



Rapport technique  
**Surveillance réseau sur NetFPGA**  
Projet 3A



Rédacteurs :

Benoit Fontaine  
Tristan Groléat  
Franziska Hubert

étudiant 3A, filière INFO  
étudiant 3A, filière INFO  
étudiant 3A, filière ELEC

Encadrants :

Matthieu Arzel  
Amer Baghdadi  
Sandrine Vatou  
Sylvain Gombault

enseignant-chercheur, département ELEC  
enseignant-chercheur, département ELEC  
enseignant-chercheur, département INFO  
enseignant-chercheur, département RSM

Octobre 2009 – Mars 2010



## Abstract

The growing use of the Internet, with services like YouTube, Dailymotion or Peer-to-Peer, raises the demand for larger bandwidths continuously. This and the technological evolution leads to traffic which is becoming more and more important. With the increasing bandwidth, the observation and control of the traffic also needs to be accomplished at a higher speed. This new challenge is being encountered by using hardware implementations of monitoring algorithms.

During our third-year project at Télécom Bretagne we worked on such a hardware implementation using a NetFPGA, a PCI board containing an FPGA and four ethernet ports. Starting with a simple ethernet hub that we added to the existing structure of the NetFPGA, we continued with the implementation of a detection of TCP SYN packets. In order to count these SYN packets, needed for the detection of SYN flooding attacks, we used a special stream mining algorithm to reduce the memory needed : the CMS algorithm.

## Résumé

Internet se popularise de plus en plus et les nouveaux services, tels que le streaming ou le Peer-to-Peer, sont toujours plus gourmands en bande passante. Ceci, renforcé par les évolutions technologiques en terme de débit, mène à un trafic en croissance permanente. Les outils de surveillance réseau doivent suivre cette évolution et être capables d'analyser de plus en plus de données de plus en plus vite. L'utilisation d'implémentations matérielles d'algorithmes de surveillance est une réponse à ce nouveau défi.

Notre projet de troisième année à Télécom Bretagne fut l'occasion de travailler sur une telle implémentation matérielle. Nous avons utilisé pour cela un NetFPGA, une carte PCI contenant un FPGA et quatre ports ethernet. Nous avons commencé par transformer le NetFPGA en hub ethernet pour ensuite lui ajouter une fonctionnalité de détection de paquets TCP SYN afin de lui permettre de détecter les attaques de type SYN flooding. Afin de réduire l'emprunte mémoire nécessaire à cette détection, nous avons utilisé un algorithme spécial de stream mining : l'algorithme CMS.



# Table des matières

<b>Introduction</b>	<b>2</b>
<b>1 Nos moyens</b>	<b>5</b>
1.1 Le NetFPGA . . . . .	5
1.2 Le stream mining : l'algorithme CMS . . . . .	6
<b>2 Organisation</b>	<b>9</b>
2.1 Ressources . . . . .	9
2.2 Trac . . . . .	10
2.3 Réunions et division des tâches . . . . .	11
<b>3 Architecture de notre solution</b>	<b>13</b>
3.1 Le rôle du NetFPGA . . . . .	13
3.2 La séparation logiciel/matériel . . . . .	13
3.3 L'architecture matérielle . . . . .	14
<b>4 Réalisation technique de notre solution</b>	<b>17</b>
4.1 Le module NetFPGA network_monitoring . . . . .	17
4.2 Le logiciel de détection de SYN Flooding . . . . .	20
<b>Conclusion</b>	<b>23</b>
<b>Bibliographie</b>	<b>25</b>

# Table des figures

3.1	Modules dans le routeur de référence, selon netfpga.org . . . . .	15
4.1	Schéma du module NetFPGA network_monitoring . . . . .	17
4.2	Fenêtre de détection d'attaques par SYN flooding . . . . .	20

---

# Introduction

Les volumes de trafic en cœur de réseau ne cessent de croître à une vitesse exponentielle. En effet un nombre croissant d'applications requièrent des débits de plus en plus importants. On peut citer le peer-to-peer ou le streaming vidéo (Youtube, Dailymotion) qui permet maintenant de regarder des vidéos en haute définition (720p essentiellement). De plus le déploiement actuel de la fibre optique jusqu'à l'utilisateur terminal ne va pas calmer le rythme car il va rendre possible l'utilisation de nouveaux services encore plus gourmands en bande passante.

Les débits actuellement utilisés sont 10 Mb/s, 100 Mb/s (Fast Ethernet), 1 Gb/s (Gigabit Ethernet) et 10 Gb/s. Cependant le groupe de travail IEEE P802.3ba [2] travaille depuis Novembre 2007 à l'implémentation des normes 40 Gb/s et 100 Gb/s Ethernet. Cette augmentation de débit présente un enjeu supplémentaire pour la surveillance de trafic dont l'objectif est la détection des anomalies dans le trafic réseau comme les dénis de service, les scans ou les virus.

La surveillance doit ainsi s'adapter aux nouvelles vitesses des réseaux, mais aussi faire face à la diversification des menaces de sécurité. Ces deux facteurs doivent être considérés lors du développement de nouvelles techniques au niveau software, hardware, mais aussi au niveau des protocoles pour adapter la surveillance trafic aux nouveaux besoins.

Dans le cadre du projet de 3ème année à Télécom Bretagne d'une durée de 4 mois (de novembre 2009 à mars 2010), nous nous sommes intéressés à la réalisation d'algorithmes de surveillance de trafic utilisant l'accélération matérielle de cartes dédiées. Pour ce projet qui réunit les départements informatique, électronique et réseaux de l'école, nous disposions d'une carte NetFPGA. Il s'agit d'une carte intégrant un FPGA<sup>1</sup> connecté à 4 ports Ethernet, permettant de l'intégrer facilement dans un réseau. Notre but était de réaliser sur cette carte un algorithme de surveillance de trafic. Nous avons choisi de détecter les attaques par SYN Flooding. Il s'agit d'attaques qui tirent partie d'une faiblesse du protocole TCP et qui consistent à envoyer de très nombreux paquets TCP d'un type spécial (SYN), à un serveur pour le surcharger et le rendre inutilisable.

---

<sup>1</sup>Field Programmable Gate Array : circuit intégré programmable en Verilog ou VHDL, langages décrivant la manipulation des bits par des portes logiques

Les objectifs du projet étaient multiples :

**Prendre en main le NetFPGA** C'est-à-dire installer l'environnement de développement.

**Analyser et utiliser l'architecture du NetFPGA** C'est-à-dire être capable d'intégrer notre projet dans cette architecture.

**Implémenter un algorithme de surveillance de trafic** C'est-à-dire réaliser effectivement l'algorithme, permettant de se rendre compte des gains apportés par l'accélération matérielle aux performances.

Les résultats de notre projet pourront servir ensuite dans le cadre plus large d'un projet Européen auquel participe Télécom Bretagne par le biais de l'institut Télécom : le projet DEMONS (DEcentralized, cooperative, and privacy-preserving MONitoring for trustworthinesS ) qui commence en 2010 pour 3 ans, et auquel participe plus particulièrement Sandrine Vaton. Ce projet traite de la surveillance du trafic, et insiste sur la nécessité d'utiliser l'accélération matérielle. Le NetFPGA pourra donc servir dans ce cadre à Télécom Bretagne, et l'algorithme de détection d'attaques par SYN flooding pourra être un bon point de départ.

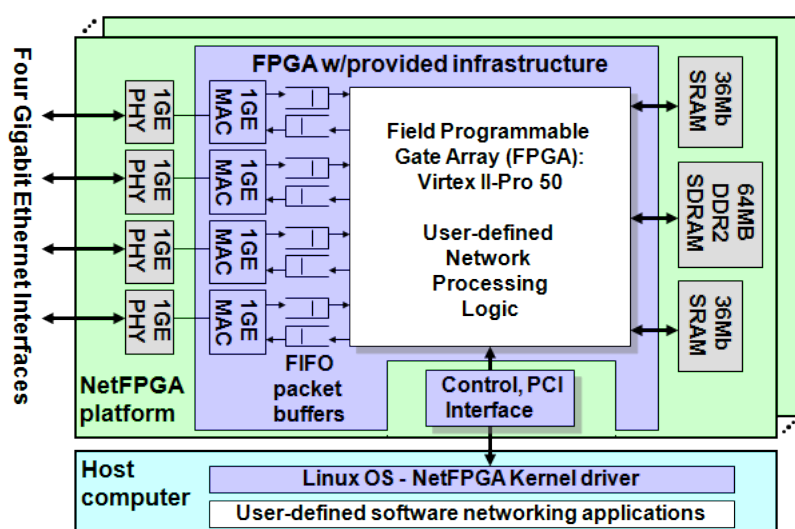


# Chapitre 1

## Nos moyens

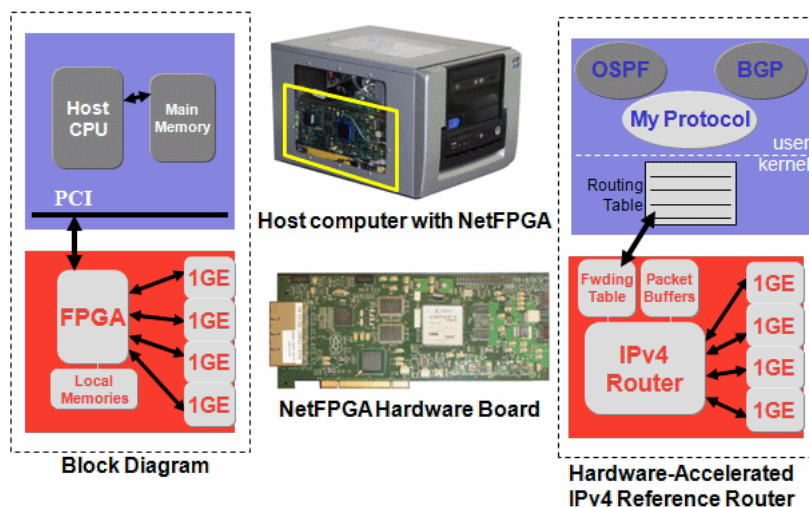
### 1.1 Le NetFPGA

L'élément principal de notre projet est le NetFPGA. Le NetFPGA est une carte PCI dotée d'un FPGA Xilinx Virtex2-Pro 50, de 4 ports Gigabit Ethernet et d'une mémoire embarquée.



Il a été conçu à l'université de Stanford à Palo Alto, CA et permet aux étudiants et chercheurs de réaliser des prototypes de systèmes réseau à haute vitesse en bénéficiant d'une accélération matérielle. Les étudiants de Stanford ont déjà travaillé à réaliser un routeur IP à l'aide du NetFPGA. Une présentation de leurs travaux est disponible sur le site officiel [9].

La carte NetFPGA possède une interface PCI standard et peut donc être connectée à un ordinateur de bureau ou un serveur. Le NetFPGA permet de décharger le processeur de la machine hôte du traitement des données. Le CPU de la machine hôte a accès à la mémoire principale et peut effectuer des opérations DMA de lecture et d'écriture des registres et mémoires du NetFPGA. Contrairement aux autres projets open-source le NetFPGA offre une approche matériellement accélérée. Le NetFPGA fournit une interface matérielle directement connectée aux quatre ports Gigabit et à de multiples banques de mémoire installées sur la carte.



## 1.2 Le stream mining : l'algorithme CMS

### 1.2.1 Le Stream Mining

Il s'agit d'un ensemble de techniques visant à analyser un flux de données avec les contraintes suivantes :

- en temps réel;
- sans stocker l'intégralité du flux dans la mémoire.

Dans notre cas, nous voulons analyser un flux de paquets IP sur un lien à très haut débit (100Gb/s) pour détecter des menaces ou classifier les services utilisés. Le temps réel est donc nécessaire pour pouvoir réagir immédiatement (bloquer certains paquets, en rendre d'autres prioritaires...). Étant donné le débit des données manipulées, nous ne pouvons pas stocker l'intégralité du flux : 100Gb/s stockés pendant 5 minutes nécessiteraient un disque de 30Tb/s, et l'accès au disque serait beaucoup trop lent.

Les techniques de Stream Mining sont particulièrement utilisées pour faire du comptage. Ici, en comptant les requêtes de type TCP SYN, nous pouvons détecter des tentatives d'attaques par SYN flooding depuis une adresse IP donnée.

Comme nous ne stockons pas toutes les données, nous ne pouvons pas faire un comptage exact. Nous pouvons cependant donner une réponse approchée qui peut être :

- déterministe : on sait que l'estimation se trouve à une distance inférieure à  $\epsilon$  de la valeur réelle
- probabiliste : on a une très faible probabilité  $\delta$  que l'estimation se trouve à une distance supérieure à  $\epsilon$  de la valeur réelle.

La conséquence de  $\epsilon$  dans notre cas est nulle, car nous devons fixer un seuil à partir duquel nous détectons une menace. Il suffit donc de relever ce seuil pour éviter les fausses alertes. En revanche  $\delta$  est une probabilité d'erreur. C'est donc une probabilité de détecter une attaque alors qu'il n'y en avait pas. Il faudra donc le garder le plus petit possible, et étudier les conséquences

d'une fausse alerte.

Le Stream Mining permet deux types de détection [10] :

- la détection des “heavy hitters” : les sources qui génèrent beaucoup plus de trafic d'un certain type que les autres : c'est la méthode utilisée pour détecter le SYN flooding ;
- la détection des “heavy changes” : les sources qui génèrent à un moment donné beaucoup plus de trafic d'un certain type que quelques instants auparavant.

### 1.2.2 L'algorithme CMS

L'algorithme Count Min Sketch a été mis au point par Cormode et Muthukrishnan en 2004. Il est simple, ce qui rend possible son implémentation lors de la prise en main du FPGA. Il peut être utilisé pour la détection de heavy hitters, nous allons donc le présenter ci-dessous en l'appliquant à la détection d'attaques par SYN flooding.

Il prend en compte un flux d'éléments avec une marque donnée, et un identifiant donné, et donne une évaluation probabiliste de la somme des marques d'éléments reçus d'un même identifiant. Dans le cas de la détection de Syn Flooding, les éléments du flux sont les paquets TCP de type SYN, et l'identifiant pourrait être simplement l'adresse IP destination. La marque serait toujours 1.

Pour faire fonctionner l'algorithme, on choisit une précision  $\epsilon$  et une probabilité d'échec  $\delta$ . On définit  $w = \frac{\epsilon}{\delta}$  et  $d = \ln(\frac{1}{\delta})$ . Les identifiants sont dans l'ensemble  $\{1, 2, \dots, N\}$ . On prend  $w$  fonctions de hashage aléatoires, uniformes au deuxième ordre et indépendantes qui vont de  $\{1, 2, \dots, N\}$  dans  $\{1, 2, \dots, w\}$ . On utilise alors un tableau de taille  $d \times w$  qui représente l'espace mémoire nécessaire pour faire fonctionner l'algorithme. Pour que l'utilisation de cet algorithme ait un sens, il faut que  $d \times w \ll N$ . C'est ce qui contraint le choix de  $\epsilon$  et  $\delta$ .

A chaque paquet reçu, on applique chaque fonction de hashage à l'identifiant, et on ajoute dans la case désignée par le couple {numéro de la fonction de hashage, valeur obtenue par hashage de l'identifiant} la marque du paquet.

L'estimation de la somme des marques pour un identifiant donné à un moment donné est la valeur minimum trouvée dans les cases désignées par les couples {numéro de la fonction de hashage, valeur obtenue par hashage de l'identifiant} pour toutes les fonctions de hashage.

	$\phi_1$	$\phi_2$	$\phi_3$	$\phi_4$	$\phi_5$
1 = $\phi_3(IP_0)$	0	0	<b>1</b>	0	0
2 = $\phi_1(IP_0)$	<b>1</b>	0	0	0	0
3	0	0	0	0	0
4 = $\phi_2(IP_0) = \phi_5(IP_0)$	0	<b>1</b>	0	0	<b>1</b>
5	0	0	0	0	0
6 = $\phi_4(IP_0)$	0	0	0	<b>1</b>	0

TAB. 1.1 – Exemple de remplissage du tableau lors de la réception du premier paquet ( $IP_0$ ) de marque 1 avec  $\phi_1 \dots \phi_5$  fonctions de hashage dans 1..6

Les complexités en temps de cette algorithme, pour l'ajout d'un paquet ou pour l'estimation

---

du poids d'un identifiant, sont en  $O(\ln(\frac{1}{\delta}))$ . La complexité en mémoire est de  $O(\frac{1}{\epsilon} \times \ln(\frac{1}{\delta}))$ .

Il est ensuite possible d'appliquer cet algorithme à la recherche d'objets massifs. Dans notre cas, les objets massifs sont les attaques probables. On considère alors tous les identifiants dans l'estimation du poids à l'instant  $t$  est supérieure à une constante multipliée par la somme des poids de tous les identifiants au même instant (la constante est strictement inférieure à 1, elle définit le seuil à partir duquel une alerte est lancée).

La complexité de la recherche d'objets massifs est comparable au simple algorithme CMS. On fait une évaluation du poids total de l'identifiant à la réception de chaque paquet, et on maintient une liste des paquets massifs et la somme totale des marques.

Cet algorithme (recherche d'objets massifs incluse) a été implémenté en C par S. Muthu Muthukrishnan en 743 lignes [7]. Nous ne connaissons pas d'implémentation existante en Verilog ou en VHDL. Cette implémentation fait donc partie de notre projet.

### **Le problème de l'inversion du hashage**

L'algorithme CMS pose un problème soulevé dans les recherches : est-il possible d'inverser le hashage opéré sur les adresses IP : nous pouvons évaluer le poids d'une adresse IP donnée, mais pouvons-nous par exemple a posteriori chercher dans la table les poids les plus forts, et retrouver leurs adresses IP ?

La réponse est clairement non si nous choisissons les fonctions de hashage au hasard. Cependant des chercheurs dirigés par Robert Schweller [6] ont implémenté sur FPGA un algorithme qui permet cette inversion, en choisissant intelligemment les fonction de hashage et les identifiants (une fonction de l'IP au lieu de l'IP directement) utilisés.

Ceci permet des analyses a posteriori au lieu des analyses incrémentales comme la recherche d'objets massifs. On peut ainsi faire des détections plus poussées, et utiliser moins de mémoire (il n'est plus nécessaire de stocker la liste des objets massifs potentiels à chaque instant par exemple).

Cela nous donne une piste d'amélioration de l'algorithme, même si ce n'est pas nécessaire pour une implémentation simple de la détection d'attaques par SYN flooding.

# Chapitre 2

## Organisation

### 2.1 Ressources

#### Ressources humaines

Ce projet possède l'originalité de s'étaler sur trois filières : ELEC, INFO et RSM. Deux élèves de filière INFO et une élève de filière ELEC travaillent sur le projet. Deux encadrants sont de filière ELEC, un de filière INFO et un de filière RSM.

#### Ressources matérielles

Le département électronique nous a mis à disposition un de ses laboratoires (au bâtiment K2) ainsi qu'une station de travail dotée d'une interface réseau, de la carte NetFPGA et d'une distribution Linux Ubuntu Karmic 9.10. Cette station est connectée au réseau Salsa-Lite de l'école. Il est à noter que ceci impose de taper sur le portail captif son identifiant et mot de passe école à chaque fois que l'on est inactif plus de quelques dizaines de minutes sur Internet.

Nous avons très vite eu besoin d'une nouvelle carte réseau afin de tester le NetFPGA sans devoir changer à chaque fois les câbles et perdre du même coup notre connexion vers le Net.

Nous avons découvert par la suite que le NetFPGA ne fonctionne qu'avec des interfaces 1000BaseT (Gigabit) et pas avec les interfaces classiques 100BaseT. Or nos ordinateurs portables ne supportent que le 100BaseT. Nous avons donc emprunté une seconde station de travail disponible dans le laboratoire qui dispose elle d'une interface Gigabit. Nous pouvons ainsi simuler un réseau de deux ordinateurs avec le NetFPGA situé en coupure pour l'analyse des communications.

La première étape dans le déroulement du projet fut de faire fonctionner le NetFPGA sur les ressources qui nous ont été attribuées. En effet l'environnement supporté officiellement par celui-ci est différent du notre. Nous disposions d'une Ubuntu récente alors que le site du NetFPGA recommandait une CentOS plus ancienne. Il a donc fallu effectuer des modifications dans le code de la partie software du NetFPGA pour le faire fonctionner sur un noyau et des bibliothèques plus récentes. Cette phase était essentielle sans quoi le projet n'aurait jamais pu continuer.

---

## 2.2 Trac

Afin de rendre la gestion du projet plus aisée, nous utilisons le moteur **Trac** couplé au système de gestion de versions **Subversion**. Trac est un système Open Source de gestion complète de projet par Internet.



Il inclut :

**Un wiki** Nous écrivons l'ensemble de la documentation du projet dans ce wiki. Dès que nous découvrons ou mettons en place quelque chose de nouveau, un article y est tout de suite consacré. Ainsi des pages sont consacrées aux outils que nous utilisons, d'autres aux algorithmes de surveillance, d'autres à l'utilisation et au développement sur NetFPGA et enfin une partie est dédiée à l'organisation avec un planning des événements.

**Subversion** Subversion est un logiciel de gestion de versions. Nous gérons avec cet outil le code du NetFPGA ainsi que les rendus écrits en  $\text{\LaTeX}$ (comme celui-ci). Cela nous permet de travailler simultanément sur les même fichiers et de garder un historique de nos modifications avec la possibilité d'y associer des commentaires (les messages de "commit").

**Un système de tickets** Les tickets sont très pratiques pour communiquer et se coordonner sur une tâche à accomplir ou un problème à résoudre. Il est possible de laisser des commentaires, ainsi les discussions ne sont pas perdues mais stockées dans Trac et disponibles pour de futurs contributeurs au projet. De cette manière on peut vite voir si un problème auquel on est confronté n'a pas déjà été résolu. Chaque personne associée au ticket est notifiée par email des actions prises sur celui-ci.

**Une gestion des feuilles de route** Les feuilles de route permettent de s'organiser sur les échéances. Une feuille de route correspond en fait à une échéance. On lui donne un intitulé, une description, une date de fin puis on y associe un certain nombre de tickets. Au fur et à mesure que les tickets sont fermés, la barre de progression de la feuille de route se met à jour et nous montre le chemin qui nous sépare encore de sa réalisation totale. Par exemple une feuille de route a été créée pour la réalisation de ce rapport de synthèse et cinq tickets y ont été associés.

**Un historique** La page d'historique est très importante car elle permet de suivre l'évolution du projet en un simple coup d'œil. Cette page regroupe les dernières modifications de tickets, de feuilles de route, de commits subversion et du wiki. Il est même possible d'accéder à

son contenu au format RSS afin d'être en permanence au courant de l'évolution du projet grâce à son lecteur de flux RSS préféré.

Trac [8] nous permet donc de travailler de façon collaborative et de mutualiser nos contributions au projet. C'est un outil essentiel à tout projet comptant s'inscrire dans le temps. La page web de notre Trac est disponible à l'adresse suivante : <http://trac.benoute.fr/netfpga> [3].

## 2.3 Réunions et division des tâches

Les réunions sont un élément important non seulement pour s'organiser mais aussi pour produire et partager les connaissances acquises. Nous organisons plusieurs réunions chaque semaine. Il est possible de répartir ces réunions en trois catégories :

**Réunions avec les encadrants** Nous avons décidé en début de projet d'effectuer une réunion chaque lundi avec nos encadrants. C'est le moment de leur faire part de notre progression mais aussi de nos problèmes.

**Réunions de travail** Nous nous réunissons au moins deux fois par semaine dans notre laboratoire de projet. Ces réunions sont en fait des sessions de travail. Elles durent en général un après-midi voire une journée entière. C'est durant ces sessions que se fait le développement du projet. Soit nous nous repartissons des tâches et nous travaillons en parallèle, soit nous travaillons à plusieurs sur la même tâche.

**Réunions d'organisation** Ces réunions ont lieu lorsqu'il est nécessaire de s'organiser sur une échéance. Aucun développement n'est réalisé, seule l'organisation est traitée. Ces réunions sont plus courtes et durent de quelques dizaines de minutes à une heure.



## Chapitre 3

# Architecture de notre solution

### 3.1 Le rôle du NetFPGA

Nous souhaitons faire de la surveillance de trafic sur un lien à très haut débit en utilisant l'accélération matérielle fournie par le NetFPGA. Pour cela, nous avons mis le NetFPGA en coupure du lien à surveiller de manière à ce que tous les paquets qui passent par ce lien traversent le NetFPGA. Nous utilisons donc deux des quatre interfaces Ethernet que possède le NetFPGA.

Ceci signifie que le rôle premier du NetFPGA doit être de transmettre tous les paquets qu'il reçoit sur une interface à l'autre interface. Pour simplifier l'utilisation et le test de notre solution, nous avons décidé de ne pas imposer les deux ports à utiliser sur le NetFPGA. Il faut donc que le NetFPGA copie tous les paquets qu'il reçoit sur n'importe quelle interface sur toutes les autres interfaces.

Ce comportement est celui d'un élément bien connu en réseau : le hub Ethernet. Notre première tâche au niveau du développement a donc été de transformer le NetFPGA en simple hub Ethernet. Nous aurions pu compliquer son comportement en lui faisant jouer le rôle de switch Ethernet, qui apprend à qui il est connecté, ou même celui de routeur IP. Mais cela aurait été peu utile car nous ne voulons utiliser que deux interfaces, et cela aurait compliqué la configuration et le test du NetFPGA.

### 3.2 La séparation logiciel/matériel

Maintenant que nous avons décidé que le NetFPGA agirait comme un hub Ethernet, le problème suivant est de savoir comment la surveillance du trafic doit être faite. Il y a trois grandes étapes dans cette surveillance :

1. la détection des paquets qui nous intéressent, ici les paquets TCP SYN ;
2. le comptage de ces paquets (algorithme CMS) ;
3. l'analyse des résultats, et le possible déclenchement d'alertes.

Si nous ne faisons pas la première étape sur le NetFPGA, nous sommes obligés de renvoyer tous les paquets reçus à l'ordinateur. Or la communication avec l'ordinateur est assez lente. La conséquence serait donc une réduction importante du débit maximal auquel le NetFPGA peut

---

fonctionner. Pour utiliser les capacités d'accélération matérielle du NetFPGA, nous avons donc décidé de faire la première étape dans le NetFPGA.

La seconde étape n'est déclenchée qu'à la réception d'un paquet qui nous intéresse, ici un TCP SYN. Et la seule information qui nous intéresse est l'adresse IP source du paquet. Il serait donc envisageable d'envoyer l'adresse IP source à la réception de chaque paquet TCP SYN à l'ordinateur pour qu'il le traite. Mais ceci ne fonctionnerait que si nous recevons suffisamment peu de paquets SYN. Or le but de notre projet est de détecter les attaques qui se font par envois massifs de paquets TCP SYN. Il est donc indispensable de pouvoir garantir le fonctionnement de notre solution avec une grande quantité de paquets TCP SYN reçus. C'est pourquoi nous avons aussi effectué la seconde étape sur le NetFPGA.

La troisième étape en revanche est déclenchée quand nous le souhaitons. La réaction immédiate aux attaques de SYN flooding n'entre en effet pas dans le cadre de notre projet : nous souhaitons seulement détecter ces attaques. C'est pourquoi nous pouvons n'analyser les résultats que périodiquement, à un intervalle fixe. Cette analyse est faite par l'ordinateur qui peut utiliser sa grande puissance de calcul pour faire des analyses complexes, générer des statistiques. . .

En résumé, nous avons implémenté la détection et le comptage sur le NetFPGA, et développé sur l'ordinateur un programme qui accède périodiquement aux données du NetFPGA pour les analyser.

## **3.3 L'architecture matérielle**

### **3.3.1 L'architecture de référence NetFPGA**

Le NetFPGA est programmé à partir du langage Verilog. Il s'agit d'un langage qui spécifie précisément la manipulation des signaux binaires. Il existe un programme Verilog appelé "reference NIC", fourni avec le NetFPGA, qui transforme le NetFPGA en simple carte Ethernet à 4 ports (tous les paquets reçus sont remontés à l'ordinateur, qui peut aussi en envoyer). C'est cette référence que les projets comme le notre utilisent comme base.

Le parcours d'un paquet Ethernet dans le "reference NIC" est divisé en modules. Chaque module reçoit le paquet, fait son traitement puis renvoie le paquet, éventuellement modifié, au module suivant. Si les modules ont besoin de se communiquer des données extérieures au paquet, ils peuvent ajouter ou modifier des en-têtes spécifiques au NetFPGA au début du paquet. [1]

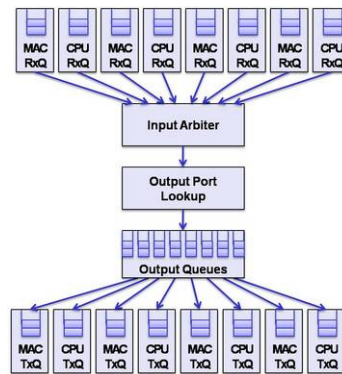


Fig. 3.1 – Modules dans le routeur de référence, selon netfpga.org

Le premier module reçoit les trames Ethernet, les découpe, et leur ajoute une en-tête disant d'où ils viennent. Le module suivant, appelé “output\_port\_lookup”, modifie l'en-tête pour indiquer par quelles interfaces le paquet doit ressortir. Les modules se succèdent ainsi jusqu'au dernier qui renvoie les paquets sur les bonnes interfaces.

### 3.3.2 Les modifications apportées pour notre projet

La première chose que nous voulions faire était de transformer le NetFPGA en hub Ethernet. Nous sommes pour cela partis du “reference NIC”, et nous avons modifié le module “output\_port\_lookup” pour que les paquets soient redirigés de la même manière que dans un hub : vers toutes les interfaces sauf celle d'entrée. Nous avons effectué ces modifications guidés par un tutoriel pour apprendre à manipuler le NetFPGA trouvé sur Internet [4].

Enfin, il nous fallait décider d'un endroit où placer la surveillance de trafic. Nous avons pour cela décidé d'ajouter un module NetFPGA juste après le module “output\_port\_lookup”. Nous avons appelé ce nouveau module “network\_monitoring”. Il se charge de :

- détecter les paquets TCP SYN qui passent, en évitant d'arrêter le trafic ;
- compter les paquets détectés pour chaque IP, en utilisant l'algorithme CMS.



## Chapitre 4

# Réalisation technique de notre solution

### 4.1 Le module NetFPGA network\_monitoring

#### 4.1.1 La division en modules Verilog

Le module NetFPGA network\_monitoring est un module Verilog qui fait appel à plusieurs autres modules Verilog pour réaliser sa tâche. Cette décomposition permet de rendre plus lisible le fonctionnement du module.

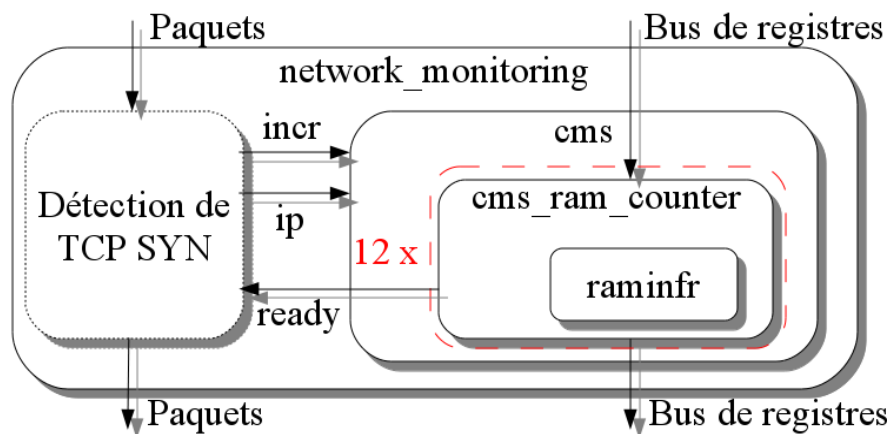


Fig. 4.1 – Schéma du module NetFPGA network\_monitoring

#### 4.1.2 La détection de paquets TCP SYN

La détection de paquets TCP SYN est faite directement dans le module "network\_monitoring" qui laisse les paquets le traverser sans les modifier.

Ce module lit les champs qui l'intéressent dans les paquets qu'il reçoit en même temps qu'ils passent. Il ne limite donc pas le débit du trafic reçu. Il est cependant nécessaire qu'il puisse prendre le temps d'enregistrer l'arrivée d'un paquet TCP SYN quand il en détecte un. Pour

---

cela, nous utilisons une file d'attente, qui est ouverte tout le temps sauf quand le module est en train d'enregistrer l'arrivée d'un paquet TCP SYN.

Un paquet reçu est considéré comme un paquet TCP SYN si :

- le champ “ethertype” de l'en-tête Ethernet est à 0x800 : indique un paquet IP ;
- le champ “protocol” de l'en-tête IP est à 6 : indique un paquet TCP ;
- le 5ème drapeau (SYN) de l'en-tête TCP est à 1 : indique un paquet SYN.

Le début du paquet est identifié grâce au signal de contrôle envoyé par le module au-dessus : ce signal prend une valeur spécifique lorsque l'en-tête spécifique au NetFPGA est reçue, or cette en-tête précède toujours le début d'un paquet.

### 4.1.3 Le compteur CMS

#### Les caractéristiques

Le compteur est réalisé par le module Verilog appelé CMS. Son fonctionnement se base sur le code en C développé par S. Muthu Muthukrishnan [7].

Il comporte deux entrées “incr” et “ip” et une sortie “ready”. L'entrée IP est réglée à l'IP source des paquets reçus, tandis que incr est un bit mis à 1 uniquement quand un paquet SYN est détecté. La sortie “ready” est à 1 si le compteur est prêt à compter un nouveau paquet.

Même si l'algorithme CMS diminue la mémoire nécessaire, il faut quand même une mémoire importante. Nous disposons sur le FPGA Virtex2Pro de 4 276 224 bits de RAM en blocs synthétisable au maximum [11], mais le reste des modules NetFPGA en utilise déjà une partie.

Nous avons décidé d'utiliser des compteurs de 32 bits, pour ne pas avoir de dépassement de capacité lors d'attaques par SYN flooding. Nous avons ensuite choisi une précision  $\epsilon$  et une probabilité d'échec  $\delta$  pour l'algorithme CMS de manière à ne pas utiliser trop de mémoire, tout en gardant des résultats corrects :  $\epsilon = 0,000664$  et  $\delta = 0,000006$  donnent une mémoire de 12 lignes et 4096 colonnes. Nous avons donc besoin de  $12 \times 4096 \times 32 = 1\,572\,864$  bits.

#### La réalisation

Pour réaliser l'algorithme CMS, nous avons besoin d'incrémenter une colonne donnée de chaque ligne, ce qui donne 12 lectures, puis 12 écritures à faire, soit 24 cycles d'horloge car on ne peut accéder à la RAM qu'une fois par cycle. Pour accélérer ce traitement, critique si nous recevons de nombreux paquets SYN, nous avons utilisé 12 RAMs : nous faisons ainsi 12 lectures, puis 12 écritures, ce qui permet de réaliser le comptage d'un paquet TCP SYN en 2 cycles seulement (le calcul de hashes nécessaire pour déterminer les colonnes à incrémenter est fait en mois d'un cycle, juste avant la lecture).

Pour réaliser ceci, nous utilisons le module “raminfr” qui est une mémoire RAM synchrone standard à double port en lecture et un seul en écriture de 4096 mots de 32 bits, ce qui représente une ligne de l'algorithme CMS. Le module “cms\_ram\_counter” est un adaptateur qui lit et écrit dans la RAM et permet de demander simplement l'incrémentation, qui se fera en deux cycles. Il indique quand il a terminé grâce à sa sortie “ready”.

Les modules “cms\_ram\_counter” et “raminfr” sont instanciés 12 fois, ce qui correspond aux 12 lignes de l’algorithme CMS. Le bon numéro de colonne à incrémenter est envoyé à chaque instance, grâce aux calculs de hashes faits dans le module “cms”.

#### 4.1.4 L’export de la mémoire vers l’ordinateur

Le dernier problème qui se pose pour réaliser notre surveillance de trafic est la lecture par l’ordinateur de la mémoire RAM que les modules “raminfr” utilisent pour enregistrer les résultats de l’algorithme CMS. Cette lecture est prévue dans l’architecture globale du NetFPGA grâce à un bus de données qui forme un anneau en passant par tous les modules du NetFPGA.

Un module Verilog appelé “generic\_register” existe qui permet d’implémenter quelques registres ou compteurs et de les rendre accessibles depuis le NetFPGA et l’ordinateur. Il se connecte au bus de registres qui forme un anneau en passant par tous les modules NetFPGA. Cependant nos besoins importants en mémoire nous ont poussés à utiliser la RAM des modules “raminfr” pour stocker nos données car le module “generic\_register” ne peut pas gérer plus de 256 compteurs.

Nous nous sommes donc inspirés du module “generic\_register” pour exporter nous-même les valeurs présentes dans notre RAM. Ce travail est fait par les modules “cms\_ram\_counter” qui utilisent le second port de lecture du module raminfr pour répondre aux requêtes de lecture de la RAM provenant de l’ordinateur. Les modules sont connectés au bus de registres les uns derrière les autres, et ils ne répondent qu’aux requêtes qui les concernent.

Les adresses auxquelles les requêtes doivent être envoyées sont générées automatiquement par les scripts de synthèse du NetFPGA grâce à un fichier xml de description de nos registres (taille et nombre). Des fichiers C et Python sont générés à cette occasion contenant les adresses pour faciliter la lecture des registres par l’ordinateur.

---

## 4.2 Le logiciel de détection de SYN Flooding

### 4.2.1 Les caractéristiques

Nous avons réalisé le logiciel de détection de SYN Flooding en C++ pour pouvoir utiliser les bibliothèques C disponibles pour le NetFPGA et pour l'algorithme CMS. L'interface graphique est en Qt et fut réalisée à l'aide de l'IDE Qt Creator [5].

Il faut indiquer une adresse IP à surveiller au logiciel qui applique alors l'algorithme CMS en calculant les différents hashes de l'adresse IP. Il en déduit les colonnes auxquelles il doit aller lire les compteurs pour chaque ligne, puis il traduit ces coordonnées {ligne, colonne} en adresse dans le NetFPGA. Il lit enfin ces valeurs toutes les demi-secondes, et calcule le minimum pour donner l'estimation du nombre de paquets SYN envoyés par cette adresse IP.

Si le nombre de paquets SYN envoyés par l'IP augmente trop vite (plus de 200 par seconde), le logiciel considère qu'il s'agit d'une attaque par SYN flooding et l'affiche. Il pourrait alors décider la coupure du réseau pour cette adresse IP, mais cette fonction n'entrait pas dans le cadre du projet et n'a pas été réalisée.

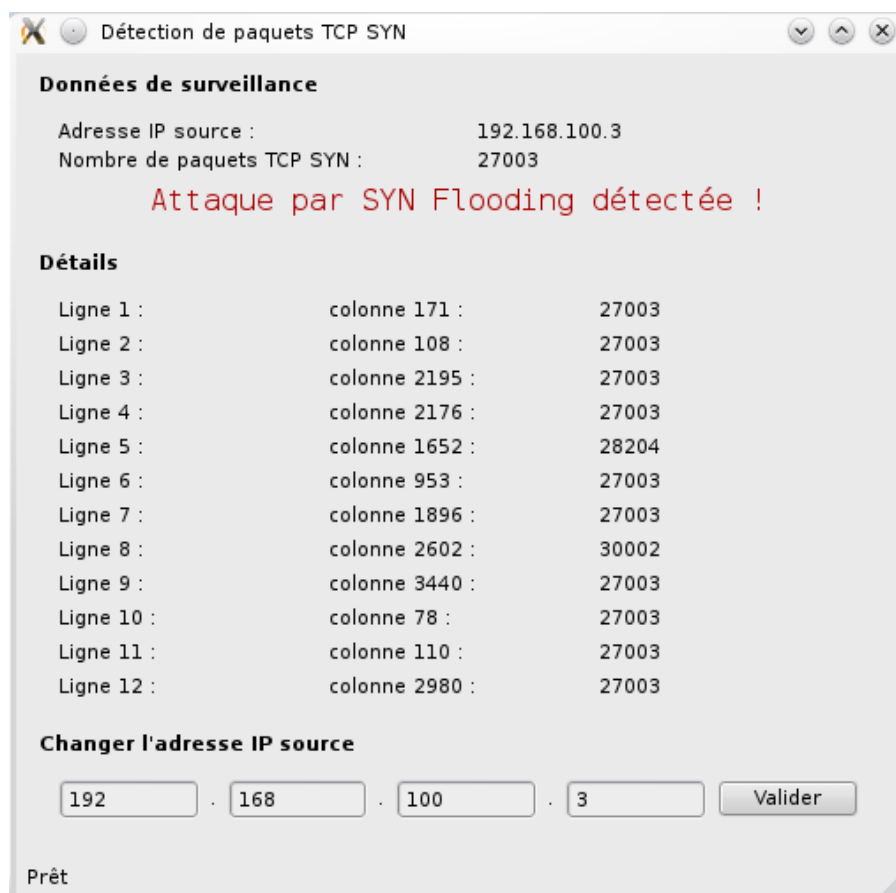


Fig. 4.2 – Fenêtre de détection d'attaques par SYN flooding

### 4.2.2 La lecture dans le NetFPGA

La lecture de la mémoire du NetFPGA par l'ordinateur s'effectue grâce à des pilotes fournis par NetFPGA qui font passer l'interface de connexion avec le NetFPGA pour une interface réseau. Elle est facilitée par des bibliothèques fournies en C pour lire/écrire des registres à une adresse donnée. Les adresses sont données par un fichier d'en-têtes C généré automatiquement lors de la synthèse de notre module NetFPGA.

Nous avons inclus ces bibliothèques C dans notre programme en C++. Elles nous permettent de lire une valeur en indiquant son adresse. Nous connaissons grâce au fichier d'en-têtes généré automatiquement l'adresse de début des valeurs concernant la surveillance réseau, ainsi que l'incrément entre chaque adresse. Nous avons connecté nos modules de telle manière que chaque compteur 32 bits de l'algorithme CMS ait une adresse, et les différentes lignes du tableau CMS sont mises les unes derrière les autres. Il est donc facile au programme de savoir à quelle adresse lire à partir de la ligne et de la colonne données par l'algorithme CMS.



# Conclusion

Les objectifs de ce projet étaient multiples :

**Prendre en main le NetFPGA** Nous avons mis en place l'environnement de développement du NetFPGA pour la première fois à TELECOM Bretagne. Les étapes que nous avons suivies pour le mettre en place sont consultables sur le wiki et le SVN du projet. La machine installée pourra aussi être utilisée pour d'autres projets sur le NetFPGA.

**Analyser et utiliser l'architecture du NetFPGA** Nous avons documenté dans le wiki du projet la méthode qui permet d'insérer son propre module dans l'architecture du NetFPGA, et d'utiliser ses fonctionnalités : lecture des paquets reçus, communication avec l'ordinateur... Ces informations pourront servir à tous les projets futurs sur NetFPGA à Télécom Bretagne.

**Implémenter un algorithme de surveillance de trafic** Nous avons développé un algorithme de détection d'attaques TCP SYN utilisant l'accélération matérielle fournie par le NetFPGA. Les performances de cet algorithme peuvent être comparées à celles des implémentations purement logicielles. Ce code peut aussi servir de base à l'implémentation d'algorithmes plus sophistiqués.

En plus d'être intéressant, ce projet fut très enrichissant au niveau des connaissances acquises. Tout d'abord nous avons pris conscience de certaines problématiques du domaine du système embarqué comme la gestion parcimonieuse des ressources. Ce point fut à l'origine de la décision de l'implémentation de l'algorithme CMS car il permet de limiter l'empreinte mémoire due au stockage des résultats d'analyse du trafic. Le NetFPGA aura aussi été l'occasion d'utiliser un FPGA dans un environnement complexe, c'est-à-dire dans un environnement avec beaucoup de code déjà écrit et reposant sur une architecture évoluée. Avant d'être en état de développer, il a fallu se plonger dans cette architecture en lisant documentation et code. Cela nous a permis d'en découvrir les rouages et de comprendre comment réaliser nous-même des choses similaires. Notre formation scolaire nous amène à utiliser la suite Xilinx : ce projet nous a permis de confirmer ces enseignements mais aussi de réaliser que l'on pouvait utiliser ces outils d'une façon différente. Nous avons enfin découvert l'autre gros langage de conception, le Verilog. Ce point est très important car seul le langage VHDL est enseigné à Telecom Bretagne.

Les bénéfices pour l'école ne sont pas moins importants. La phase d'installation du NetFPGA est entièrement maîtrisée ainsi que la phase de développement. Il est donc maintenant possible

---

d'implémenter des fonctionnalités poussées et intéressantes pour le/les groupes qui reprendront le projet l'année prochaine. L'école est aussi à présent en mesure d'utiliser nos acquis pour intégrer le NetFPGA dans le cursus scolaire de 3ème année. Enfin nous avons gardé à jour un large wiki [3], contenant toutes nos découvertes et développements, que l'école pourra exploiter librement.

Les perspectives d'avenir du projet sont variées : premièrement en ce qui concerne la surveillance de trafic, l'algorithme de détection d'attaques que nous avons implémenté pourra être utilisé pour stopper ces attaques. Des mesures de performance plus poussées pourront aussi être effectuées et des algorithmes plus efficaces pourront être testés. Enfin les connaissances acquises sur le NetFPGA pourront servir à tous les projets qui visent à accélérer matériellement des algorithmes ayant à s'intégrer dans un réseau.

# Bibliographie

## RÉFÉRENCES

- [1] GUIDE NETFPGA : <http://netfpga.org/foswiki/bin/view/NetFPGA/OneGig/Guide>, 2010.
- [2] IEEE P802.3BA 40Gb/s AND 100Gb/s ETHERNET TASK FORCE : <http://www.ieee802.org/3/ba/>, 2010.
- [3] LE TRAC DU PROJET : <http://trac.benoute.fr/netfpga>, 2010.
- [4] NETFPGA ETHERNET HUB TUTORIAL : [http://comp519.cs.rice.edu/index.php/Ethernet\\_Hub\\_Tutorial\\_-\\_Implementation](http://comp519.cs.rice.edu/index.php/Ethernet_Hub_Tutorial_-_Implementation), 2008.
- [5] QT CREATOR : <http://qt.nokia.com/products/developer-tools>, 2010.
- [6] ROBERT SCHWELLER, ZHICHUN LI, YAN CHEN, YAN GAO, ASHISH GUPTA, YIN ZHANG, PETER A. DINDA, MING-YANG KAO, GOKHAN MEMIK : Reversible sketches : Enabling monitoring and analysis over high-speed data streams, 2009.
- [7] S. MUTHU MUTHUKRISHNAN : ALGORITHME CMS : <http://www.cs.rutgers.edu/~muthu/massdal-code-index.html>, 2004.
- [8] SITE DU PROJET TRAC : <http://trac.edgwall.org/>, 2010.
- [9] SITE OFFICIEL DU NETFPGA : <http://netfpga.org>, 2010.
- [10] TIAN BU, JIN CAO, AIYOU CHEN, PATRICK P. C. LEE : A fast and compact method for unveiling significant patterns in high speed networks, 2007.
- [11] XILINX : Virtex-ii pro and virtex-ii pro x platform fpgas : Complete data sheet, 2007.