Hardware and software co-design toward flexible terabits per second traffic processing

Franck Cornevaux-Juignet

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Électronique

Par
Franck CORNEVAUX-JUIGNET

Hardware and software co-design toward flexible terabits per second traffic processing

Thèse présentée et soutenue à l'IMT Atlantique, site de Brest, le 4 juillet 2018
Unité de recherche : Lab-STICC CACS
Thèse N° : 2018IMTA0081

Rapporteurs avant soutenance :

Yvon SAVARIA	Professeur, Polytechnique Montréal
Christophe JÉGOT	Professeur, Bordeaux INP

Composition du Jury :

Président :	Frédéric ROUSSEAU	Professeur, Université Grenoble Alpes
Examinateurs :	Yvon SAVARIA	Professeur, Polytechnique Montréal
	Christophe JÉGOT	Professeur, Bordeaux INP
	Christine HENNEBERT	Docteure Ingénieure de recherche, CEA
	Matthieu ARZEL	Maître de Conférences, IMT Atlantique
	Pierre-Henri HORREIN	Docteur Ingénieur de recherche, OVH
Dir. de thèse :	Christian PERSON	Professeur, IMT Atlantique

Invité

Tristan GROLÉAT	Docteur Ingénieur de recherche, OVH

Remerciements

Les travaux de recherche présentés dans ce manuscrit sont le fruit d'un travail de plus de 3 ans. J'aimerais en introduction remercier l'ensemble des personnes qui ont permis l'achèvement de ces travaux.

Je souhaite premièrement remercier les membres extérieurs de mon jury de thèse pour leur déplacement : Madame Christine Hennebert ainsi que Messieurs Yvon Savaria, Christophe Jégo et Frédéric Rousseau. Merci d'avoir fait le déplacement, j'ai apprécié échanger avec vous pendant et après la soutenance.

Ce long travail n'aurait pas pu être mené à son terme sans le soutien de mon équipe encradrante. Je remercie mon directeur de thèse, Christian Person, dont la bienveillance a permis d'éviter un arrêt prématuré de ma thèse. Je tiens à remercier tout particulièrement Matthieu Arzel et Pierre-Henri Horrein, mes encadrants, pour leur support et leurs conseils avisés pendant toute la durée de la thèse. J'ai grandement apprécié travailler avec vous, et j'espère que nous pourrons collaborer dans le futur.

Je remercie ma famille qui m'a soutenu tout au long de ces années. Je souhaite dire un grand merci à mes amis et mes collègues thésards pour votre soutien, et en particulier à Paul, Erwan, Benoît, Valentin, Pierre, Paul et André. Je remercie aussi toutes les personnes venues aux séances de rugby à toucher le jeudi midi avec lesquelles j'ai partagé des moments très conviviaux.

Je tiens enfin à remercier l'ensemble du personnel du département ELEC de l'IMT Atlantique pour leur accueil. Vous m'avez permis de travailler dans une très bonne ambiance qui a sans nul doute contribué à l'achèvement de mes travaux.

Résumé

La fiabilité et la sécurité des réseaux de communication nécessitent des composants efficaces pour analyser finement le trafic de données. La diversification des services ainsi que l'augmentation des débits obligent les systèmes d'analyse à être plus performants pour gérer des débits de plusieurs centaines, voire milliers de Gigabits par seconde. Les solutions logicielles communément utilisées offrent une flexibilité et une accessibilité bienvenues pour les opérateurs du réseau mais ne suffisent plus pour répondre à ces fortes contraintes dans de nombreux cas critiques.

Cette thèse étudie des solutions architecturales reposant sur des puces programmables de type Field-Programmable Gate Array (FPGA) qui allient puissance de calcul et flexibilité de traitement. Des cartes équipées de telles puces sont intégrées dans un flot de traitement commun logiciel/matériel afin de compenser les lacunes de chaque élément. Les composants du réseau développés avec cette approche innovante garantissent un traitement exhaustif des paquets circulant sur les liens physiques tout en conservant la flexibilité des solutions logicielles conventionnelles, ce qui est unique dans l'état de l'art.

Cette approche est validée par la conception et l'implémentation d'une architecture de traitement de paquets flexible sur FPGA. Celle-ci peut traiter n'importe quel type de paquet au coût d'un faible surplus de consommation de ressources. Elle est de plus complètement paramétrable à partir du logiciel. La solution proposée permet ainsi un usage transparent de la puissance d'un accélérateur matériel par un ingénieur réseau sans nécessiter de compétence préalable en conception de circuits numériques.

Mots-clés : Surveillance de trafic, FPGA, architecture hétérogène, traitements haute performance, co-conception logicielle/matérielle

Abstract

The reliability and the security of communication networks require efficient components to finely analyze the traffic of data. Service diversification and throughput increase force network operators to constantly improve analysis systems in order to handle throughputs of hundreds, even thousands of Gigabits per second. Commonly used solutions are software oriented solutions that offer a flexibility and an accessibility welcome for network operators, but they can no more answer these strong constraints in many critical cases.

This thesis studies architectural solutions based on programmable chips like Field-Programmable Gate Arrays (FPGAs) combining computation power and processing flexibility. Boards equipped with such chips are integrated into a common software/hardware processing flow in order to balance shortcomings of each element. Network components developed with this innovative approach ensure an exhaustive processing of packets transmitted on physical links while keeping the flexibility of usual software solutions, which was never encountered in the previous state of the art.

This approach is validated by the design and the implementation of a flexible packet processing architecture on FPGA. It is able to process any packet type at the cost of slight resources overconsumption. It is moreover fully customizable from the software part. With the proposed solution, network engineers can transparently use the processing power of an hardware accelerator without the need of prior knowledge in digital circuit design.

Keywords: Traffic monitoring, FPGA, heterogeneous architecture, high performance computing, hardware/software co-design

Résumé étendu

1 Introduction

Les réseaux de communication informatiques sont la clé de voute de la société actuelle, axée sur le développement du numérique. L'infrastructure réseau est l'épine dorsale permettant le bon fonctionnement des activités modernes les plus prospères. Le commerce en ligne, les réseaux sociaux, la vidéo à la demande sont des examples parmi les nombreux services existants grâce à un réseau performant. Cette importance donne un status critique aux réseaux, nécessitant une résilience aux pannes et aux comportements malveillants. Afin maintenir la fiabilité et la sécurité des communications, les opérateurs de réseau contrôlent le trafic en circulation. Il est possible d'extraire une vision de l'état du réseau afin de réagir de façon efficace aux problèmes. Pour avoir un suivi le plus précise possible, les systèmes de surveillance doivent être capable de suivre le trafic au débit des liens.

En réponse à l'évolution de la demande, les sondes réseaux sont confrontées à la constante agumentation des débits des liens. Les utilisateurs disposent de connexions de plus en plus rapides afin de profiter de l'ensemble des services proposés. Selon les index de Cisco, Cisco Visual Networking Index (VNI) [Cisa], 46% de la population mondiale était connectée en 2016, avec un débit de connexion moyen de 27.5 Mbps. Le trafic IP mondial était de 1.2 Zettaoctets (10^{21} octets) pour l'année 2016, soit 3.2 Exaoctets (10^{18} octets) de trafic par jour, et un trafic moyen de 292 Tbps. Le réseau va continuer à être de plus en plus utilisé pour atteindre en 2021, selon les prévisions de Cisco, un trafic annuel de 3.3 Zettaoctets, soit un trafic moyen de 874 Tbps. Il est nécessaire de pouvoir surveiller minutieusement une telle bande passante. En effet, le botnet Mirai [OVHb] a montré en septembre 2016 qu'il était possible d'utiliser cette bande passante afin d'envoyer une quantité massive de trafic. 1 Tbps de trafic Transmission Control Protocol (TCP) malicieux envoyé à partir d'objets connectés infectés a ciblé le fournisseur de service web OVH. Sur un lien à très haut débit, même un petit flot de données ou un petit laps de temps représente une quantité de données non négligeable et potentiellement dommageable. 1% d'un trafic de 1 Tbps représente tout de même 10 Gbps. Les acteurs du réseau doivent pouvoir avoir à leur disposition des sondes précises et réactives afin d'éviter tout impact potentiellement considérable sur l'infrastructure.

En plus de l'augmentation des débits, les équipements réseaux doivent faire face à un trafic disparate dont la composition change constamment. Ce trafic est composé d'un ensemble de paquets issus d'une multitude de services différents qui ne sont pas utilisés de façon uniforme, que ce soit temporellement ou spatialement. Cette disparité peut créer des pics de trafic correspondant à des pics d'utilisation de certains

services. En 2016, les heures de haute fréquentation généraient un pic de 1 Pbps (10^{15} bits) [Cisa]. Les opérateurs réseaux doivent dimensionner leurs infrastructures pour continuer un bon fonctionnement même pendant ces heures de pointe. De plus, dans ce trafic de composition très variée, les paquets correspondants à chaque service ont des besoins de transmission spécifiques. Appels vidéos, streaming, navigation internet ou jeux en lignes sont des exemples de différents besoins en termes de débit, latence or gigue. Cette coexistence de différents types de trafic nécessite d'avoir des outils qui peuvent supporter les modifications de la forme du trafic tout en respectant les différents usages du réseau. Pour supporter cette diversité et de possibles futures évolutions, les infrastructures sont actuellement construites avec des systèmes de surveillances compatible avec un paradigme, Software Defined Networking (SDN) [Zil+15]. Ce paradigme, recommandant la séparation du plan de données et du plan de contrôle, permet d'avoir un réseau programmable. Il est alors possible de gérer globalement le réseau et de calibrer dynamiquement l'architecture au contenu du trafic.

La performance et la sécurité d'un réseau dépendent de la réponse des systèmes surveillance à ces nombreuses contraintes. Les sondes réseaux, briques de base réparties dans l'infrastructure pour collecter le trafic, doivent alors répondre à certains critères :

- fiabilité,
- très haut débit,
- flexibilité,
- passage à l'échelle,
- réactivité.

Réaliser une sonde combinant ces différents critères est un défi clé pour concevoir les réseaux de demain.

Cette thèse étudie des solutions architecturales permettant de lever les différentes contraintes s'appliquant sur des sondes réseaux. Après une étude approfondie de la littérature, nous allons A partir de la littérature, il est possible de déterminer que les solutions reposant sur des puces programmables de type Field-Programmable Gate Array (FPGA) sont les plus à Elles allient puissance de calcul et flexibilité de traitement. En intégrant des cartes équipées de telles puces dans un flot de traitement commun logiciel/matériel, il est possible de compenser les lacunes de chaque élément. Cependant, les flots de développement FPGA courants limite la flexibilité des solutions. C'est pourquoi une architecture matérielle conforme à cette approche est proposée. Cette approche sera validée avec la conception d'une application de classification de paquets rapide, fiable et flexible.

2 Systèmes de surveillance de trafic

Savoir gérer le trafic est un élément crucial pour la bonne gestion d'un réseau. Quelle que soit l'application, il est nécessaire de récupérer des informations à partir des paquets en circulation. Pour cela, de nombreuses plate-formes ont été conçues et dévelop-

pées pour la manipulation du trafic. Ce large spectre de solutions offre plusieurs compromis entre performance et flexibilité. Afin de supporter des sondes adaptées aux contraintes de réseaux à très haut débit, une plate-forme de développement doit répondre à certaines critères :

- analyser précisemment le trafic à très haut débit,
- offrir de la flexibilité pour :
  - être conforme aux besoins du SDN,
  - être continuellement adaptable au trafic,
- permettre une adaptation des traitements avec une faible latence comparé à l'échelle du temps du lien,
- être facilement réutilisée pour des plus gros réseaux avec des débits plus importants,
- être accessible à n'importe quel ingénieur réseau sans connaissance préalable du matériel,
- être portable pour être intégrée dans différents systèmes de surveillance de trafic.

Lors de la sélection d'une plate-forme, il est nécessaire de prendre en compte que les contraintes de traitement du trafic viennent majoritairement du traitement des paquets composant le trafic. En effet, ils ont une structure variable qui dépend des besoins des services émetteurs. Leur taille est de plus variable, allant de 64 octets à 1522 octets pour les liens Ethernet considérés. Cela signifie qu'à un débit fixé, il y aura plus de paquets à traiter si les paquets transmis sont de taille minimale que de taille maximale. Par exemple, un lien à 40 Gbps correpond à 59,523 millions de paquets de 64 octets par seconde contre 3,251 millions de paquets de 1522 octets par seconde. Comme le trafic en circulation n'est pas maîtrisé, le pire cas est à prendre en compte.

Dans les plate-formes existantes, la plus courante est une plate-forme basée sur du matériel du commerce. Elle est composée d'une ou plusieurs cartes réseaux, un processeur généraliste (Central Processing Unit (CPU)) et potentiellement un processeur graphique (Graphics Processing Unit (GPU)) pour accélérer les traitements. Un tel type de plate-forme sur étagère a un coût relativement faible, la rendant aisément accessible. Elle est largement répandue, et la généricité du CPU offre une facilité d'utilisation et une grande portabilité. Cela permet d'avoir une grande banque d'outils disponibles pour le traitement des paquets. Ces solutions sont toutefois limitées par leur généricité. Malgré l'optimisation des pilotes de communications avec les cartes réseaux, elles n'arrivent pas à suivre la montée en débit, même avec l'utilisation intensive de GPU. La puissance de calcul d'une seule plate-forme est trop faible pour tenir les débits actuels de 40 Gbps ou 100 Gbps, et encore moins les débits à venir. La seule possibilité pour passer à l'échelle est la construction de datacenters, ce qui rend la solution très onéreuse.

À l'opposé du matériel du commerce se trouvent les solutions à base de puces spécifiques à une application (Application-Specific Integrated Circuit (ASIC)). Grâce à une spécialisation des traitements et une grande possibilité de parallélisme, l'application

considérée peut atteindre des débits bien plus importants qu'avec un CPU. Cependant, un ASIC étant dédiée à une application, il est nécessaire d'en créer une nouvelle quand les besoins ou l'application changent. Le temps et le coût de développement de telles puces en font une solution peu adaptée pour les besoins de flexibilité d'une sonde réseau. Il est possible d'intégrer plusieurs ASICs à côté d'un processeur généraliste dans un élément appelé processeur réseau pour avoir plus de flexibilité. Cependant, cette solution groupe les inconvénients des deux précédentes solutions quand il s'agit de prendre en considération plusieurs applications.

Une solution à mi-chemin est l'utilisation de FPGAs. En s'appuyant sur la spécialisation des traitements et du parallélisme, de même que pour un ASIC, une application peut atteindre des débits lignes à très haut débit. Cependant, un FPGA offre une puissance de calcul moindre qu'un ASIC, mais cela est compensé par la possibilité de reconfigurer le FPGA. Il est possible de changer tout ou partie de l'application en fonctionnement sur le FPGA. L'accessibilité à cette flexibilité est assurée par des outils de développement haut niveau, bien que cela nuise à la portabilité et à la réactivité de la plate-forme. Avec une grande puissance de calcul et de la flexibilité de traitement, le FPGA est la plate-forme de choix pour le développement de sondes réseaux à très haut débit.

3 Éléments réseaux matériels/logiciels

L'adaptation du FPGA pour répondre aux contraintes réseaux a attisé l'intérêt pour le développement de solutions de traitement. Celui-ci est notamment montré par l'intégration de FPGAs dans des cartes réseaux intelligentes, smart Network Interface Card (NIC), à côté d'une carte réseau commune. Les smart NICs peuvent être insérées, comme leurs contre-parties sans FPGA, dans la chaîne de traitement du matériel du commerce. La puissance de calcul du FPGA, plus proche de l'interface réseau, offre la possibilité de décharger des traitements du CPU tout en réduisant la quantité de trafic transférée à l'hôte. Une collaboration entre le FPGA et le CPU assure la flexibilité des traitements tout en améliorant les capacités de traitement d'au moins un ordre de magnitude. L'intégration d'un smart NICs dans du matériel du commerce permet aussi de conserver l'accès à l'ensemble des outils réseaux existants.

Ce rapprochement est néanmoins limité par le flot de développement couramment utilisé pour les FPGAs. Outre l'utilisation d'outils propriétaires limitant la portabilité, il repose sur l'utilisation du FPGA comme accélérateur hautement spécialisé devant être changé pour chaque application. Bien que bien moins coûteuse que pour les ASICs, l'étape de génération d'un nouveau design a une durée qui se compte tout de même en heures, voire en jours pour les FPGAs les plus récents. De plus, la reconfiguration d'un FPGA entraîne la nécessité d'arrêter les traitements pendant la procédure. Même avec les améliorations apportées par la reconfiguration partielle, cette technique est restreinte à de rares mises à jour firmware. Il n'est alors pas possible de profiter pas de la proximité du FPGA avec le CPU pour raffiner en temps réel les traitements en fonction du trafic observé.

Néanmoins, avec la bonne architecture matérielle, cette approche est modifiable afin de tirer partie de la puissance de calcul du FPGA tout en ayant la possibilité de configurer les traitements en temps réel. En effet, baser les traitements matériels sur

des paramètres modifiables donne la possibilité de les changer à la volée. Le FPGA n'est plus uniquement configurable, il est aussi paramétrable. Le FPGA du smart NIC peut ainsi être intégré dans un flot de traitement logiciel flexible tout en conservant ses capacités de haute performance. Les opérateurs de réseaux ont alors accès à un matériel aussi accessible que le logiciel tout en maîtrisant complètement la chaîne de traitement. Ils possèdent la capacité d'avoir un équipement efficace, réactif et complètement digne de confiance dont le comportement est parfaitement connu.

Cependant, l'optimisation des performances d'une telle approche nécessite une séparation adéquate des applications réseaux entre le matériel et le logiciel. Tous les traitements ne profitent pas de manière égale de l'accélération apportée par le FPGA. L'augmentation des débits de traitement passe par la conservation au mieux des avantages des deux domaines logiciel et matériel dans le flot de manipulation des paquets. L'étude des points critiques est nécessaire pour avoir des sondes prenant en compte les spécificités du réseau. Pour cela, l'approche combinée logicielle et matérielle est utilisée sur deux applications.

Dans un premier temps, un générateur de trafic flexible illustre la nécessité de séparer les actions de décision et la gestion des paquets afin d'obtenir un traitement haute performance et flexible. Afin d'atteindre les débits maximums du lien, la gestion de l'envoi des paquets est laissée à la charge du FPGA. La sélection des paquets à envoyer, leur ordre et le débit sont pilotés par des paramètres de configuration envoyés du logiciel. Le programme de génération des paramètres permet à l'utilisateur d'avoir la main mise sur le trafic émis. Cela offre la possibilité de créer des motifs de trafic bien précis afin de tester et potentiellement mettre en défaut des équipements réseaux. L'architecture ne permet pas, pour l'instant, la création d'un trafic avec le contrôle absolu paquet par paquet. Un groupe de paquets peut être configuré et rejoué successivement jusqu'à ce qu'une nouvelle configuration soit envoyée par la partie logicielle. Grâce à cette architecture, il est tou de même possible de générer du trafic jusqu'à 160 Gbps sur la carte de test disponible, avec une composition du trafic qui évolue en fonction du temps d'une façon choisie par l'opérateur. Les débits atteints sont inaccessible pour des générateurs complètement logiciels. De plus, le contrôle instantané de la forme du trafic est une inovation par rapport aux autres solutions matérielles déjà existantes. Cette application, émettrice de trafic, est néanmoins privilégiée par la connaissance *a priori* des paquets à transmettre.

La deuxième application se concentre sur la réception des paquets par une sonde. En étudiant le traitement d'un paquet par le logiciel, il est possible de déterminer plusieurs étapes :

- l'analyse du paquet lors de sa réception,
- sa classification avec un ensemble de règles pour le filtrage et la distribution du paquet,
- l'utilisation du paquet par l'application finale.

Les deux premières étapes sont communes à toutes les applications fonctionnant sur le logiciel, elles doivent être exécutées à la vitesse du lien et profitent du parallélisme du FPGA. Les applications finales, au contraire, réalisent des décisions sur les paquets et

ne profitent pas toutes de l'accélération du FPGA. Leur accélération est moins critique et peut être décidée au coup par coup.

En réception, une sonde souffre de la non connaissance du trafic en circulation. L'étude du logiciel montre que les applications décident des paquets qui leur sont délivrées. Afin de reproduire ce phénomène, il est possible de créer une boucle de rétro-action entre le logiciel et le matériel. Les applications, en fonction des résultats des traitements, prennent la décision de changer la configuration du traitements effectués sur le matériel. Le pré-traitement de chaque paquet doit pour cela avoir une architecture matérielle configurable.

4 Architecture d'analyse de paquets novatrice pour de la surveillance de trafic adaptative

Les étapes de traitement des paquets à considérer pour l'architecture matérielle sont avant tout l'analyse des paquets et leur filtrage. L'étape d'analyse interprète les bits du paquet arrivant sur le lien et détermine les protocoles en présence ainsi que les différentes caractéristiques de ces protocoles. Le filtrage ne sert qu'à déterminer si un paquet doit être délivré à une application. Là où le filtrage des paquets est une étape spécifique dont l'exécution peut être laissée au soin de chaque application le nécessitant, l'analyse est une étape commune à effectuer en amont pour toutes les applications afin d'éviter les opérations en doublon. Sa configurabilité détermine la flexibilité sur les types de paquets qui peuvent être traités par la sonde. Son temps de réaction à la gestion de nouveau paquet est critique pour l'ensemble de la sonde. Il est donc extrêmement important de réussir à faire un analyseur de paquets qui extrait les types de paquets et leurs caractéristiques dynamiquement.

L'étude de la structure d'un paquet montre que toutes les informations nécessaires pour désencapsuler un paquet sont embarquées dedans. L'organisation des octets d'un en-tête est défini selon une spécification, chaque champ ou caractéritique étant à une place précise dans l'en-tête. De plus, les en-têtes de chaque protocoles sont soit de taille fixe, soit contiennent un champ à un endroit fixe de l'en-tête donnant sa taille. Les en-têtes ont aussi un champ permettant de déterminer le protocole utilisé dans l'en-tête de niveau supérieur. Ces informations peuvent être utilisées à chaque niveau d'encapsulation pour extraire les informations voulues.

Les solutions d'analyse de paquets existantes fixent ces paramètres d'en-tête lors de la génération de l'architecture matérielle. Des outils sont utilisés pour accélérer les développement, mais un changement de protocoles envisagés entraînent toujours une reconfiguration du FPGA, que ce soit totale ou partielle. A l'inverse, ce manuscrit propose une architecture qui utilisent les paramètres des en-têtes comme paramètres de traitement. Il est alors possible de changer les types d'en-têtes et les caractéristiques extraites sans avoir besoin de reconfigurer le FPGA. Dans un premier temps, un pipeline d'analyse d'en-tête est utilisé pour supprimer la séquentialité entre les différents niveaux d'encapsulation. Les différents en-têtes extraits sont ensuite envoyés dans un ensemble d'extracteurs de caractéristiques en parallèle. Cette extraction en deux temps permet d'extraire n'importe quel nombre de caractéristiques des en-têtes à n'importe quel niveau d'encapsulation.

Le dimensionnement de l'architecture proposée permet de tenir le débit de paquets du lien, même dans le pire cas, comme les architectures existantes. L'avantage de l'innovation proposée, l'utilisation du FPGA en statique, permet d'éviter la perte de paquets lors du changement de paramètres. Ce changement est de plus complètement contrôlé à partir d'une suite logicielle sur la machine hôte, ce qui évite d'avoir besoin de connaissance spécifique en FPGA pour configurer l'architecture. Cependant, la généricité du design vient avec un coût pour la consommation en ressources de la puce. En comparaison avec les autres solutions, cette consommation est légèrement plus élevée et est largement compensée par la diversité de protocole pouvant être traités. Il est tout de même bon de noter le nombre de caractéristiques à extraire est le facteur majoritaire de cette consommation de ressources. Quand le nombre de caractéristiques à extraire augmente, la taille du design augmente en proportion.

Cette influence est à prendre en compte pour la transmission des caractéristiques extraites aux applications après l'analyseur dans la chaîne de traitement. En effet, afin d'avoir une sonde complètement dynamique, ces caractéristiques doivent pouvoir être distribuées dynamiquement aux éléments de traitement. Une caractéristique doit pouvoir être délivrée à un ou plusieurs éléments de traitement à la demande des applications utilisatrices. Comme les chemins sur le FPGA sont fixes après la configuration, le moyen le plus simple est d'extraire une caractéristique pour chaque entrée des éléments de traitement, quitte à dupliquer l'extraction de certaines caractéristiques. Cette duplication a néanmoins un coût important en termes de consommation de ressources, ce qui limite le nombre total de caractéristiques extraites.

Un moyen plus efficace de connecter l'analyseur aux moteurs de traitement est l'utilisation d'un élément spécialisé appelé réseau d'interconnexion. La capacité d'un réseau d'interconnexion à connecter une entrée à certaines sorties dépend de sa topologie. Pour répondre aux besoins de l'architecture, le réseau à utilisé doit être asymétrique, avoir des connexions réarrangeables, une latence de transmission constante et pouvoir supporter des connexions en multicast. Cette propriété doivent être répondues avec un minimum de points d'interconnexion afin de minimiser la consommation de ressources.

Une famille de réseaux d'interconnexion répondant à ces critères est l'ensemble des réseaux d'interconnexion à plusieurs étages, Multistage Interconnexion Networks (MIN). Cependant, aucune implémentation pour l'asymétrie n'a été trouvée. Une implémentation basique d'un réseau asymétrique est faite à partir d'étages de duplication et de réseaux de Benes, topologie d'interconnexion symétrique couramment utilisée. Cette solution offre alors un réseau d'interconnexion asymétrique capable de connecter les entrées à des milliers de sorties. Une solution plus générale basée sur les réseaux de Clos semble plus adaptée aux besoins en théorie mais elle nécessite une étude approfondie de nombreux paramètres pour être implémentée. Bien que non optimal, le choix effectué permet de réduire de façon significante la consommation en ressources comparé à la duplication de caractéristiques.

Avec la combinaison de l'analyseur de paquets et du réseau d'interconnexion proposés, l'architecture exécute l'extraction de n'importe caractéristiques de n'importe quel en-tête et peut les distribuer à n'importe moteur de traitement. Cette unique architecture est un bloc de base important pour la réalisation d'applications de traitement de paquets flexibles à très haut débit.

5 Vers de la surveillance de trafic agile à très haut débit

Le développement de sondes de surveillance de trafic permet l'évaluation de la pertinence de l'approche de conception commune logicielle et matérielle. Cela permet de tester la capacité de l'architecture proposée à s'adapter aux paquets arrivant sur le lien. Étudier le comportement de telles applications donne les clés nécessaires pour assurer la tenue des débits du lien tout en conservant un maximum de flexibilité. Cette partie s'intéresse à la mise en place de deux applications de surveillance de trafic conservant une agilité importante même à très haut débit.

Afin d'évaluer les performances des applications, une plate-forme de test capable de générer un trafic de stress est utilisée. Constituée d'une partie génération et d'une partie réception basées sur des cartes NetFPGA SUME, cette plate-forme génère du trafic à 40 Gbps. Le trafic généré sature le lien, même dans le pire cas de paquets de 64 octets, et est de composition variable. Pour cela le générateur de trafic présenté dans ce manuscrit est utilisé. Des programmes créés par l'utilisateur génèrent des profils de trafic qui peuvent être répétés afin d'être utilisés comme gabarits de test.

Une partie importante de la protection des réseaux vient de l'étude des anomalies dans le trafic. La détection et la compréhension de ces anomalies permettent de révéler des motifs d'attaques. Ces motifs d'attaque peuvent être constitués de toute la diversité de paquets existante. Le suivi de tels motifs impose d'être capable de s'adapter dynamiquement au trafic pour rafiner la récupération de données.

Dans cet exemple, une application de traitement de trafic simple est exécutée avec l'analyseur d'en-tête proposé. Chaque caractéristique extraite des paquets est associée avec un compteur de motif. Les compteurs sont ensuite exportés en logiciel où une application développée en Python détecte les anomalies en cours et change les paramètres pour rafiner les traitements. Grâce à l'unique flexibilité de l'architecture matérielle, cette application est capable de collecter une information plus diversifiée sur le trafic que les applications habituelles. En effet, les applications logicielles ne prennent en compte que les champs discrimants des paquets pour cause de problèmes de performance. Les solutions matérielles utilisent ces mêmes champs à cause du coût de la reconfiguration du FPGA. L'adaptation en temps réel de la sonde proposée permet de surveiller des protocoles non communs qui peuvent devenir une source de danger parce qu'ils sont moins surveillé, surtout à des débits de 40 Gbps. Finalement, le réseau est plus sûr malgré la simplicité de la sonde.

La seconde application testée combine l'architecture de l'analyseur de paquet et du réseau d'interconnexion and des processeurs de règles pour créer un classificateur de paquets à très haute performance. Les processeurs de règles utilisent les champs extraits de l'analyseur pour déterminer si un paquet doit être transmis aux applications en logiciel pour un traitement plus poussé sur un débit moins élevé. Cette sélection en matériel peut ensuite être rafinée au niveau logiciel si il y a besoin. Une bibliothèque logicielle permet aux applications de manipuler la configuration de l'architecture sans avoir besoin de connaissances préalables. Il y a une boucle de rétro-action entre le logiciel et le matériel qui laisse la possibilité aux applications de contrôler les informations issues de la carte selon leur besoin.

La haute programmabilité de cette architecture offre une plus grande liberté dans la configuration des règles qu'aucun autre classificateur à très haut débit. Grâce à

l'analyseur configurable, l'utilisateur a la possibilité de configurer les caractéristiques utilisées dans chaque processeur de règles, ce qui rend le classificateur proposé unique. La généricité de l'architecture coûte de la consommation de ressources sur le FPGA, moins de règles pouvant être effectuée que d'autres solutions matérielles existantes. Cela est compensé par sa flexibilité qui autorise l'utilisation d'un processeur de règles pour n'importe quel type de paquet. De plus, le test vérifie le fonctionnement de la sonde à 40 Gbps sans perte de paquets, mais l'architecture est capable de réaliser la classification jusqu'à 120 Gbps de trafic constitué de paquets de 64 octets.

Bien que simples, les applications présentées mettent en exergue la pertinence de l'approche proposée dans le manuscrit. Avec des blocs de base adaptés en matériel, cette approche amènent au développement de sondes innovantes capable de :

- soutenir le débit du lien sans perte de paquet, même à très haut débit,
- exécuter des traitements flexibles,
- contrôler le nombre de données traitées par le CPU,
- augmenter la réactivité des traitements.

L'unique combinaison de ces propriétés est rendue possible par une conception commune logicielle et matérielle. Cela rend les sondes au courant du trafic en circulation qui peuvent s'adapter et être résilientes à un grand nombre de types de trafic. Des larges attaques volumétriques aux petites attaques cachées, les opérations réseaux peuvent parfaitement suivre l'évolution du trafic en évitant toute faille de sécurité.

6 Conclusion

Le développement des réseaux informatiques va de pair avec une augmentation de la quantité et de la diversité de trafic. Cette évolution amène de nouvelles contraintes pour les systèmes de surveillance de trafic protégeant les réseaux. Les solutions actuelles n'arrivent pas à faire face à ce nouveau set de contraintes. Les solutions logicielles ne sont plus assez performantes et les solutions matérielles pas assez flexibles. L'intégration de FPGAs dans des cartes réseaux amène une solution intermédiaire réussissant à lever les contraintes.

Cependant, ces smart NICs sont majoritairement utilisées comme des accélérateurs de traitement spécialisés. La proximité avec le CPU n'est pas pleinement utilisée, ce qui limite la flexibilité et la réactivité des traitements nécessaires au bon fonctionnement des équipements réseaux. Cette thèse propose une approche combinant le logiciel et le matériel dès la conception d'une sonde de surveillance. Sans déléguer toute une application au FPGA, mais seulement les parties critiques de la réception des paquets, les traitements sont accélérés mais gagnent en flexibilité. Pour cela, il est nécessaire d'avoir une architecture matérielle adéquate.

La conception d'un analyseur de paquets paramétrable montre qu'il est possible de d'échanger un peu d'espace sur une puce pour gagner beaucoup de flexibilité en comparaison avec les solutions existantes. Deux applications agiles et performantes sont conçues grâce à l'architecture proposée. La première, une application de détection d'anomalies dans le trafic, prouve que l'approche commune logicielle et matérielle

gagne en flexibilité et diversité de paquets traités. La seconde application est une application agile d'analyse de trafic. Avec l'ajout d'un classificateur matériel paramétrable, il est possible de rafiner les traitements sur certains paquets choisis parmi l'ensemble du trafic. Par rapport à l'existant, l'architecture de classification ne sacrifie que peu de ressources pour un gain en agilité et en diversité de paquets traités important. L'intégration de ces solutions dans une infrastructure de test permet de valider que les performances sont toujours au rendez-vous. Les applications sont capables de capturer tout le trafic généré à 40 Gbps, même pour les plus petits paquets de 64 octets, avec le générateur flexible proposé dans le manuscrit.

La validation des architectures n'est effectuée que sur 40 Gbps à cause des équipements de test. Le passage à l'échelle de la solution est à étudier pour envisager les réseaux informatiques de demain à des débits toujours plus importants. Avec de nouvelles cartes disponibles, équipées d'interfaces plus récentes, il sera possible de tester l'architecture sur des débits 160 Gbps à 400 Gbps. Passer à l'échelle nécessite d'optimiser la consommation en ressources du pré-traitement, afin de laisser de la place pour accélérer les applications utilisateurs. Dans cette optique, l'utilisation d'un réseau d'un réseau de Clos pour la distribution des caractéristiques issues de l'analyseur est à étudier. Il est aussi possible d'étudier l'intégration des évolutions technologiques des FPGAs, comme la reconfiguration partielle, dans l'architecture.

Les FPGAs, de par leur flexibilité et leur puissance de calcul, amènent de nouvelles possibilités pour les applications réseaux, et permettent de continuer à suivre l'évolution des débits. Un avantage important est la constante évolution technologique des FPGAs qui ouvrent de nouvelles possibilités pour les réseaux. On peut se demander si croissance démentielle des réseaux va finir par dépasser les capacités de traitements des cartes réseaux basées sur le FPGA. Cependant, comme les autres plateformes sont déjà obsolètes, les seules solutions pour faire face aux réseaux de demain, avec un trafic divers à débits de plusieurs terabits par seconde, viennent de la meilleure exploitation du potentiel des puces reconfigurables.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Network design issues

In the current Information Age, network communications are the keys of society development. From business to entertainment by way of mobile communications, network infrastructure is the backbone supporting the most modern and successful activities. E-commerce, social networks, video on demand are examples of services existing thanks to an efficient network.

To answer the evolution of the demand, the bandwidth of network data links has been largely increased. Users are connected with fast connections to enjoy proposed services. In 2016, according to Cisco Visual Networking Index (VNI) [Cisa], 46% of the world population is connected with an average connection speed of 27.5 Mbps. The resulting global IP traffic was 1.2 Zetabytes (10^{21} bytes) for the year 2016, making 3.2 Exabytes (10^{18} bytes) of traffic per day. This corresponds to an average throughput of 292 Tbps.

Nonetheless, this traffic is not evenly geographically and temporally distributed. Depending on their utility, multiple services are used across the day, with different traffic consumptions. During busy hours, the traffic spike was 1 Pbps (10^{15} bits) in 2016 [Cisa]. Therefore, network operators must be sized to sustain traffic during high demand periods. While aggregation of 10 Gbps links is currently the standard, it is common to see backbone architectures of main actors based on 40 Gbps or 100 Gbps links. For example, the French service provider OVH is currently upscaling its fiber optic communication network to 100 Gbps links in order to offer a global capacity of 12 Tbps [OVHc].

Global traffic is mainly composed of business traffic, consumers traffic and to a lesser extent network management traffic. Each service transmits packets through the network with specific transmission requirements. Video calls, video streaming, web browsing or gaming are examples of different requirements in terms of throughputs, latencies and jitters. Coexistence of different traffic types on the same network requires advanced management tools which can handle the modification of the traffic shape according to the different network usages.

1.1.1 Reliable networks

The growing importance of network has become a source of instability. The more services depend on network communications, the more pressure there is on the infrastructure. Indeed, if a network is down, all services relying on it are unavailable.

This critical status makes it sensible to failures or attacks. As a capital element of transmission, the infrastructure is a target of choice for any malicious behavior. In addition, any connected device may fail resulting in the generation of unwanted traffic.

To ensure reliability of the infrastructure, network actors control the network status with traffic monitoring tools. They have the vital role to extract information from packets in transit to deduce the network state. Knowing the traffic allows specialized packet processing for the different services.

For instance, firewalls accept packets inside a subnetwork according to user-defined policies. Traffic management systems handle packets priority and dynamic routing to guarantee a good Quality of Service (QoS) for all services and to avoid congestion. Indeed, traffic like video streams is much more sensible to delay than web browsing, thus requiring priority routing. To have an accurate control over the network, equipment has to tend to a lossless packet processing.

Thus, monitoring the traffic is crucial in terms of security and network management. The detection of abnormal behaviors is the key to avoid congestion, potential threats or service disruption. To offer an accurate management of the infrastructure, monitoring systems must be able to follow traffic packets at line rate.

1.1.2 High-speed networks

With the growth of network-oriented services, network transmissions require more and more bandwidth. Data links are regularly upgraded to absorb the growing traffic generated by networked applications. Increased throughput is equivalent to increased attack possibilities.

For instance, the growing number of Internet of Things devices is a major source of permanent threat to the entire infrastructure. Once deployed, a wide number of the same units are in service for an extended period of time. Representing 5.8 billions devices [Cisa] in 2016, 34% of all the connected devices, IoT objects offer a leverage to generate massive Distributed Denial of Service (DDoS) attacks [BI17]. Denial of Service (DoS) attacks aim at exhausting the target resources to disrupt the provided service.

One precedent was the Mirai botnet in autumn 2016. It took advantage of 150,000 poorly protected security cameras to generate DDoS attacks with traffic throughput never encountered before. 1 Tbps of malicious Transmission Control Protocol (TCP) traffic targeted the web service provider OVH in September 2016 [OVHb]. In October 2016 [Dyn], a similar attack targeted the Dyn company, provider of many well-known service companies.

Despite the traffic data rate, this was a simple well-known attack. Nonetheless, it paved the way to massive DoS attacks. The same leverage could be used by future malicious attempts to create attacks more complex to counter, like the combination of multiple classic attacks.

In high-speed traffic, even little data flows represent non negligible and potentially damaging quantity of data. 1% of 1 Tbps traffic is still 10 Gbps. In such a case, accuracy is the key to avoid any problem. Line rate packet processing is even more important in high-speed networks.

1.1.3 Flexible networks

A network is the center of the cohabitation of a wide number of different traffic flows. Services are not evenly and constantly used in a day. As a consequence, the resulting traffic composition is constantly in movement. Moreover, because of connection possibilities, a network is proned to transmit traffic comming from new services not anticipated before.

To cope with the current variety of traffic and its future evolution, a well-designed infrastructure is built with monitoring systems compatible with Software Defined Networking (SDN) [Zil+15]. This network paradigm recommends the dissociation of the data plane and the control plane. The goal is to abstract the network infrastructure to improve the maintenance with a global management. The data plane is focused on packet operations and offers a configuration interface. The control plane handles exceptions coming from data plane and adapts the data plane behavior. This approach allows to separate common operations on high speed links and management of more complex operations.

While respecting SDN standard requires to have programmable network equipment, the resulting benefits are a flexible architecture and a reduced global complexity. It is possible to dynamically tune the architecture to the traffic content. With the development of IoT [PSS16] and the arrival of 5G [And+14], programmable network will be required to handle concurrent intensive usage.

1.1.4 Scalable networks

The evolution of networks is linked with the evolution of dependent services. Estimations forecast [Cisa] that, in 2021, annual Internet Protocol (IP) traffic will reach 3.3 Zettabytes, for an average traffic of 874 Tbps and busy hours traffic of 5 Pbps. IoT modules will generate 5.1% of this traffic and smartphones will create 33% of it.

Thus, network monitoring importance is continuously increasing. The links will be upgraded to absorb the traffic generated by connected devices consuming more and more bandwith. Considering this future growth, monitoring systems must be scalable.

1.1.5 Reactive networks

Network actors face increasingly diverse and dense traffic. Considered data rates reduce the fault tolerance while the number of services increases the number of possible threats. Therefore, network infrastructure must provide a nearly perfect transmission service.

Monitoring systems must be not only flexible, but highly reactive too. Thus, flexibility of monitoring systems must not be limited to sparse upgrades. Indeed, anomalies in the network can quickly have a considerable impact. To efficiently protect a

network, monitoring systems must be adapted as fast as possible after the detection of an anomaly.

1.1.6 Monitoring challenge

Addressing these contraints is the key for network service providers to have a secure and well-functioning infrastructure. Monitoring probes are at the heart of the monitoring system since they collect information throughout the whole network. Designing them for tomorrow's networks requires overcoming multiple challenges.

How is it possible to monitor the traffic in an efficient and flexible way ? How to simultaneously cope with SDN requirements, high reactivity and high throughput support ? What is the scalability of such a monitoring system ?

1.2 Thesis structure

This manuscript studies different ways of lifting constraints on SDN monitoring probes in high throughput networks. An architecture is proposed to take advantage of hardware and software combination. The resulting probe is able to process high speed traffic with high configuration reactivity. Scalability is ensured by the portability of the solution, with the possibility to change the development platform.

In chapter 2, multiple existing traffic monitoring platforms are compared. Their compliancies with a set of essential constraints will be checked.

From limitations found in the literature, chapter 3 develops a novel approach for packet processing. Integration advantages of smart Network Interface Card (NIC) platforms are used to consider an architecture combining hardware and software. Separation of common packet processing steps allows to set high performance processing while keeping the flexibility. Moreover, a software Application Programming Interface (API) keeps the solution accessible for the final users, network engineers. In addition to accessibility, network operators preferably use trustworthy equipment whose behavior can be verified. Therefore, all the work presented in this manuscript is available as open-source.

Chapter 4 introduces an hardware architecture compliant with the global approach. A packet parser extracts features of interest from the incoming packets. An interconnection network distributes these features to the following processing elements. The whole architecture is based on parameters configurable from the software part. Thus, features are extracted from incoming packets on demand from processing applications.

The hardware architecture gives building blocks to set up a flexible and efficient monitoring probe. Chapter 5 is focused on the integration of these blocks in a global architecture to test the performance brought by the architecture. After counters set behind filtered extracted features, a complete programmable rule classifier is set. A software application is able to dynamically set up rules in order to detect anomalies at data rates over 40 Gbps. Performance and reactivity of these tests demonstrate the viability of the work.

Chapter 6 concludes the thesis. The different contributions are summarized. A discussion is engaged about the limits of the presented architecture. Development

possibilities with this platform are given. Finally, some tracks are explored to improve the solution.

Chapter 2

Systems for network monitoring

2.1 Introduction

Network infrastructure is currently under the pressure of the booming usage of network-oriented services. Managing the traffic going through a network is crucial for QoS and security. Network links are constantly upgraded to sustain a growing and diverse traffic. Monitoring systems face the challenge of combining high performance, flexibility and reactivity.

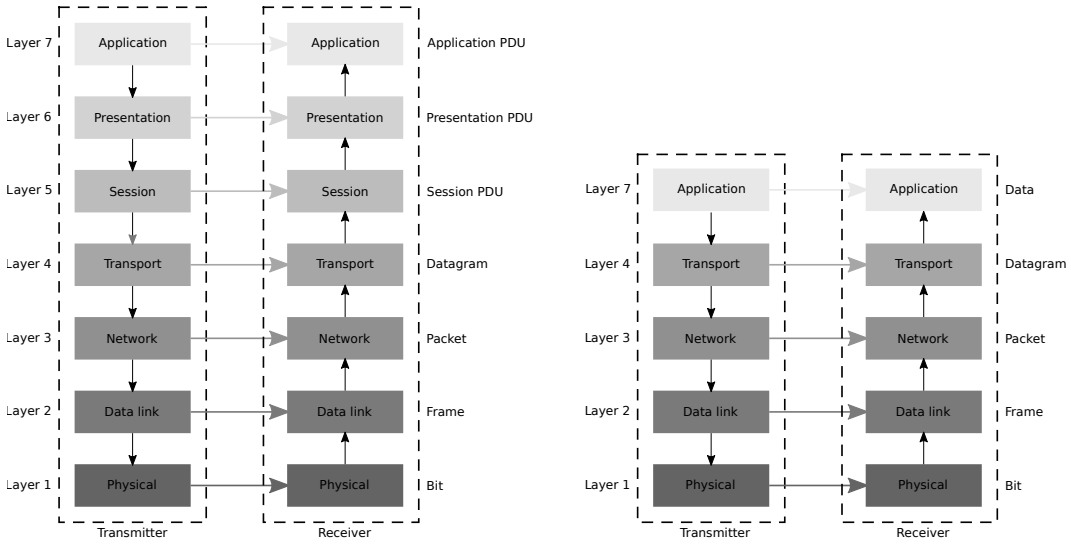
To overcome this challenge, monitoring systems must be composed of adapted probes. Numerous development platforms exist for the design of packet processing architectures. From classic CPU-oriented systems to Application-Specific Integrated Circuit (ASIC) specialized hardware, the wide spectrum of solutions offers multiple tradeoffs between performance and flexibility.

High speed network constraints define criteria for solution selection. To be considered adapted, a platform must fulfill the following requirements:

- precisely analyze the traffic at high speed,
- offer flexibility in order to:
  - be compliant with SDN,
  - continuously adapt the monitoring platform to the traffic,
- allow a low latency reaction with monitoring adaptation at link time scale,
- be easily adapted to larger throughputs and larger networks,
- be accessible to any network engineer without specific hardware knowledge,
- be portable for integration on various monitoring platforms.

Combined with a processing architecture, the resulting probe can sustain link rate processing of 40 Gbps links and beyond while being programmable and reactive.

Monitoring constraints are mainly decided by packets structure. Main constraints of monitoring come from packet transmission in network. After describing packet processing constraints, this chapter investigates the limits, strengths and weaknesses, of the different monitoring systems options. Each selected solution has a specific answer



(a) Original OSI model (b) Internet model

Figure 2.1 – Examples of layered communication

to high performance live monitoring, impacting the networking architecture. From these limitations, a platform is selected to develop a packet processing architecture respecting the requirements of current and future networks.

2.2 Packet processing

2.2.1 Packet structure

Network packet is the building block of network transmissions. When a communication is needed, information from the transmitter application is embedded inside one or multiple packets alongside control data. Control is used by the interconnection elements to deliver the payload to the destination. A packet is raw data formatted after communication protocols.

The dynamic and incremental nature of network gathers heterogeneous interconnection devices. Thus, the Open System Interconnection (OSI) reference model [Cisc] defines a layered conception of a communication system, where each layer is assigned to specific tasks. Figure 2.1a shows a communication example from an transmitter application to a receiver application for the original OSI model with 7 layers. The path of the packet is marked in black. Communications are made between two instances of the same layer with messages of a common standard protocol called Protocol Data Units (PDUs). Explicit names are given to messages corresponding to commonly used protocols:

- bit for the layer 1,
- frame for the layer 2,
- packet for the layer 3,

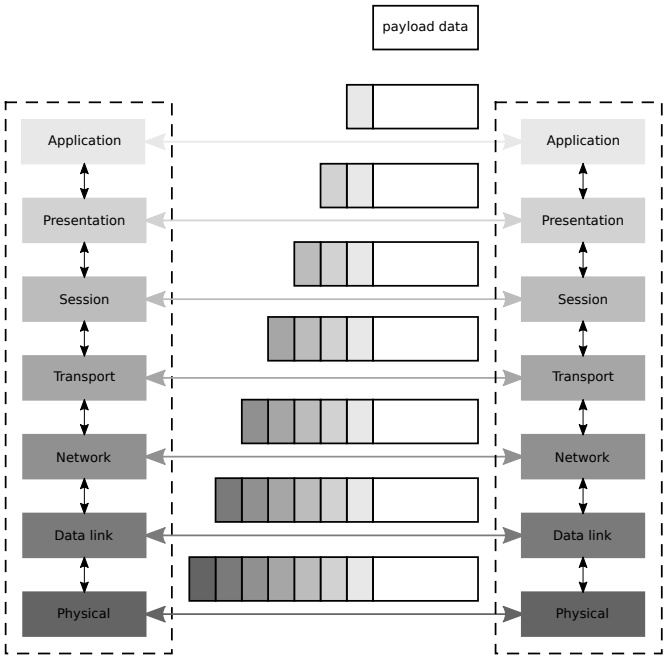


Figure 2.2 – Packet structure for OSI model

- datagram or segment for the layer 4.

A layer provides services for the upper layer. For example, the transport layer offers the transmission of data between two points on a network at higher layers. The goal is to abstract the network infrastructure for systems interoperability. This concept can be adapted according to the needs with a variable number of layers. Figure 2.1b represents the layer model applied to Internet communications where layers 5 and 6 are commonly not used.

This hierarchical structure of network communications is reflected on packet structure. At transmission, each layer adds control information in the form of a header and an optional trailer. This information is used by the corresponding layer at the receiving side. Figure 2.2 demonstrates this encapsulation with the resulting packet at each layer stage. At the physical layer, the packet is transmitted between network devices.

Data encapsulation is reversible at the reception side to extract the payload. Protocol header for one layer contains information to determine the higher level protocol. This decapsulation allows to remove headers one by one to deliver the packet payload to the right application. Removal of a packet header is dependent on information from lower layers. As a consequence, decapsulation process as well as encapsulation process are sequential. Processing parallelization is not possible on one packet.

Dynamic connected services create coexisting protocols at different layer levels. While lower layers of the Internet network use standard protocol for users traffic, higher layers are composed of as many protocols as existing services. Moreover, management traffic is transmitted alongside user traffic. As a consequence, there is a wide variety of packet structures in transit on a network, all of which have their specific needs regarding monitoring and protection against failures or attacks.

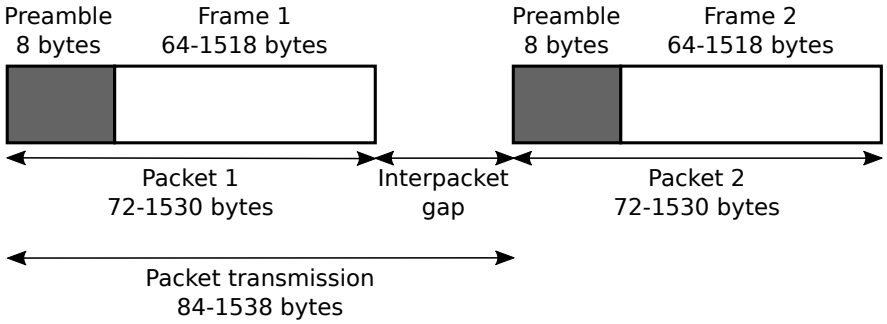


Figure 2.3 – Transmission of 2 successive Ethernet packets

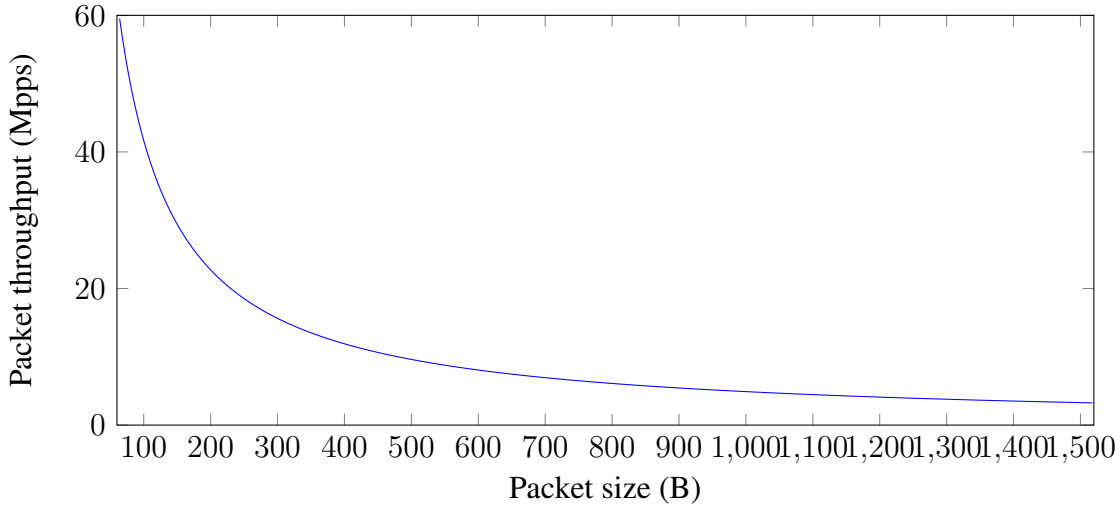


Figure 2.4 – Packet throughput evolution as a function of packet mean size for a 40 Gbps link

2.2.2 High data rate links: a packet density issue

A network component has a standardized physical access. The main communication protocol is Ethernet [Cisb], ruling the data link layer and partly the physical layer. It defines rules for transmission of packets between two nodes of the network and the composition of corresponding frames. Physically, this protocol is used for physical communication over a single network. This is a legacy vision of the network when the coaxial cable was shared among users. This cable has now been replaced by other equipments, mainly "hubs" and "switches".

When a frame is sent, a preamble of 8 bytes mark the beginning of the transmission forming an Ethernet packet. Between two packets, an interpacket gap of 12 bytes has to be respected. Moreover, by definition, an Ethernet frame is limited in size, from 64 bytes to 1518 bytes. Figure 2.3 summarizes the transmission of two successive packets. The restriction on the frame size brings the total transmission size between 84 bytes and 1538 bytes per packet.

The packet rate on the link can be evaluated with the mean packet number and the

Link throughput (Gbps)	64-byte packet throughput (Mpps)	1518-byte packet throughput (Mpps)
1	1.488	0.081
4	5.952	0.325
10	14.881	0.813
40	59.523	3.251
100	148.809	8.127
160	238.095	13.004
400	595.238	32.510

Table 2.1 – Limit packet throughputs for different data link speed

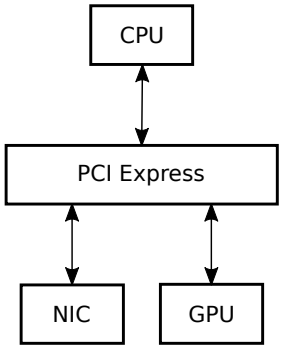


Figure 2.5 – Common commodity hardware architecture

link throughput, as seen in Equation 2.1.

$$T_{packet} = \frac{T_{link}}{(mean\_size + 20) * 8} \tag{2.1}$$

Figure 2.4 displays the packet throughput as a function of packet mean size for a 40 Gbps link. It can be easily observed that packet throughput is higher for 64-byte packets. The non-linearity of packet rate makes these packets a critical element for a network infrastructure. This is the worst-case scenario for packet processing. Table 2.1 lists minimum and maximum packet rates for different link rates. While a link is not constantly full of smallest packets, monitoring systems must be sized to prevent this case of overwhelming traffic. We will see in next sections what different monitoring systems offer and how they deal with the different traffic profiles.

2.3 Commodity hardware

2.3.1 Common architecture

Commodity hardware is based on common computer architecture to build a platform capable of processing packets. Widespread off-the-shelf CPU-based hardware allows a large accessibility. Combined with ease of use, this platform is an accessible target.

Figure 2.5 describes the architecture commonly used. A NIC handles packet transmission on the link. Specialized for this task, the NIC determines the maximum

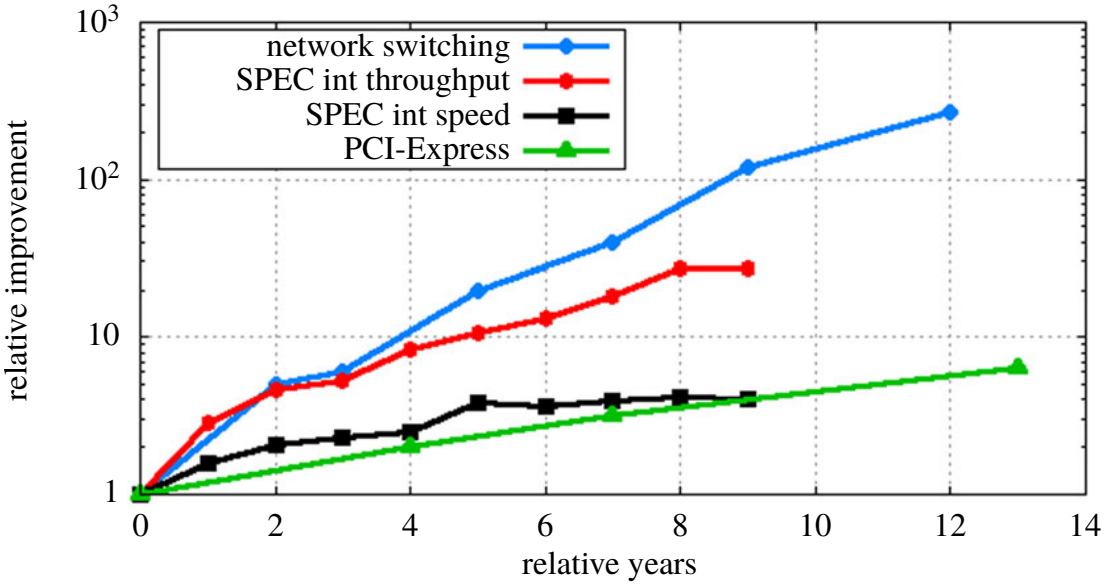


Figure 2.6 – Evolution of processing throughputs for 15 years until 2016 [ZMC16]

throughput that can be guaranteed by the probe. When received, packets are transferred to the Central Processing Unit (CPU) *via* the Peripherical Component Interconnect express (PCIe) bus. After interpretation, the CPU handles the main packet processing. A Graphics Processing Unit (GPU) is an optional element that can be added to offload the CPU in order to improve processing capabilities of the probe.

CPUs are conceived to be versatile. Software systems are greatly flexible allowing a reactive modification of processing and multiprocessing. Thus, software is the main development platform for packet processing engines. It is the reference for SDN development. Combined with the accessibility and portability, a large ecosystem of libraries and tools exists to manipulate network packets.

However, for the last 20 years, networking devices have improved datapath bandwidth more than CPU devices. In [ZMC16], Zilberman *et al.* studied this evolution. Results are summarized in Figure 2.6. Standard Performance Evaluation Corporation (SPEC) CPU2006 benchmark [Hen06] is a standardized and widely used benchmark to test performance of CPU devices. It can be observed that network devices increased their throughput far more quickly than CPU or PCIe.

While the NIC is designed to sustain the incoming traffic, the architecture of commodity hardware contains multiple possible bottlenecks. As main component of a CPU-based machine, the CPU is the major limitation. A high-performance GPU is a solution to offload processing from the CPU. However, between NIC and GPU communications, the PCIe bus must be able to sustain the total throughput.

2.3.2 PCI Express

High-speed communications over PCIe are ensured by specifications maintained by the PCI-SIG (Peripherical Component Interconnect (PCI) Special Group of Interest) [PS]. Table 2.2 shows throughput performance of PCIe generation 3 and generation 4 for multiple lanes width. Generation 3 is the current most widespread version of PCIe

PCIe version	number of lanes				
	x1	x2	x4	x8	x16
Gen 3	7.88	15.76	31.52	64.04	126
Gen 4	15.76	31.52	64.04	126	252

Table 2.2 – Theoretical throughputs (Gbps) per direction of PCIe

while generation 4 begins to be integrated in new machines. The results demonstrate the capability of commodity hardware to transfer the received network traffic.

It is common to see a CPU with 32 or 40 parallel PCIe Gen 3 lanes [Intc]. Commodity hardware with this configuration is able to sustain 100 Gbps of traffic with 16 lanes. Available lanes allow the connection of a GPU for processing acceleration.

However, protocol overhead and traffic overhead impact final performance of a PCIe link. Xilinx [Xil14] and Intel Programmable Solutions [Int17] have demonstrated that, for a PCIe Gen 3 link, the efficiency is 86% for write transactions and 76% for read transactions. Moreover, using the link in full duplex decreases even more the performance. Even given these issues, PCIe can still be sized to sustain real throughput for link up to 100 Gbps.

2.3.3 CPU computation

2.3.3.1 Common network stack

With an adapted NIC and PCIe bandwidth, commodity hardware is able to send and receive up to 100 Gbps traffic. Packet delivery is not sufficient to provide applications in high-speed traffic. Network applications are mainly developed on Linux for reasons of high accessibility and evolution possibilities.

Compatibility orientation and flexibility of modern Operating Systems (OSs) impact packet capture performance. The network stack of Linux kernel offers limited packet processing speed. A few improvements have been integrated to mitigate the overload of CPU with high density of packets.

Receive Side Scaling (RSS) RSS is a mechanism provided by NICs [Int16] to distribute received packets across multiple hardware queues. A compliant RSS driver [Mic17] allows to associate one CPU core to one queue. The developer configures distribution rules. The configured value corresponds to a hash value computed by the NIC based on some packet fields. While the linearity of packet processing avoids to distribute a packet on multiple cores, this approach enables to concurrently process multiple packets.

New Application Programming Interface (NAPI) NAPI [Fou16] is a way for the Linux kernel to reduce the impact of incoming packets on the CPU. NAPI compliant drivers improve capture performance with two main ideas:

- **Interrupt mitigation** avoids the reception of interruptions for each received packet. Interruptions are disabled when the first packet is received and a read is scheduled. During this read, newly received packets are fetched in batch and

interruptions are reactivated. The CPU is not overloaded with intensive interruption reception of high-speed traffic.

- **Packet throttling** avoids useless CPU processing for dropped packets. When the CPU is overloaded, packets are dropped at NIC level instead of dropping packets at kernel level. This saves CPU resources to process more packets.

2.3.3.2 High-end operations optimization

Despite these improvements, Linux network stack is still not adapted to sustain high-speed traffic. Further work on the architecture is necessary to upgrade performance. Multiple capture engines [Fou] [RDC11] [Riz12] [Han+10] [Bon+12] [Mor+15] propose different tradeoffs between common techniques to improve the supported data rate.

Memory resources pre-allocation The allocation of all needed resources for packet storage is made at the beginning of capture process. An incoming packet is associated with the already allocated memory. At packet removal, the memory is made available but not freed and re-used for another packet. Expensive per-packet allocation and deallocation is avoided, saving 2300 CPU cycles per packet [LZB11].

Queues consumption parallelism Although RSS queues allow different cores to concurrently receive packets, the Linux network stack merges all packets with one consumer process. With a modification of communication between kernel and user levels, independent cores can consume packets from different RSS queues. Keeping complete parallel paths allow to concurrently process incoming packets.

Memory mapping Mapping to user space memory regions allocated to packet storage avoids kernel to user copy operation. If such a region is writable by the NIC, packet data is immediately available for user space application and no copy is required. Otherwise, one kernel to kernel copy is still required.

Affinity In Non Uniform Memory Access (NUMA) architecture [Lam13] of modern computers, each processor has a privileged memory. Accesses to this memory is faster for the local processor than for the others. The capture process must be on the core attached to the PCIe slot where the NIC is plugged. Affinity must be defined to maximize the throughput. Moreover, processes using these packets will have to be set on the same processor too.

Batch processing The reception of a packet creates system calls, implying a context switch between user and kernel level. Processing batches of packets avoids creating a context switch per packet, costing 1000 CPU cycles each [LZB11].

Prefetching Fetching a packet in advance while the previous is processed allows to load the cache memory. The resulting fewer cache misses improve the overall performance.

2.3.3.3 Limited performance

A study of these different packet capture engines [Mor+15] demonstrates a significant increase in performance when compared to the standard Linux network stack. When using the standard stack on a 10 Gbps link, in the best configuration case, 75% of incoming packets can not be processed. This number is reduced to less than 5% for the best configuration cases for improved capture engine frameworks.

The test machine is equipped with two 6-cores CPUs and one NIC with a 10 Gbps interface. The best results are shown for less than 4 queues for all the capture engines. Despite the emphasis on parallel processing, capture is not optimal for the 6 supplied cores. This lack of scalability prevents the support of very high-speed NICs.

Moreover, it is worth noting that processing cores are nearly fully busy with the capture of packets. PFQ [Bon+12], DPDK [Fou] and HPCAP [Mor+15] use more than 99% of the CPU. While PF_RING [RDC11] uses only 77.8%, Packet Shader [Han+10] 77.4% and netmap [Riz12] 66.2%.

Capture is the first step of packet processing. Packets are received and delivered to following applications. All these frameworks have API to control communications between user applications and capture engines. However, once the capture is executed, few CPU processing power is available for applications. This is a real problem if processing is required for an elaborate and accurate monitoring.

Non specialization of processors makes for a difficult optimization of packet processing. The study of multiple frameworks [Gal+15] demonstrates that the most efficient driver between DPDK, PF_RING and netmap takes at least 100 CPU cycles to receive and send one packet. At 3.3 GHz, one CPU core must be able to receive 66 Mpps and transfer 33 Mpps, far more than 10 Gbps of 64-byte packets.

However, the executed application after the capture engines has a large importance on performance. The analysis of a simple processing, packet forwarding, with a lookup table of 256 MB in size leads to 7.1 Mpps processed [Gal+15]. This is less than 14.88 Mpps of 64-byte packets that need to be processed in a worst-case scenario on a 10 Gbps link.

A statistical classification application was built on the Packet Shader framework [Rio+12]. The equipment is scaled to process two links at 10 Gbps with one 4-core CPU per reception NIC. The final classification process is only able to process 15.2 Mpps out of 25.2 Mpps received for a worst case scenario. Intensive per-packet sniffing and flow construction require high CPU processing power and process parallelization and high CPU affinity are necessary. However, this way of processing is not scalable because of the limited number of available processing cores.

NDPI [Der+14] is a Deep Packet Inspection (DPI) tool using PF_RING to capture packets to study application level protocols. While the targeted throughput of 10 Gbps is achieved, results must be taken with a grain of salt. Average size of test packets is 316 bytes, which is far from 64-byte packet traffic.

2.3.4 GPU computation

With General Purpose Processing on Graphics Processing Unit (GPGPU), it is possible to execute applications formerly reserved to the CPU on GPU. High parallelism offered by GPU increases the overall processing power of commodity hardware.

Among the previous capture engines, Packet Shader framework [Han+10] has the possibility to unload captured packets processing to GPU. Internet Protocol version 4 (IPv4) forwarding sustains nearly 39 Gbps of 64-byte traffic where a CPU only architecture is capable of processing less than 30 Gbps. Although there is an incontestable gain, the acceleration largely depends on types of considered applications. For instance, Internet Protocol version 6 (IPv6) forwarding is able to process 38 Gbps with GPU processing instead of 8 Gbps with only CPU where, flow monitoring only achieves 34 Gbps instead of 28 Gbps.

GPU devices are accessible because GPU development is performed using traditional software development tools. However, applications efficiently accelerated meet certain constraints [NVI18a].

Efficient offload Computation gain of operations offloaded to GPU must compensate overheads of data transfer from CPU to GPU.

Massive parallelization The high number of cores offered by a GPU is optimally used if operations are performed in parallel. Algorithms with independent per-packet processing are an adapted target.

Coalesced memory To take advantage of small caches in GPU, data structures with grouped memory are recommended, like arrays or hash tables.

Single Instruction Multiple Data (SIMD) instructions GPU development requires a change in development paradigm, from independent threads to SIMD. GPU threads are divided in multiple groups called warps. The minimum is 32 threads per warp. In one group, all the threads have to execute the same instructions. It is then impossible to offload code with conditional branches.

For algorithms checking these cases, GPU processing has an impact for networking applications:

- The latency of global processing is increased by the two way trip from CPU to GPU and the waiting time to have a full batch of packets. The average latency added is 200µs [Han+10]. Such overhead could be critical for network services requiring a global low latency like 5G [And+14].
- Once concurrently processed, packet reordering is necessary for some applications, further increasing the global latency.

Even though GPUs accelerate applications with high computation, high regularity and memory intensive operations, GPU improvement does not reach targeted throughput. With an architecture sized for 80 Gbps of traffic equipped with two 4-core CPUs, each one receiving 40 Gbps and having an offload GPU, Packet Shader [Han+10] is only able to sustain 39 Gbps of worst-case packets for the most efficient application, an IPv4 forwarding. Similarly, Kargus [Jam+12] implements an Intrusion Detection System (IDS) on a machine sized for 40 Gbps of traffic with two 6-core processors, each one associated with 20 Gbps of traffic and a GPU. Final results show processing of only 30 Gbps traffic of smaller Ethernet packets.

A more recent implementation of Packet Shader [Kim+15] demonstrates that a commodity hardware system is not easily scalable. With two 8-core processors linked to a 40 Gbps NIC and high-end graphic cards, this machine is only able to process 62 Gbps out of 80 Gbps of 64-packets traffic for IPv4 forwarding.

Recent GPUs [NVI18b] have the capability to receive directly traffic from NIC without going through CPU thanks to Remote Direct Memory Access (RDMA). Nonetheless, obligation of SIMD limits greatly the possible algorithms.

2.3.5 Optimization solutions

With PCIe connection, commodity hardware has the bandwidth to transfer high-speed traffic. The combination of CPU and GPU makes possible to have efficient systems. However, sustaining high data rates is paired with multiple drawbacks.

Less flexibility Modifications made by capture engines to sustain rates specialize the architecture. Processes are associated with predefined cores, which greatly reduces the accessibility and flexibility of the platform.

Less portability Developed applications use specific frameworks with incompatible optimization. Some frameworks have specific languages to ease the development [Bon+14]. Software code is not portable or at the cost of high modifications, and the results is not guaranteed.

Poor scalability Commodity hardware systems have a poor scalability on a machine. Duplicating the machine is a solution to targeted high data rates. However, 100 Gbps implies to have at least four parallel machines. The scalability of commodity hardware requires the construction of a computer farm, which is an expensive solution.

Low reliability An OS hosts multiple concurrent tasks. Monitoring tasks have so important timing constraints that any outside task can lower monitoring performance. Therefore, the OS must be tuned to ensure a minimum reliability.

For the research of high performance, the different capture frameworks have optimized all kernel operations to reduce the number of CPU operations per packet. These works result in the specialization of processing on an architecture not specialized for packet processing. This usage of CPU is counter intuitive.

As a consequence, despite good improvements, the low scalability of commodity is an obstacle to process high packet rates. An alternative solution is to execute an adaptive sampling [Bra+13]. When the traffic rate is low, it is possible to watch all the packets. At the opposite, during traffic spikes, the CPU is not overwhelmed by packets improving the overall performance.

Supplying the right level of specialization to capture engines is equivalent to the usage of a dedicated commodity hardware. If a dedicated machine is required, specialized hardware must be considered. Indeed, for the past 20 years, processing bandwidth of more specialized systems has more increased than CPU and GPU [Zil+15].

2.4 Specialized hardware

2.4.1 Application specific hardware

While commodity hardware has processing power capability, it lacks operations optimized for packet processing. A common way to improve processing power for an application is the development of specialized hardware for this application. ASICs provide specific digital circuits tuned for targeted applications.

ASICs offer operations and instructions adapted to the application needs. It is possible to use parallelization to provide concurrent processing as well as highly pipelined designs. The association of these features allows to reach higher processing power [Goo]. An other advantage is an optimal consumption per operation.

Despite a substantial acceleration, dedicated hardware specialization leads to multiple major drawbacks for network applications [Xilb]:

- Hardware is not upgradable
- Time to market is very long
- Production costs are very high
- Proprietary designs are not specialized for customers

ASIC-based systems flexibility is restricted to the launch of a new product limiting at the same time the reactivity. Thus, this approach is not viable for emerging SDN.

Moreover, despite their importance, network components represent a niche market where network operators want to control final processing. The outlet is not of millions of copies, making production costs prohibitive. ASICs are mainly used for simple widespread high performance hardware like switches [Jun]. However, with growing adoption of SDN, even these elements require a certain level of programmability.

2.4.2 Network processors

Instead of ASICs, it is possible to use hardware designed to support a whole class of applications. Network Processing Units (NPUs) are specialized versions of CPUs for networking, offering a midway solution between CPUs and ASICs. NPUs are conceived to accelerate a collection of networking functions in addition to more classical operations packaged in CPUs.

In an NPU, processing units are directly connected to network interfaces. With adapted accelerated network functions, traffic processing is offloaded and performed at full rate. A maximum computation time is spared by the main processing unit to execute applications.

A wide variety of architectures exists for network processors depending on the different vendors. It is still possible to extract common elements [Com04]:

- Network interfaces communicates with the data link. They define the targeted data rate.

- One or multiple processors work like in a CPU in commodity hardware. Specialized for handling traffic, they have a reduced instruction set with special instructions to use co-processors.
- Network co-processors are ASICs associated with a networking task. These diverse co-processors give the specialization of the NPU.
- Global memory is available for processors and local memory is associated with co-processors if needed.

This architecture allows NPUs to sustain high data rates while keeping flexibility in processing possibilities. Development for these platforms is eased with the utilization of common language subsets and APIs. For example, Mellanox has a network processor [Mel] which is announced to support up to 240 Gbps of traffic. Simulation tools allow to have a cycle accuracy on the results. Nokia networks revealed the FP4 [Nok], a promising network processor for traffic up to 2.4 Tbps.

However, these platforms are specific to a vendor. Moreover, architectures are specific to a problem. The combination of these constraints leads to multiple drawbacks.

Reserved performance Proclaimed performance of an NPU is obtained in a configuration decided by the vendor. Usually, this configuration is chosen as a situation for which the architecture has been specially optimized. For example, a classification engine developed on Mellanox NP-5 processor [Pan+17] demonstrates that only 100 Gbps is reached only with packets longer than 127 bytes. It is far from 240 Gbps announced by Mellanox. It is easy to imagine that results would be far worst with 64-byte packets. When processing outside of co-processors scope is needed, performance collapses.

Limited flexibility For performance issues, flexibility of applications is limited by the offered co-processors. No update is possible to support unanticipated needs, except the production of a new NPU.

Lack of portability The code developed for a platform uses a language specific to a vendor. For performance improvements, assembly optimizations are advised by vendors [Com04]. Therefore, an application is specific to a vendor, and even to a platform with a specific co-processor, limiting the portability.

By merging CPU and ASIC approaches, NPUs seem to take drawbacks of both solutions. An application is reduced to choose between efficient but restricted or flexible but inefficient operations. To compensate, NPU vendors [Mel] tend to embedded a wide variety of co-processors in NPUs. As a consequence, end-users have NPUs with unused hardware overhead.

As NPUs are not adapted for many situations, one can observe the desertion of multiple leading actors. For instance, Intel produced its last NPU in 2007 [Inta]. Unfortunately, NPUs do not match the growing requirements of SDN wide flexibility and reactivity.

2.5 Novel network processors: FPGAs

2.5.1 Adapted platform for high performance networking

2.5.1.1 Platform features

Field-Programmable Gate Arrays (FPGAs) are integrated circuits that can be configured to behave like any digital circuit. At crossroads between commodity hardware and ASIC, FPGAs offer at the same time processing power close to ASIC and a level of programmability. Primarily used for ASIC incremental design, circuits can be tested and modified without the production of new chips at every modification.

An FPGA uses a programming paradigm similar to an ASIC combined with configuration possibility. This gives FPGA devices unique features:

- High parallelization can achieve high data rate processing. Largest FPGAs currently have millions of parallel logic cells [Xilc]
- Reconfiguration of the FPGA allows to change an existing design, update or substitution, offering a wide flexibility.
- Design implemented on an FPGA is controlled in terms of functionality and latency.

These features are interesting to solve challenges linked with constraints of network applications. It offers a compromise between high data rate processing and flexibility.

2.5.1.2 High performance processing

The main issue in network monitoring is the considerable flow of packets. When an application is accelerated on an FPGA, common acceleration factors are:

- design specialization,
- massive parallelism,
- data pipelining.

A design targeting an FPGA device is fully controlled by the developer. It is possible to specialize the behavior of an FPGA to act like an digital circuit. Therefore, as with ASIC development, operators dedicated to the targeted application can be used. This specialization of FPGA designs is one the keys to achieve high data rates. By unloading the per-packet processing to an FPGA, Antichi *et al.* [ACG12] succeeded to reduce the CPU utilization of nearly 100% for a full 64-byte 1 Gbps link. At the same time, packet loss was reduced from nearly 100% to 0%.

To increase the processing throughput of an FPGA design, the sequential steps of an application can be executed in parallel forming a processing pipeline. Packet data is sequentially transferred through pipeline stages allowing the consideration of dependent operations. At a given moment, the steps work on different data. For instance, packet parser architectures extensively use processing pipelines to achieve high data

rates. In [AB11], implementation results show that the presented architecture can process up to 400 Gbps of worst-case traffic. An other design [PKK12] [PKK14] demonstrates that 400 Gbps is not out of reach. In addition to pipeline processing, these two solutions take advantage of the possibility to use a specific datapath on FPGAs. A 2048-bit datapath allows to achieve these very high performance. Unfortunately, even if FPGAs have the processing power, no board with enough connection interfaces existed at the time to have a final test-bed.

Other applications hit the same wall. For instance, flow classification with Support Vector Machine (SVM) [GAV14] takes advantage of high parallelism to implement multiple computation units. As each computation unit processes one packet, traffic speed can be processed at up to 400 Gbps. However, with a board equipped with only four 10 Gpbs interfaces, tests are limited only to 40 Gbps.

With massive parallelism and data pipelining, the structure of FPGAs is adapted to process packets at line rate on high-speed links. Designs making use of FPGA features have high performance, even processing data rates better than those proposed by input and output connections. With recent and future FPGA generations, Xilinx [Xilc] and Intel [Intb] announce devices supporting 400 Gbps data rate.

2.5.1.3 SDN compliance

Like ASICs, FPGAs' gain comes in large part from application specialization. However, while designs implemented on an FPGA are specialized, the chip itself is not and can be reprogrammed. This feature offers a large flexibility to end-users. FPGAs are not bound to a single task. A single FPGA can be used to run different networking application.

Moreover, it is possible to adapt some applications to the different resources of an FPGA. For instance, in [FHS17], performance of multiple classification engines are compared on the XC7VX690T FPGA. These different architectures provide different tradeoffs between logic and Random Access Memory (RAM) consumption to achieve a similar application. This capability enables the organization of resources according to the requested operations for a high number of monitoring possibilities.

However, the flexibility is limited for applications relying on specific rare resources. For performance constraints, it can be not possible for multiple applications to concurrently use these resources. For instance, Forconesi *et al.* [For+13] succeeded in upgrading the number of concurrent flows tracked by the monitoring device from 16,384 to 786,432 with the usage of an external Quad Data Rate (QDR) memory. More accurate monitoring results are obtained because of less collisions in the hash function used. Despite superior performance, this design prevents the usage of the unique communication port to external memory for any other applications added on the spared resources of the FPGA. In [Var+15], Varga *et al.* developed a switch enhanced with traffic flow management capabilities to selectively distribute a 100 Gbps traffic on multiple other monitoring devices. Parallelism on an FPGA is limited by the resource consumption.

2.5.1.4 Portability

While each implementation is specific to an FPGA chip, this specificity is obtained *via* implementation tools. Designs are mainly developed with common Hardware Descrip-

tion Language (HDL) languages, Verilog and VHSIC Hardware Description Language (VHDL), supported by Xilinx and Intel. Implementations tools map the design to the resources of an FPGA chip. Therefore, proposed networking applications are portable across multiple platforms. This situation brings two main advantages.

Full flexibility Portability of designs allows to access to a wide variety of accelerated applications. The final functionality can be tuned according to the needs of the user. Similar blocks can be found even for vendor specific Intellectual Properties (IPs).

Scalability Having portable design boosts the scalability of network equipment. Indeed, it is possible to target recent hardware with a tried and tested design. Most recent hardware often provides more inputs and outputs connection, more resources and better target frequencies. As performance is often limited by clock frequency or resources consumption, portability to most recent hardware is linked to performance improvements.

2.5.2 Improvement of FPGA development

2.5.2.1 Automated design generation

Despite multiple advantages, FPGA systems are limited by long development time and specific expertise. For equivalent functionality, it is more complex to develop for FPGA than commodity hardware [Che+08]. In addition, applications have more lines of codes, and so are more complex to maintain.

To attenuate this limitation, work has been done to ease the development of algorithms for FPGA. High-Level Synthesis (HLS) tools use algorithm descriptions in a subset of a language, like C or C++, to generate a corresponding HDL design with the described functionality. Common HLS tools are Vivado HLS and CatapultC. With the development of a traffic manager in C, Benacer *et al.* [BBS17] demonstrated the possibility to have a complete networking application in HLS. This application succeeded to reach 15.8 Gbps of 64-byte traffic, which is enough for the 10 Gbps interface of the board.

Despite its proximity to common software, personal experiments with the design of networking applications have shown two main drawbacks of common HLS tools: lack of control and limited portability. Controlling parameters of the final design like latency, throughput requires mastery of the tool. The inference of the resulting generated circuit is not as obvious as with HDL languages. Moreover, as hardware systems are targeted, HLS solutions are nearly new languages to learn. Each tool uses a specific subset of common languages. Therefore, although the HDL code is portable, high level code is bound to the tool.

Common HLS tools are not adapted to networking usage constraints. Yet, software assisted development is a necessity to give access to FPGA processing power. For critical parts of a design, more specific HLS solutions have been studied.

Brebner [Bre09] defined a first packet-centric language, G language. A description of the design is compiled towards an architecture composed of specific blocks ensuring data rate. A label switched router is designed with G sustaining 40 Gbps traffic, with a

meaningful reduction of design time from 13 days for C-to-gates tools to 2 days for G approach. A flow monitor [McG+10] has been developed with G language, noting similarly design improvement results. This work was followed and developed with multiple other solutions. In addition to an architecture capable of processing up to 400 Gbps of packets, PP-based parsing stages [AB11] are based on an updatable microcode. If too significant changes are done between two updates, the parser architecture must be regenerated. PX language [BJ14] introduced an automated management of updates by the tool flow. A generated OpenFlow packet classifier processes packets at data rate of 100 Gbps. After multiple iterations, the outcome of Xilinx's research is the SDNet full product [Xila]. It combines automated generation and partial reconfiguration of FPGA to increase the flexibility while sustaining 100 Gbps.

Gorilla [LDC12] is an other design generation tool using a NPU like architecture. Processing is divided in multiple block operations, each coded in a subset of C. Operations are mapped on processing engines on an hardware architecture. A network router processing 100 Gbps is implemented with this solution.

Software-Defined Monitoring [KPK14] [Kek+16] uses the same approach in blocks. Multiple parsers extract metadata from packets transferred to computation units according to defined rules. Computation units are specific functions declared in C or C++ and compiled with HLS tools. It allows to easily add new computation units [Ben+14]. Packet parsers can be specifically generated for a protocol from P4 language definition [BPK16].

Despite different approaches, HLS tools ease the development of design targeting FPGAs for non specialist users. However, it is worth noting that these tools are using their own subset of common languages or even their own languages. As a consequence, FPGA design trades ease of use for less portability.

2.5.3 Limited reactivity

2.5.3.1 Reconfiguration cost

While FPGAs bring flexibility to network equipment, this flexibility is quite rigid. The modification of an already programmed design has two main drawbacks. Even if the complex and time-consuming design creation is attenuated by the usage of high-level tools, the generation takes hours, even days for complex architectures. In addition, programming an FPGA chip forces to stop current processing.

In [Fie+16], classification is done partly in hardware and partly in software to gain flexibility. However, when the rule set manager modifies rules on hardware part, the specialized rule classifier [Hag+14] must be regenerated. This operation comes with a cost:

- 45 minutes are necessary to generate a new design.
- 17 seconds are necessary to reprogram the FPGA.

In this case, the generation time is reduced because the design is a small specialized design. The generation time is a significant boundary for flexiblity.

Due to limited FPGA resources, designs are often tuned to fit requirements of specific processing for space optimization or performance issues. Therefore, it is not

possible to integrate processing for all expected configurations and then dynamically select the adapted operation. To mitigate the generation time, it is possible to use a set of pre-designed firmwares and to select a solution according to the situation. However, the FPGA configuration still requires several seconds during which processing is completely shut down. At 40 Gbps, the system has to buffer and recover or drop tens of GigaBytes of packets. Furthermore, if an unpredicted case occurs, a new firmware must be generated. This work is complex and time-consuming, even with the help of high-level tools, to meet the required performance in terms of latency and throughput while fitting FPGA constraints.

Common FPGA-based solutions provide a coarse-grain agility ideal for sparse firmware updates. Fast runtime adaptation of designs to the incoming traffic is not possible and requires a different approach.

2.5.3.2 Partial reconfiguration

A method to increase the reactivity is the recent possibility of partial reconfiguration of FPGAs. It allows to divide the chip in multiple independent reconfigurable areas. Therefore, an area can be reconfigured without impacting applications of the others.

A Network on Chip (NoC)-Enhanced packet parser [BAB15] hosts an header parser on each node. Routes between parsers can be reconfigured to change the decapsulation of the packet. The ultimate purpose of this design is to hold dynamic protocol configuration with a partial reconfiguration of router nodes.

Hager *et al.* [HBS15] defined a reconfigurable zone for core processing inside a static design. When a new processing is required, a bypass pipeline is used to continue to transfer packets to the following elements of the design. The probe is blind but no packet is lost during the reconfiguration.

Zazo *et al.* [Zaz+16] combined automated design generation and partial reconfiguration for a dynamic filter at 100 Gbps. The low occupation space of one specialized filtering pipeline allows to plan two parallel areas that can be reconfigured in ping-pong. When a new configuration is required, tools produce a specialized design from the new rule set. Then, filters can be changed without stopping the processing probe alternating between these two zones.

The interesting flexibility brought by partial reconfiguration has led its consideration as a research subject for networking applications inside the laboratory. The work focused on bitstream relocation aiming to minimize the number of bitstreams required for an FPGA with multiple reconfigurable areas [Lal+16]. This solution has been applied in the case of traffic generation [Lal17].

Despite the gain with reconfigurable zones, the designer must take into account the biggest fitting candidate design leading to resources overconsumption and limiting capabilities. Furthermore, the large size of newest FPGAs impacts the generation time for even parts of the FPGA, and specific tools are still required to handle the generation and the configuration. Partial reconfiguration is a good tool for the reconfiguration of non critical parts of a packet processing pipeline without impacting the rest.

2.5.4 Towards CPU offload

FPGAs offer a good compromise between high data rate processing and flexibility. However, the flexibility is provided at the cost of low reactivity. Continuous need of flexibility leads to consider using FPGAs paired with CPUs.

With FPGA-enhanced NICs [Bia+06], two paths are built for packets. The slow path reproduces the common path of commodity hardware through the network stack while the fast path transfers packets on a direct link between NICs. With classification abilities, FPGAs are able to select the right path for the packet. By bypassing the network stack, 400 Mbps processing was upgraded into 3 Gbps processing with a better usage of the PCI link.

In [Gar+12], Garnica *et al.* processed packets to detect suspicious IPv4 addresses. When a packet is tagged, it is transferred to the host CPU where Uniform Resource Locator (URL)'s legality is verified. Tested on a 10 Gbps link, this design has the scalability to sustain URL filtering on a 100 Gbps link, out of reach for commodity hardware.

As packet preprocessing is the key to reduce CPU processing load, Velan *et al.* [VP15] proposed an architecture where an FPGA extracts packet headers in CPU behalf. Headers are transferred to CPU memory *via* PCIe link like full packets on commodity hardware. This enhanced commodity hardware allows to process 160 Gbps of 128-byte packets. While only half of the traffic is processed for 64-byte packets, it is still an improvement compared to the solution with only commodity hardware.

To gain in flexibility, CPUs must be able to control the unloaded preprocessing. HyPaFilter solution [Fie+16] [Fie+17] paved the way to collaboration between CPU and FPGA. An automated tool compiles a set of rules partly on hardware and partly on software. Therefore, complex rules can be continued in software after a first reduction in hardware. Moreover, software rules are easily updated when needed without impacting the 40 Gbps traffic processing. Yet, the flexibility is limited due to the impossibility to quickly generate and reconfigure the hardware part.

Combination of flexibility and performance is the attractive trait of FPGAs. Associated to a CPU, an FPGA creates the next generation NPU. With the presence of an FPGA on the packet path, a CPU is able to offload processing with accelerated application on demand. Relevance of FPGAs for high speed SDN requirements has upgraded their status from prototyping platform to end-user product. Recent networking solutions are considering the usage of NICs enhanced with FPGAs [Bre15], called smart NICs, as seen in Microsoft Azure [Fir16].

However, monitoring high data rate links requires a fast reaction time to anomalies in the traffic evolution. To become a platform for future networking, FPGA-based solutions must compensate the low reactivity brought by classic conception flows.

2.6 Conclusion

Network monitoring faces high-speed links and a wide variety of packet structure. These constraints are the main challenges for monitoring platforms. Moreover, reactivity is an essential feature to avoid failures. In order to follow traffic evolution, scalability must be considered. In addition, a monitoring systems must be accessible

Criteria \ Solution	CPU	CPU+GPU	ASIC	NPU	FPGA
Throughput	very low	low	very high	very high	very high
Flexibility	very high	very high	very low	low	high
Reaction time	high	high	none	medium	low
Scalability	very low	very low	very high	very high	very high
Accessibility	very high	high	very low	medium	low
Portability	very high	high	very low	low	high

Table 2.3 – Comparison of platforms with networking criteria

and portable on multiple hardware targets to ease the development for network engineers. Multiple monitoring solutions offer a variety of compromises between these constraints which are summed up in Table 2.3.

Commodity hardware is the original platform for network monitoring. Accessibility, portability, flexibility and reactivity are main features of software development. Therefore, numerous tools for packet processing exist and ease the development of applications. However, packet processing power is limited, and the scalability means the construction of datacenters.

At the opposite, ASICs and NPUs offer high processing throughputs but answer few of other constraints. FPGA is a midway solution, programmable and capable of processing packets at line rate. Although accessibility is ensured by high level development tools, it comes at the cost of portability and processing flow restricts the reactivity. With the development of smart NICs, collaboration between the FPGA and CPU enables a gain in flexibility and an access to the wide pool of existing networking tools, but still requires further investigations, proposed in the next chapter.

Chapter 3

Hardware/software network devices

3.1 Introduction

With high processing power and flexibility, FPGAs make great solutions to answer networking constraints. The performance gain of the parallelization and specialization of processing is undeniable, but reconfiguration drawbacks restrain the flexibility to sparse firmware upgrades. This limited flexibility of the common working flow for FPGA acceleration is not high enough to answer the live adaptation requirement. Highly flexible processing needs a modification of the approach used for the development flow taking advantage of the integration of FPGAs in smart NICs.

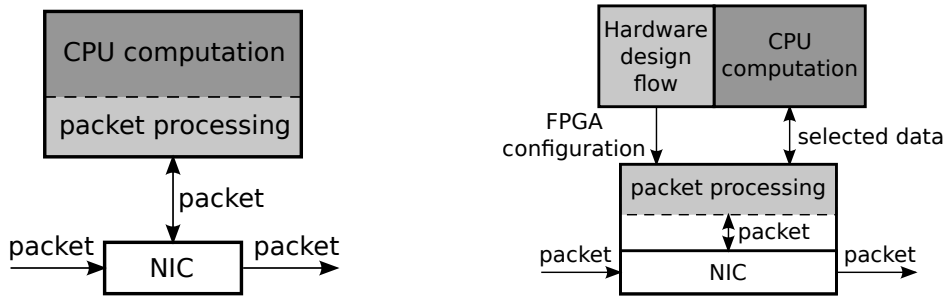
After presenting the current approach used with smart NICs, this chapter focuses on their usage for the development of flexible high performance architectures. Using both hardware and software strengths, these mixed architectures answer constraints of network monitoring while bringing more reactivity for better management. A major consideration will be the study of a separation of networking applications between hardware and software. In addition to accessibility, network operators preferably use trustworthy equipment whose behavior can be verified. Therefore, for a guarantee of transparency, all the work presented in this manuscript is available as open-source.

A configurable packet generator will first be designed with a feed-forward mechanism. The software creates configuration parameters and assigns them to the hardware part. The generation is only driven by scripts in software. While this solution is appropriate for some situations, a more complex separation between hardware and software is needed for packet monitoring. The second design will consider a feedback loop allowing the continuous adaptation of a global processing chain to incoming packets. The different parts of this chain are detailed in the next chapters.

3.2 Smart NIC approach

3.2.1 Smart NIC system

Major networking tools work on commodity hardware, the most widespread development. It is then impossible to bypass commodity hardware without loosing accumulated networking knowledge. Recent solutions bring intelligence to the common NICs.



(a) Commodity hardware with common NIC

(b) Commodity hardware with smart NIC

Figure 3.1 – Smart NIC integration in commodity hardware

These smart NIC devices aim at offloading CPU processing while taking advantage of the commodity hardware architecture.

Smart NICs offer programmable accelerators at the network interface level. The goal is the execution of a first stage of processing directly on the packet datapath in order to reduce the information transmitted to the CPU. These solutions combine hardware processing power with software flexibility to have high performance adaptable networking applications. Multiple approaches exist to implement this concept, but they often require specific development skills locked to a vendor's platform.

A widespread solution is the integration of an FPGAs into the commodity hardware packet processing flow. FPGA-based smart NICs take advantage of a well known development flow, more accessible, where the processing design is portable between different FPGA chips. The combination of high processing power and flexibility has already seduced multiple leading networking companies, like Microsoft [Mic] or OVH [OVH]. Figure 3.1 shows the difference between standard and FPGA-based smart NICs.

In Figure 3.1a, the totality of packets is transmitted between NIC and CPU. The CPU handles the whole network flow and processing chain. Basic packet processing, which provides no real value but is required before actual computation, can quickly overwhelms the CPU with a high quantity of simple operations, as seen in 2.3. The processor is then left unable to process any useful advanced computation.

In Figure 3.1b, an FPGA is inserted between the NIC and CPU. Intensive packet processing is moved to the FPGA part [Bre15]. FPGA processing power ensures link rate processing without packet loss. A reduced and adapted quantity of useful selected data is sent to the CPU. Moreover, FPGAs offer reconfigurable acceleration devices to offload CPU processing which can be adapted according to the needs [Fir16].

The combination of FPGA performance and CPU flexibility in a smart NIC based platform makes it a pertinent choice for high speed SDN requirements. Thus, smart NIC is a pertinent platform for high speed SDN requirements. The compromise of smart NICs between high processing power and flexibility has enhanced the status from prototyping platform to end-user product [Xilb].

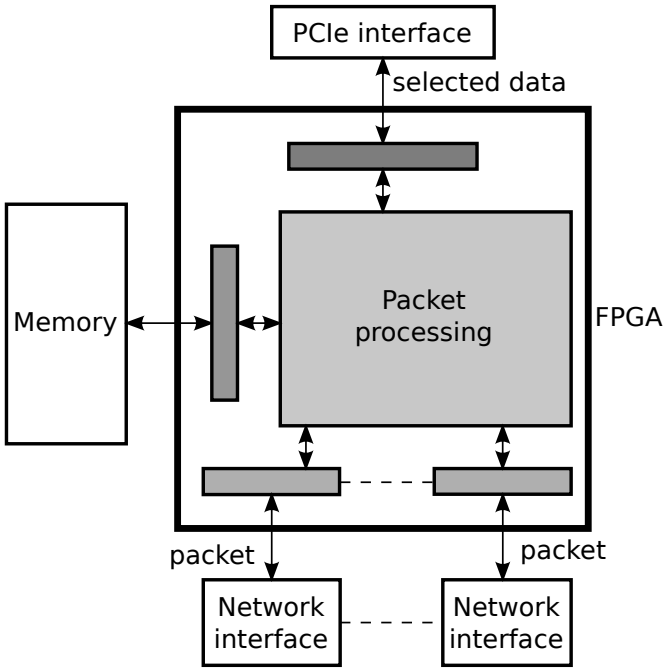


Figure 3.2 – Smart NIC architecture

3.2.2 Smart NIC development boards

3.2.2.1 Board features

The development of smart NIC architectures is linked to cards with adapted features. In addition to FPGA, such cards must be able to communicate with network link and the local host.

Figure 3.2 illustrates the architecture composition of a smart NIC. A smart NIC is composed of several key elements [Fri+13].

- The FPGA is the main component allowing a configurable specialized design to efficiently process incoming packets.
- The PCIe interface is the common connection interface to the local host of commodity hardware.
- Network interfaces handle the communication with other network devices thanks to Ethernet traffic on wired links.
- Complementary memory in addition to FPGA on-chip RAM is required for large packet storage, like Double Data Rate (DDR) Dynamic Random Access Memory (DRAM), or high-throughput accesses, like QDR Static Random Access Memory (SRAM).

In addition to packet processing, the FPGA part must have glue logic to control the different interfaces. Multiple boards present different tradeoffs for these features.

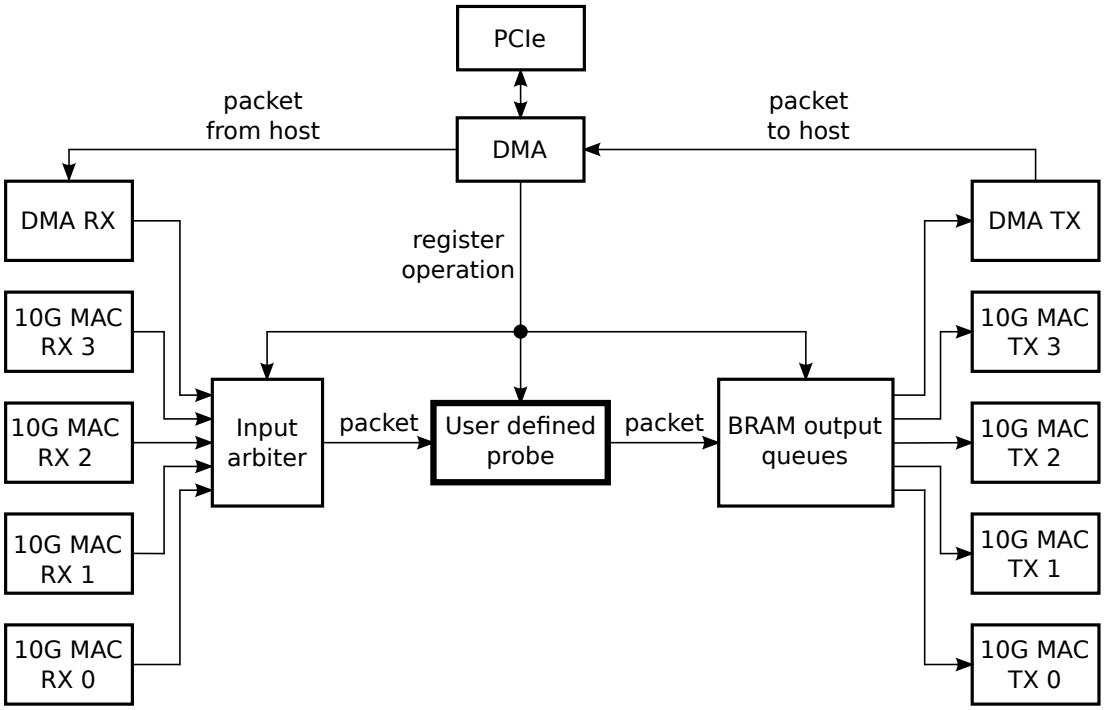


Figure 3.3 – NetFPGA flow pipeline

3.2.2.2 Prototyping: NetFPGA SUME board

NetFPGA project The NetFPGA project aims to offer the possibility to validate research work in high performance networking field. The NetFPGA SUME is an affordable platform designed for smart NIC prototyping. The Virtex 7 XC7VX690T FPGA offers high throughput performance for flow processing. The board is primarily designed for 40 Gbps of traffic with four enhanced Small Form-factor Pluggables (SFPs+) interfaces, but is able to reach 120 Gbps [Zil+14] traffic with extra interfaces added via an extension card. The board contains a PCIe Gen 3 connection giving the possibility to communicate with a host.

Design integration In addition to an adapted board, the project facilitates the integration of user-designed modules for the processing of a 40 Gbps traffic. Figure 3.3 shows the processing flow where incoming packets are focused and transmitted to pipelined modules. With only interfaces adaptation, a user-defined design can be inserted inside the flow to sustain the full data rate.

Network interfaces are supplied with FPGA IPs. A Direct Memory Access (DMA) system handles the PCIe controller for communications with the host computer. This implements the separation between register operation, reading and writing, and packet transmission. A software driver brings an easy manipulation of the multiple operations. Register operations are useful to transfer configuration parameters to the design. Processing results are sent to the host through local forged packets with user-defined protocols. The ease of integration makes the NetFPGA SUME a valuable source of test-bed for architecture tests.

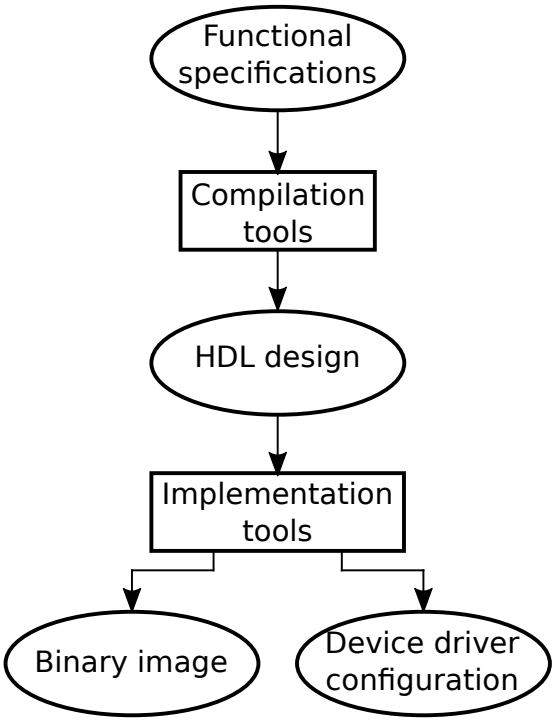


Figure 3.4 – High level network design flow

3.2.2.3 Commercial boards

The evolution of smart NIC requirements for networking has been followed by the production of adapted boards. Multiple leading FPGA board vendors like Terasic, HiTech Global or BittWare currently provide smart NICs. The market is galvanized by main network actors using smart NICs for networking applications. For instance, Microsoft [Mic] uses smart NICs for 30 Gbps networking operations and OVH [OVH] uses smart NICs in the attack mitigation architecture.

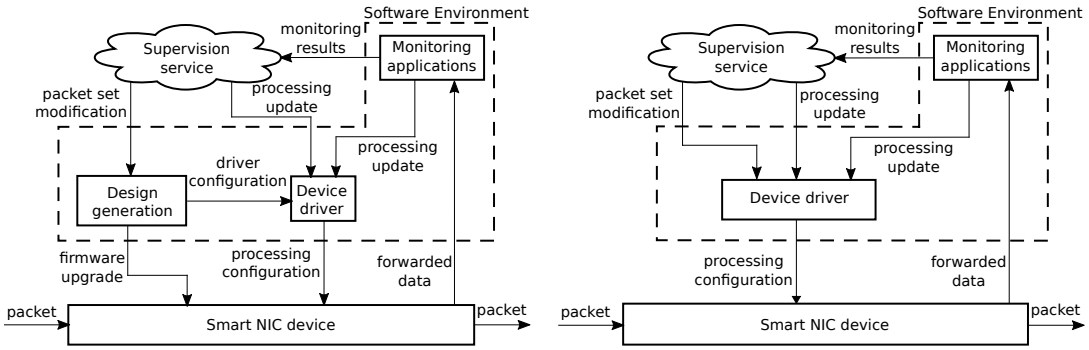
This concurrence has lead to the improvement of smart NICs. A major improvement is the direct integration in silicon of common connection controllers, like PCIe or Ethernet controllers. Removing this glue logic of high constraints reduces the pressure on the packet processing design implementation.

3.3 Hardware/software packet processing

3.3.1 Reactive processing flow

3.3.1.1 Considered approach

In current utilization of smart NICs, the FPGA is used as a classic hardware accelerator with upgrade possibilities. The conception flow of a specialized design is summarized in Figure 3.4. A set of specifications is used to determine the final architecture and its constraints. HLS and automated generation tools add a conception layer for accelerating the creation of a specialized HDL design for FPGA development tools compared to human development. Implementation on a target creates a binary image fitting FPGA



(a) Common (b) Proposed

Figure 3.5 – Smart NIC device adaptation loop

constraints and optional configuration files to drive the final device. This process is time-consuming, varying from hours to days for the most recent FPGA chips. Moreover, failure is possible when the design does not fit the target occupation constraints or the required throughput.

Packet processing designs are mainly specialized for a specific set of packets and specific information from packets. With the volatile nature of network traffic, some situations require extracting additional information as a complement to the current monitored set. Even if some solutions offer dynamic reconfiguration, the scope is focused on minor modifications of processing behavior like filtering actions or classification rule set. The support of volatile traffic and information is not considered, as seen in Section 2.5. Figure 3.5a shows the software feedback loop when a modification is required for a smart NIC. Monitoring processes rely heavily on the generation of new designs to handle traffic volatility. In addition to generation time, the reconfiguration process makes the FPGA chip unable to process incoming packets meanwhile, so the probe is blind during this time.

This problem is easily mitigated by the redundancy of networking critical systems with the reconfiguration of one system at a time avoiding any processing interruption. However, this procedure is heavy to set up and requires numerous operations to have a stable global system. Therefore, it is often reserved for planned maintenance or security issues. Moreover, the reconfiguration time is not reduced, the system still lacks reaction time. A standard FPGA design flow limits the reactivity of systems to sparse firmware upgrades. While sufficient for SDN flexibility, it is not enough for live adaptation to incoming traffic.

In order to tackle this problem, this work is focused on architectures based on a different paradigm. Smart NICs offer the possibility to use the combined advantages of FPGA and commodity hardware. Instead of generating HDL for each modification, it is possible to gain flexibility with the design and reuse of hardware processing elements for the maximum number of different packets without FPGA reconfiguration. With configuration parameters provided dynamically from the software, the generic architecture is only configured once on the FPGA; but the behavior can be modified according to the needs. Parameters are used for runtime adaptation, while reconfiguration is kept as a way to provide less frequent upgrade. Figure 3.5b shows the implied modifications in the software feedback loop. While a static hardware architecture is

not ideal for flexibility, using a processing based on update parameters reduces this limitation. A software API is an efficient way to integrate the configuration in a classic software flow for end-users. Moreover, handling all the parameters in software allows to track the configuration for a fast content-aware monitoring application. Combining hardware and software is the key to have developer friendly, high throughput and adaptive monitoring applications.

3.3.1.2 Network users friendly

The configuration of an FPGA-based system requires a specific knowledge to manipulate the tools and to resolve encountered problems. With configuration parameters, full control of the probe is executed in software without using specialized external tools. Controlling driver operations with an API abstracts the usage of hardware. Network engineers require no prior knowledge. Network applications interact transparently with hardware processing functions.

3.3.1.3 High throughput

Raw performance of FPGA guarantees to sustain a targeted packet rate with stability. The global monitoring architecture is then reliable and resilient to saturation traffic. It is possible to offload CPU from per-packet processing at full link speed. Using a static hardware design, the proposed approach does not need to stop monitoring for new configurations. The traffic is fully processed without packet loss.

3.3.1.4 Live flexibility

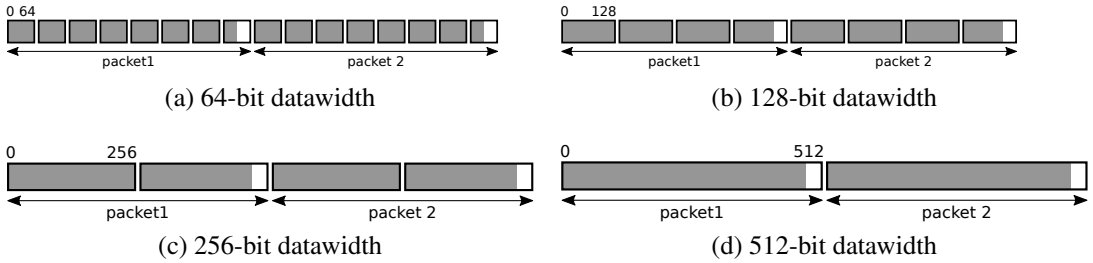
While the static design ensures the throughput, configurable parameters provides live specialization of processing units. The full software control of the configuration offers flexibility without the need of time-expensive FPGA reconfiguration. Runtime modifications are possible, allowing the probe to be content-aware and to adapt itself to the incoming traffic. Specialization of the hardware design brings a controlled low latency. The feedback loop is then low latency, enabling a reactive probe. Moreover, this approach is also fully compliant with slow firmware updates.

3.3.2 Hardware high packet rate processing

One main advantage of FPGA processing is the possibility to have a datapath adapted to applications. Where CPU datapath is limited to 64 bits, wider datapath can be used on FPGA. However, FPGA flexibility is linked to smaller operating frequencies. The announced maximum frequencies, 891 MHz for Virtex Ultrascale+ [Xil18] and 1.1 GHz for Stratix 10 [FPG17], are largely under frequencies of CPUs. With designs and components diversity, these theoretical values are never reached after implementation. Wider datapath is then mandatory to process high-speed data flow.

Link data rate (Gbps)	Datapath frequency (MHz)
10	14.881
40	59.523
100	148.809
160	238.095
400	595.238

Table 3.1 – Clock frequency to sustain different data link throughputs



(a) 64-bit datawidth

(b) 128-bit datawidth

(c) 256-bit datawidth

(d) 512-bit datawidth

Figure 3.6 – Ratio between link packet rate and FPGA packet rate

3.3.2.1 Fully parallel datapath

The maximum number of packets on a link is easily calculated with minimal size packets, as seen in Section 2.2. A simple solution is to have a data width equivalent to packet width to process one packet in one cycle. The operating frequency must be set to sustain the packet rate of the link. Table 3.1 lists frequencies for different targeted link speeds. Target frequencies can be easily achieved.

However, using this solution on full packets is limited by packet size. Indeed, packet size on FPGA datapath ranges from 60 bytes to 1514 bytes, because Ethernet 4-byte trailer is added or removed by the interface. A huge 12112-bit datapath must be used, which puts a lot of constraints on the implementation. Variable size packets are most of the time not max size packets, so the datapath is not efficiently filled. In addition, the mutual dependence of elements inside a packet creates high propagation times, preventing the design to meet final timing constraints.

This approach is not adapted for full packet processing. Nonetheless, a low operating frequency is an interesting property. It can be used for processing elements which do not require a whole packet.

3.3.2.2 Pipelined datapath

The previous limitations are mitigated thank to the transmission of packets with multiple pipelined chunks on smaller datapaths. Common datawidths used are 64, 128, 256 and 512 bits. Figure 3.6 shows chunk divisions of a 64-byte packet for the different widths. The packet has a size of 60 bytes without the Ethernet trailer. Datapaths over 512 bits are not considered because too large for minimal size packets. Therefore, the design must be able to process multiple packets at the same time. This is equivalent to have multiple concurrent instances of the same design processing packets on a 512-bit wide datapath.

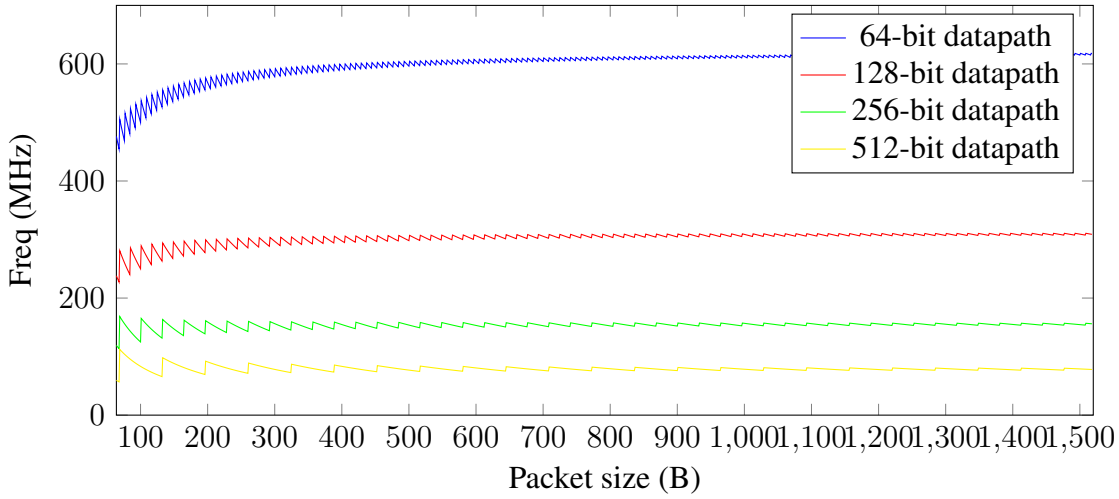


Figure 3.7 – Datapath frequency as a function of packet size for a link of 40 Gbps

The throughput of transferred packets is dependent on the datapath width and the associated clock frequency, as seen in Equation 3.1.

$$T_{packet/fpga} = \frac{Freq_{datapath}}{ceil(\frac{(mean\_size-4)*8}{datapath\_width})} \tag{3.1}$$

The size of the packet on the FPGA is reduced by 4 bytes because of the removed Ethernet trailer. It is possible to set the frequency to reach a specific packet throughput. Equation 3.2 gives the formula to calculate the frequency necessary to fully absorb the traffic of a link. To have the real size of the packet on the link, 20 bytes corresponding to the preamble and the interpacket gap must be added to the packet's size.

$$Freq_{datapath} = \frac{ceil(\frac{(mean\_size-4)*8}{datapath\_width})}{(mean\_size+20)*8} * T_{link} \tag{3.2}$$

Figure 3.7 displays the frequency evolution as a function of packet size for different datapath widths on a 40 Gbps link.

The spikes are created by the final chunk of a packet transmission, which can be incompletely filled. The transmission is still done with a full chunk leading to an overhead for packets whose sizes are not multiples of the chunk length. The real throughput is then lower than the raw throughput. This phenomenon is softened thanks to smaller datapaths and longer packets. The datapath must operate at the maximum frequency to avoid loosing packet of any size.

Table 3.2 summarizes the minimum frequencies to use for the different datapaths and the corresponding raw throughputs. It can be observed that 64-bit and 128-bit datapaths work at raw throughputs, but base frequencies are high enough to be prohibitive for high data rates. Wider datapaths allow to have frequencies more appropriate for FPGA design. For example, the NetFPGA SUME project uses a margin with 180 MHz for 40 Gbps traffic and a datapath width of 256 bits.

Link T'put (Gbps)	Datapath width	Min frequency (MHz)	Associated T'put (Gbps)
10	64	156.250	10.000
	128	78.125	10.000
	256	42.136	10.786
	512	28.125	14.400
40	64	625.000	40.000
	128	312.500	40.000
	256	168.540	43.146
	512	112.500	57.600
100	64	1562.000	100.000
	128	781.250	100.000
	256	412.349	107.865
	512	281.250	144.000
512	64	2500.000	160.000
	128	1250.000	160.000
	256	674.158	172.584
	512	450.000	230.400

Table 3.2 – Datapath frequency and throughput for different targeted link throughputs

3.4 Feed forward software to hardware: a mixed traffic generator

3.4.1 Overview

3.4.1.1 Generation requirements

Traffic generation is a necessity to be able to test network equipment. Although a direct access to real traffic is good, a controlled environment is required to stress the target with uncommon but probable traffic. The ability to generate synthetic traffic is the key to study limitations of a system. The generation of the traffic must be deterministic to be reproducible. A flexible and reactive architecture must be associated with live variation of traffic shape.

A large number of software tools enables an easy manipulation of packets for flexible packet generation. However, performance of software monitoring engines limits possible data rates, as seen in section 2.3. Moreover, creation process is conducted with processing of the full packet, still reducing the final performance of a software generator.

Adding an hardware block to sustain high data rates must not greatly reduce the easy manipulation of packets brought by the software. To generate variable traffic, the final packet generation architecture has multiple constraints.

- Packet must be generated at line rate.
- Several features of the generated synthetic traffic must be controlled from the software part:

  - the traffic composition,
  - the packet succession on the link,
  - the modification of traffic composition at runtime.

- A test must be reproducible.

The software control of traffic flows is ideal for packet manipulation taking advantage of regular software tools like scripts or PCAP files. During the packet generation, packets sent to the link are known and controlled in advance by the user. The goal of such a generation engine is the generation of controlled load traffic to fill the link. Dynamic replay of PCAP traffic at X$\times$10 Gbps is not possible because it is going through the CPU.

3.4.1.2 Solutions limitations

A common solution is the purchase of a commercial network generator. The high cost of these systems is an element of consideration. Proprietary, it is difficult to judge performance in terms of flexibility and throughput. For instance, a Xena generator was tested and unable to sustain the advertised 20 Gbps with 64-byte traffic. Having multiple hardware for flexibility is not a viable option.

An alternative solution is to use FPGAs to program an adapted generator. The generator used in the OSNT project [Ant] was the best promising solution with 20 Gbps of 64-byte traffic. It is presented [Ant+14] with multiple concurrent generation engines. Each engine is able to replay PCAP previously loaded on the FPGA RAM or to generate multiple synthetic traffic flows. In addition, this generator is designed to work on the NetFPGA SUME board, the same platform acquired for our tests. No supplementary hardware is required.

Unfortunately, only the replay engines are currently available for OSNT. These engines are designed to work with a full PCAP replay file. Therefore, does not allow to load data during packet generation. The solution is an increased size of the loaded file. However, the transfer time is 21 seconds for 10 MB files up to 28 minutes for 1 GB files [Ant]. 1 GB is equivalent to 0.2 second at 40 Gbps. Having 1 second of traffic requires to have at least 5 GB memory on the board for 40 Gbps traffic and 12.5 GB for 100 Gbps traffic. This solution is then not scalable and is not adapted for live traffic generation.

Groléat *et al.* [Gro+13] developed a synthetic traffic generator for 10 Gbps 64-byte traffic. Concurrent flow generators create a traffic stream corresponding to one type of packets, each one based on one different header skeleton. Successive blocks execute a specific operation, like random bits generation or value increment. Results of these operations are used to modify configurable bytes of the skeleton. Packets out of stream generators are merged on the link without control on the order. The offered flexibility is relative because stream generators are specialized for one type of packet. If a new type of packet is required, all the stream generators must be replaced.

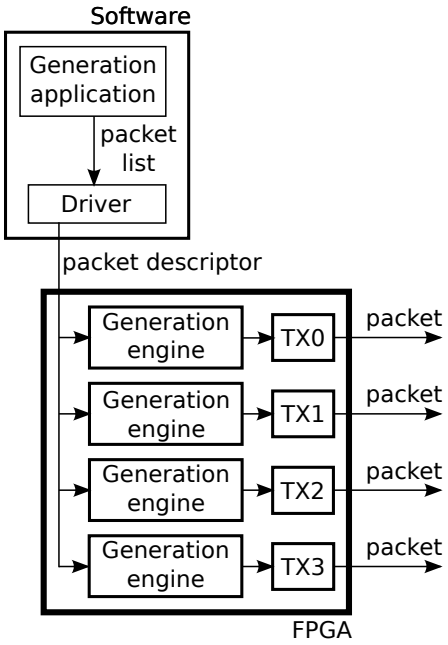


Figure 3.8 – Hardware and software packet generation architecture

3.4.2 Packet generator implementation

3.4.2.1 Objectives

The required packet generator aims to execute stress test on targeted devices. However, the goal is the generation of a dynamic load traffic, not the generation of synthetic traffic mimicking real traffic. The generated traffic can be composed of a limited amount of packets, while it is possible to change the composition.

As seen previously, using defined streams of packets allows a total control on the composition and the order of packets sent on the link, but large streams reduce the flexibility. The time taken to transfer the streams to the FPGA makes impossible runtime modification of the traffic. Thus, this generator focuses on the replay of small packet streams which can be dynamically changed at runtime. This streams can be generated from software scripts or PCAP files.

3.4.2.2 Architecture

The proposed packet generator creates synthetic traffic from the dynamic update of a replayed packet stream. The generation is divided into two distinct parts. The software part handles the creation of packet flows and the configuration parameters. The hardware part handles the traffic generation from the replayed packets at a precise link rate. Figure 3.8 shows hardware and software interactions for an architecture with four network interfaces.

An application requests the generation of traffic from one or multiple packet lists at a precise link rate. The list of packets is the list of bytes to send on the link. The software driver translates this packet list into a packet descriptor list. A packet descriptor contains information related to each packet:

- control parameters for the generation engine configuration,

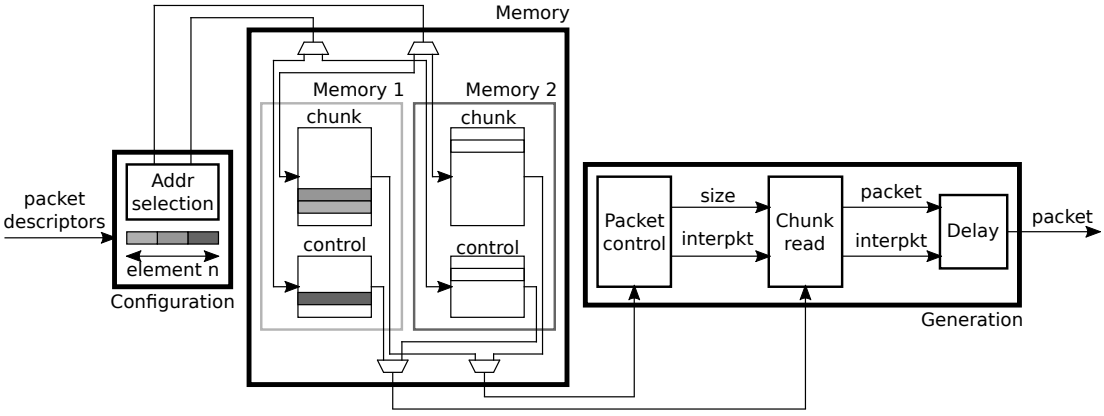


Figure 3.9 – Generation engine architecture

- bytes to sent for this packet.

One descriptor list is transferred to one generation engine.

Generator engines, one by interface, are in charge of the replay of the packet stream sent by the software driver with a specific configuration. The configuration is used to transmit the replayed packets at a targeted rate. This hardware engine ensures that the packet order and the data rate are observed. One generator is able to load a new configuration while transmitting with the current configuration. This feature is essential to dynamically change the transmission rate or the traffic composition.

This task separation enables the abstraction of the generation for the software. The driver can consider the different interfaces as a unique global link or as separated links. This change is done only with software configuration. For instance, if the four interfaces on Figure 3.8 are considered 10 Gbps interfaces, it is possible to have one 40 Gbps link, four 10 Gbps interfaces or any set division between the two.

Although it is not considered, traffic could be concurrently monitored to have a context-aware packet generator. The main usage of this generator is the saturation of a link with controlled packets for edge cases.

3.4.2.3 Generation engine

Figure 3.9 describes operations inside the generation engine. A generation engine is composed of three parts: a configuration part, a memory part and a generation part. While implemented on the NetFPGA SUME board, the design is fully generic and can be adapted according to the needs.

Memory A memory block is the combination of a control memory and a chunk memory. The chunk memory contains the bytes of packets to transmit. The transmission on a fixed datapath on the FPGA requires the separation of packets into successive data chunks. This division is already saved in the memory, each line corresponding to the chunk of one packet.

The control memory contains information useful for the packet stream generation:

- The address of the first chunk of a packet inside chunk memory.

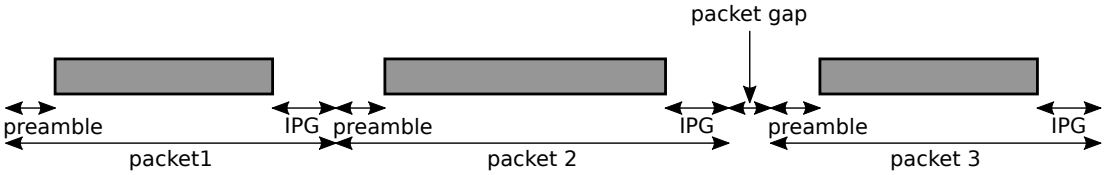


Figure 3.10 – Gaps between consecutive packets on a link when not full

- The length of a packet to know the number of chunks to use.
- The interpacket delay with the next packet of the stream.

With an interpacket delay adapted to the length of packets, it is possible to approximate a target throughput. This will be demonstrated in the next section.

The memory block is composed of two distinct memories. A memory is filled with a new configuration while the other is used for generation with the current configuration. At the end of configuration, the two memories are switched. This ping-pong solution allows the non-stop generation of traffic on the link.

Configuration The configuration block receives packet descriptors from the software. Information is separated between control information of a packet and the packet itself. The unused memory block is selected and configuration data are dispatched to control and chunk memories.

Generation The generation block uses the configuration to generate the successive packets on the link. After reading control information, chunks of the packet are read until the size is reached. These are transferred to the link, one at each clock cycle. When a packet is completely sent, the delay module blocks the transmission of the following packet during the configured interpacket gap time. It allows to observe the packet throughput set by the software.

3.4.2.4 Software architecture

Software integration At a software level, a driver handles the configuration of the hardware architecture. Applications have the possibility to use an API to control the generator. A list of packets is sent to the driver alongside a targeted link throughput for a specified interface. The driver handles the translation of the packet list into a descriptor list, and sends this configuration to the generator. It is currently possible to generate traffic directly with C++ and Python programs, allowing an easy access to network engineers.

Throughput and interpacket delay When the targeted throughput is smaller than the maximum throughput, the link is not filled with packets. A gap is created between two packets in addition to the common InterPacket Gap (IPG). Figure 3.10 demonstrates such a gap between packets 2 and 3.

A mean interpacket gap can be calculated corresponding to the gap if all the packets are uniformly distributed on the link. It can be considered as if wider packets were

transmitted on the link, leading to the packet throughput expressed in Equation 3.3. The mean interpacket gap in bytes is calculated in Equation 3.4.

$$\begin{aligned} T_{packet/link} &= \frac{T_{reduced\_link}}{(mean\_size + 20) * 8}, \\ &= \frac{T_{full\_link}}{(mean\_size + 20 + mean\_interpkt\_gap) * 8}, \end{aligned} \tag{3.3}$$

$$mean\_interpkt\_gap = (mean\_size + 20) * (\frac{T_{reduced\_link}}{T_{full\_link}} - 1), \tag{3.4}$$

$$mean\_gap\_cycles = \frac{Freq_{datapath}}{T_{packet/link}} - ceil(\frac{(mean\_size - 4) * 8}{datapath\_width}) \tag{3.5}$$

To achieve the same throughput as on the link, this interpacket gap must be applied to the transmission of packets on the FPGA. The only way is the insertion of gap cycles between the transmission of two consecutive packets. From the mean interpacket gap, it is possible to compute the mean number of cycles to leave between two packets, as seen in Equation 3.5. This value is always greater than or equal to zero with a datapath frequency sustaining the worst case scenario.

Packet descriptor creation In addition to the packet, a packet descriptor contains the packet size and the interpacket delay, the number of cycles to let blank after the transmission of the associated packet. Therefore, this interpacket delay is an integer. During the creation of the list of descriptors, the driver sets the interpacket delay of each packet to estimate the mean number of gap cycles corresponding to the targeted throughput.

The mean number of gap cycles can be approximated in two ways. When the approximation is constantly over the targeted value, the resulting throughput is always under the targeted throughput, it is the maximal throughput wanted on the link. When the approximation is constantly under the targeted value, the resulting throughput is always over the targeted throughput, it is the minimal throughput wanted on the link. Moreover, the longer the packet stream is, the better the approximation of the value is. If the memory depth allows it, the packet stream is duplicated to create a longer stream with the same packet composition.

With this approach, it is possible to create a packet stream composed of multiple flows. By controlling the weight ratio of each flow inside the packet stream, it is possible to control the throughput of each flow in relation to the total throughput. The generation process currently handles only uniformly distributed traffic. With easy software improvements, it would be possible to have a complex distribution interpacket delays inside the stream like Poisson distribution.

3.4.3 Strengths and limitations of the traffic generator

3.4.3.1 Implementation results

This generator is able to send continuously evolving traffic. Packet streams definition in software ensures a full control on sent packets, and configuration can be modified dynamically, which provides interesting flexibility. The hardware part ensures the

Number of chunks	LUTs (%)	FFs (%)	BRAM (%)	Slices (%)	LUT/FF pairs (%)
512	8.68	6.04	12.52	14.77	4.42
1024	8.69	6.04	16.60	14.95	4.43
2048	8.69	6.07	24.76	15.17	4.44
4096	8.70	6.11	45.44	15.64	4.44
8192	8.72	6.19	83.81	17.67	4.40

Table 3.3 – 4*40 Gbps packet generator implementations for different maximum sizes of packet streams on XC7VX690T FPGA

configured data rate on the link, allowing link saturation which is otherwise difficult to check. The software API allows to drive the generator from common networking programs.

The performance of the generator is tied to the performance of the hardware design performance. The resource consumption on the chip is the limiting factor for performance as well as scalability. Table 3.3 displays the resource consumption for different maximum numbers of 256-bit chunks in the packet streams. All the designs have a successful implementation for four interfaces at a clock of 180 MHz, allowing 40 Gbps per interface. Unfortunately, the NetFPGA SUME test board is only equipped with four 10 Gbps interfaces. Results in the table contains a flat resource consumption corresponding in particular to the glue logic of the different network and PCIe interfaces.

Two main observations can be extracted from Table 3.3. This simple generator is using few logic elements, even when the packet stream length increases. The generator will be easily scalable for upgraded interfaces. However, the memory consumption heavily increases with the number of chunks. The limited number of Block RAMs (BRAMs) available on FPGAs is the limiting factor for the creation of longer and more diverse packet streams.

3.4.3.2 Generator adaptation and perspectives

This simple traffic generator is an adapted solution for the creation of load traffic for the test of networking devices. It answers the requirements of multiple criteria:

- high-throughput traffic,
- live variation of traffic composition,
- control on the exact succession of packets on a link,
- an easy traffic generation from software,
- scripted generation thanks to the software API allowing the reproduction of tests.

This solution is used for the test-beds presented in chapter 5. As for all the test materials, it is available under an open-source license.

Multiple minor improvements have not been added to the proposed architecture because of time shortage.

- Packet reuse in memory for different places in the packet stream would allow a more efficient usage of the memory, enabling the generation of longer packet streams.
- Changing only one packet in the stream during the generation would remove the necessity to use two alternate memories. In addition, the modification of few packets would not trigger the reconfiguration of a full packet stream which can be time-consuming.
- The development of a management Graphical User Interface (GUI) would create a fully functional standalone generator for a better software integration.

An easy improvement of the architecture is the increase of the diversity of replayed packets. With increased memory, it is possible to have longer packet streams. However, this solution is not advisable as, for large packet streams, the transmission time from the host to the FPGA limits the flexibility and the reactivity, as seen for OSNT generator [Ant]. The low logic requirements of this design offers the possibility to transform memory into logic by the direct generation of packets on the FPGA. It would then be possible to reduce the amount of data transferred between the host and the FPGA. Thus, large packet streams would be possible.

3.4.3.3 Task separation validation

This solution validates the approach using smart NICs for more than just specialized reconfigurable accelerators. With a right task partition, the interaction between software and hardware makes the flexible generation of packets at high-speed link rate possible.

The hardware architecture is dedicated to high-speed packet generation, but it is designed with runtime configuration capabilities. Interactions with the user application, inducing complex decision and generation modification, are made in software for more flexibility. Modifications are translated in a new set of configuration parameters for the hardware part.

The key element for a working collaboration is the reduction of the amount of data transferred between the host and the FPGA chip compared to a solution in commodity hardware. The transmission of minimal information between the host and the FPGA avoids the overload of the software.

3.5 Hardware/Software feedback enabled probe

3.5.1 Packet processing steps

Monitoring applications use information extracted from packets, mainly packet headers, to determine the status of the network. CPU-based platforms usually used by network engineers are designed for general usage and reuse possibility. Therefore, mutual processing steps have been standardized for software packet processing. This allows resources optimization and avoids the new development of an existing task. Despite Linux stacks improvements, such steps are still present, as seen in section 2.3. Figure 3.11 illustrates these steps with the example of a packet processing architecture.

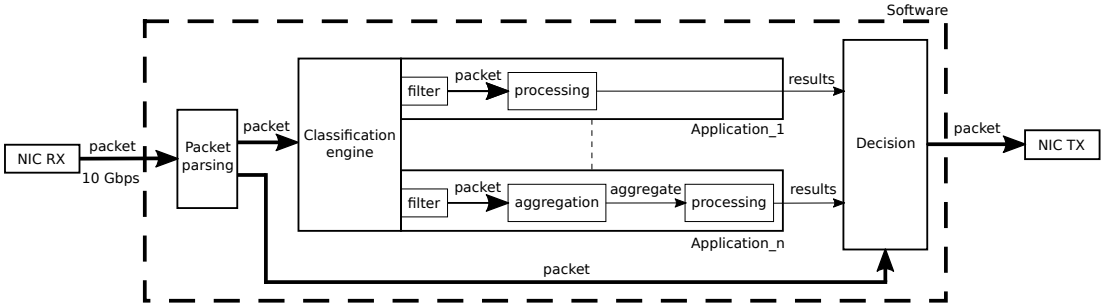


Figure 3.11 – Example of packet processing on a software architecture

3.5.1.1 Packet parsing

Packet parsing is a mandatory step for every packet processing. Packets are decapsulated and interpreted by the software. A data structure is filled with all the information about the packet. This common task is done by the network stack of the OS.

All data of interest are extracted from headers, like specific fields or selected bytes. A packet is ready to be processed by any networking application. Each packet is then distributed to monitoring programs running on the platform.

3.5.1.2 Packet selection

Networking applications are most of the time working on a subset of packets. Indeed, monitoring protects a service or a group of services. Thus, an application is interested only in packets related to these services. For example, an application processing Domain Name Service (DNS) traffic is not interested by Internet Control Message Protocol (ICMP) traffic.

Software applications have the possibility to define filtering rules in order to select only the traffic of interest. Multiple filtering frameworks, like Berkeley Packet Filter (BPF) [MJ93] or netfilter [Net], exist to create the adapted rule set. Rules are compiled and processed by the network stack in order to distribute packets to the corresponding applications.

3.5.1.3 Monitoring applications

Monitoring applications execute the end-user processing. Two types of applications are used in networking:

- Applications working directly on the content of packets. For instance, DPI applications require to go all over the packet payload.
- Applications creating statistics on incoming packets and working on these statistics. For instance, some classification applications use mean values on a flow of packets to determine the nature of this flow.

Results of applications processing have variable usage. Monitoring applications have different purposes from data collector for a global SDN controller to a local firewall. The results are used according to the purpose of the application, either sent to

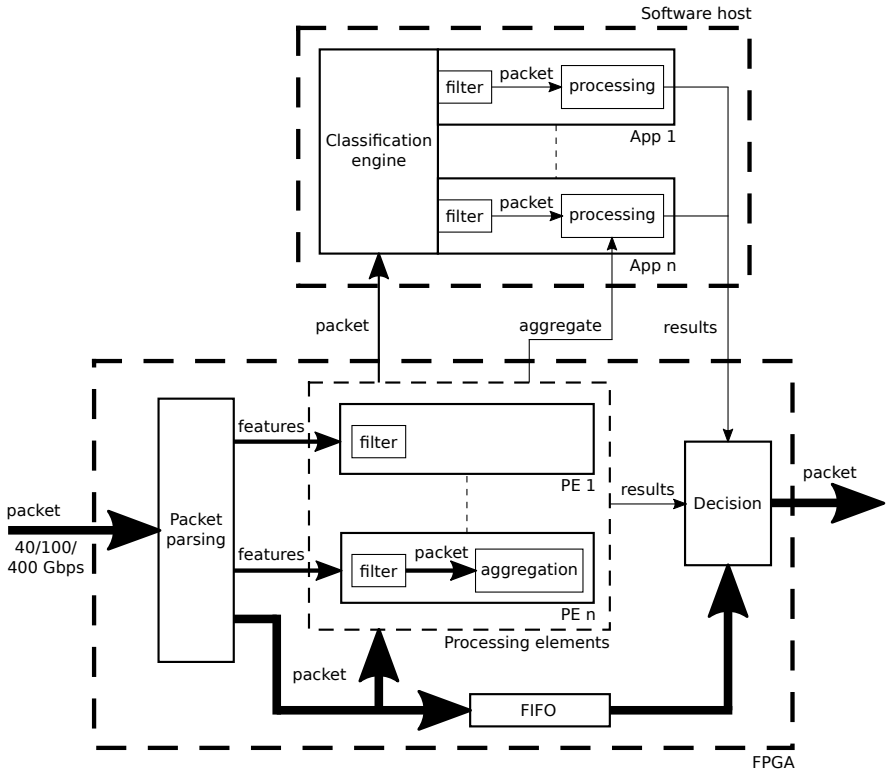


Figure 3.12 – Example of a mixed probe architecture

a master controller (or logged on the machine),or used to execute commands on incoming packets. In Figure 3.11, applications results are used to take a decision on the continuity of the packet on the link.

3.5.2 Accelerated architecture

As shown in section 2.3, the full software architecture has limited packet processing power and is not scalable. Packet throughputs for links over 10 Gbps are too high to be processed only by software engines. The key to increased processing power is to offload packet intensive processing to more specialized architectures.

Figure 3.12 displays a common accelerated architecture on a smart NIC corresponding to the software packet processing of Figure 3.11. Front end traffic is handled by FPGA processing in order to reduce the data bandwidth sent to the software. The CPU then works only on data of interest extracted from the main traffic and corresponding to the needs of applications. Main elements on the packet path have been moved to the FPGA to free CPU processing for software applications. While global processing steps architecture are still present, these steps have received several modifications.

3.5.2.1 Common smart NIC hardware architecture

Packet parsing Hardware implementation of this block ensures that all packets can be processed, even when working with high data rates. Instead of creating a full memory structure to assign the packet features, the packet parser directly extracts the fea-

tures of interest for the following applications. The classical network 5-tuple, IP addresses, TCP/User Datagram Protocol (UDP) port and protocol, is commonly used, however the parser is not limited to these usual elements. These extracted features are then transferred to the corresponding processing elements.

Processing elements Processing elements execute filtering and accelerated processing adapted to the different monitoring applications. Implementations are specialized for the task.

3.5.2.2 Software architecture

The migration of some computing from the software to the hardware reduces the complexity of the software architecture. This improvement has two major impacts.

- Packets are filtered at the smart NIC level. The traffic reduction lowers the communication bandwidth required between the FPGA and the CPU in a controlled way.
- The removal of packet intensive processing reduces the load on the CPU. As a consequence, software applications have the possibility to use a maximum CPU workload to realize complex monitoring adapted to the incoming traffic.

Offloaded applications must be chosen with care. While the core processing is required to be offloaded to the FPGA for performance, some applications do not take advantage of FPGA parallelization or intensively use scarce resource impacting the overall performance. The high flexibility offered by the software part is also useful for networking systems with many updates.

3.5.2.3 Flexible packet processing

The approach used for the presented probe is the common approach used for FPGA platforms. It does not take advantage of the close relation between the smart NIC and the commodity hardware, but it simply uses the FPGA as a static acceleration device with a reconfiguration capability. Despite the flexibility of high level applications, the specialization of generated designs limits the flexibility because of the implementation and reconfiguration times.

A main issue of traffic monitoring is the lack of knowledge about the received packets. Unlike the packet generator presented in the previous section, the traffic composition is not known at design generation time. Applications must be able to adapt the extracted pieces of information in reaction to the incoming traffic.

A novel solution is the consideration of the smart NIC as a dynamic accelerator, configurable at runtime. Figure 3.13 shows modifications brought to the architecture in order to create a reactive design. The adaptation of the probe is done through an advanced collaboration between hardware and software. An innovative paradigm is the consideration of feedback loop mechanism for packet processing. The results obtained with software processing can be used to instantly refined the configuration of the hardware probe. Using this paradigm requires a compatible processing partition. The high configuration importance is different depending on the position on the processing datapath.

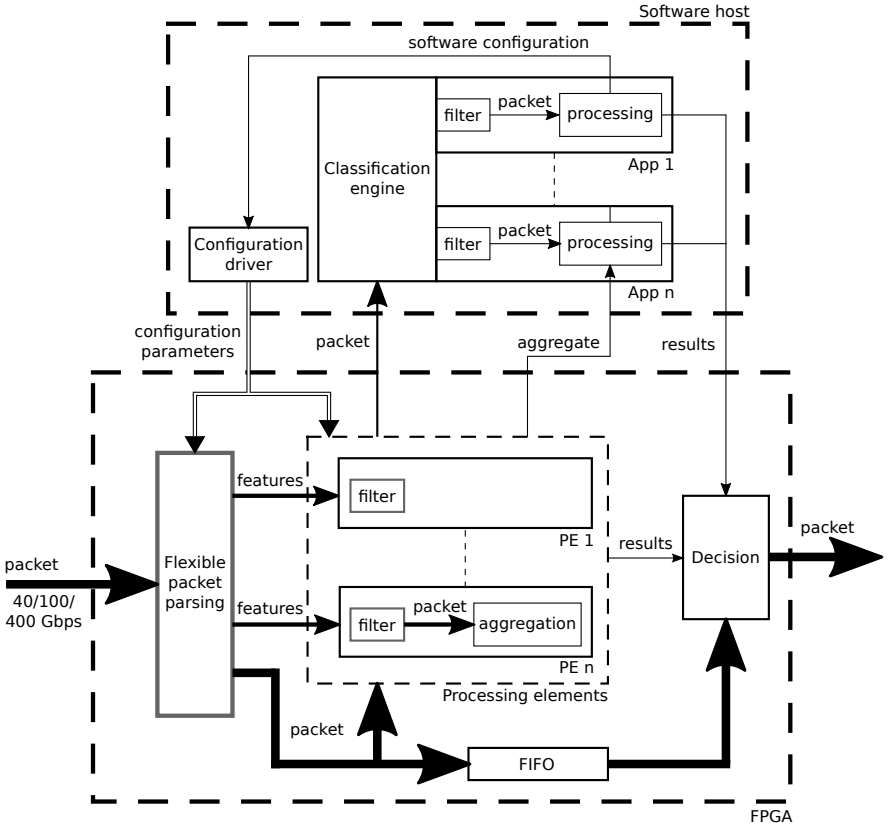


Figure 3.13 – Example of a reactive mixed probe architecture

Packet parsing The packet parser is the front-end of packet processing on the packet path. Hence, stopping the packet parser for reconfiguration removes the distribution of vital information for the following processing elements.

Having a configurable packet parser with a static architecture is the key to follow the dynamic evolution of the traffic. With configuration parameters, packets parsed are set at runtime. Thus, this packet parser can be adapted to the situation enabling agile processing.

Processing elements The flexibility required by this solution aims at promoting the partition of applications with a configurable part on hardware. However, some applications are very specific. As all the processing elements are able to compute packets concurrently, the modification of one element does not impact the operating capabilities of another one. Therefore, partial reconfiguration can be considered. Moreover, this approach fits the upgrade of a software application, which is shut down and restarted, with the reconfiguration of the hardware part when the software is upgraded.

On the contrary, when processing elements are critical for the correct behavior of multiple software applications, they must be configurable at runtime, in order to avoid any unwanted processing interruption. An example would be a hardware classification and filtering engine shared between all applications. A dynamic packet filter allows applications to select incoming packets at runtime. Thus, the processor can apply dynamic complex packet extraction without sub-sampling or being overloaded.

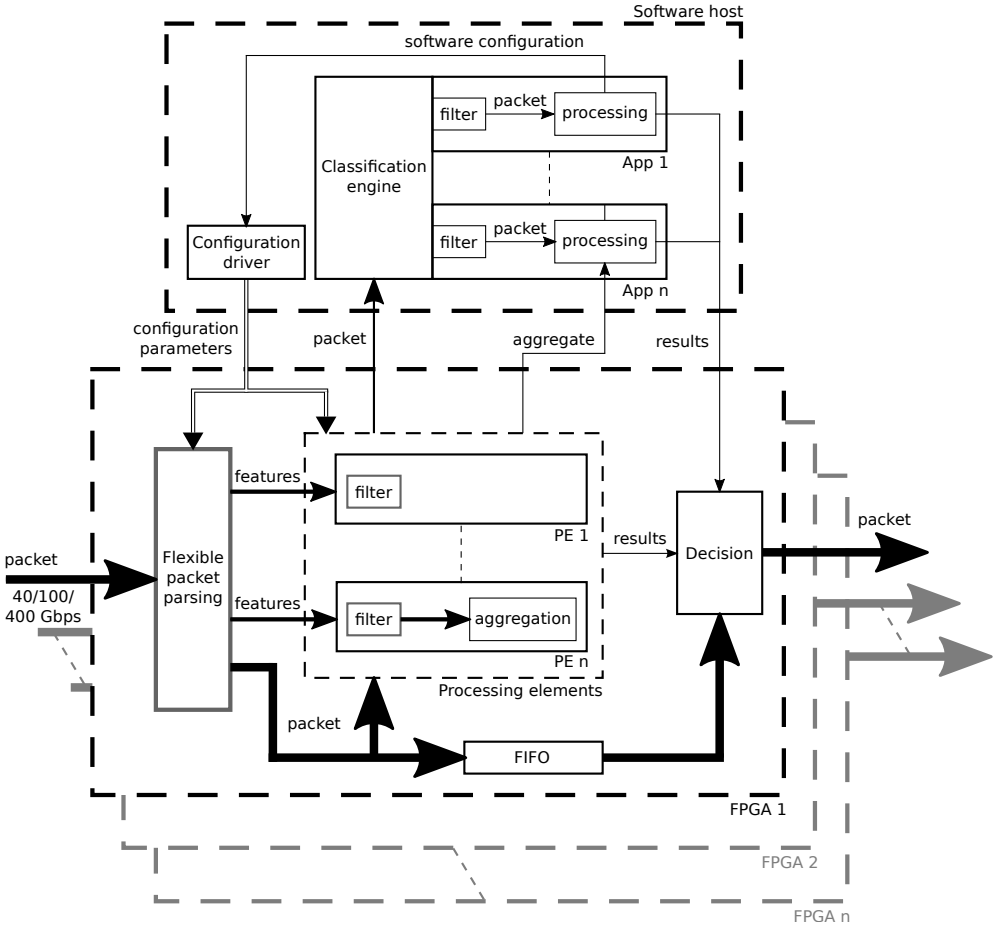


Figure 3.14 – Example of a probe with a software part and multiple smart NICs

3.5.2.4 Opened perspectives

With the feedback loop, the proposed paradigm offers dynamic adaptation of monitoring applications with a full control while keeping the packet processing at line rate. Nonetheless, this solution is compatible with the common usage of specialized designs.

With the right partition and accelerated applications, the software part would be greatly offloaded. With such a performance gain, it is possible to consider that a software element is capable of controlling multiple smart NICs at the same time, as illustrated in Figure 3.14. The resulting probe would have an increased link capacity tending towards hundreds of Gigabits with current smart NICs, and even beyond Terabit. One control software element would allow an extremely fast coordination of the processing on the different smart NICs.

3.6 Conclusion

Combined with commodity hardware, smart NICs improve the processing performance by at least an order of magnitude. Processing power of FPGA, which is closer to the network interfaces than the CPU, offers high performance for the offloaded processing while reducing at the same time the quantity of traffic transferred to the host.

However, current solutions do not take advantage of the proximity of the smart NIC and the CPU. FPGAs are only used as highly specialized accelerators. The flexibility is limited by a long generation time and the processing interruption during the reconfiguration time. In this chapter, the approach considered for this thesis was presented. It is based on the partition of an application between hardware and software for a combined processing. Advantages from both domain can be used to achieve line rate processing with high flexibility.

This approach was illustrated with a traffic generation application. The feed forward paradigm used in the design allows the dynamic modification of replayed packet streams. Thanks to the transmission of configuration parameters, the software part is able to control the replayed packets sent at line rate by the static hardware architecture. With streams modification possibility during generation, this generator is able to use all the capabilities of software packet manipulation to create user-defined reproducible load traffic ideal to test flexible networking equipment. It can be concluded that processing on packet path and decision actions must be separated for a high performance and flexible networking device.

However, this paradigm is not sufficient for all networking applications. In the case of monitoring applications, the received traffic is not known in advance. Packet processing must be adapted in reaction to the traffic composition at one time. A feedback loop must be created between the hardware and the software. Application partition between hardware and software must consider the decomposition of the packet processing steps. An appropriate combined usage of FPGA and CPU is the key to create a monitoring probe capable of adapting its processing to the received packets. The study of this approach has led to two publications [CJ+17c] [CJ+17a]. In the next chapter, I will investigate the application of this concept to a critical function in packet processing: packet parsing.

Chapter 4

A novel flexible packet parser architecture for live monitoring

4.1 Introduction

Smart NICs offer reconfigurable devices capable of offloading packet processing operations from the CPU sustaining high data rates. The closer location of these devices to the network interfaces when compared to the CPU allows the reduction of data transmitted between the CPU and the data link. However, a modification of the common development paradigm is necessary to extend the flexibility and the reaction time of commodity hardware. Instead of having custom-made designs for a specific problem, static and flexible elements are required on the packet datapath to ensure continuous processing of incoming packets.

The packet processing procedure can be decomposed into multiple steps. Packet parsing is the front-end operation executing the decomposition of the packet and the extraction of features of interest from the packet. These features are used to filter the packet if needed, and then transferred to the following processing applications. Instead of being repeated by every processing element, the critical step of packet parsing can be executed one time before all the applications. For the realization of a flexible probe, the packet parser must be able to dynamically extract the features and distribute them among the processing elements. It influences the performance of the whole packet processing design.

This chapter is focused on the design of a hardware architecture for dynamic packet parsing. This architecture is compliant with the paradigm defined in the previous chapter. The resulting design is completely static and ensures packet processing at line rate of high-speed links. Parameters allow the configuration of the processed headers and extracted features. The modification of the packet types processed is done at runtime by an operator on the software host.

For a totally dynamic probe, extracted features from packet headers must be dynamically distributed to the processing elements. A feature must be delivered to one or multiple processing elements on demand from end-users applications. As datapaths on FPGAs are fixed once the chip is configured, a straightforward solution is the extraction of one feature for each processing input. This oversized packet parser is then able to duplicate the extraction of a specific feature for multiple processing elements. But,

extraction duplication is a costly process in terms of resource consumption limiting the total number of possible features to extract. A better solution is investigated to offer a good compromise between space occupation and flexibility.

The latter section of this chapter studies optimized data distribution through the usage of interconnection network. After formulating the requirements of such a network for the proposed solution, a simple interconnection network is proposed. Perspectives are given with the study of Clos networks for a more efficient interconnection.

4.2 High performance and flexible packet parsing

4.2.1 Packet parsing challenge

Packet parsing is the first step responsible for detecting and extracting required features in a packet, often in protocol headers. A packet parser supplies inputs from packets to next operations in the processing chain.

Although headers to process are known, incoming packet composition is not. Network packet construction flexibility allows to create as many different packets as needed. Therefore, a high performance flexible parser must face multiple challenges [Gib+13].

Line-rate throughput Incoming packets must be processed at link speed. The front-end location of a packet parser is critical for accurate processing. Hardware implementation has to ensure features are produced without packet loss.

Headers sequential dependency A network packet is sequentially built. Information identifying a header is contained in the previous header, corresponding to the protocol layer directly under this protocol. The decapsulation task is inevitably sequential, processing one header after an other.

Protocols and headers heterogeneity With the openness of packet construction, numerous communication protocols co-exist on the same layer level. In addition, more recent encapsulations are more complex than the standard OSI model. The packet parser must be able to process header formats for different protocols found at different places in the packet.

Header format programmability Common network applications use a fixed parser extracting a constant feature set. These features are source and destination IP addresses, source and destination UDP or TCP ports, and the transport protocol. Despite rich information, this 5-tuple limits the network analysis to track these flows. However, networking algorithms could need complementary protocol information on packets. Layer 7 protocol header fields bring additional information on the final application.

Moreover, monitored protocols format is subject to change. Some network operators create their own custom protocol to allow maintenance or identification traffic. New usage leads to the creation of new protocols.

For these reasons, formats used by the packet parser have to be generic. Configuration of the decapsulation is primordial to have flexible and reactive packet processing.

Feature distribution to processing elements The packet parser must assign the right feature sets to the right processing elements. Feature sets are considered completely independent. One feature may be in multiple sets.

4.2.2 Existing parsers limitations

The evolution of network traffic places packet parsers at the center of networking devices. A few solutions tried to solve the high performance generic packet parsing problem.

Pus *et al.* [PKK14] presented a pipelined design for packet parsing. This architecture extracts fixed features from a fixed headers set at a throughput of 100 Gbps. It is the basis for an automatic high level parser generation tool [BPK16]. The presented P4-to-VHDL parser generator uses a P4 description to create an HDL packet parser design. The resulting architecture is parsing packets, as the original, at 100 Gbps after synthesis.

The successive development of packet parsing languages for software assisted development, such as G [Bre09], PP [AB11] and PX [BJ14] has led to the development of the parser for the full SDNet product [Xila]. The packet parsing [AB11] pipeline is able to process traffic beyond 100 Gbps. Parsing cores can be reconfigured with partial reconfiguration to modify possible packet features extracted at a processing stage.

Bitar *et al.* [BAB15] proposed a NoC-Enhanced FPGA packet parser. A NoC hosts a header parser on each node and can reconfigure routes between parsers. Based on synthesis results, they extrapolate a design working at 400 Gbps with 512-byte packets. Although the ultimate purpose is to hold dynamic parser configuration with a partial reconfiguration of router nodes, this architecture is not developed. The high parallelism of the NoC is an interesting idea for packet distribution to the different processing elements.

Despite good performance, these two solutions fail to answer multiple requirements:

- full or partial FPGA reconfiguration is necessary,
- features sent to the processing elements are not dynamically selected.

4.2.3 Feature extraction requirements

Presented parser architectures are focused on the specialization of parser engines on a set of known protocols. The flexibility is left to the reconfiguration possibility of the FPGA. However, packet structure is designed to be reversible in a standardized way for the decapsulation by any networking device.

The decapsulation of a packet relies on the knowledge that, for each layer, the protocol header contains two pieces of information:

- its current length or fixed length,
- the identification of the type of the next header, called the Service Access Point (SAP).

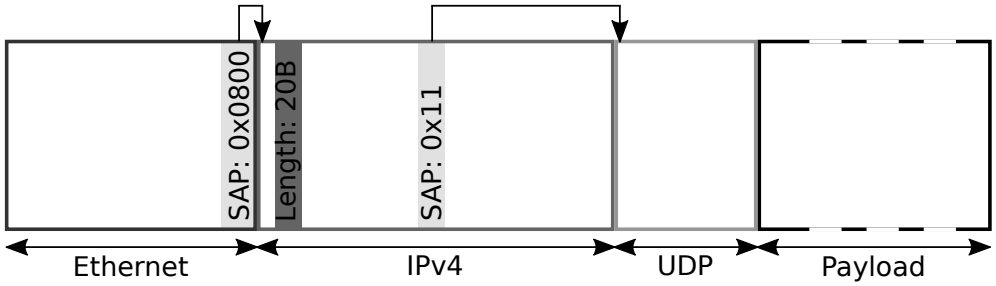
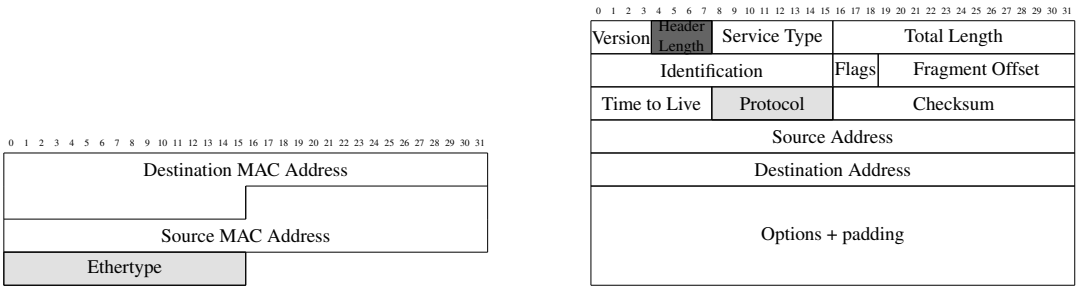
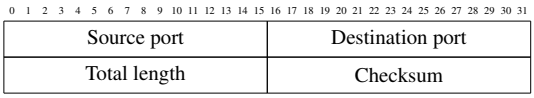


Figure 4.1 – Packet decapsulation process



(a) Ethernet

(b) IPv4



(c) UDP

Figure 4.2 – Header specifications for different protocols

Figure 4.1 illustrates the decapsulation to obtain the UDP header inside a packet. Finding a specific header requires to process all the previous headers.

Interpretation of the succession of raw bytes inside each header is done according to the header specification of the protocol. Figure 4.2 shows header specifications for Ethernet (4.2a), IPv4 (4.2b) and UDP (4.2c) protocols. A header feature, one field or aggregate of multiple successive fields, begins and ends at fixed offset from the beginning of the header. For instance, the UDP destination field is located between bits of and an offset of the previously found UDP header, for a size of 16 bits.

In conclusion, a parser must execute two actions to extract a specific feature from a packet:

- the target header must be located,
- the feature must be extracted from the corresponding header with the definition of its lower and upper offsets.

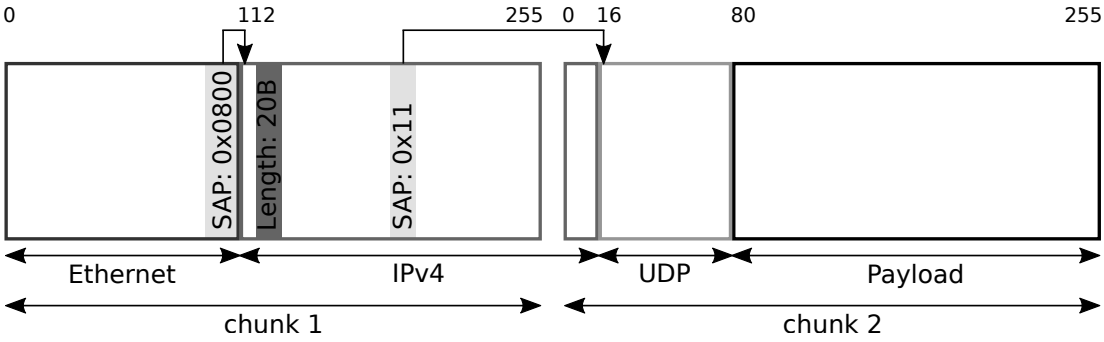


Figure 4.3 – Packet chunks example for a UDP 64-byte packet on a 256-bit datapath width

4.3 Packet parser architecture

4.3.1 Global architecture

4.3.1.1 Chunk processing

Previous example in Figure 4.1 shows the processing of a packet for a continuous stream of bytes. The transmission of a packet in chunks on the FPGA datapath make the packet not continuous. Figure 4.3 displays the decomposition in two chunks of the previous 64-byte Ethernet packet on a 256-bit wide datapath. It can be observed that the IPv4 header is broken in two parts, one in each chunk. Moreover, inside the IPv4 header, the IP destination address field is cut in half between the two chunks.

With variable lengths, packet headers have the possibility to span on multiple chunks. While this is also valid for features, header fields of interest are limited in length. For instance, common used longer fields are IPv4 and IPv6 addresses, which are 32 bits and 128 bits. Thus, it is safe to assume that a feature will not be on more than two consecutive chunks. In any case, extraction of wider fields can be executed with the extraction of multiple consecutive smaller fields.

Extraction actions must be adapted to FPGA constraints. Translated requirements are:

- headers may be located in every chunks,
- a feature is extracted from a header with offsets definition,
- a feature is limited in size and is contained by at most two consecutive chunks.

4.3.1.2 Processing steps

From previous requirements definition, it is possible to consider the feature extraction as a 2-step operation. Figure 4.4 illustrates this operation. The header parser handles the location of the different headers inside the chunks of incoming packets. The feature selector extracts the features from the packet chunks based on headers location. Both parts are working on configuration parameters sent from the software host.

This separation is crucial for a generic and flexible design. Indeed, headers are composed of a different number of fields. The number of features to extract for each

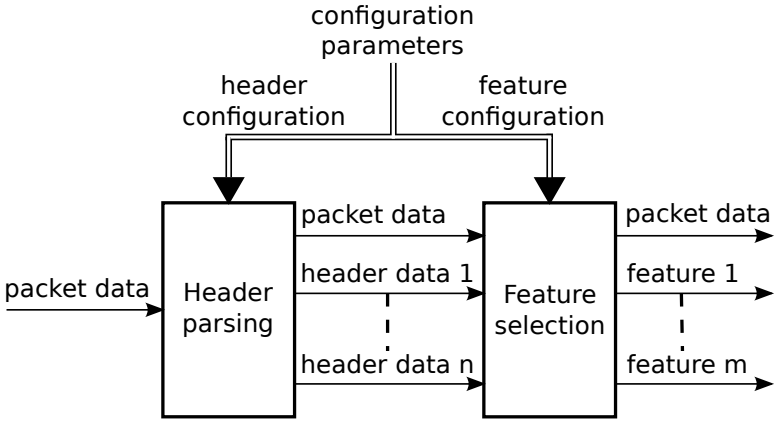


Figure 4.4 – Packet parser global architecture

protocol is not known *a priori*. The number of features allocated to each protocol can be adapted according to the needs with a global pool of extracted features.

This allows to share a global pool of extracted features. The number allocated to each protocol can be configured and adapted to the needs.

Moreover, feature results are outputted at the same time, synchronized on the end of the packet. The working data path is detached from the chunk transmission datapath. It is equivalent for the following processing units to work at packet arrival rate, so at a lower frequency than line frequency. For instance, the packet parser is able to process at maximum 60 Mpps of 64-byte packets with a clock frequency of 180 MHz and a datapath of 256 bits. Therefore, a clock frequency of at most 60 MHz is necessary for the following processing.

4.3.2 Header parsing

4.3.2.1 Architecture

Due to data encapsulation in network packets, protocol headers need to be processed in encapsulation order. It is therefore not possible to concurrently handle all headers of the packet. To achieve high processing throughputs, the sequential processing of consecutive headers must be pipelined. Figure 4.5 shows the pipelined architecture. Each stage takes care of one layer of encapsulation.

One header parsing stage identifies the protocol location of the associated layer in chunks and determines the following header type if possible. On the FPGA, the type of protocol corresponds to the unique identifier of a protocol during the packet parsing process. The next header type and the current offset in the packet are transmitted to the header parser stage to continue the decapsulation process. In this way, the different protocols are unstacked from the packet.

A synchronization block delays header information of each stage waiting for the rest of header processing. As a consequence, all header data are concurrently transferred for feature selection at the same time as the corresponding packet chunks.

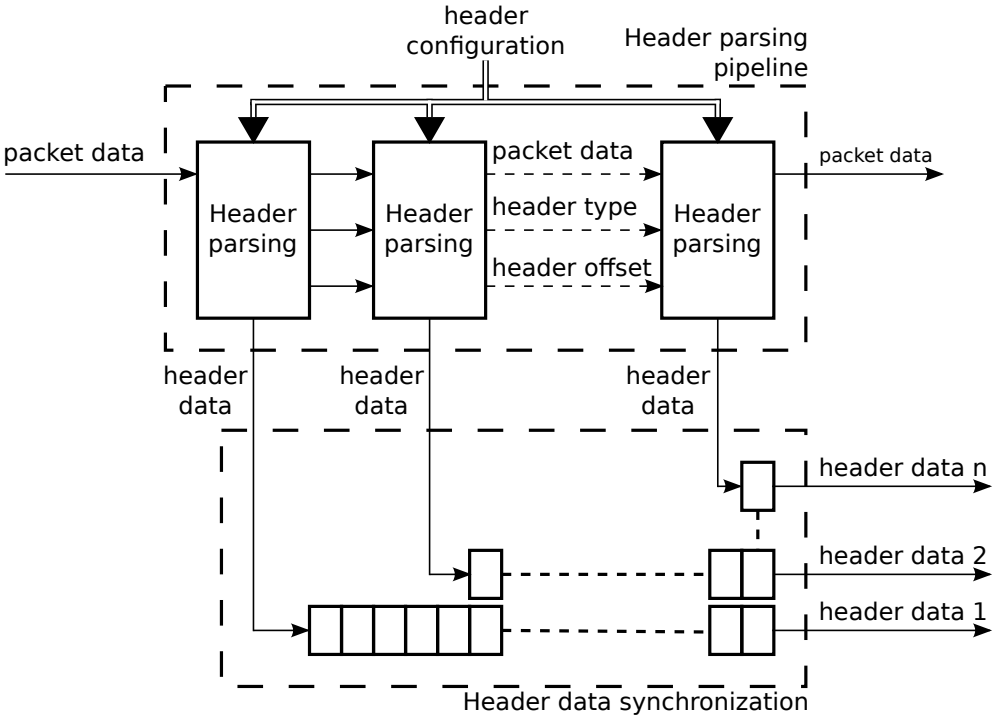


Figure 4.5 – Header parser pipeline

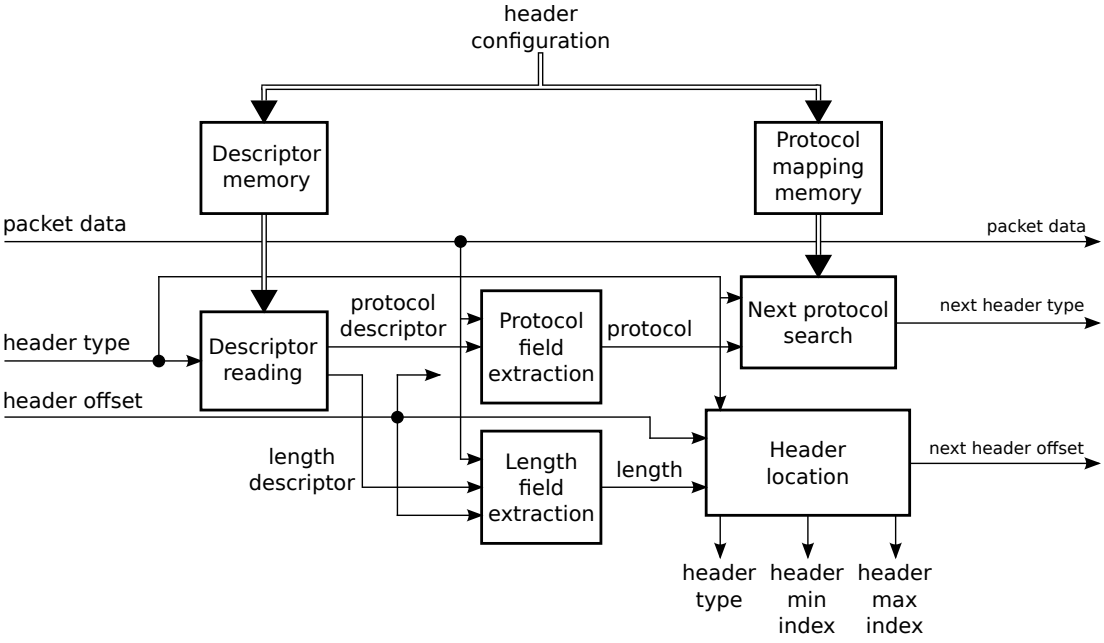


Figure 4.6 – Packet parser header parsing

4.3.2.2 Header parsing module

This generic module decapsulates one header of the packet. If the current header type has no associated configuration, packet header processing is finished. Figure 4.6 summarizes the header parser module. Configuration is held in two Content-Addressable Memories (CAMs). The first memory contains a protocol descriptor used to process the current header. The second memory contains the translation to get the next protocol from the extracted value inside the current header.

The first step in the module is to fetch the descriptor corresponding to the type given by the previous block. This descriptor is composed of configuration parameters for the selection of the current protocol header:

- length field low and high offsets,
- protocol field low and high offsets.

The search key in the memory is the identifier of the header. Length and protocol are extracted from the current header with their offsets as shown later in subsection 4.3.3.2. Special combinations of offset values allow determining if the header has a fixed length or is the last protocol in the packet.

Length and protocol extracted values are then used to process the current header. The location of the header is found by computing the min and max indexes in each packet data chunk. The protocol field value is combined with the current protocol id in order to find the next header protocol id. This protocol is processed by the next block in the pipeline.

The global processing pipeline is configured to input one chunk per cycle in order to target the link throughput. The header parser block must be fully pipelined to sustain the chunk rate. While it is not shown for clarity, used data and outputs are registered to be synchronized on chunks of the datapath. An other advantage of the pipelined processing is a controlled and constant latency. One header parser takes eight clock cycles to process incoming chunks.

Referring to the example in Figure 4.1, configuration parameters values stored in the second stage header parser to decapsulate IPv4 header and get the UDP header are:

- IPv4 length field low offset : 4 bits
- IPv4 length field high offset : 8 bits
- IPv4 protocol field low offset : 72 bits
- IPv4 protocol field high offset : 80 bits
- UDP protocol field value : 6

4.3.2.3 Parse graph dimensions

A common way to represent the header sequence inside a packet is to use a parse graph. The parse graph associated with a device describes all the packets that can be recognized by this device. As the design does not changed once configured on the FPGA, the number of different parse graphs which can be mapped on the running design is limited by parameters of the architecture set at generation time. These parameters are:

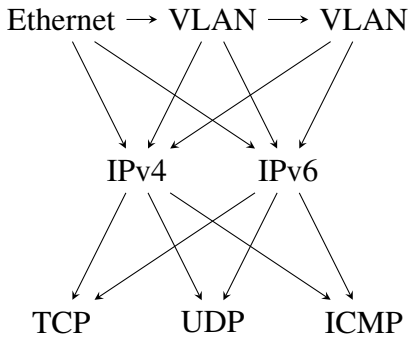


Figure 4.7 – Example parse graph

- the number of consecutive header parser modules inside the pipeline,
- the size of the descriptor memory of a header parser module,
- the size of the mapping memory of a header parser module.

Figure 4.7 displays a parse graph presented in [Gib+13]. Figure 4.8 is an equivalent staged representation of the parse graph more adapted to the architecture. Protocol headers have been duplicated when necessary. Each stage of the graph corresponds to headers that must be processed at the same stage of the header parsing pipeline. The first stage has always Ethernet as a unique protocol because it is linked to the physical layer.

From this representation, it is easy to deduce the impact of the generation parameters. The number of parser modules defines the depth of a path from the root, so the number of possible encapsulated headers. The size of the descriptor memory defines the number of different protocols that can be processed at a given distance from the root, so the number of possible headers at a given layer of the packet. The size of the mapping memory defines the total number of possible transitions between two stages.

4.3.3 Feature selection

4.3.3.1 Architecture

After header parsing, the sequential dependency between headers is removed. Information on every processed header is sufficient to locate the associated header. Moreover, header fields are not dependent on information inside packets, but can be determined with fixed offsets from header beginning. It is then possible to process all features in parallel. Parallel feature processing minimizes the overall latency overhead.

Figure 4.9 describes the parallel feature extraction architecture. Features from the same header use offsets from the same part of the packet. Thus, using the same block to extract all these features reduces the FPGA resource consumption by sharing similar resources. The number of features extracted by each selector must be set with attention to avoid overhead. This parameter is configurable at generation time, as well as features maximal width. For instance, IPv6 addresses are the biggest features to extract with 128 bits. It is four features of 32 bits. Most protocols have at least four fields of interest, so it is a good compromise.

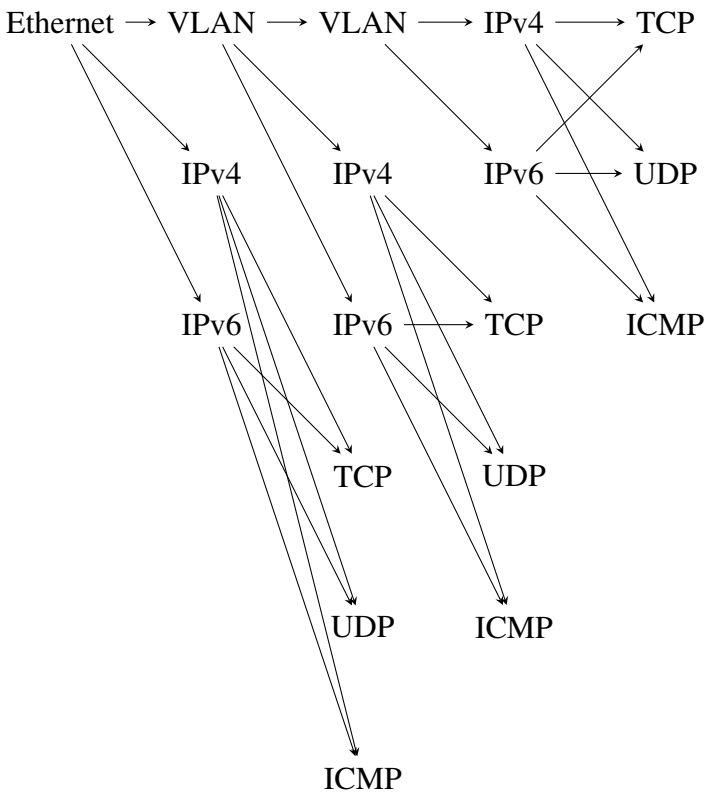


Figure 4.8 – Transformed parse graph

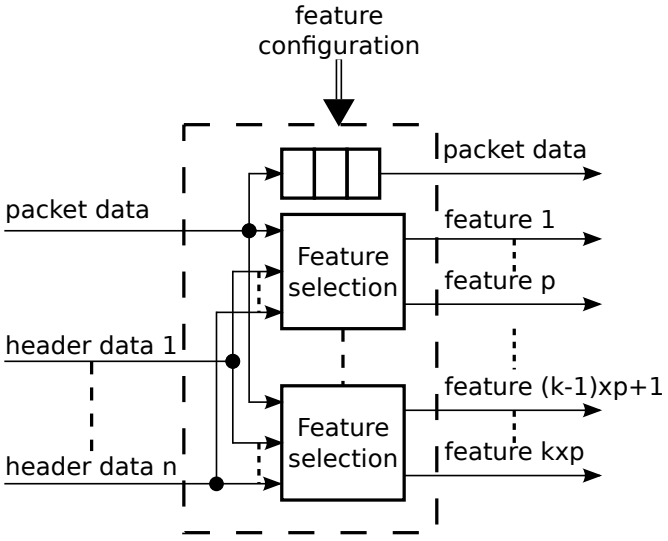


Figure 4.9 – Feature selection architecture

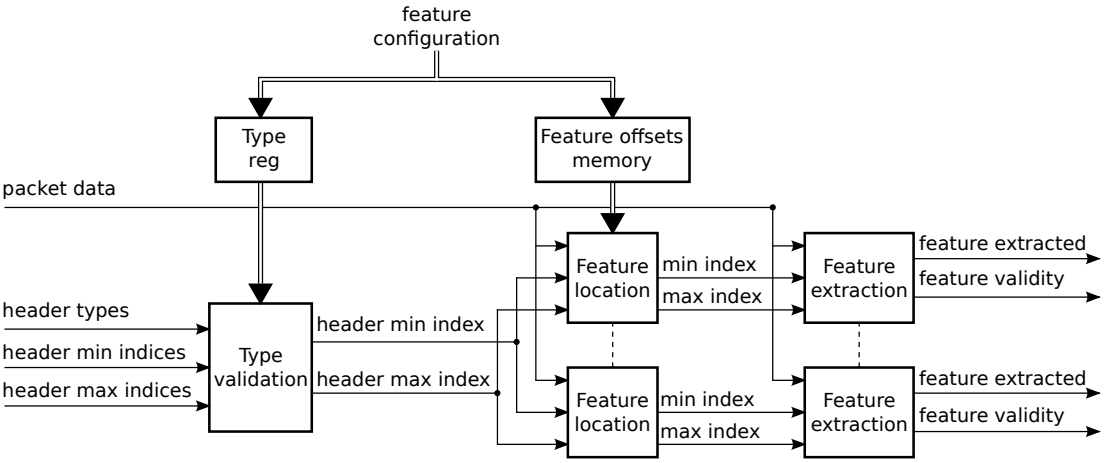


Figure 4.10 – Feature selection

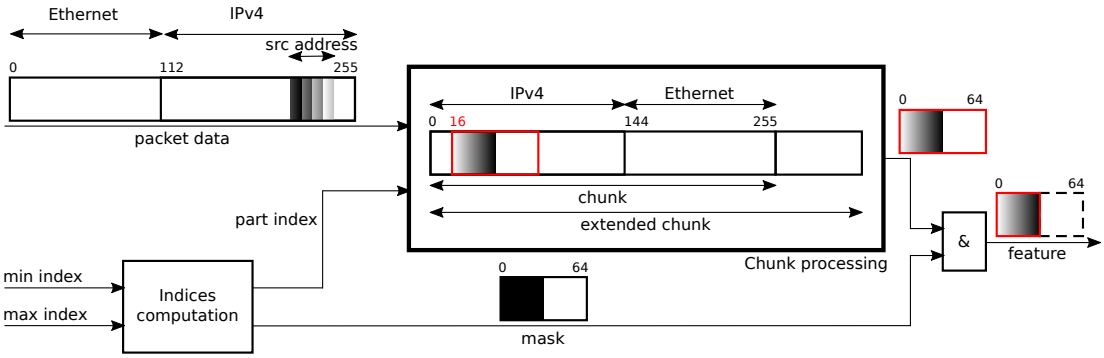


Figure 4.11 – Extraction of IPv4 source address from 256-bit chunk

4.3.3.2 Selection component

The selection module extracts header fields according to their definition in the protocol header specification. One selector is able to extract several features from a header. Figure 4.10 presents the different steps leading to features extraction. This block relies on two configuration parameters set dynamically in memory. The header type determines which protocol header is processed by the block. Minimum and maximum offsets of features are used to locate features inside the header.

The first step is the selection of proper header information. All header location data are received from the header parser. The configured type is used to get header data corresponding to the wanted protocol. In this way, a selector can process the same header located at different layer level in packets.

Header location indices are then combined with feature offsets to compute the location indices of the associated feature. As some features are not aligned on byte values, the extraction of features requires a bit precision. Figure 4.11 summarizes the feature extraction procedure on the IPv4 source address field for 256-bit packet chunks and a feature max width of 64 bits.

To correctly align data on receiving feature register, the big-endian byte ordering of network data is transformed in little-endian ordering inside each chunk. Then, an extended chunk is created to provide all the possible feature alignments. Calculated

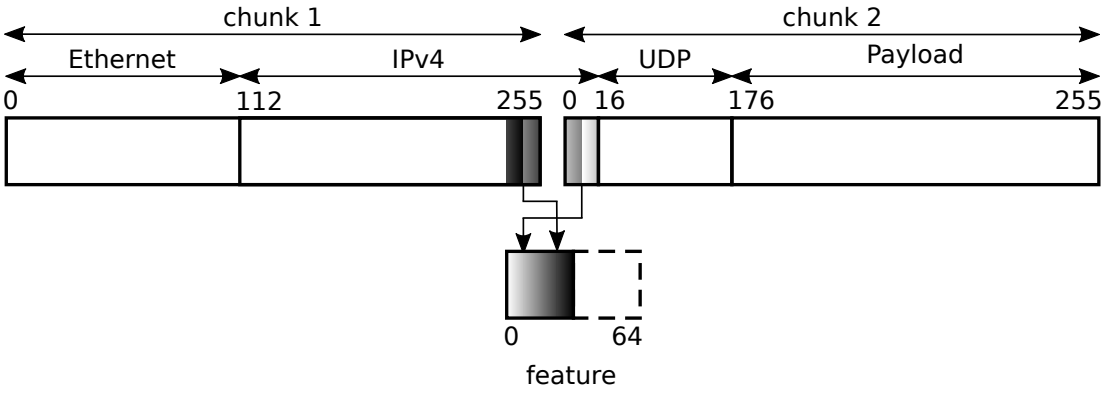


Figure 4.12 – Multi-chunk extraction of IPv4 destination address from 256-bit chunks

indices are used to select data with the right alignment and a mask of the size of the feature to extract. The extracted feature is finally obtained by making a bitwise AND between data extracted from the chunk and the mask.

A limit case is when a feature is spread over two chunks. It must be extracted in two times from the two consecutive chunks. Endianness of network data implies that the first part contains the most significant bits. When all the parts of the feature are processed, the feature is considered extracted. Figure 4.12 demonstrates this process with the IPv4 destination address field. All the extracted features are sent at the same time as the packet last chunk. If a feature is not completely extracted, it is considered as not valid.

For example, to get UDP source destination port field, the begin offset, 16 bits, and the end offset, 31 bits, of the field are needed. Referring to the example in Figure 4.1, configuration parameters values stored to get the UDP destination port field are:

- protocol type associated to UDP protocol, for instance 3,
- low offset from the beginning of UDP header : 16 bits,
- high offset from the beginning of UDP header : 31 bits.

As for the header parsers, processing is fully pipelined in order to sustain the throughput. The latency of a single feature selector is 6 clock cycles.

4.3.4 Architecture results

4.3.4.1 Flexible and reactive architecture

The full processing architecture is based on abstract parameters. As long as these configuration parameters are properly set, the packet parser is able to decapsulate any protocol header based on length and protocol fields. Any feature at a fixed offset from the header head can be extracted.

As parameter values are stored in memory, they can be defined from outside the design, and even software without any consequence for the running processing. With a well-designed API, it is possible to translate end-user specifications to configuration parameters for the architecture. This runtime modification of parameters gives the packet parser high flexibility and reactivity.

Unlike cutting edge parsing solutions, this architecture is configurable at runtime without stopping probe processing, thanks to simple one-cycle read and write operations in integrated SRAM memory. Since supported data rate is only determined by data width and clock frequency on the FPGA, the flexible packet parser can be programmed to support high data rates alongside being highly flexible.

4.3.4.2 Packet parser dimensions

The design is able to dynamically process incoming packets, but size parameters of the hardware architecture must be set at design time. These parameters have influence on the capabilities of the design:

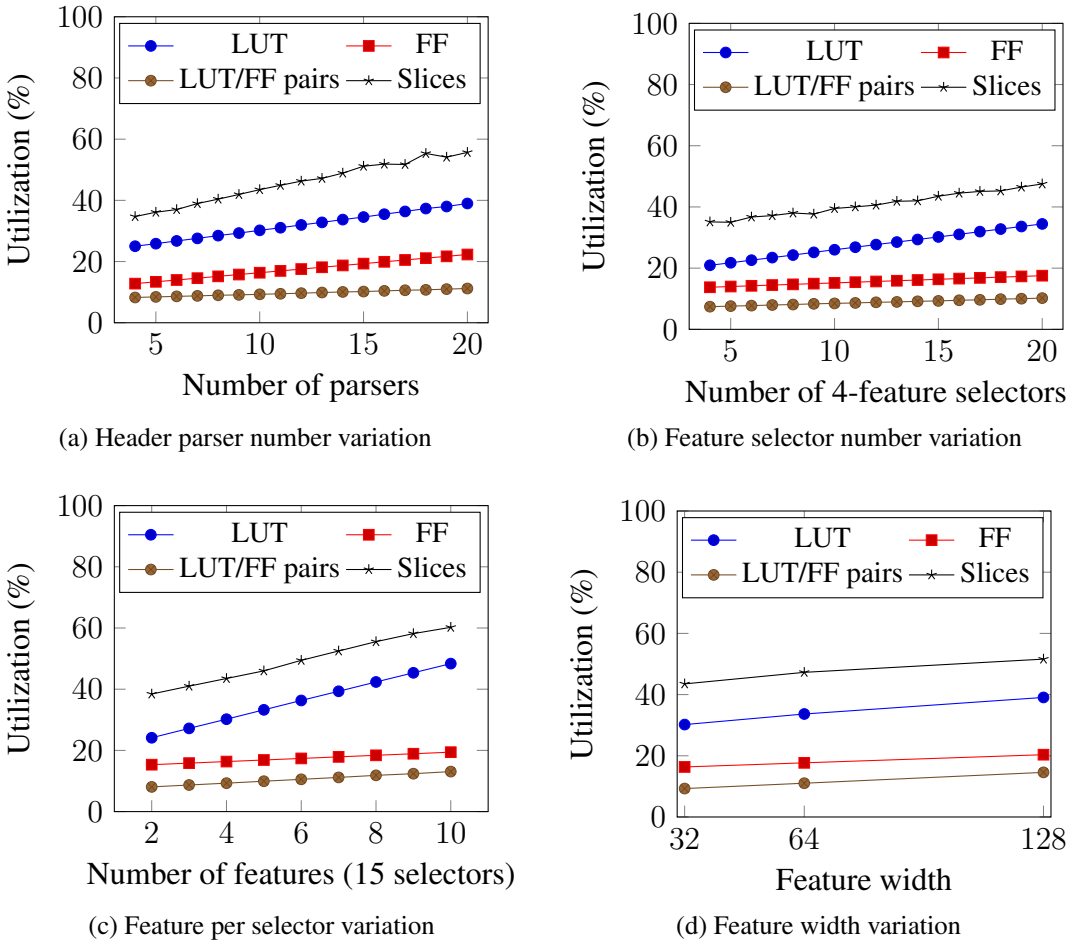
- the encapsulation depths of headers inside packets,
- the number of protocols processed at each stage,
- the number of possible features to extract.

Increasing these parameters to process a wider diversity of packets increases the size of the design. Moreover, a design without protocol specialization brings flexibility but could lead to resources over-utilization. For complex traffic, this drawback can be mitigated thanks to the possibility to share resources between protocols.

Hardware resources are not infinite on an FPGA. As a front-end processing element, it is interesting to study the impact of the packet parser on resource consumption. Spare resources are important to have the possibility to set processing elements on the FPGA. Finding a good compromise for design sizing is crucial to ensure processing a maximum of packets at hardware level. Specific packets with unanticipated composition must be transferred to the software part for further processing.

The study of design parameters impact is done with the variation of these parameters around a base reference design. The reference packet of network devices is a classic TCP/IP packet. Parameters of the base design are calculated to have a test design capable of processing parse graphs of all the use cases proposed in [Gib+13]. The values are set high on purpose to highlight the impact of design variation. To avoid a too large oversize of the design during the tests, the architecture is not capable of processing the union of all the use cases used in [Gib+13] at the same time. However, it is worth noting that this unnecessary large parse graph is set for a fixed parser unable to be configured for a specific use case unlike the architecture proposed in this manuscript. The base design is then composed of:

- 10 header parsers,
- each header parser has local memories up to 32 values which is largely enough to process all the 28 protocol nodes of the wider parse graph of [Gib+13],
- 15 feature selectors,
- 4 feature extracted per selector because most of the considered protocols have at least 4 fields,
- feature width of 32 bits to be able to extract the .



(a) Header parser number variation

(b) Feature selector number variation

(c) Feature per selector variation

(d) Feature width variation

Figure 4.13 – Packet parser relative resource utilization on XC7VX690T

Figure 4.13 summarizes the impact of the variation of different configurations on the consumption of FPGA resources. When not variating, base values are used. These results correspond to a complete and successful implementation process with Vivado 2016.4 tool with NetFPGA interfaces overheads on Virtex 7 XC7VX690T for a 180 MHz clock.

It can be observed that resource consumption is a linear function of the different parameters. The resource consumption of a given configuration can be easily computed. In addition, the widest full design uses less than 60 % of FPGA's resource. With the linearity of the consumption, it can be assumed that reasonable designs would take less than 30 % of resources, which lets plenty of space available to implement further preprocessing.

It is worth noting that the design does not efficiently use the Look-Up Table (LUT) and Flip-Flop (FF) pairs on the FPGA. The LUT consumption is far more important than the FF consumption and increases more with the size of the design. This is probably mostly due to the necessity of routing large datapaths. This low efficiency is the limiting factor of the design scalability.

One key component of the design not shown on graphs is the processing latency. The design latency is another key component. Each header parser has a latency of

	proposed	[AB11]	[PKK14]
Datapath width	512	1024	2048
Raw T'put (Gbps)	160	325	333
Clock period (ns)	3.2	3.154	n.c.
Latency (ns)	96	309	25.9
Slices (% FPGA)	10	12.4	3.9

Table 4.1 – Packet parser solutions on XC7VXH870T

8 cycles and each selector has a latency of 6 cycles. Parsers are designed as serial pipelines, so the global latency of header parsing is the addition of each latency. On the opposite, selectors are parallel, so the global latency of feature selection is 6 cycles. The packet parser processing latency linearly increases with the number of header parsers, from 38 to 166 clock cycles in the examples. The latency of the extreme header parser case corresponds to 929.6 ns with a 180 MHz frequency. This latency to the packet path is negligible when compared to the transfer latency to the host of hundredth of microseconds [Han+10].

In Figure 4.13b and Figure 4.13c, it can be observed that the consumption of FPGA's resources increases with the number of extracted features. While feature duplication is a possible solution to distribute features among the different processing elements, resource consumption is a strict limitation to the number of features that can be extracted. Indeed, for 150 extracted features, the design has already a slice occupation of more than 60 %. The study of an interconnection network is necessary to connect more processing elements outputs.

4.3.4.3 Packet parser architectures comparison

Despite the design singularity, comparing it with other generic architectures gives a good overview of the performance. Table 4.1 shows the comparison between the proposed solution with a 512-bit wide datapath and other packet parser designs on Virtex 7 XC7VXH870T. Results are given for a *TCPandIP4andIP6* specialized parser after synthesis for the proposed design and [PKK14], and after implementation for [AB11]. These solutions are focused on feature extraction, not on the distribution of these features to the following processing elements.

In order to provide a fair comparison between the related works and the proposed architecture, the parser was configured with 3 header parsers, 3 feature selectors and 64-bit feature width. This allows parsing the same protocols, with all the important features with spare ones.

The different solutions are compared in terms of throughput, latency, and resource usage. Flexibility is also discussed in terms of expected agility. Even if this is not measurable, by studying the architecture and its integration, a good idea of its adaptability can be obtained.

Despite its high flexibility, the proposed approach has a comparable resource usage with [AB11], but greatly improves the latency and the flexibility. Comparison with [PKK14] is difficult on a fair basis, since this solution is very dedicated, with almost no flexibility. Resource usage is 2.5 times as large for the proposed solution, but it

stays acceptable. Latency is 4 times as large but, once again, remains acceptable at 100 ns, which means a buffering of only 4 kbits. This comparison is done on a simple example and more complex and diverse traffic could lead to compensate the overhead on resource usage by resource sharing between different protocols.

This is compensated by the clear advantage of flexibility, and by the independence from proprietary tools provided by the approach. Adding a new protocol can be done in software only, using the API, while the compared solutions require a new synthesis and configuration of the FPGA. This requires using proprietary tools and stopping current processing on the probe. [AB11] achieves some level of flexibility, by allowing upgrades with partial reconfiguration by SDNet solution [Xila]. However, partial reconfiguration requires to reserve space on FPGA for the wider design and imposes strong routing constraints, impacting final performances. It also takes longer than a simple change in parameters.

Another interesting difference lies in the datapath. The compared solutions use a wider datapath. The width of the datapath is linked to the parallelism level of the architecture. The higher it is, the higher the expected throughput is. However, it also creates a constraint on the minimal packet size. If 64-byte packets must be processed at line rate , the datapath cannot be wider than 64 bytes (512 bits). Saturating the link with 64-byte packets on the compared solution will lead to packet loss. Higher parallelism also means more duplication in the resources, which might limit the preprocessing.

4.4 Interconnection architecture

4.4.1 Interconnection network definition

4.4.1.1 Flexibility constraints

Even if the packet parser dynamically extracts features, one or more processing elements may require the same feature. One way to supply the feature to each processing element is to duplicate the extraction of this feature. However, this reduces the total number of available analyzers for different features. It is possible to make sure to have more analyzers than needed, but it is costly in terms of FPGA resources, especially if a large number of processing elements is required. Another way is the dynamic distribution of features with an interconnection element.

Figure 4.14 shows an example of feature distribution between multiple processing elements. To complete the flexible packet parser, the interconnection element must respect multiple constraints:

- one feature must be sent to any processing element,
- one feature can be supplied to multiple elements at the same time,
- the distribution of features must be programmable,
- the distribution is considered fixed between two configurations,
- extracted features synchronization must be kept.

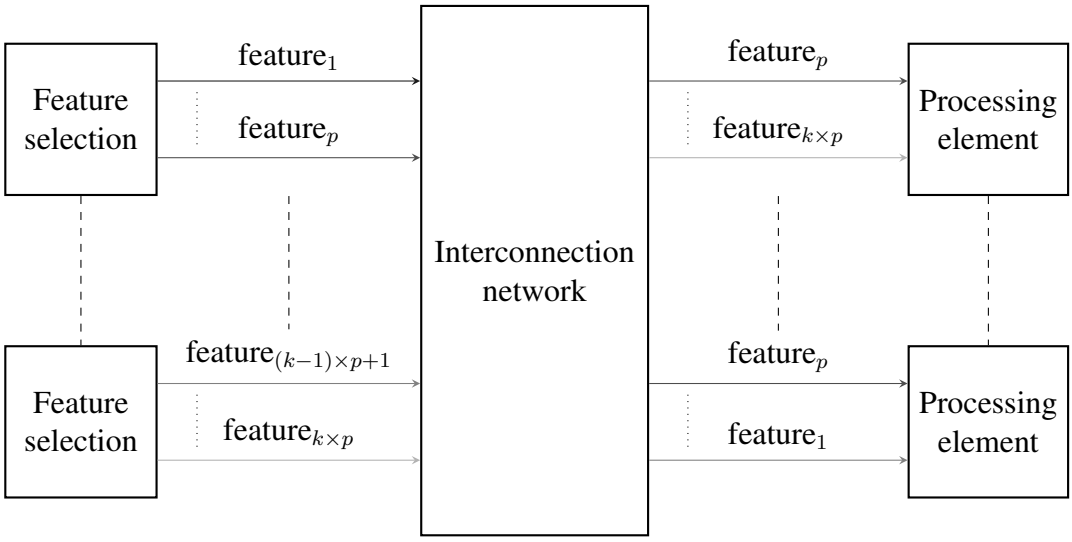
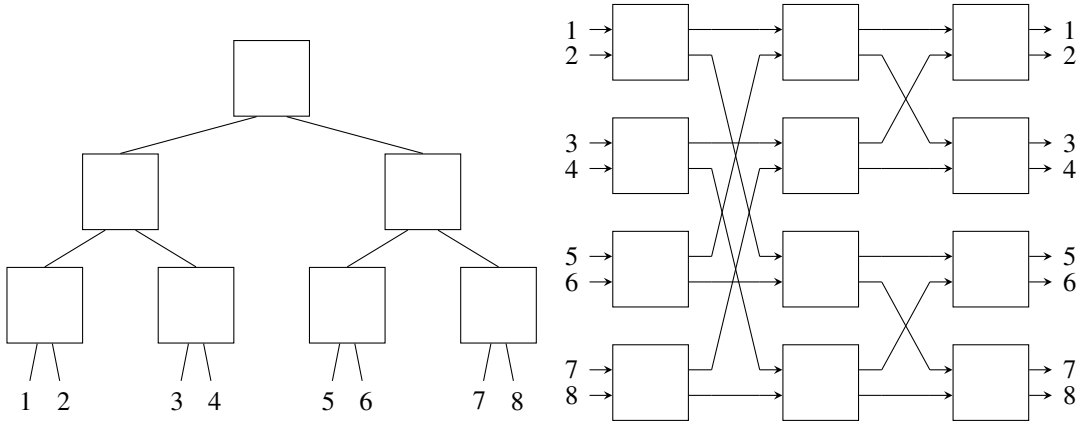


Figure 4.14 – Feature distribution example



(a) Tree (b) Butterfly

Figure 4.15 – Network topology examples

4.4.1.2 Network properties

An interconnection network is used to simultaneously connect input nodes to output nodes. It is composed of a collection of smaller connection devices with local routing tables in order to maintain paths between inputs and outputs. These routers have only an interconnection purpose, reducing the complexity and the resource cost compared to NoCs. Permutations possibilities [CP15] of the network offer a coprocessor for special computations, as for instance sorting algorithms [CP17]. Some properties of networks [Fen81] influence the interconnection capabilities. These properties must be considered to select a network adapted to the needs.

Topology The topology is the organization of the router inside the network. Depending on the topology, it is possible to create one-sided, where outputs are the same as inputs, or two-sided networks. Figure 4.15 shows two different network topologies. It

can be observed that the topology impacts the communication latency. Figure 4.15a displays a tree network with 8 inputs and outputs where the latency is dependent of the positions of the transmitter and the receiver. In Figure 4.15b is represented a (8x8) butterfly network. In this case, the transmission latency has a constant value.

Symmetry It is important to consider the symmetry of a network. Some topology can only be used to create symmetric networks with the same amount of inputs and outputs while it is possible with others to have asymmetric networks with a different number of inputs and outputs.

Switching mechanism The switching mechanism of the network determines the way information is transferred inside the network.

Packet switching distributes the routing across all the routers of the network. Transmitted data is embedded in packets where the header contains the routing information. The same routing algorithm is used by all the routers to transfer packets through the network using the header overhead. The header information can be for instance the source or the destination of the transmission. Depending on the routing scheme, a link between two routers can be consecutively used for packets of different connections.

With circuit switching, paths from inputs to outputs are reserved by previous negotiations. In the opposite of packet switching, this path is dedicated to the current communication and can not be used for other transmissions. A global controller takes care of the selection of the different connections. Between two configurations, the network is considered static.

Both of these methods have an overhead compared to a simple communication. Packet switching adds information to transmit alongside data and the local routers are more complex. Circuit switching has a computation overhead before the transmission.

Blocking When two communication paths have to use the same section between two internal routers, the network is called a blocking network. Indeed, it is not possible to have two concurrent communications on the same link. Arbitration is required to decide which communication has priority.

A network can have three non-blocking conditions [Hwa03]. A strictly non-blocking network is always able to connect any input to any output regardless on the already used connections. A wide-sense non-blocking network is always able to connect any input to any output if all the connections follow the same routing algorithm. A rearrangeably non-blocking network is able to establish any connection, but might require a new organization of the connections inside the network. Lighter non-blocking conditions reduce the complexity of the network, as they usually require less routers. However, they usually impose constraints on their configuration. For example, the selection of communication paths in rearrangeably non-blocking networks is a non-trivial problem.

Multicasting ability Interconnection networks are mostly used to set unicast connection, a point-to-point connection from one source to one destination. However, some networks offer the possibility to have multicast connections going from one input to multiple outputs, and even broadcast connections (one input to all outputs).

4.4.1.3 Network selection conditions

The selection of a network topology is linked to its adaptation to interconnection needs. Therefore, it is necessary to translate flexibility constraints into selection conditions.

Inputs and outputs constraints As any feature can be transmitted to multiple output processing elements, the network must support multicast. In addition, the design aims to have the maximum possible number of processing elements, and then the network must be asymmetric.

Synchronized transmission The synchronization of features ensures that, at each clock cycle, information about a whole packet is transmitted. This synchronization must be kept and imposes that the features must be transmitted concurrently at the same constant latency inside the network. The constant latency implies a non-blocking architecture to avoid information loss.

Configurable connections The goal of the interconnection is the control of the distribution of input features among the output features to have a maximum flexibility. The state of the network only has to be modified when the operator or a software application require new specifications.

This configuration condition is compliant with a circuit switched network as no dynamic routing is required between two configurations. With no data transmission overhead and less complex routers, this solution is preferred over a packet switched network. Furthermore, the computation of connections can be directly computed in software to reduce the complexity of the design.

A new configuration from the software changes the network state and triggers the reconfiguration of the connections. A strong non-blocking condition is then not necessary for the network. A rearrangeably non-blocking network is sufficient to ensure that all the connections can be made at configuration time, while requiring as little resources as possible.

As a conclusion, the final interconnection network must be:

- asymmetric,
- supporting multicast communications,
- with constant latency,
- circuit switched,
- rearrangeably non-blocking.

4.4.2 Adapted interconnection architecture

4.4.2.1 Crossbar matrix

The crossbar matrix is the most basic interconnection architecture. It is composed of crosspoints dedicated to the connection of one input to one output. For $N1$ inputs and

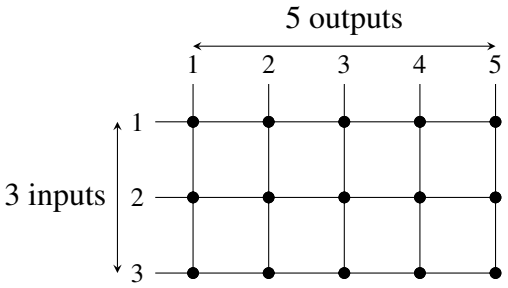


Figure 4.16 – Crossbar matrix with 3 inputs and 5 outputs

$N2$ outputs, the crossbar is composed of $N1N2$ crosspoints. Figure 4.16 displays an example crossbar with 3 inputs and 5 outputs.

The crossbar contains all the possible connections between inputs and outputs. Therefore, this architecture has enough connection points to ensure that the resulting network is strictly non-blocking for unicast communication [Clo53]. Moreover, a crossbar can be configured to support strictly non-blocking broadcast connections [MP10]. It is a one-stage switching network and it is possible to have operations processed in constant latency.

This interconnection network satisfies the requirements set by the parser design. However, the crosspoint matrix is not a scalable structure. Increasing the size of a crossbar matrix is not conceivable due to its high cost.

4.4.2.2 Multistage network

Widespread architectures are the multistage interconnection networks [Fen81]. They use multiple stages of parallel crossbar switches to concurrently connect an arbitrary number of inputs to an arbitrary number of outputs. The executed permutation is selected by a control algorithm determining the state of each switch of the network.

The different topologies of this network family are not focused on the execution of the same subsets of permutations. The permutation set executed by the network directly influences the number of switching elements composing the network as well as the interconnection between these elements. A network with a wider permutation set requires more switching elements and a larger transmission latency. For instance, the butterfly network presented in Figure 4.15b has a reduced permutation set compared to the Benes network in Figure 4.17, but is smaller. It is possible to design a multistage interconnection network for the execution of specific permutations with an optimized number of switching elements [Soo83]. The searched interconnection network must be able to connect any input to any output, and then must execute any permutation between inputs and outputs.

In addition, these networks can be design with a non-blocking topology which support multicast connections [YM91]. Multistage interconnection networks are an adapted solution for the feature distribution problem.

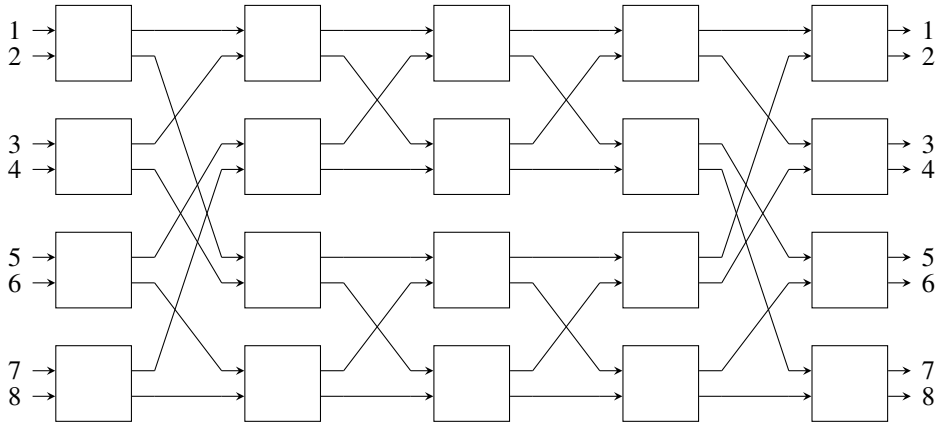


Figure 4.17 – A (8x8) benes network

4.4.2.3 Simple implementation

The Benes network [Ben62] is a symmetric rearrangeably non-blocking multistage network based on (2x2) crossbar switches. Figure 4.17 shows a (8x8) Benes network architecture. This simple structure has well-known performance: $N = 2 * r$ inputs with $2 * log_2(N) - 1$ stages and $N/2$ crossbar per stage. A Benes network is capable of sustaining multicast connections [MF97]. Its simplicity makes it ideal for a simple implementation.

The purpose of the proposed interconnection network is the connection of the N_1 inputs to a wider number of outputs N_2. As the Benes network is not designed to support asymmetric inputs and outputs, the construction of an asymmetric network is done by using parallel benes networks, each connecting the N_1 inputs to a subset of N_1 outputs. Figure 4.18 describes this solution where $N_2 = k \times N_1$. A first step executes k duplications of each input to supply the N_1 inputs to each Benes network. The second step connects the inputs to the outputs through k Benes networks. The rearrangeably non-blocking property of Benes networks ensures that any output can be associated with any input. As these networks support multicast connections, it is possible to distribute one input to multiple outputs and even all outputs.

Table 4.2 summarizes the resource consumption relative to the Virtex 7 XC7VX690T FPGA for the transmission of 32-bit data for multiple inputs and outputs. Thanks to the usage of Benes networks, the complexity of the number of crossbars required in the overall design is $O(N_2 log_2(N_1))$, which is confirmed by the results.

Moreover, the LUT/FF pairs consumption in Table 4.2c is nearly the same as the LUT consumption in Table 4.2a. This architecture takes advantage of the division of LUT6 in two LUT3 to execute two (2x2) crossbars. This design allows to increase the number of inputs and outputs with a limited spread on the chip seen in Table 4.2d. This highly distributed interconnection architecture is very compact, allowing to reach a very high number of inputs and outputs.

Despite efficient occupation space, these results must be compared to results obtained for the packet parser in Figure 4.13 to estimate the impact of the overhead of the interconnection network. For 150 extracted features, the packet parser uses 60 % of the FPGA slices and 40 % of the LUT. For the same space on the chip, a packet parser with an interconnection network is capable of distributing 32 features to 512 outputs

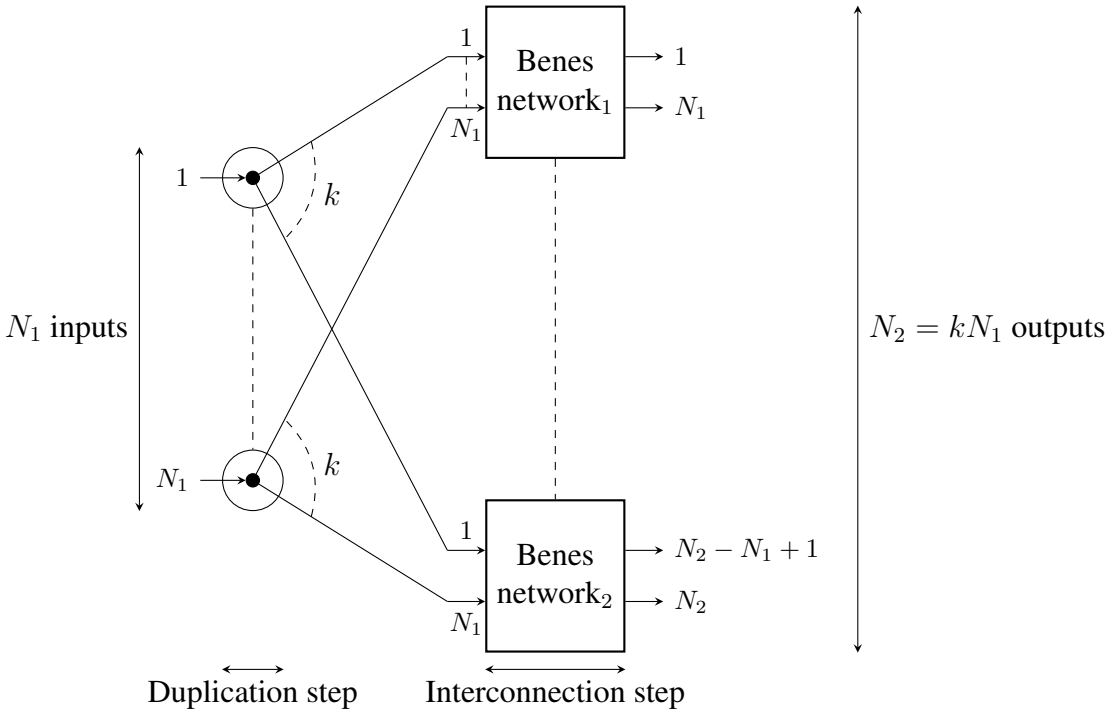


Figure 4.18 – Implemented interconnection solution

O \ I	32	64	128
256	8.517	10.405	12.295
512	17.032	20.809	24.588
1024	34.181	41.616	49.175
2048	68.242	83.466	n.a.

(a) LUTs consumption

O \ I	32	64	128
256	9.254	11.210	12.869
512	18.425	22.503	26.405
1024	36.412	44.852	53.003
2048	72.623	88.841	n.a.

(b) FFs consumption

O \ I	32	64	128
256	8.512	10.403	12.293
512	17.023	20.803	24.585
1024	34.161	41.605	49.169
2048	68.203	83.446	n.a.

(c) LUT/FF pairs consumption

O \ I	32	64	128
256	11.693	15.765	16.080
512	22.966	29.845	36.350
1024	44.789	54.791	64.846
2048	85.669	95.326	n.a.

(d) Slices consumption

Table 4.2 – Interconnection network resource utilization on XC7VX690T (%)

or 64 features to 256 outputs which is a substantial gain. However, adding the interconnection network increases the processing latency of the packet processing, with the formula in Equation 4.1.

$$latency = log_2(\frac{N_2}{N_1}) + 2log_2(N_1) - 1 \tag{4.1}$$

In the presented cases, the latency is increased by 25 and 15 clock cycles, or 139 and 84.3 ns at 180 MHz.

Despite good performance, this solution is not the most efficient in terms of resource consumption. The usage of Benes networks leads to overheads in resources consumption. Indeed, the number of inputs and outputs of a Benes network must be aligned on a power of two, imposing $N_1 = 2^{p-1}$ with p the number of stages of the Benes network. It is also necessary to have $N_2 = kN_1$ These problems have been studied in the literature. Chang *et al.* [CM97] proposed a recursive structure for the creation of arbitrary size Benes network where the number of crossbars linearly increases as a function of inputs and not by stages. When the number of inputs and outputs are not aligned on a power of two, the number of interconnection points saved is substantial. This structure has been improved in [SH16] to consider an arbitrary size Benes with less outputs than inputs, which can be used in our case when N_2 is not a multiple of N_1.

4.4.2.4 Toward efficient interconnection network: Clos network

While the proposed architecture answers the required constraints of an interconnection network, more efficient solutions exist. Clos networks [Clo53] are a set of symmetric and asymmetric interconnection networks composed of arbitrary size crossbar switches designed to be the most efficient as possible. The Benes network is a specific example of a symmetric Clos network composed of only (2x2) crossbar switches. This section studies the ability to use a Clos network for the dynamic distribution of the packet features to multiple processing elements.

The interconnection scheme of Clos network is made up of three different stages relying on smaller interconnection switches. Figure 4.19 describes the general decomposition of an asymmetrical Clos network denoted $\mu(m, n_1, r_1, n_2, r_2)$. It is separated into an input, a middle and an output stages. The input stage is composed of r_1 $(n_1 \times m)$ crossbar switches. The output stage is composed of r_2 $(m \times n_2)$ crossbar switches. The middle stage has m parallel smaller interconnection networks with r_1 inputs and r_2 outputs. The total number of inputs is $N_1 = n_1 r_1$ and the total number of outputs is $N_2 = n_2 r_2$.

With the right parameters, this solution offers the most optimized number of crosspoints used in comparison to crossbars while being strictly non-blocking if $m >= (n_1 - 1) + (n_2 - 1) + 1$ [Clo53]. An other advantage of Clos networks is their recursive construction. Indeed, interconnection networks of the middle stage can be considered as Clos networks on their own. They can be divided with the same principle in three stages constituting a global interconnection network with five stages. It is possible to repeat this division until the middle stage is constituted of basic crossbars.

Primarly described for unicast communications, these results have been extended to multicast connections. As the multicasting ability increases the complexity of in-

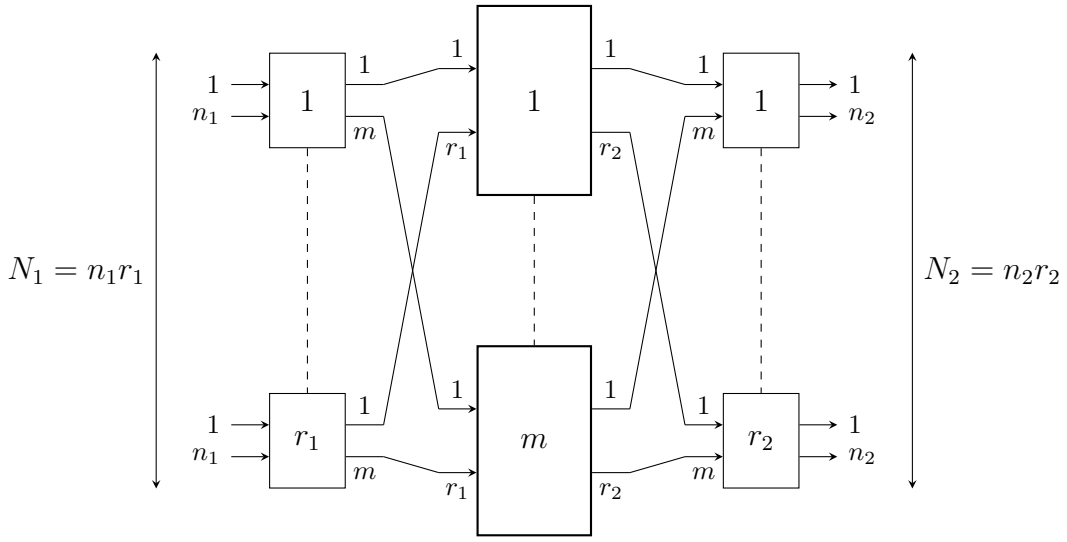


Figure 4.19 – (N_1xN_2) Clos network denoted $\mu(m, n_1, r_1, n_2, r_2)$

terconnection switches, multiple models exist depending on the stage of the network which has no multicasting ability [Hwa03]:

- model 0: all the stages have multicast connections,
- model 1: the input stage is not able to create multicast connections,
- model 2: the middle stage is not able to create multicast connections,
- model 3: the output stage is not able to create multicast connections.

These models requires a different number of interconnection points to answer the non-blocking conditions. An advantage of the model 2 is a lower network hardware complexity or control mechanism of the middle interconnection. Masson *et al.* [MMWJJ72] demonstrated that the model 2 Clos network is rearrangeably non-blocking for multicast connections, and the condition was refined in [Hwa05] to $m >= max\{min\{n_1 r_2, N_2\}, min\{n_2, N_1\}\}$.

The limit of Clos networks is efficient routing because the rearrangeability of a multicast Clos network is an NP-complete problem [JK10]. However, for a model 2 multicast network, adding a new connection costs at most $r_2 - 1$ rearrangements [MMWJJ72]. The proposed solution is able to compute the new arrangement in software. The configuration is done at the same time on the network, connections are not singularly moved. A rearrangeably non-blocking Clos network only ensures that any connection can be routed through the network.

Rearrangeably non-blocking multicast Clos networks are adapted for the connection of the packet parser to the processing elements. In addition to fill the requirements, this type of network can be easily built in a recursive manner and is more scalable than classic crossbars. However, many parameters can be tuned to optimize the network and multiple model exist for multicast communications. As no implementation was found on FPGA, further work is required to study the influence of these parameters on the number of crosspoints, latency and FPGA resources consumption.

4.5 Conclusion

This chapter has introduced a novel flexible packet parser architecture. The design relies on a static hardware architecture to be able to sustain determined high rates of traffic. Unlike solutions of the literature, this design is entirely based on configuration parameters to process incoming packets. This approach is highly flexible allowing the modification of considered headers as well as extracted features at runtime. This flexibility makes possible the selection of processed headers from the software according to the immediate needs of end-user applications. The design and the implementation of this architecture has led to a publication [CJ+17b].

However, the genericity of the design does not come for free. The increase of resources consumption is not negligible especially when adding new extracted features. The distribution of features to the processing elements costs lots of interconnection resources, especially if the same feature must be supplied to multiple processing elements.

An efficient way to connect the packet parser and the processing elements is the use of a specialized element called an interconnection network. To answer the requirements of the architecture, this network must be asymmetric, with a low number of crosspoints, having rearrangeable connections and a constant transmission latency and sustaining multicast connections. Multistage interconnection networks are a class of networks answering the needs. A basic implementation of an asymmetric interconnection network is built with duplication stages and parallel Benes symmetric networks. No alternative has been found in the literature. This solution offers an asymmetric network capable of connecting inputs to thousands of outputs. While this is not the optimal choice, results demonstrate that, when integrated in the packet parsing architecture, this interconnection network leads to better occupation space than feature duplication. Clos networks, the generalized version of Benes networks, seem to offer the best interconnection alternative, but a large number of parameters must be tuned to obtain the right network. As no implementation on FPGA devices has been found, future works will focus on the study of these parameters for optimized resource consumption.

With the combination of the proposed packet parser and interconnection network, the resulting packet parser executes the extraction of any feature from any header and can distribute these features to any processing element. This unique architecture is a great building block for flexible packet processing applications. Indeed, in the next chapter, based on this work, I will propose a novel agile high-speed network probe with properties never encountered before.

Chapter 5

Towards agile high-speed network monitoring

5.1 Introduction

The work presented in the previous chapters aimed at finding clues for the design of architectures for flexible packet processing on high-speed links. In this chapter, the concepts and architectures previously defined are used to develop monitoring probes. These novel probes are used to execute packet monitoring applications at 40 Gbps with a flexibility never encountered before. After a succinct description of the test-bed used for high-speed network processing, the two considered application cases are presented and evaluated.

A main part of network protection is the study of traffic anomalies for the detection of attack patterns. Network forensics is a use case of interest, because it requires a dynamic adaptation to the traffic content for the refinement of collected data. For this example, a simple forensic application is executed based on the proposed packet parser. With the unique flexibility of the hardware architecture, this application is able to collect more diverse information on the traffic than common applications. Indeed, for performance issues, common monitoring applications targeting high-speed links only consider information from the main network traffic. The proposed application is capable of adapting at runtime the monitoring probe to process non common protocols, which can be a main threat to the network because they are less monitored. As a consequence, the proble makes the network more secure despite its simplicity.

The second test is focused on a more complete application for live packet analysis. The hardware packet parser is associated with hardware rule processors to create a high performance packet classifier. The simple application defines hardware packet selection rules to extract packets of interest from the network traffic for further processing on a reduced data rate. This hardware selection is refined in software with a second layer of advanced rules. This highly programmable architecture offers more advanced rule configuration possibilities than any current high-speed classifier solution. In addition to the feature values, the features used in each rule can be defined. The large possible rule set and the hardware/software feedback loop allow a fine control of the traffic according to the needs of end-user applications.

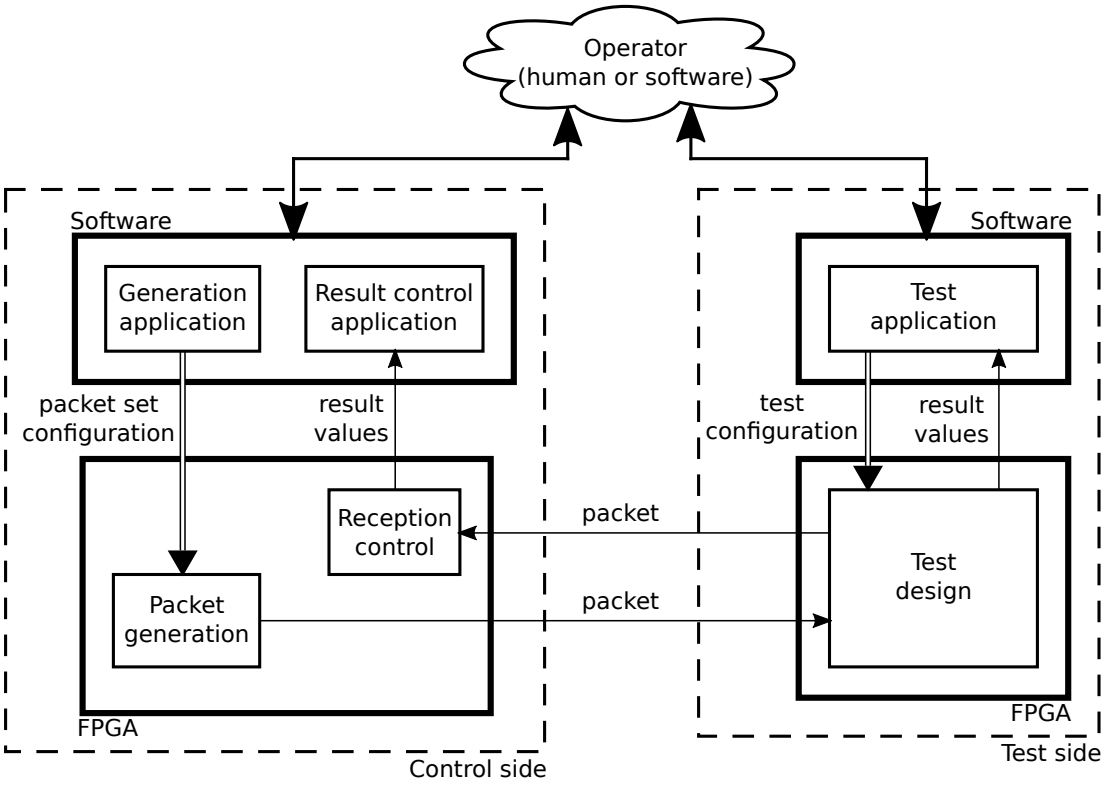


Figure 5.1 – Test solution architecture

5.2 Test-bed architecture

5.2.1 Test organization

The test infrastructure aims at verifying the functionality and the performance of the design in a controlled environment. A stress test with a synthetic load traffic is applied to the tested probe. Figure 5.1 illustrates the test structure which is divided into two parts.

The test side contains the application to stimulate. The design to verify is set on the FPGA of the smart NIC. The software environment of the test host offers to the operator the possibility to add a software test application. This application is able to monitor the probe results and to configure the probe during the test duration.

The control side is used to set the test environment. A generator sends the stress traffic to the tested equipment. With the generator presented in Section 3.4, it is possible to have a load traffic with composition variation. An optional reception engine can be added to control the transmitted packets when the tested element transfers packets. This functionality is not used in the tests presented in the following sections.

C++ and Python APIs are available for interaction with the hardware part of the generator and the receiver. This software-oriented approach makes the development of high-speed tests accessible to network engineers, which are the most capable of verifying the functionality of the devices under test.

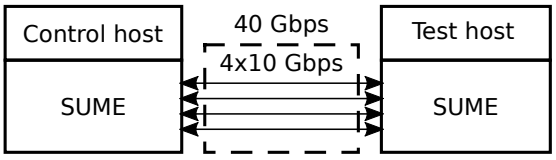


Figure 5.2 – Test-bed with NetFPGA SUME board

5.2.2 40 Gbps test-bed

The main test platform is built with two hosts containing NetFPGA SUME boards. Figure 5.2 shows the interconnection of the two smart NICs with their four 10 Gbps interfaces. An FPGA design is able to concentrate the links on one common datapath. Therefore, the resulting link can be used as four different 10 Gbps links or as one 40 Gbps link. This is the platform used for the tests in this chapter.

5.3 Flexible high-speed packet parser validation

5.3.1 Experimental probe

A common threat to computer networks is a volumetric attack. The malicious traffic aims to exhaust resources of a server with the reception of a massive number of packets. The wide presence of connected objects facilitates the creation of malicious traffic and increases the intensity with any protocol as a target [BI17]. Moreover, volumetric attacks can be used as vectors of other attacks which are hidden inside the massive traffic.

The overload orientation of these attacks makes them detectable if enough processing power is available and if it is possible to monitor the targeted protocol. Such an application is a good candidate to validate the probe: it saturates data links, and it requires a good agility to adapt to the protocol being targeted by the attack. The ability to efficiently protect a network from such an attack relies only on the ability to monitor all affected protocols at a sufficient speed. Absorbing all the overload traffic is necessary to determine if no other problem is hidden.

The proposed experimental setup aims at providing sufficient proof that the packet parser is able to perform at the expected performance level, and to evaluate its impact on applications in terms of configuration latency and ease of use. A simple configurable hardware monitoring is created by coupling the packet parser to a basic preprocessing unit, as illustrated in Figure 5.3. The hardware part is completed with a software application, which performs the actual attack detection and takes decisions to adapt the configuration. The hardware test design has a limited consumption of FPGA's resources: 34.22% of the LUTs, 15.17% of the FFs, 46.58% of the slices, 40.15% of the LUT/FF pairs. Spare resources are available to implement accelerated application processing on the FPGA.

The preprocessing unit is composed of filtered counters on selected features and an anomaly detection based on destination IPv4 count. One parametrable filter and one counter are available for each possible extracted feature. The anomaly detection is done with a change point detection over Count Min Sketch (CMS) [SVG10] coupled

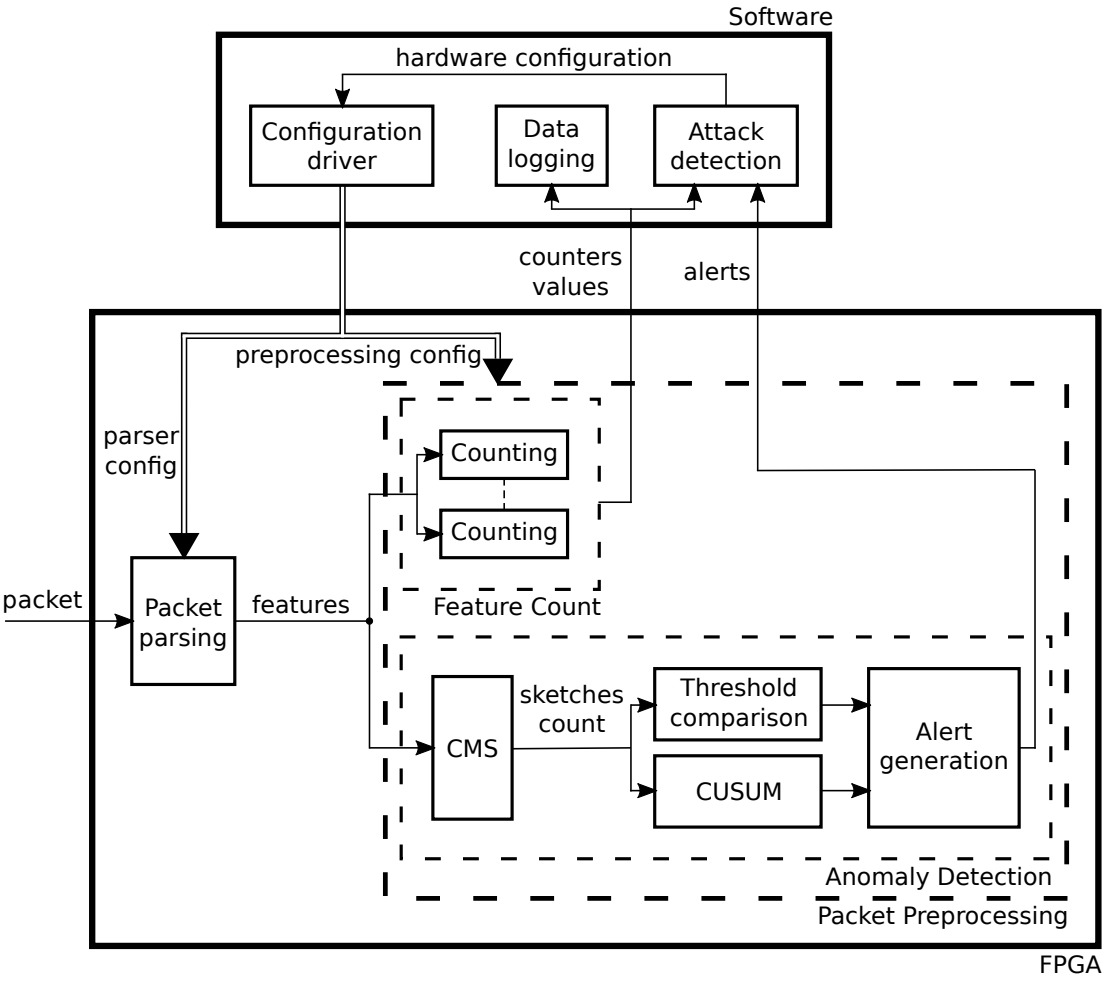


Figure 5.3 – Test solution architecture

with threshold detection. Counters and alerts generated are transferred to the host machine via the PCIe connection for further processing.

Based on this simple probe implementation, a first remark can be made: making use of features from hardware is as easy as processing a FIFO queue. This means that common data processing elements can be easily integrated, removing the complexity induced by data extraction and frame management. Using a generic unit, the same preprocessing can also be applied to different fields without any hardware change. For example, with a counter able to process up to six-byte fields, the same architecture can be used to count Ethernet packets or IP packets.

Required information is decided by the software part with the configuration of fields to monitor. In addition, filters can be programmed to count only one value for a feature. With duplicate extraction, it is possible to watch the evolution of multiple values for the same field. The configuration is entirely done through software calls to a simple API, which completely abstracts the hardware implementation. The use of hardware for preprocessing is completely hidden from the developer view.

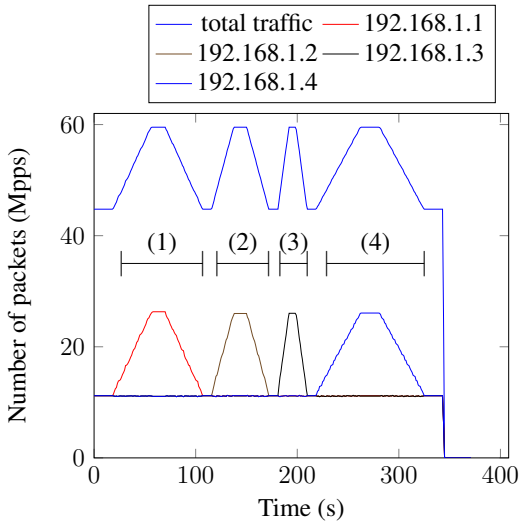


Figure 5.4 – Incoming packet counters

5.3.2 Benchmark scenario

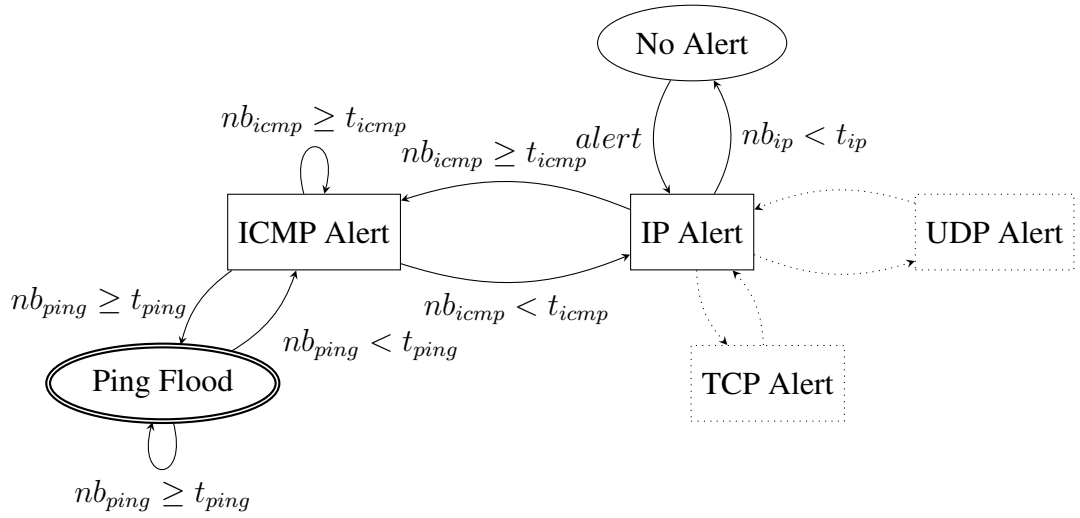
The test probe is fed with a synthetic traffic composed of a base traffic mixed with attacks on different protocols. Considered attacks in the experimentation are common ones like ICMP ping flood, TCP syn flood, DNS QUERY flood and HyperText Transfer Protocol (HTTP) GET flood [OVHa]. This list only affects the software part: any attack which can be detected by counting occurrences of a field and comparing to a threshold can be processed by the probe, as long as a configuration is provided.

For the sake of clarity, each attack is targeting a distinct IP address. The base traffic creates a floor of 30 Gbps of minimal size 64-byte packets, being 44.642 Mpps. Attacks are sent successively completing the base traffic to reach 40 Gbps, 59.523 Mpps, the maximum rate achievable with the current card interfaces. Global traffic shape received by the probe is visible on figure 5.4. Different spikes in traffic correspond to different attacks:

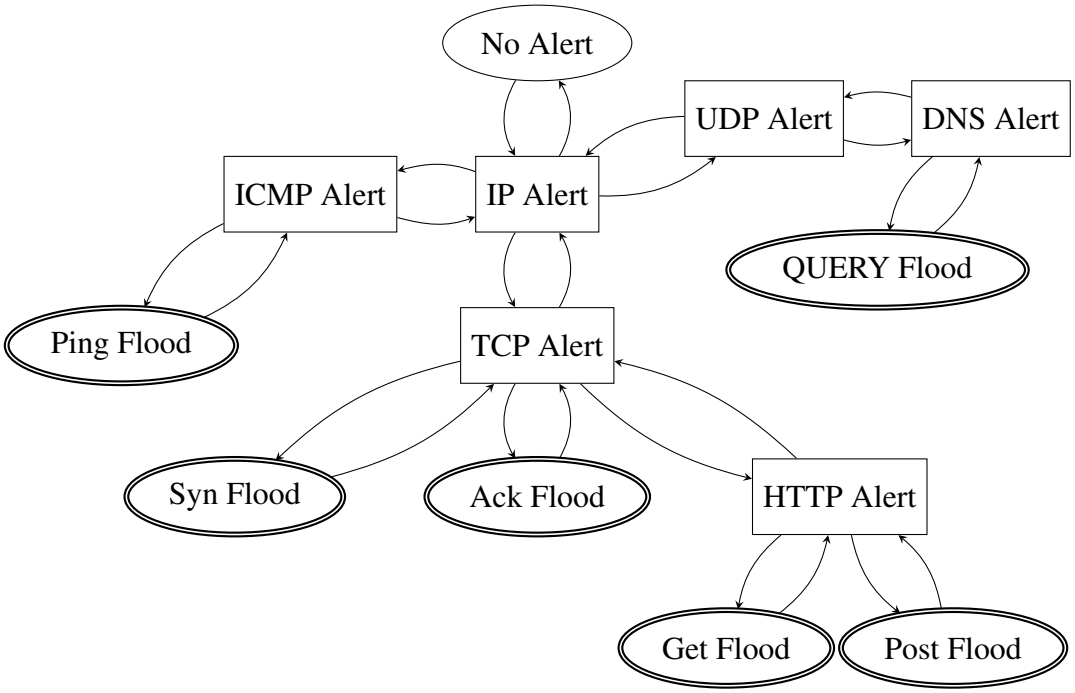
- ICMP ping flood on 192.168.1.1 (1)
- TCP syn flood on 192.168.1.2 (2)
- DNS QUERY flood on 192.168.1.3 (3)
- HTTP GET flood on 192.168.1.4 (4)

This test-bed is available under an open-source license on NetFPGA SUME for verification [CJb].

A program developed on purpose in Python is monitoring this incoming traffic. This very basic program detects volumetric attacks, including the ones inserted inside the test, on a destination IP address. This program decides which kind of protocol is monitored. As many features as decided at design time can be simultaneously analyzed. A first interesting advantage of the proposed solution is that the huge amount of data is hidden from the software program.



(a) Branch detection



(b) Global decision steps

Figure 5.5 – Detection decision steps

When an alert is set, parameters of the probe are incrementally computed by the Python program, to refine the detection with adapted counter values while keeping track of other possible threats. Decision steps are shown in Figure 5.5.

Figure 5.5b describes all the states of the monitoring program and their transitions. The No Alert state is the normal state when no anomaly exists on the network. Ellipse with double lines are detection states when an attack is confirmed and countermeasures are executed. Each step leads to the modification of the parameters sets with the addition or removal of extracted features adapted to the observed traffic.

Figure 5.5a refines the description with the addition of the transition conditions for Ping Flood detection. Dotted branches are not described in the diagram. This simple application uses the comparison of the counter values from the chip with a threshold to determine if further forensics is required. For one transition, the used threshold is associated to a set of features.

This procedure takes advantage of the flexibility of the approach to selectively monitor protocols according to the traffic shape at a given time. Any protocol can be described using the proposed set of parameters, and software integration is as simple as calling the corresponding functions. This application is a proof of concept, and only implements basic functionality. More complex forensic applications could obviously take full advantage of the proposed agile and high data rate probe.

5.3.3 Test results

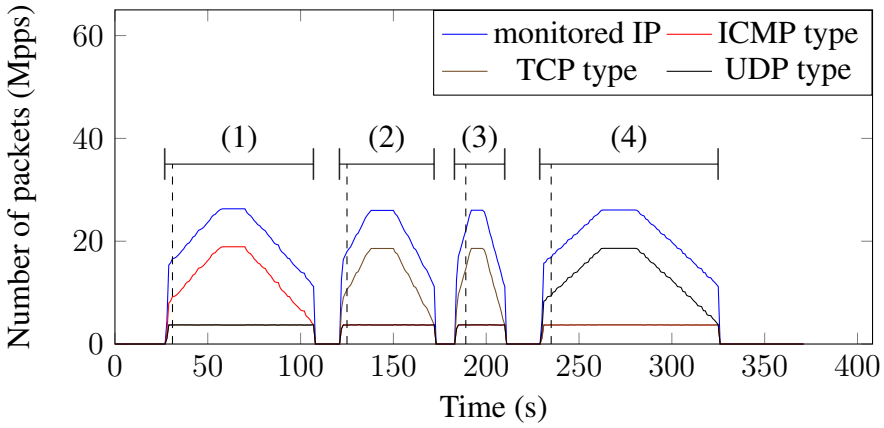
Figure 5.6 summarizes counter values over time for traffic content-aware specialization on IP, TCP and HTTP protocols, as response forensics for received traffic shown in Figure 5.4. Figure 5.6a shows the recorded traffic for a specific IP, Figure 5.6b a refinement on TCP traffic, and Figure 5.6c a refinement on HTTP traffic.

Holes in the curves represent time intervals when no anomaly is detected, and then no recording is necessary for these protocols. The same behavior can be observed with ICMP, UDP and DNS test traffic. All configuration refinements are done at runtime on demand of the software. Dash vertical lines represent the moment when attacks are detected. The detection of these attacks are performed at the beginning of each traffic spike, showing the short reaction time of the probe. Counter values export interval is configurable in a range from micro-seconds to seconds, directly influencing detection time. The probe is able to differentiate protocols at a rate of 40 Gbps.

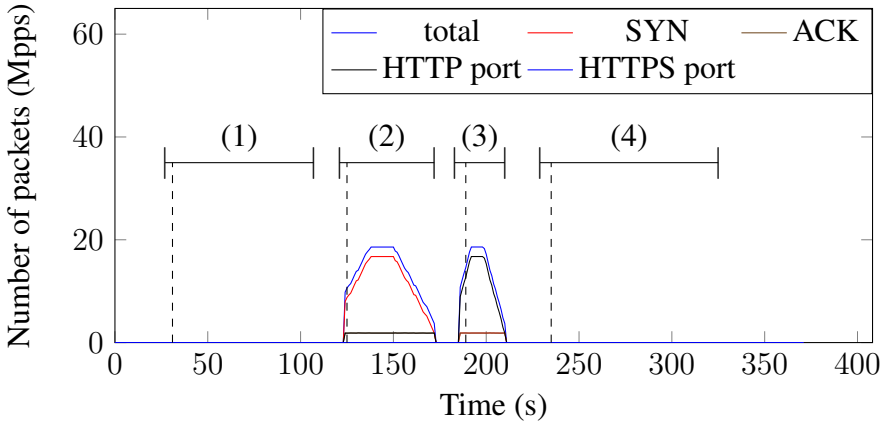
This example shows the possibility to monitor high data rate traffic in an agile way. To monitor the same type of traffic, a classic FPGA approach will reserve resources for each protocol, even if the usage is not optimal. Finally, on a classic approach, if a new protocol needs to be added to the monitoring list, a new binary needs to be generated and the FPGA needs to be reconfigured. For the proposed design, adding simply corresponding protocol parameters to settings is enough and done by the software.

This test shows that the probe is adaptive and always tuned for current incoming traffic, even at 40 Gbps. In addition to be easily done in Python via an API, parametrization of the packet parser allows considering the same protocols as in software. This test probe even processes data rates way beyond those processed by its full software counterpart.

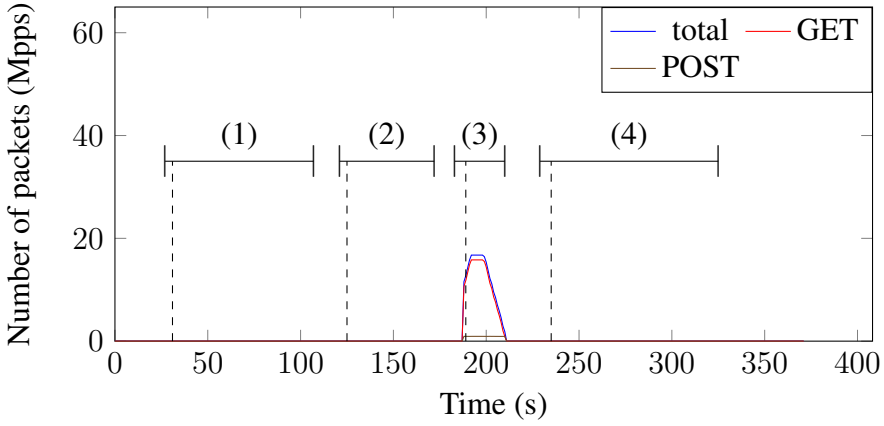
The presented packet parser combines high performance, high agility and ease of



(a) IP counters



(b) TCP counters



(c) HTTP counters

Figure 5.6 – Traffic distribution seen by the probe during the dynamic attack detection

Rules	Description
R_1	$F_1 = 10$ and $F_2 = 30$
R_2	$F_2 \geq 0$ and $F_2 \leq 20$
R_3	$F_1 = 10$ and $F_2 = 5$ and $F_3 = 30$
R_4	$F_1 = 5$ and $F_3 \geq 10$ and $F_3 \geq 20$
R_5	$F_2 = 40$ and $F_4 = 10$ and $F_5 = 50$

(a) Rules

Field set	Composition	Associated rule sets
$\mathcal{F}$	F_1,F_2	$\mathcal{R}$
$\mathcal{F}'$	F_1,F_2,F_3	$\mathcal{R}'_1$,$\mathcal{R}'_2$
$\mathcal{F}''$	F_2,F_4,F_5	$\mathcal{R}''$

(b) Field sets

Rule sets	Description
$\mathcal{R}$	R_1,R_2
$\mathcal{R}'_1$	R_1,R_2,R_3
$\mathcal{R}'_2$	R_1,R_2,R_4
$\mathcal{R}''$	R_2,R_5

(c) Associated rule sets

Table 5.1 – Field sets and associated rule sets

use for end-users, as never encountered before. The test probe is able to switch between observed protocols at runtime which is not possible with packet parsers in the literature. The presented application is easily scalable with a smart NIC with more interfaces, or with multiple smart NICs connected at the same host, as seen in Chapter 3.

5.4 Flexible packet classifier

5.4.1 Packet classification

Packet classification is an important tool for network equipment. Monitoring applications heavily rely on classification engines to discriminate the different packet flows. Determining the nature of packets is the key to distribute them across the right following monitoring applications. These applications are in charge of taking adapted actions, such as packet filtering or traffic priority routing.

The discrimination of packets is done with a classifier composed of a collection of rules based on a set of fields extracted from the packet header. A rule is the definition of specific values for the different fields of a field set. Table 5.1a describes five different rules using five different fields of a packet header. One rule is matched when values of the packet fields correspond to those defined by the rule.

The achievable rule set depends on the field set used. Different field sets lead to different rule sets. Table 5.1b shows different field sets and the possible associated rule sets with the defined rules. Table 5.1c describes these rule sets. The classification process determines the rule matched by incoming packets inside the defined set of rules.

The comparison of a packet with a rule set can be done in multiple ways. Gupta *et al.* [GM01] presented a wide variety of existing classification algorithms. From basic linear search to complex tree-based search, the different data structures provide multiple tradeoffs between search speed, storage requirement, update speed and scalability.

SDN requirements impose to dynamically handle the rule sets in order to have a fine-grained and dynamic packet management. The widespread off-the-shelf CPU-based hardware, with software flexibility and ease of use, is the main development platform for classification engines. Except for high-end classification applications, popular software classification frameworks are built around BPF [MJ93] and netfilter [Net]. Even if it is not the most efficient approach, they offer complete semantics for rule set specifications and are easy to update. This justifies their wide usage as a front-end to distribute packets among following applications.

5.4.2 Hardware classification

Performance limitations of commodity hardware with high speed links reduce the performance of conventional software classification engines. Growing importance of FPGAs and smart NICs have led to the proposition of architectures offloading the classification from CPUs.

5.4.2.1 Specific classification engines

The most straightforward solution is the creation of a circuit fitting the considered classifier. This approach is commonly used when processing acceleration is needed. With the reconfiguration ability of FPGAs, the design can be modified when the classifier changes. A preprocessing part translates the desired rule set into a specific circuit built for a search structure. So, the FPGA logic is tuned to implement an algorithm for the given problem. The rule set as well as the field set are fixed once implemented. A good example is the MPFC [Hag+14] approach processing 64-byte packets at 100 Gbps where the common linear search takes advantage of parallelization to execute all the rule matching in parallel. Such specific designs trade flexibility with long generation time for low occupation of the FPGA.

It is possible to combine this solution with partial reconfiguration of the FPGA to remove the processing stop during reconfiguration time [HBS15]. The ping-pong solution between two reconfigurable classification pipelines of Zazo *et al.* [Zaz+16] allows to execute a specific BPF rules set without stopping the card. This design is able to process a fully filled 100 Gbps link of 64-byte packets without loss.

5.4.2.2 Configurable classification

To achieve dynamic rule set management with rule modification, insertion or deletion, generic and configurable classification engines are required. The most common configurable architecture are CAM and Ternary Content-Addressable Memory (TCAM) architectures. They provide the possibility to concurrently match multiple rules on the same cycle, with an input key built on the full field set. The FPGA architecture is however not adapted to host such designs leading to resource overconsumption, limiting frequency performance and design scalability [Xil17].

Alternative solutions implement decomposition-based approach to use FPGA parallelism and to increase the performance. This approach is focused on the separation of packet header fields or subfields processing. Partial results are merged to compute rules and to obtain the classification results. In [PK09], partial results are processed

concurrently and merged to create a key for a hash-based technique. Performance is gained at the cost of a lot of memory consumption and an extensive number of accesses. With the StrideBV solution [GP12] [GJP14], exact match and range search subrules are sequentially matched in a pipeline and merged with a bitwise AND. The final result is obtained after the last stage. Qu *et al.* [QP16] refined this approach with a two dimension pipeline to increase the overall performance.

The configurability of these designs allow to change the rule set, but the field set is fixed once programmed. These solutions have a limited flexibility using an unique search key for the rules. As it is not possible to constantly extract a large set of fields due to resource consumption, these solutions use the classic 5-tuple used by networking applications for relevant information. This tuple is composed of the source IP address, the destination IP address, the port source, the port destination and the transport protocol, UDP or TCP. No complementary information can be obtained for the classification.

5.4.2.3 Architecture combination

More recent solutions combine multiple architectures in order to compensate the flaws of the different solutions. Kekely *et al.* [Kek+14] used multiple parallel architectures for effective FPGA utilization and fast search to handle smallest packets at 100 Gbps. An hash-based search engine is used for exact rule match while a binary search tree is focused on a prefix match. Both these engines are configured at runtime with a configuration from a memory. Fiessler *et al.* [FHS17] sequentially set a specific classifier for immutable rules and a generic classifier for dynamic rules.

While this approach optimizes the space consumption, these solutions use rule sets based on a static field set too. Thus, these designs have a limited flexibility.

5.4.2.4 Mixed architecture

To compensate the fixed key, HyPaFilter [Fie+16] and its upgraded version HyPaFilter+ [Fie+17] combine an hardware and a software classification engines. While the hardware engine is generated for exact matches on one rule set, this hybrid management gives access to advanced rules on a reduced traffic manageable by the software.

Even if this solution allows to apply more advanced rules on the packet with the software part, a traffic spike of unanticipated traffic would be able to blind this probe because of the lack of fast configuration of the hardware. For maximum flexibility and probe reactivity, it is necessary to have an adaptable rule set at the hardware level. Classification must be able to dynamically modify the rule set at runtime with rule inputs from any field from any header and from any protocol level. Modification of the field set is then required too.

5.4.3 Hardware/software packet classification

5.4.3.1 Flexible classification hardware architecture

The packet parser proposed in this manuscript dynamically supplies features to a classifier. To produce an innovative architecture for high throughput packet classification,

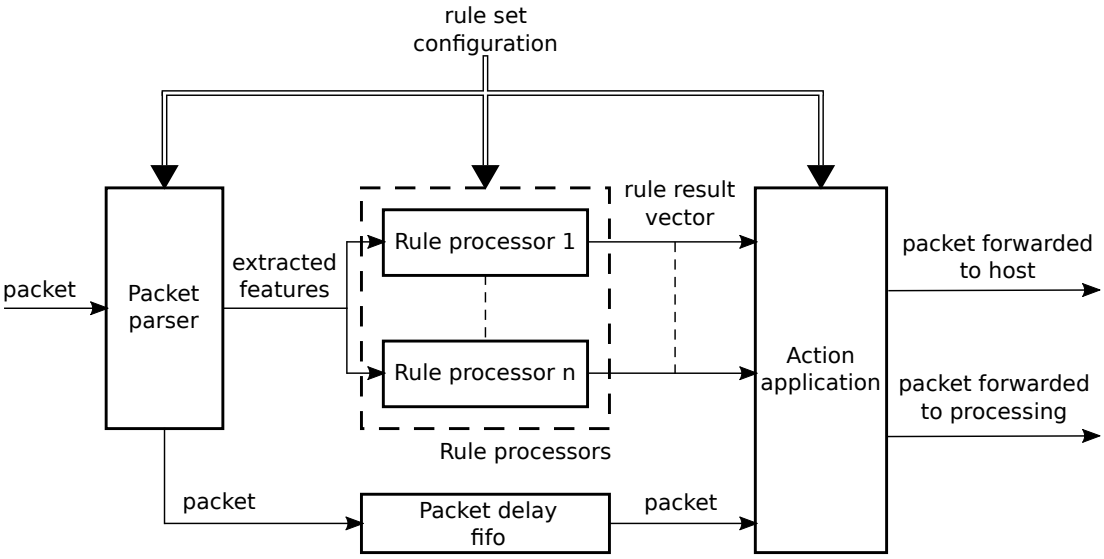


Figure 5.7 – Classification hardware architecture on the Smart NIC FPGA

the packet parser is associated with a configurable classifier. The architecture is composed of the packet parser and a set of rule processors, as described in Figure 5.7.

The packet parser decapsulates any configured packet header. Features of interest extracted from packet headers give distinctive information about the packet, even on application level protocols. Combined with the interconnection network, the packet parser is able to supply any extracted feature to any input of the classifier. The configuration of the parser from the software allows to extract information at will. For instance, this packet parser is able to change the supplied field set from $\mathcal{F}'$ to $\mathcal{F}''$ in the example presented in Table 5.1b.

The classifier uses the input field set to determine monitoring rules matched by the incoming packet. To support high throughput and to limit the processing latency, the classification architecture takes advantage of full parallelization on the FPGA. The latency of the classifier is equivalent to the latency of one rule processor.

Rule processors concurrently execute rule matching, each one with its own set of feature. The interconnection capability of the packet parser is handy to distribute features across the parallel rule processors. Classification produces a result vector used by a decision module to apply an action on the packet. The result is a bit vector where one bit describes if the corresponding rule is validated. Depending on the matched rules, the packet is forwarded to the host or to specific accelerated elements for further processing.

In order to dynamically process incoming packets, the classifier must be programmable in addition to the packet parser. Programmable rule processors are necessary to have a classifier which can be adapted to the needed rule configuration. In the example of Table 5.1c, rule processors configured to process the rule set $\mathcal{R}'_1$ must be able to be updated to process $\mathcal{R}'_2$ or $\mathcal{R}''$ with a new field set. With control API in software, it is possible to manage the hardware classifier as easily as a software classifier.

Hardware implementation forces to have parameters set at design time:

- the number of rules,

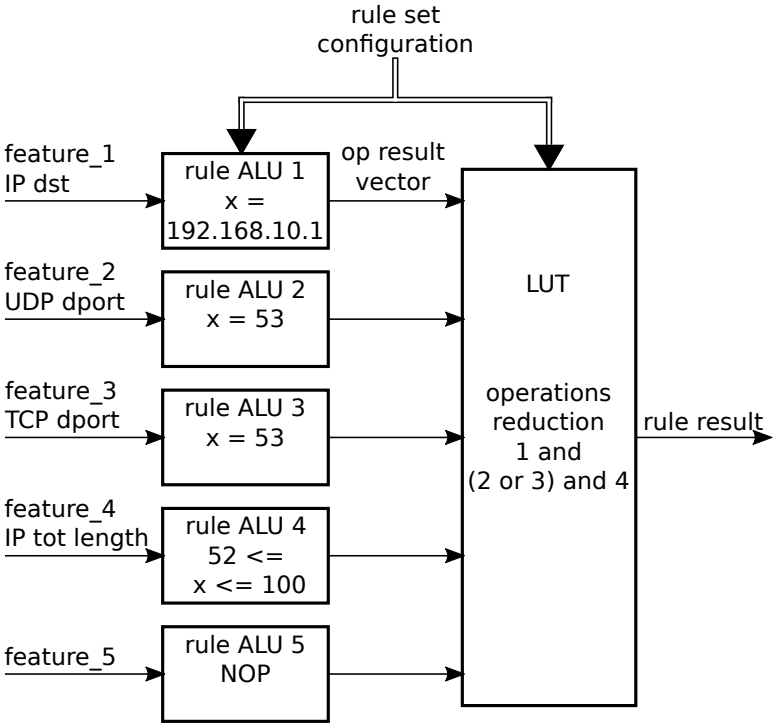


Figure 5.8 – Flexible rule processor of size 5

- the number of inputs per rule.

Packet parser header depth and maximum number of extracted features must be set at design time too.

The combination of the packet parser and multiple independent rule processors gives the classifier a unique range of abilities, never encountered in the literature:

- configurable field set,
- configurable rule set,
- input selection for each rule,
- line rate packet matching,
- rule selection entirely handled in software for user-friendly usage.

This hardware supports high throughput processing applications with high-flexibility and precise packet selection for advanced processing.

5.4.3.2 Flexible rule processor

Rule processors are the building blocks of the classification engine. One rule processor has a fixed number of inputs to validate the associated rule. The execution of a rule is a 2-step process following a map and reduce approach. Multiple parallel elementary matching operations are done by parallel Arithmetic Logic Units (ALUs). The bit vector result of operations is used by a LUT to compile the total rule.

The number of parallel ALUs gives the maximum length of the rule that can be processed. Every ALU uses one input feature of the processor. Therefore, the number of inputs gives the possible length of rules. While the number of available rule processors and the size of processed rules are limited on the hardware, further processing are possible on a reduced traffic to complete complex rules of the rule set. This partition of the packet classification between the hardware and the software is compliant with the paradigm presented in Chapter 3.

Figure 5.8 shows the different steps of rule validation on a rule processor of size 5. The example is associated with the rule `dst host 192.168.0.1 and (udp dst port 53 or tcp dst port 53) and ip[2:2] >= 52 and ip[2:2] <= 500` written in BPF syntax. This rule tracks packets traveling to a machine used as a DNS server and with a specific size range.

The behavior of the different ALUs is configurable from the computer host. ALU are able to execute a range of matching operations, but only one is configured at one time. The simple operation set is composed of equal, smaller than, smaller than or equal to, range inclusion and no operation.

Opposite operations are possible with the negation included in the LUT capability. The LUT combines the different elementary operations to produce the rule. The main advantage of a LUT is to be programmed to execute any logic function between its inputs. Therefore, the reduction can be configured to operate any combination between the elementary elements.

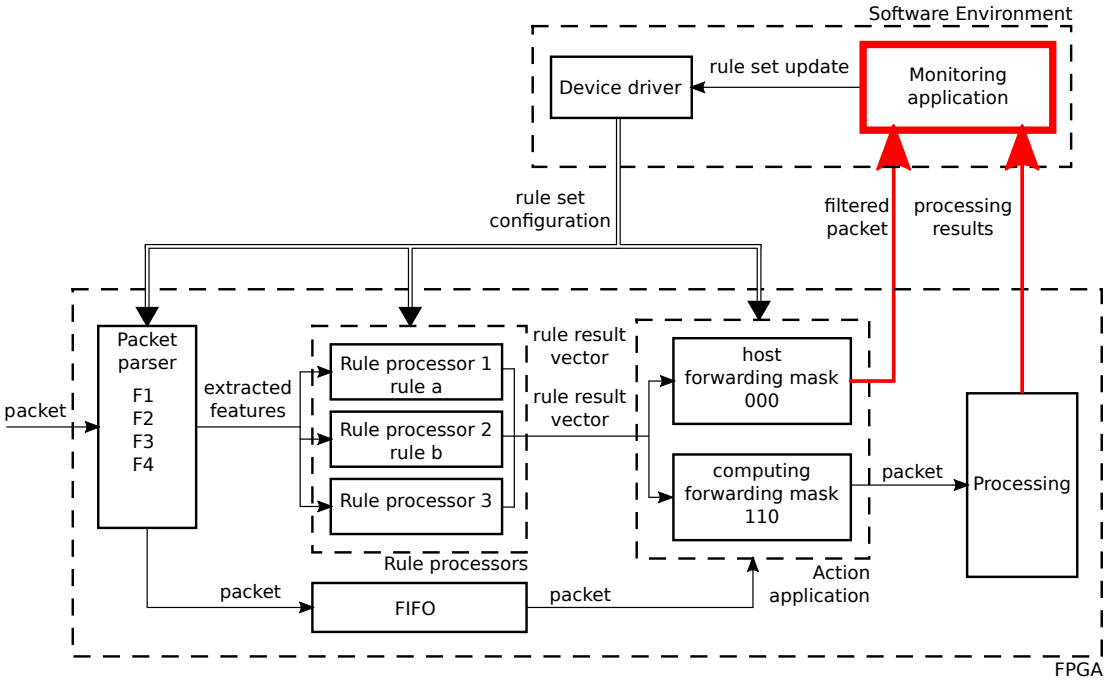
This classification architecture takes advantage of FPGA architecture to have parallel rule processors. Thus, it is possible to sustain high data rates and low latency. One processor takes two cycles to process one rule: one for ALUs computation, one for LUT operation. For instance, this represents less than 12 ns with a 180 MHz clock. This is negligible in the larger processing latency.

These rule processors have programmable operations and programmable feature operands from incoming packets. The classification has a unique flexibility, capable of dynamically processing more diverse rule sets than current hardware classification solutions. Moreover, since it is based on an FPGA, common reconfiguration is still available to upgrade or to resize the design according to the needs.

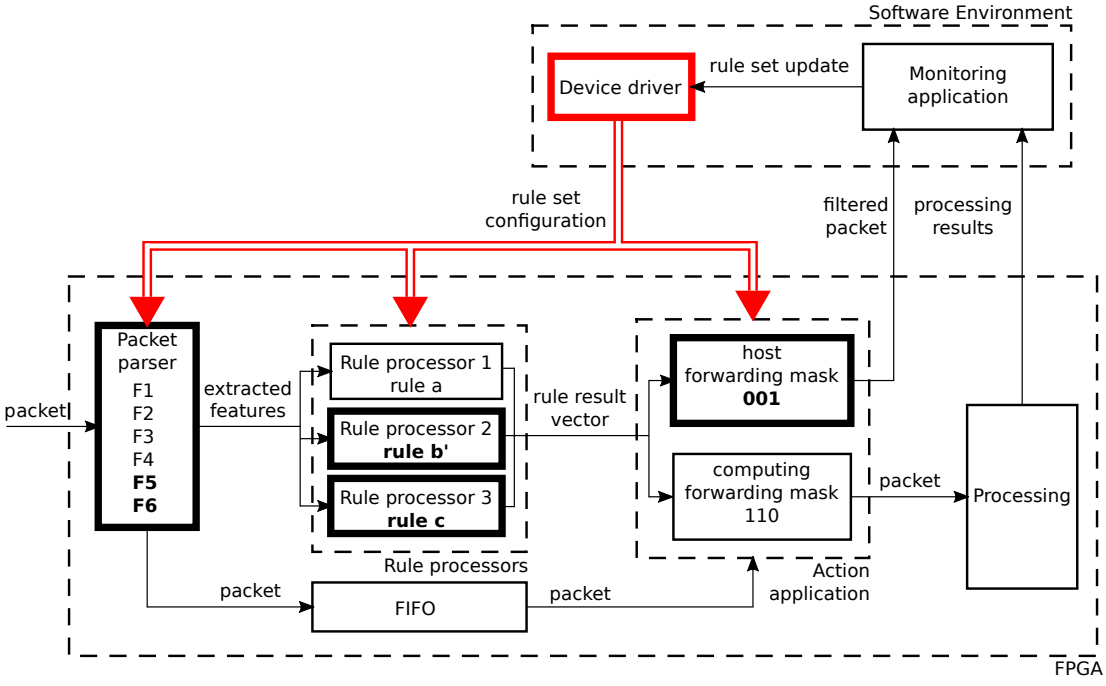
5.4.4 Hardware/software synergy

Classification engines produce a result vector used to determine actions to take on the packet. The pool of available actions is determined by the user at design time. It can vary from forwarding masks to decisions on most important rule matched with a priority encoder. This step transfers the packet to the subsequent accelerated processing stage, the combined host or simply drops the packet. While these decisions and the associated rules are prone to be modified according to the traffic evolution, they are configurable on the fly.

Figure 5.9 shows an example of a live feedback loop possible with the presented architecture. An anomaly detection in software triggers the reconfiguration of the classification chain. Based on some results from accelerated processing, the software monitoring application detects an anomaly and decides to refine the rule set in order to obtain detailed information in Figure 5.9a.



(a) Anomaly detection by the monitoring application



(b) Rule set update

Figure 5.9 – 2-step detection and reconfiguration

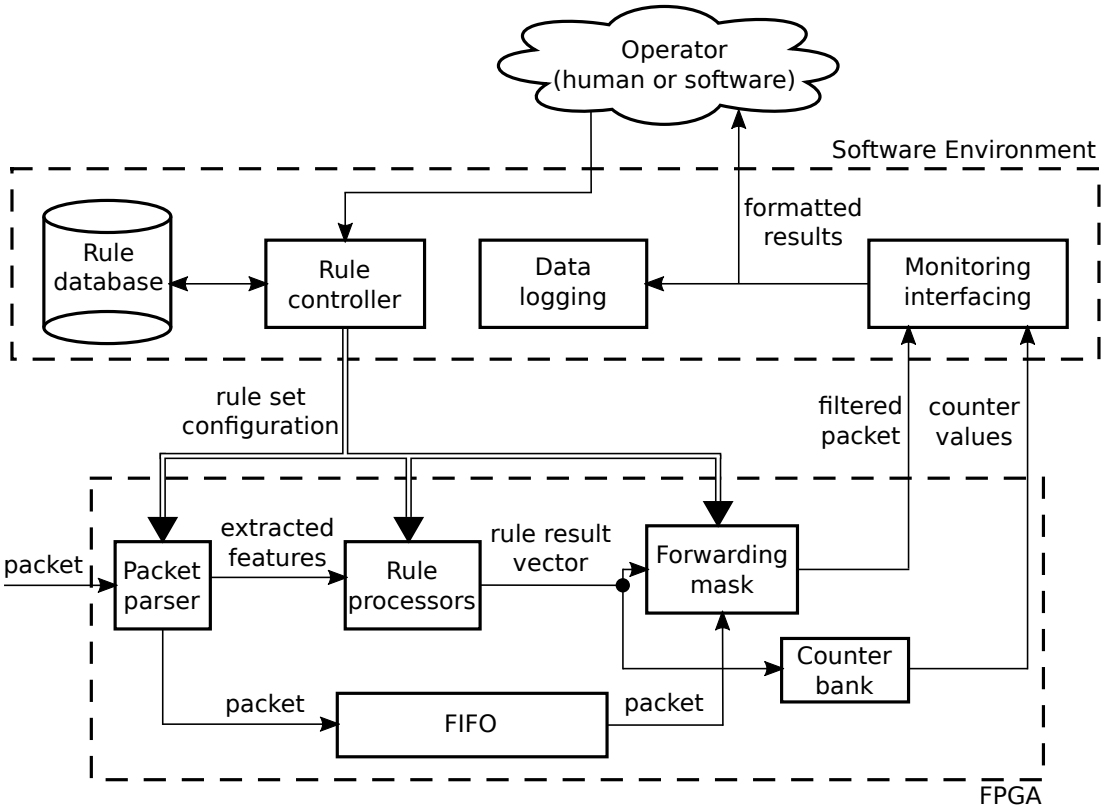


Figure 5.10 – Experimental probe architecture

In Figure 5.9b, the update process is modifying the processing chain according to the application requirements. New fields, F5 and F6, are extracted from the incoming packets. The rule set is changed to take advantage of these 2 new fields. Rule b is replaced by rule b' and rule c is added in hundreds of microseconds. Alongside the set modification, packets matching the rule c are forwarded to the host for further investigation.

As the whole design is based on configuration parameters, the control of the proposed probe is entirely done by the host software, and API calls allow manipulation of the probe. This software oriented approach completely abstracts the hardware part, thus improving the user experience. Moreover, final applications have a full control and knowledge on the preprocessing on packets received.

Thanks to this architecture, a high level of collaboration can be achieved between the hardware part and the end-user software applications. A possible application is a multi-layer filtering application, like in [Fie+16], with dynamic adaptation. A first step of traffic reduction is possible in hardware on the full link, and a second layer of advanced rules is done on a reduced traffic. This division allows complex classification with the CPU on the whole traffic and a reactivity of the probe as never done before, paving the way to future agile high-speed packet processing.

5.4.5 Experimental results

5.4.5.1 Experimental probe

The proposed architecture has been deployed on the NetFPGA smart NIC to build a 40 Gbps experimental probe. This setup focuses on the test of the flexibility and performance of the proposed classification architecture. Therefore, in addition to the classifier, the probe offers basic packet processing elements. Figure 5.10 summarizes hardware and software steps of the experimental probe.

Hardware processing is completed with a forwarding filter mask and a counter bank after the rule processors. One counter is tracking the number of times one rule is matched. It is then possible to know the distribution of the packets inside the global traffic. The forwarding mask allows to select packets matching specific rules for host transmission. These packets can be studied in detail while avoiding to overwhelm the host.

The final monitoring application uses the counter values to track the traffic evolution on the monitored high speed link. Depending on the observed results, an operator is able to refine the monitoring by updating the rule set. Rules are selected from a precompiled rule database filled in advance. This database can be dynamically completed according to the needs. The result is a live adaptive packet sampling probe providing an on-demand detailed view of the current traffic.

Based on this simple probe implementation, it is worth noting that the decision process for fields to monitor is totally handled by software through simple API calls. Software modifications of the implementation can be done with the abstraction of hardware implementation. Thus, the use of hardware for preprocessing is completely hidden from the developer. Moreover, hardware common processing elements do not depend on the classification part, and can be easily integrated, no overhead is necessary. This solution is then easy to use and to port between multiple end-user applications.

5.4.6 Benchmark scenario

The test scenario is built around the acceleration of BPF filtering rules. Although the test is centered on one filtering framework, the classifier can be used to process rules with any syntax if an adapted compiler is provided. Nonetheless, the growing importance of the BPF framework [Cil18] for network filtering in the Linux kernel makes essential the support of this framework.

The test traffic is again a volumetric traffic in order to test performance limits of the probe. The proposed test probe is fed with a synthetic traffic composed of a base traffic mixed with traffic spikes on different protocols. This kind of traffic offers a suitable validation ground for the test probe with saturated data links. Packet processing limitation of software makes the management of traffic spike out of reach. Continuous live adaptation to incoming traffic is required to avoid to saturate the probe with overwhelming traffic.

Figure 5.11 shows the profile of the traffic received by the probe. The base traffic creates a floor of 30 Gbps of minimal size 64-byte packets, being 44.642 Mpps. This traffic is composed of a combination of UDP and TCP packets. Traffic spikes are injected on the link alongside the base traffic to reach 40 Gbps, 59.523 Mpps, the

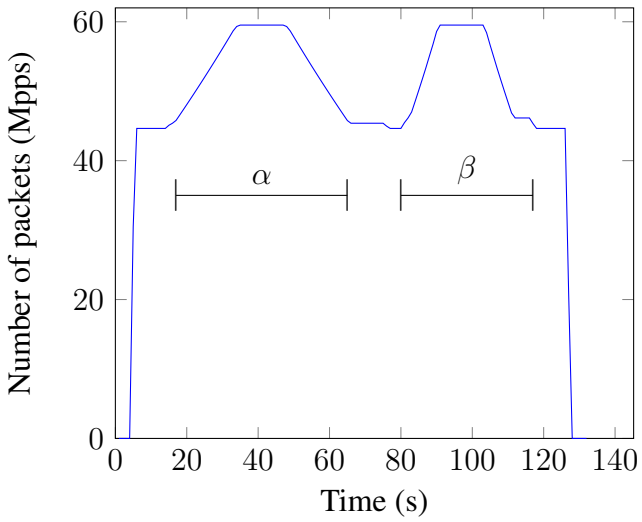


Figure 5.11 – Incoming total traffic

maximum rate achievable with the current card interfaces. This new traffic is a mix between the base traffic and packets targeting specific protocols. The different traffic spikes are visible in the global traffic shape on Figure 5.11. Spike α contains mainly ICMP packets and spike β contains mainly DNS packets. The generation of this traffic is done by a custom generator with the possibility of producing a 40 Gbps traffic of 64-byte packets. The test-bed is available under an open-source license on NetFPGA SUME for verification [CJa].

The reception of this traffic is handled by the proposed experimental probe. The hardware test design has a limited consumption of FPGA's resources: 28.80% of the LUTs, 12.66% of the FFs, 39.78% of the slices, 34.37% of the LUT/FF pairs. Spare resources are available to implement additional processing on the FPGA.

The rule set configuration classifies incoming packets with four permanent rules using constantly the first four rule processors. The last rule processors are left empty for live rule set adaptation when anomalies are detected because of the traffic spikes. The four static rules are, in BPF syntax [Bpf]:

1: `net 192.168.1`

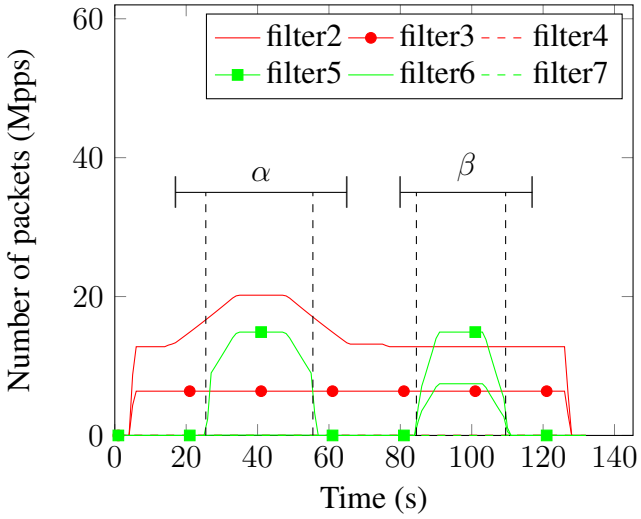
2: `ip host 192.168.1.10 and not 192.168.1.20`

3: `tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net 192.168.1`

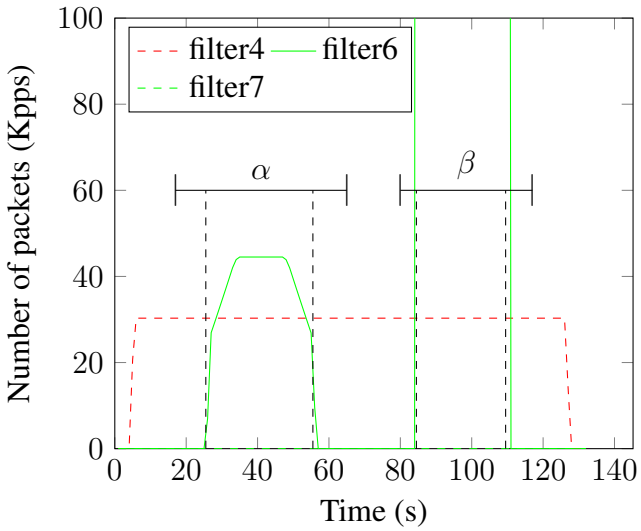
4: `port 80 and dst ip 192.168.1.10 and (((ip[2:2] - ((ip[0]&0xf) « 2)) - ((tcp[12]&0xf)»2)) != 0)`

As operation dependency in not yet handled by the rule processors, the first part of the rule 4 is done in hardware and the second part is continued in software.

Literature solutions have limited answer to this scenario because of limited flexibility. If observed traffic is unanticipated, a new firmware generation is needed with the new rule set. This time is too long to hope adapting the probe to incoming traffic



(a) Global rule counters



(b) Zoomed scale

Figure 5.12 – Counter values after packet filtering on the FPGA

spikes. For several solutions, the static search key makes impossible to process ICMP packets. This lack of flexibility leaves limited options for network protection. Either all the unanticipated traffic is blocked, which can not be acceptable, or it is not watched creating a massive security flaw.

5.4.7 Test results

The stress test aims at analyzing if the proposed design succeeds in achieving flexible link speed packet classification. At the same time, new rules configuration must be set at runtime without loosing incoming packets. Therefore, studying counter values associated to the different rules allows us to have a glimpse of the traffic seen by the probe.

Figure 5.11 shows the evolution of the counter values over time for the rule 1

matching all the packets of the incoming traffic. Figure 5.12 is focused on the counters on the FPGA for the other rules, each matching a subset of the packets. Some packet flows have few packets compared to the global traffic and the count is not visible on Figure 5.12a. So Figure 5.12b presents a zoomed version of the previous one. From figures, it can be observed that the probe is able to track a wide variety of amount of traffic. It is possible to monitor a full 40 Gbps link of 64-byte packets, nearly 60 Mpps as shown in Figure 5.11. Figure 5.12 demonstrates that, at the same time, the probe is able to extract multiple smaller flows of packets from the global traffics, and even a small flow like the one matched by rule 4 of 30000 pps, representing 0.05% of the total traffic. It offers the possibility to concurrently maintain coarse-grained and fine-grained packet processing. With subsampling, a probe has a very low probability to watch packets of small flows. Thus, seeing all the traffic removes a security flaw. Indeed, a subtle attack can be covered by a massive but simple one.

An omniscient probe is a fine tool, but configuration adaptation is the key to refine the observation depending on the traffic distribution. Vertical dashed lines in Figure 5.12 represent moments when a new configuration is sent to add or remove new filtering rules following the detection of abnormal packets by the operator. The rule set is modified on the rule processors 5, 6 and 7 to watch complementary packets. Holes in the counts show exactly when these rule processors are not configured, in the opposite to those always in place in the rule set.

Traffic spike α triggers a software anomaly detector which automatically asks for 2 rules to be added to follow the evolution of ICMP traffic. These 2 rules are, in BPF syntax [Bpf]:

5a: `icmp`

6a: `icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply`

These rules are removed when counters show a reduced abnormal activity. Similarly, spike β leads to the automatic insertion of 3 new rules monitoring DNS traffic based on an anomaly detection by software:

5b: `udp port 53`

6b: `udp port 53 and ip dst 192.168.1.20`

7b: `udp port 53 and ip[2:2] >= 700`

It can be observed in Figure 5.12 that the configuration time of one rule in hundreds of microseconds is seen as instant. The configuration length is mainly due to the NetFPGA driver performance. With a more efficient driver, this time can be greatly reduced. It is worth noting that the modification of the rule set on the probe has no impact on the filters already in place. Thus, the design offers a proper way of rapid adaptation to incoming traffic without loss of packets.

To monitor the same type of traffic, a classic FPGA approach will specialize the design to meet the wanted application. The addition of new fields to the monitoring rules list leads to the generation of a new design and the reconfiguration of the FPGA. The proposed design succeeds in building an adaptive probe without packet loss. The

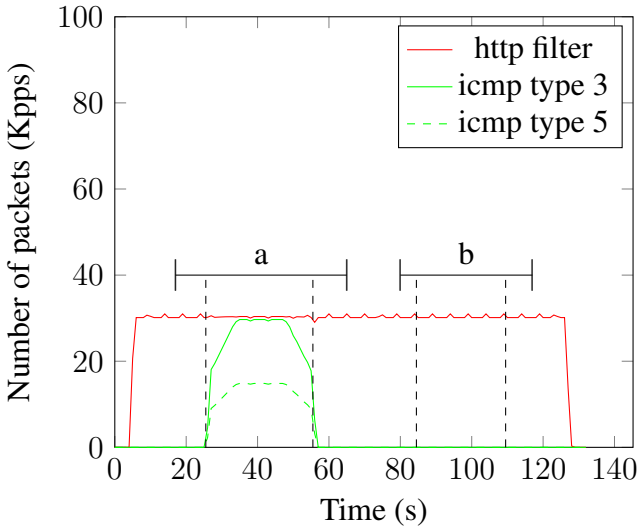


Figure 5.13 – Counter values of software filtering

different levels of stress do not impact performance and rapid modifications of the setup. Thus, it is validated that the proposed design allows to track incoming packets under any condition.

Figure 5.13 shows the counter values of packet flows composed of packet forwarded from the FPGA to the host. These packets are matched by the rules 4 and 6a. The rule 4 results lead to the observation that the hardware part tracks packets with more precision. Indeed, the evolution of values in Figure 5.12b is smoother than the ones in Figure 5.13. Approximations in Figure 5.13 are created by software disruptions during packet processing.

The rule 6a extracts ICMP packets which are not ping or reply from the principal traffic. Figure 5.13 shows a simple example of advanced processing. An advanced rule divides ICMP packets transferred to the host into ICMP packets of type 3 and type 5.

More than a flexible probe, the design offers a tool for puncturing precise traffic flows from the global traffic and refining the monitoring in software. With an adapted application driving the probe, monitoring the traffic can evolve following the distribution of the traffic. This highly dynamic rule set management enables the execution of precise packet processing applications on high data rate traffic. This performance is unmatched by other solutions in the literature. Alongside a relatively limited configurability, hardware only solutions are not able to define complex rules in software. HyPaFilter+ [Fie+17] considers a similar approach, but the fixed hardware rules limits the performance. This is a problem for very high speed networks with variable traffic where even a fraction of traffic can overwhelm the software.

In addition, this classification architecture works at the frequency of the NetFPGA based design, which is 180 MHz. However, the packet parser unties the packet from the chunk datapath with the production of one set of feature results at each packet. As the fully pipelined design is capable of processing a new packet at each clock cycle, it would be possible to process up to 180 Mpps with a packet parser fast enough, corresponding to 120 Gbps of 64-byte packets. The classification design could be working at smaller frequency, releasing some of the placement constraints.

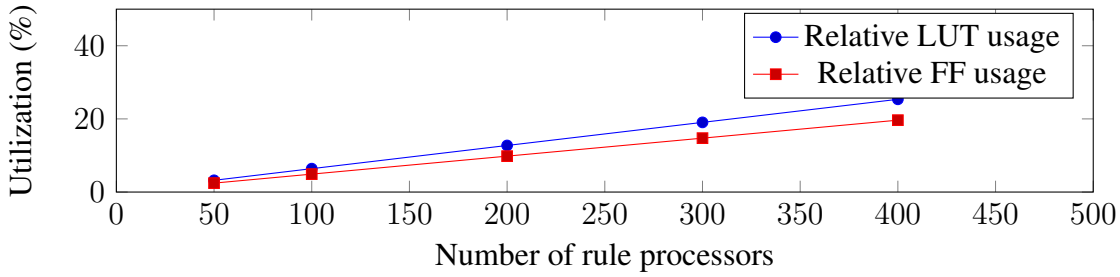


Figure 5.14 – Resource consumption for different rule processors number with 4-depth rules

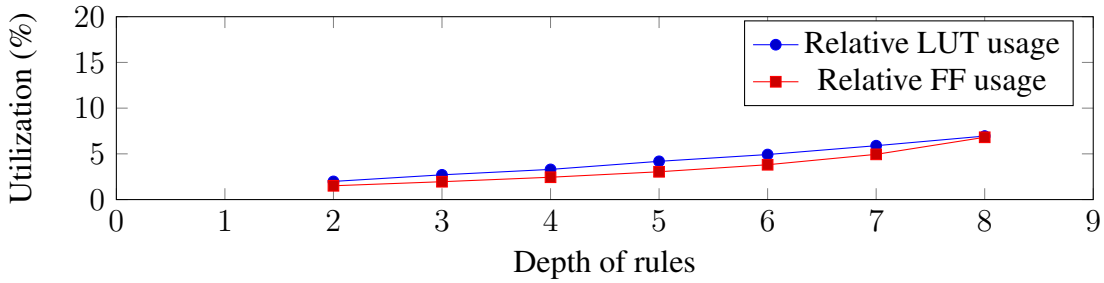


Figure 5.15 – Resource consumption for different rule depths with 50 rule processors

5.4.8 Rule processor study

The finite number of resources on an FPGA inevitably limits the scalability of the design. Meaningful elements of the design are the number of parallel processors and the rule depth capability of the processors. These parameters determine the number of rules the classifier can support as well as the length of these rules.

Figures 5.14 and 5.15 summarize the impact of the variation of such parameters on the occupation space of rule processors relative to the resource of Xilinx XC7VX690T FPGA. Figure 5.14 is expending the number of concurrent rule processors until the implemented design is not able to sustain the required timing target. Rule processors are sized to process rules of a depth of 4 with 32-bit inputs. It can be observed that the resource consumption is growing linearly with the number of rule processors. In Figure 5.15, occupation results are shown for multiple depth configurations of 50-rule processors. While the number of LUTs is increasing linearly, the number of FFs is increasing slightly faster.

The obtained results can be used to extract the resource consumption for one rule, and therefore to estimate the space occupation for a target configuration. This is the key to achieve an efficient hardware and software partition of the application. The purpose of this architecture is designed to cut the number of packets that an associated software has to process. Therefore, rules of limited length can be executed in hardware to avoid resource overconsumption, as larger rules being are allowed by the following software processing.

Table 5.2 shows the comparison between the proposed solution and multiple classification solutions presented in [FHS17]. All these results are obtained after implementation on the same platform, the NetFPGA SUME, and on the same FPGA. These

	proposed	MPFC [FHS17]	StrideBV [FHS17]	TCAM [FHS17]
LUTs/rule	280	2.01	11.65	65.82
FFs/rule	426	3.46	13.70	193.40
BRAM/rule	0.00	0.00	0.27	0.00

Table 5.2 – Comparison of per-rule resource consumption

results are focused on the per rule consumption of FPGA resources. The designs in [FHS17] are operating on a static input key representing the standard network 5-tuple composed of 104 bits. For the sake of fair comparison, the rule processors have been tuned to be able to support rule classification on equivalent parameters. Thus, processors take 4 inputs of 32 bits for a total of 128 bits, approximating the key of other solutions.

The observation of Table 5.2 highlights the cost of the flexibility brought by the proposed solution. Compared to other solutions, presented rule processors are able to process a dynamic rule set as well as a dynamic field set instead of using a static search fixed key. Moreover, the proposed solution is using extended matching operations leading to more diverse packet matching but more resource consumption too. Less rule integration is compensated by the possible high collaboration with the software brought by the flexibility. When integrated inside an end-user probe, this architecture offers a plug-in to control packets from a software application at high data rate. Such processing block is not designed to support a very large rule set sparsely updated, but aims to be dynamically used to refine information on the traffic while reducing the quantity traffic to process in the software part.

In opposition to other solutions, this approach defines a packet classifier with a unique flexibility combining a flexible packet parser and rule processors. The processor-oriented architecture allows to adapt the processors to provide extended or reduced classification capabilities at a cost of resources or to save resources. Thus, the architecture can be tuned according to the needs.

Finally, the addition of a global controller can optimize the usage of processors. Currently, one processor is used for one rule for every packets. However, headers of incoming packets can be used to tune processors configuration. Such an architecture would be able to process rules for different packets on the same processor.

5.5 Conclusion

This chapter is focused on the validation of the concepts introduced in the previous chapters. Two experimental applications are used to test the functionality and the performance of the designs. These applications observe constraints not met by the solutions in the literature. Tests are executed with a test-bed capable of dynamically saturating a 40 Gbps link.

The first application uses only the proposed packet parser to create a simple forensic application. Thanks to the flexibility of the packet parser, it is possible to refine monitored features to have an accurate view of the traffic composition when an anomaly is detected. The forensic probe is able to detect different volumetric attacks

on different packet types on a saturated 40 Gbps link. This capability can not be acquired with any other packet parser of the literature.

The second application combines the packet parser, an interconnection network and rule processors to create an advanced solution for hardware and software packet classification. This unique classifier makes possible to update the field set and the rule set at runtime without packet loss. The software-oriented usage thanks to the API allows to define the rule as any software classifier. While the test is done on 40 Gbps traffic, the classification design is capable to process 120 Gbps of 64-byte packets provided faster interfaces. The design of this innovative packet classifier has led to a publication [CJ+18], which is currently under review.

These tests validate the proposed approach of a flexible probe based on mixed hardware and software processing. With proper base blocks, this paradigm allows the construction of innovative probes capable of:

- sustaining high throughput links at line rate without packet loss,
- executing flexible processing,
- controlling the amount of data processed by the CPU,
- enabling active and reactive software processing.

The unique combination of all these properties makes the designed probes content-aware with adaptation and resilience to a large panel of traffic. From large volumetric to small covered attacks, common networking operations can keep up with the traffic while avoiding any security flaws, as never done before.

Chapter 6

Conclusion

6.1 Main contributions

This thesis aims at studying packet processing solutions meeting current and future computer networks requirements. The quick expansion of networks usage requires cutting edge networking elements able to sustain at the same time the flexibility required by SDN and the high data rate offered by network links. Current networking solutions are facing a real challenge to provide a secure and manageable infrastructure to end-users.

6.1.1 High performance monitoring systems

While optimizations can be done at algorithmic level, the performance of a networking application is mainly linked to the hardware platform used. Each system offers different advantages and drawbacks. In order to provide fitting solutions to modern network requirements, the following criteria must be met, in order of importance:

- high throughput (from 10 Gbps up to 1 Tbps),
- high flexibility (SDN compliance),
- low reaction time (at link time scale),
- high scalability (from a 10 Gbps link to many links of 100 Gbps),
- high accessibility (for network engineers),
- high portability (not hardware vendor specific).

I studied already existing solutions in the light of these minimal requirements.

Commodity hardware Commodity hardware is the most common platform used for packet processing. Indeed, CPU-based platforms are widespread due to a large accessibility, flexibility and portability. Applications can be easily updated and a large number of tools exists to perform networking actions. The platform is extendable with, for instance, the addition of GPUs if more processing power is needed.

However, this genericity of commodity hardware limits the throughput of processed packets. 20 Gbps and more per CPU chip is nearly impossible to obtain, except for some highly optimized solutions. Such a solution removes all the flexibility aspects of software, especially accessibility and portability, to build a solution which can be considered as no longer commodity hardware. Moreover, no off-the-shelf motherboard exists with more than two CPU sockets, so a solution based on commodity hardware is not scalable for larger links.

Specialized hardware As is the case with most applications, performance issues can be tackled by adopting ASIC devices. They take advantage of high parallelism and processing specialization to accelerate specific applications. However, the lack of flexibility of ASIC devices restricts their usage to specialized accelerators. They can only perform operations included at design time, which means usually months or years prior to actual usage.

The use of specialized processor (NPUs) combines the general approach of CPU with ASICs used as coprocessors. Despite good performance, these elements are vendor specific, limiting portability and accessibility, and cover only a fraction of networking operations. When incoming traffic or end-user applications do not match the foreseen accelerated blocks, NPUs are as good as CPUs for packet processing. An NPU combines both the drawbacks of CPU and ASIC.

Reconfigurable hardware FPGAs offer a similar approach as ASICs with high parallelism and specialization, but they trade some performance for reconfigurability. They can still provide high processing power, even if at lower frequencies and with lower density than ASIC. Yet, they can be reconfigured "in the field", which offers the possibility to modify existing operations. This flexibility has an higher cost than software flexibility, mainly with high design and compilation time, and processing interruption during reconfiguration. The low accessibility of FPGA design can be compensated by the existence of assisting tools and the portability of designs ensured by HDL languages.

The good ratio between performance and flexibility and recent developments in FPGA technology, with larger chips and partial reconfiguration, have changed the status of FPGAs from ASIC prototyping elements to end-user products. In networking, this increased interest has lead to the development of smart NICs, which are NICs enhanced with the integration of an FPGA chip. These new NICs have the ability to offload packet processing for CPU at the link level. Scalability is ensured by the development of new chips with increased input and output bandwidth.

All these solutions are not able to answer alone the constraints of live traffic processing. The arrival of smart NICs offers the possibility to combine FPGA and commodity hardware. These new solutions can counterbalance hardware and software weaknesses to achieve results inaccessible until now.

6.1.2 Hardware and software design

The key of smart NICs integration into commodity hardware is the mitigation of the FPGA drawbacks by the software. Current solutions try to improve the common design

flow of FPGAs. Reduced development time with high level tools and partial reconfiguration are used for the production of highly specialized architectures. However, the implementation process of an architecture on an FPGA is time-consuming, can fail if the design does not fit on the targeted FPGA and is mandatory.

Unlike other solutions, I considered a different approach from the common top down design. Instead of using the FPGA as the main flexibility provider in the hardware, a parameterized static architecture is used. With a well partitioned application and configurable hardware blocks, it is possible to use a static architecture as a programmable accelerator with parameters modification at runtime. Runtime flexibility is thus provided by parameters, while updatability is provided by the reconfigurable nature of FPGA. This division aims at using the strengths of both FPGA and CPU to offer a compromise between high flexibility and very high performance. I defined two division concepts.

Feed forward The software is responsible for the definition of configuration parameters and the hardware executes accelerated processing according to these parameters. I demonstrated the relevance of this paradigm with the design of a synthetic traffic generator. The software part takes advantage of common networking tools to create packet streams to replay and dynamically transfers the corresponding configuration parameters. The hardware part ensures the generation of the requested traffic at 40 Gbps, and even 160 Gbps, even with the smallest Ethernet packets.

Feedback loop In this paradigm, software processing elements receive information from the hardware processing elements. The constant flow of information gives the knowledge about the state of the link. Thus, with a runtime configurable hardware architecture, a monitoring probe can be continuously adapted to the incoming traffic.

Contributions on this hardware and software co-design paradigm have led to the following publications:

- F. Cornevaux-Juignet et al. “Sonde matérielle-logicielle de surveillance de trafic très haute performance”. In: *Colloque du GDR SOC2*. 2017
- F. Cornevaux-Juignet et al. “Combining FPGAs and processors for high-throughput forensics”. In: *2017 IEEE Conference on Communications and Network Security (CNS)*. 2017, pp. 388–389. DOI: 10.1109/CNS.2017.8228684

6.1.3 Innovative flexible packet parser hardware architecture

The existence of a flexible monitoring probe is directly linked to the existence of flexible hardware architectures. I studied the design of a flexible front-end for packet processing, a packet parser. This element is split in two parts.

Flexible packet parser This part executes the decapsulation of incoming packets. The successive header layers of a packet are removed and features of interest for the following processing are extracted. The innovative proposed architecture enables the live configuration of the processed headers as well as the extracted features. The information collected on packets is adapted to the current needs of applications.

Efficient interconnection network The packet parser alone is not sufficient to distribute the features to the different processing elements of the next processing steps. Duplicating the extraction of a feature for multiple processing elements is not efficient in terms of resources. The development of a simple asymmetric multistage interconnection network based on Benes networks validated the advantage of using an interconnection network for the distribution of features. It is possible to connect more outputs with less resource consumption. Further improvement has been studied with the Clos asymmetric interconnection networks, but this is still in its early phase, and need some further work.

Contributions on the implementation of the flexible packet parser have led to the following publication:

- F. Cornevaux-Juignet et al. "Open-source flexible packet parser for high data rate agile network probe". In: *Workshop on Network and Cloud Forensics hosted by 2017 IEEE Conference on Communications and Network Security (CNS)*. 2017, pp. 610–618. DOI: 10.1109/CNS.2017.8228685

6.1.4 Validation of the flexible approach

For a complete performance evaluation of the proposed approach, I designed a testbed for the validation of networking devices. This platform is based on NetFPGA SUME boards working at 40 Gbps with a generator capable of generating traffic with a controlled packet composition. Two simple networking applications were developed to test both the throughput and the functionality of the previously presented architectures.

In the first use case, the packet parser is completed with counting and anomaly detection modules to create an application for network forensics. Thanks to the parser flexibility, the software application is able to monitor the traffic with incremental refinement of the watched protocols to detect a volumetric attack on the link. This unique functionality ensures the capability of the probe to process packets with any protocol composition, as in software, at the link rate.

The second use case considers the addition of rule processors to the packet parser to produce a highly flexible hardware packet classifier. With the definition of a rule set translated in configuration parameters by a driver, applications can precisely select specific packets inside the full link traffic. It is possible to dynamically change the rule set as well as the feature set at runtime, as never seen in the literature. In addition, the packet selection can be further refined with a second layer of complex classification in software. This packet classifier is the base for dynamic packet analysis at 40 Gbps and beyond.

Contributions on the implementation of the packet classifier have led to the following publication:

- **submitted:** F. Cornevaux-Juignet et al. "High data rate dynamic rule processor for live network packet analysis". In: *IEEE/ACM Transactions on Networking* (2018)

The recent evolution of the amount of generated network traffic and its diversity left common networking solutions helpless to accurately handle the traffic. Costly or

poorly adapted architectures are commonly used to cope with the required processing power. Even if the latest solutions tried to tackle the increase of traffic throughput, none was focused on the live adaptation of processing to the traffic content. Therefore, they were unable to provide accurate and reliable network management and security. I provided guidelines for hardware and software collaboration and the first basic blocks to start building the next generation of smart NICs.

6.2 Perspectives

Because of the growing importance of computer networks in the everyday life, the quantity of traffic is constantly increasing. The diversity of the traffic is maintained by a wide number of network services. Moreover, it is not possible to predict the leading services of tomorrow and their impact on the network traffic. Today networking devices must be able to foresee this growth of traffic and must be adapted for the new applications.

The work proposed in this manuscript validates the design of flexible architectures for high-speed network at 40 Gbps. As the scalability of the system is of prime interest for future networks, it is planned to upgrade the test-bed to sustain 160 Gbps traffic. The packet preprocessing part, the packet parser and rule processors, must efficiently consume resources of the FPGA, saving a maximum resources for accelerated applications in order to meet the link throughput requirement. A point of optimization is the interconnection network with the study of Clos interconnection networks to reduce the number of crosspoints. Tuning the different configuration parameters would lead to the discovery of the interconnection network the most suitable for the right number of inputs and outputs. Further scalability of the designs will be studied with the collaboration of multiple parallel boards driven by one software device, and with the upgrade of smart NICs equipped with 400 Gbps interfaces.

The increased complexity of the network traffic is difficult to emulate with a simple traffic generator. The proposed generator is a first step for the dynamic generation of synthetic traffic. Dynamic traffic can be generated at 40 Gbps, and even 160 Gbps, but the architecture can be improved to generate more complex traffic profiles for a better approximation of real traffic. First, reducing the quantity of data to transmit between the host and the FPGA would allow more frequent parameters modifications. Second, the transmission of a new configuration still takes hundreds of microseconds and is principally limited by the PCIe driver and hardware blocks supplied by the NetFPGA project. More optimized PCIe communications would allow a closer collaboration between hardware and software with more frequent replayed packet stream modification.

Reconfigurable hardware offers a compromise for line rate packet processing at 40 Gbps and beyond as well as flexibility. FPGAs have the potential to solve the issue of high performance networking while keeping a similar flexibility as current platforms. This potential have led multiple major network actors, like Microsoft or OVH, to use FPGAs in their infrastructures. The emergence of smart NICs has changed the status of FPGA from prototyping platform to end-user product, stimulating the market and the research.

A significant advantage is that the evolution of FPGA technology opens networking possibilities. For example, with partial reconfiguration and bitstream relocation,

it would be easy integrate to change the behavior of established architectures with the integration of new accelerated applications. Moreover, as FPGAs are not dedicated to networking, any improvement created for other fields could benefit networking devices.

Nonetheless, the adoption of FPGA-based devices by the community would require an easy access from the currently widespread networking tools. Despite the promising performance of this work for high-speed packet analysis, a software capture framework is necessary to ease the accessibility for end-users. The addition of a control GUI would allow an easier manipulation of the monitoring probe. For instance, the creation of a plug-in for high-speed packet analysis in Wireshark would be an immense feature for the adoption.

We could wonder if the huge growth of network will outmatch the processing capabilities of FPGA-based smart NICs. As any other platform is already obsolete, solutions to face the challenges of tommorow's networks, a diverse traffic and terabits per second throughput, will only come from the best exploitation of the potential of reconfigurable devices.

Glossary

ALU An Arithmetic Logic Unit (ALU) is a fundamental combinational block of computing circuits performing configurable base arithmetic and logic operations for a larger processing element. 89, 90

API An Application Programming Interface (API) is a specified interface a program makes available, so as to communicate with other programs. It can be made of function calls, network requests.... 4, 15, 19, 33, 40, 42, 62, 66, 78, 80, 83, 88, 92, 93, 100

ASIC An Application-Specific Integrated Circuit (ASIC) is designed at hardware level using basic logic and arithmetic gates to realize a specific function at very high speed. vi, vii, 7, 18–21, 26, 102

BPF Berkeley Packet Filter (BPF) is a widespread software framework for packet filtering. It contains a way to collect packet and defines a language to set filtering rules. 44, 86, 90, 93, 94, 96

BRAM A Block RAM (BRAM) is a memory module directly available on an FPGA. It is smaller than external memory but more efficient than logic to locally store data. 42

CAM A Content-Addressable Memory (CAM) is an associative memory comparing the input search key against stored data and returning the matched data. 58, 86, 111

CMS A Count Min Sketch (CMS) is a probabilistic algorithm to store a list of counters in a constrained memory space. 79

CPU A Central Processing Unit (CPU) is the integrated circuit used to make all basic operations in a computer. It may contain multiple cores to be able to process multiple operations concurrently. There may also be more than one CPU cooperating in a computer to increase the parallelism. vi, vii, xii, xvi, 12–20, 25, 26, 28, 33, 37, 43, 45, 46, 48, 49, 51, 86, 92, 100–103, 110

DDoS Distributed Denial of Service (DDoS) attacks use multiple computers connected to the Internet to send more traffic to a target than it can handle, so as to make it unresponsive. 2

DDR Double Data Rate (DDR) is a way of synchronous data transfer. Transfers occur on both rising and falling edges of the clock. This allows a double transfer rate. 29

DMA Direct Memory Access (DMA) is a feature allowing hardware to access the system memory without going through the CPU. 30

DNS Domain Name Service (DNS) is a protocol that maintains an association between easy-to-remember domain names and routable IP addresses. It provides simpler addresses to contact machines. 44, 81, 83, 90, 94, 96

DoS Denial of Service (DDoS) attacks try to exhaust a target resources to make it unavailable. 2

DPI Deep Packet Inspection (DPI) is a traffic classification technique which consists in reading the full content of each received packet and check if it fits some predefined signatures. Each signature belongs to a class of applications to which the packet is then assigned. 15, 44

DRAM A Dynamic Random Access Memory (DRAM) stores bits in capacitors. Memory must be periodically refreshed because capacitors leak and eventually loose stored data. 29, 111

FF A Flip-FLop (FF) is a bistable multivibrator used in clocked circuits to store a bit of data. The output is only changed at one of the clock edges. 64, 71, 79, 94, 98

FIFO First In First Out (FIFO) is a method used to queue data. All received data items are stored in an ordered way. The read item is always the one that was stored first. Once an item is read, it is removed from the queue first. 80

FPGA A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be configured as many times as necessary at a very low level by connecting logical gates and registers together. The main languages used to represent the configuration are VHDL and Verilog. v, vii–xiii, xvi, xix, xxi, 20–35, 37–39, 41–43, 45, 46, 48, 49, 51, 53, 55, 56, 58, 59, 63–66, 71, 74, 75, 78, 79, 83, 86–88, 90, 94, 96–99, 102, 103, 105, 106

GPGPU General Purpose Processing on Graphics Processing Unit (GPU) uses GPU to offload computation traditionaly made by CPU. The large number of parallel cores provides a mean to improve computation time. 15

GPU A Graphics Processing Unit (GPU) is a specialized integrated circuit designed for images manipulations. They are also used in other situations. They are particularly suited for highly parallel floating-point calculations on important amounts of data. vi, 12, 13, 15–17, 101

GUI A Graphical User Interface (GUI) is a communication tool between a computer and a human based on visual representations on a screen. It is the most current kind of interface used on computers. 43, 106

HDL An Hardware Description Language (HDL) is a specialized language used to describe electronic circuits. 21, 22, 31, 32, 53, 102

HLS High-Level Synthesis (HLS) is an automated process generating an hardware design from a high-level algorithmic description in common software language. 22, 23, 31

HTTP HyperText Transfer Protocol (HTTP) is the communication protocol used by the web. 81, 83

ICMP Internet Control Message Protocol (ICMP) is used by network devices to request and send status messages. It is mostly known for its "ping" feature that is made to check the responsiveness of an IP address. 44, 81, 83, 94, 95, 97

IDS An Intrusion Detection System (IDS) is an application monitoring a network to detect and mitigate malicious behaviors. 16

IoT Network of connected devices. 2, 3

IP Intellectual Property is a term used to design closed-source entities provided by third-parties in electronics. 22, 30

IP Internet Protocol (IP) is the base protocol used on the Internet. 3, 46, 52, 55, 63, 80, 81, 83, 87

IPv4 Internet Protocol version 4 (IPv4) is the current version of the protocol. It is slowly being replaced by version 6. xix, 16, 17, 25, 54, 55, 58, 61, 62, 79

IPv6 Internet Protocol version 6 (IPv6) is a new version of the protocol being currently deployed. It has not yet replaced the current version 4, but should soon. 16, 55, 59

LUT A Look-Up Table (LUT) realizes a function that takes a word made of a certain number of bits as input, and outputs another word made of another number of bits, depending only on the input. It can be configured to link each input word to any output word. 64, 71, 79, 89, 90, 94, 98

NAPI New Application Programming Interface (NAPI) is an improvement of the Linux kernel to support high-speed networking with a lighter load on the CPU. 13

NIC A Network Interface Card (NIC) is a board that can be connected to a computer to provide it a network connection. The most common wired boards use the low-level Ethernet protocol. They usually manage the lowest parts of the network stack to help the CPU. A driver is needed for the CPU to communicate with the board. vii, viii, xii, xix, 4, 11–15, 17, 25–32, 43, 45, 46, 48, 49, 51, 78, 79, 85, 86, 93, 102, 105, 106

NoC A Network on Chip (NoC) is a communication system between multiple elements on the same chip. 24, 53, 67

NPU A Network Processing Unit (NPU) is a specialized integrated circuit designed for network applications. It has a direct access to network interfaces, and some specialized instructions (CRC computation for example) make frequent operations on packets faster. 18, 19, 23, 25, 26, 102

NUMA Non Uniform Memory Access (NUMA) is a computer architecture where each processor is assigned an area of RAM with low access delays. Other areas are accessible but slower. 14

OS An Operating System (OS) is a system software providing hardware resources and software resources management. 13, 17, 44

OSI The Open System Interconnection (OSI) model is a model partitioning a communication system into abstraction layers to aim at communication devices interoperability. 8, 52

PCAP PCAP (Packet CAPture) is both an API to capture packets on a network interface, and a file format used to save a trace of captured packets. Implementations exist for Windows and Linux. 37, 38

PCI Peripheral Component Interconnect (PCI) is a standard for a local communication bus used to connect extension cards to the motherboard of a computer. 12, 25

PCIe Peripheral Component Interconnect express (PCIe) is a standard for a local serial communication bus. It is used to connect extension cards to a motherboard. xxi, 12–14, 17, 25, 29–31, 42, 80, 105

PDU A Protocol Data Unit (PDU) is a message specific to a protocol. 8

QDR Quad Data Rate (QDR) is a way of synchronous data transfer. Reads and writes are made on separated clocks, and transfers occur on both rising and falling edges of the clock. This allows to quadruple the transfer rate with interleaved cycles. 21, 29

QoS Quality of Service (QoS) is a measure of the quality assured on a netwokr link. Many parameters can be used: inter-packet delay, jitter, data rate, etc. 2, 7

RAM A Random Access Memory (RAM) is a memory with fast read and write operations at any address. The memory is volatile, which means that data is lost when power is cut. 21, 29

RDMA Remote Direct Memory Access (RDMA) is a system to transfer data directly from one device to another device without going through the CPU. 17

RSS Receive Side Scaling (RSS) is a technology used by some NICs to send received packets to the host CPU in different queues, which can be handled in parallel by the CPU. 13, 14

SAP The Service Access Point (SAP) names the direct upper-layer protocol of OSI model where the information is delivered. Its value indicates the protocol to deliver to. 53

SDN Network conception paradigm setting the separation between the data plane and the control plane to get a network abstraction. v, vi, 3, 4, 7, 12, 18, 19, 28, 32, 44, 86, 101

SFP+ An enhanced Small Form-factor Pluggable (SFP+) is a small pluggable transceiver that converts optical signal to electrical signal and vice versa. 30

SIMD Single Instruction Multiple Data (SIMD) describes a computer architecture where multiple elements simultaneously perform the same operation on different data. 16, 17

SPEC The Standard Performance Evaluation Corporation (SPEC) is an organization producing standardized performance benchmarks for computers. Obtained results are published on the organization's web site. 12

SRAM A Static Random Access Memory (SRAM) stores bits in flip-flop. In the contrary of DRAM, no periodic refresh is needed and SRAM is faster but more expensive. 29

SVM Support Vector Machine (SVM) is a supervised learning algorithm used for classification. It is based on finding hyperplanes between categories. 21

TCAM A Ternary Content-Addressable Memory (TCAM) is a CAM where the input key can have a "don't care" state. 86

TCP Transmission Control Protocol (TCP) is a transport protocol very commonly used over IP. It is designed for reliable connected data transfer. iv, 2, 46, 52, 63, 81, 83, 87, 93

UDP User Datagram Protocol (UDP) is a transport protocol very commonly used over IP. It is designed for simple data transfer. xix, 46, 52, 54, 55, 58, 62, 83, 87, 93

URL Uniform Resource Locator (URL) is a string indicating the location of a resource on a computer network. It is most commonly used to reference web pages. 25

VHDL VHSIC Hardware Description Language is an hardware description language used to describe electronic circuits. 22

Bibliography

[And+14] J. G. Andrews et al. “What Will 5G Be?” In: *IEEE Journal on Selected Areas in Communications* 32.6 (2014), pp. 1065–1082. ISSN: 0733-8716. DOI: 10.1109/JSAC.2014.2328098.

[Ant] Antichi. *OSNT*. Accessed: 2018-03-05. URL: http://osnt.org/.

[ACG12] G. Antichi, C. Callegari, and S. Giordano. “An open hardware implementation of CUSUM based network anomaly detection”. In: *Global Communications Conference (GLOBECOM), 2012 IEEE*. 2012, pp. 2760–2765. DOI: 10.1109/GLOCOM.2012.6503534.

[Ant+14] G. Antichi et al. “OSNT: open source network tester”. In: *IEEE Network* 28.5 (2014), pp. 6–12. ISSN: 0890-8044. DOI: 10.1109/MNET.2014.6915433.

[AB11] M. Attig and G. Brebner. “400 Gb/s Programmable Packet Parsing on a Single FPGA”. In: *2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. 2011, pp. 12–23. DOI: 10.1109/ANCS.2011.12.

[Bpf] *BPF filter syntax*. Accessed: 2017-12-19. URL: http://alumni.cs.ucr.edu/~marios/ethereal-tcpdump.pdf.

[BBS17] I. Benacer, F. R. Boyer, and Y. Savaria. “A high-speed traffic manager architecture for flow-based networking”. In: *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*. 2017, pp. 161–164. DOI: 10.1109/NEWCAS.2017.8010130.

[Ben62] V. E. Beneš. “Heuristic remarks and mathematical problems regarding the theory of connecting systems”. In: *The Bell System Technical Journal* 41.4 (1962), pp. 1201–1247. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1962.tb03276.x.

[BPK16] P. Benáček, V. Pu, and H. Kubátová. “P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers”. In: *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2016, pp. 148–155. DOI: 10.1109/FCCM.2016.46.

[Ben+14] P. Benáček et al. “Change-point Detection Method on 100 Gb/s Ethernet Interface”. In: *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS ’14. Los Angeles, California, USA: ACM, 2014, pp. 245–246. ISBN: 978-1-4503-2839-5. DOI: 10.1145/2658260.2661773.

[BI17] E. Bertino and N. Islam. “Botnets and Internet of Things Security”. In: *Computer* 50.2 (2017), pp. 76–79. ISSN: 0018-9162. DOI: 10.1109/MC.2017.62.

[Bia+06] A. Bianco et al. “Boosting the performance of PC-based software routers with FPGA-enhanced network interface cards”. In: *2006 Workshop on High Performance Switching and Routing*. 2006, 6 pp.–. DOI: 10.1109/HPSR.2006.1709693.

[BAB15] A. Bitar, M. S. Abdelfattah, and V. Betz. “Bringing programmability to the data plane: Packet processing with a NoC-enhanced FPGA”. In: *2015 International Conference on Field Programmable Technology (FPT)*. 2015, pp. 24–31. DOI: 10.1109/FPT.2015.7393125.

[Bon+14] Nicola Bonelli et al. “A Purely Functional Approach to Packet Processing”. In: *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS ’14. Los Angeles, California, USA: ACM, 2014, pp. 219–230. ISBN: 978-1-4503-2839-5. DOI: 10.1145/2658260.2658269.

[Bon+12] Nicola Bonelli et al. “On Multi—gigabit Packet Capturing with Multi—core Commodity Hardware”. In: *Proceedings of the 13th International Conference on Passive and Active Measurement*. PAM’12. Vienna, Austria: Springer-Verlag, 2012, pp. 64–73. ISBN: 978-3-642-28536-3. DOI: 10.1007/978-3-642-28537-0_7.

[Bra+13] L. Braun et al. “Adaptive load-aware sampling for network monitoring on multicore commodity hardware”. In: *2013 IFIP Networking Conference*. 2013, pp. 1–9.

[Bre09] G. Brebner. “Packets everywhere: The great opportunity for field programmable technology”. In: *2009 International Conference on Field-Programmable Technology*. 2009, pp. 1–10. DOI: 10.1109/FPT.2009.5377604.

[Bre15] G. Brebner. “Programmable hardware for high performance SDN”. In: *2015 Optical Fiber Communications Conference and Exhibition (OFC)*. 2015, pp. 1–3. DOI: 10.1364/OFC.2015.Th3J.3.

[BJ14] G. Brebner and W. Jiang. “High-Speed Packet Processing using Reconfigurable Computing”. In: *IEEE Micro* 34.1 (2014), pp. 8–18. ISSN: 0272-1732. DOI: 10.1109/MM.2014.19.

[CM97] Chihming Chang and Rami Melhem. “Arbitrary Size Benes Networks”. In: *Parallel Processing Letters* 7 (1997), pp. 279–284.

[Che+08] S. Che et al. “Accelerating Compute-Intensive Applications with GPUs and FPGAs”. In: *2008 Symposium on Application Specific Processors*. 2008, pp. 101–107. DOI: 10.1109/SASP.2008.4570793.

[CP15] R. Chen and V. K. Prasanna. “Automatic generation of high throughput energy efficient streaming architectures for arbitrary fixed permutations”. In: *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. 2015, pp. 1–8. DOI: 10.1109/FPL.2015.7293944.

[CP17] R. Chen and V. K. Prasanna. “Computer Generation of High Throughput and Memory Efficient Sorting Designs on FPGA”. In: *IEEE Transactions on Parallel and Distributed Systems* 28.11 (2017), pp. 3100–3113. ISSN: 1045-9219. DOI: 10.1109/TPDS.2017.2705128.

[Cil18] Cilium. *Why is the kernel community replacing iptables with BPF?* Accessed: 2018-04-27. 2018. URL: https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables/.

[Cisa] Cisco. *Cisco Visual Networking Index*. Accessed: 2018-02-02. URL: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html.

[Cisb] Cisco. *Ethernet documentation*. Accessed: 2018-02-13. URL: http://docwiki.cisco.com/wiki/Ethernet_Technologies.

[Cisc] Cisco. *Networking basic documentation*. Accessed: 2018-02-13. URL: http://docwiki.cisco.com/wiki/Internetworking_Basics.

[Clo53] C. Clos. “A study of non-blocking switching networks”. In: *The Bell System Technical Journal* 32.2 (1953), pp. 406–424. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[Com04] Douglas Comer. “Network Processors: Programmable Technology for Building Network Systems”. In: *The Internet Protocol Journal* 7.4 (2004). URL: https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-30/network-processors.html.

[CJ+17a] F. Cornevaux-Juignet et al. “Combining FPGAs and processors for high-throughput forensics”. In: *2017 IEEE Conference on Communications and Network Security (CNS)*. 2017, pp. 388–389. DOI: 10.1109/CNS.2017.8228684.

[CJ+18] F. Cornevaux-Juignet et al. “High data rate dynamic rule processor for live network packet analysis”. In: *IEEE/ACM Transactions on Networking* (2018).

[CJ+17b] F. Cornevaux-Juignet et al. “Open-source flexible packet parser for high data rate agile network probe”. In: *Workshop on Network and Cloud Forensics hosted by 2017 IEEE Conference on Communications and Network Security (CNS)*. 2017, pp. 610–618. DOI: 10.1109/CNS.2017.8228685.

[CJ+17c] F. Cornevaux-Juignet et al. “Sonde matérielle-logicielle de surveillance de trafic très haute performance”. In: *Colloque du GDR SOC2*. 2017.

[CJa] Franck Cornevaux-Juignet. *Packet classifier sources*. URL: https://redmine.telecom-bretagne.eu/projects/cyberthd_packetclassifier.

[CJb] Franck Cornevaux-Juignet. *Packet parser sources*. URL: https://redmine.telecom-bretagne.eu/projects/\\cyberthd_packetparser.

[Der+14] L. Deri et al. “nDPI: Open-source high-speed deep packet inspection”. In: *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. 2014, pp. 617–622. DOI: 10.1109/IWCMC.2014.6906427.

[Dyn] Dyn. *Dyn Mirai attack*. Accessed: 2017-02-27. URL: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[FPG17] Intel FPGA. *Intel Stratix 10 Device Datasheet*. 2017. URL: https://www.altera.com/documentation/mcn1441092958198.html.

[Fen81] Tse yun Feng. “A Survey of Interconnection Networks”. In: *Computer* 14.12 (1981), pp. 12–27. ISSN: 0018-9162. DOI: 10.1109/C-M.1981.220290.

[FHS17] A. Fiessler, S. Hager, and B. Scheuermann. “Flexible line speed network packet classification using hybrid on-chip matching circuits”. In: *2017 IEEE 18th International Conference on High Performance Switching and Routing (HPSR)*. 2017, pp. 1–8. DOI: 10.1109/HPSR.2017.7968678.

[Fie+16] A. Fiessler et al. “HyPaFilter - A versatile hybrid FPGA packet filter”. In: *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. 2016, pp. 25–36. DOI: 10.1145/2881025.2881033.

[Fie+17] A. Fiessler et al. “HyPaFilter+: Enhanced Hybrid Packet Filtering Using Hardware Assisted Classification and Header Space Analysis”. In: *IEEE/ACM Transactions on Networking* 25.6 (2017), pp. 3655–3669. ISSN: 1063-6692. DOI: 10.1109/TNET.2017.2749699.

[Fir16] Daniel Firestone. "SmartNIC: Accelerating Azure's Network with FPGAs on OCS servers". In: *OCP U.S. SUMMIT 2016*. San Jose, CA, 2016. URL: http://files.opencompute.org/oc/public.php?service=files\&t=5803e581b55\\e90e51669410559b91169\&download\&path=//SmartNIC\%20OCP\%2020\\16.pdf.

[For+13] M. Forconesi et al. "Accurate and flexible flow-based monitoring for high-speed networks". In: *2013 23rd International Conference on Field programmable Logic and Applications*. 2013, pp. 1–4. DOI: 10.1109/FPL.2013.6645557.

[Fou] The Linux Foundation. *DPDK*. Accessed: 2018-02-15. URL: http://www.dpdk.org/.

[Fou16] The Linux Foundation. *NAPI*. 2016. URL: https://wiki.linuxfoundation.org/networking/napi.

[Fri+13] Stepan Friedl et al. *Designing a Card for 100 Gb/s Network Monitoring*. Tech. rep. CESNET, 2013.

[Gal+15] S. Gallenmüller et al. "Comparison of frameworks for high-performance packet IO". In: *Architectures for Networking and Communications Systems (ANCS), 2015 ACM/IEEE Symposium on*. 2015, pp. 29–38. DOI: 10.1109/ANCS.2015.7110118.

[GJP14] T. Ganegedara, W. Jiang, and V. K. Prasanna. "A Scalable and Modular Architecture for High-Performance Packet Classification". In: *IEEE Transactions on Parallel and Distributed Systems* 25.5 (2014), pp. 1135–1144. ISSN: 1045-9219. DOI: 10.1109/TPDS.2013.261.

[GP12] T. Ganegedara and V. K. Prasanna. "StrideBV: Single chip 400G+ packet classification". In: *2012 IEEE 13th International Conference on High Performance Switching and Routing*. 2012, pp. 1–6. DOI: 10.1109/HPSR.2012.6260820.

[Gar+12] J. J. Garnica et al. "A FPGA-based scalable architecture for URL legal filtering in 100GbE networks". In: *2012 International Conference on Reconfigurable Computing and FPGAs*. 2012, pp. 1–6. DOI: 10.1109/ReConFig.2012.6416719.

[Gib+13] G. Gibb et al. "Design principles for packet parsers". In: *Architectures for Networking and Communications Systems*. 2013, pp. 13–24. DOI: 10.1109/ANCS.2013.6665172.

[Goo] Google. *Google supercharges machine learning tasks with TPU custom chip*. Accessed: 2018-02-19. URL: https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html.

[GAV14] T. Groléat, M. Arzel, and S. Vaton. "Stretching the Edges of SVM Traffic Classification With FPGA Acceleration". In: *IEEE Transactions on Network and Service Management* 11.3 (2014), pp. 278–291. ISSN: 1932-4537. DOI: 10.1109/TNSM.2014.2346075.

[Gro+13] Tristan Groléat et al. “Flexible, Extensible, Open-source and Affordable FPGA-based Traffic Generator”. In: *Proceedings of the First Edition Workshop on High Performance and Programmable Networking*. HPPN ’13. New York, New York, USA: ACM, 2013, pp. 23–30. ISBN: 978-1-4503-1981-2. DOI: 10.1145/2465839.2465843.

[GM01] P. Gupta and N. McKeown. “Algorithms for packet classification”. In: *IEEE Network* 15.2 (2001), pp. 24–32. ISSN: 0890-8044. DOI: 10.1109/65.912717.

[HBS15] S. Hager, D. Bendyk, and B. Scheuermann. “Partial reconfiguration and specialized circuitry for flexible FPGA-based packet processing”. In: *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 2015, pp. 1–6. DOI: 10.1109/ReConFig.2015.7393333.

[Hag+14] S. Hager et al. “MPFC: Massively Parallel Firewall Circuits”. In: *39th Annual IEEE Conference on Local Computer Networks*. 2014, pp. 305–313. DOI: 10.1109/LCN.2014.6925785.

[Han+10] Sangjin Han et al. “PacketShader: A GPU-accelerated Software Router”. In: *SIGCOMM Comput. Commun. Rev.* 40.4 (Aug. 2010), pp. 195–206. ISSN: 0146-4833. DOI: 10.1145/1851275.1851207.

[Hen06] John L. Henning. “SPEC CPU2006 Benchmark Descriptions”. In: *SIGARCH Comput. Archit. News* 34.4 (Sept. 2006), pp. 1–17. ISSN: 0163-5964. DOI: 10.1145/1186736.1186737.

[Hwa03] F. K. Hwang. “A survey of nonblocking multicast three-stage Clos networks”. In: *IEEE Communications Magazine* 41.10 (2003), pp. 34–37. ISSN: 0163-6804. DOI: 10.1109/MCOM.2003.1235592.

[Hwa05] F. K. Hwang. “A unifying approach to determine the necessary and sufficient conditions for nonblocking multicast 3-stage Clos networks”. In: *IEEE Transactions on Communications* 53.9 (2005), pp. 1581–1586. ISSN: 0090-6778. DOI: 10.1109/TCOMM.2005.852839.

[Inta] Intel. *IXP4XX Product Line of Network Processors*. Accessed: 2018-02-19. URL: https://www.intel.com/content/www/us/en/intelligent-systems/previous-generation/intel-ixp4xx-intel-network-processor-product-line.html.

[Int16] Intel. *Intel 82599 10 GbE Controller Datasheet*. 2016. URL: https://www.intel.com/content/www/us/en/embedded/products/networking/82599-10-gbe-controller-datasheet.html.

[Intb] Intel. *Intel FPGA stratix 10 family*. Accessed: 2017-02-21. URL: https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html.

[Intc] Intel. *Intel Processor E5-2699A v4*. Accessed: 2018-02-15. URL: https://www.intel.com/content/www/us/en/products/processors/xeon/e5-processors/e5-2699a-v4.html.

[Int17] Intel. *PCI Express High Performance Reference Design*. 2017. URL: https://www.altera.com/en_US/pdfs/literature/an/an456.pdf.

[Jam+12] Muhammad Asim Jamshed et al. "Kargus: A Highly-scalable Software-based Intrusion Detection System". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 317–328. ISBN: 978-1-4503-1651-4. DOI: 10.1145/2382196.2382232.

[JK10] A. Jastrzębski and M. Kubale. "Rearrangeability in multicast clos networks is np-complete". In: *2010 2nd International Conference on Information Technology, (2010 ICIT)*. 2010, pp. 183–186.

[Jun] Juniper. *EX Series Ethernet Switches*. Accessed: 2018-02-19. URL: https://www.juniper.net/us/en/products-services/switching/ex-series/.

[KPK14] L. Kekely, V. Puš, and J. Kořenek. "Software Defined Monitoring of application protocols". In: *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*. 2014, pp. 1725–1733. DOI: 10.1109/INFOCOM.2014.6848110.

[Kek+14] L. Kekely et al. "Fast lookup for dynamic packet filtering in FPGA". In: *17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*. 2014, pp. 219–222. DOI: 10.1109/DDECS.2014.6868793.

[Kek+16] L. Kekely et al. "Software Defined Monitoring of Application Protocols". In: *IEEE Transactions on Computers* 65.2 (2016), pp. 615–626. ISSN: 0018-9340. DOI: 10.1109/TC.2015.2423668.

[Kim+15] Joongi Kim et al. "NBA (Network Balancing Act): A High-performance Packet Processing Framework for Heterogeneous Processors". In: *Proceedings of the Tenth European Conference on Computer Systems*. EuroSys '15. Bordeaux, France: ACM, 2015, 22:1–22:14. ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741969.

[Lal+16] A. Lalevée et al. "AutoReloc: Automated Design Flow for Bitstream Relocation on Xilinx FPGAs". In: *2016 Euromicro Conference on Digital System Design (DSD)*. 2016, pp. 14–21. DOI: 10.1109/DSD.2016.92.

[Lal17] André Lalevée. "Towards highly flexible hardware architectures for high-speed data processing: a 100 Gbps network case study". PhD thesis. 2017.

[Lam13] Christoph Lameter. "NUMA (Non-Uniform Memory Access): An Overview". In: *Queue* 11.7 (July 2013), 40:40–40:51. ISSN: 1542-7730. DOI: 10.1145/2508834.2513149.

[LDC12] Maysam Lavasani, Larry Dennison, and Derek Chiou. “Compiling High Throughput Network Processors”. In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. FPGA ’12. Monterey, California, USA: ACM, 2012, pp. 87–96. ISBN: 978-1-4503-1155-7. DOI: 10.1145/2145694.2145709.

[LZB11] G. Liao, X. Znu, and L. Bnuyan. “A new server I/O architecture for high speed networks”. In: *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. 2011, pp. 255–265. DOI: 10.1109/HPCA.2011.5749734.

[MMWJJ72] G M. Masson and B W. Jordan Jr. “Generalized Multi-Stage Connection Networks”. In: 2 (Jan. 1972), pp. 191 –209.

[MP10] Guido Maier and Achille Pattavina. “Multicast three-stage Clos networks”. In: *Computer Communications* 33.8 (2010). Special Section on Hot Topics in Mesh Networking, pp. 923 –928. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2010.01.022.

[MJ93] Steven McCanne and Van Jacobson. “The BSD Packet Filter: A New Architecture for User-level Packet Capture”. In: *Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings*. USENIX’93. San Diego, California: USENIX Association, 1993, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=1267303.1267305.

[McG+10] J. McGlone et al. “Design of a flexible high-speed FPGA-based flow monitor for next generation networks”. In: *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. 2010, pp. 37–44. DOI: 10.1109/ICSAMOS.2010.5642096.

[Mel] Mellanox. *NP-5 Network Processor*. Accessed: 2018-02-19. URL: http://www.mellanox.com/related-docs/prod_npu/PB_NP-5.pdf.

[Mic] Microsoft Azure. *Maximize your VM’s Performance with Accelerated Networking – now generally available for both Windows and Linux*. Accessed: 2018-04-16. URL: https://azure.microsoft.com/en-au/blog/maximize-your-vm-s-performance-with-accelerated-networking-now-generally-available-for-both-windows-and-linux/.

[Mic17] Microsoft. *Introduction to Receive Side Scaling*. 2017. URL: https://docs.microsoft.com/fr-fr/windows-hardware/drivers/network/introduction-to-receive-side-scaling.

[MF97] N. Mir-Fakhraei. “Evaluation of a multistage switching network with broadcast traffic”. In: *Professional Program Proceedings. Electronic Industries Forum of New England*. 1997, pp. 143–147. DOI: 10.1109/EIF.1997.605383.

[Mor+15] V. Moreno et al. “Commodity Packet Capture Engines: Tutorial, Cookbook and Applicability”. In: *IEEE Communications Surveys Tutorials* 17.3 (2015), pp. 1364–1390. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2424887.

[NVI18a] NVIDIA. *CUDA C Best Practices Guide*. 2018. URL: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html.

[NVI18b] NVIDIA. *Developing a Linux Kernel Module using GPUDirect RDMA*. 2018. URL: http://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[Net] *Netfilter project*. Accessed: 2017-12-18. URL: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/.

[Nok] Nokia. *FP4: Delivering performance and capability without compromise*. Accessed: 2018-02-19. URL: https://networks.nokia.com/solutions/fp4-network-processor.

[OVHa] OVH. *Classical attack types on network link*. Accessed: 2017-02-20. URL: https://www.ovh.com/fr/anti-ddos/principe-anti-ddos.xml.

[OVH] OVH. *Comment OVH protège ses clients contre les attaques SYN flood*. Accessed: 2018-04-16. URL: https://www.ovh.com/fr/blog/comment-ovh-protege-ses-clients-contre-les-attaques-syn-flood/.

[OVHb] OVH. *OVH Mirai attack*. Accessed: 2017-02-27. URL: https://www.ovh.com/fr/a2367.goutte-ddos-n-a-pas-fait-deborder-le-vac.

[OVHc] OVH. *OVH network infrastruture*. Accessed: 2018-02-02. URL: https://www.ovh.com/fr/apropos/reseau.xml.

[PS] PCI-SIG. *PCI Special Group of Interest*. Accessed: 2018-02-15. URL: http://pcisig.com/specifications.

[Pan+17] V. Pant et al. “Efficient Neural Computation on Network Processors for IoT Protocol Classification”. In: *2017 New Generation of CAS (NGCAS)*. 2017, pp. 9–12. DOI: 10.1109/NGCAS.2017.55.

[PSS16] Nisha Panwar, Shantanu Sharma, and Awadhesh Kumar Singh. “A survey on 5G: The next generation of mobile communication”. In: *Physical Communication* 18 (2016). Special Issue on Radio Access Network Architectures and Resource Management for 5G, pp. 64 –84. ISSN: 1874-4907. DOI: https://doi.org/10.1016/j.phycom.2015.10.006. URL: http://www.sciencedirect.com/science/article/pii/S1874490715000531.

[PKK14] V. Puš, L. Kekely, and J. Kořenek. “Design methodology of configurable high performance packet parser for FPGA”. In: *17th International Symposium on Design and Diagnostics of Electronic Circuits Systems*. 2014, pp. 189–194. DOI: 10.1109/DDECS.2014.6868788.

[PKK12] V. Puš, L. Kekely, and J. Kořenek. “Low-latency Modular Packet Header Parser for FPGA”. In: *Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. ANCS ’12. Austin, Texas, USA: ACM, 2012, pp. 77–78. ISBN: 978-1-4503-1685-9. DOI: 10.1145/2396556.2396571.

[PK09] Viktor Puš and Jan Korenek. “Fast and Scalable Packet Classification Using Perfect Hash Functions”. In: *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. FPGA ’09. Monterey, California, USA: ACM, 2009, pp. 229–236. ISBN: 978-1-60558-410-2. DOI: 10.1145/1508128.1508163.

[QP16] Y. R. Qu and V. K. Prasanna. “High-Performance and Dynamically Updatable Packet Classification Engine on FPGA”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.1 (2016), pp. 197–209. ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2389239.

[Rio+12] Pedro M. Santiago del Rio et al. “Wire-speed Statistical Classification of Network Traffic on Commodity Hardware”. In: *Proceedings of the 2012 Internet Measurement Conference*. IMC ’12. Boston, Massachusetts, USA: ACM, 2012, pp. 65–72. ISBN: 978-1-4503-1705-4. DOI: 10.1145/2398776.2398784.

[Riz12] Luigi Rizzo. “netmap: A Novel Framework for Fast Packet I/O”. In: *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. Boston, MA: USENIX Association, 2012, pp. 101–112. ISBN: 978-931971-93-5. URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo.

[RDC11] Luigi Rizzo, Luca Deri, and Alfredo Cardigliano. “10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and new Proposals”. In: 2011. URL: http://luca.ntop.org/10g.pdf.

[SH16] A. M. Sadek and A. I. Hussein. “Flexible FPGA implementation of Min-Sum decoding algorithm for regular LDPC codes”. In: *2016 11th International Conference on Computer Engineering Systems (ICCES)*. 2016, pp. 286–292. DOI: 10.1109/ICCES.2016.7822016.

[SVG10] Osman Salem, Sandrine Vaton, and Annie Gravey. “A scalable, efficient and informative approach for anomaly-based intrusion detection systems: theory and practice”. In: *International Journal of Network Management* 20.5 (2010), pp. 271–293. ISSN: 1099-1190. DOI: 10.1002/nem.748. URL: http://dx.doi.org/10.1002/nem.748.

[Soo83] A. K. Sood. “Design of multistage interconnection networks”. In: *IEE Proceedings E - Computers and Digital Techniques* 130.4 (1983), pp. 109–115. ISSN: 0143-7062. DOI: 10.1049/ip-e:19830025.

[Var+15] P. Varga et al. “C-GEP: 100 Gbit/s capable, FPGA-based, reconfigurable networking equipment”. In: *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*. 2015, pp. 1–6. DOI: 10.1109/HPSR.2015.7483084.

[VP15] P. Velan and V. Puš. “High-density network flow monitoring”. In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 2015, pp. 996–1001. DOI: 10.1109/INM.2015.7140424.

[Xila] Xilinx. *SDNet*. Accessed: 2017-02-23. URL: https://www.xilinx.com/products/design-tools/software-zone/sdnet.html.

[Xil17] Xilinx. *Ternary Content Addressable Memory (TCAM) Search IP for SDNet (PG190)*. 2017. URL: https://www.xilinx.com/support/documentation/ip_documentation/tcam/pg190-tcam.pdf.

[Xil14] Xilinx. *Understanding Performance of PCI Express Systems (WP350)*. 2014. URL: https://www.xilinx.com/support/documentation/white_papers/wp350.pdf.

[Xil18] Xilinx. *Virtex UltraScale+ FPGA Data Sheet:DC and AC Switching Characteristics*. 2018. URL: https://www.xilinx.com/support/documentation/data_sheets/ds923-virtex-ultrascale-plus.pdf.

[Xilb] Xilinx. *Xilinx smart networks*. Accessed: 2018-02-19. URL: https://www.xilinx.com/publications/prod_mktg/smarter-networks-backgrounder.pdf.

[Xilc] Xilinx. *Xilinx virtex ultrascale+ family*. Accessed: 2017-02-21. URL: https://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html.

[YM91] Y. Yang and G. M. Masson. “Nonblocking broadcast switching networks”. In: *IEEE Transactions on Computers* 40.9 (1991), pp. 1005–1015. ISSN: 0018-9340. DOI: 10.1109/12.83662.

[Zaz+16] J. F. Zazo et al. “Automated synthesis of FPGA-based packet filters for 100 Gbps network monitoring applications”. In: *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 2016, pp. 1–6. DOI: 10.1109/ReConFig.2016.7857156.

[Zil+14] N. Zilberman et al. “NetFPGA SUME: Toward 100 Gbps as Research Commodity”. In: *IEEE Micro* 34.5 (2014), pp. 32–41. ISSN: 0272-1732. DOI: 10.1109/MM.2014.61.

[Zil+15] N. Zilberman et al. “Reconfigurable Network Systems and Software-Defined Networking”. In: *Proceedings of the IEEE* 103.7 (2015), pp. 1102–1124. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2435732.

[ZMC16] Noa Zilberman, Andrew W. Moore, and Jon A. Crowcroft. “From photons to big-data applications: terminating terabits”. In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 374.2062 (2016). ISSN: 1364-503X. DOI: 10.1098/rsta.2014.0445.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Co-conception matérielle et logicielle pour du traitement de trafic flexible au-delà du terabit par seconde

Mots clés : Surveillance de trafic, FPGA, architecture hétérogène, traitements haute performance, co-conception logicielle/matérielle

Résumé : La fiabilité et la sécurité des réseaux de communication nécessitent des composants efficaces pour analyser finement le trafic de données. La diversification des services ainsi que l'augmentation des débits obligent les systèmes d'analyse à être plus performants pour gérer des débits de plusieurs centaines, voire milliers de Gigabits par seconde. Les solutions logicielles communément utilisées offrent une flexibilité et une accessibilité bienvenues pour les opérateurs du réseau mais ne suffisent plus pour répondre à ces fortes contraintes dans de nombreux cas critiques.
Cette thèse étudie des solutions architecturales reposant sur des puces programmables de type Field-Programmable Gate Array (FPGA) qui allient puissance de calcul et flexibilité de traitement. Des cartes équipées de telles puces sont intégrées dans un flot de traitement commun logiciel/matériel afin de compenser les lacunes de chaque élément. Les composants du réseau développés avec cette approche innovante garantissent un traitement exhaustif des paquets circulant sur les liens physiques tout en conservant la flexibilité des solutions logicielles conventionnelles, ce qui est unique dans l'état de l'art
Cette approche est validée par la conception et l'implémentation d'une architecture de traitement de paquets flexible sur FPGA. Celle-ci peut traiter n'importe quel type de paquet au coût d'un faible surplus de consommation de ressources. Elle est de plus complètement paramétrable à partir du logiciel. La solution proposée permet ainsi un usage transparent de la puissance d'un accélérateur matériel par un ingénieur réseau sans nécessiter de compétence préalable en conception de circuits numériques.

Title : Hardware and software co-design toward flexible terabits per second traffic processing

Keywords : Traffic monitoring, FPGA, heterogeneous architecture, high performance computing, hardware/software co-design

Abstract : The reliability and the security of communication networks require efficient components to finely analyze the traffic of data. Service diversification and throughput increase force network operators to constantly improve analysis systems in order to handle throughputs of hundreds, even thousands of Gigabits per second. Commonly used solutions are software oriented solutions that offer a flexibility and an accessibility welcome for network operators, but they can no more answer these strong constraints in many critical cases.
This thesis studies architectural solutions based on programmable chips like Field-Programmable Gate Arrays (FPGAs) combining computation power and processing flexibility. Boards equipped with such chips are integrated into a common software/hardware processing flow in order to balance shortcomings of each element. Network components developed with this innovative approach ensure an exhaustive processing of packets transmitted on physical links while keeping the flexibility of usual software solutions, which was never encountered in the previous state of the art.
This approach is validated by the design and the implementation of a flexible packet processing architecture on FPGA. It is able to process any packet type at the cost of slight resources overconsumption. It is moreover fully customizable from the software part. With the proposed solution, network engineers can transparently use the processing power of an hardware accelerator without the need of prior knowledge in digital circuit design.