, on remarque que *P2AH* présente approximativement la même fréquence d'allocation que *NAH*.

Le gain au niveau de la fréquence s'explique par la convergence de l'allocation vers la solution optimale. L'allocation calculée se rapproche de l'allocation optimale qui satisfait le plus possible les exigences du trafic et les contraintes liées aux ressources. La croissance de la fréquence s'explique par le fait que sous d'importantes perturbations, il est très difficile de trouver le point de convergence d'où une allocation régulière afin d'adapter le système aux changements rapides des exigences et des contraintes.

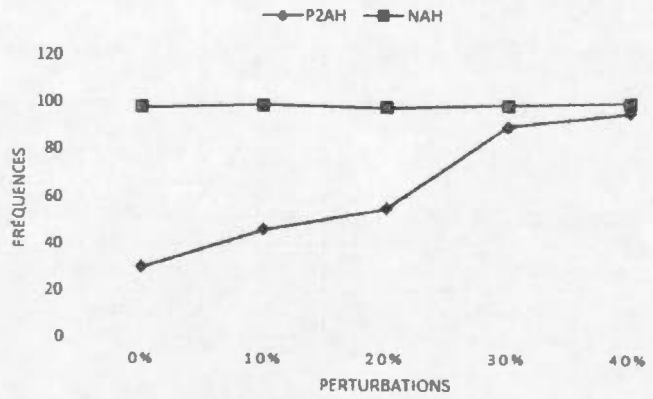


Figure 4.9: Fréquence de gestion des ressources

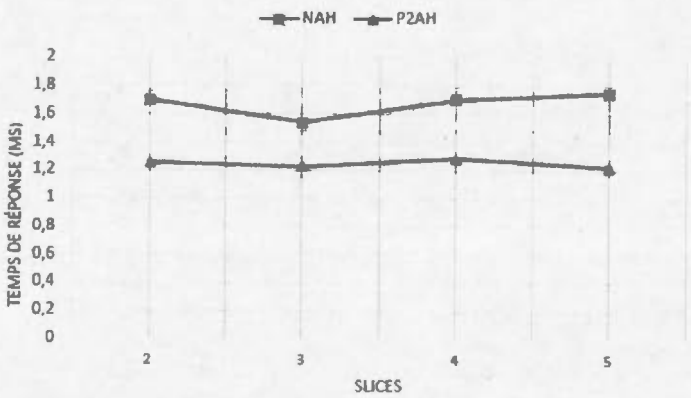


Figure 4.10: Temps de réponse

Par ailleurs, on évalue le temps de réponse correspondant aux gestionnaires des ressources en fonction du nombre de *slices* réseau (voir figure 4.10). On constate que le temps de réponse de *P2AH* est nettement inférieur à celui de *NAH*. *P2AH* présente un temps de réponse relativement stable aux alentours de 1,2ms. Par contre le temps de réponse de *NAH* varie entre 1,5ms et 1,8ms. Cet écart se traduit par les architectures différentes utilisées par les deux gestionnaires des res-

sources. Contrairement à *NAH*, *P2AH* exécute au plus deux niveaux de contrôle, l'*Allocator* et le *Slicer*. *NAH* exécute au pire des cas trois niveaux de contrôles ce qui engendre un plus grand délai de calcul, cela sans compter les boucles qui peuvent se créer à travers les signaux de contrôle. Ainsi, *P2AH* assure une gestion performante des ressources tout en minimisant les coûts liés aux contrôles des ressources en termes de calcul et d'exécution des allocations.

Conclusion

À travers ce chapitre nous avons pu évaluer *P2AH* et étudier ses points forts et ses points faibles. Durant les évaluations nous avons comparé *P2AH* à *NAH* (une solution existante) ainsi qu'à une approche d'allocation statique. Pour l'évaluation, nous avons considéré trois critères spécifiques : (1) la modélisation du traitement de paquet, (2) l'optimisation des ressources et (3) le coût lié à la virtualisation. *P2AH* modélise le traitement de paquet en fonction de la nature des tâches ainsi que les ressources qui sont sollicitées. Il s'est avéré que les accélérations de traitement de paquet qui sollicite des tâches complexes comme la recherche LPM et la classification 5-uplet, nécessitent plus de ressources. Par ailleurs, la gestion dynamique des ressources introduite par *P2AH* tout comme *NAH*, garantit une efficacité en termes d'utilisation des ressources en réservant uniquement les ressources nécessaires et perturbe faiblement le traitement de paquet. Toutefois, la gestion dynamique des ressources peut causer une perte de paquets qui dépasse souvent les seuils tolérés. Enfin, *P2AH* assure une gestion efficace des ressources en minimisant les coûts liés à l'allocation et au partage des ressources.

CONCLUSION

Le réseau d'aujourd'hui connait une révolution importante, que ce soit au niveau de l'infrastructure matérielle qui déploie des technologies de plus en plus sophistiquées, ou au niveau logiciel où on voit apparaître de nouveaux paradigmes controversés. En effet, la virtualisation des fonctions réseau séduit énormément les opérateurs et fournisseurs de services pour les bénéfices que cela peut engendrer. Cependant, l'infrastructure matérielle est de nouveau mise à l'épreuve. Des solutions innovantes et des techniques performantes sont fortement recommandées, particulièrement, dans un environnement où les accélérations pour le traitement de paquet sont exigibles et les contraintes liées aux ressources sont strictes.

Dans ce mémoire, nous avons jugé qu'il est intéressant d'exploiter une architecture matérielle flexible pour ajuster le traitement de paquet aux exigences du trafic et aux contraintes des ressources. Le travail présenté porte sur la mise en place d'un hyperviseur pour une architecture à processeur multi-cœurs. L'objectif du travail est de déterminer des patrons d'accélération de traitement de paquet en utilisant les ressources modulables et programmables du processeur multi-cœurs virtualisé. Tout au long des chapitres, nous avons décrit notre proposition, argumenté nos choix, évalué notre modèle et enfin analyser nos résultats. Nous avons comparé la solution proposée aux solutions existantes pour mieux identifier les points forts et les points faibles de notre conception.

Les évaluations réalisées témoignent du potentiel de la solution proposée. Grâce à un mécanisme avancé d'analyse et d'allocation des ressources matérielles, nous avons constaté que *P2AH* associe les tâches aux différentes ressources de manière

à respecter les exigences du traitement de paquet. En effet, pour un besoin en accélération relativement moyen, les ressources disponibles sont distribuées entre les instances en fonction du coût de traitement des paquets. Le traitement de paquet qui sollicite des tâches de classification complexes nécessite plus de ressources, 33-38% de l'allocation des ressources. Cependant, les accélérations ne sont plus satisfaisantes sous des perturbations dépassant les 20-30% et nécessitent des optimisations au niveau des algorithmes de classification et l'utilisation, si possible, des accélérateurs matériels pour respecter les contraintes du traitement de paquet.

Par ailleurs, il est important de relever que l'un des avantages de *P2AH* se décrit à travers l'optimisation des ressources. En modélisant les ressources pour les instances de traitement de paquet, *P2AH* alloue parfaitement l'ensemble des ressources nécessaires pour satisfaire les besoins en accélération et pour éviter le gaspillage des ressources disponibles aux dépens des autres instances concurrentes. Les résultats ont prouvé que l'optimisation de l'utilisation des ressources peut aller au-delà des 45%. Nous avons aussi constaté que *P2AH* assure un partage de ressources équitable. Quelles que soient les perturbations, *P2AH* maximise le profit de chacune des *slices* réseau qui s'appliquent aux instances de traitement. Contrairement aux propositions existantes, P2AH réduit considérablement le coût engendré par la virtualisation en minimisant la dégradation du traitement de paquet d'environ 20%. En effet, les expérimentations ainsi que les résultats démontrent que *P2AH* converge rapidement vers la stratégie de partage de ressources la plus satisfaisante de point de vue du besoin en accélération et garantit un temps de réponse nettement plus rapide.

Cependant, nous avons constaté que le mécanisme d'allocation dynamique peut éventuellement causer des perturbations au niveau du traitement de paquet qui parfois, provoquent le dépassement des seuils tolérés pour la stabilité du système. Ces dépassements varient en fonction des perturbations et du nombre de *slices*.

Ces dépassements atteignent éventuellement les 20%. Cela s'explique par le mécanisme de re-modélisation et de re-dimensionnement des instances qui nécessite l'interruption temporaire du traitement. Néanmoins, ce mécanisme a démontré sa valeur dans la détermination des ressources optimales pour satisfaire les besoins en accélération du traitement de paquet. Ainsi, à travers les résultats obtenus, nous avons prouvé que notre proposition vérifie les principaux critères suivants : (1) la modélisation du traitement de paquet, (2) l'optimisation de l'utilisation des ressources, (3) le partage équitable des ressources et (4) la minimisation du coût de la virtualisation.

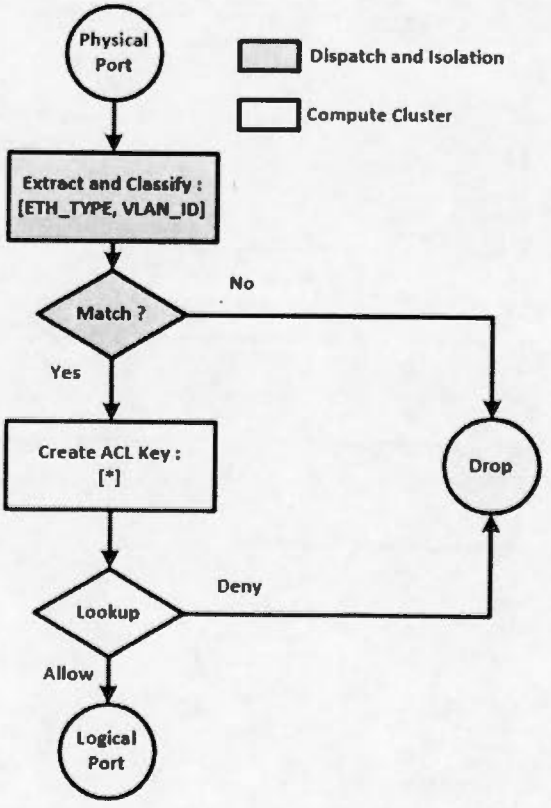
Bien que nous jugeons que notre proposition est intéressante dans le cadre de la virtualisation, on observe un certain scepticisme dans la communauté à l'égard de l'allocation dynamique. En effet, certains stipulent que le besoin d'allouer les ressources dynamiquement n'est pas assez justifié et qu'avec un bon dimensionnement il est possible de résoudre les problèmes. Toutefois, force est de constater que le réseau est en perpétuelle extension et que celui-ci nécessitera plus d'automatisation, d'auto-gestion et de flexibilité pour suivre son évolution.

Nous estimons ainsi que l'approche employée est prometteuse, toutefois, nous laissons le champ libre à davantage d'améliorations autant au niveau de l'implémentation qu'au niveau conceptuel. Par exemple, nous pouvons évaluer la modélisation du traitement de paquet à travers l'utilisation d'une plateforme plus variée, qui incorpore des accélérateurs matériels pour supporter différentes tâches de traitement de paquet et ainsi manipuler un ensemble de ressources hétérogènes. Aussi, nous pouvons penser à prendre en compte plusieurs métriques de performances relatives aux accélérations de traitement de paquets offertes par les instances afin d'identifier des modèles de traitement de paquet plus spécifiques sous diverses contraintes et exigences.

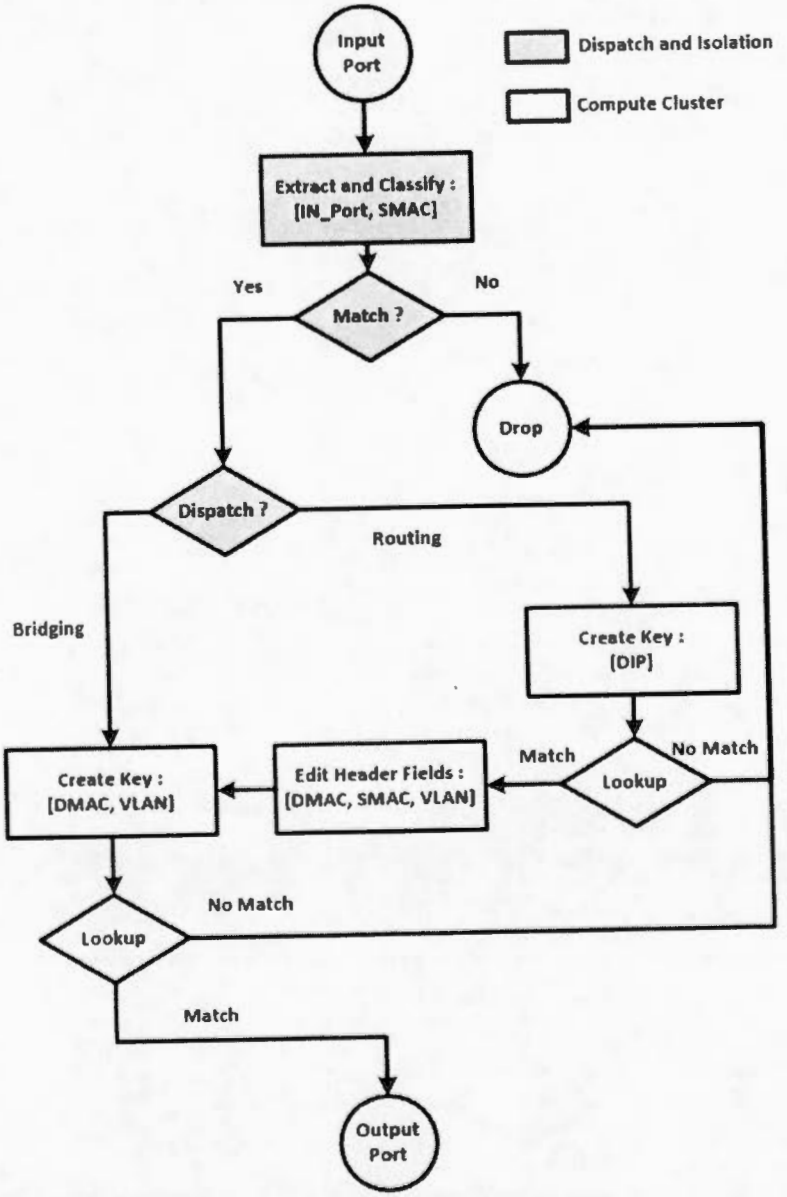
ANNEXE A

CHEMINS DE TRAITEMENT

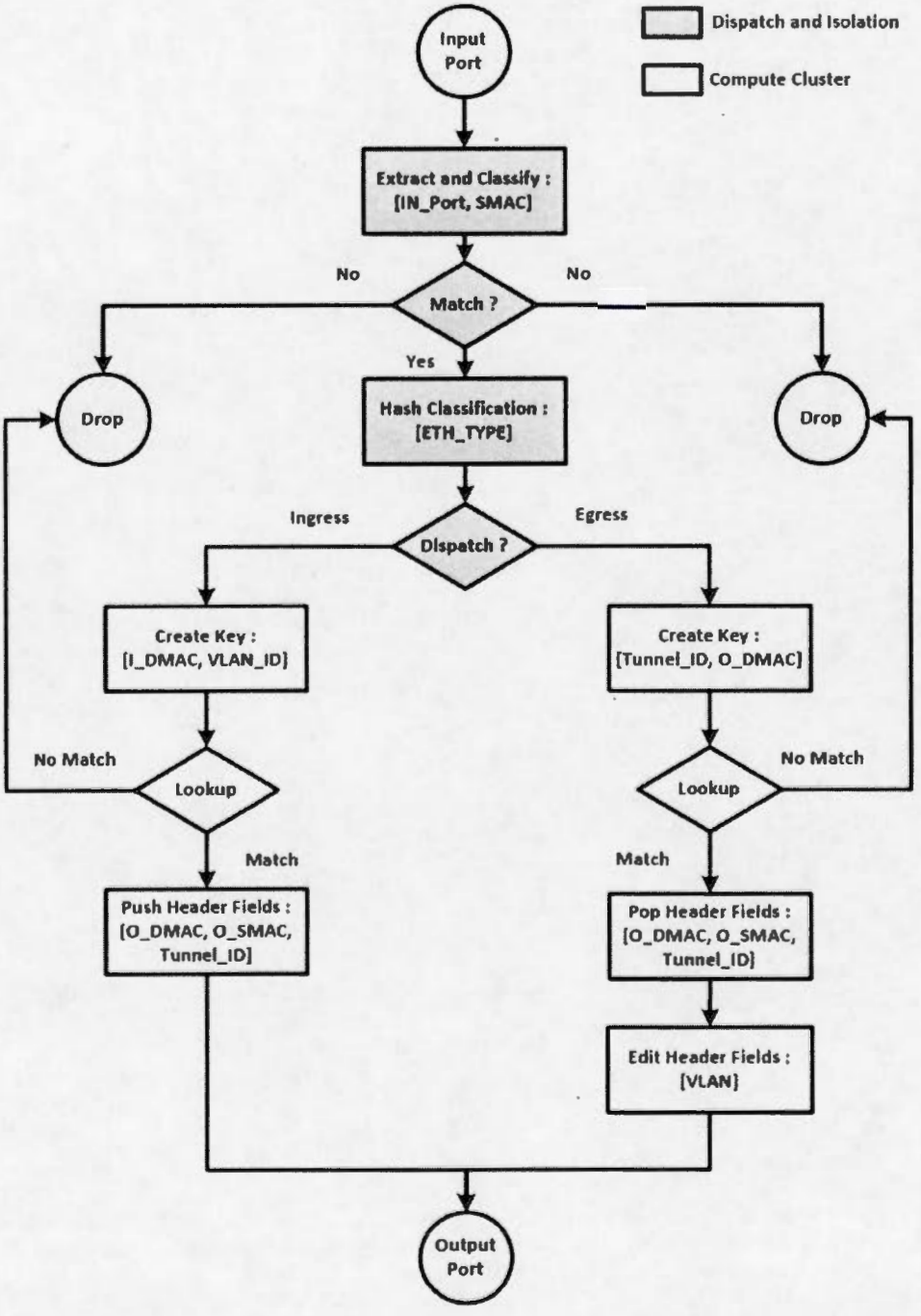
FILTRAGE



ROUTAGE


Input Port
Dispatch and Isolation
Compute Cluster
Extract and Classify : [IN_Port, SMAC]
Yes
Match ?
No
Drop
Dispatch ?
Routing
Bridging
Create Key : [DIP]
Match
No Match
Lookup
Edit Header Fields : [DMAC, SMAC, VLAN]
Create Key : [DMAC, VLAN]
No Match
Lookup
Match
Output Port

TUNNEL


Dispatch and Isolation
Compute Cluster
Input Port
Extract and Classify : [IN_Port, SMAC]
No
Match ?
No
Yes
Drop
Hash Classification : [ETH_TYPE]
Drop
Ingress
Dispatch ?
Egress
Create Key : [I_DMAC, VLAN_ID]
Create Key : [Tunnel_ID, O_DMAC]
No Match
Lookup
No Match
Lookup
Match
Match
Push Header Fields : [O_DMAC, O_SMAC, Tunnel_ID]
Pop Header Fields : [O_DMAC, O_SMAC, Tunnel_ID]
Edit Header Fields : [VLAN]
Output Port

ANNEXE B

STRUCTURES DE RECHERCHE

Instance	Table	Type	Clé	Méta-données
Routage	Bridging Flow Table	Hash	Vlan Id DMAC	-
	Routing Flow Table	LPM	DIP	SMAC DMAC Vlan Tag
Filtrage	ACL Policy Flow Table	5-Tuples Classifier	SIP DIP Protocol DP SP	-
Tunnel	Ingress Bridging Flow Table	Hash	Vlan Id Ethertype Inner SMAC Inner DMAC	Outer DMAC Outer SMAC Tunnel Id
	Egress Bridging Flow Table	Hash	Tunnel Id Ethertype Outer SMAC Outer DMAC	Vlan Tag

RÉFÉRENCES

(2010). *NFP-3200 Network Flow Processor*. Rapport technique, Netronome Systems Inc.

(2014). *Openflow Data Plane Abstraction (OF-DPA) : Abstract Switch Specification*. Rapport technique, Broadcom Corp.

(2015). *NP-4 100Gbps NPU for Carrier Ethernet Applications*. Rapport technique, EZchip Semiconductor Inc.

Baccelli, F., Cohen, G., Olsder, G. J. et Quadrat, J.-P. (1992). *Synchronization and linearity : an algebra for discrete event systems*. John Wiley & Sons Ltd.

Blaiech, K. et Cherkaoui, O. (2015). Virtual fabric-based approach for virtual data center network. Dans *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, 31–38. ACM.

Blaiech, K., Mounaouar, O., Cherkaoui, O. et Beliveau, L. (2014). Runtime resource allocation model over network processors. Dans *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, 556–561. IEEE.

Chao, H. J. et Liu, B. (2007). *High performance switches and routers*. John Wiley & Sons.

Chowdhury, N. M. K. et Boutaba, R. (2010). A survey of network virtualization. *Computer Networks*, *54*(5), 862–876.

Dong, Y., Yang, X., Li, J., Liao, G., Tian, K. et Guan, H. (2012). High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing*, *72*(11), 1471–1480.

Feldman, M., Lai, K. et Zhang, L. (2005). A price-anticipating resource allocation mechanism for distributed shared clusters. Dans *Proceedings of the 6th ACM Conference on Electronic Commerce*, EC '05, 127–136. ACM.

Gupta, P. et McKeown, N. (1999). Packet classification using hierarchical intelligent cuttings. Dans *Hot Interconnects VII*, 34–41.

Hu, Y., Long, X., Zhang, J., He, J. et Xia, L. (2010). I/o scheduling model of virtual machine based on multi-core dynamic partitioning. Dans *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, 142–154. ACM.

Kokku, R., Riché, T. L., Kunze, A., Mudigonda, J., Jason, J. et Vin, H. M. (2004). A case for run-time adaptation in packet processing systems. *ACM SIGCOMM Computer Communication Review*, *34*(1), 107–112.

Koutsopoulos, I. et Iosifidis, G. (2010). Auction mechanisms for network resource allocation. Dans *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010 Proceedings of the 8th International Symposium on*, 554–563. IEEE.

Kreutz, D., Ramos, F. M., Esteves Verissimo, P., Esteve Rothenberg, C., Azodolmolky, S. et Uhlig, S. (2015). Software-defined networking : A comprehensive survey. *Proceedings of the IEEE*, *103*(1), 14–76.

Le Boudec, J.-Y. et Thiran, P. (2001). *Network calculus : a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media.

León, X. et Navarro, L. (2011). Limits of energy saving for the allocation of data center resources to networked applications. Dans *INFOCOM, 2011 Proceedings IEEE*, 216–220. IEEE.

Marbach, P. (2002). Priority service and max-min fairness. Dans *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, 266–275. IEEE.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. et Boutaba, R. (2015). Network function virtualization : State-of-the-art and research challenges.

Pfaff, B., Pettit, J., Koponen, T., Jackson, E., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P. *et al.* (2015). The design and implementation of open vswitch. Dans *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 117–130.

Qi, Y., Xu, B., He, F., Yang, B., Yu, J. et Li, J. (2007). Towards high-performance flow-level packet processing on multi-core network processors. Dans *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, 17–26. ACM.

Qu, Y. R., Zhang, H. H., Zhou, S. et Prasanna, V. K. (2015). Optimizing many-

field packet classification on fpga, multi-core general purpose processor, and gpu. Dans *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems*, 87–98. IEEE Computer Society.

Qu, Y. R., Zhou, S. et Prasanna, V. K. (2014). Performance modeling and optimizations for decomposition-based large-scale packet classification on multi-core processors. Dans *High Performance Switching and Routing (HPSR), 2014 IEEE 15th International Conference on*, 154–161. IEEE.

Radunović, B. et Le Boudec, J. (2007). A unified framework for max-min and min-max fairness with applications. *IEEE/ACM Transactions on Networking*, *15*(5), 1073–1083.

Rizzo, L. (2012). Netmap : a novel framework for fast packet i/o. Dans *21st USENIX Security Symposium (USENIX Security 12)*, 101–112.

Rizzo, L., Carbone, M. et Catalli, G. (2012). Transparent acceleration of software packet forwarding using netmap. Dans *INFOCOM, 2012 Proceedings IEEE*, 2471–2479. IEEE.

Schmitt, J. B. et Zdarsky, F. A. (2006). The disco network calculator : a toolbox for worst case analysis. Dans *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, p. 8. ACM.

Snaiki, I. (2014). Allocation des ressources dans un routeur virtualisé.

Thiele, L., Chakraborty, S., Gries, M. et Künzli, S. (2002). Design space exploration of network processor architectures. *Network Processor Design : Issues and Practices*, *1*, 55–89.

Wells, P. M., Chakraborty, K. et Sohi, G. S. (2009). Dynamic heterogeneity and the need for multicore virtualization. *ACM SIGOPS Operating Systems Review*, *43*(2), 5–14.

Wu, Q. et Wolf, T. (2012). Runtime task allocation in multicore packet processing systems. *Parallel and Distributed Systems, IEEE Transactions on*, *23*(10), 1934–1943.