Lightweight Emulation to Study Peer-to-Peer Systems *

Lucas Nussbaum and Olivier Richard

Laboratoire Informatique et Distribution - IMAG
ENSIMAG - Antenne de Montbonnot - ZIRST
51 avenue Jean Kuntzmann, 38330 Montbonnot Saint-Martin, France
{Lucas.Nussbaum,Olivier.Richard}@imag.fr

Abstract

The current methods used to test and study peer-to-peer systems (namely modeling, simulation, or execution on real testbeds) often show limits regarding scalability, realism and accuracy. This paper describes and evaluates P2PLab, our framework to study peer-to-peer systems by combining emulation (use of the real studied application within a configured synthetic environment) and virtualization. P2PLab is scalable (it uses a distributed network model) and has good virtualization characteristics (many virtual nodes can be executed on the same physical node by using process-level virtualization). Experiments with the BitTorrent file-sharing system complete this paper and demonstrate the usefulness of this platform.

1. Introduction

Peer-to-peer systems have become more and more popular over the last few years, and this popularity often required changes that made them more and more complex. Due to this ever increasing complexity, the development and the study of peer-to-peer systems have become more difficult : we need ways to ensure that a peer-to-peer application will work properly on thousands of nodes, or ways to understand applications running on thousands of nodes.

Distributed applications are traditionally studied using mathematical modeling, simulation, and execution on a real system. Simulation consists in using a model of the application's code in a synthetic environment. This method is widely used, and gives valuable results easily. However, it is often difficult to simulate efficiently a large number of nodes using a complex model : a trade-off between the realism of the model and the number of nodes always has to be made.

On the other side, one can run the real application to study on a real-world experimentation platform like PlanetLab [5]. But the environment is then difficult to control and modify (since it depends heavily on the real system itself), and results are often difficult to reproduce (since the environmental conditions may vary a lot between experiments). This kind of real-world experiments is needed when developing a peer-to-peer system, but it doesn't satisfy all needs.

Between those two approaches, this paper explores an intermediate solution using emulation and virtualization, and shows that such an approach can provide interesting results when used to study peer-to-peer systems.

Emulation and Virtualization have to be distinguished :

Emulation consists in providing a modified environment to the studied application, to match the conditions of the experiment. Determining which resources to emulate (and with which precision) has to be considered as a trade-off between realism and cost (a precise emulation of conditions can be very CPU-intensive). For example, when studying peer-to-peer systems, network emulation is important, while emulation of different types of hard disks is probably not necessary. Emulation is often costly, and its cost is often difficult to evaluate, because it depends both on the quality of emulation and on the emulation parameters : emulating an high-latency network will be more expensive than emulating a low-latency one.

Virtualization of resources allows to share a resource between several instances of an application. It is

*This work has been done within the ID laboratory jointly supported by CNRS, INPG, INRIA, and UJF. Computer resources are provided by the Grid5000 platform (further information at `http://www.grid5000.fr/`).

required to be able to study a large number of co-existing nodes. In the field of distributed systems, virtualization allows to execute several instances of the application or the operating system on the same physical machine, thus increasing the number of nodes available for the experiment. Of course, since the resources of the physical machine are shared between instances, the fairness of this share is an important issue. Like the precision of emulation, the quality of the fairness is a trade-off.

2. Related Works

A lot of work occurred recently in the virtualization area, with different approaches. Linux Vserver [12] is a patch for the Linux kernel adding *contexts* and managing interactions between them, allowing several environments to share the same kernel with a very low overhead. User Mode Linux [7] is a port of the Linux kernel to a Linux process. And Xen [1] uses *paravirtualization* to give the ability to run simultaneously several operating systems. Those operating systems have to be modified to run over the Xen microkernel, which is in charge of sharing resources and providing virtual devices.

Regarding network emulation (by far the most important resource to control while studying peer-to-peer systems), there are both low-level tools which work at the packet level to emulate different network connections (varying bandwidth and latency) and higher level tools which allow to build a complex synthetic network topology.

The low level tools include Dummynet [14], which runs on FreeBSD and is the most popular network emulator, NISTNet [4] (which runs on Linux 2.4), and Linux 2.6's Traffic Control (TC) and iproute2 tools [15] (using NetEm [9]). Those tools schedule inbound and/or outbound packets to control bandwidth and delay, and emulate network problems such as packet loss.

Higher level tools use the former low-level tools to emulate complex topologies. NetBed [17] combines real nodes using RTC or DSL lines, nodes using Dummynet, and simulated nodes (using Network Simulator Emulation Layer) to provide experimental environments. Modelnet [16] uses cluster nodes split in two groups : the application under study runs on *edge nodes* while the *Modelnet core nodes* use Dummynet to emulate a network topology. Modelnet uses a *distillation* phase to make a trade-off between accuracy and scalability to be able to emulate the network on a small number of nodes.

The current virtualization tools target high realism and have a relatively poor virtualization ratio (number of virtual nodes / number of physical nodes). One of the problems is that they virtualize a full operating system (kernel, libraries, application). This is often not needed for peer-to-peer applications, since they are relatively well self-contained.

Another issue is network emulation : current tools target a realistic emulation of the core network (congestion, routing, etc inside the core links). Most peer-to-peer applications are used on the edge nodes of the Internet, often on home computer with DSL connections. Therefore, even if some aspects of the Internet core are important (e.g latency, for experiments involving locality), the emphasis can be put on the emulation of the link between the edge nodes and their Internet Service Provider. This link is the bottleneck in most (if not all) cases.

3. P2PLab

3.1. Overview

P2PLab is our emulation tool for studying peer-to-peer systems. It targets high efficiency (large number of virtual nodes can be studied on a low number of physical nodes), and scalability (experiments can be done with thousands of nodes).

P2PLab virtualizes at the process level, not at the system level, to provide better scalability. It runs on FreeBSD, since it uses Dummynet for network emulation. A decentralized approach is used to emulate network topologies, allowing better scalability.

First, we will verify that FreeBSD is a suitable platform for P2PLab by checking that its scheduler is scalable and provides a good level of fairness. Then, P2PLab's virtualization system will be described.

In a second part, the network emulation model of P2PLab will be presented.

3.2. Virtualization

While most virtualization systems virtualize on the operating system level, it is not mandatory here since the goal is to study peer-to-peer systems. It was therefore decided to virtualize the process' network identity by binding each process to its own IP address.

The FreeBSD operating system was chosen for P2PLab because of the availability of Dummynet [14], FreeBSD's network traffic shaper. But it was still necessary to test whether FreeBSD was a suitable platform to run a very large number of processes without compromising our experiment's results. FreeBSD has two schedulers : the classic 4BSD scheduler, and the more modern ULE scheduler (which is similar to Linux 2.6's

scheduler). It was decided to evaluate both. The evaluation took place on nodes of the GridExplorer system, part of the French grid research project Grid5000 [3]. The nodes are Dual-Opteron 2Ghz with 2 Gb of RAM and gigabit ethernet.

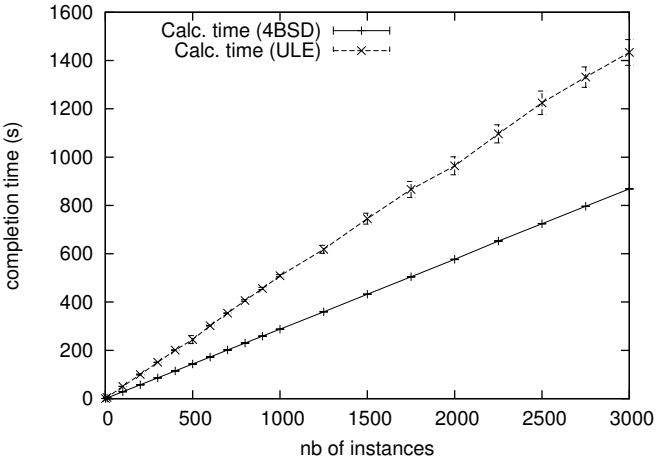

Figure 1. Total calculation for several instances of the same process. Swap is not used.

The first experiment was to evaluate the scheduler's ability to run multiple processes by running a large number of concurrent non-memory-intensive processes (each process, when started alone, needed about half a second of CPU time to complete). Figure 1 shows that total calculation time scales linearly with both the 4BSD and the ULE scheduler. However, the overhead is higher with the ULE scheduler.

Then, the same experiment was done with memory-intensive processes. Figure 2 shows clearly that results shouldn't be trusted as soon as swap is used (the figure only shows results with the 4BSD scheduler. Results with the ULE scheduler were similar).

Fairness between processes is also important. To measure it, concurrent instances of a CPU-intensive process were started every second, and the real run time of each instance was measured. Figure 3 shows that there are serious fairness problems in the ULE scheduler (some processes seem to run alone on the second CPU !). Fairness of the 4BSD and Linux 2.6 schedulers were better but not perfect. This kind of process schedulers try to maximize throughput and interactivity, and fairness is not a priority. Some different settings were tried for the timer interrupt frequency (HZ) and the quantum (time allocated to a process before switching to another one), but it didn't improve the results significantly.

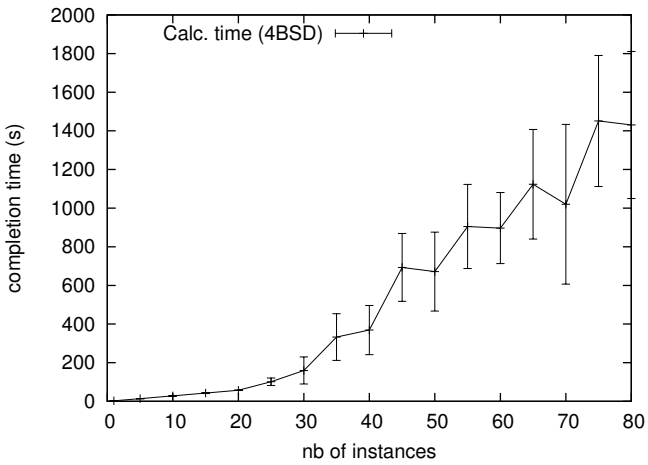

Figure 2. Total calculation for several instances of the same process. With more than 25 instances, swap is used.

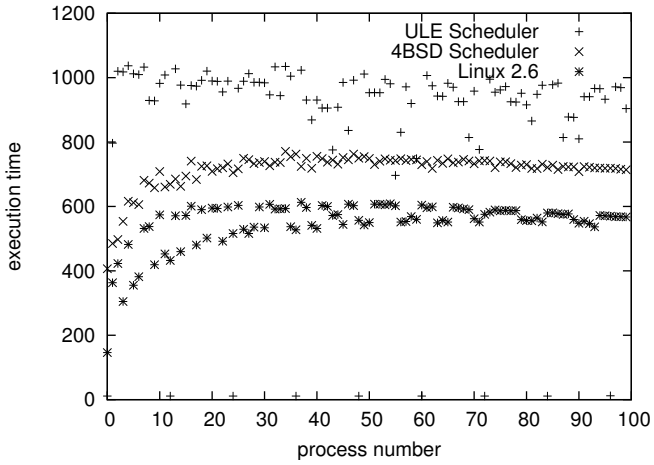

Figure 3. Completion times of concurrent instances of the same program, started with a 1 second interval.

After those first experiments, the 4BSD scheduler was chosen for P2PLab.

As said earlier, virtualization is made at the level of the process' network identity : instances on the same physical system share all resources (filesystem, memory etc.) as normal processes do. However, each process has its own IP address on the network. The main IP address of each physical system is kept for administration purposes. The IP addresses of the virtual nodes are configured as *interface aliases* as shown in figure 4 (most UNIX systems, including Linux and

FreeBSD, allow each network interface to be assigned several IP addresses through an aliasing system). Evaluation showed that *interface aliases* produced no overhead compared to the normal assignment of an IP address to an interface.

To avoid namespace conflicts, the adresses of the virtual nodes were chosen in different subnet. Figure 4 shows an example configuration using the 192.168.38/24 network for administration and the 10.0.0.0/8 network for virtual nodes.

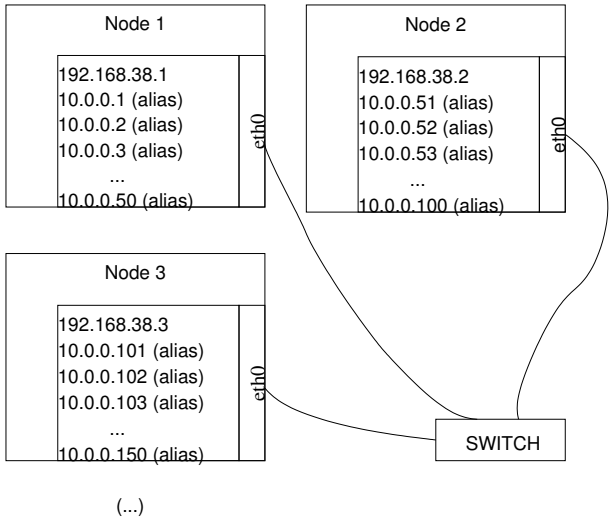

Figure 4. On each physical node, IP addresses for virtual nodes are configured as interface aliases.

To bind an application to a particular IP address, we chose to intercept some network system calls. Several options are available : modifying the application under study, linking the application with a specific library, using `ptrace()` to intercept system calls, modifying the kernel, or modifying the C library.

In P2PLab, we chose to modify the FreeBSD C library, which provided a good compromise between complexity and efficiency. We modified the `bind()`, `connect()` and `listen()` library calls using a naive approach :

- When `bind()` is called, the `my_addr` parameter is modified to restrict the bind to the IP address specified by the `BINDIP` environment variable.
- When `connect()` or `listen()` are called, `bind()` is called before to restrict `connect()` or `listen()` to the IP address in `BINDIP`. If another `bind()` was made before, this one will fail, but we ignore the error in this case.

This approach doubles the number of system calls for `connect()` and `listen()` and only works for TCP connections. A similar approach is possible for UDP.

Tests showed that this libc modification was working as expected with programs in Perl, Python, Ruby, TCL, C, Java (with Sun's JDK). The only case where this approach failed to work was statically compiled C programs.

Then, we evaluated the overhead of this approach during the establishment of TCP connections. The test program was connecting to a local server and disconnecting as soon as the connection was established. We measured that the duration of a connection/disconnection cycle was $10.22\mu s$ without the modification, to compare to $10.79\mu s$ with the modification : the cost of this approach is very low, even in the worst case.

3.3. Network Emulation

Current network topology emulators like ModelNet [16] target a realistic emulation of the core network. But most peer-to-peer applications run on nodes on the edge of the Internet : while the traffic in the core of the Internet can influence the peer-to-peer system behaviour (congestion between providers can increase latency, for example), the main bottleneck for end nodes is often the link between the user and its Internet service provider (ISP). Therefore, it is possible to model the Internet by reproducing what the end node really *sees*, excluding what is less important from the end node point of view.

P2PLab's emulation model allows to control :

- bandwidth, latency and packet loss rate on the network link between the end node and its ISP ;
- latency between group of nodes, allowing to study problems involving locality.

Figure 5 shows an example topology which we successfully emulated using P2PLab. Figure 6 presents some `ping` results between nodes of this topology (there was no other traffic between the virtual nodes when the pings were made). Let's use the measurement between 10.1.3.207 and 10.2.2.117 as an example. The latency of 853 ms can be decomposed in :

- 20 ms of delay when the packet goes out of 10.1.3.207 (delay configured for the 10.1.3.0/24 network group) ;
- 400 ms between the 10.1.0.0/16 network group and the 10.2.0.0/16 network group ;

- 5 ms when the packet arrives at 10.2.2.117 (delay configured for the 10.2.0.0/16 network group) ;
- 425 ms when the packet goes back from 10.2.2.117 to 10.1.3.207, decomposed as above ;
- 3 ms of overhead.

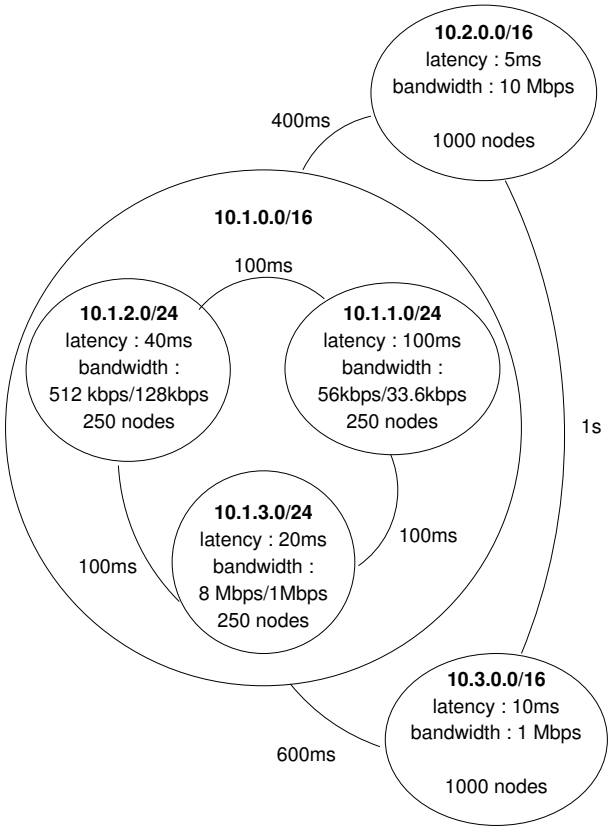

Figure 5. Emulated topology

Source	Destination	Latency * 2
10.1.1.137	10.1.1.228	462ms
10.1.1.205	10.1.3.204	476ms
10.1.1.84	10.3.0.66	1.46s
10.1.2.219	10.1.3.38	329ms
10.1.2.100	10.3.1.69	1.31s
10.1.3.207	10.2.2.113	853ms
10.2.0.51	10.2.3.48	23ms
10.3.3.181	10.3.0.133	43ms

Figure 6. Example results of `pings` between nodes of the topology from figure 5.

Network emulation is achieved in a decentralized way : each physical node is in charge of the network emulation for its virtual nodes. On each node, emulation is done with Dummynet [14], the FreeBSD traffic shaper integrated into FreeBSD's firewall (IPFW). Both incoming and outgoing packets are delayed by Dummynet. Several firewall rules are needed for each virtual node. For example, the physical node hosting 10.2.3.207 in the topology described in figure 5 will need :

- One rule for incoming packets for each virtual node hosted on this node ;
- One rule for outgoing packets for each virtual node hosted on this node ;
- One rule for each link between network groups, to delay packets going from network groups hosted on this system to other network groups.

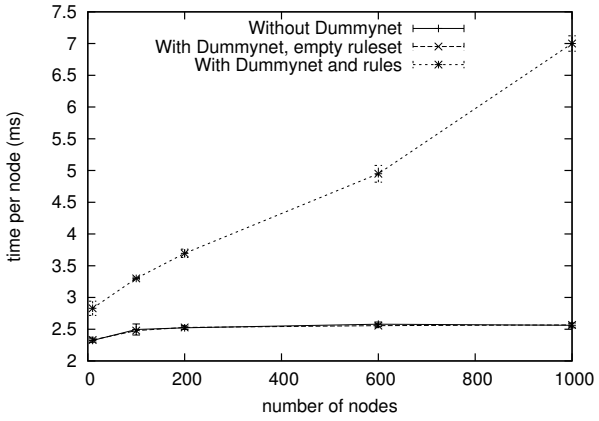

Figure 7. Overhead caused by Dummynet in a token ring

The overhead caused by Dummynet was evaluated by building a token ring on a single machine. The network of each virtual machine is emulated, causing two Dummynet rules to be added per virtual node. Figure 7 shows that the overhead caused by selecting the Dummynet rule to apply is important, because FreeBSD's IPFW examines firewall rules linearly. However, for a relatively low number of virtual nodes, the overhead is still reasonable (3.5 ms for 200 virtual nodes per physical node) compared to the latency of the networks we are going to emulate.

4. Study of BitTorrent with P2PLab

In this section, we confront our emulation platform with BitTorrent, showing that P2PLab is a suitable ex-

perimentation platform to study complex peer-to-peer system.

4.1. Introduction

BitTorrent [6] is a popular peer-to-peer file distribution system. It provides very good performance by ensuring that downloaders cooperate by sharing parts they have already downloaded through a complex reciprocation system. It has already been largely evaluated through analysis of large scale utilization [11, 10], analytical modeling [13] or simulation [2].

However, those works have never been compared to large scale studies on real world systems, or to studies using emulation. BitTorrent is an engineering work, not a research prototype, and several parts of its code are very complex. The large number of constants used as parameters of all the important algorithms makes it very hard to model accurately.

A BitTorrent 4.0.4 client was used for all experiments. It was slightly modified to allow data collection (a timestamp was added to the default output). The experiments took place on the GridExplorer system, already described in section 3.2 on page 2.

4.2. P2PLab Folding Ratio

P2PLab targets an high virtualization ratio : it should be possible to run many virtual nodes on the same physical system, thus allowing to do experiments with a very large number of nodes. But the concurrent execution of several instances of the application must not affect the results.

The first experiment compares the download of a 16 MB file by 160 clients. The file size is not important in BitTorrent, since the file is always divided in pieces of 256 KB. The file is provided by 4 seeders. All nodes (both downloaders and seeders) have a network connection with a download rate of 2 mbps, an upload rate of 128 kbps, and a latency of 30 ms, reproducing the conditions of a DSL connection. The clients are started with a 10 s interval. When the clients have finished the download of the file, they stay online and become seeders, continuing to upload data to the downloaders.

The 160 clients are deployed successively on 160 physical nodes, 16 physical nodes (10 virtual nodes per physical node), 8, 4 and 2 physical nodes.

Figure 8 shows the evolution of the download of all 160 clients when they are deployed on 160 physical nodes. One can see that with those parameters, all the cases of a BitTorrent download are represented :

- First (short) part of the download when only initial seeders are able to upload data ;

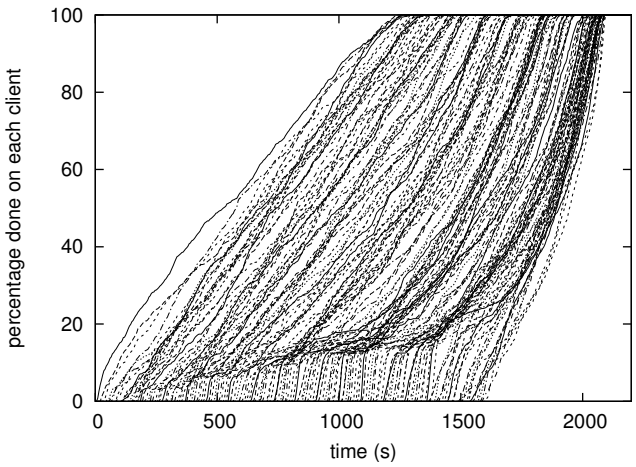

Figure 8. Evolution of the download of the 160 clients.

- Second part when all downloaders start contributing to each other ;
- Third part when the first downloaders become seeders and help other peers finishing their download faster.

Therefore, those parameters are a realistic workload to test P2PLab's virtualization capabilities.

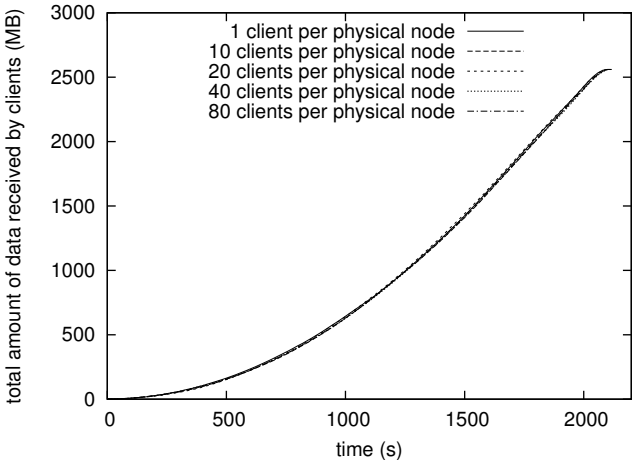

Figure 9. P2PLab folding ratio : total amount of data when downloading of a 16 MB file with 160 clients.

Figure 9 shows that the experiment took place without any overhead, even with 80 virtual nodes on each

physical node : results are nearly identical. The potential sources of overhead were investigated, and it was determined that the first limiting factor was the network speed : with other (slightly faster) emulated network settings, the platform's gigabit network was saturated by the downloads.

Other overhead sources were considered : during the experiment, we monitored the system load, the memory usage, and the disk I/O on every physical node. None of them was a problem during our experiments.

4.3. P2PLab Scalability

P2PLab targets high scalability : it should be possible to run experiments with a large number of nodes. We tried to study the download of a 16 MB file by 5754 clients, under the same network conditions as in the first experiment. The 5760 virtual nodes (5754 clients, 4 seeders, one tracker) are hosted on 180 physical nodes (32 virtual nodes per physical node). The clients are started every 0.25 s so the entire experiment took less than one hour.

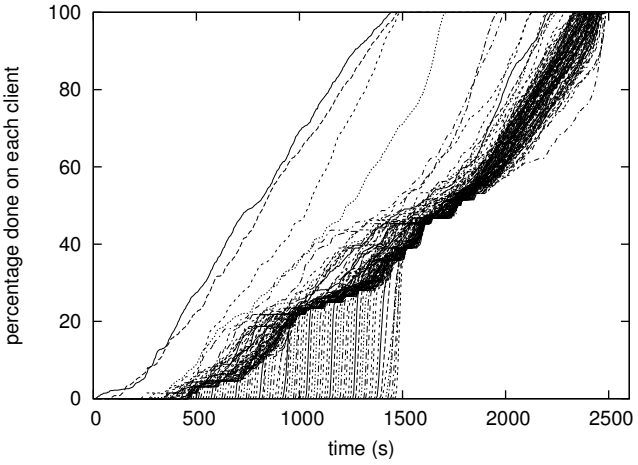

Figure 10. Evolution of the download of a 16 MB file between 5754 clients on a few selected clients.

The experiment succeeded, and figure 10 shows the evolution of the download on 115 selected clients (every 50 clients, so clients numbered 0, 50, 100, 150, ..., 5750)[1]. One can see that most clients finish their downloads nearly at the same time, at the end of the experiment. This is confirmed by figure 11, which shows the number of clients having completed their download over time.

[1] An high resolution colored graph with all the clients is available on `http://www-id.imag.fr/~nussbaum/perso.html/p2plab_bt.png`

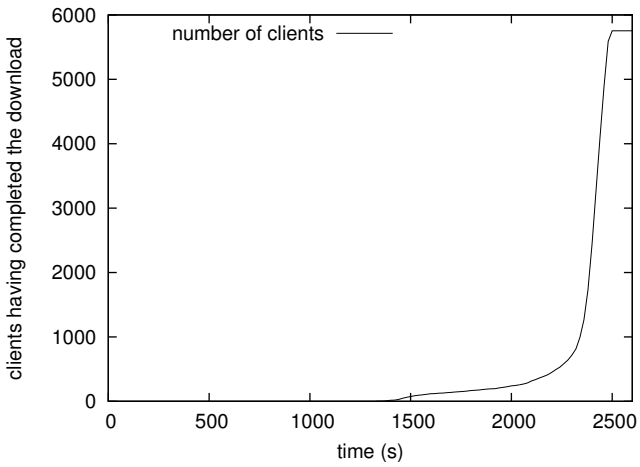

Figure 11. Number of clients having completed their download.

5. Future Work

Our decentralized network model doesn't consider the problem of congestion in the core links (between internet service providers). Its role in the performance of peer-to-peer systems need to be determined through experiments on PlanetLab [5] or DSL-Lab [8]. Then, we will be able to modify our network model to include it.

Another issue left open is the handling of shared resources. We have shown that one of the most important, CPU time, wasn't shared equally between processes. While the use of off-the-shelf software for emulation makes it difficult to provide fair-sharing, it would be interesting to improve the situation, or at least to provide it for some resources. Developing a more deterministic process scheduler would already allow solve the problem of the scheduler's fairness.

The use of FreeBSD for P2PLab wasn't easy : the dominant operating system on Grid5000 [3] (the experimentation platform we used) is Linux, and using FreeBSD instead required quite a lot of changes. After evaluating (and maybe improving) the comparable tools on Linux, we will consider a switch to Linux as the basis for P2PLab.

Another problem with FreeBSD is the way its firewall evaluates rules : as shown in section 3.3, the rules are evaluated linearly, which is clearly not optimal in the case of P2PLab, where we want to choose only one rule inside a large number of them. Implementing an hash-based mechanism for rules lookup would be much more efficient.

The experiments made with P2PLab on BitTorrent

are using sensible but unrealistic parameters. Coupling P2PLab with a realistic input generator would allow to easily generate realistic environments and behaviours.

Finally, we need to validate P2PLab against other peer-to-peer systems, to verify that it allows to answer a wide range of questions on a wide range of systems.

6. Summary and Conclusion

With the increase of the resources available on a single computer, virtualization and emulation have both been the target of a lot of research in the last years. Beyond their usual use case, they can be efficiently combined to build useful experimentation platforms, and are a promising tool to study peer-to-peer systems : they allow to use the real application on a large number of nodes in a configurable environment allowing reproduction of experiments.

This paper also contributes a simple network model different from the models usually used in network emulators to model the Internet : while those models concentrate on the core of the Internet and its routing interaction, P2PLab models the Internet from the end node's point of view.

This work also shows that, while most virtualization systems concentrate on providing a very accurate image of resources, a simpler approach might be sufficient in some cases. Our emulation platform only virtualizes what is needed to make the different virtual nodes look like real separate nodes from the outside : its network identity. This lightweight virtualization allows to maximize the virtualization ratio.

Our emulation platform, P2PLab, enabled us to perform some experiments on the BitTorrent peer-to-peer system. During those experiments, P2PLab proved to be scalable, easy to use and useful.

References

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[2] A. R. Bharambe and C. Herley. Analyzing and improving BitTorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[3] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.

[4] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *SIGCOMM Comput. Commun. Rev.*, 33(3):111–126, 2003.

[5] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[6] B. Cohen. Incentives build robustness in BitTorrent. http://www.bittorrent.com, 2003.

[7] J. Dike. A user-mode port of the Linux kernel. In *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta*, page 63. Usenix, 2000.

[8] DSL-Lab. http://www.lri.fr/~rezmerit/dsllab/.

[9] S. Hemminger. Network emulation with NetEem. In *linux.conf.au*, 2005.

[10] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. Al Hamra, and L. 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.

[11] J. A. Pouwelse, P. Garbacki, D. H. Epema, and H. J. Sips. The Bittorrent P2P file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, feb 2005.

[12] T. L. V. Project. http://www.linux-vserver.org/Linux-VServer-Paper.

[13] D. Qiu and R. 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.

[14] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997.

[15] L. A. Routing and Traffic. http://lartc.org/.

[16] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator, 2002.

[17] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.