Une Plate-forme d'Émulation Légère pour Étudier les Systèmes Pair-à-Pair

Lucas Nussbaum

RenPar'17 / SympA'2006 / CFSE'5 / JC'2006
Canet en Roussillon, 4 au 6 octobre 2006

Une Plate-forme d'Émulation Légère pour Étudier les Systèmes Pair-à-Pair *

Lucas Nussbaum

Laboratoire Informatique et Distribution - IMAG / Projet INRIA Mescal
ENSIMAG - Antenne de Montbonnot - ZIRST
51 avenue Jean Kuntzmann, 38330 Montbonnot Saint-Martin, France
Lucas.Nussbaum@imag.fr

Résumé
Les méthodes actuelles d'étude des systèmes pair-à-pair (modélisation, simulation, et exécution sur des systèmes réels) montrent souvent des limites sur les plans du passage à l'échelle et du réalisme. Cet article présente P2PLab, une plate-forme pour l'étude et l'évaluation des systèmes pair-à-pair, qui combine l'émulation (utilisation de l'application réelle à étudier à l'intérieur d'un environnement synthétique) et la virtualisation. Après la présentation des caractéristiques principales de P2PLab (émulation réseau distribuée, virtualisation légère), nous montrons son utilité lors de l'étude du système de diffusion de fichiers BitTorrent.

Mots-clés : systèmes pair-à-pair, évaluation, émulation, virtualisation, BitTorrent

---

1. Introduction

Les systèmes pair-à-pair et les algorithmes utilisés par ces systèmes ont fait l'objet d'un intérêt considérable ces dernières années. Ils sont devenus plus performants, mais aussi plus complexes, ce qui a eu pour conséquence de les rendre plus difficiles à concevoir, à vérifier et à évaluer. Il devient nécessaire de pouvoir vérifier qu'une application se comportera correctement même lorsqu'elle sera exécutée sur des milliers de nœuds, ou de pouvoir comprendre des applications conçues pour fonctionner sur un tel nombre de nœuds.
Les applications distribuées sont traditionnellement étudiées en utilisant la modélisation, la simulation, et l'exécution sur des systèmes réels. La modélisation et la simulation consistent à utiliser un modèle du fonctionnement de l'application dans un environnement synthétique. Cette méthode est largement utilisée, et permet d'obtenir des résultats de bonne qualité assez facilement. Toutefois, il est souvent difficile de simuler efficacement un grand nombre de nœuds en utilisant un modèle complexe de l'application : un compromis entre le nombre de nœuds et la précision du modèle est souvent obligatoire.
La solution opposée consiste à exécuter l'application réelle à étudier dans un environnement d'expérimentation réel comme PlanetLab [1]. Mais ces plates-formes sont difficiles à contrôler et à modifier (afin de les adapter à des conditions d'expériences particulières), et les résultats sont souvent difficiles à reproduire, car les conditions environnementales peuvent varier de

* Ce travail a été réalisé au sein du laboratoire ID-IMAG. Les différentes évaluations ont été faites en utilisant les ressources des projets GridExplorer et Grid5000 (plus d'informations sur `http ://www.grid5000.fr/`).

manière importante entre les expériences. L'expérimentation sur ce type de plate-forme est nécessaire lors du développement d'un système pair-à-pair, mais ces expériences ne sont pas suffisantes, et l'utilisation d'autres méthodes d'évaluation devient de plus en plus importante [2].
Entre ces deux approches, cet article explore une solution intermédiaire combinant émulation et virtualisation, et montre que cette approche permet d'obtenir des résultats intéressants.
Mais tout d'abord, l'émulation et la virtualisation doivent être distinguées :

L'émulation consiste à exécuter l'application à étudier dans un environnement modifié afin de correspondre aux conditions expérimentales souhaitées. Il est nécessaire de déterminer quelles sont les ressources à émuler (et avec quelle précision) : il s'agit souvent d'un compromis entre réalisme et coût. Par exemple, lors de l'étude de systèmes pair-à-pair, l'émulation réseau est importante, alors que l'émulation précise des entrées/sorties sur le disque dur n'est probablement pas nécessaire. L'émulation est souvent coûteuse (temps processeur, mémoire), et son coût est souvent difficile à évaluer, car il dépend à la fois de la qualité de l'émulation et de ses paramètres : émuler un réseau à latence importante nécessitera une quantité de mémoire plus importante qu'émuler un réseau à faible latence.

La virtualisation de ressources permet de partager une ressource réelle entre plusieurs instances d'une application. Elle est nécessaire afin de permettre l'étude d'un grand nombre de nœuds. Dans le cas des systèmes distribués, la virtualisation permet d'exécuter plusieurs instances de l'application ou du système d'exploitation sur chaque machine physique. Bien entendu, comme les ressources réelles de la machine physique sont partagées entre les différentes instances, l'équité est un problème important. Comme la précision de l'émulation, le niveau d'équité de la virtualisation est un compromis entre qualité et coût.

2. État de l'art

Beaucoup de travaux ont été effectués récemment sur la virtualisation, avec différentes approches. Linux Vserver [3] est une modification du noyau Linux qui y ajoute des *contextes* et contrôle les interactions entre ces contextes, permettant à plusieurs environnements de partager le même noyau de manière invisible pour les applications, avec un très faible surcout. User Mode Linux [3] est un portage du noyau Linux vers un processus Linux. Et Xen [3] utilise la *para-virtualisation* (exécution de systèmes légèrement modifiés au-dessus d'un système hôte) pour permettre d'exécuter simultanément plusieurs systèmes d'exploitation.
Le réseau est l'aspect le plus important à contrôler lorsqu'on s'intéresse aux systèmes pair-à-pair. Des outils de bas niveau permettent d'émuler différentes caractéristiques de liens (en faisant varier la bande passante et la latence), tandis que des outils de haut niveau permettent de construire des topologies synthétiques complexes.
Dummynet [4], sous FreeBSD, est l'émulateur réseau le plus utilisé. NISTNet [5] (sous Linux 2.4) et Linux Traffic Control (TC) [6] sous Linux 2.6 ont des fonctionnalités similaires. Ces outils se positionnent au niveau paquet et ordonnancent les paquets entrants et/ou sortants du système pour contrôler le débit et le délai, en émulant aussi des problèmes comme la perte de paquets.
Des outils de plus haut niveau permettent de recréer des topologies virtuelles : NetBed [7] combine des nœuds réels (utilisant des connexions RTC ou ADSL), des nœuds utilisant Dummynet, et des nœuds simulés (en utilisant Network Simulator Emulation Layer) pour fournir un environnement expérimental. Modelnet [8] utilise des nœuds d'une grappe partitionnée en deux sous-ensembles : l'application à étudier est exécutée sur les *edge nodes* tandis que les *Modelnet*

core nodes émulent une topologie réseau. Modelnet utilise une phase de *distillation* pour faire un compromis entre réalisme et passage à l'échelle, permettant ainsi d'émuler le réseau sur un faible nombre de nœuds.
Les outils existants de virtualisation ciblent un grand réalisme et ont un facteur de virtualisation (nombre de machines virtuelles / nombre de machines physiques) assez faible. Ils virtualisent un système d'exploitation complet (noyau, bibliothèques, applications) alors que cela n'est souvent pas nécessaire dans le cadre de l'étude des systèmes pair-à-pair, qui sont généralement des applications *verticales*, ayant peu de dépendances avec le reste du système.
Un autre problème est l'émulation de la topologie du réseau : les outils existants visent une émulation réaliste du coeur du réseau (congestion, routage, ...). Or la plupart des applications pair-à-pair sont utilisées par des personnes individuelles sur des liens de type ADSL. Si certains aspects du coeur du réseau sont importants (la latence, notamment, pour des expériences impliquant des problèmes de localité), l'accent peut être mis sur l'émulation du lien entre les nœuds et leur fournisseur d'accès à Internet, qui est le goulot d'étranglement dans la plupart des cas.

3. P2PLab

3.1. Présentation générale

P2PLab est notre outil d'étude des systèmes pair-à-pair utilisant l'émulation. Il vise une grande efficacité (un grand nombre de nœuds doivent pouvoir être étudiés sur un faible nombre de nœuds physiques) et un bon passage à l'échelle (des expériences avec plusieurs milliers de nœuds doivent être possibles).
P2PLab virtualise au niveau des processus, pas au niveau du système d'exploitation comme les autres outils de virtualisation le font. Il utilise FreeBSD pour pouvoir utiliser Dummynet pour l'émulation réseau. Une approche décentralisée est utilisée pour émuler les topologies réseau, permettant un meilleur passage à l'échelle.
Nous avons commencé par vérifier que FreeBSD offrait des capacités suffisantes de passage à l'échelle (nombre de processus concurrents et équité entre ces processus). Ce travail est présenté en détail dans [9]. Dans la section suivante, nous présenterons le système de virtualisation proposé par P2PLab, puis nous nous intéresserons à son modèle d'émulation réseau.

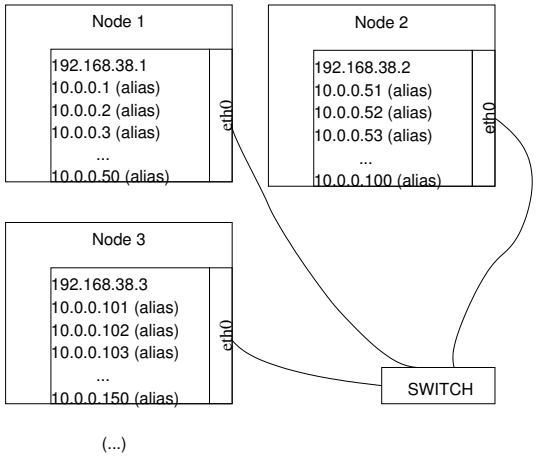

FIG. 1 – Sur chaque machine physique, les adresses IP des nœuds virtuels sont configurées comme des alias d'interface.

3.2. Virtualisation

P2PLab virtualise au niveau de l'identité réseau des processus : les instances exécutées sur la même machine physique partagent toutes les ressources (système de fichiers, mémoire, etc) comme des processus normaux, mais chaque processus virtualisé a sa propre adresse IP sur le réseau. L'adresse IP de chaque machine physique est conservée à des fins d'administration, tandis que les adresses IPs des nœuds virtuels sont configurées comme des *alias d'interface*, comme montré sur la figure 1 (la plupart des systèmes Unix, dont Linux et FreeBSD, permettent d'affecter plusieurs adresses IP à

une même interface réseau à travers un système d'alias). Nous avons vérifié que les alias d'interface ne provoquaient pas de surcoût par rapport à l'affectation normale d'une adresse IP à une interface.

Pour associer une application à une adresse IP particulière, nous avons choisi d'intercepter les appels systèmes liés au réseau. Plusieurs solutions étaient possibles : modifier l'application étudiée, lier l'application avec une bibliothèque spécifique, utiliser `ptrace` pour intercepter les appels systèmes, modifier le noyau, ou modifier la bibliothèque C.

Dans P2PLab, nous avons choisi de modifier la bibliothèque C de FreeBSD, ce qui est un bon compromis entre complexité et efficacité. Nous avons modifié les fonctions `bind()`, `connect()` et `listen()` pour toujours se restreindre à l'adresse IP spécifiée dans la variable d'environnement `BINDIP`. Une évaluation précise de cette approche [9] a montré que le surcoût engendré était très faible, même dans le pire cas.

3.3. Émulation réseau

Les émulateurs de topologies réseau existants comme Modelnet [8] visent une émulation réaliste du coeur du réseau (routage et interactions entre les *systèmes autonomes* (AS) ou entre les principaux routeurs). Mais la plupart des applications pair-à-pair sont exécutées sur des nœuds situés *en bordure d'Internet* [10] : par exemple sur des ordinateurs personnels d'abonnés à l'ADSL. Du coup, même si le trafic dans le coeur du réseau peut influencer certains aspects du comportement des systèmes pair-à-pair (la congestion dans le coeur du réseau peut influencer la latence, par exemple), le principal goulot d'étranglement reste le lien entre le système participant et son fournisseur d'accès à Internet (*Internet Service Provider*).

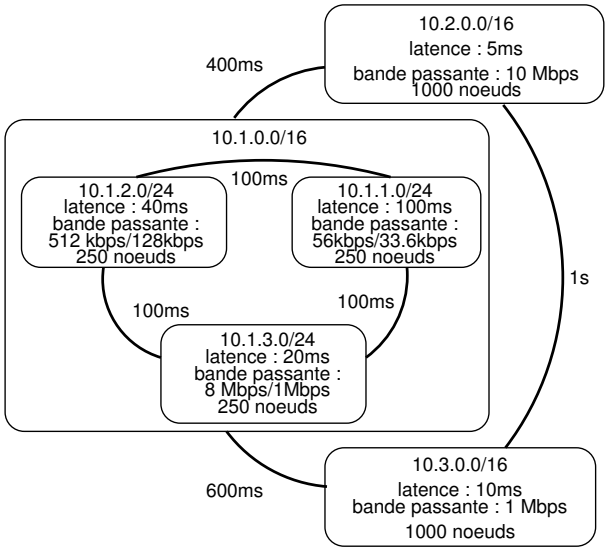

FIG. 2 – Topologie émulée

Il est donc possible de modéliser le réseau Internet en mettant l'accent sur le point de vue du nœud participant, en excluant ce qui est moins important de ce point de vue là.

Le modèle d'émulation de P2PLab permet de contrôler :

- la bande passante, la latence et le taux de perte de paquets sur les liens réseaux entre les nœuds et leur fournisseur d'accès à Internet ;
- la latence entre des groupes de nœuds, permettant d'étudier des problèmes mettant en jeu la localité des nœuds, par exemple.

La figure 2 montre un exemple de topologie que nous avons émulée avec P2PLab. Cette évaluation, ainsi que toutes les suivantes, a été réalisée sur la plate-forme GridExplorer, qui fait partie du projet Grid5000. Les machines utilisées sont des Bi-Opteron 2 Ghz avec 2 Go de mémoire vive et un réseau gigabit ethernet.

Nous avons mesuré la latence entre les nœuds 10.1.3.207 et 10.2.2.117 (il n'y avait pas d'autre trafic entre les nœuds). La latence mesurée de 853 ms se décompose en :

- 20 ms de délai lorsque le paquet est parti de 10.1.3.207 (délai pour le groupe 10.1.3.0/24) ;
- 400 ms de délai entre le groupe 10.1.0.0/16 et le groupe 10.2.0.0/16 ;
- 5 ms lorsque le paquet est arrivé sur 10.2.2.117 (délai pour le groupe 10.2.0.0/16) ;

– 425 ms lorsque le paquet est revenu de 10.2.2.117 à 10.1.3.207, comme ci-dessus ;
– 3 ms de surcoût, dûs au transit sur le réseau de la grappe, et surtout au parcours des règles du pare-feu.

Dans P2PLab, l'émulation réseau est faite d'une manière décentralisée : chaque nœud physique se charge de l'émulation réseau pour les nœuds virtuels qu'il héberge. Sur chaque nœud, l'émulation est faite avec Dummynet [4], par l'ajout de règles dans le pare-feu du système hôte. Nous avons montré dans [9] que le surcoût engendré par l'ajout de règles restait faible même avec un nombre important (plusieurs centaines) de nœuds virtuels sur chaque nœud physique.

4. Évaluation d'un système pair-à-pair avec P2PLab : le cas de BitTorrent

Dans cette partie, nous allons réaliser quelques expériences sur BitTorrent [11] en utilisant P2PLab. Dans un premier temps, nous nous attacherons rapidement à montrer l'aptitude de P2PLab à évaluer un tel système pair-à-pair. Puis nous évaluerons certaines techniques permettant à un nœud égoïste d'obtenir de meilleures performances (temps de récupération d'un fichier plus court, ou récupération en consommant moins de bande passante).

BitTorrent est un système pair-à-pair de distribution de fichiers très populaire. Il fournit de très bonnes performances grâce à un mécanisme de réciprocité complexe en s'assurant que les nœuds récupérant un fichier coopèrent en envoyant les morceaux déjà récupérés vers d'autres nœuds. Lors d'un transfert avec BitTorrent, les clients contactent un *tracker* pour récupérer une liste d'autres nœuds participant au téléchargement de ce fichier, puis se connectent directement aux autres nœuds. Les nœuds disposant du fichier complet sont appelés *seeders*.

BitTorrent a déjà été étudié par l'analyse de traces d'une utilisation à grande échelle [12, 13], ainsi qu'à l'aide de modélisation [14] et de simulation [15]. Cependant, ces travaux n'ont jamais été comparés à des évaluations à grande échelle sur des systèmes réels, ou à des études utilisant l'émulation. BitTorrent est avant tout un travail d'ingénierie, et non un prototype de recherche : plusieurs parties de son code sont très complexes, et le nombre élevé de constantes et de paramètres utilisés dans les algorithmes le rendent très difficile à modéliser avec précision.

Pour toutes les expériences qui suivent, nous avons utilisé les ressources déjà décrites à la section 3.3. Les clients BitTorrent utilisés sont BitTorrent 4.4.0 (écrit en Python par l'auteur originel de BitTorrent) et une version modifiée de CTorrent 3.1.4[2] (écrite en C++).

4.1. Rapport de virtualisation

Dans [9], nous avons montré qu'il était possible de réaliser le transfert d'un fichier sans surcoût lié à la virtualisation avec les conditions qui suivent. Nous avons également montré que le premier facteur limitant était le réseau de la grappe sous-jacente, et donc les paramètres réseau de l'émulation. Nous nous replaçons donc dans ces conditions expérimentales pour les expériences qui suivent :

– au plus 80 clients (machines virtuelles) par nœud physique ;
– bande passante descendante de 2 mbps ;
– bande passante ascendante de 128 kbps ;
– latence (en entrée et en sortie des nœuds) de 30 ms.

Ces conditions se veulent proches de celles rencontrées sur des connexions à Internet de type ADSL.

[2] Disponible sur http://www.rahul.net/dholmes/ctorrent/.

4.2. Passage à l'échelle

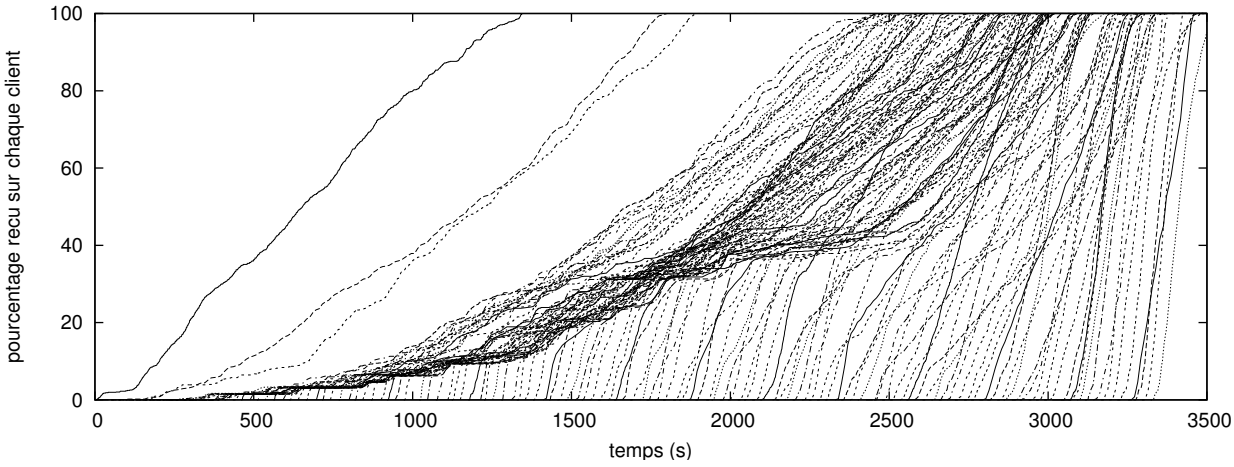

FIG. 3 – Évolution du transfert d'un fichier de 16 Mo entre 13031 clients sur quelques nœuds sélectionnés mais représentatifs

Dans cette expérience, nous cherchons à montrer qu'il est possible de réaliser des expériences avec un nombre important de nœuds. Nous transférons un fichier de 16 Mo en utilisant 13040 nœuds répartis sur 163 machines physiques. Parmi ces 13040 nœuds, il y un *tracker*, et 8 nœuds (*seeders*) qui disposent du fichier complet dès le début de l'expérience afin de le transférer vers les autres nœuds. Le *tracker* et les *seeders* sont démarrés au début de l'expérience, tandis que les clients sont démarrés toutes les 0.25 secondes. Les clients (*seeders* compris) à numéro pair utilisent CTorrent, tandis que les clients à numéro impair utilisent BitTorrent 4.4.0. Lorsqu'un client termine le transfert, il reste présent afin d'aider les autres clients.

La figure 3 montre l'avancement du transfert sur les clients numérotés 101, 200, 301, 400, 501, ... On constate que, malgré des dates de démarrage décalées, la plupart des clients terminent leur transfert à des dates proches, ce qui est confirmé par la figure 4. Cela est dû au mécanisme de réciprocité de BitTorrent : les clients préfèrent aider les clients ayant reçu peu de données que des clients proches de la fin de leur transfert, car ces derniers, une fois le transfert terminé, pourraient quitter le système et ne plus contribuer aux transferts des autres clients.

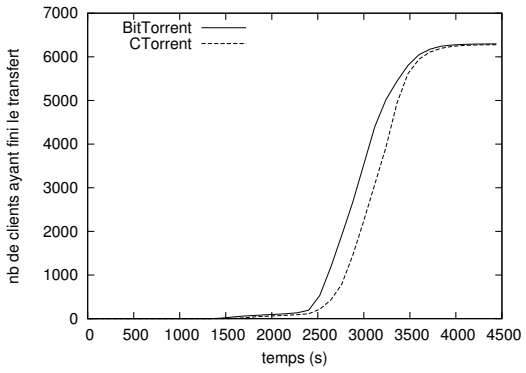

FIG. 4 – Temps de complétion du transfert sur les différents nœuds

4.3. Comparaison de différentes implémentations de BitTorrent

Puisqu'il permet d'exécuter directement des applications réelles, P2PLab est adapté à la comparaison d'implémentations différentes du même protocole. Nous l'utilisons donc pour comparer BitTorrent, le client de référence développé par l'auteur du protocole [11], et CTorrent, qui a été utilisé dans des travaux visant à montrer certaines limites du protocole [16]. Nous réalisons le transfert d'un fichier de 16 Mo entre 500 clients et 8 *seeders* (4 utilisent BitTorrent, 4 CTorrent) en nous plaçant dans les mêmes

conditions réseaux que précédemment, et lançons tous les clients en même temps, au début de l'expérience.
Pour vérifier qu'un client BitTorrent est performant, il est important de vérifier qu'il est performant à la fois lorsqu'il ne communique qu'avec des clients identiques, et lorsqu'il communique avec des clients différents. Nous comparons donc les performances de BitTorrent et CTorrent lorsqu'ils sont lancés seuls (500 clients du même type) ou en mélangeant des clients BitTorrent et CTorrent (250 clients de chaque type). Les résultats sont présentés en figure 5, et montrent que BitTorrent 4.4.0 semble plus performant dans les deux cas avec ces paramètres d'expérience. Il sera toutefois important de compléter cette expérience par d'autres expériences avec des paramètres différents.

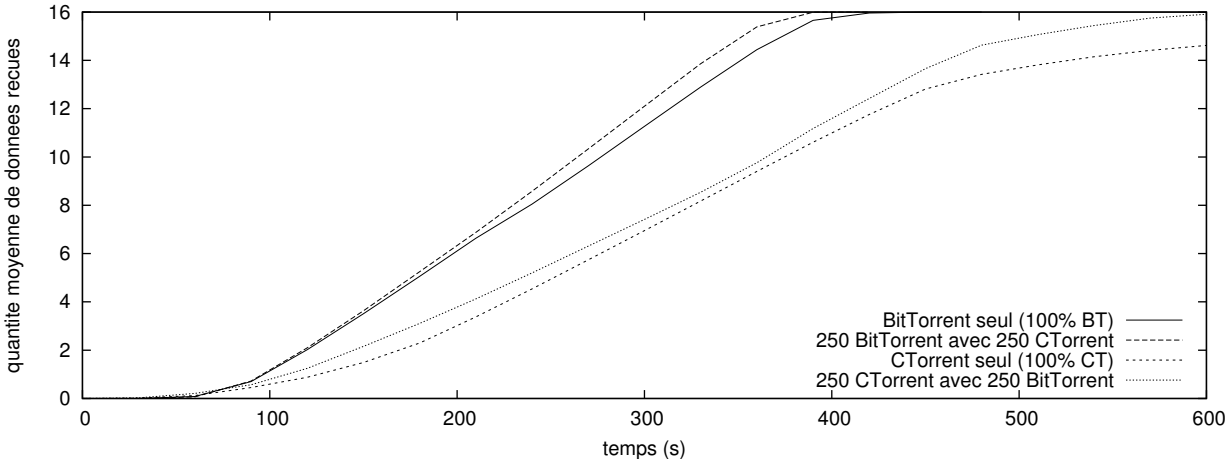

FIG. 5 – Comparaison de BitTorrent et CTorrent

5. Travaux futurs

Même si cette étude de BitTorrent démontre l'utilisabilité de P2PLab, plusieurs aspects doivent encore être approfondis. Tout d'abord, il sera important de valider notre modèle de topologie réseau en le comparant à des plates-formes réelles comme PlanetLab [1] ou DSL-Lab, ainsi qu'aux générateurs de topologie couramment utilisés par les simulateurs. Nous devrons aussi comparer P2PLab aux autres outils d'émulation de systèmes distribués comme Modelnet [8].
Le système de virtualisation que nous avons développé ne permet pas pour l'instant un partage paramétrable du temps processeur ou de la mémoire, ce qui le rend pour l'instant inutilisable pour des applications orientées calcul. Des aménagements sont donc à prévoir afin de pouvoir étudier des applications de type *Desktop Computing*.
Enfin, les expériences réalisées sur BitTorrent utilisent des paramètres peu réalistes. Il sera utile de coupler notre outil avec des solutions permettant de générer un environnement et des comportements réalistes.

6. Conclusion

Avec l'augmentation des ressources disponibles sur les ordinateurs, la virtualisation et l'émulation ont fait l'objet de beaucoup d'intérêt ces dernières années. Mais au-delà de leur usage habituel, ils peuvent être combinés pour construire des plates-formes expérimentales permettant une grande configurabilité et la reproductibilité des expériences. Ces plates-formes sont par-

ticulièrement utiles dans le cadre de l'étude des systèmes distribués, où le nombre de nœuds participant est un critère primordial.
Dans cet article, nous avons présenté P2PLab, un outil d'étude des systèmes pair-à-pair, et nous avons montré son utilité à travers quelques expériences sur différentes implémentations du protocole de diffusion de fichiers BitTorrent.
Nous avons aussi proposé un modèle simple de topologies réseaux virtuelles adapté à l'étude des systèmes pair-à-pair : à la différence des modèles de topologie couramment utilisés, nous modélisons le réseau du point de vue du nœud participant, en masquant une partie des interactions se déroulant dans le coeur du réseau. Ce modèle permet aussi une implémentation facile dans des outils d'émulation.
Notre travail permet aussi de faire penser qu'il n'est pas forcément nécessaire de recourir à des systèmes de virtualisation lourds (virtualisant l'ensemble des ressources). Notre approche plus légère permet d'augmenter le rapport de virtualisation en ne virtualisant que ce qui est absolument nécessaire : l'identité réseau de l'application étudiée.

Bibliographie

1. Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak et Mic Bowman. PlanetLab : An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3) :00–00, July 2003.
2. Andreas Haeberlen, Alan Mislove, Ansley Post et Peter Druschel. Fallacies in evaluating decentralized systems. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*, February 2006.
3. The Linux VServer Project. http ://www.linux-vserver.org/Linux-VServer-Paper.
4. Luigi Rizzo. Dummynet and forward error correction. In *Proceedings of the FreeNIX Track : USENIX 1998 annual technical conference*. Usenix, 1998.
5. Mark Carson et Darrin Santay. NIST Net : a Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3) :111–126, 2003.
6. Stephen Hemminger. Network emulation with NetEm. In *linux.conf.au*, 2005.
7. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb et Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI) :255–270, 2002.
8. Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase et David Becker. Scalability and accuracy in a large-scale network emulator, 2002.
9. Lucas Nussbaum et Olivier Richard. Lightweight emulation to study peer-to-peer systems. In *Third International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P 06)*, Rhodes Island, Greece, 4 2006.
10. Ian T. Foster et Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03)*, 2003.
11. Bram Cohen. Incentives build robustness in BitTorrent. http://www.bittorrent.com, 2003.
12. Johan A. Pouwelse, Pawel Garbacki, Dick H.J. Epema et Henk J. Sips. The Bittorrent P2P file-sharing system : Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, feb 2005.
13. Mikel Izal, Guillaume Urvoy-Keller, Ernst W Biersack, Pascal A Felber, Anwar Al Hamra et Luis Garces-Erice. Dissecting BitTorrent : five months in a torrent's lifetime. In *PAM'2004, 5th annual Passive & Active Measurement Workshop, April 19-20, 2004, Antibes Juan-les-Pins, France / Also Published in Lecture Notes in Computer Science (LNCS), Volume 3015, Barakat, Chadi ; Pratt, Ian (Eds.) 2004, XI, 300p - ISBN : 3-540-21492-5*, Apr 2004.
14. Dongyu Qiu et Ramakrishnan Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *SIGCOMM '04 : Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 367–378, New York, NY, USA, 2004. ACM Press.
15. Ashwin R. Bharambe et Cormac Herley. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.
16. Nikitas Liogkas, Robert Nelson, Eddie Kohler et Lixia Zhang. Exploiting BitTorrent For Fun (But Not Profit). In *Proceedings of The 5th International Workshop on Peer-to-Peer Systems (IPTPS '06)*, Santa Barbara, CA, February 2006.