PacketShader : transformer un PC en routeur IP ultra-rapide

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 septembre 2010. Dernière mise à jour le 27 septembre 2010

https://www.bortzmeyer.org/packetshader.html

L'augmentation colossale de la capacité des réseaux informatiques fait peser une lourde tâche sur les épaules des routeurs, qui doivent désormais faire passer des paquets sur plusieurs liens 10 Gb/s. Les routeurs dits « logiciels », basés sur du matériel non-spécialisé (par exemple un PC) peuvent-il tenir ce rythme ou bien doivent-il laisser la place aux engins chers et surtout 100 % logiciel privateur comme les Cisco ou les Juniper ? Ce remarquable article de chercheurs coréens montre que non et qu'il est possible aujourd'hui de bâtir un routeur aussi rapide que les engins de course, avec un PC et Linux. Principale percée réalisée par les dits chercheurs : sous-traiter une bonne partie des opérations au GPU...

Contrairement à ce que prétendent les commerciaux de Cisco ou de Juniper, il a toujours été possible de bâtir un routeur IP sur du matériel ordinaire, avec du logiciel libre. Ce n'est toutefois pas une tâche triviale : un routeur moderne comporte deux parties, le Contrôle ("*control plane*") et la Transmission ("*forwarding plane*"). Le premier parle les protocoles de routage comme BGP, protocoles souvent complexes, sa tâche n'est que faiblement « temps réel » et il a besoin de mémoire (un routeur BGP sans route par défaut stocke aujourd'hui dans les 330 000 routes <http://bgpmon.net/stat.php>) et de CPU. Un PC ordinaire est donc bien adapté à cette tâche, pour laquelle il existe plusieurs logiciels libres comme Quagga ou BIRD.

La Transmission, quant à elle, est implémentée dans tous les Unix depuis belle lurette, donc tout PC avec Ubuntu sait faire tout le travail de routage (sur Unix, la table de routage utilisée par la transmission s'affiche avec `netstat -rn`). Il existe aussi des projets plus complets d'un routeur entier sur un PC/Unix comme par exemple RouteBricks <http://routebricks.org/>, pris comme point de comparaison dans l'article. Mais le problème est quantitatif. Le travail du Contrôle est proportionnel au nombre de routes et au nombre de changements de celles-ci. Mais celui de la Transmission est proportionnel au nombre de paquets ! Même si les opérations à effectuer pour chaque paquet sont simples, ce travail doit être multiplié par des chiffres impressionnants. Et pas question de trainer, sous peine d'accroître la latence du réseau. Sur un lien Ethernet à 10 Gb/s (ce qui est encore luxueux sur le bureau mais est déjà la norme à l'intérieur des réseaux d'opérateurs), cela peut faire des dizaines de millions

de paquets par seconde. En débit (les 10 Gb/s) et en nombre d'interruptions (une par paquet pour une mise en œuvre naïve), cela stresse considérablement le PC de base.

Les routeurs très chers des grosses entreprises s'en tirent en utilisant des composants complètement différents pour le Contrôle et la Transmission : sur un Juniper, par exemple, un PC doté d'une variante de FreeBSD fait le contrôle et des ASIC, bien plus coûteux, assurent la Transmission. On voit que, lorsque certains enthousiastes du logiciel libre annoncent qu'un logiciel comme Quagga « permet à n'importe quel PC de faire comme un Cisco dix fois plus cher », ils simplifient exagérement : Quagga assure le Contrôle mais il reste à transmettre les paquets très vite ("*at line rate*", dit-on souvent en anglais, c'est-à-dire à la vitesse maximale que permet le réseau physique sous-jacent).

Un dernier mot sur l'architecture d'un routeur : quel est le protocole de communication entre le Contrôle et la Transmission ? Il existe une norme à l'IETF, Forces ("*Forwarding and Control Element Separation*"), spécifiée dans le RFC 5810 [1]. Mais il n'est pas déployé pour l'instant et, en pratique, le protocole de communication est toujours un protocole privé (comme netlink sur Linux, documenté dans le RFC 3549). Il n'est donc pas possible aujourd'hui de « faire son marché » en combinant des composants de divers constructeurs, connectés par un protocole standard.

Revenons maintenant à l'article dont je voulais parler. « "*PacketShader : a GPU-Accelerated Software Router*" <http://www.ndsl.kaist.edu/~kyoungsoo/papers/packetshader.pdf> », de Sangjin Han, Keon Jang, KyoungSoo Park et Sue Moon, décrit en détail la méthode utilisée pour mettre au point le logiciel PacketShader, un programme tournant sur Linux et qui permet d'obtenir d'un PC (un engin de "*gamer*", toutefois, pas un PC bas de gamme) les performances d'un routeur « matériel ». Leur but est de proposer des routeurs moins chers et plus flexibles, puisque le code pourra être modifié plus facilement que s'il est dans un noyau temps-réel. Par contre, question logiciel libre, il ne semble pas que le source de PacketShader soit disponible. Les performances annoncées sont impressionnantes, avec 40 Gb/s de transmis en IPv4, pour toutes les tailles de paquet (une partie du coût est « par octet » et une autre est « par paquet » donc effectuer les mesures avec des paquets de la taille maximale est une méthode classique pour avoir des bons chiffres dans les "*benchmarks*"). Même en IPsec, avec le coût du chiffrement, PacketShader pompe de 10 à 20 Gb/s.

Les résultats détaillés de PacketShader sont présentés dans la section 6 de l'article. Quatre applications ont été testées, transmission IPv4, transmission IPv6, OpenFlow et IPsec. Dans tous les cas, un générateur de paquets produisait des données que PacketShader devrait recevoir et transmettre. Pour le routage IPv4, la taille de la table de routage (trop grande pour tenir dans les caches du processeur) entraine beaucoup d'accès mémoire. Une table réelle était utilisée (copiée de Route Views) alors que, en IPv6, une table artificielle avait été générée (la table réelle, très petite, aurait tenu dans le cache L2). Pour IPsec, le choix a été de faire du routeur PacketShader le point d'entrée d'un tunnel ESP. À chaque fois, PacketShader a tourné dans deux modes à comparer, uniquement sur le CPU et avec le CPU et le GPU.

Les résultats sont rassemblés dans la figure 11. En IPv4, le débit théorique maximal est atteint, même sans le GPU. Ce dernier ne sert que pour les petits paquets (un gain de 40 % pour les paquets de 64 octets). En IPv6, le gain du GPU est bien plus important : la plus grande taille des adresses oblige à davantage d'accès mémoire. Pour IPsec, l'avantage du GPU est très marqué, même pour les grands paquets (1514 octets), où le GPU fait plus que doubler le débit.

Comment sont obtenues ces perfomances ? L'une des principales percées de cet article est d'utiliser un composant peu connu du PC, mais désormais largement disponible, le processeur graphique (le

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc5810.txt`

terme de "*packet shader*" vient d'ailleurs d'un terme du monde du graphique, "*shader*"). Grâce aux "*gamers*", on trouve désormais pour les PC des processeurs ultra-spécialisés, très parallèles (des centaines d'unités de calcul sur une seule puce), programmables par des bibliothèques relativement accessibles comme CUDA. (Attention, CUDA est non libre. Elle a un concurrent libre, OpenCL, mais qui semble très peu déployé en pratique, peut-être en raison de sa plus grande difficulté d'usage.) Ils coûtent facilement dans les 500 € mais augmentent nettement les performances graphiques. Ce GPU peut servir à bien d'autres choses qu'à l'affichage de monstres qu'il faut abattre le plus vite possible (pour un exemple rigolo, voir une base de données avec GPU `<http://www.parstream.com/>`). Et son rapport performances/prix augmente bien plus vite que celui du CPU. La section 2 de l'article décrit en détail les GPU, et le modèle utilisé, un Nvidia GTX 480 (ce dernier chiffre indiquant le nombre d'unités de calcul). Le GPU est certes très rapide, mais c'est en partie parce que son modèle d'exécution est différent de celui d'un processeur généraliste. De type SIMT ("*single-instruction, multiple-threads*", une catégorie absente de la taxinomie de Flynn), le GPU requiert une programmation spécifique. Sur les processeurs Nvidia, elle se fait typiquement en CUDA, un environnement qui permet de programmer en C avec extensions spécifiques, avant la compilation du code vers le GPU. La figure 2 de l'article montre ainsi ce que l'on peut gagner de temps sur une tâche simple (déterminer le prochain saut pour un paquet IPv6) par rapport au CPU traditionnel : à partir de 300 paquets en cours de traitement (ce qui permet d'exploiter à fond le parallélisme du GPU), le CPU traditionnel est battu et largement battu si des milliers de paquets se présentent à peu près en même temps.

Faire des calculs très vite ne sert à rien si on ne peut pas nourrir le processeur en données au même rythme. La capacité théorique de PCIe est de 8 Go/s mais, en pratique, le travail nécessaire, par exemple à la DMA, ne permet pas d'aller aussi vite. C'est le second gros travail de l'équipe de PacketShader.

La section 3 de l'article décrit la machine utilisée. Avec ses deux Xeon "*quadricore*", ses deux GPU Nvidia GTX 480, et ses quatre cartes réseau Intel 82599 (chacune ayant deux ports à 10 Gb/s), ce n'est quand même pas le PC de bureau de M. Michu ! Mais, à 7 000 US $, il est infiniment moins cher qu'un routeur Cisco. L'étude attentive de la bête a montré aux auteurs de l'article quelques comportements curieux (comme un débit plus élevé lorsque les données vont de la mêmoire au GPU qu'en sens inverse, si la machine a deux "*I/O Hub*"). Bon, et le logiciel ? Ubuntu avec le noyau Linux standard. Et bien sûr un bon pilote pour les cartes Ethernet et le SDK CUDA pour programmer le GPU.

Deux grands chantiers ont permis de transformer le PC en un routeur de course : l'utilisation des GPU (section 5 de l'article), pour décider de ce qu'on fait du paquet, et pour des opérations comme le chiffrement, et l'optimisation des entrées/sorties des paquets (section 4). Voyons d'abord le GPU.

Le modèle de programmation de ces puces est très différent de celui d'un processeur classique. Il a donc fallu développer un logiciel entièrement nouveau. Celu-ci tourne en mode utilisateur (alors que la transmission de paquets IP sur Linux se fait traditionnellement entièrement dans le noyau, pour éviter de perdre du temps en franchissement de la frontière entre le noyau et l'application). Le but est de pouvoir utiliser des bibliothèques comme CUDA (pour l'interface avec le GPU) et OpenSSL (pour mettre en œuvre IPsec). Le mode utilisateur fournit en outre un environnement de développement bien plus facile. Mais comment surmonter sa relative lenteur (trois fois plus lent, selon les mesures qui avaient été faites avec un autre système) ? Les auteurs citent le regroupement des paquets en lots (un seul appel système peut ainsi envoyer plusieurs paquets), une stricte correspondance entre les fils d'exécution et les files d'attente des paquets, pour éviter que deux fils ne se battent pour accéder à une file d'attente et enfin un mécanisme d'attente active sur les cartes réseaux (plutôt que d'attendre les interruptions, car une interruption est relativement lente, trop pour en faire une par paquet) mais suffisamment intelligent pour ne pas priver les autres processus (comme le serveur BGP) du processeur : PacketShader lit la file d'attente puis, lorsqu'elle est vide, remet les interruptions en service. À la prochaine interruption, il coupera les interruptions, lira la file, et ainsi de suite.

Je ne vais pas résumer ici toutes les optimisations utilisées pour que le routeur aille vite, lisez plutôt l'article original. Mais un des points importants est que, quoique l'environnement de développement CUDA permettre de prendre un programme C et de le faire tourner sur le GPU avec le minimum d'efforts, les performances demandées pour un routeur 40 Gb/s nécessitent de modifier profondément le modèle séquentiel de traitement des paquets, et de bien réflechir à ce qu'on garde sur le CPU classique (lent mais très général) et ce qu'on envoie sur le GPU (rapide pour ce qu'il sait bien faire mais très lent sur certaines tâches, notamment celles ayant peu de parallélisme). La copie des données vers le GPU ayant un coût, il faut être sûr que le gain de vitesse compensera ce coût.

Et les entrées/sorties ? Les auteurs estiment que la couche réseau de Linux est trop inefficace pour la plupart des opérations d'un routeur à très haut débit. Par exemple, chaque paquet nécessite l'allocation et la déallocation de deux tampons (un pour le paquet et un pour les métadonnées, qui sont par ailleurs trop grosses, car un routeur n'a pas besoin de beaucoup d'informations sur le paquet qu'il route). PacketShader a donc son propre système, qui comprend un seul énorme tampon pour les paquets. Les métadonnées sont, elles, réduites aux huit octets indispensables (208 dans le code de Linux original).

Autre moyen de gagner du temps : regrouper les opérations de plusieurs paquets. Un routeur « logiciel » naïf laisserait la carte Ethernet lever une interruption par paquet, puis copierait ce paquet en mémoire, puis déciderait de son sort et ferait ensuite un appel système pour passer le paquet au noyau, puis au pilote de la carte réseau. Chacune de ces opérations est coûteuse et on ne peut pas la faire pour **chaque** paquet, sans diminuer sérieusement les performances. Au contraire, RouteBricks regroupe plusieurs paquets en une seule opération et PacketShader étend ce principe même à l'application de Transmission. Le système d'entrées/sorties des paquets de PacketShader a été testé séparement (sans décision de Transmission suivant l'adresse de destination). En émission seule, les 80 Gb/s théoriques sont atteints, en réception seule, le routeur plafonne à 60 Gb/s (sans doute en raison de l'asymétrie décrite plus haut). En combinant les deux (cas d'un vrai routeur, qui reçoit des paquets et les fait suivre), les 40 Gb/s sont atteints.

Pour terminer l'article, deux intéressantes sections, la 7, consacrée à une discussion sur les limites de PacketShader et les moyens de les surmonter, et la 8, consacrée aux travaux similaires qui ont déjà été menés.

Parmi les limites, il y a le fait que PacketShader, qui gère la partie Transmission du routeur, n'a pas encore été connecté avec un logiciel qui gérerait la partie Contrôle, comme par exemple Quagga. Les performances pourraient se dégrader si le rythme de mise à jour de la table de routage était trop élevé, un problème que connaissent tous les routeurs, « matériels » ou « logiciels ».

Parmi les inconvénients qu'il y a à utiliser un GPU, la consommation électrique. Un GPU consomme plus qu'un CPU « équivalent » (594 watts pour la machine de test, contre 353 sans ses processeurs graphiques).

Le site Web officiel du projet est <http://shader.kaist.edu/packetshader/>. Comme indiqué plus haut, il ne semble pas que le code source soit disponible. Merci à Emmanuel Quemener pour ses remarques stratégiques et à tous les participants au FRnog <http://www.frnog.org/> pour la très intéressante discussion <http://www.mail-archive.com/frnog@frnog.org/msg11882.html> qui a suivi la publication de cet article.